The Conceptual Integration Modeling Framework: Abstracting from the Multidimensional Model

Flavio Rizzolo[1⋆], **Iluju Kiringa**[2], **Rachel Pottinger**[3], and **Kwok Wong**[4]

[1] University of Ottawa. frizzolo@site.uottawa.ca
[2] University of Ottawa. kiringa@site.uottawa.ca
[3] University of British Columbia. rap@cs.ubc.ca
[4] University of Ottawa. kwong080@uottawa.ca

Abstract. Data warehouses are overwhelmingly built through a bottom-up process, which starts with the identification of sources, continues with the extraction and transformation of data from these sources, and then loads the data into a set of data marts according to desired multidimensional relational schemas. End user business intelligence tools are added on top of the materialized multidimensional schemas to drive decision making in an organization. Unfortunately, this bottom-up approach is costly both in terms of the skilled users needed and the sheer size of the warehouses. This paper proposes a top-down framework in which data warehousing is driven by a conceptual model. The framework offers both design time and run time environments. At design time, a business user first uses the conceptual modeling language as a multidimensional object model to specify what business information is needed; then she maps the conceptual model to a pre-existing logical multidimensional representation. At run time, a system will transform the user conceptual model together with the mappings into views over the logical multidimensional representation. We focus on how the user can conceptually abstract from an existing data warehouse, and on how this conceptual model can be mapped to the logical multidimensional representation. We also give an indication of what query language is used over the conceptual model. Finally, we argue that our framework is a step along the way to allowing automatic generation of the data warehouse.

1 Introduction

Organizations can now access vast amounts of data drawn from a variety of data sources including operational databases (e.g., legacy product and service databases, financial databases, human resource or customer databases), business documents, spreadsheets, and totally or partially structured web documents. Data warehouses [19, 13] provide a physical, integrated repository of various heterogeneous data sources. At design time, a data warehouse schema is constructed based on the local schemas of the data sources to be integrated. At run time, the warehousing process first identifies required sources. It then successively transforms the data extracted from the sources and materializes them into a warehouse or a set of data marts according to a multidimensional relational

⋆ Also affiliated with Carleton University.

CIM Visual Model (CVM)	**CIM Data Model (CDM)**
CVL : **C**onceptual **V**isual **L**anguage	**CDL** : **C**onceptual **D**ata **L**anguage
MVL : **M**apping **V**isual **L**anguage	**MDL** : **M**apping **D**ata **L**anguage
SVL : **S**tore **V**isual **L**anguage	**SDL** : **S**tore **D**ata **L**anguage

Fig. 1. The landscape of models of the CIM Framework

schema. Finally, applications such as cubes and various mining and modeling tools on top of the warehouse generate business intelligence for running the organization.

Both the design and run-time processes are bottom-up in the sense that the warehouse's schema and data are driven only by information to be integrated in the data warehouse, but not by the needs of users and their Business Intelligence (BI) applications. In the existing bottom-up methodologies, information to be integrated into a data warehouse is the focal point of attention, and the users' requirements are taken into account only as a software engineering step for coming up with the conceptual model of the warehouse, not as a schema that can be instantiated and queried. This means that any conceptual model used is almost exclusively for the purpose of designing the data warehouse, never as an implementable data model against which queries are posed.

Existing query and report tools over data warehouses have tremendously increased productivity and enabled the management of organization performance with an ease never seen before the advent of these tools. Using these tools to generate reports is relatively easy, and these reports can also be easily generated on framework. However, the difficult and open challenges lie in issues such as expressing and executing the mappings on one hand, as well as evolving the models on the other; this paper focuses on the former set of challenges.

In contrast to the existing bottom-up techniques, we introduce an approach whose overall goal is to offer a framework for a top-down, requirements-driven data warehouse construction. This paper describes the Conceptual Integration Modeling (CIM) Framework, an approach that allows users to raise the level of data integration abstraction by specifying their needs at a high conceptual level which is then implemented in a multidimensional platform. The CIM Framework offers both design time and run time environments that implement a conceptual modeling language. This language has the two facets shown in Figure 1. The first facet — the CIM Visual Model (CVM) — comprises three distinct parts: (1) an extended Entity Relationship Model [7] inspired by MultiDim [21] and StarER [30], which are two preexisting conceptual models for data warehouse design; (2) a visual representation of the (relational) multidimensional model; and (3) a visual mapping language for translating the conceptual model into the multidimensional model. The second facet — the CIM Data Model (CDM) — is a set of XML-based, system-specific models that correspond one-to-one to the CVM constructs. The CIM Framework comprises the CDM and CVM as well as a query language for the extended Entity Relationship Model above.

Our top-down integration methodology is driven by a conceptual model: a business user specifies what business information is needed and in what form, and the system satisfies the request. Achieving this requires raising the level of abstraction significantly. Today, integration is primarily concerned with the characteristics of individual prod-

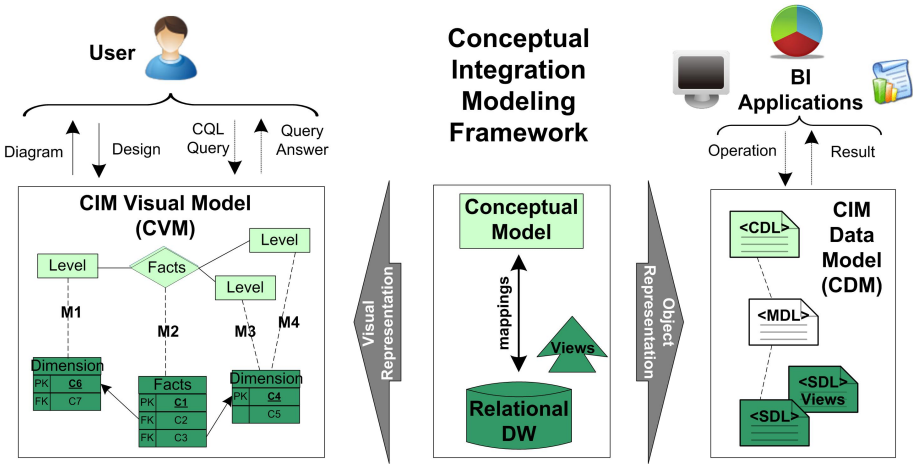

Fig. 2. The various data models that are created in the CIM Framework

ucts. We need to move to a world where users primarily focus on articulating their data needs in terms they understand. The conceptual model articulates the business objectives and requirements for the warehouse. This model can be used to both design the warehouse and to build (populate) the warehouse. As opposed to the current state-of-the-art static approach, the new paradigm we are introducing offers a dynamic methodology which generates the warehouse on-demand by using a given business conceptual model for driving the warehouse construction and the subsequent BI stages. As requirements evolve, so does the conceptual model together with the warehouse design and population mechanisms.

