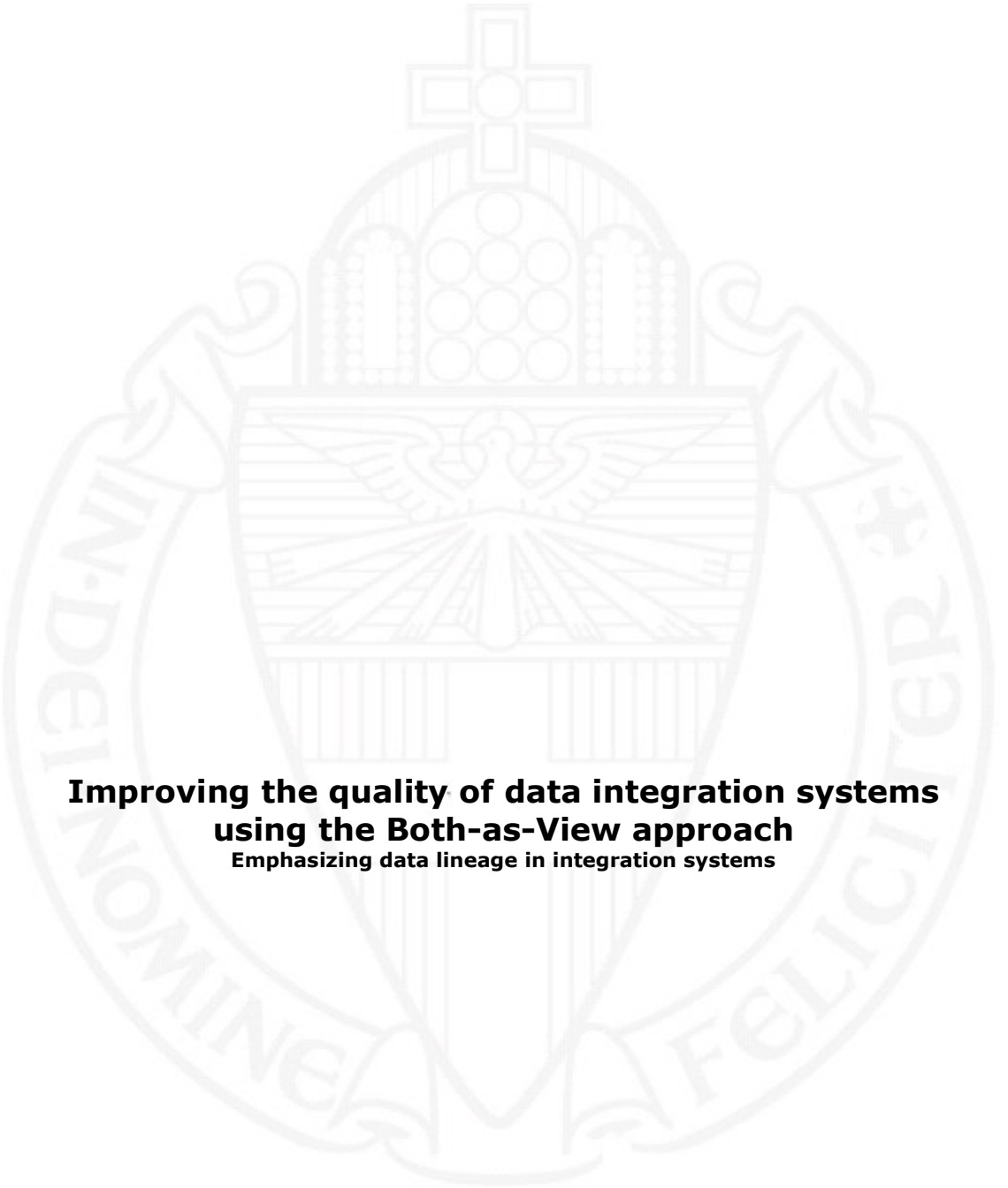

Improving the quality of data integration systems using the Both-as-View approach

Emphasizing data lineage in integration systems

Master Thesis

Author : Marc van der Wielen
Studentnr. : 0245437
Tutor : Prof. Dr. Ir. Dhr. T. van der Weide
Reviewer : Prof. Dr. Dhr. H.A. Proper

BI4U

Abstract

Information systems play an important part in nowadays society. Those systems are used to store data, which is in fact an abstract interpretation of the real world. This interpretation can be described by way of models. Integration systems are used to combine and enrich data from different information systems by expressing the relations between those models.

This master thesis discusses how different types of methods and techniques can be used to integrate data from different sources, and present it to the user though a global view over these sources.

In the past, requirements on integration systems were limited. Traditional integration approaches sustained in fulfilling the requirement of the ability to integrate. Over time, however, requirements on integration systems changed. Examples of new requirements are the ability to check results, proof correctness and tracking the overall process of integration. Traditional integration approaches didn't support these types of requirements and therefore the need for a new integration approach arises.

The Both-as-View approach, a new integration approach discussed in this thesis, addresses common issues with integration and compares its functionality with traditional integration approaches. Some advantages of the Both-as-View approach are the native support for schema evolution, data lineage and a formal way of working. Furthermore, it is designed to support every type of modeling language. Combined with the proposed test methods and techniques from the research of Rob van Kempen it can contribute to overall quality improvement of an integration system.

Preface

I hereby present you my master thesis "quality improvement of integration systems". This thesis discusses which methods and techniques can contribute to quality improvement of integration systems.

I want to thank Rob van Kempen for the fine collaboration during the last year. As a friend and as a colleague I want to say that I am proud of him. Once again the collaboration resulted in success! Thanks for your patience and all the good time.

This master thesis contributed to my personal development in research skills. A visit to the department of information science of the University of London, which has a research group for integration systems, helped me in understanding the current issues with integration systems. Therefore I want to thank H. Fan who was willing to support me during my research and Niek Jetten (BI4U) who made this trip possible.

Many hours of thinking and discussing about integration systems led to this master thesis. I want to thank our professor Theo van der Weide who was willing to guide and support me throughout this research. His vision helped me in creating a different view on the problem area. Initially, my focus was on the problem area of data lineage, after many discussions my focus moved to the contribution of data lineage to quality improvement of integration systems. In my opinion, this shift of focus improved the content and goal of my research.

In the beginning of my search for a research assignment I insisted to start a research at a company. In my opinion a research at a company adds more experience and motivates in finishing the research. Fortunately, my friend Sam Geurts was willing to invite me at a company called BI4U. Founder and managing director of BI4U, Niek Jetten, was there to give us a cross-examination on the topic 'Business Intelligence'. This interesting discussion led to the opportunity to start with my research at BI4U and I am very grateful for this. Many thanks Sam Geurts and Niek Jetten! Furthermore, I want to thank my other colleagues at BI4U for sharing their knowledge and experience.

Last but definitely not least, I want to thank my girlfriend Marieke van Kempen for her support and understanding for all the hours I have spent in doing research. I want to thank my mom and dad for their infinite faith and enabling me to graduate in two studies. Of course I want to thank my friends who have shown their interest and sympathy for my research!

Many, many thanks too all of you, you are the best!

About the author:
Marc van der Wielen was born on the first of September 1979. After his completion of the HAVO he started a study computer science at the 'HTS Arnhem'. After successfully graduating he wanted to develop his research skills and started a two year study of information science at the Radboud University of Nijmegen.

Table of contents

1. Introduction

This master thesis discusses how the quality of an integration system can be improved using a set of methods and techniques. This research refers in some cases to the research done by Rob van Kempen. His research focuses on how test and development methods can be applied to improve the quality of an integration system.

Chapter 2 introduces the problem area of data integration. The first paragraph of this chapter discusses the need for integration accompanied by some everyday life integration issues. In the next paragraph common conflicts with integration are mentioned followed by a distinction in two types of integration, namely virtual and materialized integration.

Based on the theory in chapter 2 traditional integration approaches are introduced in chapter 3. This chapter explains which approaches are commonly used to integrate data registering systems. Those approaches however lack in some functionality and therefore a new integration approach is introduced in chapter 3.3. The remaining of this chapter discusses in detail the functionality and advantages of this new approach compared to the traditional approaches.

In chapter 4 the term data lineage is introduced followed by an explanation why lineage is useful in an integration system. The following paragraphs introduce two researches on lineage. One researches focuses on algorithms for calculating lineage without the use of an integration system, the other research mentions the lack of this framework and introduces new algorithms which are based on the integration framework discussed in chapter 3.3.

Chapter 5 discusses how the quality of an integration system can be improved. First the definition of 'quality' is given to state what the meaning of this term to our opinion is. The following two paragraphs explain why schema evolution and data lineage can contribute to quality improvement of an integration system. The last paragraph shows how these methods can be used in testing and auditing methods based on the research of Rob van Kempen.

The last chapter contains a conclusion of this research, and presents recommendations for future work.

2. Information integration systems

People interpret events and entities in their own way, probably slightly different for each person. Therefore, these interpretations may be unstructured and complicated. To discuss these interpretations one can use a natural language. A natural language, however, may cause ambiguity in the description. To make an unambiguous description one needs to describe the interpretation of an entity or event in a more formal way. An example of such a formal description is the drawing an architect uses to inform engineers on what they have to construct and to conform to the customer. More general, people record their interpretation of the universe of discourse in a model in such a way that interpretation of that model is unambiguous. Commonly the interpretation of a universe of discourse by multiple persons might slightly differ. For example, a doctor interprets the properties of a patient as its external characteristics like the color of the eyes, their illness, sex and physical height while a human resource manager interprets the properties of an employee as its loan, function and competences. Actually both professions describe properties of a human being. Therefore differences may exist in a model applied by different people because of their different point of view on the same universe of discourse, see Figure 1.

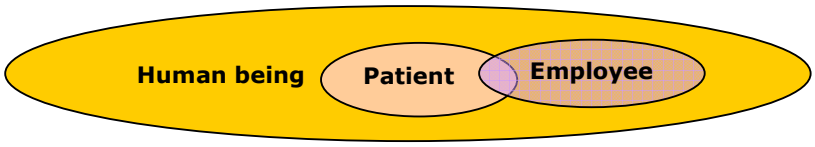

Figure 1 - Different views on the same universe of discourse

Suppose the models of a patient and an employee are integrated. Why would one want to integrate these models? One reason for integrating these two models is the enrichment of information. The integrated model provides information of employees combined with patient information. This enriches the information because there is more information available compared to the separate employee and patient model. With the integrated model a manager would be able to retrieve for example an overview of all patients who turn out to be employees categorized by function.

The process of obtaining information which is related to multiple systems can be complex because of the differences and inconsistencies in structure and semantics between models. To be able to expose the non-shared information residing in those systems the need for integration rises. By integrating different sources, facts about an object (this can be a patient or employee for example) stored in different locations can be combined. The problem that makes integration hard is the heterogeneity of models describing the universe of discourse used by the operational systems [23]. Differences between semantics and structure should become transparent as a result of information integration. In other words, a unified view of information is presented to the users, while information residing in operational systems exists of dissimilar elements and structures. The authors of [3] refer to this as the key problem of integration.

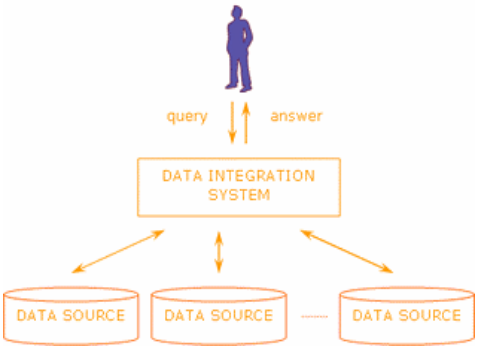

Figure 2 – [23] Common interface to users of integration systems

2.1 Schema integration

Schema integration is the problem of combining models of the same universe of discourse and providing a user with a unified view of these models. The main aspects and terminology of schema integration will be covered in this section.

According to the authors of [20], three main elements of the architecture of a schema integration system can be distinguished. These elements are:

- a global schema
- one or more source schemas
- mappings between the global and the source schemas

These three elements are shown in Figure 3.

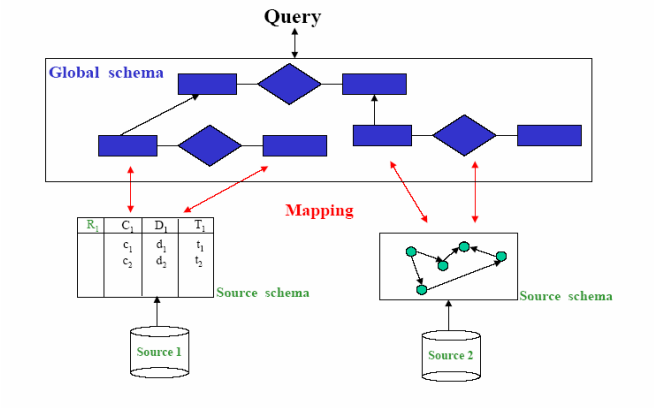

Figure 3 – [20] Schema integration

The global schema describes the structure of the model representing the universe of discourse. The mappings, or connections, describe how each element in the local schemas relates to the global schema. The process of creating mappings is referred to as "schema integration". Schema integration is the process of finding a unification of semantically similar information which is represented in a heterogeneous way across different information sources. Finding this agreement is complex because one has to find differences and similarities in each schema to be able to conform. Common differences and similarities in schemas are described more precisely in the next section.

Summarized, the essence of integration is to combine information in a logical way so information can be queried as one through a common interface [1]. The schema for each information source needs to be connected through a mapping with the global schema of the common interface to enable querying.

2.2 Schema conflict types

Schema conflicts exist because of different interpretations of entities and events in the universe of discourse. In the employee and patient example different properties of the same human being are described. These different properties may lead to conflicts between the models of an employee and a patient. Typical types of conflicts between models are: naming conflicts, structural conflicts and domain conflicts. These types of conflicts are explained in the next paragraphs.

2.2.1 Naming conflicts

Naming conflicts arise when the model of an interpretation contains an entity with a different name than another model that contains in fact the same entity. For example, model A contains an entity called "person" and model B contains an entity called "staff". Both entities are persons but the naming of those entities differs. This kind of naming conflict is called a synonym conflict. On the other hand a homonym conflict can also occur. A homonym conflict occurs when an entity in model A contains inherently different data than an entity with the same name in model B [23]. An example of a homonym conflict is the department of an employee and the department of a patient. Both entities have the same name but they contain semantically different data.

2.2.2 Structural conflicts

A second type of conflict, named structural conflicts, may exist between two models. When a construct in model A represents the same entity modeled in a different kind of construct in model B we speak of a structural construct. The example in Figure 4 of a structural conflict is an entity in model B whereas in model A the entity is modeled as an attribute.

