Dimensional Templates in Data Warehouses: Automating the Multidimensional Design of Data Warehouse Prototypes

Rui Oliveira[1], Fátima Rodrigues[2], Paulo Martins[3], and João Paulo Moura[3]

[1] Dep. Engenharia Informática, Escola Superior de Tecnologia e Gestão
Instituto Politécnico de Leiria, 2411-901 Leiria, Portugal
rui.oliveira@estg.ipleiria.pt
[2] GECAD – Grupo de Investigação em Engenharia do Conhecimento e Apoio à Decisão
Dep. Engenharia Informática, Instituto Superior de Engenharia do Porto, Porto, Portugal
mfc@isep.ipp.pt
[3] GECAD – Grupo de Investigação em Engenharia do Conhecimento e Apoio à Decisão
Universidade de Trás-os-Montes e Alto Douro, Vila Real, Portugal
{pmartins,jpmoura}@utad.pt

Abstract. Prototypes are valuable tools in Data Warehouse (DW) projects. DW prototypes can help end-users to get an accurate preview of a future DW system, along with its advantages and constraints. However, DW prototypes have considerably smaller development time windows when compared to complete DW projects. This puts additional pressure on the achievement of the expected prototypes' high quality standards, especially at the highly time consuming multidimensional design: in it, a thin margin for harmful unreflected decisions exists. Some devised methods for automating DW multidimensional design can be used to accelerate this stage, yet they are more suitable to DW full projects rather than to prototypes, due to the effort, cost and expertise they require. This paper proposes the semi-automation of DW multidimensional designs using templates. We believe this approach better fits the development speed and cost constraints of DW prototyping since templates are pre-built highly adaptable and highly reusable solutions.

Keywords: Data Warehouse, Automated Multidimensional Design, Dimensional Templates, Prototype Development.

1 Introduction

Prototypes are valuable tools in DW projects and much as been written on the subject, not only by field practitioners [1,2] but also by the scientific community [3], to mention only a few. A first benefit of DW prototypes is that they act as a preview of the satisfiable end-users' requirements considering the available data sources. This avoids later costly disappointments about the DW outcome. Secondly, DW prototypes allow predicting with a high degree of confidence the restraining factors on the future

J. Filipe and J. Cordeiro (Eds.): ICEIS 2009, LNBIP 24, pp. 184–195, 2009.

DW project, such as cost, size or deadlines. Finally, DW prototypes are ideal to materialize the benefits of future DW projects, thus helping to justify the overall investment.

DW prototypes have shorter development time windows and considerably more restrained budgets when compared to full DW projects. However, the urge to develop a DW prototype cannot endorse a low quality product. In fact, a DW prototype must be more than just a DW sketch in which design and implementation errors will vanish once the prototype is thrown away. Quite the opposite, a DW prototype must be built as a launching platform to a full-scale DW project once it has proven its point [2,3]. Therefore, if it is reasonable to expect that a DW prototype can be incomplete due to time and cost restrictions, it is less so in what concerns accuracy.

Accuracy and fast development are not easy to balance goals in DWs. This happens due to the many sensible phases requiring well-reflected decisions, such as the multidimensional design stage. Accurate multidimensional design consumes a considerable amount of time and human expert resources [4,5]: business requirements must be gathered, data sources must be deeply analysed, DW expertise must be acquired, and performance plus storage sustainability must be assured. Such tasks do not fully comply with the pressure of time, especially in highly time-restrained environments such as DW prototypes.

Aiming to accelerate the multidimensional design stage of DWs, some semi-automated methods have been devised, such as [6,7,8], among others. Although effective and justifiable in the context of a complete DW system development, these methods' requirements can be inadequate for most DW prototypes. This happens because such methods need a deep understanding of data sources by DW designers, solid multidimensional design expertise or even specific data source documentation. Such demands consume time and budget resources not likely attainable in many DW prototype projects.

