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

Emphasizing testing methods for integration systems

Master Thesis

Author	**:** Rob van Kempen
Studentnr.	**:** 9806792
Tutor	**:** Prof. Dr. Ir. Dhr. T. van der Weide
Reviewer	**:** Prof. Dr. Dhr. H.A. Proper
Date	: July 14, 2005

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 it can contribute to overall quality improvement of an integration system.

Preface

This thesis is the result of the graduation project I realised for the study Information Science ("Informatiekunde") I attended at the Radboud University of Nijmegen, Faculty of Science. I performed this research together with Marc van der Wielen, at BI4U, a consultancy agency specialized in Business Intelligence.

During the last 6 months I learned a lot. Sometimes the process of graduating seemed to be pleasant and smoothly, but there were also many times I wondered if the moment I would finish my thesis would ever take place. There are some people who, in some way, contributed to the result of this research, and who I would like to thank.

First of all, I would like to thank my friend and colleague Marc van der Wielen for the successful cooperation during this research. Marc, I had a very good time working with you, both from a professional and a personal point of view. We supplemented each other whenever needed and I'm very pleased with the results we achieved together.

During this research I spent many hours brainstorming and discussing with my tutor Theo van der Weide. His knowledge and abstract way of thinking exerted a great influence on the progress of this work. He taught me to look at problems in a different, more abstract way. Whenever I felt like I got stranded and did not exactly know how to continue, a meeting with Theo often provided a remedy for this. I'm very glad Theo guided and supported me during this research.

Furthermore, I would like to thank my friend and colleague Sam Geurts. Many discussions we held contributed to the final result. Moreover, I'm very grateful that Sam initiated the contact between me and BI4U, the company that supported this research. Niek Jetten, founder and managing director of BI4U, offered me the chance to carry out this research and to consult all knowledge available in the company. Niek, thanks for this great opportunity, and I'm glad our cooperation will continue in the future.
I also would like to thank my colleagues that advised me and provided me with useful information. Especially Paul Laman, an experienced consultant, who spent several hours discussing how all this theory could be employed in a real-life situation.

Special thanks goes to my mother, Willie Schamp, who raised me to the person I am and gave me the opportunity to attend two studies. Without the perseverance I inherited from her I wouldn't have made it this far.

For their interest in this research and the, sometimes highly necessary, leisure time I spent with them, I would like to thank my sister Marieke, my brother Koen, Henk, parents-in-law and all my friends. Alex, thanks for spell-checking this document.

And last but definitely no least I would like to thank my girlfriend Sofie Geurts. Whenever I came home after a long day of work, bad-tempered because things didn't work out as I expected, she managed to cheer me up. Sofie, thanks for everything that you are.

Rob van Kempen
July 2005

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 is realised in cooperation with Marc van der Wielen. Therefore, this thesis is to a large extent identical to the thesis written by Marc ([25]). His research focuses on how data lineage can be realized, and how data lineage contributes to improving 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 "testing" and accompanying techniques and methods are described. This is followed by an explanation on the testing of an integration system. It is based on the approach introduced in 3.3. By means of techniques used in software development, testing techniques and models for different aspects and different levels of detail of an integration system are described.

Chapter 5 discusses how the quality of an integration system can be improved. First the definition of 'quality' is given to state the meaning of this term in this document. The next part explains 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.

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.

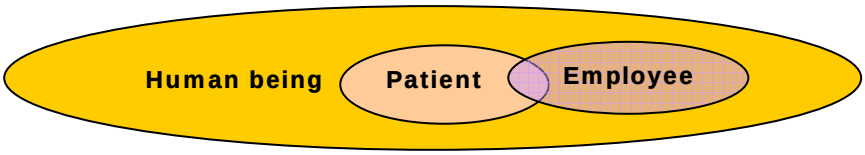

Figure 1 - Different views on the same universe of discourse

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.

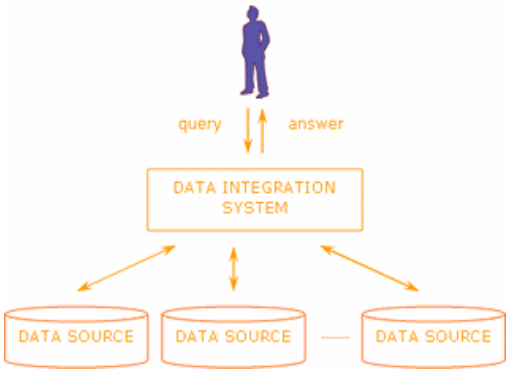

Figure 2 – [23] Common interface to users of integration systems

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.

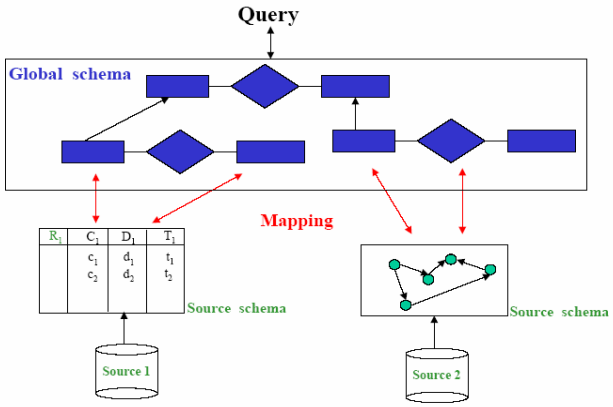

Figure 3 – [20] Schema integration

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.

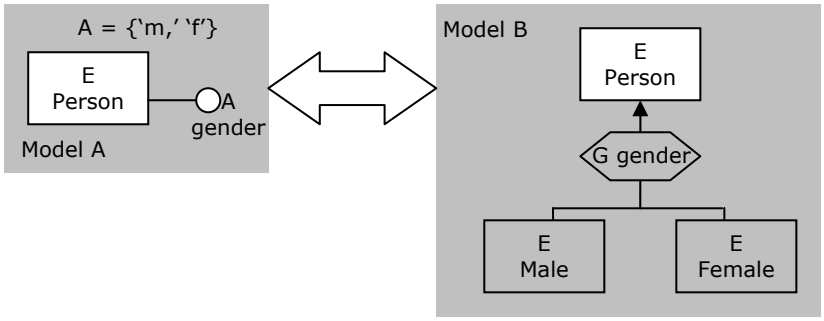

Figure 4 – Structural equivalence

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.

