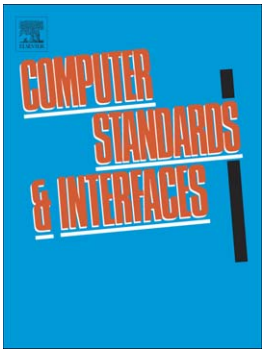

Tracing Conceptual Models Evolution in Data Warehouses by using the Model Driven Architecture

Alejandro Maté, Juan Trujillo

PII: S0920-5489(14)00007-5
DOI: doi: 10.1016/j.csi.2014.01.004
Reference: CSI 2958

To appear in: *Computer Standards & Interfaces*

Received date: 8 May 2013
Revised date: 7 December 2013
Accepted date: 2 January 2014

Please cite this article as: Alejandro Maté, Juan Trujillo, Tracing Conceptual Models Evolution in Data Warehouses by using the Model Driven Architecture, *Computer Standards & Interfaces* (2014), doi: 10.1016/j.csi.2014.01.004

Tracing Conceptual Models Evolution in Data Warehouses by using the Model Driven Architecture

Alejandro Maté[a,*], Juan Trujillo[a]

[a] *Lucentia Research Group, Department of Software and Computing Systems, University of Alicante, Carretera San Vicente del Raspeig s/n - 03690 San Vicente del Raspeig - Alicante, Spain*

Abstract

Developing a data warehouse is an ongoing task where new requirements are constantly being added. A widely accepted approach for developing data warehouses is the hybrid approach, where requirements and data sources must be accommodated to a reconciliated data warehouse model. During this process, relationships between conceptual elements specified by user requirements and those supplied by the data sources are lost, since no traceability mechanisms are included. As a result, the designer wastes additional time and effort to update the data warehouse whenever user requirements or data sources change. In this paper, we propose an approach to preserve traceability at conceptual level for data warehouses. Our approach includes a set of traces and their formalization, in order to relate the multidimensional elements specified by user requirements with the concepts extracted from data sources. Therefore, we can easily identify how changes should be incorporated into the data warehouse, and derive it according to the new configuration. In order to minimize the effort required, we define a set of general Query/View/Transformation rules to automate the derivation of traces along with data warehouse elements. Finally, we describe a CASE tool that supports our approach and provide a detailed case study to show the applicability of the proposal.

Keywords: Data warehouses, traceability, conceptual models, business intelligence, MDD, MDA, QVT

1. Introduction

Developing a data warehouse (DW) is an ongoing task where new requirements are constantly being added. Either as a result of the dynamic environment, or due to new sources of information becoming available (i.e. social media), decision makers constantly pose new requirements and questions which

*Corresponding author. Tel: +34 96 5909581 ext. 2737; fax: +34 96 5909326

Email addresses: amate@dlsi.ua.es (Alejandro Maté), jtrujillo@dlsi.ua.es (Juan Trujillo)

need to be answered by analyzing information. This information is integrated from several heterogeneous sources. Then, it is structured in terms of facts and dimensions in the DW [1]. Therefore, the development of the DW is a continuous and complex process that must be carefully planned in order to meet user needs and incorporate new requirements. To this aim, three different development approaches have been proposed: bottom-up or supply-driven, top-down or demand-driven, and hybrid [2, 3].

The first two approaches ignore one source of information until the end of the process, either requirements or data sources. This lack of information leads to failure in some DW projects [2, 4] since they either (i) ignore user needs or (ii) assume that all the necessary data is available, which is not always the case. On the other hand, the hybrid approach makes use of both data sources and user requirements [3], solving incompatibilities by accommodating both requirements and data sources in a single conceptual model before implementing the DW. Nevertheless, the current accommodation process is performed much like a schema redesign process: successive modifications are made to the schema, removing, renaming, and adding new elements according to the designer's experience. In turn, the resulting DW schema may neither match the data sources in structure nor in naming conventions. As a result, existing traceability by name matching is lost. Therefore, these correspondences must be identified again when (i) validating and reviewing old requirements, (ii) posing new requirements, or (iii) modifying data sources, all of which are error prone tasks. Consequently, time and resources required are increased while the quality of the final product is decreased [5].

In our previous works [6, 3, 7, 8], we defined a hybrid DW development approach in the context of the Model Driven Architecture (MDA) framework [9]. The idiosyncrasy of DW development favors our approach. In DW development, data sources act as both a source of additional information as well as a limiting factor. In order to implement a requirement in the final DW, the required information must be present in the data sources, either directly or by deriving it. Therefore, we can clearly identify the desired structure of the DW (requirements), and what information is supplied (data sources). The final step in this process is to adequately relate this information in order to easily trace and incorporate changes, instead of arbitrarily mixing the schema, and making it difficult to perform further analysis tasks. Additionally, unlike in software development, the reconciling process may find elements not present in the requirements model (due to an oversight) that provide relevant information for decision makers [8]. Thus, it is interesting to trace elements present in the data sources that do not have a requirement counterpart, but are present in the implementation of the DW, since they help to elicitate overlooked user requirements.

The automatic derivation is done by means of model to model transformations specified by Query/View/Transformation (QVT) [10] rules. QVT is a language defined by the Object Management Group (OMG) and proposed as a standard to create model to model transformations. This language can be used to create both DW models as well as trace models. However, due to our experience, the reconciliation task can only be done at most semi-automatically,

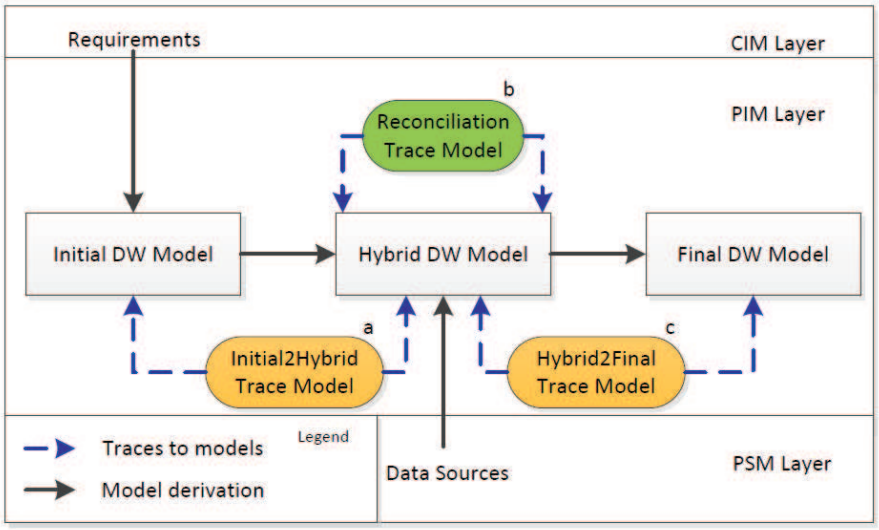

Figure 1: Overview of the approach

since there is not enough information available to fully automate it. The set of trace models employed in our approach are shown in Figure 1, and are further detailed in Section 3.

In the short version of this paper [11] we developed a set of traces for preserving the traceability of requirements at conceptual level. Now, in this extended version, we (i) provide a deeper review of the related work describing details of the existing traceability approaches, (ii) provide a formal definition of our traces, (iii) generalize a set of QVT transformations which allow us to derive the data warehouse from any trace configuration specified, and (iv) provide an extended case study which tests and shows better the application of the proposal.