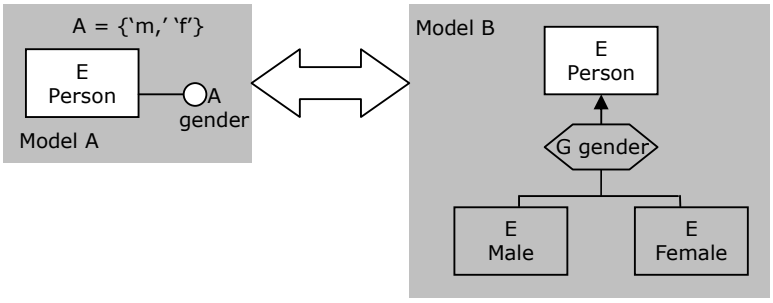

Figure 4 – Structural equivalence

2.2.3 Domain conflicts

Domain conflicts between models occur when attributes in a model differ in type. The type of an attribute describes what type of information the attribute contains. For example, the type tells us the attribute can only contain numeric values. Suppose the type of an attribute in model A tells us that it can only contain numeric values while the same attribute in model B can contain alphanumeric values, in this case there is a domain conflict between the models.

2.2.4 Constraint conflicts

The last type of conflict between models is the constraint conflicts. The constraint refers to an entity in the universe of discourse and tells us how this entity is uniquely identified. If the constraint on the same entity in model A and B differ from each other we speak of a constraint conflict. Consensus in how to uniquely identify the entities has to be found to be able to integrate.

2.3 Schema integration approaches

The preceding paragraphs explained why there is a need for integration and what kind of problems can exist with integration. As mentioned in chapter 2.1 a path (or mapping) needs to be defined which describes how each element in the local schema relates to the global schema. Several approaches in how to define such a mapping are discussed in the literature. [3], [2], [22] and [20] mention schema integration approaches called respectively Local-as-View (LAV) and Global-as-View (GAV). In chapter 3 these approaches are explained in detail with an example. For now these approaches can be seen as the way in which mappings between the local and global schemas are described. In the Global-as-View approach one describes how each element in the global schema relates to the local schemas. Whereas the Local-as-View approach describes how each element in the local sources relates to the elements in the global schema.

Beside the different approaches of schema integration a distinction is made in how the information needed is retrieved. Independent of which approach is used to define the mapping between the local and global schema a virtual or materialized integration approach determines how the information is accessed. In the next paragraphs virtual and materialized integration are explained and an example is given.

2.4 Virtual integration

The virtual integration approach leaves the information requested in the local sources. The virtual approach will always return a fresh answer to the query. The query posted to the global schema is reformulated into the formats of the local information system. The information retrieved needs to be combined to answer the query. A challenge with the virtual integration approach is the performance. Each time a query is posted all sources which provide information to answer the query need to be inquired and the results need to be combined. In case of information systems which are intensively used by operational usage an answer to the query could take long. Suppose one wants to know the cast and director of some kind of movie with a list of cinemas which currently plays this movie. To be able to answer this query several information sources need to be accessed. First the title of the movie will be posted to, for example, IMDB which is a large internet movie database on the web (http://www.imdb.com). Second, a query is posted to an

information system which provides information about cinemas which play that movie. Integrating the results coming from the information systems will answer the query posted to the integrated system and give the user a fresh result with information about the movie and in which cinemas the movies are shown. This example is schematically shown in Figure 5 below.

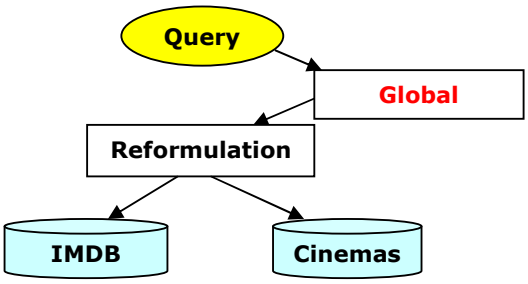

Figure 5 - Virtual integration

2.5 Materialized integration

The other way to retrieve information for answering queries is by using materialized integration. The materialized integration stores intermediate results (answers to the queries) in a separate source. The local sources in the materialized approach will not be stressed each time a query to the global schema is submitted. Researchers of [4] refer to materialized integration when integration systems keep a reproduction of information coming from the local sources. In contrast to virtual integration the information freshness of materialized integration is dependant of the frequency of updates on the intermediate results. Materialized integration is typically applied in situations where operational systems can't be stressed by the integration system and when there is a need for keeping track of history. Virtual integration can't keep track of history because the query posted to the integration will always return a fresh answer.

A concept for the materialized integration approach has been developed to simplify the complexity of integration issues. The goal of the data warehouse concept is to integrate data residing in heterogeneous source systems into a central repository for analysis, enrichment and processing of the organized information [6], [21], [15]. The basic architecture of the data warehouse concept is shown schematically in Figure 6.

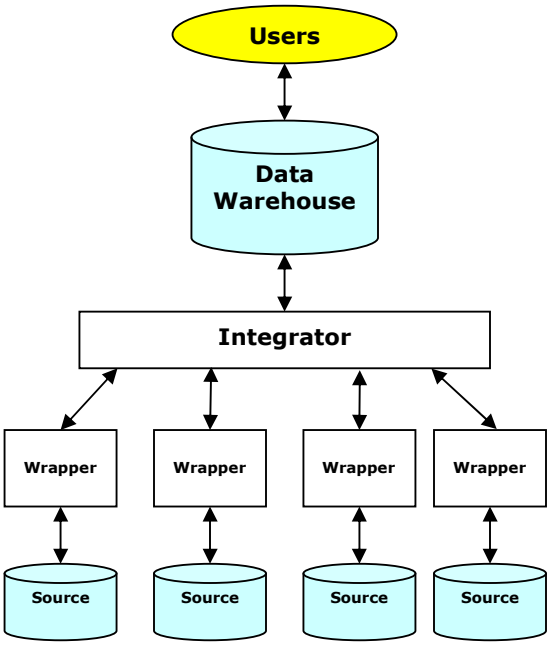

Figure 6 - Data Warehouse architecture

Information residing in the sources is accessible for the integration process. The integration process is responsible for transforming, integrating and enriching of the information. The result of this process, also called the Extract-Transform-Load (ETL) process, is stored (materialized) in the data warehouse. The data warehouse contains information which was formerly partially or completely unavailable.

2.6 Summary

People have different interpretations of entities and events in a universe of discourse. Models are made to unambiguously describe people's interpretation of these entities and events. However, the models may slightly differ in properties and structure describing an entity or event. Different types of conflicts between the models exist, for example naming conflicts, structural conflicts and domain conflicts.

A reason for integrating models is to be able to enrich information about entities. By integrating the properties of an employee and a patient, one is able to retrieve much more 'valuable' information than before. Different approaches to achieve schema integration are mentioned in the literature. For example the Global-as-View and the Local-as-View approach. Beside these integration approaches a distinction is made in how the information to be integrated is retrieved. Virtual integration will answer the query by reformulating the query posted to the global schema to the local schemas. Therefore virtual integration will return a fresh answer to the query posted. Materialized integration on the other hand will store intermediate results to answer the queries in a reproduction of the local sources. A downside with materialized integration compared to virtual integration is the dependency of the frequency of updates on the reproduction, which determines the freshness of the answer to the query posted.

3. Approaches for data integration

3.1 Looking at data through views

Data integration approaches are based on creating views over different data sources, and combine them so they can be presented to the user as if they were one. Before starting reasoning about how this can be achieved, this section will briefly explain what a view is.

A view is a virtual way of looking at a model. Instead of the entire model, only a part of a model or a joined partition of two (or more) models is presented to the user. When two models are integrated and presented to the user by way of a view, this view is a result of the correspondence of two different interpretations of a domain.

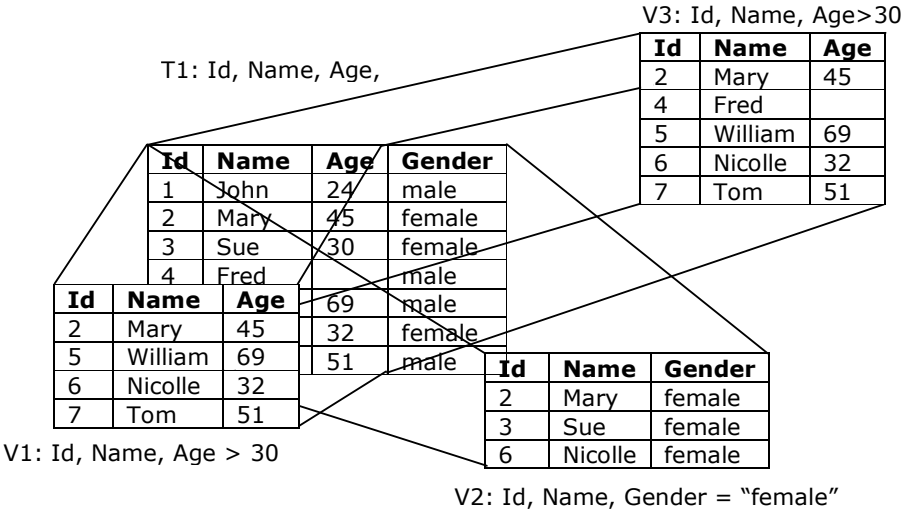

Figure 7– Example of different views

Figure 7 shows a table named T1 containing information about employees, and three views, V1, V2 and V3. T1 is a representation of a model in a relational database. V1 and V3 are views over T1, containing information about a person's Id, name and age. In these views only persons older than 30 years are shown. The difference between V1 and V3 will be explained later. V2 is also a view over T1, containing information about a person's Id, name and gender. It only contains females. Both views only show part of table T1, and leave away information that is not relevant to a certain user.

The terms *sound*, *exact* and *complete* are used to express to what degree the extent of a view corresponds to its definition. This terminology will later on be used to describe the reach of data-integration approaches. A view, defined over some data source that is *sound* provides a subset of the available data in the data source that corresponds to the definition. It delivers only, but not necessarily all answers to its definition. The answers it does deliver might be incomplete, but they are correct, hence the term *sound*.

If a view is *complete*, it provides a superset of the available data in the data source that corresponds to the definition. It delivers all answers to its definition, and maybe more. Since the set of answers might contain more than the answers corresponding to the views definition, but does contain all answers corresponding to the views definition, the term *complete* is being used. Furthermore, a view is *exact* if it provides all and only data corresponding to the definition.

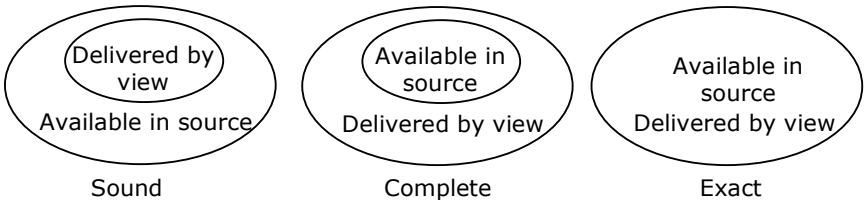

Figure 8 - Sound, complete and exact Views

For example, consider a data source containing information about employees. For some employees, but not for all, their age is known. For all employees, their gender is known. A view is created, defining that it provides a list of employees older than 30. Since not for every employee his/her age is known, a consideration has to be made. Should the view include only employees of which the age is known? In that case, some employees of whom the age is unknown might be older that 30 but aren't included in the view. Therefore, the view would become *sound*. View *V1* in Figure 7 is an example of a sound view. If the decision will be to include also those employees whose age is unknown, it might be the case that some employees not being older than 30 are included in the view. The view would become *complete*, as is view *V3* in Figure 7. If a view is created containing only female employees that view would become *exact*, just like view *V2* in Figure 7.

3.2 Traditional approaches

3.2.1 The "Global-as-View" approach (GAV)

The Global-as-View approach defines relations with the local schemas by specifying how the constructs in the global schema are related. Each mapping associates every entity in the global schema with the entities in the local schemas. This way, each entity in the global schema is defined as a view over the local schemas, hence the name "Global-as-View".

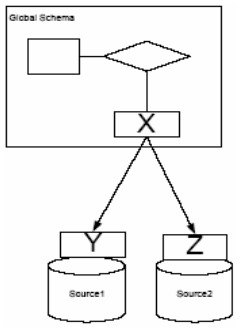

Figure 9 – [23] Global-as-View

Because each global entity is defined as a view over the local schemas, the total number of GAV-rules necessary to define the mappings in a data-integration system corresponds to the number of entities in the global schema.

Query answering in the Global-as-View happens through query unfolding. Unfolding queries to the local sources is relatively easy compared to the Local-as-View approach in which queries posted to the global schema have to be rewritten before being able to answer the query.

In practice, the extent of a global schema defined by GAV-rules is assumed to be *exact*.

Adding new sources to the integration system in the Global-as-View approach isn't easy compared to the Local-as-View approach. When a new source is added one has to find a way in which the new source can be used to obtain tuples for each relation defined in the global schema. In other words, one has to determine in which way the new source will interact with each of the existing sources in order to answer the query.

3.2.2 The "Local-as-View" approach (LAV)

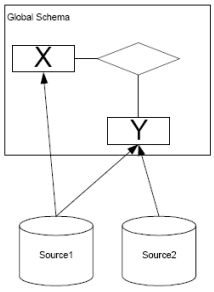

Figure 10 - [23] Local-as-View

The Local-as-View approach, see Figure 10 , describes local sources as a view defined in terms of the global schema. The global schema is predefined and for each local source is described how it delivers information to the global schema. Each mapping associates the entities in the source schemas by way of a query over the global schema. Thereby, each source schema is defined as a view over the global schema, hence the name "Local-as-View".

Each entity in the local schemas is defined as a view over the global schema. Therefore, the total number of LAV-rules necessary to define the mappings in a data-integration system corresponds to the number of entities in the local schemas.

Processing queries in the Local-as-View approach is difficult because the only knowledge of the global-schema available is through the views representing the sources. Such view only provides partial information about the data. Since the mapping associated to each source as a view over the global schema it is not clear how to use the sources in order to answer queries expressed over the global schema. Therefore extracting information from an integration system using the Local-as-View approach is a complex task because one has to answer queries with incomplete information.

Views defined by LAV-rules might be sound or complete, in terms of 3.1.

Changes in source schemas are easily carried through within the Local-as-View approach. When a new source is added one has to add a new association describing how this source contributes to the global schema. Making changes in the global schema is difficult because one has to redefine how each local source element contributes to the global schema.