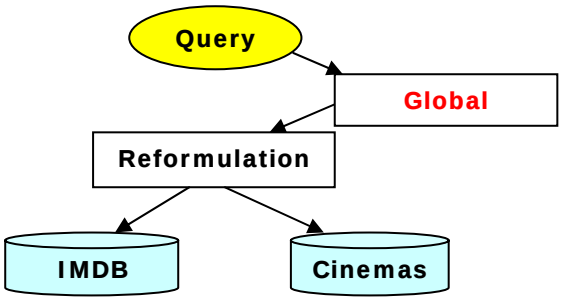

Figure 5 - Virtual integration

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.

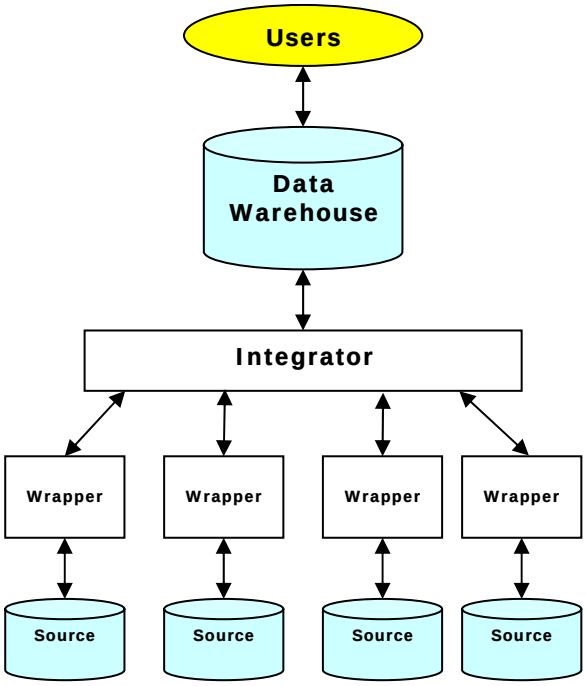

Figure 6 - Data Warehouse architecture

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.

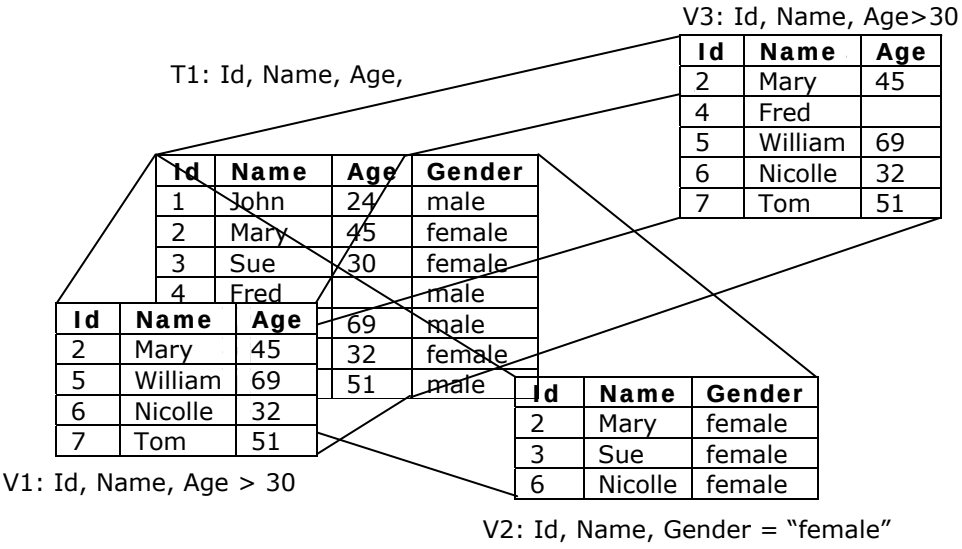

Figure 7– Example of different views

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.

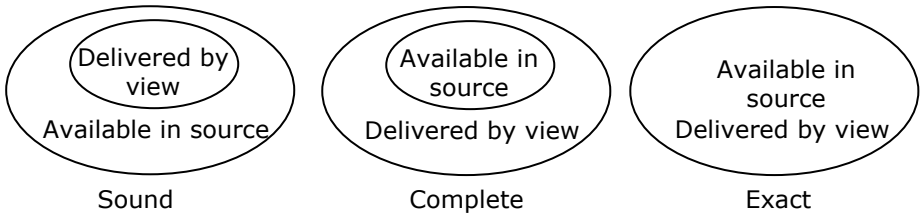

Figure 8 - Sound, complete and exact Views

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".

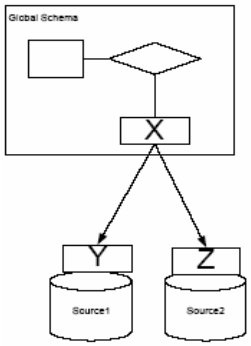

Figure 9 – [23] Global-as-View

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)

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".

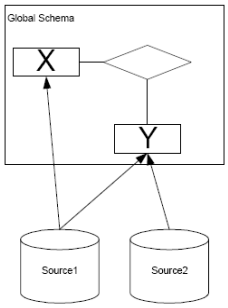

Figure 10 - [23] Local-as-View

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.

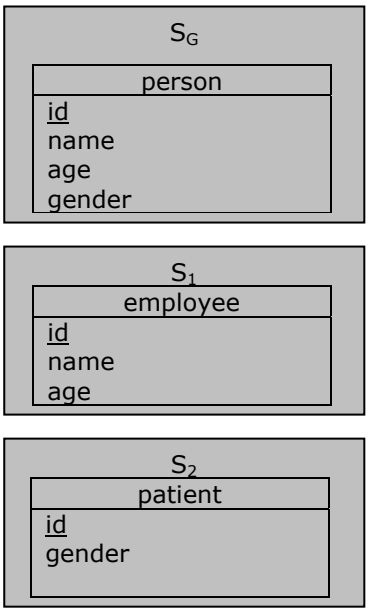

Figure 11 - Local and global schemas

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.

