An OPM-based Method for Transformation of Operational System Model to Data Warehouse Model

Dov Dori
Technion – Israel Institute of Technology
Haifa, 32000, Israel
dori@ie.technion.ac.il

Roman Feldman
Technion – Israel Institute of Technology
Haifa, 32000, Israel
feldmanr@tx.technion.ac.il

Arnon Sturm
Ben Gurion University of the Negev
Beer Sheva, 84105, Israel
sturm@bgu.ac.il

Abstract

Data warehouse modeling is a complicated task, which involves both knowledge of business processes and familiarity with operational information systems structure and behavior. Several modeling techniques were suggested to utilize the operational system structural or behavioral model in order to construct a data warehouse conceptual model. However, these techniques are hardly applicable for modeling large-scale systems, as they require acquaintance with the business processes and ability to select relevant transactional entities. They require multiple manual actions because discovering measures and relevant dimensional entities are unassisted, and they usually disregard the process perspective. In this paper we propose a technique for data warehouse model construction based on the Object Process Methodology, and demonstrate it on a case study emphasizing its advantages over other techniques.

1. Introduction

Data warehousing (DW) [10] is a rapidly growing area in the information systems field. Many companies, which invested resources in a new generation of operational information systems during the 1990s, now try to gain added value from their systems by applying different tools for information analysis, decision support, and strategic planning, jointly known as "Business Intelligence". These tools analyze massive amounts of operational data in order to produce the information needed for decision making processes while providing the end user with autonomy and flexibility in browsing and analyzing the operational data. The requirements from these tools led to data warehousing—a new approach to data analysis built on top of multidimensional databases [10] that go beyond normalized relational databases. Off-the-shelf data warehousing products supply solid physical data models and On-Line Analytical Processing (OLAP) foundation. Yet, conceptual and logical design of the data warehouse remains the task of system analysts of the organization running these data warehouses. Data warehouse design usually follows the Star schema diagram [10]. A Star schema consists of a fact table and a single de-normalized dimension table for each dimension of a data model. The dimension tables can be normalized to create a Snowflake schema, which can then support attribute hierarchies. Both the Star schema and the Snowflake schema present data as a single cube, which is the basic data warehouse structure that allows performance of multidimensional data analysis by the user.

To support the design of data warehouses, several conceptual data warehouse models other than the Star and Snowflake schemas were also developed [5, 6, 15, 17]. These conceptual models assist in modeling, while implementation issues are partially addressed, often implying that the physical database design will lead to the Star or Snowflake architectures. Object oriented data warehouse design was introduced in the framework of the Object Relational View (ORV) approach [7]. The ORV is obtained by a transformation of the Snowflake schema. UML [18] extensions for data warehouse conceptual modeling have also been proposed [12, 14, 16]. OMG [2] also suggested an approach for data warehouse modeling called, the Common Warehouse Metamodel (CWM), which is capable of expressing main multidimensional properties [12].

Most data warehouse modeling techniques refer mainly to the user requirements [10] and advocate building a new multidimensional model regardless of existing models of the underlying operational systems. This approach raises several problems indicated by [3]:

1. User requirements are unpredictable and subject to change over time, so using them yields an unstable

basis for design. When utilizing the operational system design as the basis, one may have better chances of foreseeing future analysis requirements, such as the need for additional dimensions.
2. Incorrect designs are possible if the designer does not understand the underlying relationships among the various data types.
3. Premature aggregation causes loss of information that limits the options to analyze the data in fine-grained resolutions.

These drawbacks of relying on user requirements as a basis for data warehouse design have motivated studies to use operational system models as a basis for modeling, designing, and constructing data warehouses. Since data warehouses depend on the underlying operational systems as their data supply sources, the latter are important for the design of data warehouse conceptual and logical schemas. Most of these methods utilize the structure of the operational system models, leading to the following disadvantages:

1. The data warehouse designer is assumed to be familiar with the organization's business processes,
2. The behavioral aspects of the system are ignored, and
3. Multiple manual transformations are needed to obtain data warehouse model.

The fundamental role of data warehousing is business performance measurement and redesign support [8, 11]. With this in mind, several techniques were offered to create data warehouse models from business processes models. Only one model, [0], creates a data warehouse from relatively generic business process models, and it can be applied to various content areas. Another transformation technique, [9], is content-specific, applied to Rule-Based CRM systems.

List et al. [11] offered a specific data warehouse model aimed to analyze business workflow performance and use a workflow management system as the data source. However, this approach can create data warehouse models that cannot be attained by data loading from operational sources, because it does not take into account the data structure in the operational systems.

To overcome the limitations of existing techniques, in this paper we propose a method based on Object-Process Methodology (OPM) for constructing a data warehouse model from its corresponding OPM-based operational model. OPM was the modeling method of choice primarily because it unifies all system aspects within a single view. The proposed method aims at creating a multidimensional conceptual data warehouse model using both the structural and behavioral aspects of the underlying operational systems. We will also pursue simplification and automation of the DW design process by utilizing knowledge about the business processes in order to analyze their performance.

We present a process-based approach overcoming some of the problems of the existing techniques. The approach is evaluated according to a pre-defined set of criteria.

The paper is organized as follows. In Section 2, we present the Object-Process Methodology as the core element within the proposed method. Section 3 introduces a trading catalog system case study, which is used in our demonstration. In Section 4 we present the proposed method and exemplify its use. Finally, in Section 5 we evaluate our method and conclude.

