Designing Integration Flows Using Hypercubes

Kevin Wilkinson
HP Labs
1501 Page Mill Rd.
Palo Alto, CA, USA
kevin.wilkinson@hp.com

Alkis Simitsis
HP Labs
1501 Page Mill Rd.
Palo Alto, CA, USA
alkis@hp.com

ABSTRACT

The design and implementation of an ETL (extract-transform-load) process for a data warehouse proceeds from a conceptual model to a logical model, and then a physical model and implementation. The conceptual model conveys at a high level the data sources and targets, and the transformation steps from sources to targets. The current state of the art is to express the conceptual model informally using text descriptions and diagrams. This makes the process of deriving a logical model time-consuming and error-prone. Our work is towards a system that covers the whole ETL lifecycle by injecting several layers of optimization and validation throughout the whole process starting with the business level objectives and ending with flow execution. In this paper, we focus on the ETL conceptual layer and present a solution that assists consultants in their task of defining the needs and requirements at the early stages of an integration project. We present a conceptual model for ETL based on hypercubes and hypercube operations. This is a formal model that captures the semantics of ETL at a high-level but that can also be machine-translated into a logical model for ETL. The use of hypercubes at the conceptual level renders a design that can be easily understood by business users and so reduces design and development time and produces a result that accurately captures service level agreements and business requirements.

Categories and Subject Descriptors

H.2.7 [**Database Administration**]: Data warehouse and repository.

General Terms

Algorithms, Management, Design, Languages.

Keywords

Integration flows, ETL design, Business requirements, Data Warehouse, Hypercubes, Conceptual Model.

1. MOTIVATION

The current state of the art for creating a conceptual model for an ETL (extract-transform-load) process for a data warehouse is to describe the ETL concepts such as data sources, targets, and operations informally using a combination of text descriptions and diagrams. Business consultants and practitioners with significant experience with ETL engagements state there is little support for conceptual designs and, typically, they need to build ETL logical designs using requirements written in an ad hoc way in text documents or spreadsheets (also see the upper part of Figure 1). There is no standard notation or vocabulary; each practitioner uses their own best practices. Consequently, the details captured by these models vary widely across ETL projects. Conceptual models devised for one project may not be comprehensible to others without help from those who devised the models. The use of a formal language for defining ETL conceptual models has several advantages. First, the formal model provides a common vocabulary of objects and operations so models can be understood without the help of the designer. Second, use of a formal model makes feasible the automatic generation of a logical model from the conceptual model. It also makes feasible the computation of properties over the model such as differences between successive versions of the model or provenance information such as which targets are derived from which sources.

This paper describes the use of *hypercubes* as a conceptual model for ETL. This design choice has been made due to several reasons. First, hypercubes are a natural formalism to domain experts, as business managers and analysts, who provide the ETL business requirements. Processes expressed using hypercube operations can be readily understood by them and this helps ensure the ETL design captures the business requirements. Second, hypercube operations are easily translated into a logical ETL model because logical ETL operators are table-oriented and hypercubes are a just a generalization of relational tables. This reduces development time for creating the logical model. In summary, the use of a formal, yet comprehensible language for a conceptual model reduces development time for ETL and improves accuracy and the use of hypercubes and hypercube operations as the formal model facilitates communication between ETL designers and the business experts who are the end-users of the system.

2. SOLUTION APPROACH

2.1 System Overview

For a period of two years, we have systematically worked on a system dealing with the ETL and live business intelligence ecosystem. The ultimate goal is to provide a unified solution for the whole ETL lifecycle that starts with capturing business requirements into a high level, easily grasped, yet formal conceptual model, then continues with the production of logical level design that can be optimized for a variety of both quantitative objectives –like performance, recoverability, fault tolerance, freshness– and qualitative ones –like maintainability, manageability– and finally,