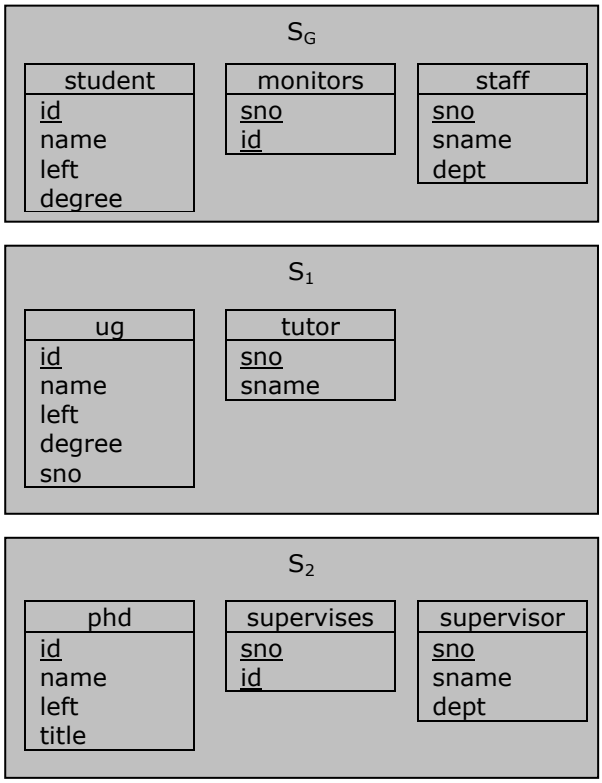

Figure 12 - Global and local schemas

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

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

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

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

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

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

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

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.

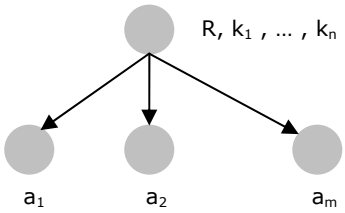

Figure 13 – [1] Construct of hyper graph-based data model

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):

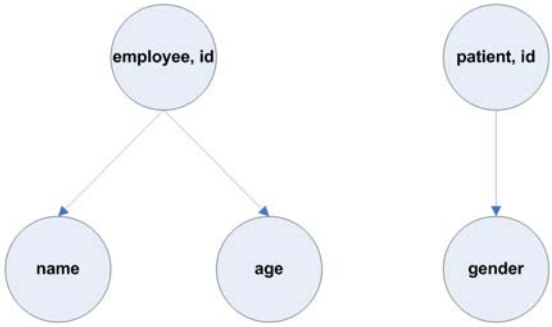

Figure 14 – Local schemas S_1 and S_2

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⟩⟩})

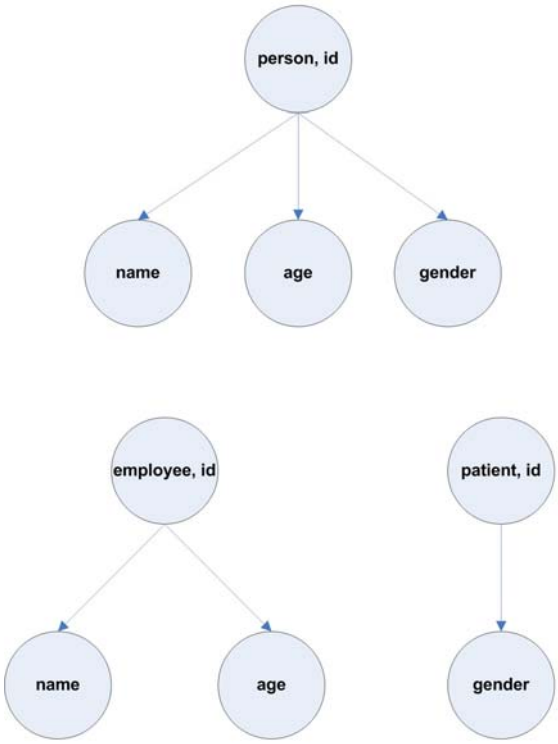

Figure 15 - Integrated schema S_1 and S_2 into S_g

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⟩⟩})

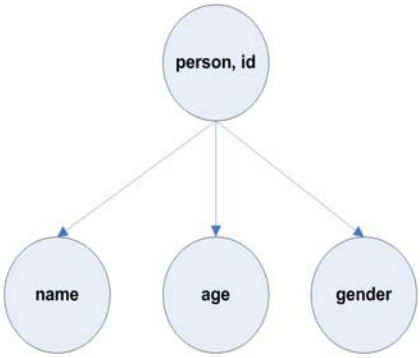

Figure 16 - Global schema S_g after applying all transformation rules

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):

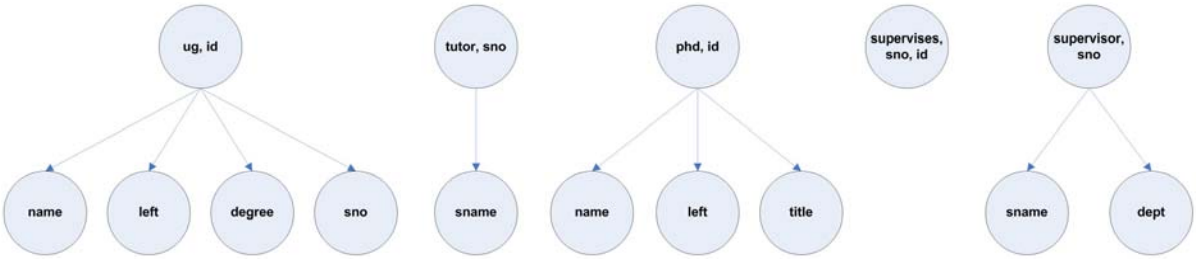

Figure 17 - Local schemas S_1, and S_2 ({ug,tutor},{phd, supervises, supervisor})

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⟩⟩)