3.2.3 Schema integration using Local-as-View and Global-as-View

This chapter illustrates how both integration approaches are used, by means of some examples. A calculus-like notation is used as query language for those examples. This notation subsumes SQL in expressiveness as stated in [2]. This notation can best be seen as a filter that describes what data-tuples are and are not allowed according to the schema. The notation will be clarified during the examples.

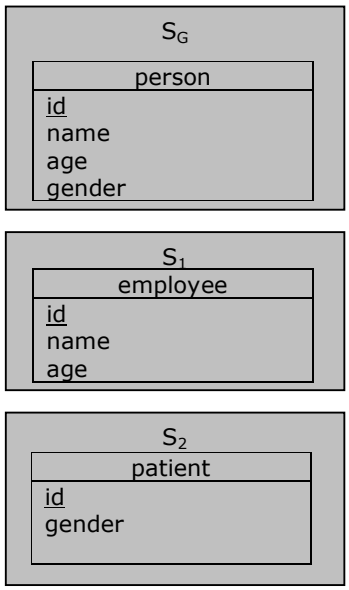

Figure 11 - Local and global schemas

The first example will be a very straightforward situation. Consider the following schemas:

S_1 and S_2 are local schemas, S_G is a global schema. An integration approach is used to establish the connection between the entities of S_1 and S_2 and the entities of S_g . As stated in section 3.2, for the GAV approach each entity of the global schema is described in terms of the local schemas. The GAV-rule that describes S_G's table *Person* is:

G_1 Person(Id, Name, Age, Gender)={x,y,z,w|⟨x,y,z⟩∈Employee ∧ ⟨x,w⟩∈User}

GAV-rule G_1 states what values are allowed for the tuples of *Person* in S_G . A tuple in *Person* contains attributes *id, name, age* and *gender*, represented by respectively *x, y, z, w*. The formula encapsulated in braces ({}) works as a filter, telling which values for *x, y, z, w* are allowed. Only values so that the tuple *Id, name, Age* exists in *Employee* and the tuple *Id, Gender* exists in *User* are allowed. Since *x* is compared to *Id* in *Employee* as well as in *User*, this means that those values must be the same. Note that this rule states that only persons that exist in both the *Employee* schema and in the *User* schema exist in the *Person* schema. Dependent on the information required in S_G it might be possible that also employees that aren't registered as a user, and users that aren't registered as an employee have to be included in S_G . In that case, $G_{1'}$ will be the resulting GAV-rule:

$G_{1'}$ Person(Id, Name, Age, Gender)={x,y,z,w|⟨x,y,z ⟩ ∈ Employee ∧ ⟨x,w⟩ ∈ User ∨ ⟨x,y,z⟩ ∈ Employee ∧ w='' ∧⟨x,_ ⟩ ∉ User ∨ ⟨x,w⟩ ∈ User ∧ y='' ∧ z=''}

Note that simply replacing the "∧" symbol in G_1 for a "∨" symbol would result into an erroneous mapping. By doing so, no information would be provided about the values of *name*, *age* and *gender* for persons that do not exist in both schemas. The result would be that a tuple where *id* exists only in *employee* with made-up values for *name* and *age* would pass the filter.

GAV-Rule G_1 and $G_{1'}$ show two different ways to describe how the global schema is defined as a view over the sources. The name "Global-as-View" confirms this way of thinking.

The LAV approach describes each entity of the sources, stated in terms of the global schema. Therefore, two LAV-rules are necessary, the first describing S_1's entity *Employee* and the second describing S_2's entity *User*:

L_1 Employee(Id, Name, Age) = {x,y,z | ⟨x,y,z,_⟩∈ Person}

L_2 User(Id, Gender) = {x,y|⟨x,y,_,_⟩∈ Person}

LAV-rule L_1 states that tuples in *Employee* are those tuples that exist in *Person*. Here, the age of an employee doesn't matter. Therefore, an underscore ("_") is placed at the tuples position of *Gender*, meaning any value of *Gender* will pass. In a similar way, L_2 describes how *User* can be defined. These rules define each source schema as a view over the local schema, hence "Local-as-View". When a query is posted against the global schema, either the GAV- or the LAV-rules are being used to redistribute parts of the query to the relevant sources.

The following example is taken from [1], and is a little more complex. It is intended to express the shortcomings and advantages of both approaches.

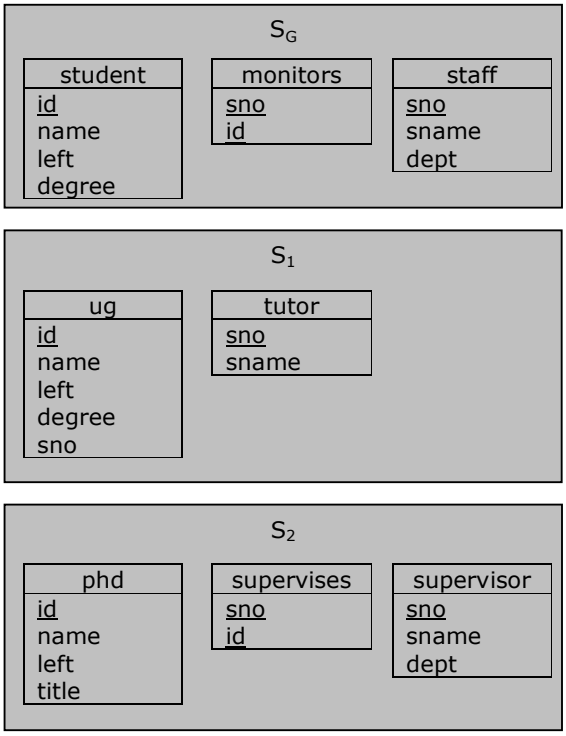

Figure 12 - Global and local schemas

Consider the relational schema's described in Figure 12. Schema S_1 contains an entity *ug* holding information about undergraduate students. Attribute *sno* is a foreign key, referring to the students' tutor. Table *tutor* contains information about the member of staff who was the student's tutor. Schema S_2 describes the relation between PhD-students and their supervisors. A PhD-student keeps the same *id* when he previously studied as an undergraduate student. Global schema S_g describes students, staff-members and their relations.

The following GAV-rules can be created to define the mappings between the entities in the source and entities in the global schemas:

G_1 $\text{student(id, name, left, degree)} = \{\text{x,y,z,w,l} \langle \text{x,y,z,w,\_} \rangle \in \text{ug} \wedge \langle x, \_, \_, \_ \rangle \notin \text{phd} \vee \langle x, y, z, \_ \rangle \in \text{phd} \wedge w = \text{'phd'}\}$

G_2 $\text{monitors(sno, id)} = \{x, y \mid \langle x, y \rangle \in \text{tutor} \wedge \langle x, \_, \_, \_, \_ \rangle \notin \text{phd} \vee \langle \text{x,y} \rangle \in \text{supervises}\}$

G_3 $\text{staff(sno, sname, dept)} = \{\text{x,y,z} \mid \langle \text{x,y} \rangle \in \text{tutor} \wedge \langle \text{x,\_,\_} \rangle \notin \text{supervisor} \vee \langle x.y.z \rangle \in \text{supervisor}\}$

These view definitions are used to rewrite queries posted to the global schema, into distributed queries over the sources. G_1, G_2 and G_3 define the constructs of the global schema according to the GAV approach. All of these rules are exact, speaking in terms of 3.1.

The following transformation-rules define the mappings between entities in the source- and global schemas according to the LAV-approach:

L_1 $\text{tutor(sno, sname)} = \{\text{x,y} \mid \langle \text{x,y,\_} \rangle \in \text{staff} \wedge \langle \text{x,z} \rangle \in \text{monitors} \wedge \langle \text{z,\_,\_,w} \rangle \in \text{student} \wedge \text{w} \neq \text{'phd'}\}$

L_2 $\text{ug(id, name, left, degree, sno)} = \{\text{x,y,z,w,v} \mid \langle \text{x,y,z,w} \rangle \in \text{student} \wedge \langle \text{v,x} \rangle \in \text{monitors} \wedge \text{w} \neq \text{'phd'}\}$

L_3 $\text{phd(id, name, left, title)} = \{\text{x,y,z,w} \mid \langle x, y, z, v \rangle \in \text{student} \wedge v = \text{'phd'} \wedge \text{w} = \text{null}\}$

L_4 $\text{supervises(sno, id)} = \{\text{x,y} \mid \langle \text{x,y} \rangle \in \text{monitors} \wedge \langle \text{x,\_,\_,z} \rangle \in \text{student} \wedge \text{z} = \text{'phd'}\}$

L_5 $\text{supervisor(sno, sname, dept)} = \{\text{x,y,z} \mid \langle \text{x,y,z} \rangle \in \text{staff} \wedge \langle \text{x,w} \rangle \in \text{monitors} \wedge \langle \text{w,\_,\_,v} \rangle \in \text{student} \wedge \text{v} = \text{'phd'}\}$

L_1 and L_2 define the constructs of the entities in source schema S_1 . L_3 , L_4 and L_5 define the constructs of entities in S_2. Note that L_1 does not contain all tutors in S_1 . Staff members that monitor PhD-students and tutor undergraduate students do not appear in this view. Therefore, this view is *complete*. By removing the constraint *w≠'phd'* , this rule will derive a superset of the sources and therefore become *sound*. The same goes for L_2 , which is a *complete* rule also, and will become *sound* by removing the constraint *w≠'phd'*. A definition of L_1 and L_2 that is *exact* is not possible in this case, because it is not possible to distinguish which staff-members supervise PhD-students and undergraduates, and which ones only supervise PhD-students. L_3, L_4 and L_5 are *exact* rules, since they derive exactly the information that is available in the sources.

3.3 A new integration approach: the "Both-as-View" approach

In this section, a new integration approach called the "Both-as-View" aproach, abbreviated as "BAV", will be introduced. BAV is an approach that combines the best of the GAV method (Global-as-View) and the LAV method (Local-as-View). The Both-as-View approach distinguishes itself from the GAV and LAV data-integration approach by having a methodology for handling a wide range of data models where the other integration approaches assume integration is done through a single common data model. Another advantage of the Both-as-View approach compared to the LAV and GAV approach is the support for schema-evolution (on both local and global schemas). In the LAV and GAV approach changes in the local or global schema (depending on the type of approach) lead to inconsistencies in the integration system.

3.3.1 The underlying data model HDM

The writers of [1] developed a framework, which supports schema-transformation, and can be used for integrating heterogeneous data-sources. This framework is described in the next paragraph. Part of this framework is a low-level model called "Hyper graph-based data model" (HDM). Higher-level models can be expressed in terms of this low-level model. By expressing higher-level modeling languages in terms of this low-level hyper graph-based data model the ability to integrate heterogeneous schemas rises. Working with a low-level model describing the higher-level modeling languages results in a unified understanding of the model. Semantic mismatches are avoided, since the HDM provides unifying semantics for higher-level modeling constructs. An example of a hyper graph-based data model construct can be generally expressed as the construct showed in Figure 13.

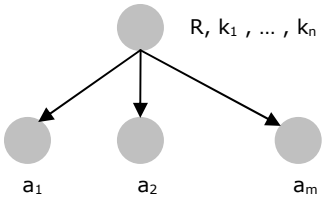

Figure 13 – [1] Construct of hyper graph-based data model

As said the hyper graph-based data model is a low-level model which can represent higher-level modeling languages like an entity-relational, xml or flat text model. An advantage of using a low-level data model is that semantic mismatches between modeling constructs are avoided.

3.3.2 Primitive transformation steps

In the Both-as-View integration approach, a schema is incrementally transformed, by applying succeeding primitive transformation-steps. Each step adds, deletes or renames just one schema constructs. Each *add* or *delete* transformation is accompanied by a query, which specifies the extent of the schema construct, in terms of the rest of the schema constructs.

In [3] a simple relational model is shown. A model in this form can be transformed by using the following primitive transformations:

addRel (<<R, k_1..k_n>>,q) -> Adds a relation R with primary key attributes k_1…k_n with n>=1. Query q is the set of primary-key values which belong to R.

addAtt(<<R,a>>,c,q) -> Adds an attribute a to the relation R. Parameter c has the value "null" or "not null". Query Q specifies the extensiveness of the binary relation between the primary key attributes of R and the newly added attribute in terms of the existing schema constructs (extensiveness is the set of pairs).

delRel(<<R,k_1 .. k_n>>,q) -> Removes relation R with its keys. To be able to delete the non-key attributes have to be removed first. Query q expresses how the set of keys can be retrieved from the remaining schema constructs.
delAtt(<<R,a>>,c,q) -> Remove attribute a from relation R.

Each of these transformations has an optional argument which can enforce a condition on the value of the data. Each of the transformations, t, has automatically a (derivable) transformation $\bar{t}$.

$t : S_x \rightarrow S_y$	$\bar{t} : S_y \rightarrow S_x$
$\text{addRel}(<< R,\vec{k}_n >> ,q)$	$\text{delRel}(<< R,\vec{k}_n >> ,q)$
$\text{addAtt}(<< R,a >> ,c,q)$	$\text{delAtt}(<< R,a >> ,c,q)$
$\text{delRel}(<< R,\vec{k}_n >> ,q)$	$\text{addRel}(<< R,\vec{k}_n >> ,q)$
$\text{delAtt}(<< R,a >> ,c,q)$	$\text{addAtt}(<< R,a >> ,c,q)$

Where $\vec{k}_n$ is a abbreviation for the set k_1,..,k_n

This derivable counterpart creates the possibility to reverse each schema-transformation. Thereby, queries can automatically be translated and distributed to different schemas.

Suppose a schema *S* is transformed into schema *S'* by way of a single primitive transformation *t*. Query *Q* posted to *S* now has to be translated into *Q'* and posted to *S'*. There are two cases in which the type of transformation of *t* is important: if *t* is of the type *delRel* or *delAtt*. If that is the case, in *Q*, instances of the relation or attribute removed by *t* have to be replaced by the query *q* that accompanies *t*. The result is a translated query *Q'*.

If more than one transformation is applied to a schema, then the substitutions described above have to be executed subsequently, to acquire *Q'*.

3.3.3 Example of Both-as-View integration

The primitive transformations mentioned earlier can be applied to the relational model in 3.2.3. The first example will show how the schemas of the simple example in 3.2.3 are described in terms of the Both-as-View approach. The second example is a more complex example and introduces two other transformation rules.