This paper focuses on one step of our vision: how the user can specify a conceptual model for the data warehouse, and then how the conceptual model can be logically implemented and used in real time. We build upon the aforementioned MultiDim and StarER to create the CVM, and the Entity Data Model (EDM) [4] to define the CDM. Similarly to how EDM includes a Conceptual Schema Definition Language (CSDL) for the conceptual level, a Storage Schema Definition Language (SSDL) for the logical level, and a Mapping Specification Language (MSL) to transform the conceptual level to the logical level [2], our work includes a Conceptual Data Language (CDL) for the multidimensional conceptual level, a Store Data Language (SDL) for the multidimensional (i.e., data warehouse) level, and a Mapping Data Language (MDL) to relate the two levels. These three models form a CIM Data Model (CDM) and are shown on the right hand side of Figure 1, which is explained in detail in Section 3.

As shown in Figure 2, there are several possible paths for the user to create and manipulate data warehouse models at design time. One path is that the user visually creates them using the CIM Visual Models (CVM), which are then translated into CDM specifications. Another path is that the user creates them using the CDM models, which then can be visualized by CVM models for possible further manipulation. In either case, users can query the CDL and CVL parts of the CDM and CVM models at run time using queries formulated in a Conceptual Query Language (CQL). A compiler translates the

MDL mappings between CDL and SDL specifications into views over SDL, so that the problem of processing CQL queries over CDL is reduced to a problem of processing queries using views. This, again, parallels the approach taken in [2], but with a much different set of algorithms due to the different settings.

Our contributions are the following:

- We describe the Conceptual Integration Modeling (CIM) Framework, which allows the top down creation of a data warehouse;
- We describe a complete framework of languages, both visual and data, to define conceptual and storage layers of the CIM Framework. While the conceptual visual language is not a major contribution (the CVL is a variant of MultiDim), the use of such a full suite and what it allows is. We describe the languages themselves in enough detail to make the contributions clear;
- We define the architecture for the CIM framework and describe how this architecture can handle all the necessary transformations;
- We describe future directions, including how to extend CIM to on-demand data warehouse creation.

The rest of the paper is organized as follows. Section 2 motivates the CIM Framework. Section 3 describes the data-centric CVM and CDM as depicted in Figure 2. Section 4 outlines the run time architecture that is under implementation for the CIM Framework described in this paper. Section 5 describes some details of how users can interact with the whole framework. We contrast our work with related work in Section 6. Finally, Section 7 concludes and describes the current state of the implementation as well as future work to create a variant of the data centered architecture in Figure 2 to allow on-demand generation of the needed parts of a data warehouse.

2 Motivation for the CIM Model

Almost every aspect of today's world, from business, to science, to digital social interactions, is drowning under a deluge of data. It is estimated that the amount of data produced in the world grows at an astounding annual rate of 60% [27]. The only way to cope with this deluge is to move toward a data-centered way of conducting human activities — a way that focuses on making sense of the data. One direct consequence of the steady trend toward a data-centered society is seen in the fact that application developers continuously deal with managing data, which is typically relational in format. This is mismatched with the developers' platforms, which are overwhelmingly object-oriented. Additionally, the object-oriented world traditionally works at the conceptual level, thus ensuring data independence from the relational level, which, in many organizations, is implemented on a variety of systems that use different representational flavors.

The divide between the conceptual and storage layer of data-centric applications requires developing mappings between representations to close the resulting representational gap. Commercial (e.g., the EDM Entity Framework [2, 4]) as well open source products (e.g., Hibernate [14]) have been proposed to bridge the representational gap. These mapping technologies are known as Object-Relational Mappings (ORM).

BI tools constitute another emerging class of data-centered applications that deal with higher orders of magnitude of data than ORM technologies. For example, Walmart operates 8,400 stores worldwide which handle more than 200 million transactions weekly [27]. On one hand, the immense flow of data that must be managed to gain insight into the retailer's business requires using enormous multidimensional data warehouses. On the other hand, mid-level and top executives want to make sense of the vast amount of data at their disposal in business terms they understand — without having to become ultra-sophisticated data analysts. We must bridge the gap between the world of those executives who make decisions based on high-level, business-oriented data representations and the world of multidimensional models. The CIM Framework is intended to bridge those worlds.

Many existing models (e.g., MDX and its data model ADO MD [26], Universes [15], Mondrian [23], and Framework Manager [31]) could have been used as a conceptual modeling language for multidimensional modeling instead of our conceptual models (i.e., CDL and CVL). We describe them more fully in Section 7. In contrast, this section, highlights why we do not use them as our candidate conceptual layer.

There are two reasons not to use the four aforementioned models for our purposes. First, we are interested in a user-oriented, lightweight modeling language. None of the other candidates displays an ease of use: for example, one can hardly imagine an average executive who would be an expert in MDX, which is a highly technical language whose knowledge requires a deep expertise in multidimensional modeling. In addition, the other candidates present a level of complexity that puts them at a level of abstraction that is midway between the traditional multidimensional schemas and our CIM. Second, we want a modeling language whose constructs can be easily ascribed a clear and declarative semantics with respect to its persistence mechanism. We are working on such a semantics by expressing the main components of the CDM using Datalog rules. None of the other candidates displays a clearly defined and — to the best of our knowledge — known semantics that serves as its formal foundation. Using Datalog rules to express the foundations of the mapping models does not conflict with the goal of a user-oriented language since such a formalization is transparent to the users.