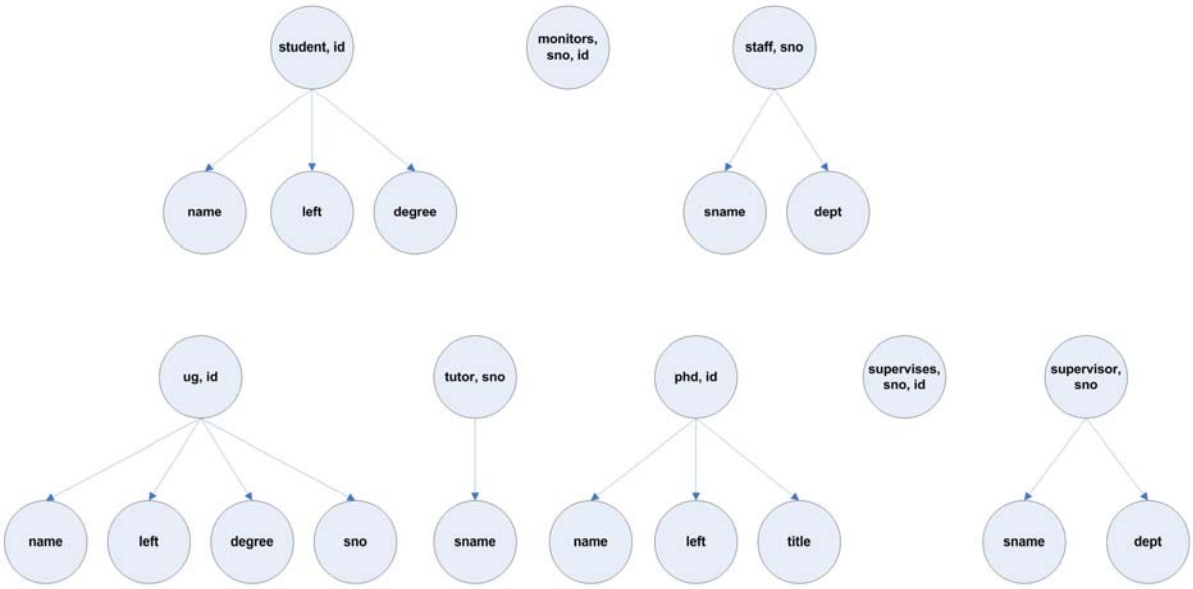

Figure 18 – Integrated schema of S_1, and S_2

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⟩⟩})

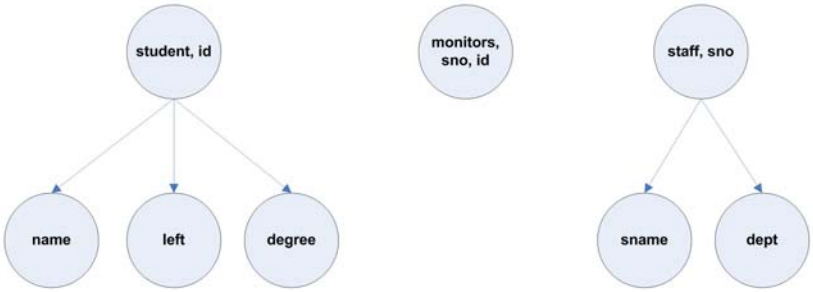

Figure 19 - Global schema S_g after applying all transformation rules

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.

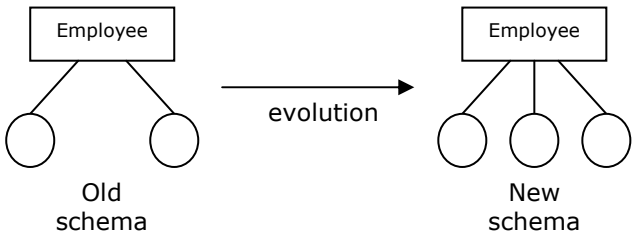

Figure 21 – Schema evolution

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.

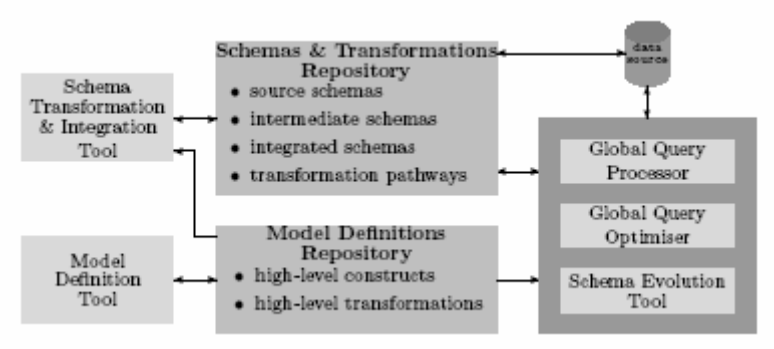

Figure 22 – The AutoMed architecture

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.

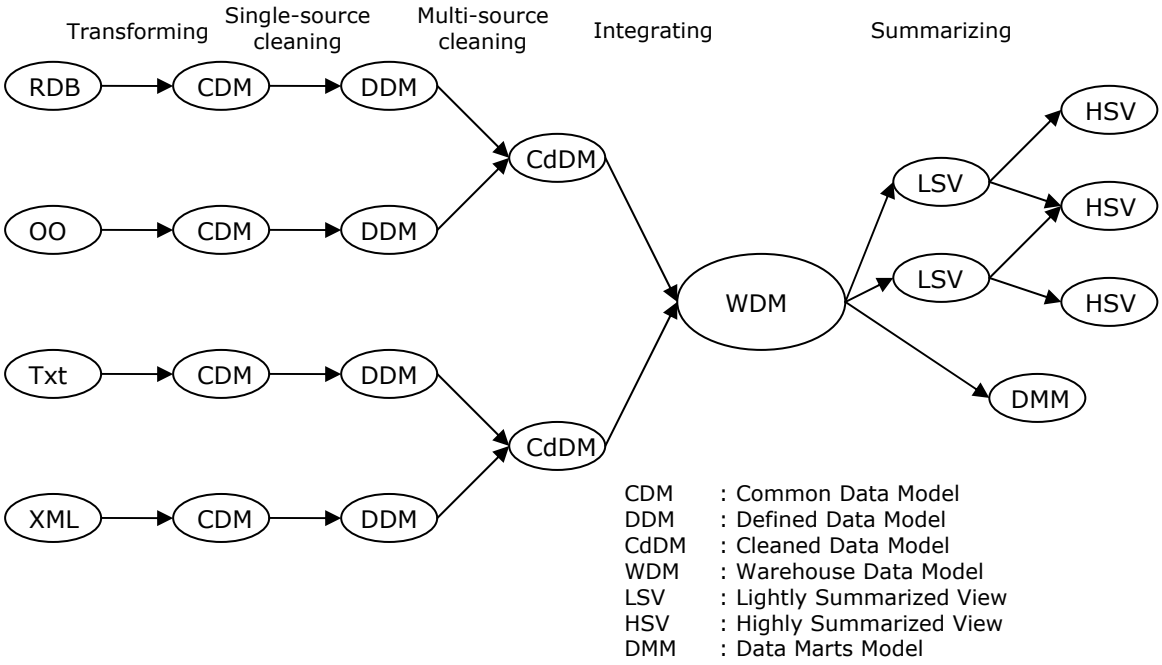

Figure 23 - [24] Logical design of an AutoDW

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. Testing a BAV-system

4.1 Introduction

This section will introduce the fundaments of testing, and how testing can be applied to improve the quality of a data-integration system using the Both-as-View approach. Many research has been done about testing with regard to software-development. Although the development of an integration system differs from development of a software application, there are also many similarities between both. Therefore, available knowledge about testing can be used and projected at the terrain of data-integration.
Up to here, the amount of data that had to be integrated has been rather small. The "employee/patient"-example and the "students"-example were comprehensible, and the pathway from source to global schemas could be instinctively thought out. In practice, integration systems are used in far more complex situations, where hundreds of elements influence the pathway and contribute to the final result. When the situation is anything but comprehensible, a reliable method for developing and testing such a system is vital to guarantee its quality.