2. Object-Process Methodology

Object-Process Methodology (OPM) [3] is an integrated approach to the study and development of systems in general and information systems in particular. The basic premise of the holistic OPM paradigm is that objects and processes are two types of equally important classes of things, which together describe the function, structure, and behavior of systems in a single, domain-independent model. OPM unifies the major system lifecycle stages – initiation, development, and deployment – within one frame of reference, using a single diagramming tool – a set of Object-Process Diagrams (OPDs) and a corresponding subset of English, called Object-Process Language (OPL). In OPM, system classes, class attributes, physical devices, human users, and environmental interfaces, are modeled as object classes. An object class can be either systemic (internal to the system) or environmental (external to the system). In an orthogonal manner, a class can be either physical or informatical (logical). An object class can be at one of several states, which are possible internal status values of the class objects. At any point in time, each object is at some state, and objects are transformed (generated, consumed, or affected, i.e., their state is changed) through the occurrence of a process. Unlike the object-oriented approach, behavior in OPM is not necessarily encapsulated as an operation or method within a particular object class construct. A process class can involve any number of object classes rather than be an operation of exactly one object class, as imposed by the object-oriented encapsulation principle. By allowing for the existence of stand-alone processes, rather than often having to twist reality to conform to the encapsulation principle, one can model behavior that involves several object classes intuitively and in a straightforward manner. Moreover, OPM enables the modeler to specify that some of the involved objects are transformed, while others enable the process without being transformed. The modeled behavior is integrated into the system's structure in a single model

that also reflects the system dynamics in a balanced way. Processes are connected to the involved object classes through procedural links, which are classified into enabling, transformation, and event links.

OPM's built-in scaling (refinement/abstraction) mechanisms, which are unfolding/folding, in-zooming/out-zooming, and state expression/suppression, help manage the system's complexity by controlling the visibility of details in any OPD and providing for creating new OPDs. Unfolding/folding is applied by default to objects for exposing/hiding their structural components (parts, specializations, features, or instances). In-zooming/out-zooming is applied by default to processes for exposing/hiding their sub-process components and details of the process execution. A third scaling mechanism, state expression/suppression, enables showing or hiding the states of an object class.

An OPM specification of an operational information system includes a hierarchical set of object-process diagrams (OPD set). The root of the hierarchy, the System Diagram (SD), is an OPD featuring the high-level business processes, their environment and interactions. Lower level diagrams are obtained by applying the scaling mechanisms, mainly in-zooming of the higher-level processes or unfolding of the objects.

We found OPM adequate for the task of transformation operational system specification into a data warehouse specification for the following reasons:

1. OPM's scaling mechanisms and its hierarchical system structure allow presenting the operational system and the supported business processes at different levels of abstraction. This feature allows the designer to easily navigate and select the business process to be analyzed, instead of seeking and discovering measures throughout the (often very large and complicated) Entity-Relationship Diagram (as in most of the methods that utilize the operational system structure to construct a data warehouse).
2. The same scaling mechanism may be used to create cubes at different summation levels. Using processes that are located at lower levels in the hierarchy leads to creation of more detailed cubes with less summation of facts and consequently higher data resolution, and vice versa.
3. Existing methods to create multidimensional structures from operational systems construct dimensions and dimensional hierarchies by progressing from the measures (facts) along all the possible many-to-one relationship chains. While this approach may create correct data warehouse structures, it lacks the ability to collect all the needed data to investigate and analyze the business process measures. Instead, it may either insert irrelevant entities in dimensions or miss relevant entities since their relation pattern differs from the traced one. OPM allows distinction of the business objects relevant to the business functionality at any level of abstraction allowing easy classification of dimensions.
4. OPM enables clear identification of the outcomes of a business process. The outcomes are presented as newly created object(s) by the result link, or affected objects (via effect link). The data warehouse should use the objects resulting from the process to analyze the process performance. Hence, these objects' attributes become the measures in the Star or Snowflake structure.

We assume that the OPD set describing the operational system is built such that upper level processes describe main business processes.

3. A Catalog trading system case study

In this section we present a case study using OPM in order to demonstrate the proposed method for constructing a data warehouse model. A partial OPM model of a Catalog Trading System describes the Customer Handling process. The system supports the following functions: The customer contacts a salesman and places a request to purchase a certain product. To match a customer's request, the salesman searches for suitable products in the company's catalogue and issues a product proposal to the customer. The customer may choose to either accept the proposal, reject it and terminate the searching, or ask for another proposal. In case the customer chooses to wait for another proposal, the salesman repeats his search for products and issues a new proposal for another product. If the customer accepts the proposal, an order is placed in the system to fulfill the accepted proposal. Since the system serves also for purposes of incentive payments, the Bonus Wages File is also updated after each final customer decision.

Figure 1 depicts the System Diagram of the **Customer Handling** process in that system. Following is a description based on the Object-Process Language (OPL) script of this system.

A **Product Catalog** consists of many **Product**s. A **Company** consists of many **Sales Departments**, each of which hires many **Salesmen**. A **Bonus Wages File** consists of many **BWF Entries**. **Client,** which is environmental and physical, handles **Customer Handling** process. **Client** exhibits **Customer**. **Worker**, which is environmental and physical, handles **Customer Handling**. **Worker** exhibits **Salesman**. **Customer Handling** requires **Product Catalog**, **Customer**, and **Salesman**. **Customer Handling** yields **Customer Refusal** or **Order**. **Customer Handling** yields **BWF Entry** and **Customer RFP**.