Example 1 (Simple Both-as-View transformation example):

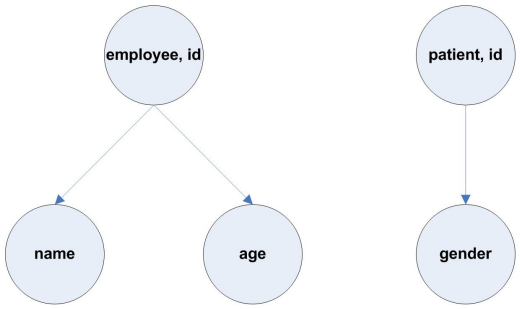

Figure 14 – Local schemas S_1 and S_2

Schema S_1 describes the structure of an employee who has two attributes named 'age' and 'name'. The other schema S_2 is the structure of a patient with one attribute named 'gender'. Given these schemas and the global schema named S_g the Both-as-View rules describing each transformation step to integrate are defined below.

1. addRel(⟨⟨person,id⟩⟩, {x | x ∈ ⟨⟨employee,id⟩⟩ ∧ x ∈ ⟨⟨patient,id⟩⟩})
2. addAtt(⟨⟨person,name⟩⟩, {x,y | (x,y) ∈ ⟨⟨employee,name⟩⟩ ∧ x ∈ ⟨⟨employee,id⟩⟩ ∧ x ∈ ⟨⟨patient,id⟩⟩})
3. addAtt(⟨⟨person,age⟩⟩, {x,y | (x,y) ∈ ⟨⟨employee,age⟩⟩ ∧ x ∈ ⟨⟨employee,id⟩⟩ ∧ x ∈ ⟨⟨patient,id⟩⟩})
4. addAtt(⟨⟨person,gender⟩⟩, {x,y | (x,y) ∈ ⟨⟨patient,gender⟩⟩ ∧ x ∈ ⟨⟨employee,id⟩⟩ ∧ x ∈ ⟨⟨employee,id⟩⟩})

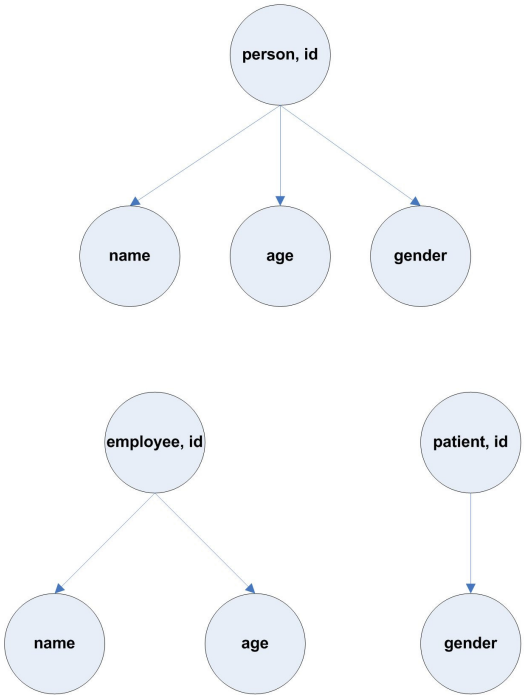

Figure 15 - Integrated schema S_1 and S_2 into S_g

First the constructs of the global schema is defined by specifying the 'addRel' and 'addAtt' rules. These rules describe how each element in the global schema relates to the local schemas. In this simple example the global schema will only contain employees which also have been a patient. As one can see the 'addRel' and 'addAtt' rules all say that x should be an element of patient AND employee. This means that an employee which has never been a patient won't exist in the global schema.

5. delAtt(⟨⟨employee,age⟩⟩, {x,y | (x,y) ∈ ⟨⟨person,age⟩⟩ ∧ x ∈ ⟨⟨employee,id⟩⟩ ∧ x ∈ ⟨⟨patient,id⟩⟩})
6. delAtt(⟨⟨employee,name⟩⟩, {x,y | (x,y) ∈ ⟨⟨person,name⟩⟩ ∧ x ∈ ⟨⟨employee,id⟩⟩ ∧ x ∈ ⟨⟨patient,id⟩⟩})
7. delRel(⟨⟨employee,id⟩⟩, {x | x ∈ ⟨⟨person,id⟩⟩} ∧ x ∈ ⟨⟨employee,id⟩⟩ ∧ x ∈ ⟨⟨patient,id⟩⟩)
8. delAtt(⟨⟨patient,gender⟩⟩, {x,y | (x,y) ∈ ⟨⟨person,gender⟩⟩ ∧ x ∈ ⟨⟨employee,id⟩⟩ ∧ x ∈ ⟨⟨patient,id⟩⟩})
9. delRel(⟨⟨patient,id⟩⟩, {x | ⟨⟨person,id⟩⟩ ∧ x ∈ ⟨⟨employee,id⟩⟩ ∧ x ∈ ⟨⟨patient,id⟩⟩})

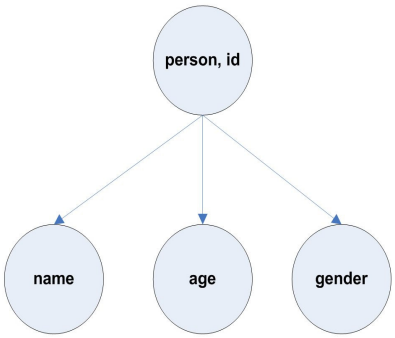

Figure 16 - Global schema S_9 after applying all transformation rules

The last transformations describe how the constructs of the local schemas can be removed. The 'delRel' and 'delAtt' rules describe how each element in the local schema relates to the global schema. When these transformation steps have been applied the only constructs left are those of the global schema.

Example 2 (more complex Both-as-View integration example):

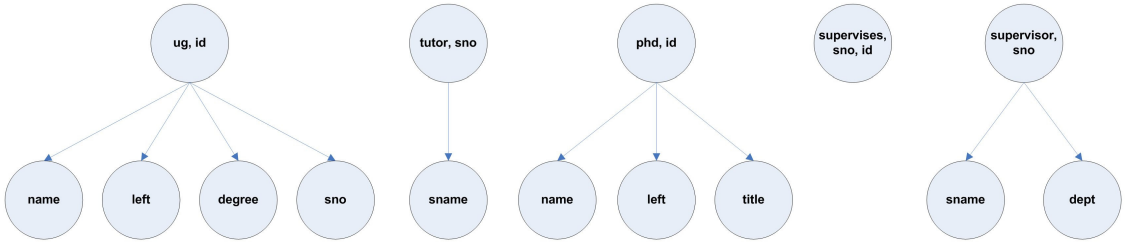

Figure 17 - Local schemas S_1, and S_2 ({ug,tutor},{phd, supervises, supervisor})

The goal in this example is to integrate the two local schemas into one global schema in such a way that the global schema gives information about current and past students with their last degree that they studied and when they left the university. Note that this example is taken from [1].

1. addRel(⟨⟨student,id⟩⟩, {x | x ∈ ⟨⟨ug,id⟩⟩ ∨ x ∈ ⟨⟨phd,id⟩⟩})
2. addAtt(⟨⟨student,name⟩⟩, notnull, {x,y | (x,y) ∈ ⟨⟨ug,name⟩⟩ ∧ x ∉ ⟨⟨phd,id⟩⟩ ∨ (x,y) ∈ ⟨⟨phd,id⟩⟩})
3. addAtt(⟨⟨student,Left⟩⟩, null, {x,y | (x,y) ∈ ⟨⟨ug,Left⟩⟩ ∧ x ∉ ⟨⟨phd,id⟩⟩ ∨ (x,y) ∈ ⟨⟨phd,Left⟩⟩})
4. addAtt(⟨⟨student,Degree⟩⟩, notnull, {x,y | (x,y) ∈ ⟨⟨ug,Degree⟩⟩ ∧ x ∉ ⟨⟨phd,id⟩⟩ ∨ x ∈ ⟨⟨phd,id⟩⟩ ∧ y = 'phd'})
5. addRel(⟨⟨staff,sno⟩⟩, {x | x ∈ ⟨⟨tutor,sno⟩⟩ ∨ x ∈ ⟨⟨supervisor,sno⟩⟩})
6. addAtt(⟨⟨staff,sname⟩⟩, notnull, {x,y | (x,y) ∈ ⟨⟨tutor,sname⟩⟩ ∧ x ∉ ⟨⟨supervisor,sno⟩⟩ ∨ (x,y) ∈ ⟨⟨supervisor,sname⟩⟩})

7. addAtt(⟨⟨staff,Dept⟩⟩, null, {x,y | (x,y) ∈ ⟨⟨supervisor,Dept⟩⟩ ∨ x ∈ ⟨⟨tutor⟩⟩ ∧ x ∉ ⟨⟨supervisor,sno⟩⟩ ∧ y = null})
8. addRel(⟨⟨monitors,sno,id⟩⟩, {x,y | (y, x) ∈ ⟨⟨ug,sno⟩⟩ ∧ y ∉ ⟨⟨phd,id⟩⟩ ∨ (x,y) ∈ ⟨⟨supervisor,sno,id⟩⟩}, (x,y) ∈ ⟨⟨monitors,sno,id⟩⟩ → x ∈ ⟨⟨staff,sno⟩⟩ ∧ y ∈ ⟨⟨student,id⟩⟩)

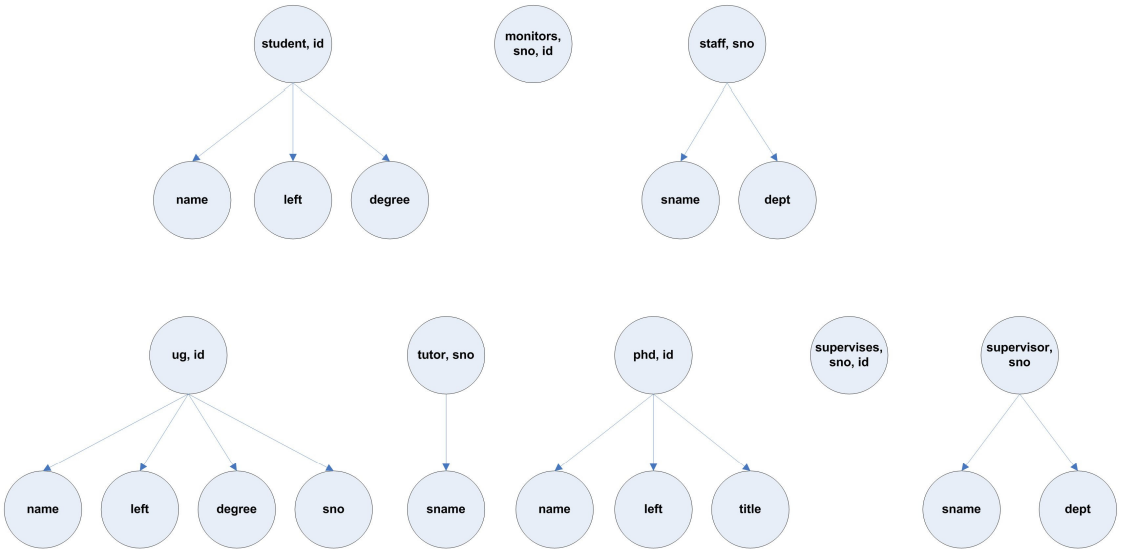

Figure 18 – Integrated schema of S_1, and S_2

The 'addRel' and 'addAtt' specify how the constructs of the global schema are defined.

9. conAtt(⟨⟨tutor,sname⟩⟩, notnull, {x,y | (x,y) ∈ ⟨⟨staff,sname⟩⟩ ∧ x ∈ ⟨⟨tutor,sno⟩⟩})
10. conRel(⟨⟨tutor,sno⟩⟩, {x | x ∈ ⟨⟨staff,sno⟩⟩ ∧ (y,x) ∈ ⟨⟨ug,sno⟩⟩})
11. conAtt(⟨⟨Ug,name⟩⟩, notnull, {x,y | (x,y) ∈ ⟨⟨student,name⟩⟩ ∧ x ∈ ⟨⟨Ug,id⟩⟩})
12. conAtt(⟨⟨Ug,Left⟩⟩, null, {x,y | (x,y) ∈ ⟨⟨student,left⟩⟩ ∧ x ∈ ⟨⟨ug,id⟩⟩})
13. conAtt(⟨⟨Ug,degree⟩⟩, notnull, {x,y | (x,y) ∈ ⟨⟨student,degree⟩⟩ ∧ x ∈ ⟨⟨ug,id⟩⟩})
14. conAtt(⟨⟨Ug,sno⟩⟩, notnull, {x,y | (y,x) ∈ ⟨⟨monitors,sno,id⟩⟩ ∧ x ∈ ⟨⟨ug,id⟩⟩})
15. conRel(⟨⟨tutor,sno⟩⟩, {x | x ∈ ⟨⟨student,id⟩⟩ ∧ (x, 'phd') ∉ ⟨⟨student,degree⟩⟩})
16. delRel(⟨⟨supervises,sno,id⟩⟩, {x,y | (x,y) ∈ ⟨⟨monitors,sno,id⟩⟩ ∧ y ∈ ⟨⟨phd,id⟩⟩}, (x,y) ∈ ⟨⟨supervises,sno,id⟩⟩ → x ∈ ⟨⟨supervisor,sno⟩⟩ ∧ y ∈ ⟨⟨phd,id⟩⟩)
17. delAtt(⟨⟨supervisor,sname⟩⟩, notnull, {x,y | (x,y) ∈ ⟨⟨staff,sname⟩⟩ ∧ x ∈ ⟨⟨supervisor,sno⟩⟩})
18. delAtt(⟨⟨supervisor,dept⟩⟩, notnull, {x,y | (x,y) ∈ ⟨⟨staff,dept⟩⟩ ∧ x ∈ ⟨⟨supervisor,sno⟩⟩})
19. delRel(⟨⟨supervisor,sno⟩⟩, {x | x ∈ ⟨⟨staff,sno⟩⟩ ∧ (x,y) ∈ ⟨⟨staff,dept⟩⟩ ∧ y ≠ null})
20. delAtt(⟨⟨phd,name⟩⟩, notnull, {x,y | (x,y) ∈ ⟨⟨student,name⟩⟩ ∧ x ∈ ⟨⟨phd,id⟩⟩})
21. delAtt(⟨⟨phd,Left⟩⟩, null, {x,y | (x,y) ∈ ⟨⟨student,left⟩⟩ ∧ x ∈ ⟨⟨phd,id⟩⟩})
22. conAtt(⟨⟨phd,title⟩⟩, notnull, void)
23. delRel(⟨⟨phd,id⟩⟩, {x | x ∈ ⟨⟨student,id⟩⟩ ∧ (x, 'phd') ∈ ⟨⟨student,degree⟩⟩})

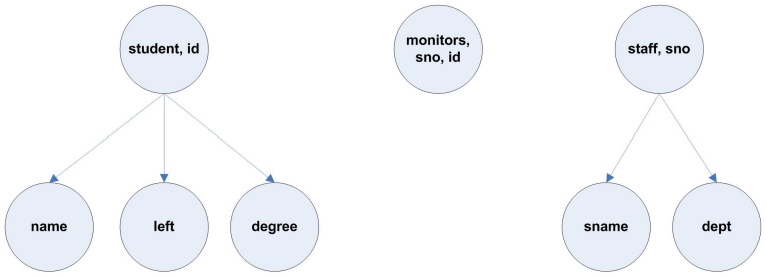

Figure 19 - Global schema S_g after applying all transformation rules

The transformation steps 1-23 are necessary to integrate S_1 and S_2 into S_g. First thing to do in this integration process is to create unification in a single schema $S_1 \cap S_2$, whose constructs are precisely those of S_1 and S_2. For simplicity this step will be skipped, since there are no commonly-named relations in S_1, S_2 and S_g the relation names will be used directly. Steps 1 to 8 describe how the constructs of S_1 and S_2 are used to create S_g. In fact, S_g is defined in terms of the local schemas. This corresponds to the GAV-approach and makes the schema grow. Steps 9 to 23 remove the constructs of S_1 and S_2 from this intermediate schema. Now the constructs of S_g remain. Note that the queries accompanying steps 9 to 15 show how the constructs of S_1 can be restored from those of S_g, and that the queries accompanying steps 16 to 23 do the same for S_2. Therefore this part of the transformation corresponds with the Local-as-View approach.