4.2 What is "testing" and why should we do it?

"Testing is any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results" [7].

This definition advances the idea of testing as an activity aiming at finding errors. Bill Hetzel, from whom this citation originates, suggests that there are many more ways to evaluate a system without actually executing it. This does not only apply to testing software, but does also apply to testing a data-integration system.

Testing is a technique, being used to check and control quality. The term "quality" is used by everybody, but there is no universal agreement on its meaning. In this document, the term "quality" is used to quantify the degree in which an object conforms to its specification.

Testing can be applied at different levels of detail, different aspects of an objects behaviour can be tested and various methods can be used to perform testing. The most important forms of testing regarding these aspects will be explained in the following section.

An object under test is entitled as "correct" if testing demonstrates that the object is implemented according to its specifications. When the output of a test demonstrates that an object does not conform to its specification, the object is entitled as "incorrect". The issue that causes the non-compliance to the specification is called an error.

4.2.1 Direct testing versus indirect testing

Direct testing or dynamic testing involves performing tests on the object under test. Test cases are formulated and executed at the object under test. By observing the objects behaviour a judgement about the correct functioning of the object can be passed.
Indirect testing or static testing involves any tests that do not concern the actual object under test itself. Indirect testing is done by inspecting source code, documentation and their connection with the object under test.

4.2.2 Black box testing versus white box testing

A common distinction between forms of testing is the one between black box and white box testing. For white box testing, knowledge about the internal composition of an object is being used. It is, for example, based on an objects definition, technical design or the construction of the separate parts of an object.
An example of white box testing regarding software testing is "code reviewing", which is used to discover errors in a program's source code.

As the name states, for black box testing the object under test is treated as a black box. That means that no knowledge is available of what happens inside the object. This form of testing is also referred to as functional testing. The object under test is viewed primarily by it's input and output characteristics. Does a certain input indeed deliver the expected output, within a reasonable time span in a certain environment?
In most cases it is impossible to test all possible input values and compare it to the expected output values, simply because the amount of input values is infinite. Therefore, the tester can

divide test cases in partitions, in such a way that for a certain test case the system is expected to act the same for all test cases in the same partition. This is called "equivalence partitioning". An other technique to minimize the amount of test cases is "boundary value analysis", whereby input values at the boundaries of the input domain are tested.

4.3 When is a test successful?

A test is designed to see if the implementation of a system does meet its requirements. When the test is very limited, and does not cover all requirements the system has to fulfil, it's very likely to get a positive test result (the object under test is correct), while the result should be negative. On the other hand, when a test is designed in a bad way, it is possible to get a negative test result, while the object under test is correct.
More strictly, the following applies to a test:

Implementation='correct'<-> Test-result='correct'
Implementation='incorrect' <-> Test-result='incorrect'

In this document, a test is considered "reliable" if the following applies:

P(other test demonstrates error | current test does not demonstrate error) ≈ 0

In other words, when a test is reliable, it will demonstrate all errors in the object under test, and nothing more.

4.4 Testing a BAV-system

4.4.1 What to test?

There are different elements that influence the quality of a data-integration system. These elements exist at different levels of detail in the data-integration system. By ensuring a correct functioning of these elements, the quality of the system can be controlled. Testing is used to ensure that those different elements do function as is expected, and that any errors present in the system are detected in an early stage.

Schemas and transformations

In data-integration system using the both-as-view method mainly consists of schemas and transformations. Schema's can be divided in source-, global- and intermediate schemas. For a system where the source schema is transformed into the global schema in *n* transformation-steps, that would be the local schema S_0 global schema S_n and intermediate schemas S_1 .. S_{n-1} . The transformations T_1 .. T_n form the pathway between those schemas. Each transformation T_m , $0<m<=n$ results in an intermediate schema S_m . S_m is a model that describes the available information on position *m* in the transformation pathway.

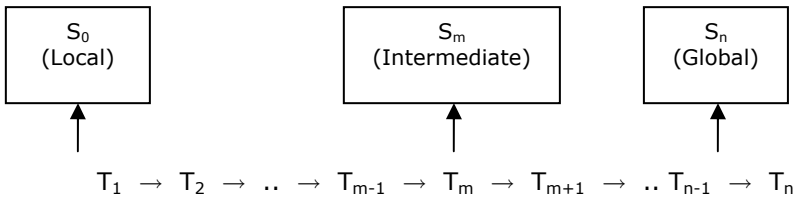

Figure 24 – **Schemas and transformations**

Transformation macros

Primitive transformations can be used as building blocks to create transformation macros, which behave like more advanced transformations. How this is done is described in section 3.3.4. These macro-definitions might be constructed incorrectly and therefore need to be part of the objects that are to be tested.

Architecture
A transformation-pathway that transforms a local schema into a global schema often is part of a chain of transformation-pathways. A single element in this chain is called a stage. Each stage is designed for fulfilling a certain objective, which is modelled in the global schema.

A typical example of a chain of transformation-pathways is data-integration in a data warehouse, where data is homogenised, transformed, cleaned, integrated, summarized and distributed. This results in a data warehouse architecture, designed to fulfil the needs of the owner. Section 3.5 describes this architecture in more detail.

