A Comparative Analysis of Data Warehouse Data Models

Ivan Bojičić, Zoran Marjanović, Nina Turajlić,
Marko Petrović, Milica Vučković
Faculty of Organizational Sciences
University of Belgrade
Belgrade, Serbia
{ivan.bojicic, marjanovic.zoran, turajlic.nina,
petrovic.marko, vuckovic.milica}@fon.bg.ac.rs

Vladan Jovanović
Allen E. Paulson College of Engineering and Information Technology
Georgia Southern University
Statesboro, USA
vladan@georgiasouthern.edu

***Abstract*— The main purpose of data warehouses (DW) is to maintain large volumes of historical data (originating from multiple heterogeneous data sources and representing the different states of a system caused by various business events or activities) in a format that facilitates its analysis in order to support timelier and better decision-making, at both the operational and strategic level.**

In order for a data warehouse to be able to adequately fulfill this purpose, its data model must enable the appropriate and consistent representation of the different states of a system. In effect, a DW data model, representing the physical structure of the DW, must be general enough, to be able to consume data from heterogeneous data sources (where all of the data should be treated as relevant data and it must be possible to trace it back to its source) and reconcile the semantic differences of the data source models, and, at the same time, be resilient to the constant changes in the structure of the data sources.

One of the main problems related to DW development is the absence of a standardized DW data model. In this paper a comparative analysis of the four most prominent DW data models (namely the relational/normalized model, data vault model, anchor model and dimensional model) will be given. These models will be analyzed and compared on the basis of the following criteria: (1) semantics (i.e. the fundamental concepts), (2) resilience of the data model with regard to changes in the structure of the data sources, (3) temporal aspects and (4) completeness and traceability of the data.

By identifying the strengths and weaknesses of each of these models it would be possible to establish the foundation for a new DW data model which would more adequately fulfill the posed requirements.

Index Terms-data warehouse; data models; relational/normalized model; data vault model; anchor model; dimensional model;

I. INTRODUCTION

In view of the heterogeneity and volatility of business functions and their interrelationships, as well as the unreconciled goals of the various departments of an enterprise, a data warehouse (DW) must fulfill certain requirements in order to be able to adequately support the decision-making, planning and coordination processes in a given business system. It is, therefore, necessary for the data warehouse to be clearly oriented towards the business system's needs, in the sense that it can unambiguously retrieve the state of the system at a certain point or period in time. In addition a data warehouse should also encompass the sequence of the states of the system, spanning an interval of time, in order to, on the one hand, enable the analysis and comparison of past states, while, on the other hand, allow for predicting and planning future states of the system. Finally, taking into account the heterogeneity of data sources, both from the technological as well as the business aspect, data warehouses must have the capability of data integration, wherein the integration needs to be accomplished in such a way that the gathered data is not significantly altered, i.e. without changing the semantics of the imported data.

A data warehouse can, thus, be defined as a model of a concrete business system representing a set of all of the states of that system during a given interval of time. The constant changes (organizational, legislative, functional, etc.) that a business system faces also reflect on the supporting data warehouse. Hence, one of the main issues, related to DW development and maintenance, is the inconsistency, between the actual system and its supporting data warehouse, which increases over time. Overcoming this discrepancy requires a flexible DW data model i.e. a data model which could be easily adaptable to the frequent changes in the business system.

An additional issue in the field of DW development is the absence of a standardized model for representing the structure of a data warehouse (i.e. a standardized DW data model). Existing approaches propose that the data should be organized in compliance with the third normal form (3NF) [1] or the multi-dimensional model [2]. Both approaches exhibit some limitations related to the difficulty in maintaining the data warehouse when the structure of the data sources changes. On the other hand both approaches are standardized by means of corresponding metamodels defined in the Common Warehouse Metamodel (CWM) [3]. Two additional approaches, aimed at addressing these limitations, have emerged in recent years, namely the Anchor model [4], based on data that has been normalized into the sixth normal form (6NF), and the Data

Vault model [5] which can (but need not) also store data normalized into the 6NF. It should be noted that the Anchor and Data Vault models are not standard extensions of CWM.

By identifying the strengths and weaknesses of each of these models it would be possible to establish the foundation for a new DW data model which would more adequately fulfill the posed requirements.

The remainder of the paper is organized as follows: first the fundamental concepts of the DW data models will be identified and elaborated. The main part of the paper is devoted to the analysis of the four most prominent DW data models which will be analyzed and compared on the basis of the following criteria: (1) built-in semantics, (2) resilience to change, (3) temporal aspects and (4) completeness and traceability of the data. The final Section gives a brief summary of the findings.

II. Data Warehouse Data Models – Fundamental Concepts

Data models are intellectual instruments for specifying the static characteristics of systems, i.e. for describing the objects, their attributes and their interrelationships in a stationary state [6]. As a data warehouse is defined as a model of a concrete business system representing a set of all of the states of that system during a given period of time, it is imperative that the underlying data model, be able to, not only support the specification of the system as it transitions through states, but also withstand changes in the business system or data sources.

At the highest level of abstraction, all of the described models are based on several fundamental concepts, as depicted in Table I.