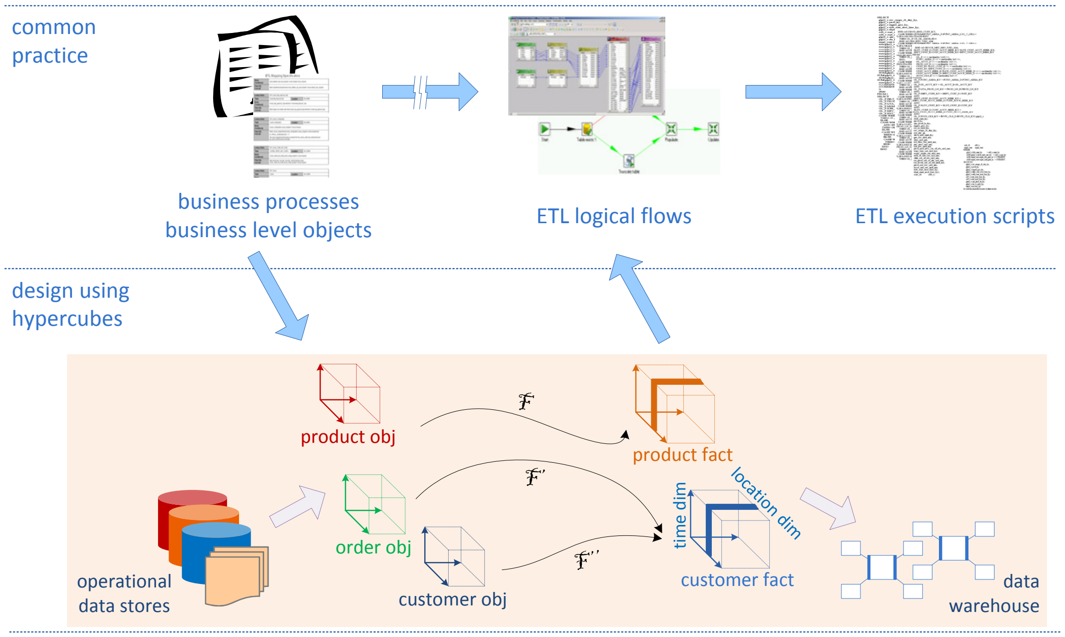

Figure 1: ETL design lifecycle

produces an executable package that after undergoing some further optimization, performs the integration task. We use the term QoX to refer to the set of quality objectives. During this lifecycle several validation steps occur and control the process. The heart of this system is the QoX optimizer that optimizes logical and physical flows for a number of QoX metrics like the ones mentioned above. Typical optimization strategies considered by our QoX optimizer include parallelization techniques (e.g., clustering, MPP), NMR for fault tolerance, streaming processing techniques and operations, choice of execution engines, resource management, and so on. Parts of our system have been described elsewhere (see [4], [5], [8], [9], and [11]). In this paper, we focus on the conceptual layer (see the lower, highlighted part of Figure 1) and present a methodology that assists consultants in their task of capturing business needs into a comprehensive design that can be (semi-)automatically translated into a logical model, which feeds our QoX optimizer.

2.2 Hybercubes

A conceptual model for an ETL process should convey the data source objects, the warehouse target objects, and a high-level description of the transformations required to convert the sources to the targets. High-level means that the model is in terms of business objects and operations as opposed to the IT-level objects presented by the ETL logical and physical models (e.g., files and tables). In other words, the conceptual model should be comprehensible to the business domain expert for the flow. Typically, an ETL flow comprises three phases, extract, transform and load. Our approach adapts these three phases for a hypercube model. Specifically, the output of the extract phase is a set of hypercubes for the data extracted from the sources. The transformation phase uses the source hypercubes to produce a set of load hypercubes. The load phase then updates the data warehouse from the load hypercubes. The details of converting between relational and hypercube models are encapsulated (and hidden) in the extract and load phases. We describe in more detail how the conceptual model represents each phase.

Extract phase. At a conceptual level, the extract phase models the interface between the data sources and the ETL transformations. The data sources are the information objects in the operational systems of an enterprise (e.g., order-entry, supply-chain, shipping, invoices). These information objects may themselves represent data objects or business processes. Our technique supports both. For each instantiation of the ETL process, a subset of information from the data sources is extracted as input to the transformation phase. Consequently, the conceptual model for extract must convey the data source objects and the particular subset of interest (e.g., all orders in the last week, yesterday's invoices).