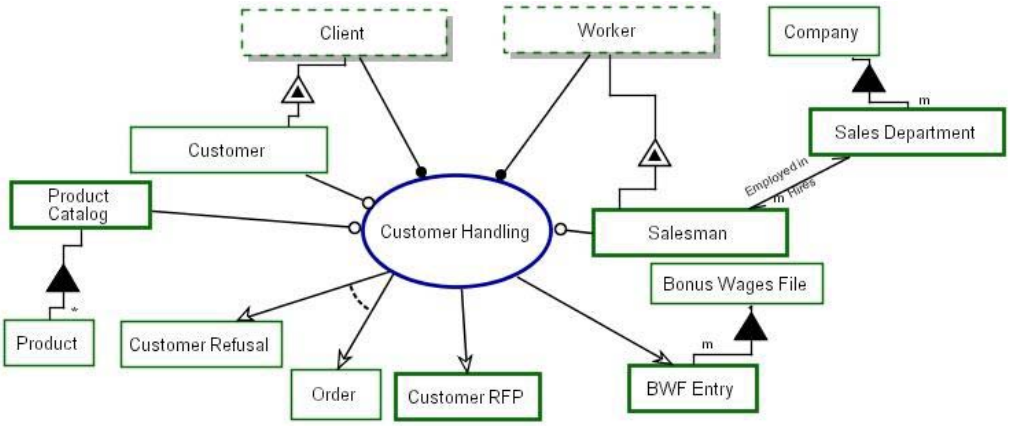

Figure 1. A top-level OPD describing a Customer Handling process in a trading organization

Figure 2(a) is an in-zoomed OPD of the **Customer Request Handling** process. **Customer Handling** zooms into **Customer Request Placing**, **Customer-Salesman Negotiating**, **Negotiation Restarting**, and **Final Decision Handling**, as well as **Product Proposal** and **Customer Decision Product Catalog** consists of many **Products**. **Client**, which is environmental and physical, handles **Customer Request Placing**. **Worker**, which is environmental and physical, handles **Final Decision Handling** and **Customer-Salesman Negotiating**. **Customer Handling** requires **Customer** and **Salesman**. **Customer Decision** can be ask for more products, accept, or final refuse. **Customer Request Placing** requires **Product Catalog** and **Worker**. **Customer Request Placing** yields **Customer RFP**. **Customer Request Placing** invokes **Customer-Salesman Negotiating**. **Customer-Salesman Negotiating** requires **Product Catalog**, **Product**, and **Client**. **Customer-Salesman Negotiating** affects **Customer RFP**. **Customer-Salesman Negotiating** yields **Product Proposal** or **Customer Decision**. **Negotiation Restarting** consumes ask for more products **Customer Decision**. **Negotiation Restarting** invokes **Customer-Salesman Negotiating**. **Final Decision Handling** requires **Product Proposal** and **Product**. **Final Decision Handling** consumes final refuse **Customer Decision** or accept **Customer Decision**. **Final Decision Handling** yields **Customer Refusal** or **Order**. **Final Decision Handling** yields **BWF Entry**.

Figure 2(b) zooms into the **Customer-Salesman Negotiating** process.

Customer-Salesman Negotiating zooms into **Product Proposing** and **Customer Decision Making**. **Client**, which is environmental and physical, handles **Customer Decision Making**. **Worker,** which is environmental and physical, handles **Product Proposing**.
Customer Decision can be ask for more products, accept, or final refuse. **Product Catalog** exhibits many **Products**.
Customer - Salesman Negotiating requires **Salesman**, **Product Catalog**, and **Product**. **Product Proposing** requires **Product Catalog**. **Product Proposing** affects **Customer RFP**. **Product Proposing** consumes ask for more products **Customer Decision**. **Product Proposing** yields **Product Proposal**. **Customer Decision Making** requires **Product Proposal**, **Worker**, and **Customer RFP**. **Customer Decision Making** yields **Customer Decision**.

Figure 2(c) presents zooming into the **Final Decision Handling** process, revealing two horizontally ordered sub-processes: **Order Placing** and **Refuse Recording**. Horizontal ordering means that there are no precedence requirements between the processes, and they may run concurrently. In our case, only one of the sub-processes will run since they are mutually exclusive since each requires **Customer Decision** to be at a different state. Although this is an in-zooming of the **Final Decision Handling** process, **Logical Salesman**