TABLE I. Fundamental Concepts of the Data Models

	Object	***Relationship***	***Attribute***	***Identifier***
Normalized model	Relation	Foreign Key	Domain	Primary Key
Data Vault model	Hub	Link	Satellite	Business/Primary Key
Anchor Model	Anchor/Knot	Tie	Attribute	Primary Key
Dimensional model	Dimension	Fact	Attribute	Business/Primary Key

A. Relational model normalized into the 3NF

The *Relational Model* is a formal mathematical model based on set theory. The emergence of the *Relational model* was soon followed by the development of supporting database management systems (DBMS), which lead to it becoming the most widely used models for developing transactional systems.

The *Relational model* was envisaged by *Edgar Frank Codd*, with the aim of eliminating the problems related to the redundancy and inconsistency of data which were inherent to network and hierarchical models, as well as to overcome the complexity of those models. *Codd* established the main concepts in 1969 [7] and further expounded on this work a year later [8].

The central concept of this model is a *Relation* which is defined as a subset of the Cartesian product of a collection of sets $S_1, S_2, \ldots, S_n$. Each of these sets represents a *Domain* of the relation. *Domains* can be *predefined* – sets of data corresponding to the built-in data types supported by database management systems (DBMS) or *semantic* – user-defined sets built from predefined or other semantic domains and assigned a particular meaning [6]. The number of sets (i.e. domains over which the relation is defined) represents the *Degree* of the relation. The *Cardinality* of a relation represents the number of *n-tuples* of the relation. Given that the *Relation model* mandates that no two *n-tuples* can be the same, a domain (or a combination of domains), which uniquely identifies an *n-tuple* in the table, must exist, and it is called the *Key* of the relation [7]. In his subsequent work [8] *Codd* introduced the term *Primary Key*, denoting the actual key of a relation, i.e. the chosen identifier (since a relation can have more than one key uniquely identifying an *n-tuple*) and, in order to enable cross-referencing, the concept of a *Foreign Key* as an attribute (possibly complex) of relation *R* which represents the primary key of another relation *S*.

In [8] *Codd* also introduced the concept of *normalization* as a procedure for ensuring that the values of all the relation's attributes are always atomic, wherein such a relation is deemed to be in the *First normal form* (1NF). Namely, in his first paper [7], the *Relational model* permitted the "relation in a relation" concept (wherein the elements of a *Domain* are themselves *Relations*) which was, in a way, the predecessor of object-oriented and XML models. In the following years *Codd* introduced the *Second* (2NF), *Third* (3NF) and *Boyce-Codd* (BCNF) *normal forms* in order to completely eliminate the redundancy in relations. *Normalization* can, therefore, be regarded as a general technique for decomposing relations, into relations representing the fundamental objects of a system and their relationships, which guarantees minimal data redundancy and anomaly-free relations [6].

B. Data Vault Model

The *Data Vault (DV) Model* is an empirical model which has been in use for data warehouse development for more than two decades. It was conceived by *Dan Linstedt* with the aim of enabling the complete traceability of data, as well as greater scalability and adaptability in comparison with the data models utilized at that time [9]. *Bill Inmon*, who proposed the architecture for next generation data warehouses – DW 2.0 [10], recognized the *Data Vault model* as the optimal choice for the DW 2.0 architecture.

When conceiving the *Data Vault model*, *Linstedt's* starting point was the observation that business systems objects usually have an unchangeable unique identifier, while the remaining characteristics of the objects are volatile over time [11]. Hence, an object type is actually defined by two concepts: an entity (with its unique identifier) and its descriptive attributes.

The fundamental concept of a *DV model* is the *Hub* [12] which represents a real-world or abstract object of a business system that can be uniquely identified. The identifier is actually

the *Business Key* of the object that is stable over time, or has a very low propensity relative to change.

The structure of a *Hub*, which is subject to change, is defined through one or more *Satellites*. A *Satellite* [12] is a domain (possibly complex) representing one or more descriptive properties of the *Hub*. The properties of a *Hub* can be split into *Satellites* on the basis of a number of different criteria such as: the source system, the rate of change, organizational unit, type of information, etc.

Hubs may be related through system events, structural or other relationships, which is represented by the *Link* concept [12]. *Links* can also have additional descriptive characteristics – *Satellites*. The empirical *Data Vault model* permits *Link-Link* connections. In addition, the *Link* concept allows for establishing an *n-ary* relationship of *Hubs*.

All of the core concepts implicitly contain some additional metadata such as:

- *Surrogate key* – which uniquely identifies the *Hub*, *Link* or *Satellite* object.
- Record source – an attribute recording the source from which the data was originally loaded.
- *Audit identifier* – a link to a log table which stores audit information such as: the loading date and time, the time taken to complete the loading, and other ETL process-related information.

In addition, a *Satellite* also contains at least two extra metadata elements which define the validity period of the attribute values recorded in the *Satellite*:

- *Load Date* – representing the date and time when the attribute values were loaded into the data warehouse (or recorded in the *Satellite*) and
- *Load End Date* – representing the date and time when the values of an attribute were overwritten by new values.

It should be emphasized that a concrete *Data Vault model* is solely a physical model which is created by applying a set of simple rules through which the concepts of a given source model (e.g. relational) are transformed into the concepts of the relational target model. For instance, the *Foreign Key* concept of the *Relational Model* will always be mapped to a *Link* concept, which provides the necessary resilience to future changes in the structure of the source, if, for example, the cardinality of the relationship between two concepts changes so that the original relationship becomes an aggregation (i.e. a relationship which has its own additional properties).