It is worth noting that the extract phase may itself transform the input data objects into different objects that are more amenable to processing in the transformation phase. Those extract transformations may or may not be exposed to the business user. The important point is that the output of the extract phase is a set of business data objects that are input to the transformation phase. Those data objects are represented in the conceptual design as hypercubes, rather than files or tables which are the typical storage format in operational systems. For example, the conceptual objects extracted from an order-entry system might be a hypercube for recent orders and individual hypercubes for new products, new customers, and so on. In contrast, a logical (and physical) ETL model presents an IT-level view in which data is extracted into tables and files. This low level of detail is proven not appropriate for domain experts. Hypercubes present a business view and make the conceptual model independent of changes at the logical and physical levels; for example, new tables, indices that might be added for performance reasons but do not modify the underlying conceptual objects, and so on. Note that the specific business objects produced by the extract phase depend on the needs of the business. For example, information about new orders could be extracted into a single hypercube or into two hypercubes, one for order summary information (date, customer, amount) and a second for order detail information (product number, quantity sold, unit price). The choice depends on the business.

More formally, in the conceptual model the output of the extract phase is a set of hypercubes: $XC_1, ..., XC_n$, where the schema for a hypercube defines its dimensions (e.g., date, time, customer, product) and the contents of its cells (e.g., quantity sold, sale amount, taxes). To enable semi-automatic translation of the conceptual model to a logical model, the conceptual model requires two additional specifications. It includes a list of the logical source objects: $S_1, ..., S_k$, in the operational systems, e.g., the specific tables and files that are read during the extract. It also includes a mapping from the source objects to the extract hypercubes, i.e., $\{S_i\}\times\{XC_j\}$. In general, these two additional specifications are not exposed to the domain expert. The conceptual model only need present the extracted hypercubes. However, there may be cases where a business need requires a domain expert to know details of a particular mapping (e.g., for provenance or auditing) so there is no blanket prohibition. The point is that, depending on the business needs of a project, the conceptual model may present various views, each with different levels of abstraction and detail.

Load phase. The load phase of an ETL conceptual model describes how business data objects are incorporated into the target data objects. Typically, the target is a data warehouse that is modeled as a star schema in which a large fact table (e.g., orders) references numerous, smaller dimension tables (e.g., products, customers, stores, dates). The star schema is naturally modeled as a hypercube where the cells of the hypercube correspond to values in the fact table (e.g., order amount, total tax) and the dimensions of the cube correspond to the dimension tables (e.g., customer details, product details) that are referenced from the fact table. In the conceptual model, each fact table and its dimensions could be modeled as a single hypercube or it could be modeled as one hypercube for each dimension plus one for the facts. The choice depends on the business requirements. For example, if a dimension has some identity and properties that may change over time, e.g., a customer, then it may make sense to create a separate business object (hypercube) for that object. Other objects such as date and time have no real identity and so do not merit a separate hypercube and should be incorporated directly into the fact hypercube.

Formally, the load phase can be specified by a set of load hypercubes: $LC_1, ..., LC_m$, each with their own schema, a set of target objects in the logical model of the data warehouse: $T_1, ..., T_r$, and a mapping $\{LC_i\}\times\{T_j\}$ between the two. As with the extract phase, the mappings are needed for the generation of a logical model and are generally not exposed in views of the conceptual model, except when relevant to some business need.

Transformation phase. Given the source hypercubes produced by the extract phase and the target hypercubes required for the load phase, the transformation phase expresses the business rules that map the sources to the targets. Since both sources and targets are hypercubes, the business rules are expressed as a series of transformation steps where each step is a hypercube operation. We assume a set of intrinsic hypercube operators, e.g., slice, join, diff, rollup, and so on (see [2]). It is important to be able to define higher-level, abstract operations to represent a series of hypercube operations. This makes the model more readable and enables reuse of functionality. Therefore, we assume a set of intrinsic macro operators that are implemented using intrinsic operators and other intrinsic macro operators. The macro operators may have parameters so their instantiation (or expansion) may have different forms, depending on the parameters. In addition, an ETL designer may define additional macro operators that are specific to an ETL project. For example, surrogate key generation is a frequent operation in ETL and is an appropriate candidate for an intrinsic macro operation (see the Identity Resolution macro in Figure 6). On the other hand, normalizing postal addresses may not be a good candidate for an intrinsic macro operation due to the wide variety of encodings of addresses and the various tools for cleaning and transforming addresses.