3 Conceptual Integration Modeling Framework

3.1 Visual Model

Our CVL model extends MultiDim [21], which in turn extends Peter Chen's ER model to support multidimensionality. This section both briefly reviews conceptual data modeling of multidimensionality and introduces CVL. Readers interested in more background than can be presented here should refer to another source, e.g., [21].

Typically, data warehouses are described and defined by a multidimensional model in which data are points called *facts* in an n-dimensional space called a *data cube*. The *dimensions* of a data cube are the perspectives used to analyze data. A dimension consists of a set of hierarchies where a hierarchy contains a set of *levels of granularity*. Facts are represented by *fact relationships*, which relate entities in different dimensions.

Throughout this section we use a CVL representation of a data warehouse about Olympic events and attendees (See Figure 3). In CVL, fact relationships are represented

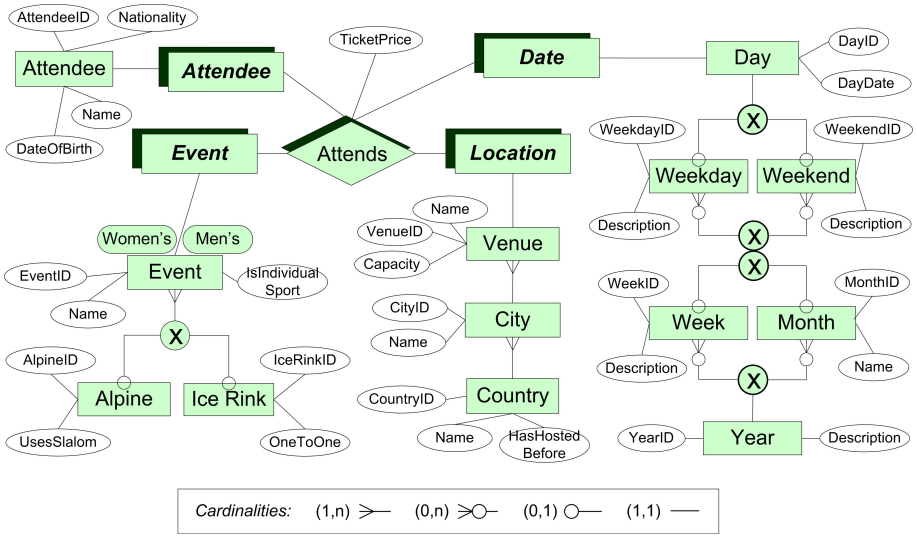

Fig. 3. Olympic CVL Model

by shadowed diamonds (e.g., Attends); shadowed rectangles depict dimensions of the fact relationships (e.g., Location, Event), and levels are represented as unshadowed rectangles (e.g., Venue, City). Unlike MultiDim, dimensions are first-class citizens in CVL diagrams, and thus are explicitly represented. Levels and fact relationships may also have properties/measures, represented by ovals (e.g., TicketPrice, VenueID). The directly attached level to a dimension is the bottom level of that specific dimension, which determines the granularity of the facts in the respective fact relationship. For instance, the TicketPrice values in Attends are given by Venue, Day, Event and Attendee.

A dimension may represent more than one hierarchy, so we introduce a flattened oval to denote the specific hierarchy (e.g., Women's and Men's). If a dimension has only one hierarchy, or all hierarchies in a domain contain the same levels, we omit the hierarchy oval. For example, each of the Location, Date and Attendee dimensions contain only one hierarchy, so a separate hierarchy oval is omitted.

Instances of levels are related hierarchically, unlike the flat, two way relationships in ER diagrams. For example, City is a child of Country. A hierarchy may have different options for a level of granularity. For example, a Day may roll up to either a Weekday or a Weekend, but not both. Similarly, a Year may drill down to exclusively a Week or a Month. This so called splitting/joining parent-child relationships are expressed with parent-child relations labeled by an encircled '**X**'. In contrast to MultiDim, which has splitting and joining levels instead of parent-child relationships, the '**X**' CVL relationships can be combined. This is exemplified by the double '**X**' connecting Weekday and Weekend to Week or a Month, representing four mutually exclusive parent-child relationships: Weekday–Week, Weekend–Week, Weekday–Month, and Weekend–Month. This is different from a disjoint inheritance in Chen's ER, where entities can have different types and one entity cannot inherit from two entities.

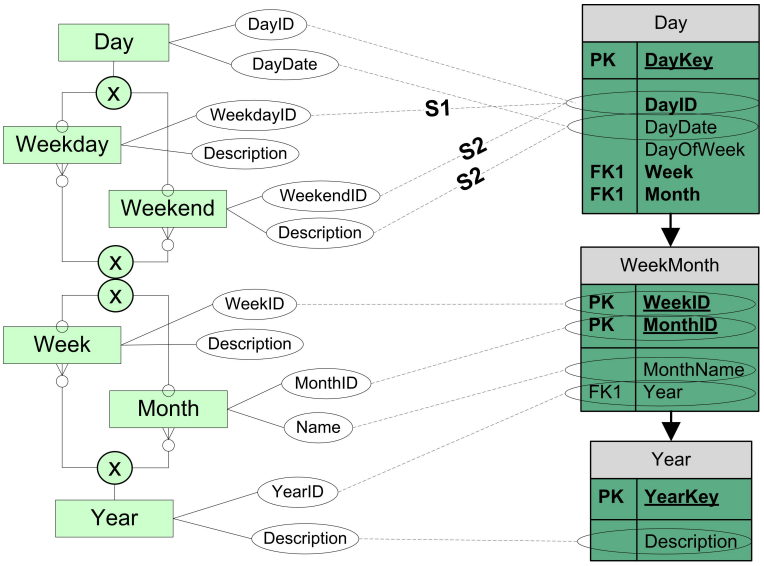

Fig. 4. CVM model of a portion of the Olympic example

Parent-child relationships have cardinality constraints indicating the minimum and maximum number of members in one level that can be related to a member in another level. For example, City is related to Country with a (1,n) to (1,1) cardinality: every city belongs to a country and each country has many cities.