It is our belief that DW prototyping provides the perfect conditions for introducing the use of generic multidimensional solutions rather than personalized ones. In that sense, the current paper proposes the innovative use of templates as a way of performing multidimensional design in DW prototypes. This approach aims to solve some key problems that effectively delay this development stage. In fact, templates are a widely accepted mechanism in many informatics areas with the purpose of accelerating software development speed. Even though the resulting product turns out to be neither personalized nor optimised, its structure has a quality guaranty and allows additional refinements. Also, the use of templates avoids the need to gather expert knowledge in order to perform common tasks. Finally, templates are generic and therefore adaptable to the needs of a wide number of particular scenarios. From our perspective, such positive qualities can be successfully adapted to the DW prototyping domain, as addressed in this paper.

The paper is structured as follows. Section 2 briefly presents the state of the art on multidimensional design semi-automation methods, template usage and DW prototyping tools. Section 3 describes the overall concept of the proposed approach and the construction of dimensional templates. Section 4 presents the algorithm for generating multidimensional structures from dimensional templates. Finally, section 5 concludes the paper.

2 Related Work

In the past, valuable methods have been devised to semi-automate the multidimensional design of DWs, with focus on diminishing the time consumed while maintaining high quality standards. Some of these methods provide semi-automated multidimensional design based on specific formats such as E/R schemes [9], XML schemes [6], object oriented [10,11] or even oriented to ontologies [12]. At some extent, such methods help reducing DW projects' development time and DW experts' involvement. The main drawback in using these powerful techniques is that they require source data to be well documented using a specific format, like E/R schemes, UML diagrams or even an ontology language. Even though these are scientifically accepted specifications, many documentation problems may easily rise concerning source data's documentation [13]: poor maintenance, physical implementations differing from logical designs, the use of non-standard formats and, the worst case of all, no documentation at all.

Other multidimensional semi-automated methods avoid the need of specific documentation formats to describe data relationships and processes at source systems. For instance, [8] proposes a method for multidimensional design automation by applying a generation algorithm to data sources previously tagged with multidimensional markers. To be effective, the method simultaneously requires a thorough inspection of data sources and the existence of dimensional expertise from the ones in charge of that task: otherwise, the resulting multidimensional design can be far from correctness. Also, the approach assumes that a third normal form database is used, not accounting for cases in which a high degree of denormalization is found. Again, it is a valuable method assuming that the available time window for multidimensional design is comfortable and that the cost of getting DW experts' support is acceptable at the early prototyping phase.

Another known multidimensional semi-automated method based on the analysis of data sources without the need of specific documentation is [14]. In it, reverse engineering is used to obtain relational metadata. We consider the method not the best choice for DW prototyping, since it is not suitable for large and complex systems. Also, the method becomes hard to apply since it uses non-standard modelling techniques.

Also worth mentioning is the method proposed in [7], which emphasises the use of end-user requirement driven multidimensional design. Although powerful to validate end-user requirements against organizational data in an automated way, it requires a strong interaction between DW design experts and end-users. Also, it requires that organizations' processes be specified using the method's particular format.

As concerns the use of templates, they are a widely accepted approach in the informatics area for automating operations. The range of applications goes from the most simple office software to highly demanding industrial tools. That we know of, there are no proposals of template usage to automate the construction of multidimensional designs. Authors in the literature of multidimensional design, such as [15,16] among others, present techniques, guidelines and standard models for building multidimensional designs from scratch. However, their proposals cannot be seen as templates but rather as theoretical models requiring significant DW expertise to be understood and adapted.

As concerns software tools dedicated to DW prototypes' management, some choices are available at the time of this writing, like [3, 17]. However, today's

Extract- Transform-Load (ETL) tools can be effectively used to prototype DW systems, ranging from open source to commercial tools. [18] gives an extensive list of ETL tools, even though others exist. Common to the generality of DW prototyping tools (dedicated or standard ETL) is the lack of support given to the automation of multidimensional designs, which is the context of the proposed approach. Although such tools can support the crucial implementation and maintenance phases of DW prototypes, a global assumption is that a previously devised multidimensional design already exists.

3 Dimensional Templates

In this paper we propose the use of templates to automate the multidimensional design phase of DW prototypes. To distinguish the here-proposed templates from other areas' templates, ours are named *dimensional templates*.

Along this paper, the case study of a retail sales company is used to illustrate the proposed concepts, particularly its business process *retail sales* [15].