Formally, we define the transformation phase as a function that maps the extract hypercubes to a set of load hypercubes, i.e., $\{LC_i\} = F(\{XC_j\})$. The function F is itself a graph of operators as follows. Let TN be the set of all intrinsic hypercube operators and TM be the set of all macro hypercube operators, and let T be the set of all hypercube operators, i.e., $TN \cup TM$. Then F is a graph of operators that map one set of hypercubes to another set, i.e., $\{XC_j\} \rightarrow f_1 \rightarrow \{C_1\} \rightarrow f_2 \rightarrow \{C_2\} \rightarrow ... \rightarrow f_k \rightarrow \{LC_i\}$, where each transformation f is an element of T and C_i represent a set of temporary, intermediate hypercubes. Also, note that each macro transformation, TM, when expanded, is itself a series of hypercube operators as above but where each function f is an intrinsic operator in the set TN.

2.3 Producing the Logical Design

After having constructed the conceptual design, we transform it to a logical model, which can be later fed to our QoX optimizer. Figure 2 abstractly pictures this process. In the upper figure, hypercubes and hypercube operators are presented. The details of the extract from data sources and load to the warehouse target need not be exposed. In the lower figure, the sources and target tables are shown in detail along with details of the transformation steps.

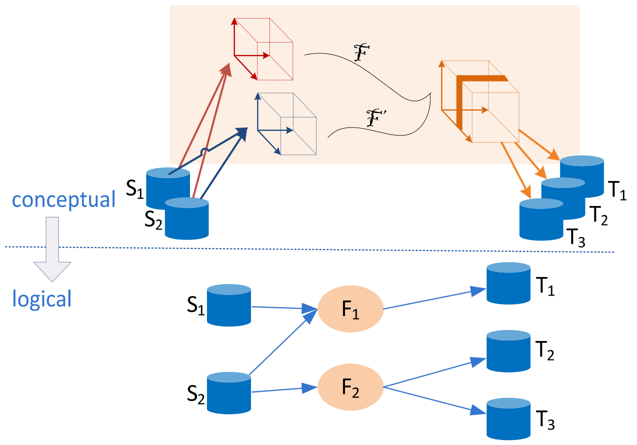

Figure 2: Conceptual (top) and logical (bottom) designs

Figure 3 shows a high-level algorithm for generating the logical ETL model from the conceptual model. The first steps are defining the extract and load hypercubes and the transformations between them. Note that there may be dependencies among the load hypercubes. For example, when loading a hypercube representing a fact table in a star schema, it is important to first load hypercubes that represent dimensions referenced by the fact hypercube. Then, for each load hypercube, a semantic consistency check is performed to ensure that the flow is valid. Next, macro expansion occurs to produce a flow consisting entirely of intrinsic hypercube operators. Finally, the logical ETL operators are generated from the intrinsic hypercube operators.