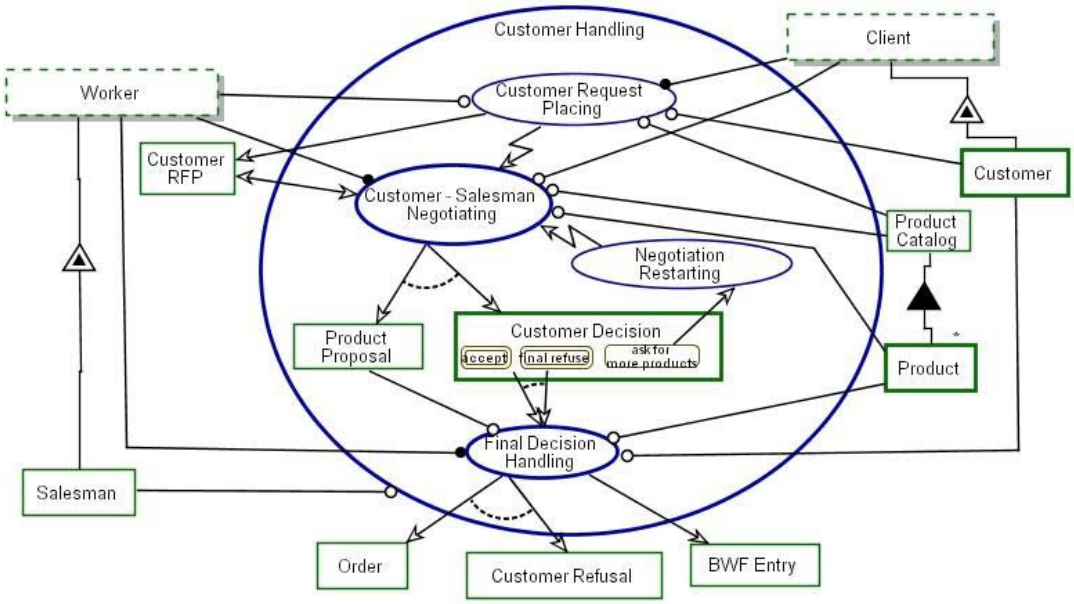

Figure 2(a). Zooming into the Customer Handling Process

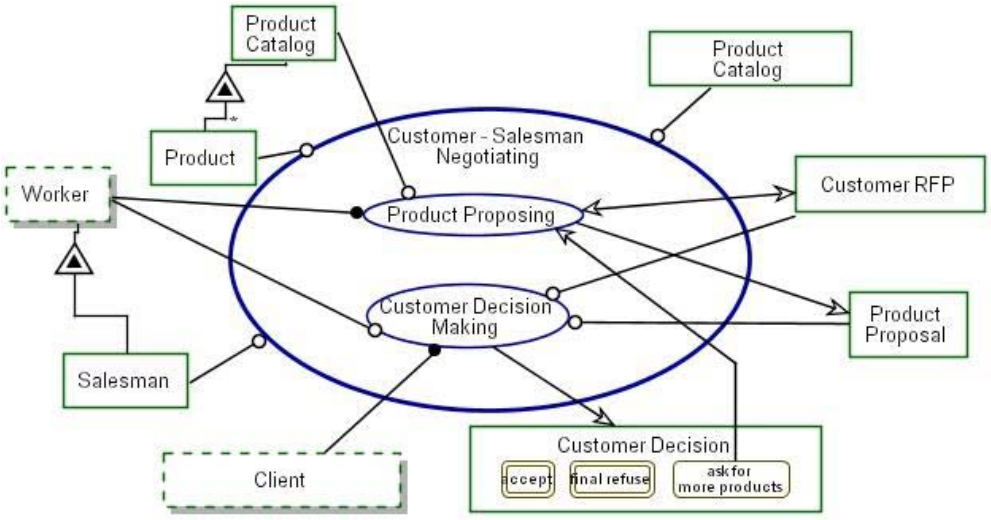

Figure 2(b). Zooming into the Customer-Salesman Negotiating process

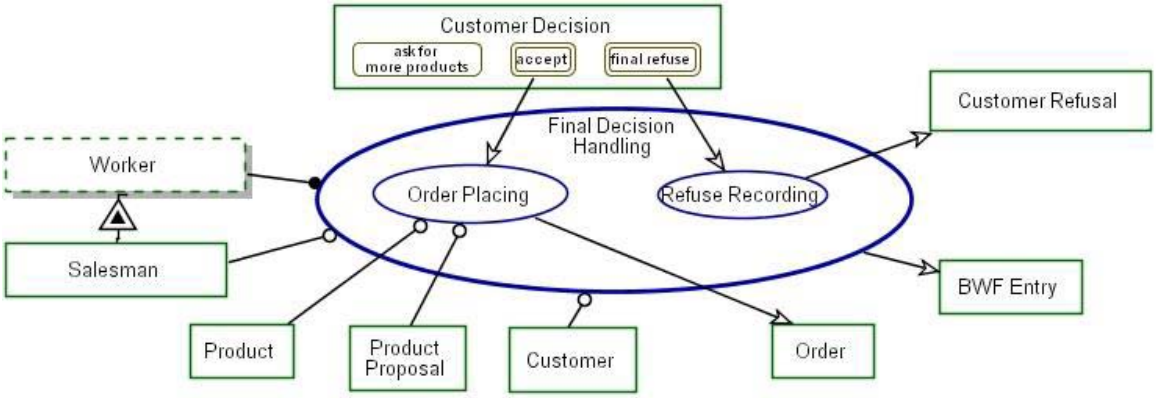

Figure 2(c). Zooming into the Final Decision Handling process

and **Logical Customer** are linked by the instrument link to its outer frame, rather than to its sub-processes. This means that they act as instruments to both subprocesses. **BWF Entry** is linked by the result link also to the outer frame, expressing the statement that any one of the sub-processes can create a **BWF Entry**.

Final Decision Handling zooms into **Order Placing** and **Refuse Recording**. **Worker**, which is environmental and physical, handles **Final Decision Handling** process. **Final Decision Handling** requires **Customer** and **Salesman**. It yields **BWF Entry**. **Order Placing** requires **Product** and **Product Proposal**. **Order Placing** consumes accept **Customer Decision**. **Order Placing** yields **Order**. **Refuse Recording** consumes final refuse **Customer Decision**. **Refuse Recording** yields **Customer Refusal**.