For transforming S_1 and S_2 into S_g two more transformation-steps are introduced: *conRel* and *conAtt*. These transformations remove a relation or an attribute like *delRel* and *delAtt*, except for the fact that they indicate that their counterpart *extRel* and *extAtt* may only partially restore the population described by *q*. These transformations are *complete* as described in section 3.1. The abbreviations "con" and "ext" stand for "contract" and "extend". Accompanying query *q* contains either the tuples being contracted or extended by this construct, or *void* when no information about the extent of the construct is available. This distinction between partial and complete (exact) derivation provides more information about a schema construct which results in a more precisely described relation between source and global schemas.

Consider the following: a new relation is added to this global schema. If we want to add the relation cs(id) containing all students following the course computer science and the fact that all students with degree='G500' study computer science, as do some students with degree='GG15'(and also that no other students study computer science), the following applies:

1. addRel(⟨⟨cs, id⟩⟩, {x | (x,y) ∈ ⟨⟨ug, degree⟩⟩ ∧ y = 'G500'})

however this would be incorrect, since it states that *cs* contains just those students on 'G500', and no other students.

2. extRel(⟨⟨cs, id⟩⟩, {x | (x,y) ∈ ⟨⟨ug, degree⟩⟩ ∧ y = 'G500'})

would be correct, since it states that *cs* contains those students on 'G500', plus some others which can in no way be determined.

3. extRel(⟨⟨cs, id, *void*⟩⟩

would be incorrect, since it states that there is no way of determining any of the students that belong to *Cs*, when in fact it's known that all 'G500' students belong to *cs*.

If it was the case that only students with degree 'G500' study CS then 1 would be correct. 3 would be correct in case some students with degree 'G500' and some students with degree 'GG15' study CS.

3.3.4 Simplifying the Both-as-View integration process using macros

The Both-as-View method combines the benefits of both the LAV- and GAV-method. Any reasoning possible with the views defined in those approaches can also be realized using the Both-as-View-definition. It can be argued that employing the Both-as-View-method and therefore creating the list of transformations makes data integration more complex, and that Both-as-View is just a parallel use of GAV and LAV. In the next section a counterargument for this idea will be shown.
Well-known schema equivalences can be expressed as Both-as-View-transformation macros. Those macros define a sequence of transformation steps, which rather simplifies the generation of the total transformation. This approach makes it possible to specify Both-as-View transformations with as much ease as GAV or LAV derivation rules. This is done in two phases:

Phase 1: Apply well-known schema equivalences to make local schemas union-compatible.

Keep in mind that a union of the different source schemas is essential for achieving a global schema. Schemas may describe similar relations in different ways, and therefore schemas need to be union-compatible. In the preceding example this would mean that the relation *ug* in S_1 would be decomposed so that the *sno* attribute was held in a separate relation.

This would make it union-compatible with relation *supervises* in S_2. This operation, which moves an attribute of a relation into a new relation, can be described by the following macro:

```
decompose(rel, att)=
    addRel(rel, {x, y | ⟨y, x⟩ ∈ att})
    delAtt(att, {x, y | ⟨y, x⟩ ∈ newRel})
```

This macro makes use of decomposition which means that a relation *R* with keys k_1 ,.., k_n and attributes a_1 ,.., a_m can be decomposed into *m* relations, each containing all keys *k* and one of the attributes *a*. This is done by applying the *decomposition rule*. We can rewrite any derivation rule of the form:

$R(\vec{k}_n, \vec{a}_m) = \{\vec{x}_n, \vec{y}_m \mid E\}$ as *m+1* equivalent derivation rules:

$$R(\vec{k}_n) = \{\vec{x}_n \mid E\}$$
$$R(\vec{k}_n, a_1) = \{\vec{x}_n, y_1 \mid E\}$$
$$\vdots$$
$$R(\vec{k}_n, a_m) = \{\vec{x}_n, y_m \mid E\}$$

This relation *R* is therefore represented by the following transformation steps:

$$\text{addRel}(<< R, \vec{k}_n >>, \{\vec{x}_n \mid D(E)\})$$
$$\text{addAtt}(<< R, a_1 >>, \{\vec{x}_n, y_1 \mid D(E)\})$$
$$\vdots$$
$$\text{addAtt}(<< R, a_m >>, \{\vec{x}_n, y_m \mid D(E)\})$$

Where function *D(E)* decomposes expression *E* in the same way as *R* is decomposed. The *decomposition rule* is used to convert Both-as-View to GAV or LAV and vice versa.

Furthermore, the *title* attribute needs to be removed from relation *phd*, since this attribute is not presented in S_g . Attribute *degree* with value "phd" has to be added to *PhD* in S_2, assuming that all PhD students are registered for a PhD degree. Those three steps make the relation *ug* union compatible with *PhD*.

Relation *tutor* needs to be extended with an attribute *dept*. Every value of *dept* has to be null, since no information about its value is present. This makes the relation union compatible with *supervises* in S_2.

Phase 2: apply a union operation to integrate the local schemas into a global schema.

In phase 1 the relations in the source schemas have been transformed in such a way that they can be united. This is done in phase 2. When joining two schemas, in conflicting situations there should be some bias to decide which schema holds the correct values.

In the example this would mean that three union operations are applied: define the *student* relation as a right union of *ug* and *PhD*. The term right union means that attribute values of *PhD* are preferred in case of similar keys. Define *staff* as a right union of *tutor* and *supervisor* and define *monitors* as a right union of *tutors* and *supervises*.

1. decompose (⟨⟨ug, sno⟩⟩, ⟨⟨tutors, sno, id⟩⟩)
2. extAtt (⟨⟨tutor, dept⟩⟩, notnull, {x, y| x∈⟨⟨tutor, sno⟩⟩ ∧ y=null })
3. conAtt (⟨⟨phd, title⟩⟩, notnull, *void*)
4. addAtt (⟨⟨phd, degree⟩⟩, notnull, {x, y| x ∈ ⟨⟨phd, id⟩⟩ ∧ y='phd'})
5. rightUnion (⟨⟨student, id⟩⟩, ⟨⟨ug, id⟩⟩, ⟨x, y⟩ ∈ ⟨⟨student, degree⟩⟩ y≠'phd',
   ⟨⟨phd, id⟩⟩, ⟨x, y⟩ ∈ ⟨⟨student, degree⟩⟩ ∧ y='phd')
6. rightUnion (⟨⟨staff, sno⟩⟩, ⟨⟨tutor, sno⟩⟩, ⟨x, y⟩ ∈ ⟨⟨staff, dept⟩⟩ y=null,
   ⟨⟨supervises, sno⟩⟩, ⟨x, y⟩ ∈ ⟨⟨staff, dept⟩⟩ ∧ y≠null)
7. rightUnion (⟨⟨monitors, sno, id ⟩⟩, ⟨⟨ug, sno⟩⟩, ⟨y, x⟩ ∈ ⟨⟨monitors, sno, id⟩⟩ ∧ x ∉ ⟨⟨phd, id⟩⟩,
   ⟨⟨supervises, sno, id⟩⟩, ⟨x, y⟩ ∈ ⟨⟨monitors, sno, id⟩⟩ ∧ y ∈ ⟨⟨phd, id⟩⟩)

Figure 20 – BAV macro specification

All those steps can be defined as a macro, as is described in [1]. Figure 20 shows the macro specification to integrate S_1 and S_2 into S_g. This specification is similar to the transformation steps in chapter 3.3.3. When all macro-steps are expanded into transformation steps, some redundant steps will be the result. For example, attributes might be introduced and be removed afterwards so in fact they cancel each other out.

3.4 Schema evolution

3.4.1 What is schema evolution?

Typically constructs of a source or target schema in an integration system may evolve. When schemas evolve it's important that the integration framework supports the process of evolution to be able to reuse preceding integrations as much as possible.

Schema evolution in the sources or targets could be for example a small change in the definition of an entity. The definition of an employee in a source schema may change over time because of changed business requirements. Think of the addition of an attribute to the employee entity for storing more information than previously needed. Adding this attribute is a change in the construct of the employee entity. This addition may or may not have impact on the integration process.

An example of schema evolution is shown in Figure 21.

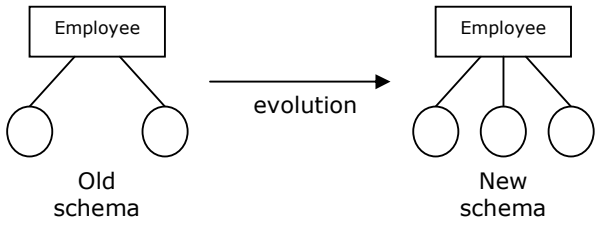

Figure 21 – Schema evolution

Researchers of [14] claim that the Both-as-View integration framework readily supports schema evolution by describing the transformation path between the old schema and new schema constructs. How schema evolution is supported by the Both-as-View approach is explained in the next paragraph.

3.4.2 Schema evolution in the BAV approach

Evolution of a schema can be expressed in a more formal way as: *schema S_i evolves to S'_I*. This section will show how Both-as-View handles evolution of the local as well as the global schema. In contrast to the Global-as-View and Local-as-View approach, transformation pathways don't have to be regenerated but can be modified when schemas evolve. Schema evolution will be treated as a single primitive transformation. Evolution consisting of a sequence of primitive transformations can be handled by applying the treatment for each single transformation step in sequence.

Researchers of [14] describe how each transformation step (rename, add, delete or extend) can be used to describe an evolution of a local or global schema. Their algorithms used for schema evolution are mainly automatic. However, domain knowledge may be required regarding the semantics of the new schema construct. This knowledge enables to determine the exact set of transformation rules for describing the transformation pathway from the old to the new schema construct.

3.5 Both-as-View applied in a data warehouse environment

3.5.1 AutoMed architecture

AutoMed stands for Automatic Generation of Mediator Tools for Heterogeneous Database Integration. This research project implements the Both-as-View integration technique. The AutoMed framework consists of a low-level data model named "hyper graph-based data model" (HDM) and a set of primitive schema transformations defined in this data model. How exactly the HDM is defined is described in section 3.3.1. Higher-level data models and schema transformations for them can be defined using the HDM and its primitive transformations. In this way, HDM's primitive transformations are used as building blocks to describe higher-level model languages. It is therefore possible to define model languages like XML, relational, UML and ER in terms of HDM, and those languages can be used to model source and global schemas. AutoMed provides a "Model Definition Tool" to assist in creating definitions of higher-level model languages. Those definitions are stored in the "Model Definitions Repository" (MDR).

Before transforming the local schemas into a global schema, some naming conventions have to be applied. Therefore, local schemas are transformed to intermediate schemas that adhere to a common dictionary or ontology. The transformation pathway between the local schemas and the global scheme is defined by a sequence of primitive transformations, existing of one or more primitive transformations. Each step incrementally transforms the schemas, by adding, deleting or renaming a single schema construct. Each *add* or *delete* step is accompanied by a query which specifies the extent of the construct. This extent is used to make the transformation process reversible. Besides *add*, *delete* and *rename* transformations AutoMed also uses *extend* and *contract* transformations. As you can see, this way of thinking fully corresponds to the BAV-approach described in section 3.3.

AutoMed provides a "Schema Transformation & Integration Tool" to create new source, intermediate and global schemas and transformation pathways between schemas. The schemas of data sources, intermediate schemas, global schemas and transformation pathways between them are stored in the "Schema and Transformations Repository" (STR).

The "Schema Evolution Tool" supports schema evolution in the MDR, as described in section 3.4. Both the Schema Evolution Tool and the Schema Transformation & Integration Tool performs optimization to the transformation pathways. That means when a composite transformation is of the form $T;t;T';\bar{t};T''$, where T, T' and T'' are transformation sequences, t is a primitive transformation and $\bar{t}$ is its inverse, the transformation can be rewritten as $T;T';T''$, provided that there are no references in T' to the construct being renamed, added or deleted by t.

Up to here the tools and repositories have been described. The intelligence of AutoMed is represented in the "Global Query Processor" and the "Global Query Optimizer". Queries posted against the global schema are translated into a functional query language named IQL (Intermediate Query Language). The query processor executes IQL queries by deriving a GAV view from the transformation pathways and schemas stored in the STR. After being optimized by the query optimizer, queries are divided in sub-queries and translated into query languages supported by the date sources.

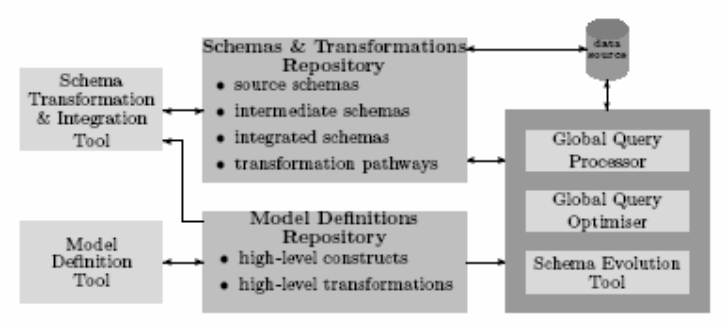

Figure 22 – The AutoMed architecture

3.5.2 AutoMed for data warehousing

In the previous sections, the AutoMed architecture has been explained. This section will describe how a data warehouse can be constructed using AutoMed, and what advantages this brings about. In AutoMed terms a data warehouse built and maintained using the AutoMed approach is named "AutoDW". An AutoDW consists of 2 parts: the AutoMed part, which includes the AutoMed Repository and the AutoMed Repository API, respectively abbreviated as AR and ARAPI. Further it includes the data warehouse part, which is the main functional unit of the AutoDW.