C. Anchor Model

Anchor modeling was first introduced in 2004 as an empirical model (and method) for data warehouse development [4] and was subsequently formalized in 2010 [13]. The primary reason for the introduction of this model was to provide an easily extensible DW data model and to support agile DW development methodologies. At the physical level the *Anchor model* is "highly normalized" i.e. it is normalized into the 6NF, which is accomplished by implementing each attribute as a separate table (associated with the table representing the object type via a foreign key). It should be emphasized that the *Anchor model* is the same at both the logical and the physical level, i.e. the transformation between the two models is defined as a 1:1 mapping.

The *Anchor model* contains four main concepts which can be semantically profiled, using predefined sub-types, to give a more detailed description of their role in a system (e.g. is the concept time-invariant or is it necessary to maintain a history of its states).

The core concept is the *Anchor* which represents a set of business system objects. The *Knot* concept is similar to the *Anchor* concept but with the key distinction that it represents a set of objects which do not change over time. Furthermore, *Knot* objects can be shared by multiple instance of an *Anchor*. The *Attribute* concepts are used for describing the structure of an *Anchor*. Several sub-types of *Attributes* are provided:

- *Static Attributes* represent those properties of an *Anchor* for which it is not necessary to maintain a history of changes.
- *Historized Attributes* are used for maintaining the history of changes in the values of an attribute.
- *Knotted Static Attributes* represent *Anchor-Knot* relationships which are stable over time.
- *Knotted Historized Attributes* represent *Anchor-Knot* relationships which can change over time.

In order to define a relationship between the concepts, the *Tie* concept is introduced, along with the *Role* concept which represents a mapping between two sets of objects. Since both *Anchors* and *Knots* represent sets of objects, two types of *Roles* are provided: *Anchor Roles* and *Knot Roles*. Thus a *Tie* represents an association between two or more *Anchors* (or *Knots*) which is composed of at least two *Roles*. As is the case with *Attributes*, four types of *Ties* are introduced to enable a more precise definition of an association:

- *Static Ties* are sets of at least two *Anchor Roles*.
- *Historized Ties* are sets of at least two *Anchor Roles* and a temporal variable.
- *Knotted Static Ties* are sets of at least two *Anchor Roles* and one or more *Knot Roles*.
- *Knotted Historized Ties* are sets of at least two *Anchor Roles*, one or more *Knot Roles* and a temporal variable.

D. Dimensional Model

The *Dimensional model* is probably the most widely used DW data model. The biggest advantage of the *Dimensional model* is that it facilitates data retrieval and analysis, given that business users can easily create queries over the DW. In addition, it provides high-performance query processing.

The *Dimensional model* is the result of academic research endeavors, conducted during the 1960s, aimed at simplifying the presentation of analytical data [14]. The leading proponent

of this model is *Ralph Kimball* who defined a complete methodology for data warehouse development based on the Dimensional model [2].

The two main concepts of the dimensional model are *Dimensions* and *Facts*. A *Fact* typically represents a set of events taking place within a business system while *Dimensions* represent descriptive information about the Facts [15]. A dimension can be denormalized, i.e. when the entire hierarchy of the dimension is redundantly stored in a single table (the *Star Schema*), or, to keep the redundancy of the dimension minimal, it can be organized into a normalized sub-model (the *Snowflake Schema*) whereby the tables belonging to the hierarchy of one dimension may also be exploited in other dimensions. When developing a data warehouse it is customary to exclusively use only one of these two approaches.

Kimball proposes *Denormalized Dimensions* [14] due to performance considerations, and also introduces the concept of *Conformed Dimensions* whose main characteristic is that they span the entire data warehouse and are shared by multiple *Data Marts*.

Furthermore, according to *Kimball*, each *Data Mart* (which is based on a *Fact* with its associated *Dimensions*) is defined in accordance with a single business unit and the conformed dimensions. In such a way a matrix of data marts is built, eliminating the possible redundancy across a given dimension in the scope of a single data warehouse.

As changes in *Dimensions* occur less frequently than changes in *Facts*, *Kimball* defined three types of *Slowly Changing Dimensions – SCD* in [14] (excluding *SCD type 0* which represents a special case in which changes are never recorded), and added four additional types in [12]:

- SCD type 1 – the old value is overwritten by the new value, thus the history of the previous values of an attribute is not preserved.
- SCD type 2 – the new value is added to the *Dimension*, which requires the use of a surrogate key since the object identifier will no longer be unique. Each *Dimension*, maintained in this manner, should have three additional attributes: *Effective Date*, *Expiration Date* and *Current Row Indicator*.
- SCD type 3 – in order to preserve the old value, a new attribute, storing the old value (i.e. the alternate reality), is added to the *Dimension*. Users can then group the data on the basis of, either the current or the alternative value.
- SCD type 4 – are used when a subset of attributes changes frequently (or is used for analytics). Such a subset is split off into a mini-dimension. The newly formed dimension will have its own surrogate key, wherein the primary keys of both dimensions are referenced by the corresponding *Fact*.
- SCD type 5 – represents a combination of types 4 and 1, as the mini-dimensions and the base dimension are related via the primary key of the mini-dimension. In the presentation layer the dimension and its corresponding mini-dimension are represented as a single table.
- SCD type 6 – represents a combination of types 1, 2 and 3, as a new *n-tuple* (along with its validity period) is added for each attribute that changes (type 2) together with an additional column which stores the currently valid value of the attribute (type 3). Furthermore, all of the existing *n-tuples* will be updated with the currently valid value (type 1).
- SCD type 7 – is adequate for *Dimensions* which contain a large number of attributes, so, in addition to the surrogate key, the *Fact* will also record the business key, i.e. the historically durable (i.e. unchangeable) object identifier. Hence a *Fact* can be analyzed both in a type 1 and a type 2 fashion.