Customer Decision can be ask for more products, accept, or final refuse. Figure 3 is the unfolded OPD of the Catalog Trading System objects showing their structure.

4. Transforming OPM specification into a data warehouse model

In this section we introduce our approach for transforming the operational system specification into a data warehouse model and demonstrate it using the case study specification presented in the previous section.

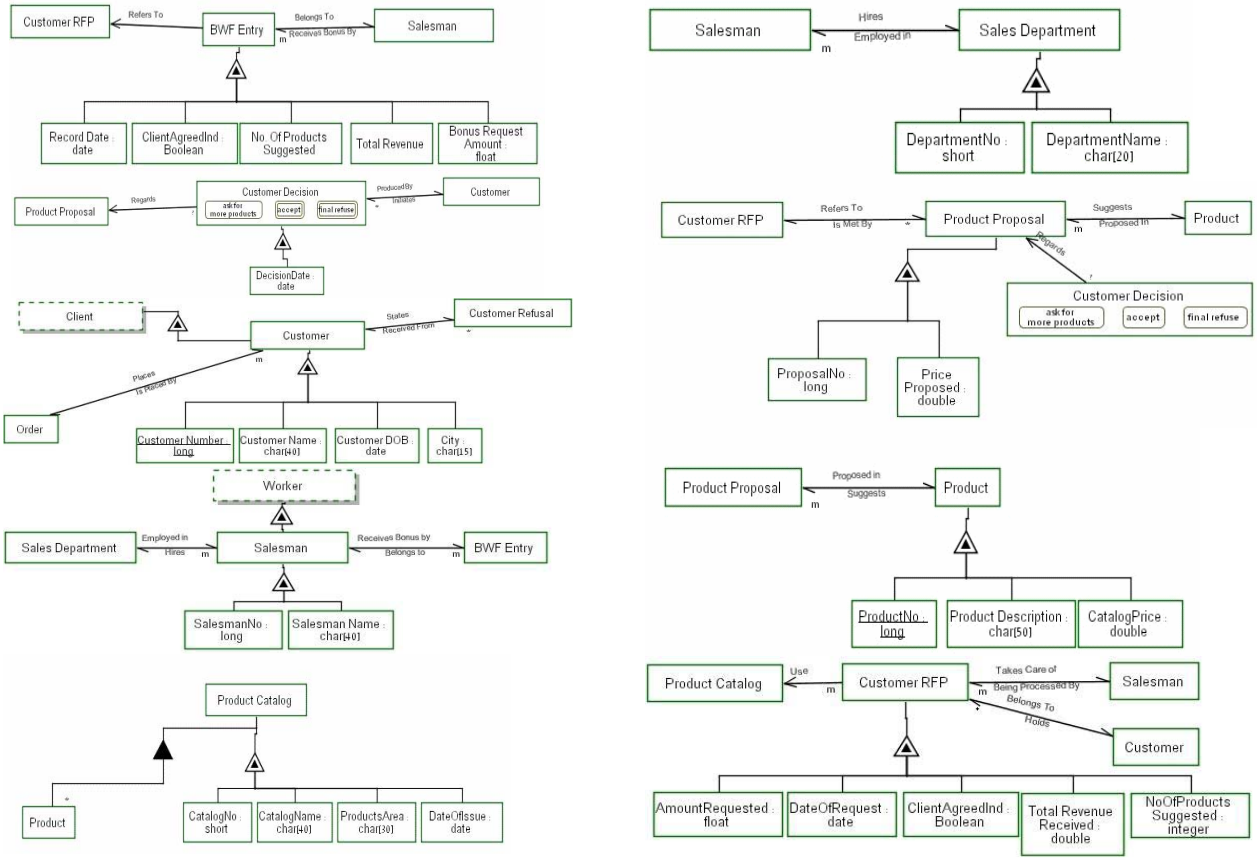

Figure 3. Unfolded OPDs of the Catalog Trading System objects

Our approach towards a construction of a data warehouse from an OPM specification (using the OPDs set) follows an iterative user-guided search process. Each iteration consists of the following steps:

1. Selection of a process by the user.
2. Proposition of possible Snowflake schemas describing the business process by the system.
3. Selection by the user of the appropriate Snowflake schema that fits his data mining needs.
4. Optionally, hierarchical navigation to a more detailed OPDs that zoom into lower level processes and repeating from step 1.

The transformation constructs Snowflake schemas. A Snowflake schema is more informative than a Star schema, as it includes also information about possible navigation in cube's queries. A Star schema can easily be obtained from the Snowflake schema by collapsing the dimensional hierarchies [13].

To demonstrate our approach, we start with the **Customer Handling** process. The construction of

possible Snowflakes from an OPM process is performed in two stages, each consisting of several action rules. Each action rule is followed by an example from the **Customer Handling** process.

Stage 1: **Creating a basic Snowflake schema**:

- Action Rule 1.1: ***Create an empty Snowflake schema for each object transformed, (affected, created, or consumed) by the selected process.*** *A transformed object is identified via the OPM links' semantics specified in [3].*

  Since a data warehouse is designed to analyze the performance of a business process, and since processes in OPM transform objects, we place the transformed (created, modified, or destroyed) objects in the middle of our cube – they will hold the facts.