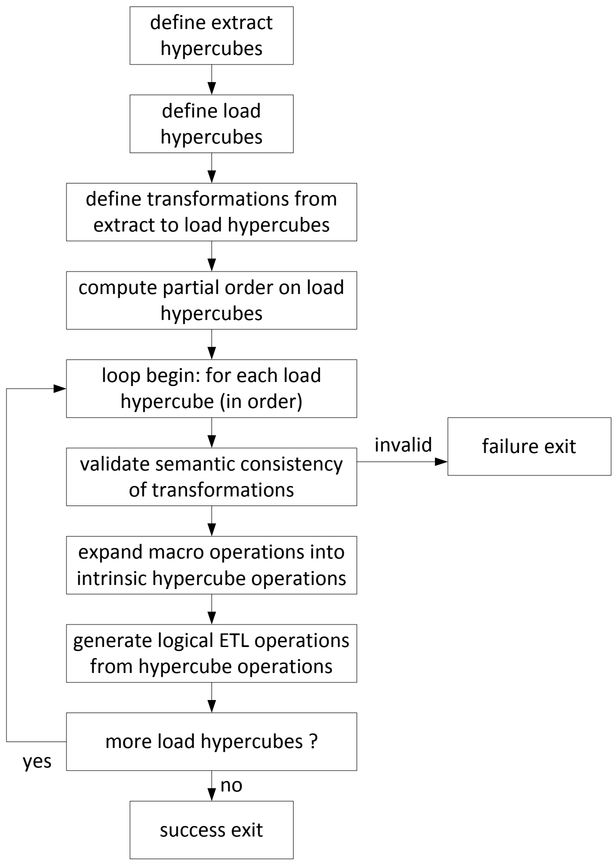
define extract hypercubes
define load hypercubes
define transformations from extract to load hypercubes
compute partial order on load hypercubes
loop begin: for each load hypercube (in order)
validate semantic consistency of transformations
invalid
failure exit
expand macro operations into intrinsic hypercube operations
generate logical ETL operations from hypercube operations
more load hypercubes ?
yes
no
success exit

Figure 3: Generating ETL logical flow from conceptual design

Next, instead of delving into a formal representation of the methodology, we demonstrate it through a series of indicative examples representing frequently used operations, we show how such operations can be modeled, and for a few of them (due to space constraints) we illustrate how they expand into detailed operations that correspond to logical level activities.

3. EXAMPLE CASES

Extract hypercube. Consider the order-entry system of an on-line retail enterprise. We can assume that the operational systems will generate order details as well as updates to customer data and product data as customers and products come and go. From a business perspective, there are three or four business objects involved, customers, products, orders and order lineitems (individual products) corresponding to three or four source hypercubes[1]. It is a business decision if the order data should be viewed as one object (an orders cube with the lineitem details) or as two separate objects (one orders summary cube and one lineitems cube). In addition, consider the customer data. For efficiency, the operational system may store customer information in multiple, normalized tables. For example, the customer profile information may be stored separately from the primary customer table; e.g., customer profile information in one table, customer demographic information in another, customer address in another and customer identity in another. The details of the logical storage schema are not relevant to the business user. In fact, including such details in the conceptual model makes it dependent such that changes to the source schemas require changes to the conceptual model. This is undesirable and violates the abstraction required at the conceptual level.

[1] Note that mathematically, a single, large hypercube is sufficient for the extract phase, but this is not natural for a business view.

Consequently, the conceptual model presents customer information as a single hypercube regardless of the logical schema at the source. The conceptual model includes annotations that specify how that hypecube is created from the source objects; e.g., the relational algebra operations to join the customer and profile, tables, surrogate key generation (IdRes, as explained later), and so on. However, these annotations need not to be exposed to the business user and are only used in creating a logical model from the conceptual model. In addition, the annotations must specify attributes used to specify the subset of source data extracted (Figure 4). For example, a source hypercube may include a timestamp attribute and annotations for the extract phase should specify the timestamp range for each instance of running the extract flow; e.g., extract customer changes for the previous hour, the previous day, last week, and so on.

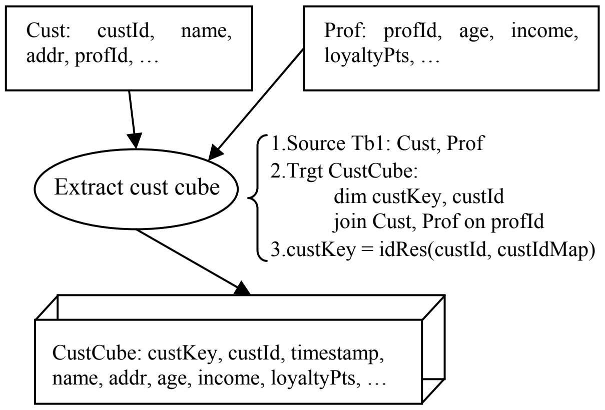
Cust: custId, name, addr, profId, …
Prof: profId, age, income, loyaltyPts, …
Extract cust cube
1.Source Tb1: Cust, Prof
2.Trgt CustCube:
dim custKey, custId
join Cust, Prof on profId
3.custKey = idRes(custId, custIdMap)
CustCube: custKey, custId, timestamp, name, addr, age, income, loyaltyPts, …