The SVL is a UML-like representation of the relational data warehouse schema, containing relational table definitions, keys and referential integrity constraints. In practice, the SVL specification is automatically generated from the underlying data warehouse schema, and is put at the user's disposal.

In addition to the CVL and SVL specifications, the user also provides visual mappings between constructs in both models; the mappings constitute an MVL specification. Figure 4 shows a CVM model of a portion of the Olympic example: the left hand side of the figure shows a subset of the CVL and the right hand side depicts part of the SVL of a pre-existing Olympic data warehouse. Between the CVL and the SVL specifications, mappings in the form of dotted lines form the MVL specification. The CVL specification corresponds to what is increasingly called the *semantic layer* in industry. Such a layer liberates users from the low-level multidimensional intricacies and allows them to focus on a higher level of abstraction. For instance, the SVL model for the logical data warehouse on the right hand side has normalized tables for Day and Year but a de-normalized joint table for Week and Month. This mix of normalized and de-normalized warehouse schemas are common in real-world applications. In contrast, the CVL model on the left hand side has separate entities for Week and Month and two additional entities that are not even represented in the logical data warehouse schema:

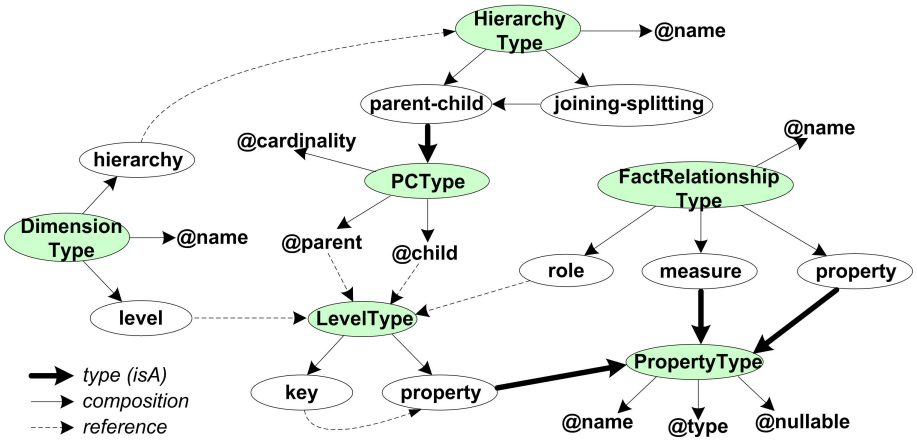

Fig. 5. CDL types

Weekday and Weekend. This divide between the conceptual and the logical levels is bridged by the MVL.

The MVL can also contain conditions to select data from an SVL table. For instance, conditions **S1** and **S2** determine what data from the Day table is mapped to Weekday and Weekend, respectively. MVL also allows for a conceptual entity to span more than one logical table, e.g., Year has one attribute mapped to the WeekMonth table and another to the Year table. Note that not all attributes need to be mapped; attributes like Description in Weekday and Week can be mapped later to the same or other data sources, or can be completed directly by the user.

3.2 Data Model

The CIM Visual Model provides a simple way for the user to model the concepts of interest and their mappings to the logical data warehouse at hand. To enable interoperability with BI applications, each visual component — CVL, MVL, and SVL — is translated into an equivalent XML-based object model — CDL, MDL, and SDL, respectively. In this XML representation, each construct is described by an XML Schema type. This separation of the CDL, MDL, and SDL layers parallels similar layers in the EDM Framework [4]. However, whereas EDM provides structure for the relational database model, our CDM provides structure for the relational multi-dimensional model. Thus, a user can design concepts in CVM that are automatically translated into CDM for run-time usage.

Figure 5 shows the CDL XML Schema types definitions and their interrelations. The green ovals are the roots of each type, the white ovals represent the elements composing the types, and attributes are prefixed by a "@". There is one type for each main construct in the CVL. As usual, a composition (solid lines) indicate nested elements in the XML instance, e.g., a **FactRelationshipType** has nested role, measure and property elements. IsA relationships (bold lines) provides typing information for an element, e.g.,

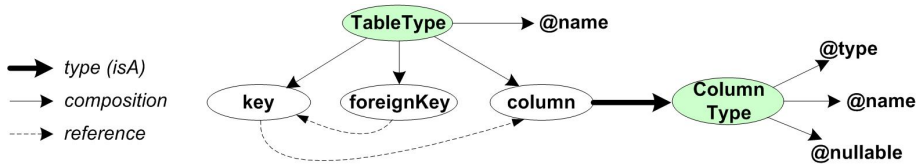

Fig. 6. SDL types

the property element within a **LevelType** is of type **PropertyType**. References (dashed lines) are referential constraints expressed by *key* and *keyref* XML Schema constraints. Although references are defined between @name attributes, in the figure we link the respective elements instead for clarity. A key constraint guarantees that the attribute in its definition has a unique value within the scope in which the key is defined. For instance, the reference between key and property elements within a **LevelType** forces each key value to refer to a property within the same level.

The root of a CDL model is a complex type that contains required levelSet, dimensionSet and factRelationshipSet, and optionally may include a hierarchySet. Each of these sets contains a list of all constructs that appear in a given model (levels, dimensions, fact relationships and hierarchies, respectively).

The SDL layer defines how the warehouse is physically stored. It builds on the relational model and thus has two XML Schema types: **TableType** and **ColumnType**. As usual, each relational table has a primary key and may have a set of foreign keys to other tables. The SDL model root is a complex type containing sets of fact tables and dimensions. Each set contains a list of all constructs in a given model. Figure 6 shows the SDL XML Schema types definitions and their interrelations.