The remainder of this paper is structured as follows. Section 2 presents related work about traceability and DWs. Section 3 introduces the necessary trace semantics in order to include traceability at the conceptual level in DWs. Section 4 presents a set of QVT rules for automatic derivation of traces. Section 5 presents a case study to show the applicability of our proposal. Finally, Section 6 outlines the conclusions and further work to be done.

2. Related Work

In this section, we will discuss existing traceability research, its benefits and problems, and its current status in the DW field. Traditionally, traceability has been focused on requirements. Either coming from the traditional Requirements Engineering (RE) [12, 13, 14, 15] or following a Model Driven Development

(MDD) approach [16, 17, 18], requirements are traced to their lower abstraction level counterparts. Therefore, traceability helps assessing the impact of changes in requirements and in rationale comprehension, by identifying which parts of the implementation belong to each requirement [19]. Additionally, it also improves reusability and maintainability [13]. However, the lack of standardization makes it difficult to apply traceability to projects, since even the basic concepts differ from author to author [16, 18]. Therefore, there is a special interest on automating traces and providing a framework with a set of basic concepts that can be extended.

In order to provide some degree of automation, recent works record the relationships between elements by following two different approaches. First, generating traces from already existing information. An advanced example is presented in [20], where the authors combine topic modeling with prospective traceability as the user interacts with the system. Second, making use of the logic behind automatic transformations. In this second approach, the transformation logic generates a set of traces in addition to the new version of a model. Traces record the relationships between elements in the source and target models, and can be analyzed in by means of algorithms that take into account their semantics. The former approach can be applied whenever a user interacts with an artifact, minimizing the necessity of manually adding traces. The latter can only be applied when models are automatically transformed. However, whereas the first approach may generate some incorrect traces, the second solution is is based on transformation logic, thus being less error prone.

Nevertheless, tracing the counterparts of a requirement at conceptual level is not always straight-forward, even when following a MDD approach and exploiting transformation logic. Elements are refined by the developer before being transformed into the next model, altering their characteristics or even their structure. This process is repeated until the final version is obtained. Therefore, in order to maintain traceability between models, the result of these operations must be traceable.

In DW development, the different steps can be clearly identified as the DW schema evolves through several conceptual models. However, the differences in the language used by decision makers, and the language used by IT, makes it difficult to reconciliate data sources and the target DW schema. This communication problem is analyzed in [21], where the authors propose to tackle this problem by means of ontologies to improve the communication between parties. A combination of ontologies and traceability could help to produce a seamlessly integration new data sources and changes into the DW. Thus, in order to validate requirements and support incremental changes, we require to trace the representation of an element from one model to its counterpart in the next model.

In order to tackle this problem, different works from the RE [12, 13] and the MDD communities [17, 18] have included traceability in software development processes. However, aside from our previous contribution in [22], where we defined a trace metamodel for tracing requirements to multidimensional structures, the traceability aspect has been overlooked in DW development. Some

works mention the existence of mappings between the models involved in DW development [2, 23], but they are not formally introduced nor include any specific semantics. Others rely on name matching to preserve implicit traceability, which cannot be applied if naming conventions and structures differ [3]. Finally, some works related to DW lineage exist, such as [24]. Although similar, these works are mainly aimed to trace instances of data by means of queries. This characteristic provides a powerful tool to inspect the source of data being analyzed. Nevertheless, they do not provide support for modeling and provide an overall view of the DW reconciliation process.

Our approach, presented in [3], applies MDD and is able to generate traces by exploiting transformation logic. While our approach applies a specific framework for DW development (MDA [9]), other development approaches [2, 23] make use of very similar layers, combined with his own conceptual representation [25, 26, 6] for modeling DWs. By generating traces between conceptual models, we are able to trace the different versions of an element, providing support for requirements validation, impact change and automated analysis, while minimizing the existing drawbacks in traceability.

Furthermore, by using MDA, we cut development time, as shown in [27, 28], where the authors define an approach to derive a Temporal DW and allow users to generate analysis queries using a visual language. Since transformations from the top layer to the final implementation are performed in a semi-automatic way, the time and effort required for the development process are reduced.

In our approach, requirements are specified in a Computation Independent Model (CIM). Then, they are automatically derived into a conceptual model [6] at the Platform Independent Model (PIM) layer. This initial PIM model records the conceptual elements specified by requirements. In order to keep this model clean for further analysis, a hybrid PIM is derived by including information coming from the data sources, obtained by means of reverse engineering [29]. Afterwards, desired elements are marked and derived into the final PIM, which conceptually represents the implementation of the DW. This way, the hybrid model can be marked with multiple configurations, allowing us to derive alternative implementations of the DW.

Summarizing, in order to maintain accurate information about the implications of each requirement and data source in the DW schema, all elements must be traced along the successive refinements performed at the PIM layer. Therefore, traces must maintain the semantics of their relationships, allowing us to support automated analysis.

3. Traceability through Conceptual Models for Data Warehouses

As aforementioned, we need to be able to trace information from both requirements and data sources up to the final conceptual model in order to support incremental changes and analysis tasks. In this section, we will first introduce our trace metamodel that provides the basic concepts used for tracing elements along the PIM models. Then, we will motivate and introduce the relationships

in the hybrid PIM model, which are specially relevant to connect elements specified by user requirements with reverse engineered elements obtained from data sources.

3.1. The Trace Metamodel

In order to trace conceptual elements up to the final PIM we require to include different semantics. These semantics will allow us to differentiate the relationships between elements, and to support further automatic operations. To this aim, we extended the metamodel proposed in the Atlas Model Weaver (AMW) [30]. The resulting trace metamodel, presented in Figure 2, includes six different semantic types to cover the steps involved in a DW development process:

First, *Satisfiability* links capture the relationships between elements in the user requirements model and conceptual elements [22]. These links allow us to trace each requirement to its multidimensional counterparts, identifying what elements support multiple requirements, and what elements are affected when user requirements evolve.

Second, *Derived_from* links capture the relationships between the data sources schema and the reverse engineered conceptual elements. These links are modeled following a similar approach as in [22] but focusing on the reverse engineering logic instead. As in the case of *Satisfiability* links, these links are out of the scope of this paper.

Third, the set of *Evolution*, *Overlap*, *Conflict* and *Rationalization*, are employed to trace elements as they evolve through the PIM layer. These semantic links are the focus of this paper and are formalized in Section 3.3:

- **Evolution** links are included to handle traceability of element changes within the same layer. These links track the different versions of an element at each PIM model. Evolution links serve as a way to navigate between models, allowing us to identify the refinements to DW elements. For example, a hierarchy level specified by requirements may have its name changed or be enriched with attributes coming from existing reports. Each element traced by an *Evolution* link is considered to represent the very same concept in different stages of the development process.