III. Comparative Analysis of the Data Models

A. Built-in Semantics

The main point of difference among the four models is the level of built-in semantics they provide.

The *Normalized model* does not presume any semantic constraints and, as such is extremely general, as the development of any given business model is based on mappings between sets. Furthermore, it does not provide any implicit concepts which would enable maintaining the history of changes of an object nor the values of its attributes.

The *Data Vault model* assumes that business objects have a stable identifier and somewhat alters the structure of the source by allowing for an object to be arbitrarily structured (i.e. its structure can be split into several *Satellites*). The *Data Vault model* is suited for tracing changes in the values of attributes, except for the *Hub* identifier (i.e. the *Business Key*).

The *Anchor model* is highly normalized. It provides two concepts, the *Anchor* and the *Knot*, for representing business objects. In addition, it enables the tracing of the history of all concepts, save for *Knots*.

The *Dimensional model* is based on the events that take place within a business system and the *Dimensions* which define them. Furthermore, it is possible to define numerical properties for expressing the quantitative aspects of the events. The tracking of the history of changes is based on the complex rules pertaining to changes in dimensions, as elaborated in the previous section.

All of the models provide a single representation of an object (or entity) except for the *Anchor model* in which concepts of a concrete system can be represented by *Anchor* or *Knot* concepts. The main difference is that the *Normalized*, *Data Vault* and *Anchor* models are normalized, while the *Dimensional* model is denormalized.

A relationship between objects is represented through the *Foreign Key* concept in the *Normalized model*, which establishes a “tight relationship”. The *Data Vault*, *Anchor* and *Dimensional* models define the relationship between objects through the *Link*, *Tie* and *Fact* concepts, respectively, wherein

the relationship is realized as a table which stores the primary keys of the objects in the relationship (which need not necessarily be binary). In addition, in *Dimensional model* it is customary to store additional derived or aggregated attributes in the structure of a *Fact*.

With regard to attributes, it should be noted that the *Data Vault* and *Anchor* models separate the structure of an object from the object itself, by using *Satellites* and *Attributes*, respectively, which reference the object via a foreign key. The difference between these two models is that, in the *Anchor model* a separate table is created for each attribute, while, in the *Data Vault model*, the attributes can, as previously mentioned, be grouped according to various criteria into multiple *Satellites* of one object. The *Normalized* and *Dimensional* models keep the attributes within the structure of the object.

In all of the models each concept, which is used for representing an object type, has an identifier. The *Data Vault model* assumes that the identifier is the actual business key, i.e. the identifier that uniquely distinguishes the object in the given business system. Somewhat similarly, the *Dimensional model*, when using *Type 2 SCDs*, also expects the existence of a business key on the basis of which the dimension values will be grouped.

B. Resilience to Change

As previously mentioned, the constant changes (organizational, legislative, functional, etc.) that a business system faces also reflect on the supporting data warehouse. Hence, one of the main issues, related to DW development and maintenance, is the inconsistency between the system and the data warehouse, which increases over time. Consequently, one of the primary requirements related to data warehouse development is to provide the ability of absorbing changes in the structure of the data sources, without changing the structure of the underlying data model, in order to facilitate future maintenance and extensions of the data warehouse.

In this section the data models will be analyzed from the viewpoint of their adaptability and extensibility. More precisely, the evaluation of this aspect will be focused on establishing whether changes in the structure of the data sources can be handled simply through the addition of new concepts, without requiring modifications of the DW physical layer.

The *Normalized model* has proven to be the optimal data model for transactional systems since, on the one hand, it guarantees minimal data redundancy and, on the other, decomposes the structure of the system down to the level of its fundamental objects. However, in the data warehouse domain it demonstrates certain weaknesses, as data warehouse requirements differ considerably from transactional system requirements. The main weakness of the *Normalized model* lies in the fact that the attributes and relationships are built into the structure of the system's objects, which leads to a number of maintenance-related issues. Namely, any change in the structure of the source (attributes or relationships) will require changes in the structure of the *Normalized model*, as depicted in Fig. 1, where the cardinality of the relationship between the ***Employee*** and ***Organizational Unit*** concepts is adjusted in order to allow for an ***Employee*** to be assigned to more than one ***Organizational Unit***. In the depicted example it is necessary to create the ***Position*** table, transfer the existing data (pertaining to the relationship between the ***Employee*** and ***Organizational Unit*** entities) into the ***Position*** table, and delete the foreign key, referencing the ***Organizational Unit***, from the ***Employee*** table.