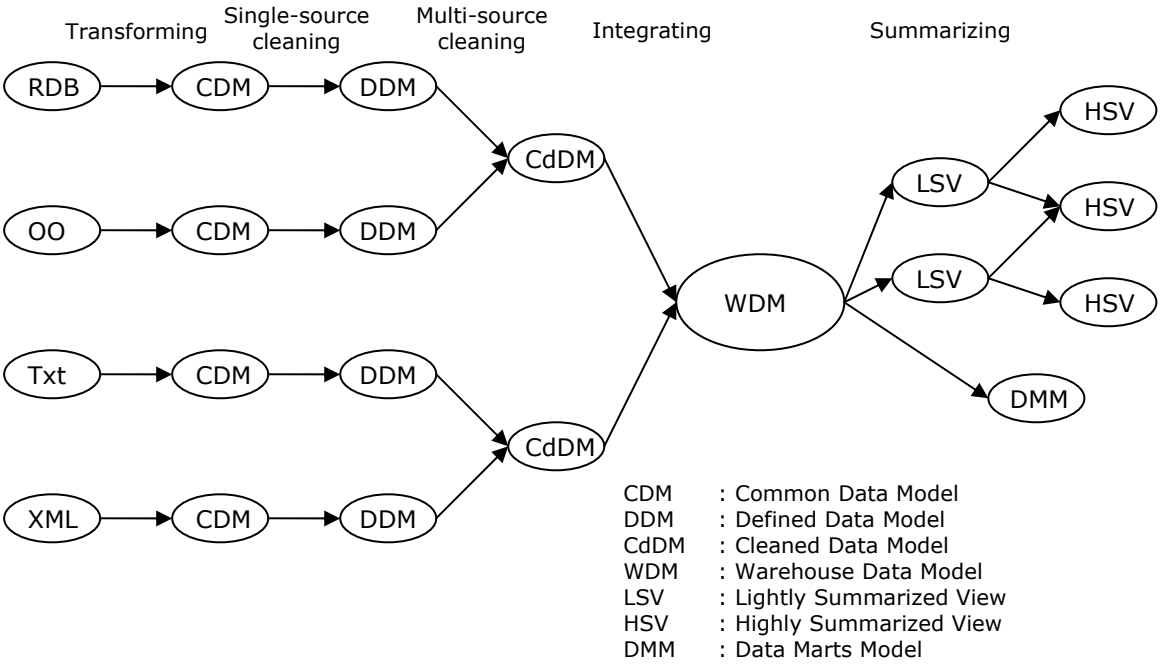

Figure 25 – Data warehouse architecture

The elements described above are some but not all elements that influence the quality of a data-integration system. The scope of this project will be restricted to these elements. Other elements that influence the quality of a data-integration system, beyond the scope of this project, are for example the technical infrastructure on which the system is build or the people that work with the system, or the data residing in the source systems.

4.4.2 V-model as a framework for testing a BAV- system

In this section, a framework for testing a data-integration system based on the Both-as-View method will be described. It should be used as a guideline for the phasing of testing throughout the development of such an integration system. For each phase, it describes which test cases need to be formulated, and when and how they need to be executed. This framework is based on the traditional V-model. The V-model is an internationally recognized development standard for software development. It is used to support a linear execution of a project. It can be adapted to be used for a cyclic approach: the order of the phases will change, the products that have to be delivered will be almost similar.

The V-model is a variation on the traditional waterfall lifecycle model. It is a method employed within software-development projects, and used to highlight the correspondence between development phases and accompanying test-cases. It tells how each step in the development process is connected to a test-case which checks the correct implementation of the current step. In this section, the V-model will be used as a framework for integrating testing into the development of a data-integration system based on the both-as-view method.

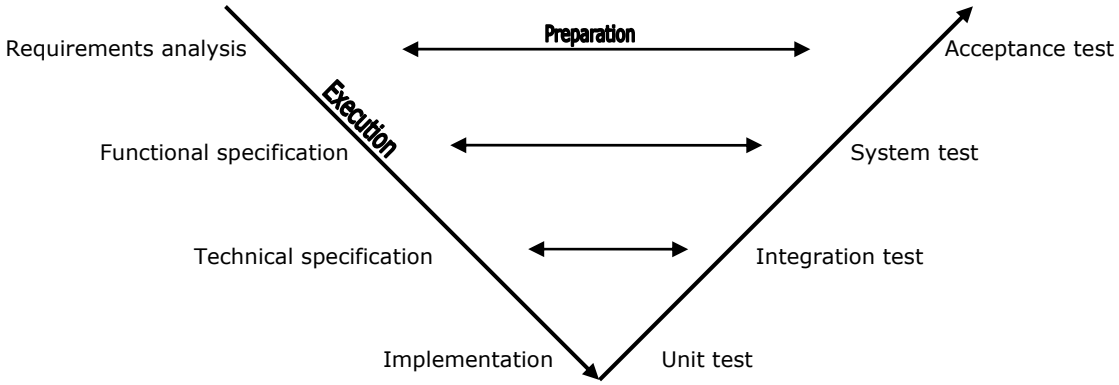

Figure 26 – V-model

There exist different definitions of the V-Model, characterised by a different set of development phases. This is mainly caused by a different level of detail that is being used.
The phases described in Figure 26 are not necessary all phases that exist in a project: it is possible to split each phase in more detailed sub-phases. Figure 26 describes the phases that are distinguished in this testing-framework, and is based on the definition of the V-model used in [9].
The fact that there are differences between definitions of the V-Model does not mean that one or another is incorrect. They are, again, abstract models of the real world, describing the execution of a project. The definition of the V-Model used here fits the case of a BAV-environment best.

The V-model uses a top-down approach for executing a project. First, the outlines of the system are formulated. Each subsequent phase describes smaller parts of the system, going into more detail. This is represented by the left leg of the "V".
For each phase of the project, a test specification is formulated. This specification defines a set of test cases, indicates how to execute them, and indicates how to interpret them. A test case for a BAV-system means a test population for the integration system. Altogether, the test-specification defines a test that is reliable, according to the definition in 4.3, for testing the compliance of the system with the definitions formulated in the current phase. The horizontal lines in the model define the order in which test-specifications are formulated.
Execution of the tests is done in a bottom-up approach, and is represented by the right leg of the "V". It starts with testing on a high level of detail. Each subsequent phase indicates that a test regarding a more global part of the system needs to be executed.

Requirements analysis
In this phase, the objective of the system that is to be developed and the requirements it has to satisfy is being determined. It is a very vital part of the process, since an incorrect identification of the requirements will lead to a system that does not meet the needs of the employer.
The output of the requirements analysis in a BAV-system is a model that describes the information that is required by the end-user. In a data warehouse architecture, as described in 4.4.1, this would be the set of global schemas in the datamarts. These models are accompanied by a document describing what the elements in the global schemas are representing.
The modelling language used to express the global schemas in can be any modelling language that can be specified by HDM.
Note that this is a very strict way of working: already in the first phase of the project, ambiguity is being prevented by using models to describe the required information.