- **Overlap** and **Conflict** links relate elements obtained from requirements with those reverse engineered from data sources. We can differentiate between three possible situations during this process. First, the necessary information for an element obtained from requirements may not be available. In this case the element will not be related to any other element. Second, the necessary information may be available, requiring only a cleaning process in order to load it into the DW. In this case, elements will be related by means of *Overlap* links. Third, the necessary information may be available, but it may be necessary to transform its structure. In this case, elements will be related by means of *Conflict* links. These links are crucial for enabling traceability support, as they allow us to

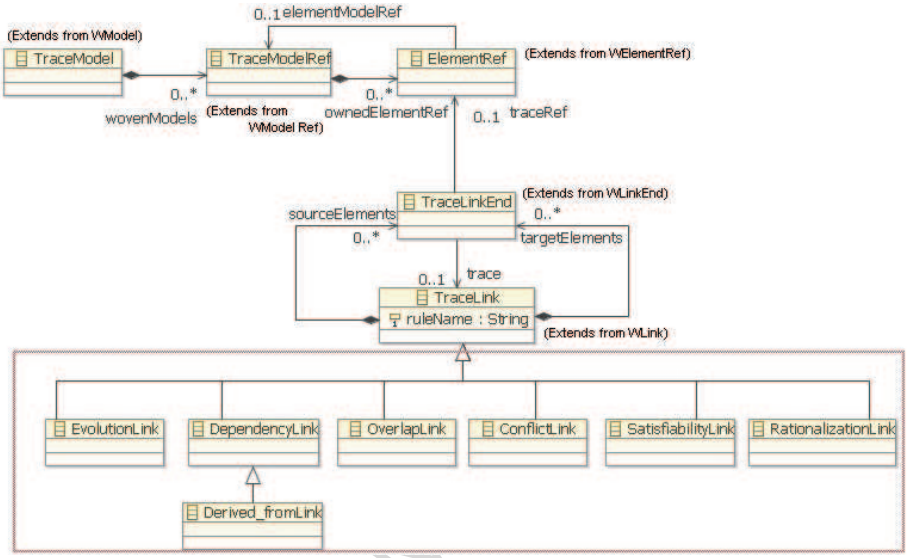

Figure 2: AMW Metamodel for traceability extended with semantic links for DWs

merge elements coming from data sources and requirements, and record the semantic of their relationship.

- **Rationalization** links are included as means of enabling the user to record his decisions, as well as to provide reconciliated solutions for existing conflicts.

3.2. Trace Models in Data Warehouses

The previously defined trace types are recorded in different trace models included in our proposal, as shown in Figure 1.

The first trace model, "a", shown in Figure 1, connects the initial PIM to the hybrid PIM in a straight-forward way by means of *Evolution* traces. The initial PIM is maintained as a clean source for transformations that combine elements from data sources and elements specified by user requirements. This model is included in order to support automatic operations which require to trace information related to requirements.

After we have derived the initial PIM, we proceed to create a hybrid PIM. First, we obtain a Platform Specific Model (PSM) from the data sources by means of reverse engineering. The resulting model is compliant with the Common Warehouse Metamodel (CWM) specification [31]. This allows us to provide a unified set of constructs for model transformations that generate multidimensional elements [29]. These transformations allow us to further abstract the CWM model into the conceptual level, thus hiding the complexities of the PSM

Layer that are not relevant for the designer at this point, such as how information is organized across tables.

As a result we obtain a hybrid PIM, characterized by capturing different versions of the same concepts simultaneously. On the one hand, elements coming from the initial PIM present the structure of the DW according to users' expectations. On the other hand, elements transformed from the PSM model present the data "as-is" in the data sources. Afterwards, the developer reconciles both versions by means of traces recorded in trace model "b" (see Section 3.3). Then, the developer marks what elements he wishes to include in the final PIM.

The derivation process generates a set of evolution traces that are recorded in trace model "c". These evolution traces record what elements were chosen by the developer to be part of the final PIM. This allows us to trace back to requirements and data sources the elements included in the DW. The exact set of traces generated will vary depending on the relationships specified in trace model "b". Therefore, some of the traces in trace model "c" may include multiple sources, and link elements from different entities.

Our set of traces allows us to easily identify (i) elements in the DW affected by a change in the data sources, (ii) elements related to each user requirement, or (iii) elements that did not obtain their information from data sources, for example because it is expected to retrieve it from external sources.

After having defined the trace models that record the evolution of conceptual elements, we will describe the reconciliation traces.

3.3. Reconciliation Traces at the Conceptual Level

Reconciliation traces represent special kinds of relationships that do not fit into the semantics of the metamodel being instantiated. Instead, these traces capture the relationships between elements from requirements and those from data sources at conceptual level. In our case, these relationships are defined by the *Overlap*, *Conflict*, and, if necessary, by *Rationalization* links.

As previously described, elements included in the hybrid PIM can be obtained from three different sources: (i) user requirements, (ii) data sources, or (iii) created by the developer to solve existing conflicts or create derived elements. The developer must manually identify what elements coming from user requirements match with each element coming from the data sources. Once identified, he records the semantic of the relationship with the corresponding traces. After the developer has captured all the necessary relationships, he can mark what elements he wishes to derive into the final PIM. In this way, any change performed afterwards will be easily traceable up to the final PIM, allowing us to analyze which requirements, or data sources, are also impacted by the change.

Reconciliation traces must be manually added since, typically, there is no knowledge about what element derived from the data sources is the counterpart to an element specified by user requirements. User requirements are described in business terms, while data sources follow IT naming conventions. Moreover,

data sources typically focus on transactional processes while user requirements are oriented to describe the decision making process. Therefore, elements may differ in name, attributes and even in the dimension hierarchy. This problem may be partially solved depending on the amount of information provided by user requirements and data sources. An example of this approach is applied in Software Engineering [20], although it is out of the scope of this paper.

Thus, in order to adequately relate user requirements with data sources, we proceed to apply the specified trace links as follows:

- **Overlap** links are employed whenever an element from user requirements, and its counterparts in the data sources, obtain their values from the same domain and their structure is equivalent. Therefore, the final concept is conceived as the fusion between the elements provided by the data sources with those specified by user requirements, renaming them accordingly to the user needs. Formally defined, given two elements $e_1 \in D_1, e_2 \in D_2 \rightarrow D_1 \cap D_2 \neq \emptyset$, where D_1 is the source domain and D_2 is the target domain.

- **Conflict** links are employed whenever an element coming from user requirements and its counterparts in data sources refer to the same concept, but their domain is different. Therefore, concepts can not be interchanged without obtaining a different schema, including different ways to aggregate the data. For example, a "Customer" level coming from user requirements includes the attributes "name" and "surname". However, data sources only present a single attribute for representing this information "full name". These attributes are conflicting, since their domains are different unless a transformation is applied. Moreover, if these attributes constitute the *Descriptor* (identifier) of the level, the resulting schema would present different aggregations depending if we used "name" or "full name". Formally defined, given two (or more) elements $e_1 \in D_1, e_2 \in D_2 \rightarrow D_1 \cap D_2 = \emptyset$, where D_1 is the source domain and D_2 is the target domain.

  In order to address this situation there are two possible solutions. First, we can choose one representation as the correct one. Then, we derive the corresponding elements. In the previous example "name", "surname", and "full name" would be related by means of a *Conflict* link. Afterwards, we would mark the desired elements to be derived into the final PIM. Second, we can provide one or more reconciliating elements by means of *Rationalization* links.

- **Rationalization** links are employed whenever the developer requires to create a new element that reconciliates an existing conflict. Using the previous example, the developer may create a new level, "Standardized-Customer", including "surname" and a new attribute "addressing name". This new attribute could include the "full name" information, as well as the way to address a customer. Additionally, *Rationalization* links may also be applied to create derived conceptual elements.

After having explained the different semantic links included in the reconciliation process, we will present the necessary QVT rules for the automatic derivation and record of traces.