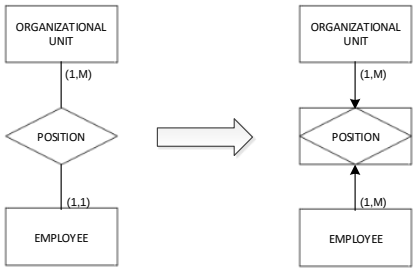

Fig. 1. Cardinality change in the *Normalized model*

In addition, the *Normalized model* has some shortcomings when it comes to the *reconciliation* of source models since, due to the potential semantic heterogeneity of the source models, it may not be possible to design a single reconciled model, as demonstrated by *Golfarelli* in [15].

The *Data Vault model* exhibits much greater flexibility with regard to changes in the structure of the sources. Its flexibility derives from the underlying language, as the structure of an object is decoupled from the object itself, and placed in a physically separate concept – a *Satellite*. Furthermore, every relationship (i.e. *Link*), regardless of its cardinality, is always created as a table representing an aggregation of the related objects.

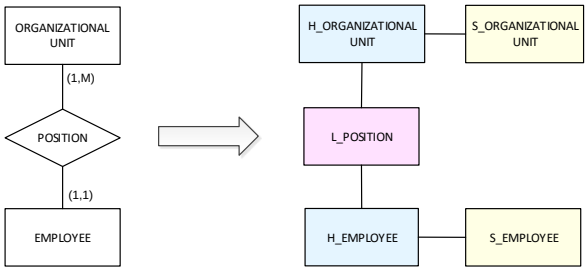

Fig. 2. A (1:M) relationship to the *Data Vault model*

As depicted in Fig. 2, in the initial structure of the source system an ***Employee*** could be assigned to one and only one ***Organizational Unit***. Consequently, the corresponding *DV model* contains the ***H_Employee*** and ***H_OrganizationalUnit*** *Hubs* with the associated *Satellites*. The *Hubs* are related through the ***L_Position*** *Link*. If the structure of the source changes to allow for an ***Employee*** to be assigned to more than one ***Organizational Unit***, the target *DV model* will not undergo any changes, as shown in Fig. 3.

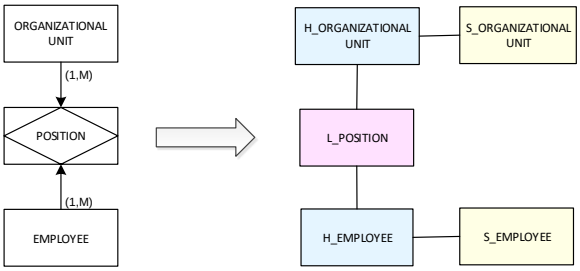

Fig. 3. An (M:M) relationship to the *Data Vault model*

Furthermore, if the ***Position*** concept is enhanced with an additional property, e.g. ***StartDate***, the *DV model* once again proves its flexibility, given that it is only necessary to extend (not modify) the existing model by adding an ***S_Position*** *Satellite* (containing the introduced property) to the ***L_Position*** *Link*, as illustrated in Fig. 4.

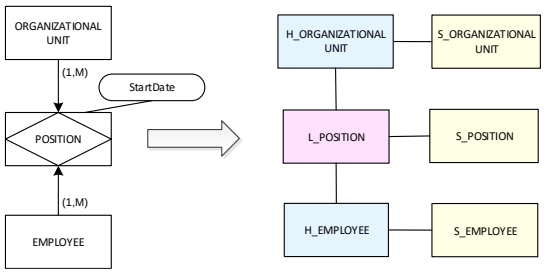

Fig. 4. The addition of attributes to a *Data Vault model*

Moreover, further modifications, e.g. relating the ***Postion*** concept to other concepts of the source model, will not require changing the existing structure of the *DV model*, given that it allows for two *Links* to be related.

However, it should be noted that, even though in the previous examples the *DV model* did not undergo any changes at the physical level (i.e. it was only extended), the semantics of the source system have implicitly been changed by representing a strong entity (i.e. the ***Position*** aggregation) as a *Link* (i.e. event or relationship) in the *DV model*.

An additional drawback of the *DV model* is the fact that, the transformation process is irreversible, i.e. obtaining the structure of the original source model from an existing DV model is not possible. Because the *DV model* is semantically much poorer than the source models, certain information contained in the source models will inherently be lost in the transformation. For instance, in the previous examples (Figs. 2 and 3) it would be impossible to determine into which of the two source models the *DV model* would be reversibly transformed.

In keeping with the same example, the adaptability of the *Anchor model*, to changes in the structure of the sources, will be examined. All of the models have been created using the original Anchor modeling tool [16].

As depicted in Fig. 5, the initial model consists of two *Anchors*: ***EM_Employee*** and ***OU_OrganizationalUnit***. The *Tie* between the two *Anchors* (***OU_engages_EM_assignedTo***), wherein an ***Employee*** can be assigned to at most one ***Organizational Unit***, is implicitly named on the basis of the roles of the related *Anchors*. Following the transformation the *Tie* becomes a table whose primary key is the identifier of the concept that participates in the relationship with a maximum cardinality of 1.

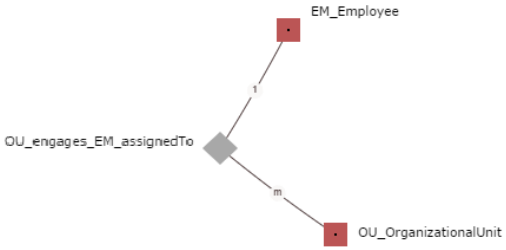

Fig. 5. A (1:M) relationship to the *Anchor model*

If the cardinality changes, in order to allow for an ***Employee*** to hold more than one ***Position*** (as depicted in Fig. 6), the only alteration of the model is extension of the primary key of the ***Organizational Unit*** with the primary key of the table representing the *Tie*.