3.1 The Overall Concept

Fig. 1 depicts the three stages through which a dimensional template can go through, described as follows:

Construction. End-user requirements (EURs) concerning generic distinct business processes (like *retail sales* or *inventory levels*) are represented using logical models named *rationale diagrams*. The set of rationale diagrams concerning a specific business process constitutes a *dimensional template*. This stage, to be conducted by DW design experts, is analysed in section 3.

Acquisition. This stage is conducted from inside the organization requiring the multidimensional design for a DW prototype. After gathering the EURs for the DW prototype, the necessary dimensional templates are acquired. This stage requires no DW knowledge.

Configuration. Dimensional templates are suitable to a non-limited number of real scenarios. In order to generate a multidimensional design for an organization's particular scenario, the dimensional templates gathered at the *acquisition stage* need to be configured and latter processed by a generation algorithm. This stage, which requires no DW knowledge, is further analysed in section 4.

3.2 Building Dimensional Templates

As mentioned, dimensional templates are composed of rationale diagrams representing generic EURs for a particular DW business process. Concerning the formal representation of EURs, much has been written. Existing methods, like [7,19], extend the original *i** framework specifically for DW development. The work of [7] was found to be the most useful for our approach, due to its simple notation. From it, some basic elements were imported. These are as follows:

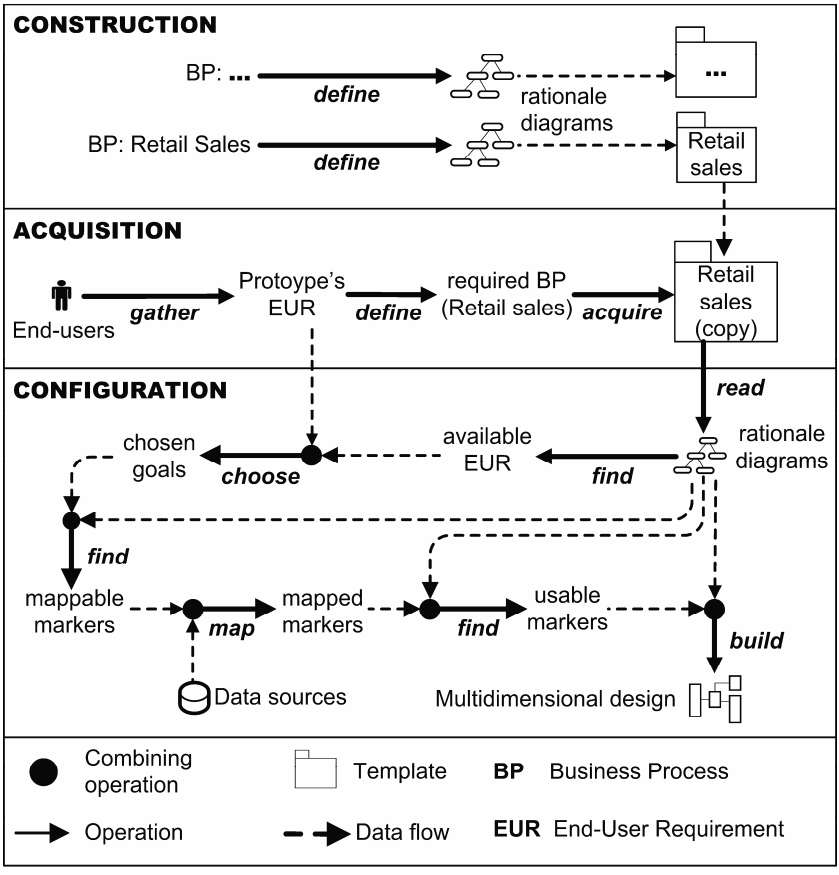


Fig. 1. Overall view of the approach using the case study of a retail-sales company requiring a DW prototype

Goal. Represents an EUR. A goal can be decomposed into more specific child goals representing more detailed versions of their parent goal.

Decomposition. Represents the division of a parent goal into several child goals. A decomposition can be an AND-decomposition (every child goal must be satisfied so that its parent goal is satisfied) or an OR-decomposition (at least one child goal must be satisfied so that its parent goal is satisfied).