4. Automatic Derivation of Traceability Models in Data Warehouses

In this section, we will discuss the necessary transformations to automatically generate the traces between conceptual elements and store them in trace models, that can be updated over time. Then, we will present our CASE tool that supports our approach, the Lucentia BI Suite.

According to our proposal for developing DWs [3, 7], we use a hybrid approach based on MDA. First, elements are derived in an initial PIM model from requirements. Then, we successively refine this initial PIM model and reconciliate it with data sources obtaining the hybrid PIM. Finally, we mark the desired elements and derive the final PIM. These transformation are done by means of QVT rules. QVT rules specify a transformation by checking the existence of a defined pattern in the source model. Once the pattern is found, a QVT rule transforms elements from the source metamodel into the target metamodel.

The process to derive the final PIM is based on two general QVT rules. Both rules make use of two different metamodels: the conceptual DW metamodel [6], and the trace metamodel, presented in this paper. These two metamodels are used in four different models involved in these rules: (i) the source DW model, in the top-left corner, (ii) the target DW model, in the top-right corner, (iii) the reconciliation trace model, in the lower-left corner, and (iv) the trace model for evolution traces, in the lower-right corner.

The first rule, *OverlapRule*, defines how overlapping elements are derived. This rule creates an *Evolution* link from the hybrid to the final PIM. At the top-left corner of Figure 3, we can find the conceptual elements from the hybrid PIM. This rule establishes that all the elements related by an overlap relationship, "E1" and "E2", act as sources of the transformation. At the top-right corner, we can find the resulting target element, "R1". The values assigned to the properties of this element, i.e. its name, correspond to the marked element. At the lower-left corner, we can find the trace link relating "E1" and "E2". In this case, the trace link is an Overlap type link that has two trace link ends, corresponding to the elements linked. Finally, at the lower-right corner of the Figure, we can find the new generated trace link, which is stored in trace model "c". This trace is an Evolution type link, containing "E1" and "E2" as sources, and "R1" as the target. In this way, the Evolution link allows us to automatically identify if the target element will be affected by a change in requirements or in the data sources.

The "C" at the center of the Figure means that the source models are checked to evaluate if the specified pattern exists. On the other hand, the "E" specifies that the target models are enforced. Thus, each time that the described pattern is found in the source models, the target patterns are generated in the resulting target models. The "When" clause establishes that this rule will apply only when an element is marked by the designer. Finally, the "Where" clause

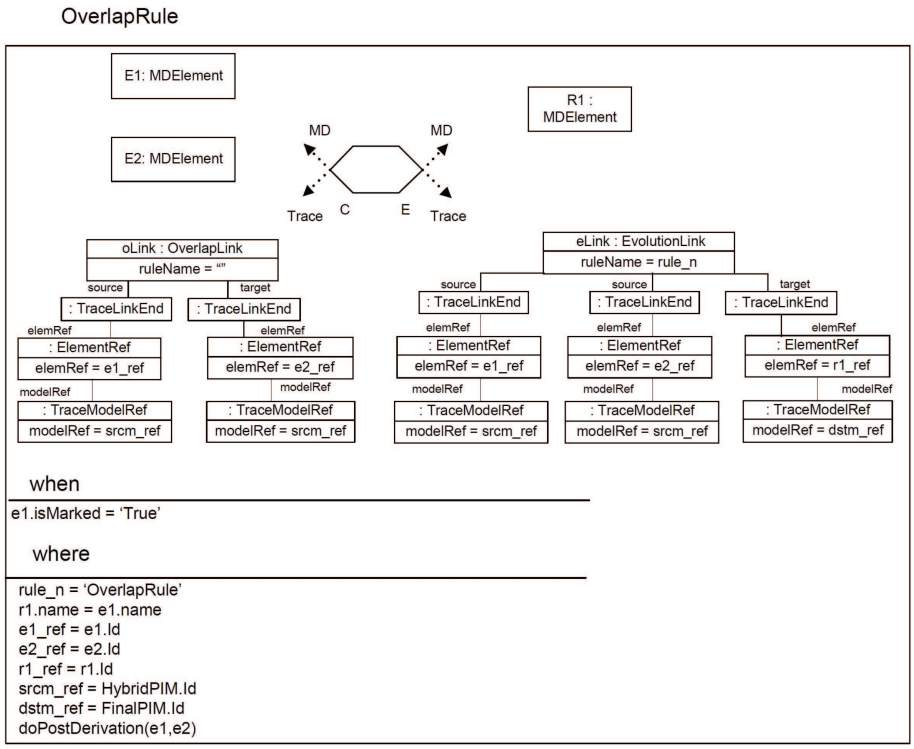

Figure 3: Generic QVT rule for deriving overlapping required elements and sources and creating their *Evolution* trace link

establishes other operations performed after the generation of "R1". The most relevant operation in this clause is the function call "doPostDerivation". This call performs actions such as analyzing the different attributes to be included, in the case of levels, and establish the roll-up hierarchies, in case of dimensions.

The second rule, *ConflictRule*, defines how conflicting elements are derived. This generic rule has a significant difference with the previous one: only the marked element is considered as the one providing information for the new generated element. Therefore, this rule establishes that only "E1", marked by the designer, acts as a source for the Evolution trace link.

Any element not marked by the designer is ignored in the derivation process. However, these elements are maintained for future analysis tasks.

Both rules presented in this section allow us to obtain the final DW conceptual model, while at the same time generating all the required traces in the process. In this way, the whole trace model from the hybrid to the final PIM is created in an automatic way and does not require user intervention.

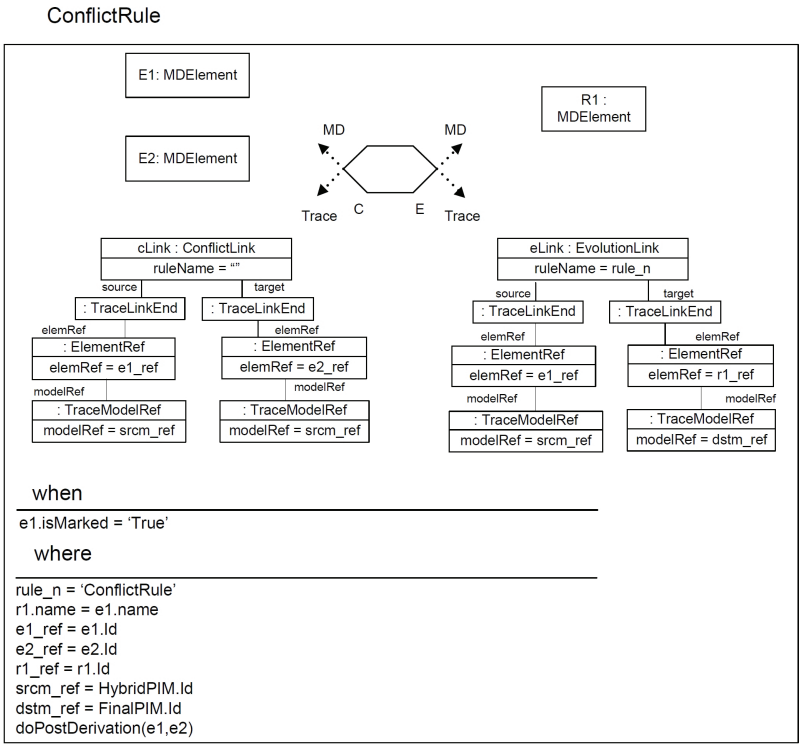

Figure 4: Generic QVT transformation for deriving a data warehouse element from conflicting required elements and sources

4.1. Tool Support

Our approach is supported by the Lucentia BI Suite CASE tool. Our tool is based on Eclipse and is composed by a set of plugins that allow us to model, transform, and derive DW models. The latest version of the Lucentia BI tool supports the elicitation of requirements from users by means of i* for DWs [22], which is implemented in Eclipse using the Ecore framework. These requirements are then derived into conceptual DW models at the PIM Layer [6] that can be modified and enriched by the DW designer, as shown in Figure 5. In this Figure, we can see one of the hierarchies packaged into its dimension package in our multidimensional editor.