Figure 4: Extract Hypercube

As another example, suppose the source system generates XML documents to represent order information where each document includes order summary and lineitem details. In addition, suppose the business requirements dictate separate hypercubes for order summary and details. In this case, the annotations would present the logical source object as an XML document and include the XSLT operators to extract the order summary information into one hypercube and the order detail information into the second hypercube.

Transform hypercube. The conceptual model for the transformation phase is expressed as a graph of hypercube operations and higher-level, macro operations. As a simple example, suppose a load hypercube presents product sales per week, i.e., the total sales for each product sold for each week. We assume the extract phase produces a hypercube with order detail information, either as a separate hypercube or combined with the order summary. In this case, the load hypercube can be expressed as a hypercube roll-up operation over the extract hypercube on the date (week) and product dimensions (product number) as shown in Figure 5.

Identity resolution. A frequent transformation for data warehouses is the generation of a surrogate key to represent each unique object in the warehouse (customer, order, product). The surrogate

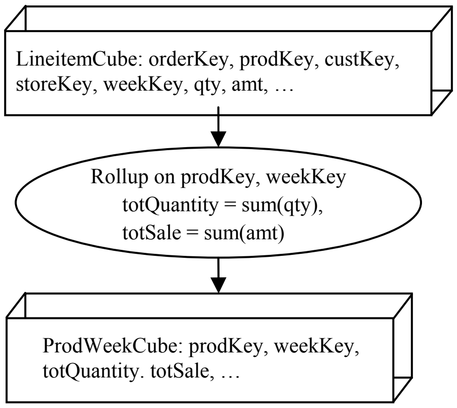
LineitemCube: orderKey, prodKey, custKey, storeKey, weekKey, qty, amt, …
Rollup on prodKey, weekKey
totQuantity = sum(qty),
totSale = sum(amt)
ProdWeekCube: prodKey, weekKey, totQuantity. totSale, …

Figure 5: Transform Hypercube

key replaces the natural object identifier from the source system; e.g., customer identifier, order number. This transformation step requires a mapping table to record the source object identifier and its corresponding surrogate key. If the object identifier does not exist in the mapping table, a new surrogate key is generated and an entry is added to the mapping table. Conceptually, this is a simple, one-step operation that is implemented by a hypercube macro method, which we refer to as Identity Resolution (Figure 6). The concept of identity resolution is meaningful at the business level but the details of checking and updating the mapping table are not relevant. When the logical model is generated, the macro will be expanded to hypercube operations that are then converted at the logical level to relational algebra and ETL operators so the appropriate mapping tables will be accessed and updated. Figure 6 shows the macro expressed as a series of intrinsic hypercube operations.

```
IdRes (Cube, Id, Map)
// Assume Cube has an Id dimension
// Assume Map has one dimension, Id, and a cell value, "Key"
Ids = slice ( Cube )  on Id
Keys = join ( Ids, Map )  on Id
NewIds = diff ( Ids, Keys )  on Id
// add an item to a cell (push), generate a new key (keyGen)
NewKeys = push ( NewIds, Id, "Key" ) as keyGen ( Id )
// update Map
Map = outer join ( Map, NewKeys )
// add Key to Cube
Cube = join ( Cube, Map ) on Id
```

Figure 6: Identity Resolution Macro (IdRes)