  ⇒ In our case study, empty Snowflakes are created for the **Customer Refusal**, **Order**, **Customer RFP** and **BWF Entry** entities. In the sequel, we refer to the **Customer RFP** and the **BWF Entry** schemas.
- Action Rule 1.2: ***Convert each object used by the process as instrument or an agent into a dimension***. ***Only objects linked to the process in the current level of detail are to be taken into account.***

  Informatical objects should be proposed as attributes (dimensions) of the cubes. In case that a physical object acts as an agent in the selected process, informatical object exhibited by the physical one is to be proposed as a dimension.

  ⇒ In the case of the **Customer RFP** and **BWF Entry** schemas, **Salesman**, **Customer** and **Product Catalog** are to be added as dimensions. **Client** and **Worker** also participate in the **Customer Handling** process, but they are represented in our schema by their logical representatives.

Stage 2: **Populating facts and dimensions:**

- Action Rule 2.1: ***Propose each quantitative attribute of transformed/created objects as a fact.***

  Quantitative (numerical or Boolean) attributes of the objects created or modified by the analyzed process (or objects which characterize it), which were put in the middle of the Snowflake schema, should be proposed automatically as facts in the fact table.

⇒ Quantitative attributes of **Customer RFP** include the following (see Figure 3): **Amount Requested**, **ClientAgreedInd**, **No. Of Products Suggested** and **Total Revenue Received**. These are transformed into facts in our schema.

⇒ Quantitative attributes of **BWF Entry** are: **ClientAgreedInd**, **No. of Products Suggested**, **Total Revenue** and **Bonus Request Amount**.

- Action Rule 2.2: ***Add all attributes of dimensional objects into the Snowflake schema.***

  Objects that were chosen to be dimensions in the previous stage inherit their entire attribute set as dimensional attributes.

  ⇒ In both schemas, the dimensional objects **Salesman**, **Customer** and **Product Catalog** transfer their attribute sets into the created diagram.
- Action Rule 2.3: ***Define foreign keys in dimensional objects as navigational attributes.***

  Attributes of dimensional objects, which participate in a structural link with other objects in the operational system model, will be proposed as navigational attributes of those objects in the cube. This way, a dimensional hierarchy is created.

  ⇒ The dimensional objects **Customer** and **Product Catalog** have no foreign keys among their attributes. **Salesman**, on the other hand, has **Sales Department** as a foreign key (a many-to-one relationship between **Salesman** and **Sales Department** is present). Therefore, we define **Sales Department Number** as a navigational attribute of the **Salesman** dimension and Snowflake it into the **Sales Department** object. Note that we do not continue this process indefinitely. For example, we do not search for foreign keys in the **Sales Department** object. Infinite snowflaking is not recommended due to performance reasons [10] and may be not supported by industrial OLAP products.
- Action Rule 2.4: ***Add basic dimensions.***

  The following dimensions should be added automatically regardless of the operational system model: time dimension (always), unit dimension (if a fact is expressed through a unit of measurement like a gallon, €, etc…), data packet (at the designer's discretion). Automatically added dimensions should be populated with the relevant data (transaction time, unit of measurement, data upload packet) from the selected transactional data object. Filling the time dimension should be done using the sub-objects of the transformed object which were modeled as date or date-time. If such an attribute does not exist, the time dimension should be populated with the data loading date. Similarly, unit dimension should be established whenever the chosen facts are of quantitative type and require a unit (such as liquid volume in liters or weight in Kilograms).

  ⇒ We add time dimensions to our Snowflakes based on the **Date of Request** attribute of **Customer RFP** and **Record Date** of **BWF Entry**. Unit of measurement dimension is not

added since none of the facts uses a unit of measurement in the original **Customer RFP** and **BWF Entry** objects.

- Action Rule 2.5: ***Remove double object references.***

  Dimensions should be eventually "cleaned up" by the removal of references to objects which are already linked directly to the fact table as independent dimensions.

  ⇒ We have no double object references in our example. In case where action rule 2.3 chooses to Snowflake into an existent dimensional object, a choice has to be made as to whether to leave the object as dimensional (and cancel the snowflaking) or to detach the object from the dimension while keeping snowflaking into it.

Applying the proposed technique on the upper-level **Customer Handling** Process, we receive four similar Snowflake schemass, in which the dimensions are the **Salesman** (with **Sales Department** as its navigational attribute), **Customer**, and **Product Catalog**. The facts come from **Customer Order**, **Customer Refusal**, **Customer RFP**, or **BWF Entry**. Figure 4 presents the results of the transformation process for the **Customer RFP** and **BWF Entry** objects.

Examination of the received Snowflake schemas by a system analyst shows that the cubes obtained by this method may be used for the following expected needs:

1. Discovering potential frauds in the application of the Bonus Wages Policy, like phony customers that call the same salesman just to raise his bonus wages and don't buy anything by the first cube, where **Salesman**, **Customer** and **CustomerAgreedInd** may be used to count suspicious **BWF Entries**.
2. Comparison between salesmen and departments in terms of their relative **Customer Handling** process performance by the first cube, where **Salesman**, **Department** and facts about Sales are present.
3. Identification of returning and serious customers in order to appoint personal account leaders to meet their needs. This is done by using the second cube, which holds information about the customers, their requests, their patience (**No. of Products Suggested**), their eventual decisions on whether to buy or not, and the volume of the sale.