In our tool, model to model transformations are implemented by means of ATL (ATLAS Transformation Language [32]). ATL allows us to implemented the QVT rules specified in the previous section, and has been adopted as the *de-facto* standard for implementing model to model transformations. Each ATL transformation included in our tool generates a target DW model and a trace metamodel that stores traceability information. However, as databases do not

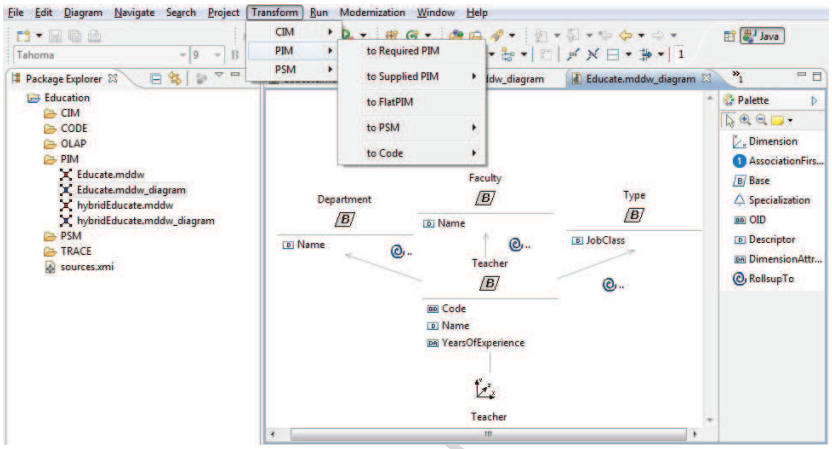

Figure 5: Multidimensional model editor included in the Lucentia BI Suite

always provide a model, and require an inspection in order to extract their metadata, we require additional code for the reverse engineering process. Therefore, the reverse engineering process is performed in two steps. First, when using the Modernization option, the target data sources are inspected by means of Java code, and a CWM model file is created (sources.xmi), as shown in Figure 6.

After the CWM file has been obtained, we generate a hybrid conceptual model using the CWM file and the initial multidimensional model as input for an ATL transformation. The resulting hybrid model contains both elements specified by requirements as well as elements obtained from data sources. Elements within this model are related by means of *Overlap* and *Conflict* traces,

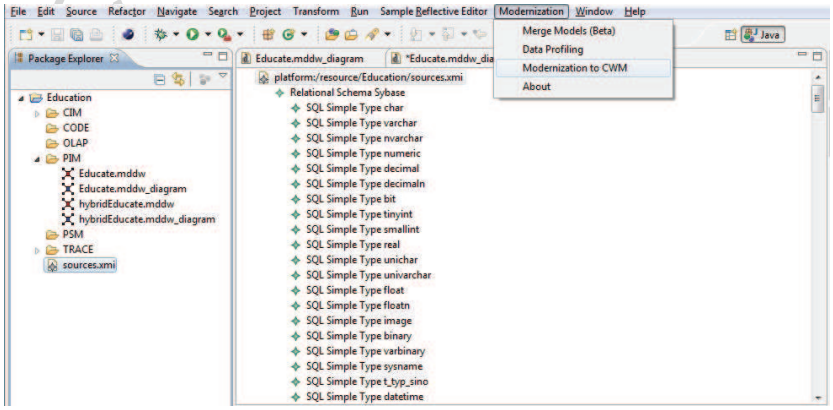

Figure 6: CWM model obtained by reverse engineering data sources

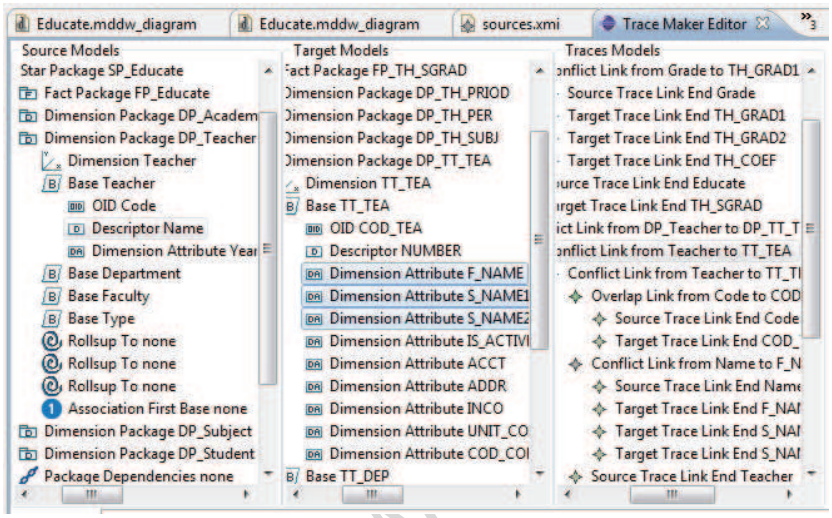

Figure 7: Trace Editor for manual addition of reconciliation traces

specified in our trace editor, shown in Figure 7. The trace editor allows the user to load multiple source and target models and save the traces specified in a trace model.

The traces defined can be inspected or edited again afterwards if necessary. Furthermore, they can be used for model to model transformation or analysis tasks, and can be seen in more detail in Figure 8.

Finally, after we have elaborated the whole set of traces, we make use of the rules specified in the previous section in order to obtain the final data warehouse

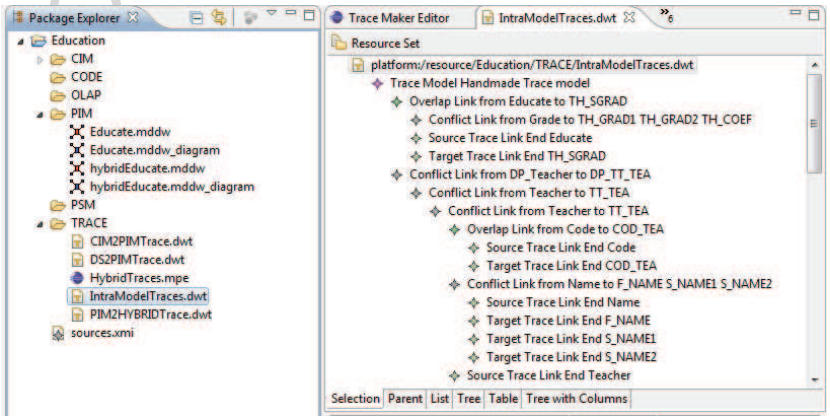

Figure 8: Reconciliation traces included in the trace model generated

model, by using both the hybrid model and the reconciliation traces as input.