The MDL layer expresses mappings between CDL and SDL elements. The attribute-to-attribute correspondences provided by the users in the MVL are grouped into *mapping fragments*. A mapping fragment contains all attribute associations between one CDL entity and one SDL table. A mapping fragment is represented in MDL as a level-mapping or factrel-mapping element containing the names of the CVL entity and the SVL table being associated. Each mapping fragment contains a set of one or more property-mapping elements and optional condition elements. Each property-mapping element represents one of the user-defined associations between a CVL property and an SVL table column. The condition elements specify the attribute values for which the mapping fragment holds.

Figure 7 shows a fragment of the CDM specification for the CVM example in Figure 4. Figure 7(a) and 7(b) contain the definition of the Weekend level in the CDL and the Day table in the SDL, respectively. Figure 7(c) shows the definition of mapping **S2**, which includes the condition that the value of the DayOfWeek column has to be either “Sat” or “Sun”.

It is important to note that our solution requires the user to only provide very simple property mappings between attributes in both models, which are later transparently compiled by the system into complex, fully-fledged, multidimensional mappings that can be used for query evaluation.

```
- <level name="Weekend">
    <Key name="WeekendID" />
    <property name="WeekendID" type="String" nullable="false" />
    <property name="Description" type="String" nullable="true" />
  </level>
```

(a) CDL Weekend level

```
- <dimension name="Day">
    <key name="Daykey" />
    <foreignkey name="Week" table="WeekMonth" column="WeekID" />
    <foreignkey name="Month" table="WeekMonth" column="MonthID" />
    <column name="Daykey" type="nvarchar" nullable="false" />
    <column name="DayID" type="nvarchar" nullable="false" />
    <column name="DayDate" type="datetime" nullable="true" />
    <column name="DayOfWeek" type="nvarchar" nullable="true" />
    <column name="Week" type="nvarchar" nullable="true" />
    <column name="Month" type="nvarchar" nullable="true" />
  </dimension>
```

(b) SDL Day table

```
- <level-mapping level="Weekend" dimension="Day">
    <property-mapping property="WeekendID" column="DayID" />
    <property-mapping property="Description" column="DayDate" />
    <condition column="DayOfWeek" expression="{Sat,Sun}" />
  </level-mapping>
```

(c) MDL Weekend-to-Day mapping

Fig. 7. CDM level instance and **S2** mapping for the Olympic example

4 Architecture

We can now describe the architecture of the CIM Framework as it is being currently implemented. The main components of this architecture are given in Figure 8.

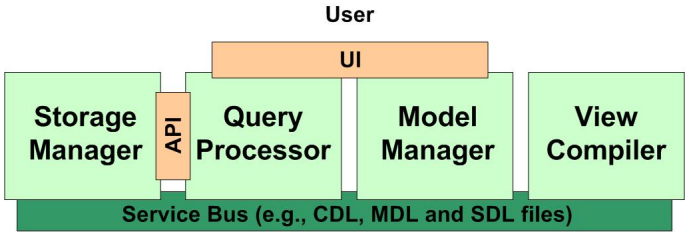

Fig. 8. CIM Framework functional architecture

The functionality of the components is described as follows:

– **Storage Manager**. This component creates the SVL models automatically from the metadata exported from the underlying data warehouse. Currently, we use Mondrian [23], an open source ROLAP engine, for this purpose. The component also maintains the SDL views created by the View Compiler component together with any other materialized and virtual view created in the system, either relational or multidimensional.

– **Query Processor**. The query processor rewrites user queries posed on the CDL model in terms of the SDL views created by the View Compiler and sends the rewritten query to the Storage Manager. This component treats the processing of queries as an instance of the classical problem of processing queries using views.
– **Model Manager**. This component takes a CVM model as input (i.e., the SVL model generated by the Store Manager, and the CVL and MVL models provided by the user). The component produces a CDM model (i.e., CDL, MDL, and SDL specifications) as output. Any changes in the CVM model by the user will be incrementally propagated to the CDM.
– **View Compiler**. This component takes the CDL, MDL, and SDL models generated by the Model Manager, and produces multidimensional views over the SDL model. The component creates a view definition for each fact relationship, level and parent-child relationship in the CDL based on the mappings (MDL) and the referential constraints that appear in the SDL. It may also restructure the hierarchies into summarizable ones [17] if needed. These views allow queries posed against a CDL schema to be rewritten and posed against the compiled SDL views.

Users interact with the Model Manager and the Query Processor components via a graphical User Interface. All modules communicate with each other via a service bus. For the first prototype under implementation, this bus is simply the operating system file system where the models are stored and read in the form of XML files. We plan to replace the file system by web services in a service oriented architecture. Finally, the Query Processor component communicates directly with the Storage Manager component via a well defined application programming interface.

5 Sample Usage Scenario

Section 3 described the major components of the CIM Framework. This section gives an overview of a usage scenario by briefly describing how a user creates CVL and CDM models and uses them at run time to answer queries.

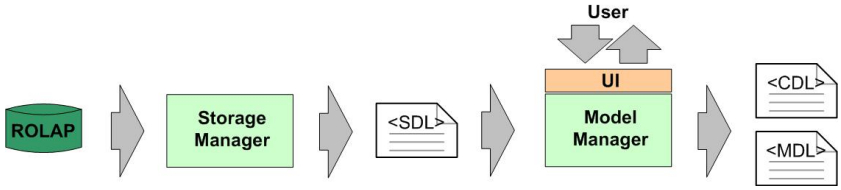

Fig. 9. Model creation dataflow

Model creation. Figure 9 outlines the first step in the usage of the CIM Framework. The user — typically, a business user — specifies a CVL model of the data warehouse (as described in Section 3.1) using a graphical user interface offered by the Model Manager. Then, using the Model Manager, the user imports the SDL that represents the

logical multidimensional schema of an existing data warehouse. The Storage Manager produces such an SDL model from that existing multidimensional schema. The imported SDL will be visualized as an SVL model for further usage. Furthermore, using the Model Manager graphical user interface, the user establishes an MVL model by mapping the CVL constructs to the SVL constructs. Finally, the Model Manager produces the CDL and MDL models corresponding to the input CVL and MVL models, respectively. Figure 4 illustrates a portion of the CVM corresponding to the Olympic example of Figure 3. It is important to notice that, similarly to the classical data independence ensured by the distinction between the conceptual and the logical layers in relational databases, the use of mappings in our framework ensures an independence of the user-defined CVL and CDL layers from the underlying multidimensional layer of the data warehouse.