The AutoMed part is the administrative part of the warehouse, and will not be accessible by the end-user who will be using the warehouse. The ARAPI schematizes all data warehouse repositories. Those repositories are for example tables, materialized and virtual views. Furthermore, the ARAPI creates pathways between those schemas. The AR is created and managed by the ARAPI, and used to store all meta-data, which describes the data warehouse activities. These data describe the warehouse architecture, data structures and data flows of the AutoDW, and is used to support data warehouse activities as query rewriting, data mining and data lineage tracing. It is under the control of the AutoDW administrator.

The designing of an AutoDW can be divided into a logical and a physical part. The logical part involves creating (intermediate) schema's and pathways between them. Figure 23 shows how schemas and transformations describe the pathway from data source to data mart.

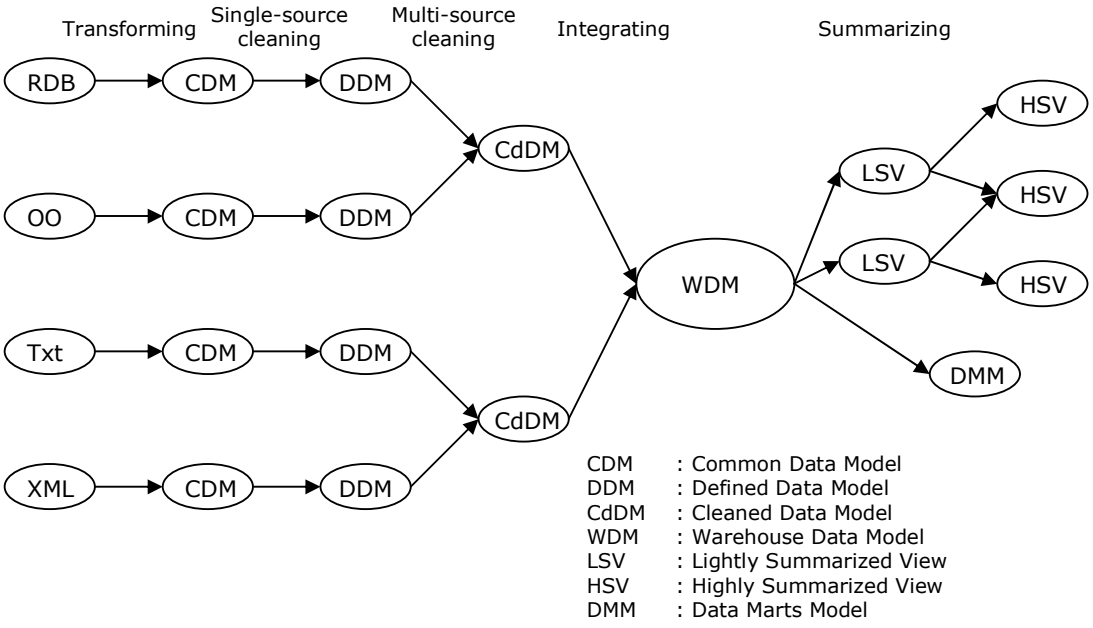

Figure 23 - [24] Logical design of an AutoDW

The first step in designing an AutoDW is creating schemas for a set of heterogeneous sources. Those schemas have to be transformed into schemas according to a common data model. Those models could be identical or heterogeneous. Next, single-source and multi-source cleaning has to be applied. DDM is the date model after single-source cleaning. DDM could be of the same type as its CDM or not. Schema's which have to be integrated for multi-source cleaning have to be similar anyway. After cleaning, cleaned data model CdDM is created. Again, different CdDm could be homogeneous or heterogeneous. The WDM, the data model for detailed data into which they are integrated is uniform or homogeneous, but does not have to be in a fixed modeling language. That is, the WDM can be either relational, XML, UML, HDM or any other kind of modeling language specified in the repository. The next steps transform the schema into light or highly summarized views, and data marts model. Modeling languages of those schemas are usually (but not necessary) similar to the WDM.

The physical part of building an AutoDW concerns the actual data and how it flows from heterogeneous sources to summarized data marts. The process has several steps. There is an ongoing discussion whether the result of each step should be materialized or un-materialized, especially for the summarized views.

The first step in the physical process is extracting data from data sources to the staging area. Data in the staging area is transformed into a common data model. No integration is done here, schemas are just being transformed. In the next step, single-source and multi-source data is cleaned. The cleaning process includes processes like removing inconsistencies from data, removing duplicate entries and resolving naming conflicts.

Next, the cleaned data is integrated into the data warehouse. Data is being enriched by way of summarizations, and stored in data marts. At this point, data is ready to be used by for example data-mining and OLAP-applications. Some data can be moved from current detailed data to older detailed data, if necessary.

4. Lineage

The meaning of the word lineage is to find the origin of an entity. People try to find their lineage for different kind of reasons which can result in a genealogical tree. The genealogical tree tells us information about our ancestors like their name, age and profession.

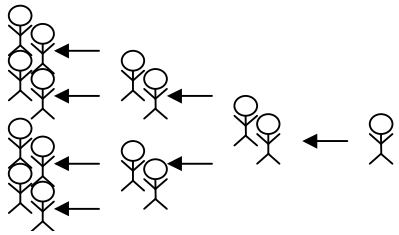

Figure 24 – Lineage of people's origin

When people try to find their lineage they actually try to find the path of ancestors which is the full lineage of their origin. The term lineage can not only be used for people trying to find their origin. Lineage can also mean trying to find the path of integrated information to its origin. For example suppose a system integrates employee and patient data. The lineage for a person in the integrated system would be its original employee and patient information as shown in Figure 25.

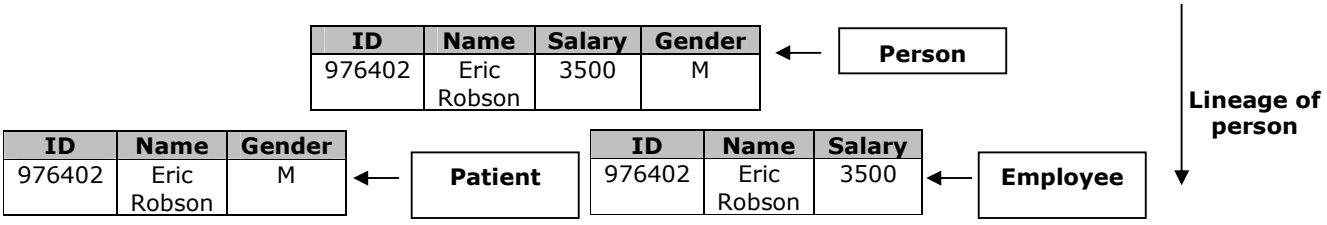

Figure 25 - Lineage in an integration system

4.1 Why lineage?

As people have several reasons why they are interested in their ancestors there are also reasons why lineage is important in integration systems. The possibility of lineage in integration systems enables tracking which sources and transformations applied to the integrated information. Tracking the integrated information gives several benefits and applications like in-depth data analyses and founding report results. Lineage may even be required, for example by the law, that one can track integrated information.

During the integration process data undergoes a series of transformations that vary from simple algebraic transformations or aggregation to complex cleansing procedures. After the integration process has taken place one isn't aware of the composition of the data in the integrated system. Tracking the path from the integration system back to the source systems is called lineage tracing. The lineage problem is stated by the authors of [10] as: '*tracing tuples in the integrated system back to its origin from which they were derived'*.

Preceding researches on data lineage limited to integration systems with relational materialized views over the source systems. A heterogeneous integration system, however, may contain cleaned, organized and summarized data imported from heterogeneous source systems, as stated by [10]. This adds more complexity to the lineage problem compared to limiting to relational materialized views. In the case of relational materialized views one has the luxury of a fixed set of operators or algebraic properties while they lack in a heterogeneous integration system.

4.2 Example of lineage in integration systems

Suppose an analyst would like to have an overview of the top two functions of employees who were ill in the last year. To be able to produce this report information needs to be gathered from source systems containing employee and patient information. After gathering the information needed several transformations need to be applied on the available data to calculate the top two functions of employees who were ill. The series of transformations applied and the sources used are listed in the figure below.

ID	Name	Function	Salary	Birth date
348734	Eric Robson	Manager	3500	04-02-69
213453	John Doe	Analist	2000	18-11-73
674543	Rodger Green	Manager	5000	23-06-65
432790	Steve Mangleton	Guard	1500	18-09-76

Source S1: Employees

ID	Gender	hospitalized
348734	Male	05-12-04
674543	Male	14-05-04
432790	Male	28-02-04

Source S2: Patients

ID	Frequency	Year
Manager	2	2004
Guard	1	2004

Result R1: EmployeeIllness

The tables shown above contain sample data. The sources contain information of employees and patients while the result table contains integrated data which contains the answer on the question which function had the most employees being ill in the last year. The process of integrating goes through a series of transformations.

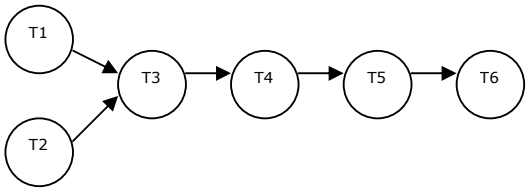

Figure 26 – Graph of transformations

T	Action
1	Select employees
2	Select patients where hospitalized = 2004
3	Join employees and patients
4	Aggregate, group by and sum
5	Remove obsolete columns
6	Add 'Year' column

Transformation steps to integrate S1 and S2

The transformation graph defines which steps need to be taken to integrate the sources into the required result. If we take a look at the results R1 we see an overview of which functions had the most employees being ill in the year 2004. The exact lineage for the row <Manager,2,2004> is the data in the figure below.

ID	Name	Function	Salary	Birth date
348734	Eric Robson	Manager	3500	04-02-69
674543	Rodger Green	Manager	5000	23-06-65

Source S2: Patients

ID	Gender	hospitalized
348734	Male	05-12-04
674543	Male	14-05-04

Source S1: Employees

Lineage results for <Manager,2,2004>

As one can see the table Employees and the table Patients contributed to the row <Manager, 2, 2004> in the table EmployeeIlllness. The row in the table EmployeeIllness tells us the most employees being ill in 2004 are managers. The results in the target table are derived from the rows in the tables shown in the figure above. This result is the exact lineage for <Manager, 2, 2004>.

4.3 Approaches to lineage

Preceding research on lineage in integration systems limited to lineage through relational views while an integration system typically contains heterogeneous sources. A study of [10] handles lineage through general transformations however a framework for expressing transformations in heterogeneous integration environments lacks, most of [10] algorithms recall the view definition and examines each item in the data source to decide if the item is the data lineage of the data being traced. This can be very expensive if the view definition is a complex one and enumerating all items in the data source is impractical and can be very expensive for large data sets. Never the less in paragraph 4.3.1 the approach of [10] is discussed and in 4.3.2 the proposed lineage approach, which has a framework for expressing transformations in heterogeneous environments, will be introduced.

4.3.1 A limited lineage approach

The authors of [10] give lineage tracing formulae for general transformations which, when available, use known structures or properties of these transformations and otherwise provide in an alternative for processing data lineage when this information is not available. The tracing algorithms are applicable on single transformations, linear sequence of transformations and random a-cyclic transformation graphs.

4.3.1.1 Definition of a transformation

According to the authors of [10] all transformations should be stable and deterministic. Stable means that a transformation *T* can never produce unreal output e.g. $T(\varnothing) = \varnothing$. A transformation is deterministic when it always generates the same output set given the same input set.

Generally a transformation expects the full input set to be able to generate each output item. In many cases however a much finer relation between each input item and its output item can be defined. An output item may be derived from a much smaller subset of input items (maybe exact 1 input item) in contrast to the full input set. It's important to have some kind of knowledge of the specification of a transformation to be able to process lineage. If there is no knowledge available of what a transformation does then every source element may have contributed to the derivation of an output item.

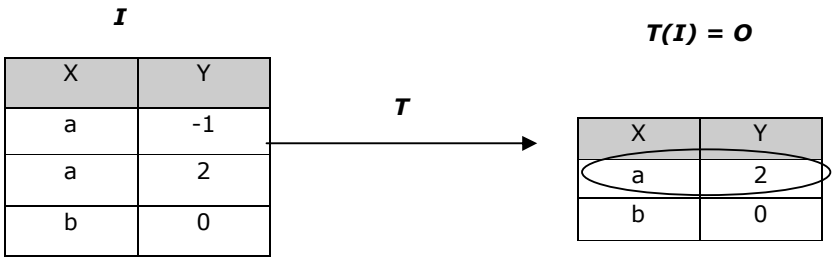

Figure 27 – A transformation instance

Given a transformation *T* -> *T(I) = O,* the lineage of <a,2> depends on the definition of transformation *T*, see Figure 27. Suppose the transformation *T* filters the input set on negative values then the lineage for output item <a,2> would only contain input item <a,2>. Now suppose the transformation groups all input items and multiplies the sum of Y with 2, the lineage for output item <a,2> would then be {<a,2>, <a, -1>}. This example showed that knowledge of the functionality of a transformation is important to be able to determine the lineage for an output item.

If a transformation is a standard relational operator or view then it's possible to determine the exact lineage of each output item. When there is no knowledge available of the functionality of a transformation the lineage of an output item needs to be defined as the full input set. In practice the truth is somewhere in between, transformations are not standard relation operators but they do contain known structures and properties which can be used for lineage calculation. For example the properties of a transformation be defined by the developer of a transformation, the properties can be derived from the definition of a transformation or maybe it's possible to study the behavior of a transformation to determine its functionality.

4.3.1.2 Properties and types of transformation

The authors of [10] define three general properties of transformations which can help in calculating lineage:

- A transformation belongs to a class of transformations (dispatcher, aggregator, black box)
- There can be one or more schema mappings available for the transformation which defines how the output elements relate to the input elements.
- A transformation can have a tracing procedure or inverse which can be used to calculate lineage.