5. Case Study

In this section, we will present a case study for our traceability proposal. We will show how traces can be used to derive different configurations of the final PIM, which represents the DW implementation. This case study is inspired on a real world project with another university. The case study describes the basic process of our proposal, while making it easier to read the data source model. All the diagrams are presented with our iconography for DWs [6], that presents UML classes stereotyped according to multidimensional elements in DWs.

5.1. Creation of the Hybrid PIM and Intra-Model Trace Links

A university wishes to improve its educative process. In order to do so, a DW is designed to store the necessary information for the decision making process. The initial PIM, shown in Figure 9, is derived from user requirements, and enriched with the expected attributes. This PIM includes 4 dimensions and a single measure. First, we have the "Subject" dimension. A subject is expected to include its code, a name, the number of credits, and a description of the subject. Additionally, subjects can be aggregated by their "Type". Next, is the "Teacher" dimension. A teacher includes a code, a name, and his years of experience. Furthermore, teachers can be aggregated according to their "Department", their "Faculty", or their job "Type". The third dimension is the "Student" dimension, that stores information regarding students. A student has a name and a code, assigned by the university registry. Students can be aggregated either depending on their "Income" range, or on the "HoursofStudy" they spend each week. Finally, the "AcademicPeriod" dimension, includes a

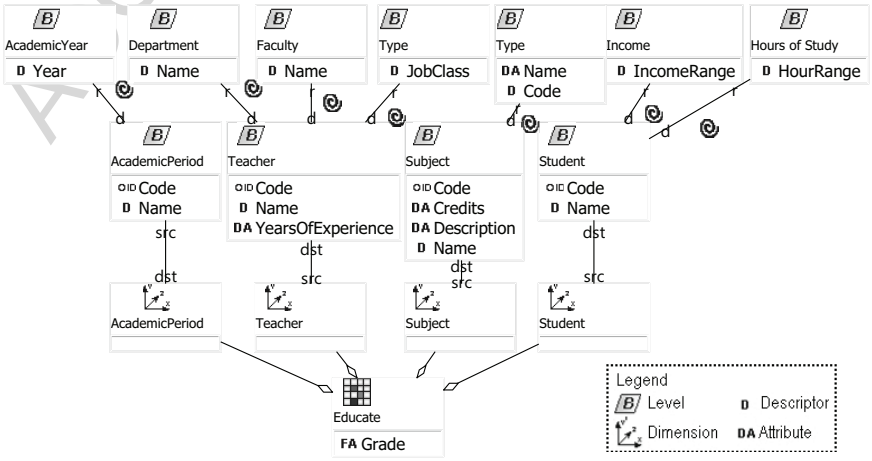

Figure 9: PIM model obtained from user requirements

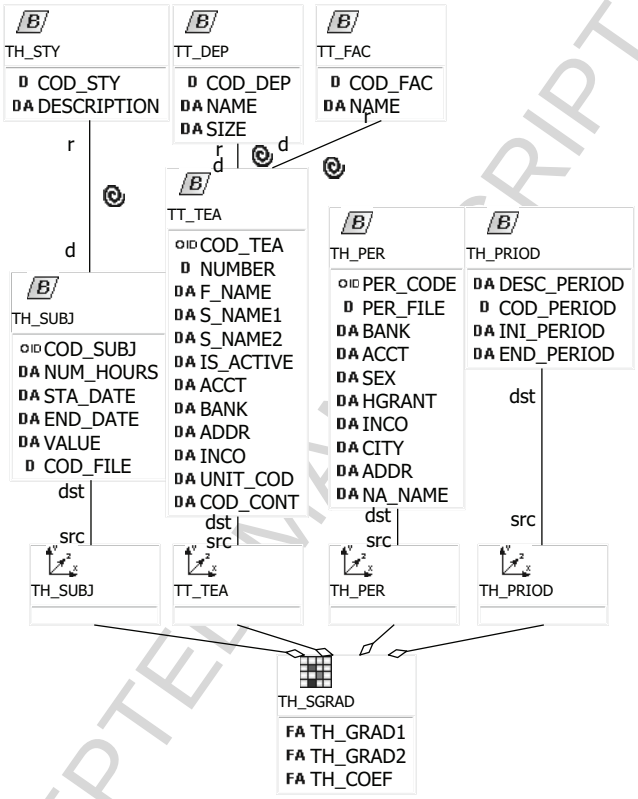

Figure 10: Datasources model obtained by reverse engineering

code and a name assigned to it. Academic periods can be aggregated into academic years. All these dimensions allow us to analyze a performance measure, the "Grade" obtained by the students.

As opposed to this initial PIM, the model created from the data sources[1] presents a higher number of attributes, different naming conventions, and fewer aggregation paths for the dimensions. The model created from the data sources can bee seen in Figure 10. The first dimension is "TH_SUBJ", which would correspond to the previous "Subject" dimension. This dimension includes: a code for the subject, as we expected, the duration in hours of the subject, a starting date, as well as an ending date, a value which cannot be easily identified, and a code for the file of the subject. Subjects may also be grouped by type, as expected, according to the data sources. The next dimension is "TT_TEA",

[1]The data sources model has been restricted to the most relevant concepts for the case study at hand, down from over a hundred tables

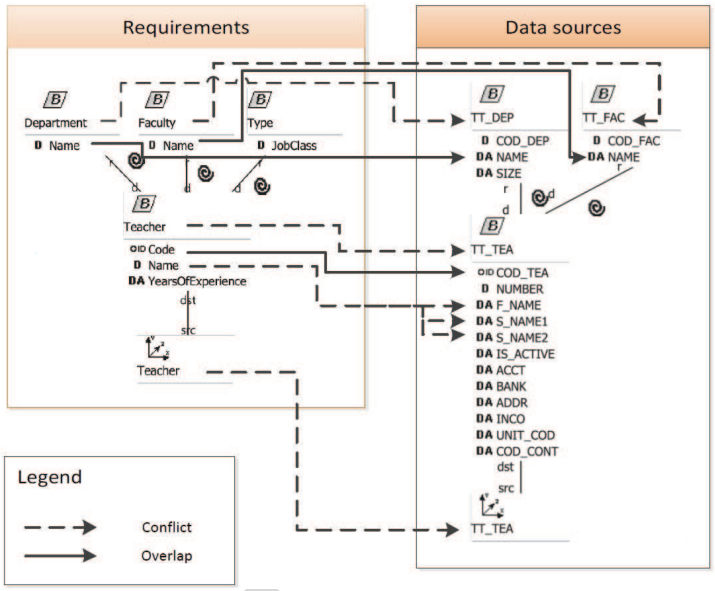

Figure 11: Traces between conceptual elements, Teacher dimension

containing the information related to teachers. The information recorded for a teacher includes his name and surname, a mark for indicating if he is active or not, his bank information, address, three different codes, and his income. According to the data sources, teachers can be grouped either by department or by faculty. If we wanted to group them by their job position, we would need additional elements. The third dimension present is "TH_PER", which stores information about the people registered in the university. The information stored includes a code for the person, his name, the number corresponding to its file, and other personal information, similar to the case of teachers. According to the data sources, this level cannot be aggregated into any other. Finally, the academic periods are stored in "TH_PRIOD", which contains the description of the period, its code, the initial date, and the final date of the period.