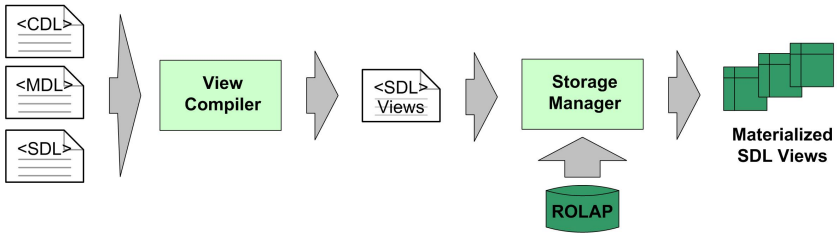

Fig. 10. View compilation dataflow

View compilation. Once the CDM model (i.e., the CDL, MDL, and SDL models) has been generated by the Model Manager, a view compiler takes this CDM model as input and produces SDL view definitions, each of which is maintained – either materialized or virtual – by the Storage Manager. CIM uses mappings of the common form $c \rightsquigarrow \psi_s$, where c is an element in the *target* schema and ψ_s is a view over the *source* schema. In CIM, the conceptual model (CDL) functions as the target schema and the store model (SDL) as the source schema. The element c in the mapping expression is either a level, a parent-child relationship, or a fact relationship, whereas ψ_s is a view over the data warehouse's multidimensional tables. CIM uses *sound* mappings, i.e., those defined by $\forall \bar{x}(\psi_s(\bar{x}) \rightarrow c(\bar{x}))$ expressions, where $\bar{x}$ denotes a set of variables. Figure 10 depicts the compilation process. Here, we assume a ROLAP data warehouse.

Query formulation and evaluation. Consider again the Olympic example in Figure 3. Suppose the user now wants to aggregate the ticket sales in the "Attends" fact relationship by weekend and for one specific venue: "Whistler Olympic Park". This query would be expressed in the CVL query model UI with an *aggregated fact relationship* construct, "Aggr_Attends", that has links to the respective fact relationship ("Attends") and the levels being aggregated ("Weekend" in Date, and "Venue" in Location). Since the user wants the aggregation for a single venue, the link from "Aggr_Attends" to "Venue" will be labeled by the selection condition, in this case "Venue.name=Whistler Olympic Park". All levels that are not being aggregated or filtered by any value condition, such as "Attendee" and "Event", do not have to be specified in "Aggr_Attends" —

they can be obtained from the respective fact relationship. In addition, each aggregated fact relationship is linked to another construct specifying the aggregation function and measure, in this case “sum” and “TicketPrice”.

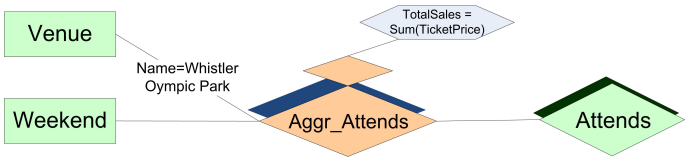

Fig. 11. Aggregation query for the olympics example.

Once the query is posed to the Query Processor (QP) via the UI, the QP uses the SDL view definitions to rewrite the query in terms of the views. For this example, one of the view definition would come from the S2 mapping in Figure 4 which selects all tuples in “Day” table with “DayOfWeek={Sat,Sun}” and is used during View Compilation to define an SDL view for weekend days. The rewritten query is then sent to the Storage Manager (SM), which process the query using the compiled SDL views, any precomputed cubes at hand and the data warehouse base data. The answer is then returned to the QP and finally to the user via the UI. Figure 12 depicts the query evaluation process.

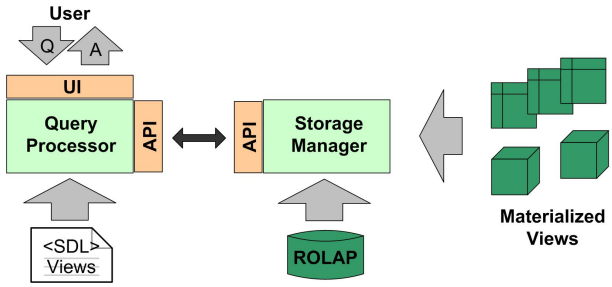

Fig. 12. Query evaluation dataflow

6 Related Work

Warehousing strategies have been classified into the following three categories [8]: data-driven, goal-driven, and user-driven. Data-driven methodologies use the data models of the production database systems as the departure point of the modeling requirements of the data warehouse. Goal-driven methodologies use the corporation business processes and requirements (goals) to systematically derive the model of the data warehouse. Finally, user-driven methodologies derive the data warehouse requirements from the user in the first place. As already mentioned above, all these categories of current strategies consider information to be integrated as the focus of the data warehousing process and any goal or user-driven aspects are considered mostly once the integration is performed. Thus, as such, they still are largely bottom-up.

We believe that, while data-driven methodologies are inherently bottom-up by nature, the goal and user-driven methodologies form one single category of stakeholder-driven methodologies, since users are a subset of stake holders which also comprise organizations. Stakeholder-driven methodologies are inherently top-down since they free the stake holders as much as possible from the burden of taking care of low level tasks. Moreover, it has been noticed in [19] that the bottom-up strategy for building the warehouse is contrasting with a top-down strategy which the authors doubt could be possible; they even go as far as to assert that using some form of conceptual model to that end is not feasible. Our CIM Framework is a challenge to this doubt: we are building a tool showing that a top-down approach is possible where stakeholder requirements are specified in some given modeling language first, and then the data warehouse design and subsequent population are derived from those specified requirements.