Banding. As another example, a common task in data warehouses is banding, i.e., mapping a numeric value range to a symbolic value such as the value for age becomes child, teen, adult, senior or the value for income becomes low, middle, upper-middle, high. This can be expressed as a single join operation between the extract hypercube, say customers, and a banding hypercube that describes the mapping. Although the conceptual model could present banding as a hypercube join operation, as with surrogate key generation, it is more meaningful to define an intrinsic macro operation, banding, that takes as arguments, the extract hypercube and relevant value and the banding hypercube and banding dimension. The implementation of the banding macro in terms of intrinsic hypercube operations is shown in Figure 7.

```
BandDim (Cube, Val, Map)
// Assume Cube has an dimension, Val, to band
// Assume Map has one dimension, Val, that is the lower
//      bound of each band and a cell value, "Label"
Vals = slice ( Cube )  on Val
Vals = merge ( Vals )   // eliminate duplicate values in dim
Bands = join ( Vals, Map )  on Val where max ( Vals <  Low )
// update cube cells with label for dimension Val
Cube = join ( Bands, Cube ) on Val
```

Figure 7: Banding Macro

Create load hypercube. Given banding and surrogate key generation as building blocks, one can envision how a more general matching operation could be implemented. For example, suppose the data warehouse records customer details in multiple objects; e.g., one warehouse dimension for customer identity, a second for customer demographics, and so on. Therefore, there would be two corresponding load hypercubes one for each warehouse dimension. Suppose the extract phase produces a single hypercube containing all the customer details. We have seen how the identity resolution macro can be used to obtain a surrogate key from the customer identifier. However, a more general matching operation is needed for customer demographics that would involve, for example, banding the customer age or income, searching the demographic dimension for a match, adding a new entry if none is found, and so on. The result would be a surrogate key for customer demographics added to the customer hypercube. Then, two load hypercubes would be populated by hypercube slice operations over the customer hypercube (see Figure 8).

```
// Assume CustCube: custKey, custId, name, addr, age,
//                          income, …
// Need to create load hypercube for
//          Warehouse Customer Dim and Demographics Dim
DemoMatch  ( CustCube )  // adds demographic key to CustCube
CustLoadCube = slice ( CustCube )
                     on  custKey, name, demoKey,…
DemoLoadCube = slice ( CustCube )
                     on demoKey,  ageBand, incomeBand, …
```

Figure 8: Creating Load Hypercubes

Load hypercube. As a final example, data warehouse dimensions may be static or slowly changing. For example, days of the week and months of the year do not change. The list of states in the U.S. may change rarely while the list of cities may change occasionally. On the other hand, the list of products and/or customers may change with some regularity. For dimensions that change, it is a business decision if the warehouse should record details of the changes (the history of the object) or not. When a customer address changes, it may not be useful to track the previous addresses. If a product price changes, it may be important to track when the price change occurred and the previous price. There are several techniques for maintaining history for slowly changing dimensions (e.g., see [6]).

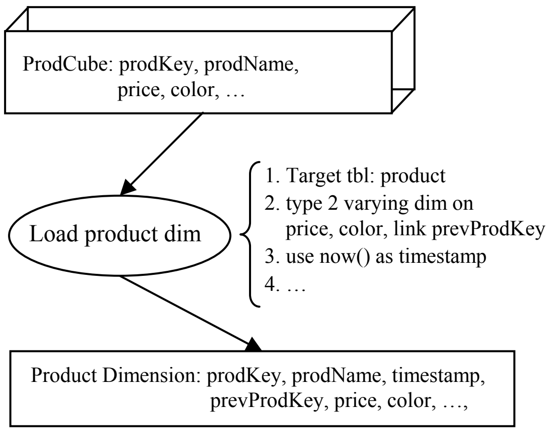
ProdCube: prodKey, prodName, price, color, …
Load product dim
1. Target tbl: product
2. type 2 varying dim on price, color, link prevProdKey
3. use now() as timestamp
4. …
Product Dimension: prodKey, prodName, timestamp, prevProdKey, price, color, …,

Figure 9: Load Hypercube

As with surrogate key transformations, at the conceptual level, there is a single, macro operation for maintaining the history. Annotations are used to hide the detailed hypercube operations so that a correct logical model can be generated. An example of a hypercube load operation using a macro for varying dimensions is shown in Figure 9.