After describing both conceptual models, we can analyze the existing differences. First, there are differences in how levels are identified. For example, according to the model obtained, teachers and subjects are identified by their number, instead of by their name. This leads to different aggregated measures in the cube than initially expected, since the same subject may have different codes when included in multiple academic plans. Moreover, some attributes are structured in a different way. For example, "Name" in "Teacher" level is actually fragmented in three different fields. Other attributes do not even appear, such as the subject name, which may be included in its description. Finally, some levels are missing, such as the "Type" of a teacher or the "AcademicYear".

All these differences can be explicitly captured by applying our approach.

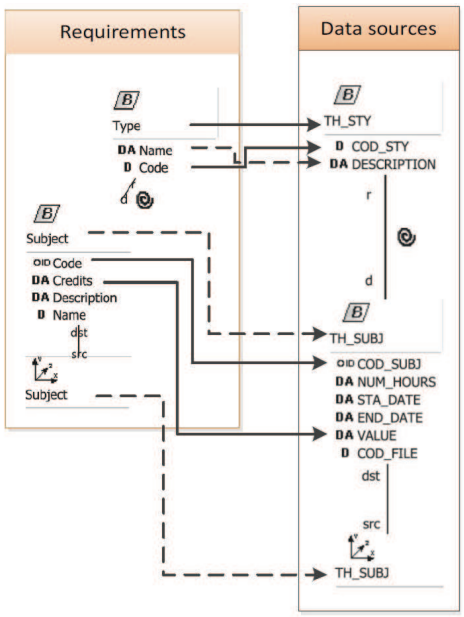

Figure 12: Traces between conceptual elements, Subject dimension

The process starts relating attributes between levels, by applying the definitions specified in Section 3.3. For each attribute in the requirements PIM model, we analyze what data source attributes provide the necessary data. Then, we apply the definitions accordingly to levels and dimensions. The result of this process is seen in Figure 11 for "Teacher" dimension, and in Figure 12 for "Subject" dimension, which are the most complex ones in this schema.

In this Figure we have related user requirements with data sources at conceptual level. As shown in the Figure, most of the levels specified are actually identified by different attributes in the data sources. Furthermore, its specially significant the case where the "Subject" level does not have any counterpart for its descriptor in the data sources. Therefore, a correspondence between the expected set of subjects and the data provided cannot be established unless the schema is modified. Finally, we can see a few attributes in the data sources that are not expected but could be useful for the analysis. For example, "IS_ACTIVE" can determine if a teacher is still active or not.

The lack of key information regarding subjects forces to implement the DW with the data provided as-is, until the missing information is provided.

5.2. Derivation of the Final PIM

After we have obtained the hybrid PIM, and defined the intra-model relationships, we can derive the final PIM. In order to obtain the final PIM, desired elements are marked to be included in the final PIM. Then, they are derived

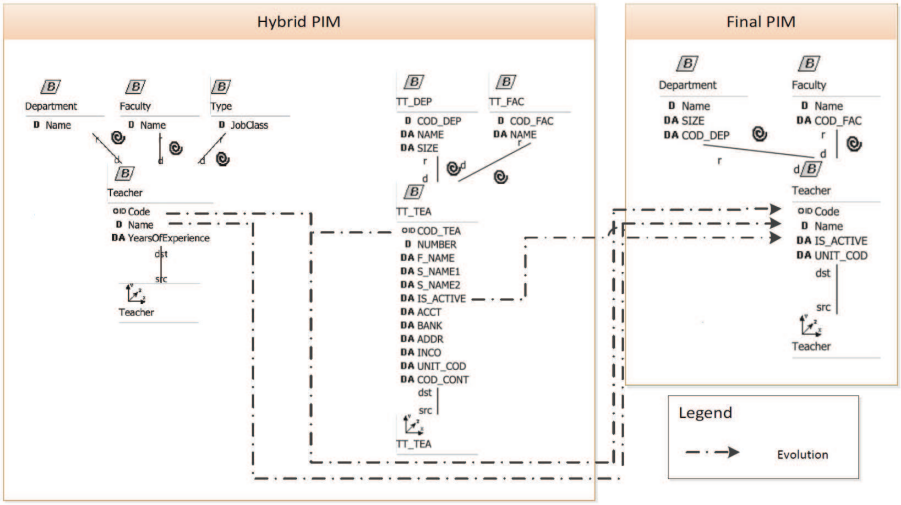

Figure 13: Evolution traces relating conceptual elements from hybrid PIM (left hand) with elements in the final PIM (right hand)

along with *Evolution* traces. Given the high number of traces present, performing this process manually would be time consuming and error prone. Therefore, our specified QVT rules allow us to avoid this pitfall by performing it automatically. The result is shown in Figure 13, where a handful Evolution traces are shown, covering the three different possibilities: (i) evolution from an overlap, (ii) evolution from a conflict, and (iii) evolution from a single element. Finally, in Figure 14, the whole final PIM schema can be seen.

The final PIM can be further refined, by renaming or deleting elements that were initially included for the implementation but are lacking the necessary information to be filled. In addition, with our approach, we can perform quick updates as changes are introduced into user requirements or data sources.

After implementing the DW, data sources are updated to include the subject name. To analyze how this change affects us, we analyze the hybrid PIM model, and evaluate the different levels related to "TH_SUBJ". The process shows that level "Subject" was initially missing the necessary information to be identified, thus it could not be implemented properly. After the update, this information is no longer missing, and the new column can be related to the descriptor. Therefore, now we can properly derive "Subject" instead of "TH_SUBJ", thus being able to analyze subjects as users initially expected, instead of differentiating them by code and obtaining different measures.

With the previous development process, this change would have required to either (i) explicitly keep track of all the missing attributes, or (ii) perform the whole reconciliation process again, since the hybrid PIM was transformed and thus, it would had to be matched against the data sources after the update.

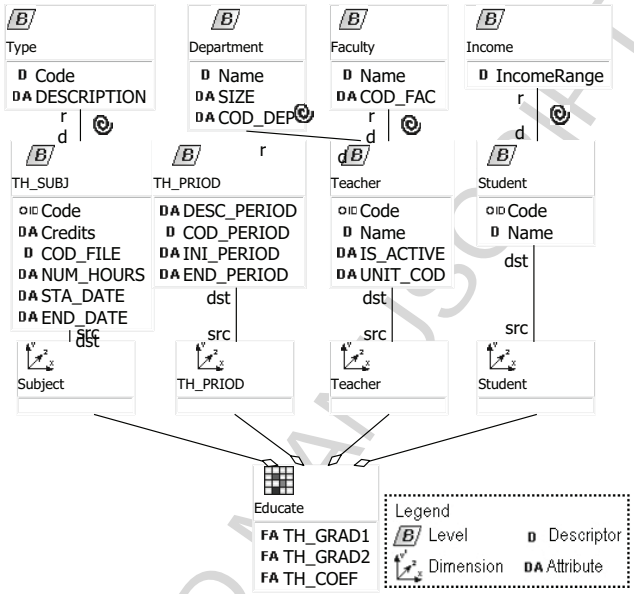

Figure 14: Final PIM representing the data warehouse implementation

Instead, by applying our proposal, new changes can be quickly identified and have a minor impact in the DW schema. Thus, they can be easily assessed and incorporated into the final DW.

6. Conclusions and Future Work

In this paper, we have proposed a traceability approach in order to record explicitly the relationships between elements at the conceptual level in DWs. We have defined the necessary trace semantics to record these relationships, and have formalized them in order to guide their application. Furthermore, we have shown how trace derivation and recording can be automated. We have also exemplified the application of the proposal by means of a case study with another university. The great benefit of our proposal is that the reconciliation task is only performed once per element, and is preserved for further derivations or changes. Therefore, we avoid having to repeteadly inspect the data sources in order to match conceptual elements coming from requirements with those coming from data sources. In turn, we reduce the amount of time and resources spent and improve the maintainability of the system.