Conventional database design includes the three well-known phases of conceptual design (using the Entity Relationship model originally proposed by Peter Chen [7]), logical design, and physical design. These phases result in the creation of database schemas at each of the conceptual, logical, and physical levels. Contrary to the assertion in [19], the last ten years have witnessed the emergence of some approaches for data warehouse design which adapt the design phases used in the conventional design of operational databases (including the conceptual design phase) to the characteristics of data warehouses [28, 21]. One such characteristic is the consideration of data provenance and availability in data warehouse conceptual models.

Despite some progress in the last ten years with respect to multidimensional modeling, the field of conceptual modeling of data warehouse is still in its infancy as recognized in [21]. As noticed in [21], many of the approaches taken towards multidimensional modeling are at the logical level. Many others restrict themselves to a subset of the features of a data warehouse. Only a few have tackled the issue of conceptual multidimensional modeling by taking into account all facets of a data warehouse [21].

Conceptual models for data warehouse design can be broadly classified into graphical [10, 25, 1, 9] and non-graphical [24] approaches. We restrict our attention to graphical approaches. There are a number of different graphical representations of conceptual multidimensional schemas for warehouse modeling, e.g., [25, 9, 5, 10]. Hahn *et al.* describe the design and implementation of a tool for automatically generating On Line Analytical Processing (OLAP) schemas from conceptual graphical models [10]. They use a multidimensional entity relationship (M E/R) model as the conceptual graphical language; M E/R is a extension of the ER model [7] for multidimensional purposes. Malinowski and Zimànyi describe a multidimensional model called MultiDim for representing OLAP and data warehouse requirements [21]. The model represents the various data warehouse concepts such as facts, dimensions, hierarchies, measures, etc., by means of a graphical notation similar to the ER model. Moreover, the model puts a special emphasis on capturing the various sorts of (often irregular) hierarchies that appear in real-world applications. Malinowski and Zimànyi also provide mapping rules for implementing the MultiDim schemas into the object-relational logical model. This mapping is based on the classical rules for mapping ER models to the relational model.

The graphical approaches mentioned above are mainly proposals for modeling languages made as part of data warehouse design methodologies. They are rarely used

as a run time environment. By contrast, our proposal provides a run-time environment that allows a user to pose queries and do business analytics at a conceptual level that is abstracted from the logical multidimensional data level.

As already mentioned in Section 1, the EDM Framework [2–4] provides a querying and programming platform that raises the level of abstraction from the logical relational data level to Peter Chen's ER conceptual level. At design time, a developer provides three artifacts: an Entity Relationship schema written in a conceptual schema definition language (CSDL), a relational database schema expressed in a store schema definition language (SSDL), and a mapping between the CSDL and SSDL schema elements. These three components make up an EDM model. In addition, a query language, Entity SQL (eSQL), is defined over the CSDL. A compiler takes an EDM model as input and generates the mapping information in the form of eSQL views that express ER constructs in terms of relational tables. At run time, eSQL queries over the CSDL are run against the eSQL views via view unfolding.

Hibernate is an Object-Relational technology that allows the creation of persistent Java classes. At design time, the developer writes a Java program that comprises classes and mappings of those classes to SQL queries over an underlying relational schema. The mappings involved here are not compiled at all: they solely serve the purpose of directly translating objects to SQL queries.

Mondrian [23] is an open-source OLAP engine implemented with Java technology. It executes queries written in MDX over data coming from a relational database backend (e.g., MySQL, PostgreSQL, Oracle), and presents the results in a multidimensional format via a Java API. Mondrian's goal is to provide data warehouse and OLAP functionality on top a relational DBMS.

Like our CIM Framework, EDM and Hibernate aim to bridge the gap between the prevalent object-oriented world of application programmers and the world of relational data. Unlike our CIM Framework, which deals with the multidimensional data model, EDM and Hibernate deal with the classical relational data model.

On the industry side, two major vendors (namely SAP Business Objects and IBM) of business analytics solutions provide proprietary conceptual levels that they call "semantic layers". SAP Business Objects' semantic layer [15], called a *Universe*, is a business representation of an organization's data asset (i.e., data warehouse as well as transactional databases). The universe lies between the end user and the organization's data asset; it hides the complex structure of the underlying data assets as well as their provenance and localization from the end user by providing the later with a familiar business-oriented interface. IBM's semantic layer, Framework Manager [31], is similar to SAP's Universes and works according to the same principles.

7 Conclusion and Current Status of the Implementation

An increasing number of business analytics application are being used by business executives and end users who are infrequently schooled in the intricacies of data warehousing, data integration, and complex query formulation technologies. Technologies are emerging in Industry to tackle the issue of bridging the gap between business decision applications, which are end-user oriented, and the existing, complex data warehousing

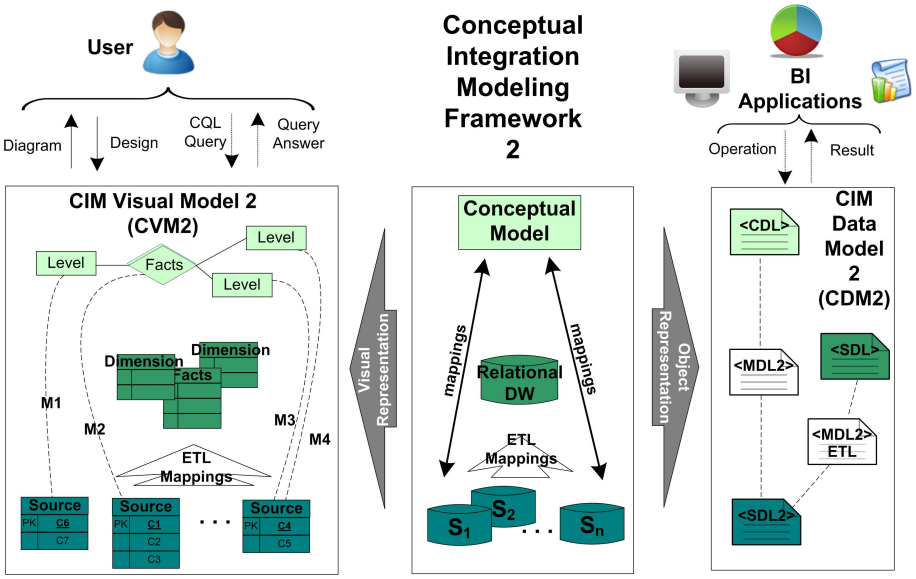

Fig. 13. Data centered architecture for automatic generation of the data warehouse

technology via the use of abstract layers. The CIM Framework fills a void in academia for studying these abstract layers in a principled way. We are currently implementing the architecture depicted in Figure 8. The implementation is using the Mondrian open source OLAP server [23] as data store.