Rationale Diagram. Logical representation of an EUR with all its decompositions. In its original form [7], rationale diagrams are used to relate EURs with *actors* and *facts*, but these two concepts were not imported.

In the context of our approach we have extended the concepts of *goal*, *decomposition* and *rationale diagram* with new key elements, as described next.

Table 2 depicts the graphical notation of our rationale diagrams' elements.

Grain-goal. Representation of a goal in the context of a specific grain (the data's granularity). The grains associated to *grain-goals* are the ones considered as being *reasonable* for the specific business process. Table 1 shows some examples of grains described as reasonable. This means that other grains may exist, yet the amount of data required to satisfy them would become unmanageable (like a *bit* grain for the Network Cable Company's scenario). Since the number of reasonable grains for each business process is small, the amount of possible grain-goals for each child goal does not compromise rationale diagrams' manageability.

Table 1. Notation used in the proposed rationale diagrams

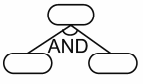
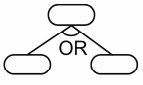
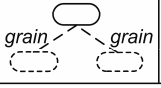
Symbol	Meaning	Symbol	Meaning
	goal		AND-decomposition
	grain-goal		
	dimensional context		OR-decomposition
	marker		grain-decomposition

Table 2. Reasonable grains for two distinct scenarios

Scenario	Business Process	Reasonable Grains
Retail Sales Company	Retail Sales	Sale Line-of-sale
	Inventory Levels	Periodic snapshot Transaction
Network Cable Company	Customer Billing	Bill Customer session Packet
	Network Traffic	Packet

Grain-decomposition. Represents the division of a goal into as many grain-goals as the number of reasonable grains.

Marker. The representation of a type of data required to exist in data source systems so that a specific grain-goal can be satisfied.

Dimensional Context. The information context into which a specific marker fits. Information contexts are detailed further on.

3.3 Rationale Diagrams

Each rationale diagram, as presented in this paper, is a logical representation of the source systems' data required to satisfy a certain EUR, that is, a logical mapping of goals to markers. Fig. 2 depicts a rationale diagram for the goal *Analyse product sales*

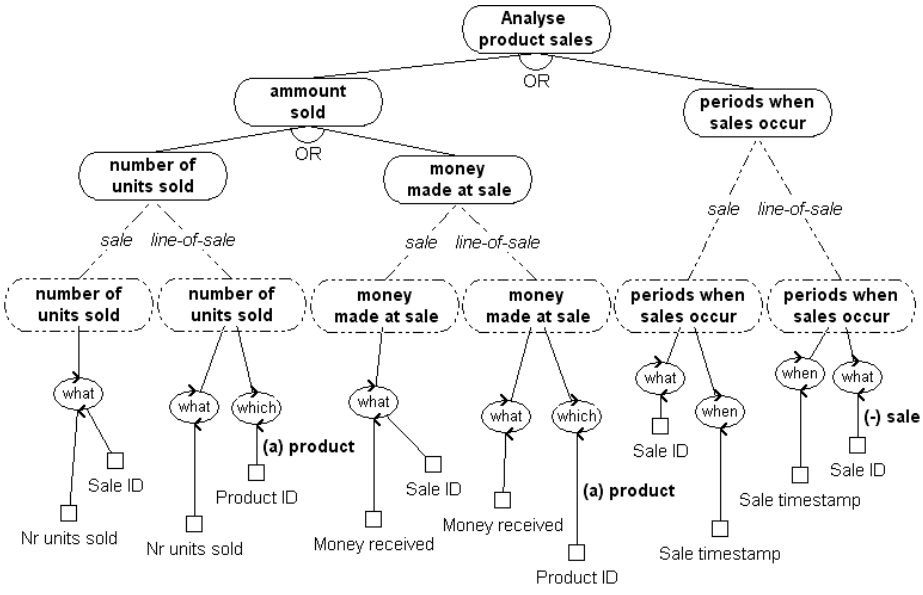


Fig. 2. Simplified rationale diagram for the EUR Analyse product sales in the retail sales company case study

(simplified for clarity sake) concerning the retail sales company case study. As shown, the parent goal is divided into more detailed versions (child goals) using OR-decompositions. At the lowest level of each OR-decomposition, a final division into grain-goals is performed. In rationale diagrams, in general, if no AND/OR decomposition is considered adequate for a goal, a grain-decomposition is applied.