The current challenges involve improving the process by semi-automating the generation of reconciliation traces and obtaining initial Extraction/Transformation/Load processes from their definition. The semi-automation of trace generation will require an approach to guide the designer on what reverse engineering

method to use, depending on the information provided by user requirements and data sources. Regarding the generation of ETL processes, this generation will require to specify model transformations to transform trace models into conceptual ETL models. To this aim, the transformations will need to consider what subsets of traces that fit best into a single ETL process.

Our future works involve developing a set of algorithms to propagate changes and to analyze the quality of the matching performed. In the long term, we plan to tackle the generation of an initial subset of traces automatically, thus helping the developer to perform the reconciliation process. The resulting algorithms will be implemented in our tool in order to improve its usability and further shorten the time and effort required to develop the data warehouse. Finally, we plan to introduce visual support for analyzing the trace models, in order to provide the designer with quick view of the relationships modeled.

Acknowledgments.

This work has been partially supported by the MESOLAP (TIN2010-14860) and SERENIDAD (PEII-11-0327-7035) projects from the Spanish Ministry of Education and the Junta de Comunidades de Castilla La Mancha respectively. Alejandro Maté is funded by the Generalitat Valenciana under an ACIF grant (ACIF/2010/298).

References

[1] R. Kimball, M. Ross, W. Thornthwaite, J. Mundy, B. Becker, The data warehouse lifecycle toolkit, Wiley, 2011.

[2] P. Giorgini, S. Rizzi, M. Garzetti, GRAnD: A goal-oriented approach to requirement analysis in data warehouses, Decision and Support Systems 45 (1) (2008) 4–21.

[3] J.-N. Mazón, J. Trujillo, An MDA approach for the development of data warehouses, Decision Support Systems 45 (1) (2008) 41–58.

[4] A. Bitterer, K. Schlegel, D. Laney, Predicts 2012: Business Intelligence Still Subject to Nontechnical Challenges (2011).
URL `http://www.gartner.com/DisplayDocument?ref=clientFriendlyUrl&id=1873915`

[5] S. Winkler, J. von Pilgrim, A survey of traceability in requirements engineering and model-driven development, Software and Systems Modeling 9 (2010) 529–565.

[6] S. Luján-Mora, J. Trujillo, I.-Y. Song, A UML profile for multidimensional modeling in data warehouses, Data & Knowledge Engineering 59 (3) (2006) 725–769.

[7] J.-N. Mazón, J. Pardillo, J. Trujillo, A model-driven goal-oriented requirement engineering approach for data warehouses, Advances in Conceptual Modeling–ER 2007 (2007) 255–264.

[8] J.-N. Mazón, J. Trujillo, A hybrid model driven development framework for the multidimensional modeling of data warehouses, SIGMOD Record 38 (2) (2009) 12–17.

[9] Object Management Group (OMG), A Proposal for an MDA Foundation Model. (2005)
URL http://www.omg.org/cgi-bin/doc?ormsc/05-04-01

[10] Object Management Group (OMG), The Meta-Object Facility 2.0 Query/View/Transformation. Final Adopted Specification. (2005)
URL http://www.omg.org/spec/QVT/

[11] A. Maté, J. Trujillo, Incorporating traceability in conceptual models for data warehouses by using MDA, Conceptual Modeling–ER 2011 (2011) 459–466.

[12] O. Gotel, S. Morris, Macro-level Traceability Via Media Transformations (2008) 129–134.

[13] B. Ramesh, M. Jarke, Toward reference models for requirements traceability, IEEE Transactions on Software Engineering 27 (1) (2001) 58–93.

[14] G. Spanoudakis, A. Zisman, Software traceability: a roadmap, Handbook of Software Engineering and Knowledge Engineering.

[15] Y. Yu, J. Jurjens, J. Mylopoulos, Traceability for the maintenance of secure software, IEEE International Conference on Software Maintenance (2008) 297–306.

[16] N. Aizenbud-Reshef, B. Nolan, J. Rubin, Y. Shaham-Gafni, Model traceability, IBM Systems Journal 45 (3) (2006) 515–526.

[17] F. Jouault, Loosely coupled traceability for atl, Proceedings of the European Conference on Model Driven Architecture (ECMDA) workshop on traceability (2005) 29–37.

[18] R. Paige, G. Olsen, D. Kolovos, S. Zschaler, C. Power, Building model-driven engineering traceability classifications, Proceedings of the European Conference on Model Driven Architecture (ECMDA) workshop on traceability (2008) 49–58.

[19] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, E. Merlo, Recovering traceability links between code and documentation, IEEE Transactions on Software Engineering 28 (10) (2002) 970–983.

[20] H. Asuncion, A. Asuncion, R. Taylor, Software traceability with topic modeling, Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (2010) 95–104.

[21] H. Vranesic, L. Rovan, Ontology-based data warehouse development process, Proceedings of the 31st International Conference on Information Technology Interfaces (ITI'09) (2009) 205–210.

[22] A. Maté, J. Trujillo, A trace metamodel proposal based on the model driven architecture framework for the traceability of user requirements in data warehouses, Information Systems 37 (8) (2012) 753–766.

[23] P. Vassiliadis, Data Warehouse Modeling and Quality Issues, Ph.D. thesis, Athens (2000).

[24] Y. Cui, J. Widom, Lineage tracing in a data warehousing system, Proceedings of the 16th International Conference on Data Engineering (2000) 683–684.

[25] A. Abelló, J. Samos, F. Saltor, YAM2: a multidimensional conceptual model extending UML, Information Systems 31 (6) (2006) 541–567.

[26] M. Golfarelli, D. Maio, S. Rizzi, The dimensional fact model: a conceptual model for data warehouses, International Journal of Cooperative Information Systems 7 (2) (1998) 215–247.

[27] C. Neil, J. Irazábal, M. De Vincenzi, C. Pons, Graphical Query Mechanism for Historical Data Warehouse within MDD, Proceedings of the XXIX International Conference of the Chilean Computer Science Society (SCCC) (2010) 183–192.

[28] C. G. Neil, C. Pons, Aplicando MDA al Diseño de un Datawarehouse Temporal, Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento (JIISIC) (2007) 181–189.

[29] J.-N. Mazón, J. Trujillo, J. Lechtenbörger, Reconciling requirement-driven data warehouses with data sources via multidimensional normal forms, Data & Knowledge Engineering 63 (3) (2007) 725–751.

[30] M. Del Fabro, J. Bézivin, P. Valduriez, Weaving Models with the Eclipse AMW plugin, Eclipse Modeling Symposium, Eclipse Summit Europe (2005).

[31] Object Management Group (OMG), Common Warehouse Metamodel. (2003)
URL `http://www.omg.org/spec/CWM/1.1/`

[32] F. Jouault, I. Kurtev, Transforming models with ATL, Satellite Events at the MoDELS 2005 Conference (2005) 128–138.

Highlights

- A set of semantic traces to support conceptual traceability in data warehouses
- Simplifies analysis of the impact of changes and integration of data new sources
- Includes a set of rules based on the QVT standard for deriving the data warehouse
- We apply our proposal to create an education data mart