A future use of the created Snowflake schemas may be the ability to perform cross-analysis of Sales performance vs. **Product Catalogs**, or **Customers**, even though they were not requested initially.

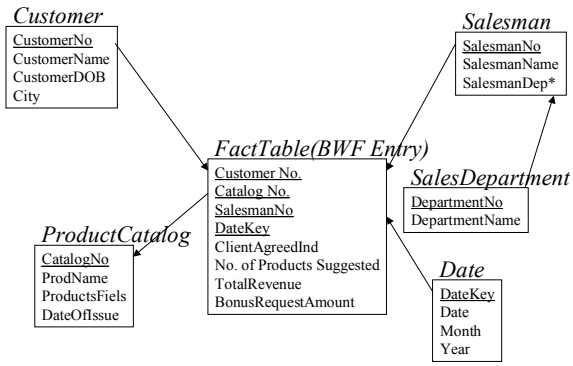

Figure 4(a). A Snowflake Schema of the Bonus Wages File Entries, drawn from the System Diagram

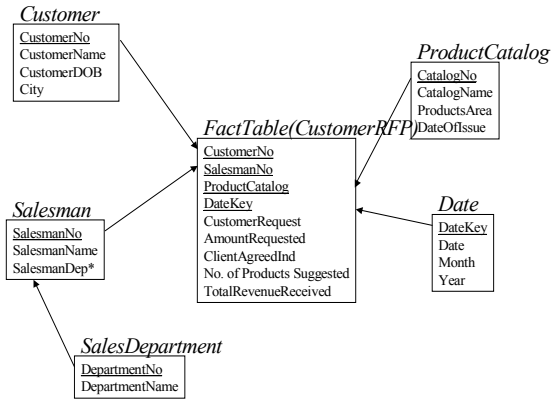

Figure 4(b). A Snowflake Schema of the Customer Requests for Proposals, drawn from the System Diagram

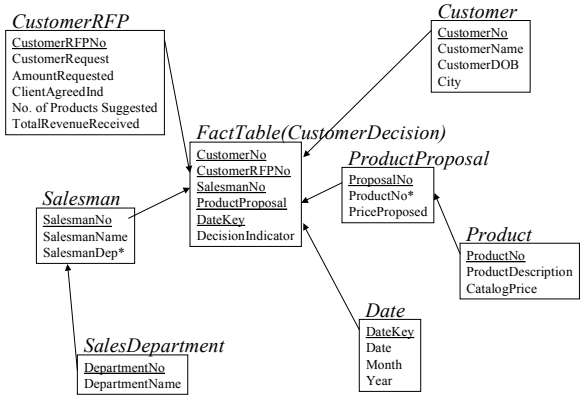

Figure 4(c). A Snowflake Schema of the Customer Decisions For Proposals, drawn from the Customer Decision Making subprocess

Let us now pick the **Customer Decision Making** process and perform the transformation operations. In this process, shown in Figure 2(b), the **Customer** produces a Decision using his own **CustomerRFP** and the **Product Proposal**.

Figure 4(c) shows the Snowflake that was generated this time, where **Customer Decision** became the fact entity, **CustomerRFP** and **ProductProposal** were transferred into dimensions, and the physical objects **Client** and **Worker** contributed their logical representatives as dimensions.
The time dimension was added, while the **Salesman** and **ProductProposal** dimensions snowflaked into **SalesDepartment** and **Product** respectively. Possible uses of this cube would include the following:

1. Discovering potential frauds in the application of the Bonus Wages Policy, like salesmen who always propose the same product regardless of customers' needs.
2. Discovering products whose existence in the catalogue is ineffective, as it does not lead to growth in sales volume.

In both cases, to discover unneeded product offerings, the end user can now find the salesmen whose number of offers for each RFP is relatively high, and crosscheck them against products they proposed to see whether certain products were offered too frequently, or not needed at all.

The process cannot be fully automated, as a system analyst is needed in order to translate customers' requests into relevant processes to be picked up and to perform some of the filtering of the results. However, the construction of the relevant Snowflake schemas is easy, and the resulting structures are more flexible than the minimal cubes needed to meet customers' requests.

5. Summary

In this section we evaluate our approach following a set of pre-defined criteria. The criteria and our related observation regarding the proposed approach follow.

- **Process automation**
  - ***Facts definition:*** the proposed approach provides comprehensive assistance in facts selection, as selecting a process leads to automatic definition of facts.
  - ***Initial model creation:*** Here also, the proposed approach is better than other methods, since action rule 1.2 creates dimensions automatically from the informatical objects used by the selected process.
  - ***Dimensions (Hierarchies) definition:*** This part is also done automatically by searching along the attribute sets of the dimensional objects.
  - ***Refinement:*** As in any other known method, no automatic assistance is provided for schema refinement.
  - ***Summarizability constraints:*** Here also, the summarizability constraints are to be added manually.
- **Applicability to large-scale systems**
  The proposed method inherits the OPM strategy of complexity management [3], enabling the architect to control the detail level of the things described in the OPM model. The transformation is performed on a selected process, which can be at any level of abstraction. The minimal set of actions performed by the designer includes browsing through a hierarchical set of diagrams. While browsing, the designer can perform top-down, bottom-up or middle-out search to find the a process at an appropriate abstraction level, which is then translated into data warehouse structures. To make things even easier, the search process can be performed using the OPM CASE tool, OPCAT, which provides such navigational features. Therefore, the size and complexity of the operational system challenge the designer much less than when using a non-hierarchical representation of the operational system entities, as proposed by other techniques of data warehouse construction using operational system models [0,4,6,13]. The following points need to be considered when using our approach.
- **Required business processes knowledge**
  Following our approach, the data warehouse modeling is a natural extension of the business and operational system modeling. As such, the data warehouse modeler is assisted by the model itself in understanding the business processes.
- **Handling business process Analysis**
  Since the OPM-based operational system model we use reflects the business processes, our approach can handle business process analysis.
- **Ability to map relevant dimensional attributes**
  In our method, relevant dimensional attributes are mapped by their relation to the analyzed process.
- **Compliance with a modeling standard or a formalism**
  We do not claim compliance to modeling standards.
- **Data transformation from the operational system to the data warehouse**
  We do not address these issues directly, but we derive the data warehouse entities from the operational source entities, a fact that brings to simple Extract, Transform and Load (ETL) processes.