Dimensional Contexts. The primary role of multidimensional structures is to enhance the ability of end-users to answer the question *why did facts occur in the source system?*. The *why* question can be decomposed into one *f*-question and five *d*-questions, each aiming to clarify facts occurrence in distinct information perspectives.

The *f*-question is *what happened in the source systems?* and it is answered by the facts and measures found in fact tables. The *d*-questions can be answered using dimension tables plus their foreign keys' links to fact tables, and are as follows:

- *How* did facts took place (what where the environmental conditions when facts occurred? E.g., promotions, discounts);
- *When* did facts occur (time);
- *Where* did facts took place (e.g., store, web, warehouse);
- *Which* agents passively participated in facts' occurrence (e.g., product, web page);
- *Who* did actively motivated the facts' occurrence (e.g., salesman, customer).

As concerns our approach, we have defined the concept of *dimensional context*, representing the informative context to which any multidimensional structure relates. Therefore, six dimensional contexts can be found in a multidimensional schema: *how*, *what*, *when*, *where*, *which* and *who*.

Markers and Dimensional Contexts. At this point, four assumptions (*A*) can be made and a corollary (*C*) can be derived: (*A1*) EURs are satisfied by multidimensional structures and their data; (*A2*) a multidimensional structure is always related to a dimensional context; (*A3*) the data contained in a multidimensional structure shares the structure's dimensional context; (*A4*) the data contained in a multidimensional structure is transformed/cleansed data originating from source systems; (*C*) every source data element which satisfies an EUR has a dimensional context (and so will the marker representing such data element).

Fig. 2 helps illustrating the previous corollary: the marker *Product ID* linked to the grain-goal *number of units sold* clearly belongs to *which* context, since it refers to something that passively participates in facts' occurrence (e.g., products). Analysing the same grain-goal, the marker *nr units sold* answers no *d*-question: then, by default, it answers the *f*-question (thus relating to the *what* context).

Tagging Markers. A marker represents a type of source systems' data required to satisfy grain-goals. Generally, the grain of a marker's data is the same as the grain-goal's grain it relates to. For instance, Fig. 2 shows that to satisfy the grain-goal *number of units sold* at the grain level *line-of-sale*, the number of units sold for each line-of-sale is required (marker *nr units sold*).

However, grain-goals may eventually require markers with a lower grain level than its own (*sale* grain is considered lower than *line-of-sale* grain because it supports less detailed data). Analysing Fig. 2, the grain-goal *periods when sales occur* at the line-of-sale grain level is satisfiable with the *Sale ID* marker, which relates to sales, while the grain-goal refers to lines-of-sale (different grain levels). This exceptions may occur with markers related to *when* and *what* dimensional contexts. Once detected, they are dealt by tagging the corresponding marker-dimensional context connection with the *(-)* symbol followed by the name of the grain the marker refers to.

As concerns markers related to *how*, *where*, *which* and *who* dimensional contexts, it is important to mention the business agent involved in the grain-goal's satisfaction. An agent is a source systems' physical actor or event to which a marker refers. For instance, in the addressed case study, common agents for *which*-related markers are *product*, while for *who*-related markers two common agents are *customer* and *employee*. Agents are represented in rationale diagrams by tagging the corresponding marker-dimensional context connection with the *(a)* symbol followed by the agent's name (see Fig. 2 for some examples of tagging with the *product* agent).

4 Using Dimensional Templates

In this section we briefly present the algorithm for generating multidimensional designs from rationale diagrams (Fig. 1, configuration stage). Some screen captures of a template configuration tool prototype developed by the authors are used to illustrate the several steps of the generation algorithm. It is worth mentioning that the theoretical concepts used to build the algorithm are the widely accepted ones of [15].

Consider following Fig. 1 (configuration stage) and Fig. 2 for a better understanding of the algorithm's explanation, since the case study of retail sales will also be used in this section. A series of definitions (*D*) will be used throughout the algorithm's explanation.