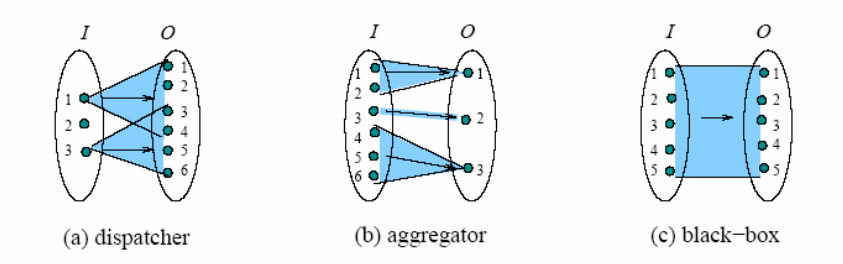

Figure 28 – [10] Main transformation classes

If a transformation contains many of these properties there needs to be determined which properties can be used best for calculating data lineage. Determining which properties can be used best can be done by using a property hierarchy defined by [10]. As mentioned earlier [10] makes a distinction in three classes of transformations. These classes are named dispatcher, aggregator and black-box transformation. There exist more fine-grained transformation classes but as we can see further on these transformations can be categorized under one of the three main classes (dispatcher, aggregator and block-box) of transformations.

A transformation is of the class dispatcher when every input element independently generates zero or more output elements. In Figure 28(a) can be seen that element 2 doesn't contribute to any output element, therefore that transformation is called a dispatcher. The formal definition of a dispatcher class is given by [10] as follows:

$$\forall I, \mathcal{T}(I) = \bigcup_{i \in I} \mathcal{T}(\{i\})$$

Figure 29 – [10] Formal definition of dispatcher class

The second main transformation class is called the aggregator class. The fundamentals of an aggregator class are that every input element should contribute to an output element and for all input sets there exists a unique partition (no overlap). As can be seen in Figure 28(b) there are three unique input partitions which contribute to an output element without having overlap. A note with the aggregation transformation class is the fact that an aggregator can also be a dispatcher, see the example below.

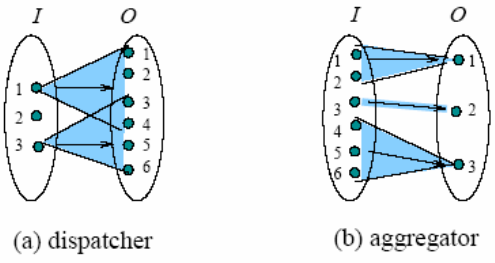

Figure 30 – [10] Aggregator is a dispatcher

In Figure 30(b) every input element generates zero or more output elements. So the aggregator has the same properties as the dispatcher. In addition to the dispatcher the transformation applies to the rule that every input element should contribute to an output element and every input set forms a unique partition, see partitions {1,2},{3},{4,5,6} in Figure 30(b). This addition makes this transformation an aggregator.

To increase the efficiency of the aggregator transformation [10] introduces two subclasses of the aggregation transformation class. These subclasses use special properties to calculate lineage more efficient than the main aggregator class.

The last main transformation class is called the black-box transformation. This class is meant for those transformations which aren't a dispatcher or aggregator class and there is no procedure available for calculating lineage. The only thing which can be said about this class of transformation is the fact that the full input set is the lineage of the output set. There is no knowledge available to calculate a fine-grained lineage result for this class of transformation. Formal definitions of the main classes and subclasses of transformations can be found in [10].

As [10] defines more subclasses of transformations to increase efficiency for calculating lineage there is no need to discuss them all. The most important thing is the distinction between classes of transformations and the unique properties of each transformation class which enables calculation of lineage. A property hierarchy is developed which enables determining which class can be used best to calculate lineage results. This property hierarchy is shown in Figure 31.

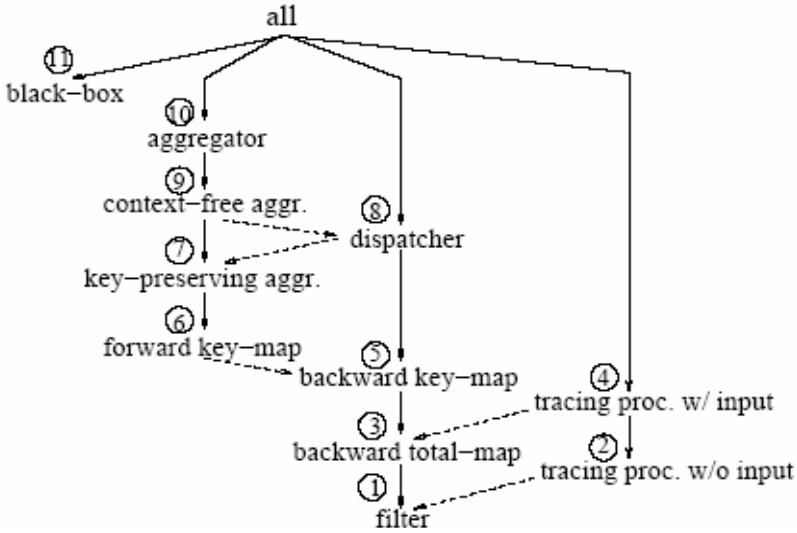

Figure 31 – [10] Property hierarchy of transformation classes

Underlying classes are never less efficient and will be in most cases more efficient than overlying classes.

4.3.1.3 Combining transformations

Now a distinction has been made between classes of transformation it's time to look into multiple transformations. Typically an integration system contains many transformations in order to integrate information. The integration example in paragraph 4.2 showed us a simple integration which takes six transformations to calculate employee illness. In practice it seems that sixty or more transformations are a realistic estimation of transformations needed for common integration processes. The researchers of [10] discuss if transformations can be combined to make lineage calculation more efficient and use the following strategy for handling a sequence of transformations:

- When a graph of transformations is defined, normalize it by grouping transformations. Determine which intermediate results should be stored based on the properties of the remaining transformations.
- Intermediate results should be stored when loading the transformations
- Lineage can be calculated for each output item with the given procedure using the normalized sequence of transformations

To be able to determine which combination of transformations contributes to the performance of a transformation graph an algorithm is given by [10]. However they note this algorithm is not optimized. For an optimal calculation a cost model should be developed based on transformation

properties, estimated resource, estimated storage and the expected workload to be able to calculate the most efficient grouping of transformations.

4.3.1.4 Performance issues

Using the properties of transformation classes combined with the algorithm for optimizing the process of lineage calculation an estimation can be made of what the performance will be for each single transformation algorithm whether they are combined or not. The results from [10] estimations are shown in Figure 32.

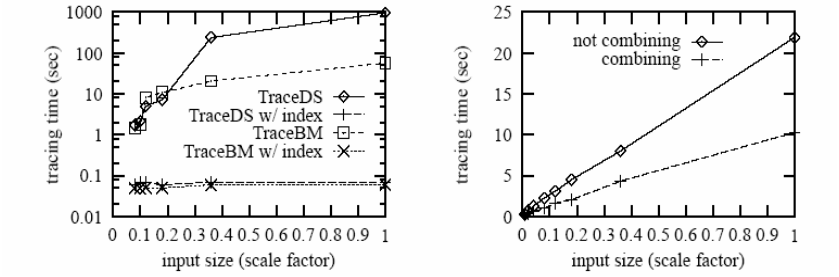

Figure 32 – [10] Performance estimation

As one can see grouping transformations using the algorithm provided by [10] or a jet to be developed algorithm drastically reduces the workload.

4.3.1.5 Summary

The authors of [10] give a useful framework for handling lineage through a sequence of transformations. A distinction is made between different classes of transformations to be able to determine what algorithm is the most efficient for calculating lineage. Besides this distinction they also discuss the advantages of combining transformation steps to increase performance. Although they give an algorithm for combining transformations a much more sophisticated algorithm could be developed to optimize the combination of transformations. Finally a brief estimation of performance impact for each lineage algorithm is given. The analysis shows the improvement in performance when combining single transformation steps into groups of transformations.

4.3.2 Proposed lineage approach

Another lineage approach developed by the authors of [11] proposes lineage based on an integration framework for heterogeneous integration environments. This framework also called AutoMed approaches integration based on the Both-as-View integration approach explained in chapter 3.3. The proposed lineage approach uses the individual steps of building schema constructs to calculate the lineage, see chapter 0 for further details of these individual steps. The lineage calculation is done by traversing the pathway of individual steps in reverse order. Suppose a local schema is integrated by a sequence of transformation steps into a global schema, the pathway to integrate has transformation steps ts(1),……,ts(n-1). The lineage for a given tuple in the global database is determined by calculating the lineage for each transformation step ts(n). For each step the lineage will be calculated until the final lineage data is retrieved from the sources. Calculating lineage by using transformation steps is called: '*lineage tracing using schema transformation pathways*' [14].

4.3.2.1 Data lineage calculation using Both-as-View

As mentioned in the preceding paragraph lineage calculation is done by using the Both-as-View integration approach. The following example is taken from [14] to demonstrate how a simple relational model is integrated into a global schema which holds the total and average student results for specific departments. As can be seen in this example the transformation steps applied are in the same form as the examples in chapter 3.3.3.

Suppose there are two source relations who hold student results for the mathematics and the information science departments. An example snapshot of the actual values could be as followed:

Mathematics			
Dept	**CID (Course)**	**SID (Student)**	**Mark**
MA	MAC01	1	7
MA	MAC02	2	6
MA	MAC01	2	5
MA	MAC03	1	8
MA	MAC02	3	4
MA	MAC01	3	5
MA	MAC03	2	9

Figure 33 - Mathematics department

Information Science				
Dept	**CID (Course)**	**SID (Student)**	**CName**	**Mark**
IS	ISC01	3	Math	5
IS	ISC02	4	English	8
IS	ISC04	4	Pascal	4
IS	ISC02	4	English	9
IS	ISC03	3	C++	3
IS	ISC01	5	Math	7
IS	ISC04	3	Pascal	6

Figure 34 - Information Science department

The integration system generates information of the total and average score of each course of a department. The table holding this kind of information could contain the following data which is based on the source tables shown in the tables above.

Course_Results			
Dept	**CID**	**Total**	**Average**
MA	MAC01	17	5,6
MA	MAC02	10	5
MA	MAC03	17	8,5
IS	ISC01	12	6
IS	ISC02	17	8,5
IS	ISC03	3	3
IS	ISC04	10	5

Figure 35 – Total and average results for courses per department

To integrate the sources into the target a set of transformation steps is defined to integrate the schema constructs. These transformation steps are similar steps as discussed in chapter 3.3.3. A subset of the transformation steps from [14] is taken to explain the integration process.

1. addRel⟨⟨Details⟩⟩ [{'MA', k1, k2}|{k1,k2}←⟨⟨Mathematics⟩⟩]++
   [{'IS', k1, k2}|{k1,k2}←⟨⟨Information Science⟩⟩]
2. addRel⟨⟨Course_Results⟩⟩ distinct [{k, k1}|{k, k1, k2}←⟨⟨Details⟩⟩]

The first transformation step combines the information from the 'mathematics' and 'information science' tables into a temporary table called 'details'. After the first integration step the target is calculated by the second transformation step. This transformation step takes all distinct values from the combined temporary table 'details' and puts them into the target table 'Course_Results'. Values for the total and average columns are calculated by other transformation steps which are shown in detail in the example of [14].

This integration example is used in the further explanation of lineage. In the next paragraph the algorithms used to calculate lineage are explained. Followed by an example which demonstrates how these algorithms and the transformation steps can be used to trace a record in the target table back to its origin.

4.3.2.2 Algorithms used for lineage calculation

The proposed lineage approach makes use of individual transformation steps in the transformation pathway to calculate lineage data. This is done by going through the transformation pathway in reverse order step by step. For each SIQL query (which is a subset of the richer IQL query language) used in a transformation step the researchers of [14] developed a data lineage tracing procedure to calculate lineage. Using SIQL instead of IQL doesn't mean that some functionality isn't available in the SIQL query language; in contrast a more complex IQL query can be rewritten as a series of SIQL queries on intermediate schema constructs.

The researchers of [14] appoint four cases in which lineage formulae differ from each other. These four distinct cases are called:

- 'Materialized Tracing data / Materialized Source data' (MtMs)
- 'Materialized Tracing data / Virtual Source data' (MtVs)
- 'Virtual Tracing data / Materialized Source data' (VtMs)
- 'Virtual Tracing data / Virtual Source data' (VtVs)

The lineage procedures for each of the four cases mentioned above are stated in [14]. Using the DLT formulae for each SIQL query in these four cases enables the researchers to develop a procedure which calculates lineage for a whole transformation pathway. This process of calculating the lineage for a whole transformation pathway is demonstrated in Figure 36.

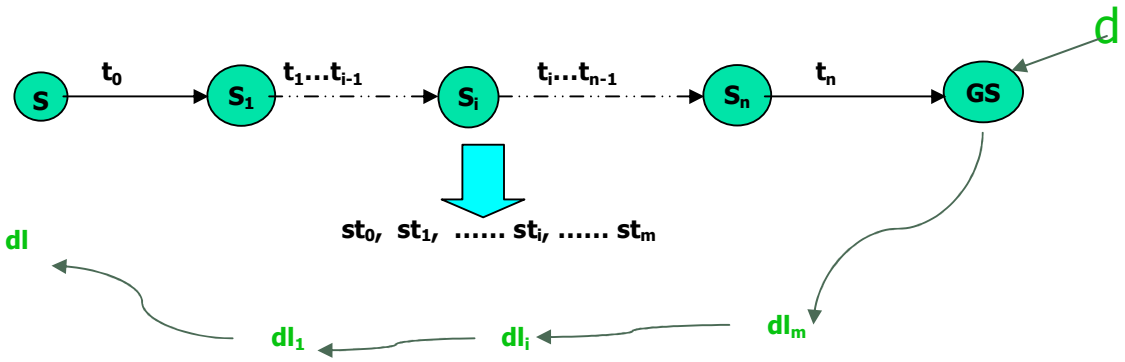

Figure 36 – [14] Data lineage by using transformation pathways

Knowing how the process of lineage calculation using schema transformations works. An example is given based on the schema integration example of 4.3.2.1. Suppose the lineage of tuple {'MA','MAC03','17','8,5'} needs to be calculated to determine which tuples from the sources contributed to the result. Following the process given in Figure 36 the transformation pathway used to integrate is traversed in reverse order to determine the lineage. The result of this process shows us that tuples {'MA','MAC03','1','8'} and tuples {'MA','MAC03','2','9'} from the table 'Mathematics' in chapter 4.3.2.1 contributed to the result tuple {'MA','MAC03','17','8,5'}.

4.4 Differences and improvements between lineage approaches

The researchers of [10] give a useful framework for calculating lineage in a sequence of transformations. However their approach limits to inspecting input sets and forces to determine if an element in the input set contributed to the result. Without a framework for expressing transformations in an integration environment, their algorithms can be very expensive when handling large data sets. The researchers of [13] address the lack of a framework for expressing transformations steps. They develop lineage algorithms based on each transformation step or a sequence of transformation steps which are expressed in the Both-as-View approach stated in chapter 3.3. The advantage of this approach is the usage of an integration framework for calculating lineage. Using the knowledge of each transformation step to determine lineage gives an advantage in accuracy and performance because one doesn't need to check if an input element had something to do with an output element, in contrast to [10]'s approach.