Functional specification
The functional specification describes the functions the system has to fulfil. It describes what the functionality of the system will be. This description is more detailed than the requirements analysis, but does not claim how the system has to be implemented. It is written from the end-user's point of view.
For a BAV-system, the functional specification is a set of regulations the information in the system has to comply with. Those regulations describe relations that can not be modelled in the global schema(s).
An example of those regulations are the business rules of an organisation:

- Total profits = (items sold * selling price) – (items purchased * purchase price)
- Employees that have been ill for more than 6 days/year have to be observed
- Customers that haven't bought a product in the past 4 months are labelled as "inactive"
- Each employee receives his salary exactly once a month

Those rules are typically applicable for a specific organization, and are the result of the way an organization is managed. There are also cases where regulations are enforced by a government agency, or where regulations have to be met in order to posses a certain hallmark. This is of great influence on the integration system. Examples of these situations are:

- "Basel II", a financial standard in the banking industry. It is used to bring stability to the financial system, control risks and create more insight in the business. Being "Basel II"-complaint means that a certain level of quality and reliability is guaranteed. This encourages cooperation between banks over the world.
- "IFRS", International Financial Reporting Standards. Government enforces all companies noted on the stock exchange to comply with these standards. Purpose of this is to procure unambiguous and universal financial reports.

In a nutshell, the functional specification of a BAV-system is a formal document, describing all the rules an regulations the system has to comply with from an end-user's point of view.

Technical specification (design)
The technical specification is a translation of the functional specification into a technical outline of the system. It describes the internal structure of the system, and the building blocks being used to create the system that is specified in the functional specification.
In a BAV-system, the technical specification describes the architecture of the system. The architecture is composed of different stages. The technical specification describes what stages exist, what their objective is and how they relate to each other, hereby forming the system's architecture.
Schemas of the source systems are defined in this phase. The global schemas of the stages are also defined in the technical specification. Those schemas define the model of the information available after executing the stage.
The functional specification is of great influence on the objectives of the stages. Each rule and regulation defined in the functional specification is coupled to a stage, in which it will be executed. This results in different objectives that have to be fulfilled to satisfy the specification of a specific stage. It might be necessary to split some rules/regulations, for technical reasons, as they apply to more than one stage.

In a nutshell: the technical specification describes the different stages that form the system architecture. For each stage, objectives are defined to determine the purpose of the stage.

Program specification (implementation)
The last phase of the development is the actual implementation of the system. In this phase, transformation pathways for the different stages are developed.
Rules and regulation(s) as defined in the technical specifications are converted into transformations. For each (set of) transformations is recorded to what objective of the technical specification it applies. This is done in the documentation belonging to the involved phase. Note that in many cases not a single transformation but a set of transformations is used to fulfil an objective specified in the technical specifications.
Transformation macros can be designed in this phase, to simplify the creation of transformation pathways.

Unit test
In software development, the unit test is used to isolate each part of the system and show that each part is correct. Hereby the correct implementation of and the absence of errors in a single unit (or module) is tested.

In a BAV-system this means that transformations, transformation macros and stages have to be tested. This involves direct testing. Since the internal structure of the items under test is known and is used to create test-cases, all tests performed during the unit test are white box tests.

Transformations are designed to fulfil objectives of the technical specification. The result of a transformation T_m is represented by the data provided by its preceding intermediate schema S_{m-1} and its succeeding intermediate schema S_m . The correct functioning of T_m is tested by creating a small integration system where S_{m-1} is a data source, populated with an appropriate set of data P_{m-1}. S_m is the accompanying global schema.
Next, the expected result of applying T_m on S_{m-1} populated with P_{m-1} is predicted. This predicted outcome population is called $P_{m,\ predicted}$, and is based on the objective that needs to be satisfied by T_m .
The population $P_{m,\ resulting}$ of S_m demonstrates the actual results of applying T_m on S_{m-1} . A comparison between $P_{m,\ predicted}$ and $P_{m,\ resulting}$ will demonstrate the correct implementation of T_m .

The above applies to a single transformation step. A set of transformations can be tested in the same way. Lets say transformations T_m .. T_n , $m<n$ have to be executed to fulfil an objective. Again, a small integration system is created where S_{m-1} is defined as a local schema and S_n is a global schema. Population P_{m-1} is defined to test the correct functioning of T_m .. T_n , based on the objective they fulfil. $P_{n,\ predicted}$ is formulated, and compared to the resulting population $P_{n,\ resulting}$.

Testing a stage is identical to testing a set of transformations, since a stage *is* a set of transformations, starting with a local schema and ending with a global schema. Therefore, the method described above is used to test the correct implementation of a stage. The predicted output population is of course based on the technical specification of the stage under test.

Transformation macros are used to replace the task that is normally fulfilled by a set of transformations. For a transformation macro, there are two things that have to be tested:

- Does the macro behave the same as the transformations it replaces?
- Do these transformations deliver the output that is expected?

The second case is already covered; it corresponds to testing a set of transformations. To test the first case, the test input is composed of a set of local n schemas S_{L1} .. S_{Ln} . Those local schemas are formulated in such a way that they cover all the tasks that have to be performed by the macro M. It can be predicted what the result of executing M on each of the local schema's will be. This leads to $S_{G1,predicted}$ .. $S_{Gn,\ predicted}$. Executing M on S_{L1} .. S_{Ln} will result in global schemas $S_{G1,resulting}$ .. $S_{Gn,\ resulting}$.
A judgement about the correct functioning if M can be accomplished by comparing the predicted schemas to the resulting schemas.

Integration test
Integration testing is the phase of the test process in which individual modules are combined and tested as a group. Those modules have been tested separately during the unit test. For a BAV-system this means testing if the different stages cooperate as expected.
The different stages defined in the system architecture are an indication to group transformations that perform a specific task. In the integration test, succeeding stages are tested. Since a stage consists of a set of transformations, succeeding stages are succeeding sets of transformations. Consider a stage A_1 succeeded by stage A_2 . A_1 consists of m transformations: T_1 .. T_m , A_2 consists of n transformations: T_{m+1} .. T_{m+n} .
By concatenating A_1 and A_2 , the global schema of A_1 , S_m becomes the local schema of A_2 . Stages A_1 and A_2 are now described by T_1 .. T_m , T_{m+1} .. T_{m+n} , which is identical to T_1 .. T_{m+n} . This way, the problem of testing stages is reduced to the problem of testing sets of transformations. This way, succeeding stages can be tested by creating a test-population based on the criteria of the succeeding stages. These criteria are defined in the technical specification. While formulating the test population, the stages are treated as black-boxes. This means the test population is not based on the inner logic of the stage, but only on its interface (local/global schemas) and its specification.

System test
In this phase, a complete, integrated system is tested to evaluate its compliance with the specified requirements ([13]). For a BAV-system this means that all stages together are being tested and evaluated against the functional specifications. This means that test populations for the source schema('s) are defined, the expected populations of the global schema('s) are predicted based on the specifications and compared to the actual population of the global schema('s). The test populations are created based on black-box techniques.
The system test is also the phase where other aspects of the system, beyond the scope of this project, are tested. This involves, among other things, performance testing, security testing, capacity testing and recovery testing.