At this point, it will be assumed that the dimensional templates required to satisfy the business processes found at the acquisition stage have been gathered. Such templates include rationale diagrams containing DW EURs in the form of goals.

4.1 Step 1: Finding Mappable Markers

The generation algorithm will not use all the goals contained inside dimensional templates, but only those who match the EURs defined at the acquisition stage (*D1: chosen goal*). The algorithm then accesses the template's rationale diagrams to determine which markers must be mapped in order to satisfy each of the chosen goals (*D2: mappable marker*). Fig. 3 depicts the mappable markers for the chosen goal *Money made at sale* (also visible in Fig. 2) for each of its grain-goals.

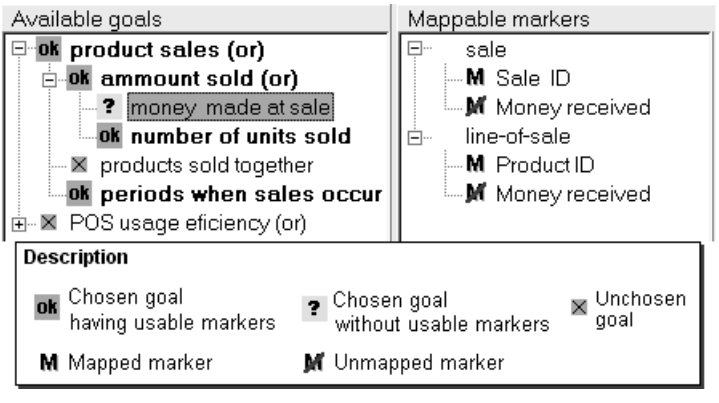


Fig. 3. Partial screen capture of a template configuration tool showing the *chosen goals* and their *mappable markers* at each grain, which can also be seen in the rationale diagram of Fig. 2

4.2 Step 2: Mapping Markers

Mappable markers are useless until they are mapped to real source data. A mapped marker consists of a marker to which the correct physical location of data has been provided (*D3: mapped marker*). This mapping operation is important to define the logical data map [20] after multidimensional structures have been generated.

4.3 Step 3: Determining Usable Markers

If all of a grain-goal's markers are mapped, those markers will be considered *usable* (*D4: usable marker*). The algorithm will only consider for multidimensional generation the usable markers found. In Fig. 3 it is visible that the goal *Money made at sale* has unmapped markers at all grains. This means that none of its markers is *usable* (the *Sale ID* and *Product ID* markers, although mapped, are not usable).

4.4 Step 4: Determining Satisfied Goals

A grain-goal is considered satisfied if it only contains usable markers (*D5: satisfied grain-goal*). A goal having child goals will be considered satisfied if (i) an AND-decomposition

is used and *all* of its child goals are satisfied or if (ii) an OR-decomposition is used and *at least one* of its child goals is satisfied (*D6: satisfied goal*). A goal having only child grain-goals, like *Money made at sale*, will be considered satisfied if it has *at least one* satisfied grain-goal.

4.5 Step 5: Multidimensional Generation

According to [15], different grains need to be addressed by separate fact tables and therefore by distinct multidimensional designs (*star-schemas*). Accordingly, our generation algorithm must be able to generate as many distinct star-schema models as the number of reasonable grains for which satisfied grain-goals exist (*D5*). In order to do so, the algorithm must be run one time for each reasonable grain (an algorithm iteration). Each iteration thus refers to a single grain (*D7: iteration grain*) and will generate its own fact table. For each algorithm's iteration, the steps are as follows:

1. With usable markers linked to *what* or *when* dimensional contexts and linked to grain-goals with the same grain as the iteration's grain, finds all distinct trios <*marker*, *marker's dimensional context*, *marker's grain*>. From Fig. 2 goal *periods when sales occur* at the line-of-sale grain, the retrieved trio for the iteration grain line-of-sale is <*Sale ID*, *what*, *sale*>. For each distinct trio found:
   - 1.1 If the dimensional context is *when*, time related multidimensional elements are required:
     - 1.1.1 If the marker's grain is the same as the iteration's grain, a foreign key is created between the iteration's fact table and the time dimension.
     - 1.1.2 If the marker's grain is lower than the iteration's grain then a measure is created in the fact table, using the marker's name.
   - 1.2 If the dimensional context is *what*, a fact table related element is necessary:
     - 1.2.1 If the marker's grain is the same as the iteration's grain then a measure is created in the fact table, using the marker's name.
     - 1.2.2 If the marker's grain is lower than the iteration's grain, a degenerated dimension is created in the fact table, using the marker's name.
2. With usable markers linked to *how*, *where*, *which* or *who* dimensional contexts and linked to grain-goals with the same grain as the iteration's grain, finds all distinct trios <*marker*, *marker's dimensional context*, *marker's agent*>. From Fig. 2 goal *number of units sold* at the line-of-sale grain, the retrieved trio for the iteration grain line-of-sale is <*Product ID*, *which*, *product*>. For each distinct trio found:
   - 2.1 If no dimension table has yet been created for the marker's agent:
     - 2.1.1 Create a dimension using the marker's agent name.
     - 2.1.2 Create a foreign key between the iteration's fact table and the new dimension.
   - 2.2 Creates a column in the marker's agent dimension, using the marker's agent name.

Fig. 4 shows a multidimensional model generated by using the fulfilled goals at Fig. 3 with the iteration grain line-of-sale. The picture is a partial screen capture from a template configuration tool devised by the authors of this paper.

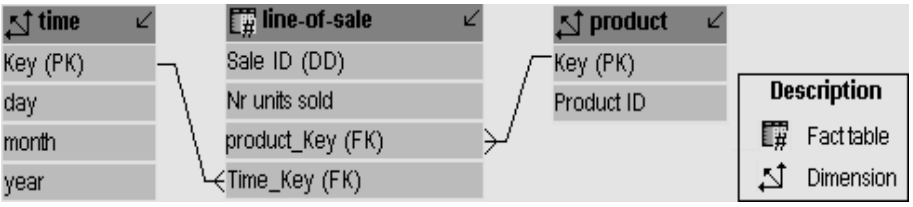


Fig. 4. Multidimensional model generated using the fulfilled goals at Fig. 3 and an iteration grain line-of-sale (*DD=Degenerated Dimension*; *FK=Foreign Key*; *PK=Primary Key*)

5 Conclusions

In this paper we have proposed the use of dimensional templates for automating the multidimensional design of DWs. Dimensional templates are built with a high level of abstraction, thus lowering their management complexity. This is achieved by the use of rationale diagrams, logical models that map end-user requirements to the types of data required to satisfy them.

We believe that our approach is particularly useful in DW prototyping environments, since (i) a dimensional template works as a pre-built solution and (ii) the configuration of templates to generate multidimensional models can be achieved without DW knowledge. These are key features regarding DW prototypes, since these systems highly benefit from a fast boot start plus low cost operations due to their experimental status.

Also, our approach suites better the purpose of automating the multidimensional design in DW prototypes than other existing proposals of the kind. These other automation methods require either extended periods of source data analysis by DW designers, DW design expertise or even exact source data documentation in specific formats: three requirements not compliant with the time and cost requirements of embryonic solutions such as DW prototypes. Even though our approach also requires DW expertise (to build dimensional templates), this initial effort is compensated by the re-usage capacity of the solution and by the lack of time-consuming interaction between DW experts/organizations' end-users that other approaches depend on.

Our approach is fully supported by two prototype tools developed by the authors: a *template builder tool* for creating and managing the rationale diagrams (used to generate Fig. 2); a *template configuration tool* for generating multidimensional designs from dimensional templates (used to generate Fig. 4) and also the related documentation in the *Common Warehouse Model* standard [21]. This last feature, not in the scope of this paper, enhances the scalability feature of the prototyped products, since many ETL tools can import the generated structures.

Interesting future work can be performed as a completion to the work presented in this paper. This includes the semi-automated creation of dimensional templates from real multidimensional designs.