5. Quality improvement by using Both-as-View integration approach

Many researches have been done on integrating heterogeneous systems. Some of these have been discussed in this master thesis with the emphasis on the Both-as-View integration approach. In the next paragraphs the advantages of using the Both-as-View approach will be discussed, and how the combination of these advantages leads to a better-quality integration system is explained. The term "quality" can be interpreted in many ways; there is no universal agreement on its meaning. Therefore quality, in this master thesis, is a term to quantify the degree in which an object conforms to its specification.

5.1 Formal way of working

The Both-as-View approach enforces a very formal way of working. This prevents errors by appointing very strictly what has to be done. The chain of mutations starts and ends with schemas. The schemas of the source systems define "what you have", and the global schema defines "what you want to know". The pathway from the source schemas to the global schema represents the integration process. This pathway describes what has to be done to get what you want to know.

Each single transformation step in the pathway results into an intermediate schema. The intermediate schema enables a fine-grained view of what happens in the process. Each mediated schema, even for the smallest transformation step, can be used as a checkpoint to see if this step does what its supposed to do.

5.2 Flexible

The fact that the Both-as-View approach supports heterogeneous sources, which can be modelled in different modelling languages, makes the system very flexible and extendible. All common modelling languages can be expressed in the low-level model used by the BAV-framework.

5.3 Schema evolution

Another topic which contributes to quality improvement is schema evolution, this paragraph explains how. Schema evolution is explained in detail in chapter 3.4, briefly it means the change of local and global schemas during a period of time.

There are many reasons why schemas change. One of these reasons is a modification in the business requirements. The business requirements may for example require the registration of a new fact which wasn't registered before. To be able to register this fact some adaptations have to be made in the schemas of the registering system. If these systems are a part of an integration system the change may or may not have impact on the result of the integration. If the integration system evolves without the registration of the changes it's hard to track how the integration system behaved before this change. For example, if an answer has to be given on an issue like: '*what information was available about my customer in the year 1985?',* the set of changes from 1985 till now need to be available to reproduce the data set of the customer.

Preserving changes made in an integration system enables auditing an integration system over time. Auditing an integration system over time gives in-depth analysis of the integration process in the past. Several practical implementations can contribute to quality improvement of the integration. For example, comparing customer *x* in time fragment *y* with customer *x* in time fragment *y+1* gives insight in the evolution of this customer. Based on this comparison one can see if there is any inconsistent data and apply proper actions to correct any failures. This is one of the reasons why schema evolution, natively supported by an integration system, can contribute to quality improvement of an integration system.

5.4 Auditing

The research of Rob van Kempen on testing methods for integration systems mentions the aspect of testing the output of an integration system against expected values. Together with the ability to derive lineage, as is discussed in the research of Marc van der Wielen, this results in the ability to audit an integration system.

A Both-as-View system is used to provide information, based on several data sources, to a user. This information is the result of a pathway of transformations. Auditing an integration system is the process of examining whether information provided by the system is correct, and how it is constructed. This can be done by combining the method for testing described in the research of

Rob van Kempen and the approach to lineage described in the research of Marc van der Wielen. An audit trail is the sequence of data and transformations that lead to a certain result in the global schema, and validate or invalidate this result.

In cases where an organization must operate in compliance with rules and regulations enforced by government agencies an audit trail can facilitate in proving this compliance. An example of such cases is given in the master thesis of Rob van Kempen.

Suppose an element E_{audit} exists in the global schema, and an audit trail is requested to prove that the value of E_{audit} is correct. Data lineage can be used to retrieve the pathway of transformations that produced E_{audit} . Each transformation in this pathway goes with an intermediate schema and accompanying population, consult the research of Rob van Kempen for further details. This combination results in series of values and transformations in the integration system, from the end (global schema) to the beginning (local schema), that resulted in the value of E_{audit} . An example of an audit trail is illustrated by Figure 37 below.

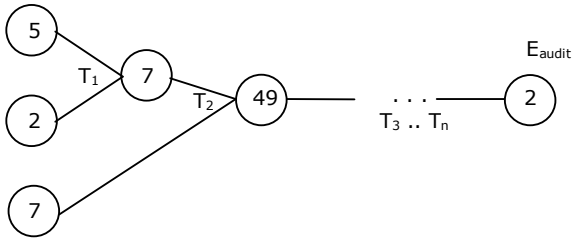

Figure 37 - Example audit trail

The combination of data lineage and methodologies for testing contributes to the overall quality of an integration system, because it's possible to check if an existing integration process complies with the given specifications. If it doesn't comply, one is able to pinpoint in which transformation step the output of the integration process diverges from the expected output, hereby indicating the error.

5.5 Horizontal and vertical drilling

The combination of schema evolution and data lineage can contribute to quality improvement of an integration system. This is explained in the next section, and clarified by way of an example.

A large organization, like a multinational, typically uses distinct systems for registering product-, sales-, employee-, procurement and customer data. These systems are distributed over different geographical locations and are commonly implemented by different vendors, e.g. the customer relation management system used by the American department is developed by SAP while the European department uses Microsoft.

The board of the multinational wants to have management information at its disposal for making strategic choices. This management information is retrieved by integrating information coming from the data registering systems. An example of retrieving management information through integrating operational systems is given in chapter 2.5. In these types of integration environments the data warehouse concept is commonly used.

By applying traditional integration approaches it is possible to retrieve the management information requested. However, without the support of data lineage or schema evolution important issues are not inherently supported. These issues can be illustrated by the following questions:

- *What information was available about my customer in 1985?*
- *How is this value calculated?*
- *What sources contributed to the results of this integration?*
- *How can I test if this value is what it should be?*
- *Can we trust this information?*
- *Are we able to proof that the annual revenue is correct and show how it is calculated?*

Questions like these can be answered if an integration system supports schema evolution and data lineage. Schema evolution is used to observe changes in the integration system trough time, while data lineage is used to observe changes in data from the source schemas to the global schema or

vice versa. Nevertheless it's worth mentioning that a combination of both enables vertically and horizontally 'drilling' through an integration system. The concept of horizontally and vertically drilling is best explained in Figure 38 below.

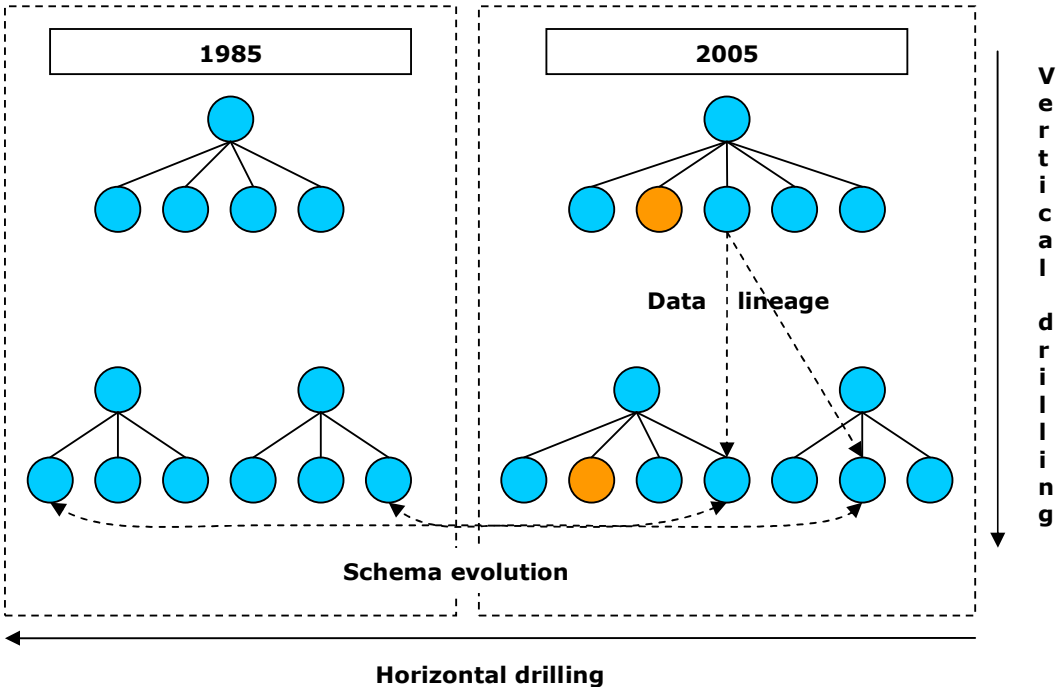

Figure 38 – Horizontal and vertical drilling

This figure shows how data lineage and schema lineage can be interpreted by the terms of vertical and horizontal drilling. Also, the power of combining data lineage and schema evolution is demonstrated. As can be seen in the figure above the combination enables a 'walkthrough' from a global schema in the present to a construct of a source schema in the past. Implementation of this concept contributes to quality improvement of integration systems and answers difficult issues, for example those mentioned on the preceding page.

6. Conclusions and future work

Integration of information systems is a complex task, because of the heterogeneity of those systems. To prevent ambiguity, models are used to describe these systems. Systems can be modeled in a different way, and in a different modeling language.

Integration systems are used to combine and enrich data in these information systems by expressing the relations between models. This thesis discussed different approaches for data integration. The Local-as-View approach and the Global-as-View approach suffered several shortcomings. Changes in source- or global schemas are not supported, they fail in expressiveness and heterogeneous sources are not considered.

Because of the growing demand for integrated information, the fact that sources are heterogeneous and are subject to change, these approaches are no longer satisfying.
This research focused on how a new integration approach, the Both-as-View approach, can contribute to overall quality improvement of an integration system.

The Both-as-View approach is designed to operate in a heterogeneous environment. It supports schema evolution, both on the source side as well as on the global side of the system. By defining primitive transformation steps, the pathway between local schemas and global schemas is defined. This involves a very formal way of working, on a conceptual level. Testing methods, techniques like schema evolution and data lineage offer insight in the integration process, which leads to overall quality improvement of the system.

These aspects, however, don't cover all aspects of an integration system which can contribute to quality improvement. Further research can be done in identifying aspects which could contribute to make the Both-as-View approach more practicable.

Support for data lineage in the new integration approach is a currently on-going research which will become more mature in the near future. A research on how to visualize data lineage using the algorithms provided would contribute to in-depth insight of the integration process. Combining this visualization with the proposed audit trail may lead to an applicable test- and audit suite for integration systems using the Both-as-View integration approach.

In the research of Rob van Kempen, a framework for developing and testing a Both-as-View system is defined. This theory does not concentrate on formulating test-cases. Test-cases in this testing framework are populations of the source systems. A future study on methods for defining test-cases that cover all aspects of an object under test would contribute to prevent malfunctioning of integration systems and thereby improving quality. Theories on creating test-cases in software development could be used as point of reference.

Because of the mathematical way of working, it might be interesting to examine if formal methods could assist in proving correctness of a Both-as-View system.

The testing framework described in the research of Rob van Kempen is based on development according to the V-Model. Research on how different development approaches could be employed would contribute to the usability of Both-as-View in different environments. Development phases, testing phases and the way of thinking in this research could be used as a foundation.

An integration system which is designed correctly, but provided with erroneous input, will still provide incorrect information. This idea is described as "garbage in, garbage out". Many theories of data quality, cleansing and filtering of data have been formed. Research on how these theories can be adapted to the Both-as-View approach could lead to a data quality framework.

7. Consulted resources

[1]. P.J. McBrien, A. Poulovassilis, *Data Integration by Bi-Directional Schema Transformation Rules,* Imperial College & University of London, 2003

[2]. P.J. McBrien, N. Rizopoulos, *AutoMed : A BAV Data Integration System for Heterogeneous Data Sources,* Imperial College, 2004

[3]. A. Cal, D. Calvanese, G. De Giacomo, M. Lenzerini, *On the Expressive Power of Data Integration systems,* University of Rome, 2002

[4]. D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, R. Rosati, *Description Logic Framework for Information Integration,* University of Rome, 1998

[5]. D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, R. Rosati, *A Principled Approach to Data Integration and Reconciliation in Data Warehousing,* University of Rome, 1999

[6]. S. Chaudhuri, U. Dayal, *An overview of data warehousing and OLAP technology*, SIGMOD Record, 1997

[7]. Y. Cui, J. Widom, J. Wiener, *Tracing the Lineage of View Data in a Warehousing Environment*, Stanford University, 1997

[8]. Y. Cui, J. Widom, *Practical Lineage Tracing in Data Warehouses,* Stanford University, 1999

[9]. Y. Cui, J. Widom, *Lineage Tracing for General Data Warehouse Transformations,* Stanford University, 2001

[10]. Y. Cui, J. Widom, *Lineage Tracing in a Data Warehousing System,* Stanford University, 2001

[11]. H. Fan, A. Poulovassilis, *Tracing Data Lineage Using Schema Transformation Pathways,* University of London, 2003

[12]. H. Fan, A. Poulovassilis, *Using AutoMed MetaData in Data Warehousing environments,* University of London, 2003

[13]. H. Fan, A. Poulovassilis, *Using Schema Transformation Pathways for Data Lineage Tracing, Univesity of London*, 2004

[14]. H. Fan, A. Poulovassilis, *Schema Evolution in Data Warehousing Environments – a schema transformation-based approach*, University of London, 2004

[15]. S. Hobo, *Guide to a succesfull datawarehouse*, University of Nijmegen, 1997

[16]. M. Jarke, M. Lenzerini, Y. Vassiliou, P. Vassiliadis, *Fundamentals of Data Warehouses,* National Technical University of Athans, 2000

[17]. F. Kemperman, *Design of Data Warehouses: Storage of data and integration of constraints, University of Nijmegen, 2000*

[18]. R. Kimball, *The Data Warehouse Toolkit 2nd edition: The Complete Guide to Dimensional Modeling,* 2002

[19]. H. van der Lek, F. Habers, M. Schmitz, *Stars and Dimensions*, DB/M, 2000

[20]. M. Lenzerini, *Data Integration: A Theoratical Perspective,* University of Rome, 2002

[21]. D. Lomet, J. Widom, *Special Issue on Materialized Views and Data Warehousing,* 1995

[22]. D. Theodoratos, *Semantic Integration and Querying of Heterogeneous Data Sources Using a Hypergraph Data Model,* University of New Jersey, 2002

[23]. http://www.doc.ic.ac.uk/project/2003/firstyeartopics/g03t06/, D. Varsani, J. Patel, K. Saleem, K. Merali, *Database Integration and Translation*, University of London, 2005

[24]. H. Fan, *Using AutoMed for Data Warehousing*, University of London, 2003