4. RELATED WORK

The current state of the art is to use informal techniques to capture business requirements and create a conceptual model. A typical design might be expressed using some combination of MS Visio diagrams, spreadsheets, and text documents. ETL designers devise their own methodologies and best practices so there is wide variation in how conceptual models are expressed. A few ETL conceptual models have been proposed in the past. Some are based on ad hoc, drawing notation and others use notation like BPMN or UML (e.g., see [1] and [7]). Our experience shows that business users are not typically familiar with such notation and wish to have simpler, more meaningful to them representation. Also, the production of later design stages like the logical model is not clear in those approaches. Therefore, adapting such approaches in our system was not useful.

In business process modeling, the business artifact centric approach has been proposed (e.g., see [3]). Business artifacts refer to objects in the OLTP databases that are manipulated by the operational business processes throughout their lifecycle. Hypercubes in contrast, follow the cube paradigm which is inherent in business analysis and decision support.

Previous work has described how to translate hypercube operations into the relational algebra required at the logical level (for example, see [2] and [10]). We adapt those results in compiling our intrinsic hypercube operators.

5. CONCLUSIONS

This paper presents a solution for eliminating the disconnect in ETL projects between the capturing of business rules and the creation of a logical model. It describes the use of hypercubes as the basis for a conceptual model and shows how to model extract, transform and load phases using intrinsic hypercube operators along with high-level macros built on other operators. This approach has several advantages over the ad-hoc techniques currently in use for conceptual modeling. It provides a formalism that is familiar to business users and analysts who are the domain experts responsible for defining business requirements and validating that the flow meets the objectives. Second, it facilitates direct translation to a logical model through conversion of the hypercube operators to relational and ETL operators. This reduces design and development time and improves accuracy because the conceptual model can be read and understood by domain experts.

6. ACKNOWLEDGMENTS

Our thanks to our colleagues Umeshwar Dayal, Malu Castellanos, and Ravigopal Vennelakanti for their feedback and collaboration, and also many thanks to Paul Watson, Paul Urban, John Bicknell, Rom Lin-hares, Heather Wainscott, Tom Harrocks, Werner Rheeder, and Blair Elzinga for sharing with us their insights and experience regarding the ETL practice and needs in real-world, large-scale projects.

7. REFERENCES

[1] Z. E. Akkaoui and E. Zimányi. Defining ETL worfklows using BPMN and BPEL. In *DOLAP*, pp. 41-48, 2009.

[2] Rakesh Agrawal, Ashish Gupta, Sunita Sarawagi. Modeling Multidimensional Databases. In *ICDE*, pp. 232-243, 1997.

[3] Kamal Bhattacharya, Cagdas Evren Gerede, Richard Hull, Rong Liu, Jianwen Su. Towards Formal Analysis of Artifact-Centric Business Process Models. In *BPM*, pp. 288-304, 2007.

[4] Umeshwar Dayal, Malú Castellanos, Alkis Simitsis, Kevin Wilkinson. Data integration flows for business intelligence. In *EDBT*, pp. 1-11, 2009.

[5] Umeshwar Dayal, Kevin Wilkinson, Alkis Simitsis, Malú Castellanos. Business Processes Meet Operational Business Intelligence. In *IEEE Data Eng. Bull*. 32(3), pp. 35-41, 2009.

[6] Kimball, R., et al. *The Data Warehouse Lifecycle Toolkit*. John Wiley & Sons, 1998.

[7] Sergio Luján-Mora, Panos Vassiliadis, and Juan Trujillo. Data Mapping Diagrams for Data Warehouse Design with UML. In *ER*, pp. 191-204, 2004.

[8] Alkis Simitsis, Kevin Wilkinson, Malú Castellanos, Umeshwar Dayal. QoX-driven ETL design: reducing the cost of ETL consulting engagements. In *SIGMOD Conference*, pp. 953-960, 2009.

[9] Alkis Simitsis, Kevin Wilkinson, Umeshwar Dayal, Malú Castellanos. Optimizing ETL workflows for fault-tolerance. In *ICDE*, pp. 385-396, 2010.

[10] Panos Vassiliadis. Modeling Multidimensional Databases, Cubes and Cube Operations. In *SSDBM*, pp. 53-62, 1998.

[11] Kevin Wilkinson, Alkis Simitsis, Umeshwar Dayal, Malu Castellanos. Leveraging Business Process Models for ETL Design. In *ER*, 2010.