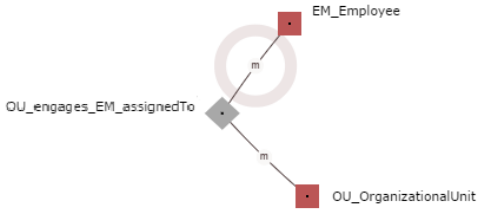

Fig. 6. An (M:M) relationship to the *Anchor model*

However, if the structure of the source system changes so that a *StartDate* is added to the relationship (***OU_engages_EM_assignedTo***), the *Anchor model* does not support this feature. Namely, the transformation of a relationship into an object, or the addition of structural features to a relationship, is not allowed. Instead, the *Tie* can reference a *Knot* representing a set of all of the start dates pertaining to the given ***Position***, as depicted in Fig. 7.

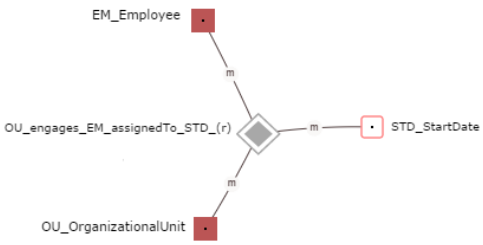

Fig. 7. The addition of attributes to a relationship in the *Anchor model*

Consequently, at the physical level the table, representing the *Tie*, is modified by including an additional attribute (the primary key of the *Knot*) which becomes part of the composite key of the *Tie*.

The *Dimensional model* exhibits the same weaknesses as the *Normalized model*, as the addition of new attributes in the source model requires changing the structure of corresponding data warehouse *Dimension*. Yet the relationships in the

Dimensional model are more stable, given that this model is process-oriented, i.e. it is designed on the basis of the core business processes of an enterprise. Nonetheless, the introduction of additional concepts in the source model (which become *Dimensions* in the data warehouse model) also requires the modification of the corresponding *Fact* in order to include a foreign key referencing the newly formed *Dimension*.

C. Temporal Aspects

By defining a data warehouse as a collection of *Time-Variant* data [1], *Inmon* established one of the main requirements for data warehouses, namely that they must enable the tracking of past, current and future states of objects and events, as well as the points in time at which the changes occurred [17]. The notion that a transactional system should support more than one time dimension was first put forward in [18] when *Snodgrass* proposed the *Bitemporal* concept, splitting the time aspect into two dimensions: *Valid time* and *Transaction Time*.

Nowadays, as stated in [19], it is customary for the time aspect to be treated through three dimensions:

- *Valid Time* – the period of time during which a concept (e.g. an object or event) can be considered to be valid with respect to the real world.
- *Transaction Time* – the time when data was stored in the data warehouse.
- *User-Defined Time* is used for representing values of temporal business attributes. As such it has no bearing on the analysis of the temporal aspects of DW data models.

As stated in [20] it can be said that a data model supports the modeling of temporal aspects if it provides built-in mechanisms for capturing both the *Valid Time* and the *Transaction Time*.

The aim of this section is to explore whether the data models support the tracking of the entire history of business system objects, and if so, through which mechanisms.

The *Normalized model* does not provide built-in mechanisms for capturing temporal aspects, rather it is left to the data warehouse designer to decide which temporal concepts will be included and how they will be modeled [20].

The *Data Vault model* implicitly incorporates the *Transaction Time* into each concept (*Hub*, *Link* and *Satellite*) via the *Load Date* and *Load End Date* concepts. As this model assumes that the structure of objects will change over time, *Satellites* can be used to capture the changes in values which occur during the life-cycle of an object. Hence the *Valid Time* concept can only be applied to the structure of an object, not the object itself nor its relationships. Consequently, and in light of the underlying premise that business keys are immutable, the *Data Vault model* does not provide a built-in mechanism for tracking the *Transaction Time* for business keys. Nevertheless, such cases can be handled by using the *Same-As Link* whereby two *Hubs*, with different business keys, are asserted to be identical [5].

The *Anchor model* provides the *Historized* option for representing time-variant *Attributes* and *Ties* which, however, only allows for capturing their *Valid Time*. When this option is used the structure of each of the chosen concepts is supplemented with a *ValidFrom* attribute [21] which captures the beginning of the validity period of the value (e.g. the official *Anchor modeling* tool generates a *ChangedAt* attribute for capturing the *Valid Time*). It is implicitly assumed that the end of the validity period of a previous value corresponds to the beginning of the validity period of the new value. However, such an assumption may not always be valid. In addition, if the *Static* option is used when creating the model, the possibility of tracking the history of value changes for such *Attributes*, or *Ties*, will be lost. The stated reason is that this option is to be used for values which are considered to be stable, e.g. the *Date of Birth*. Yet, this does not take into account the possibility that the value of an attribute may be incorrectly recorded in the source, or the possibility that several sources, storing different values of the same attribute, may exist. *Anchors* and *Knots* are considered to be immutable, which may not always be the case.