References

1. Look Before You Leap,
`http://www.intelligententerprise.com/010216/feat3_1.jhtml`

2. Data Warehouse Prototyping: Reducing Risk, Securing Commitment and Improving Project Governance, http://www.wherescape.com/white-papers/white-papers.aspx
3. Huynh, T., Schiefer, J.: Prototyping Data Warehouse Systems. In: Kambayashi, Y., Winiwarter, W., Arikawa, M. (eds.) DaWaK 2001. LNCS, vol. 2114, pp. 195–207. Springer, Heidelberg (2001)
4. The Data Warehouse Budget, http://www.datawarehouse.inf.br/Papers/inmonbudget-1.pdf
5. Adelman, S., Dennis, S.: Capitalizing the DW (2005), http://www.dmreview.com/
6. Vrdoljak, B., Banek, M., Rizzi, S.: Designing Web Warehouses from XML Schemas. In: Kambayashi, Y., Mohania, M., Wöß, W. (eds.) DaWaK 2003. LNCS, vol. 2737, pp. 89–98. Springer, Heidelberg (2003)
7. Giorgini, P., Rizzi, S., Garzetti, M.: Goal-Oriented Requirement Analysis for Data Warehouse Design. In: DOLAP 2005, 8th International Workshop on Data Warehousing and OLAP, pp. 47–56. ACM Press, New York (2005)
8. Mazón, J., Trujillo, J.: A Model Driven Modernization Approach for Automatically Deriving Multidimensional Models in Data Warehouses. In: Parent, C., Schewe, K.-D., Storey, V.C., Thalheim, B. (eds.) ER 2007. LNCS, vol. 4801, pp. 56–71. Springer, Heidelberg (2007)
9. Song, I.Y., Khare, R., Bing, D.: SAMSTAR: A Semi-Automated Lexical Method for Generating Star Schemas from an Entity-Relationship Diagram. In: DOLAP 2007, 10th International Workshop on Data Warehousing and OLAP, pp. 9–16. ACM Press, New York (2007)
10. Abelló, A., Samos, J., Saltor, F.: YAM2 (Yet another multidimensional model): An extension of UML. In: International Symposium on Database Engineering & Applications. IEEE Computer Science, pp. 172–181. IEEE Computer Society, Washington (2002)
11. Luján-Mora, S., Trujillo, J., Song, I.Y.: Extending the UML for multidimensional modeling. In: Jézéquel, J.-M., Hussmann, H., Cook, S. (eds.) UML 2002. LNCS, vol. 2460, pp. 265–276. Springer, Heidelberg (2002)
12. Romero, O., Abelló, A.: Automating Multidimensional Design from Ontologies. In: 10th International Workshop on Data Warehousing and OLAP, pp. 1–8. ACM Press, New York (2007)
13. Alhajj, R.: Extracting the Extended Entity-Relationship Model From a Legacy Relational Database. Information Systems 28, 597–618 (2003)
14. Jensen, M., Holmgren, T., Pedersen, T.B.: Discovering Multidimensional Structure in Relational Data. In: Kambayashi, Y., Mohania, M., Wöß, W. (eds.) DaWaK 2004. LNCS, vol. 3181, pp. 138–148. Springer, Heidelberg (2004)
15. Kimball, R., Ross, M.: The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling, 2nd edn. John Wiley and Sons, Inc., USA (2002)
16. Malinowski, E., Zimányi, E.: Advanced Data Warehouse Design: From Conventional to Spatial and Temporal Applications. Springer Publishing Company, Heidelberg (2008)
17. Wherescape RED, http://www.wherescape.com/
18. Alkis Simitsis's list of ETL tools, http://www.dbnet.ece.ntua.gr/~asimi/ETLTools.htm
19. Mazón, J., Pardillo, J., Trujillo, J.: A Model-Driven Goal-Oriented Requirement Engineering Approach for Data Warehouses. In: RIGIM 2007, 1st International Workshop on Requirements, Intentions and Goals in Conceptual Modeling, pp. 255–264. Springer, Heidelberg (2007)
20. Kimball, R., Caserta, J.: The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning,Conforming, and Delivering Data. Wiley Publishing, Inc., USA (2004)
21. Vetterli, T., Vaduva, A., Staudt, M.: Metadata Standards for Data Warehousing: Open Information Model vs. Common Warehouse Metamodel. ACM SIGMOD Record 29, 68–75 (2000)