Acceptance test
The acceptance test is intended to determine whether or not a system satisfies its acceptance criteria ([13]). In this phase, the end user decides if the system is built according to his (her) requirements. The environment the test is run in, is usually identical or comparable to the environment the system will be installed in. Test cases for a BAV-system are populations for the source systems, that test if all user requirements are met. Again, predicted outputs are compared to actual outputs.

4.5 Disadvantages of the V-Model
The V-Model is designed as a linear development approach. Linear approaches work best in an environment where the requirements are stable and clear, where the user is rather knowledgeable and knows what he wants, and there is no time-to-market pressure. Also, the V-Model does not provide methodology for correcting errors made in an early phase, so when a phase is not run-trough very accurately, this might cause a collapse of the project in a subsequent phase.

For situations different than described above, other models might offer a better support for managing a project. This can be, for example, an iterative method, where the phases of designing, coding and testing are run trough multiple times. The similarity between these different methods are the phases that are specified. Therefore, the definition of the phases described in 4.4.2 can be a major guideline for formulating an other model for developing BAV-systems. Therefore, the main difference between these models will be the connection between the different phases, and the order in which they are executed, not the definition of the phases.

4.6 Conclusion
A BAV-system can be considered on different levels of detail, each level outlining a model that describes a part of "the real world". By determining what the system looks like on each level of detail, and by defining how it can be tested that each of those levels is implemented in a correct way, it is possible to create an integration system that is very likely to deliver correct information. These aspects could be investigated in future research.
An important aspect of testing that is not discussed in this chapter, is the definition of the test-cases. There are many theories, regarding software development, about formulating test-cases that cover all aspects of the system that has to be tested. It is very likely that those theories can contribute to creating test-cases for a BAV-system. This could also be investigated in future research.

Different phases for developing and testing a BAV-system can be defined, corresponding to the different levels of detail that can de distinguished in such a system. The V-Model can be used as a guideline for executing these phases, but it should not be seen as "the truth". In different situation, different models might offer better possibilities. The formulation of the phases, though, should not differ very much within different development-methods.

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 on testing methods for integration systems described in section 4 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 [25], 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 and the approach to lineage described in [25]. 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 section 4.4.2.

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 section 4 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 27 below.

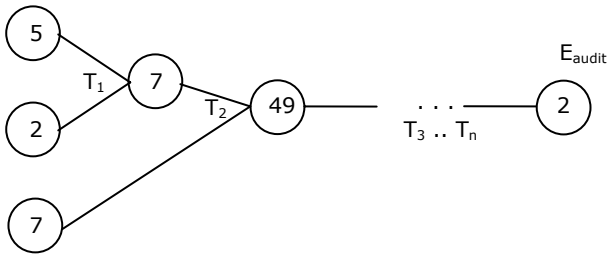

Figure 27 - Example audit trail

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 28 below.

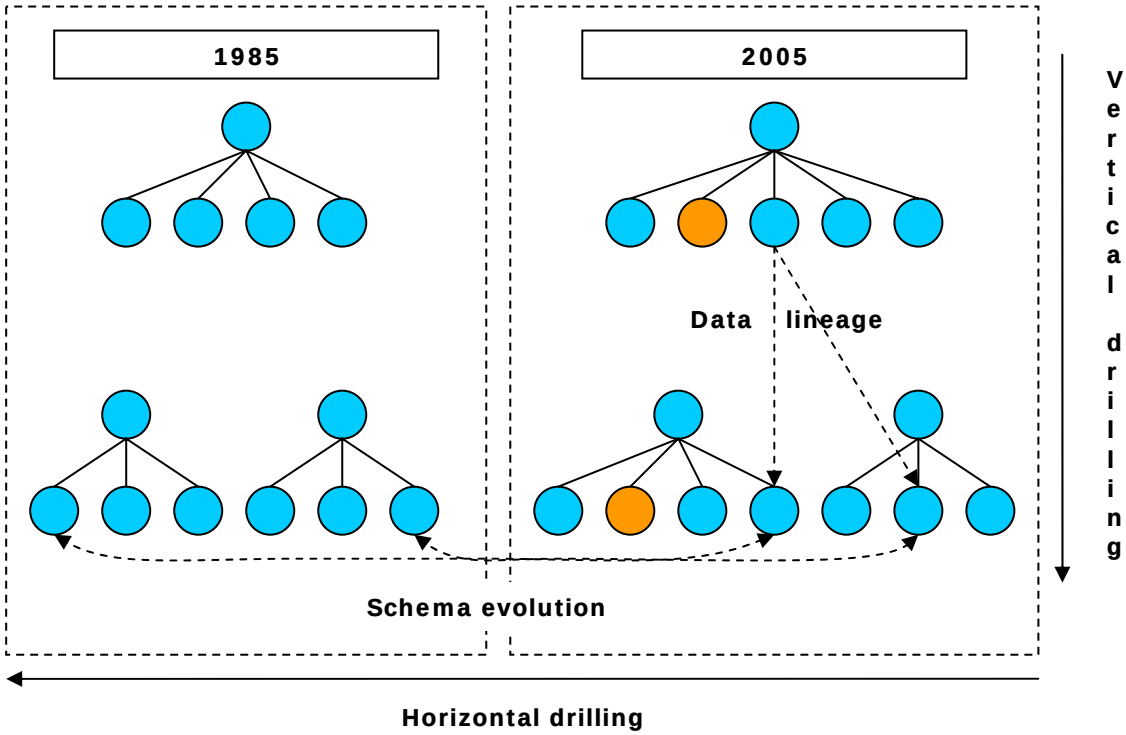

Figure 28 – Horizontal and vertical drilling

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 section 4, 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 section 4 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 section 4 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]. B. Hetzel, Bill, *A Complete Guide to Software Testing*, QED Information Sciences, 1988

[8]. T. B. Hilburn, *Software Quality: A Curriculum Postscript?,* SIGCSE Bulletin, 2000

[9]. T. Koomen, R. Baarda, *TMAP Test Topics*, Tutein Nolthenius, 2004

[10]. M. Pol, R. Teunissen, E. van Veenendaal, *Testen volgens TMAP*, Tutein Nolthenius, 1999

[11]. J. Tretmans, *Testing with formal methods*, University of Nijmegen, 2002

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

[13]. *IEEE Standrd Computer Dictionary*, IEEE, 1990

[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

[25]. M. van der Wielen, *Improving the quality of data integration systems using the BAV-approach, emphasising on data lineage,* University of Nijmegen, 2005