The *Transaction Time* (i.e. *RecordedAt*) is not incorporated into the structure of the concepts representing business system objects, but is stored as metadata in separate tables instead [13].

The *Dimensional model* enables the tracking of *Valid Time* through the various *SCD types* it offers (excluding *Type 1* and *Type 3 SCDs* which do not allow for tracking the *Valid Time* of an object). However, one of the main issues, related to tracking changes in dimensions, is the choice of the *SCD Type*, as switching to a different *SCD Type* at a later point in time inevitably leads to changes in the structure of the dimension. In addition, if the *Type 2 SCD* is chosen, changes is in the *Business Key* of an object cannot be tracked, as it is the basis for tracking changes in the other attributes.

In the *Dimensional model* the *Transaction Time* is not incorporated into the structure of the concepts representing business objects, instead it is recorded in separate log tables.

D. Completeness and Traceability of Data

In accordance with *Inmon's* definition [1] the second crucial characteristic of data warehouses is the *Non-Volatility* of data, which assumes that all of the data that has been integrated into a data warehouse must be retained in the data warehouse, unmodified, in order to ensure that a given query, executed at any point in time, will always produce the same, consistent result [22].

The completeness and traceability requirements are in direct collision with the *Single Version of the Truth* concept which assumes that the data, that is to be stored in the data warehouse, is prepared and filtered in advance. The *Single Version of the Truth* is, thus, the result of integrating data from multiple sources with the aim of providing a uniform basis for analysis and avoiding redundancy [23]. However, in order to achieve this aim, it is customary when designing a data warehouse, to choose, out of all of the data relating to a single concept and extracted from various sources, which one will represent the "truth" and, as such, be stored in the data warehouse. All other occurrences (i.e. those which do not

conform to the "truth") are discarded and will not be loaded into the data warehouse. As a result, given that the data warehouse stores only part of the source data, it is impossible to conduct analyses on all of the values which exist in the source systems.

Consequently, the *Single Version of the Facts* concept emerged which assumes that the data warehouse stores *all* of the source data [24], thereby making it possible to provide different views for different users according to their specific needs. In essence, this approach promotes an ELT (Extract-Load-Transform) process in which the transformation of data takes place after the data warehouse has been loaded, as opposed to the traditional ETL (Extract-Transform-Load) approach in which the data is transformed prior to its being loaded into the data warehouse (i.e. on its way from the staging area to the data warehouse. As a result, the data warehouse will store all of the available data from all of the sources spanning the entire history of the business system. Therefore, the analytical processing of the data will be performed on-demand depending on the business user's needs [24].

The aim of this section is to explore whether the data models provide mechanisms for, on the one hand, ensuring the completeness of the data which is transferred from the sources and, on the other, for tracing the stored data back to the sources from which it originated.

The *Normalized model*, as was the case with the temporal aspect, does not implicitly provide concepts to support the traceability of data or the recording of data originating from multiple sources. Traceability can, for example, be accomplished by recording additional metadata for each *n-tuple*, if the *Single Version of the Truth* approach is adopted. However, if the *Single Version of the Facts* approach is adopted the *Normalized model* demonstrates certain weaknesses. Namely, storing data from several sources in the same model would require storing one *n-tuple* per source in which the same business concept is present. This leads to the issue of relating these *n-tuples*, i.e. it requires using additional concepts for relating *n-tuples* representing the same business object. Furthermore, it introduces redundancy (as the *n-tuples* representing the same business object contain a number of attributes with the same values) thereby eliminating one of the good traits of the *Normalized model*.

The *Data Vault model* supports the traceability of data by requiring that the source, from which the *Hub* was initially loaded, be recorded. Likewise, the *Satellite* and *Link* concepts include built-in attributes for recording the source from which the loaded values originated (i.e. the *RecordSource* attribute) [11]. However, the main shortcoming of this model lies in the fact that the structure of a *Hub* contains the business key, the value of which is assumed to be stable or rarely changes. Moreover, the structure of the business key might also change. These situations are resolved by introducing a new *Hub* which will be related to the original *Hub* via a "*Same-As*" *Link*. Consequently, more than one *Hub*, with the same attributes and the same relationships with other model elements, will exist. In addition, the *Data Vault model* enables tracing data from multiple sources, by recording the data source, along with the value of an attribute, in a *Satellite*.

The *Anchor model* supports the traceability of data, via the *metadata* concept, as all time-variant concepts (thus excluding *Anchors* and *Knots*) can reference the metadata capturing the *source of the data*. In effect, this also means that it is not possible to record that, for a single object two identical values, originating from two different sources, are both valid at the same point it time. The reason for this is that, at the implementation level, there exists a *Unique Constraint* over a group of attributes: the identifier, the *Valid Time* and the value of the attribute.

The *Dimensional model* does not provide built-in mechanisms for tracing the stored data back to the sources. Furthermore, it does not allow for recording multiple values (originating from multiple sources) for a single object, during the same period of time.