Conceptual models are known to be very expressive and processing queries over them is hard [6]. The next steps in the materialization of the CIM Framework include the study of tractable and yet practical cases of processing queries over our conceptual model. A further avenue that we will pursue in the future is to relax the assumption that the data warehouse with its schema are given and that the only remaining task to do is to build a conceptual level over the given data warehouse schema as well as a mapping between the conceptual level and the warehouse schema. Figure 13 presents a data-centered architecture for automatic generation of the data warehouse, assuming that only the conceptual model along with source schemas are given and that the data warehouse with its schema (as well as the mappings to the conceptual schema and to the sources) must be generated and populated as automatically as possible. We intend to expand our CIM Framework to such an architecture.

Acknowledgements. We thank Daniele Barone for his insightful comments on the manuscript. This research was supported in part by the NSERC (Natural Sciences and Engineering Research Council) Business Intelligence Network, IBM Canada and SAP Business Objects division.

References

1. A. Abelló, J. Samos, and F. Saltor. Yam (yet another multidimensional model): An extension of uml. *Information Systems*, 32(6):541–567, 2006.
2. A. Adya, J.A. Blakeley, S. Melnik, and S.P. Muralidhar. The ado.net team. anatomy of the ado.net entity framework. In *SIGMOD*, 2007.
3. J.A. Blakeley, D. Campbell, S. Muralidhar, and A.S. Nori. Entity framework: Making the conceptual level real. *SIGMOD Record*, 35(2):32–39, 2006.
4. José A. Blakeley, S. Muralidhar, and Anil Nori. The ado.net entity framework: Making the conceptual level real. In *ER*, pages 552–565, 2006.
5. L. Cabibbo and R. Torlone. From a procedural to a visual query language for olap. In *SSDBM*, 1998.
6. Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Conceptual modeling for data integration. In *Conceptual Modeling: Foundations and Applications*, pages 173–197, 2009.
7. P.P. Chen. The entity-relationship model - toward a unified view of data. *ACM TODS*, 1(1):9–36, 1976.
8. B. List *et al.* A comparison of data warehouse development methodologies: Case study of the process warehouse. In *DES*, 2002.
9. M. Golfarelli and S. Rizzi. A methodological framework for data warehouse design. In *DOLAP*, 1998.
10. K. Hahn, C. Sapia, and M Blaschka. Automatically generating olap schemata from conceptual graphical models. In *DOLAP*, 2000.
11. Alon Y. Halevy, Naveen Ashish, Dina Bitton, Michael J. Carey, Denise Draper, Jeff Pollock, Arnon Rosenthal, and Vishal Sikka. Enterprise information integration: successes, challenges and controversies. In *SIGMOD Conference*, pages 778–787, 2005.
12. A.Y. Halevy. Answering queries using views: a survey. *The VLDB Journal*, 10(4):270–294, 2001.
13. J. Han and M. Kamber. *Data Mining: Concepts and Techniques, Second Edition*. Morgan Kaufmann, New York, 2006.
14. Hibernate Project. http://www.hibernate.org/.
15. Cindi Howson. *BusinessObjects XI (Release 2): The Complete Reference*. McGraw-Hill, 2006.
16. R. Hull. Managing Semantic Heterogeneity in Databases: A Theoretical Perspective. pages 51–61, 1997.
17. C. Hurtardo and A. Mendelzon. Reasoning about summarizability in heterogeneous multidimensional schemas. In *ICDT*, 2001.
18. M. Jarke, M. Lenzerini, Y. Vassiliou, and P. Vassilisadis. *Fundamentals of Data Warehouses, Second Edition*. Springer, Berlin, 2003.
19. R. Kimball, L. Reeves, M. Ross, and W. Thornthwaite. *The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing, and Deploying Data Warehouses*. John Willey and Sons, New York, 1998.
20. M. Lenzerini. Data integration: a theoretical perspective. In *Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, Madison, Winsconsin, 2002.
21. E. Malinowski and E. Zimányi. *Advanced Data Warehouse Design: From Coventional to Spatial and Temporal Applications*. Springer, Berlin, 2008.
22. S. Melnik, A. Adya, and P.A. Bernstein. Compiling mappings to bridge appications and databases. In *SIGMOD*, pages 461–472, 2007.
23. Mondrian Project. http://mondrian.pentaho.org/.

24. T. Pedersen, C.S. Jensen, and C. Dyreson. A foundation for capturing and querying complex multidimensional data. *Information Systems*, 26(5):383–423, 2001.
25. C. Sapia, M. Blaschka, G. Höfling, and B. Dinter. Extending the e/r model for multidimensional paradigm. In *ER*, 1998.
26. George *et al.* Spofford. *MDX Solutions. Second Edition*. Wiley, 2008.
27. The Economist. Data, data everywhere: A special report on managing information, February 27, 2010.
28. R. Torlone. *Conceptual Multidimensional models*, pages 69–90. Idea Group, 2003.
29. J. Trujillo, M. Palomar, J. Gomez, and I. Song. Designing data warehouses with oo conceptual models. *IEEE Computer*, 34(12):66–75, 2001.
30. N. Tryfona, F. Busborg, and J. Borch. Starer: A conceptual model for data warehouse design. In *DOLAP*, 1999.
31. Dan Volitich. *IBM Cognos 8 Business Intelligence: The Official Guide*. McGraw-Hill, 2008.