IV. Conclusion

In light of the previous discussion it can be concluded that none of the analyzed models completely fulfill all of the necessary requirements pertaining to data warehouse data models (as summarized in Table II). In brief, the *Normalized model* is not immune to any of the considered problems. The *Data Vault model* does not explicitly incorporate the *Valid Time* concept. Furthermore, both the *Data Vault model* and the *Anchor model* have limitations when it comes to complex modifications of the structure of the data sources. In addition, the *Anchor model* has some shortcomings the due to its underlying *Single Version of the Truth* concept. Finally, even though the *Dimensional model* is probably the most suitable model for preparing and analyzing data, from the DW data model point of view its main drawbacks are related to the fact that it is not resilient to change, along with the fact that it does not allow for tracing multiple values of the same data coming from different sources.

TABLE II. Comparison of DW Data Models

	Normalized	***Data Vault***	***Anchor***	***Dimensional***
Built-in semantics	No	Yes	Yes	Yes
Resilience to change	No	Partially	Partially	No
Time aspect	No	Partially	Partially	Yes
Completeness	No	Yes	Partially	No
Traceability	No	Yes	Yes	No

Future work, which is currently in progress, is aimed at designing a physical data warehouse model that fulfills all of the posed requirements while building on the best features of the analyzed models.

References

[1] W. Inmon, Building the Data Warehouse. Wiley, 2002.

[2] R. Kimball, L. Reeves, M. Ross and W. Thornthwaite, The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing, and Deploying Data Warehouses. Wiley, 1998.

[3] Object Management Group, Common Warehouse Metamodel, available at: http://www.omg.org/cgi-bin/doc?formal/03-03-02.pdf, 2002.

[4] O. Regardt, L. Rönnbäck, M. Bergholtz, P. Johannesson and P. Wohed, "Anchor Modeling: An Agile Modeling Technique Using the Sixth Normal Form for Structurally and Temporally Evolving Data", in Proc. of ER'09 (Brazil), [LNCS, vol. 5829, no.1, pp. 234-250], 2009

[5] D. Linstedt, "Data Vault Modeling Specification v1.0.9. ", available at: http://danlinstedt.com/datavaultcat/standards/dv-modeling-specification-v1-0-8/, 2010.

[6] B. Lazarević, Z. Marjanović, N. Aničić and S. Babarogić. Baze podataka. Fakultet organizacionih nauka, 2010 (*Textbook in Serbian*).

[7] E. F. Codd, "Derivability, Redundancy and Consistency of Relations Stored in Large Data Banks", IBM Research Report, San Jose, California, 1969.

[8] E. F. Codd, "A Relational Model of Data for Large Shared Data Banks", in Communications of the ACM, Volume 13 Issue 6, pp. 377-387, June 1970.

[9] D. Linstedt, "Data Vault Series 1 - Data Vault Overview", available at: http://www.tdan.com/view-articles/5054/, 2002.

[10] B. Inmon, "Data Warehousing 2.0 - Modeling and Metadata Strategies for Next Generation Architectures", White Paper, available at: http://www.devgear.co.kr/pdf/bill-inmon-data-warehousing-2-0-whitepaper.pdf, 2010.

[11] D. Linstedt, Super Charge Your Data Warehouse: Invaluable Data Modeling Rules to Implement Your Data Vault. CreateSpace Independent Publishing Platform, 2011.

[12] D. Linstedt, "Data Vault Series 2 - Data Vault Components", available at: http://www.tdan.com/view-articles/5155/, 2003.

[13] L. Ronnback, O. Regardt, M. Bergholtz, P. Johannesson and P. Wohed, "Anchor modeling — Agile information modeling in evolving data environments", in Data & Knowledge Engineering, Volume 69, Issue 12, pp. 1229-1253, 2010.

[14] R. Kimball and M. Ross, The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling. Wiley, 3rd ed., 2013.

[15] M. Golfarelli and S. Rizzi, Data Warehouse Design – Modern Principles and Methodologies. McGraw-Hill, 2009.

[16] Anchor Modeling Tool, available at: http://www.anchormodeling.com/modeler

[17] E. Malinowski and E. Zimanyi, Advanced Data Warehouse Design – From Conventional to Spatial and Temporal Applications. Springer-Verlag Berlin Heidelberg, 2008.

[18] R. T. Snodgrass and I. Ahn, "Temporal Databases", in IEEE Computer 19(9), pp. 35–42, September 1986.

[19] C. Jensen, J. Clifford, R. Elmasri, S. K. Gadia, P. J. Hayes and S. Jajodia, "A Consensus Glossary of Temporal Database Concepts", in SIGMOD Record, 23(1), pp. 52–64, 1994.

[20] H. Gregersen and J.S. Jensen, "Temporal Entity-Relationship models a survey", in IEEE Transactions on Knowledge and Data Engineering, vol. 11, pp. 464-497, 1999.

[21] L. Ronnback, O. Regardt, M. Bergholtz, P. Johannesson and P. Wohed, "From Anchor Model to Relational Database", available at: http://www.anchormodeling.com/wp-content/uploads/2010/09/AM-RDB.pdf

[22] M. Golfarelli, J. Lechtenbörger, S. Rizzi and G. Vossen, "Schema versioning in data warehouses: Enabling cross-version querying via schema augmentation". In Data & Knowledge Engineering, 59(2), pp. 435-459, 2006.

[23] B. Inmon, "The Single Version of The Truth", Business Intelligence Network (Powell Media LLC), available at: http://www.b-eye-network.com/view/282, 2004.

[24] R. Damhof, "The next generation EDW", available at: prudenza.typepad.com/files/damhof_dbm0508_eng-1.pdf, 2008