In summary, in this work we have adopted Object-Process Methodology as the method for modeling the operational system and providing guidelines for transforming this system specification into a data warehouse schema. We utilize both the structural and procedural modeling aspects of the underlying operational system, taking advantage of OPM scaling

mechanisms and its ability to identify the relevant data entities and outcomes of business processes. Using our method, a data warehouse designer selects processes to be represented in data warehouse structures by hierarchically browsing the operational system diagram set. The system then proposes adequate data warehouse structures based on the abstraction level of the selected process. We demonstrated the proposed approach via a case study and evaluated it by a set of criteria. That evaluation showed that the OPM-based approach is appropriate for the task of data warehouse construction. Our conclusion is that OPM models utilization overcomes existing technique limitations and provides a more comprehensive, easy-to-use and efficient approach for constructing data warehouse models based on existing operational systems models.

Future research will include development of a software tool for assisting in semi-automatic creation of data warehouse structures from an OPM model of the underlying operational system. With this tool, the approach will be examined on real life case studies.

6. References

1. Boehnlein, M. and. Ende, A. U, "Business Process Oriented Development of Data Warehouse Structures", In Proc. DW2000, Friedrichshafen, Germany, 2000.
2. Chang, D.T., and Poole, J.D. "Extending UML for Data Warehousing and Business Common Warehouse Metamodel". OMG's First Workshop on "UML in the .com Enterprise", Nov. 6-9, 2000, Palm Springs, CA.
3. Dori, D., Object-Process Methodology, A Holistic Systems Paradigm, Springer Press, 2002.
4. Golfarelli, M., Maio, D. and Rizzi, S., "Conceptual Design of Data Warehouses from E/R Schemes", In Proc. 31st HICSS, Hawaii, USA, 1998.
5. Golfarelli, M., Maio, D. and Rizzi, S., "The Dimensional Fact Model: a conceptual Model for Data Warehouses", Int. Journal of Cooperative Information Systems, 7, 2&3, 1998.
6. Golfarelli, M. and Rizzi, S., "Designing the Data Warehouse: Key Steps and Crucial Issues", Journal of Computer Science and Information Management, Vol. 2, N. 3, 1999.
7. Gopalkrishnan, V., Li, Q., and Karlapalem, K., "Issues of Object-Relational View Design in Data Warehousing Environment", In Proc. IEEE SMC Conference 1998, pp. 2732-2737.
8. Kaldeich, K., Oliviera e Sa, J., "Data Warehouse Methodology: A Process Driven Approach", In Proc. CAiSE 2004, Lecture Notes in Computer Science 3084, 2004.
9. Kim, H., Lee, T.H., Lee, S., Chun, J., "Automated Data Warehousing for Rule-based CRM Systems", In Proc. 14th Australasian Database Conference (ADC), Adelaide, Australia, 2003.
10. Kimball, R., "The Data Warehouse Toolkit", John Willey & Sons, Inc., 1996.
11. List, B., Schiefer, J., Tjoa, A.M., Quirchmayr, G., "Multidimensional Business Process Analysis with the Process Warehouse", in: Abramowicz, W., Zurada, J. (Eds.): "Knowledge Discovery for Business Information Systems", Kluwer Academic Publishing, Boston 2000, pp. 211-227, .
12. Medina, E., Trujillo, J., "A Standard for Representing Multidimensional Properties: The Common Warehouse Metamodel (CWM)", In Proc. 6th ADBIS, Bratislava, Slovakia, 2002; Lecture Notes in Computer Science, 2435, 2002.
13. Moody, D. L. and Kortink, M. A. R., "From Enterprise Models to Dimensional Models: A Methodology for Data Warehouse and Data Mart Design", In Proc. 2nd DMDW, Stockholm, Sweden, 2000.
14. Mora, S.L., Trujillo, J. and Song, I.Y., "Extending the UML for Multidimensional Modeling", In Proc. UML 2002, 290-304; LNCS, 2460, 2002.
15. Sapia, C., Blaschka, M., Hofling, G. and Dinter, B., "Extending the E/R model for the Multidimensional Paradigm", In Proc.1st Intl. Workshop on Data Warehouse and Data Mining (DWDM), 1998,LNCS , 1552, 1998.
16. Trujillo, J., Palomar, M., Gomez, J. and Song, I.Y., "Designing Data Warehouses with OO Conceptual Models", IEEE Computer, 2001.
17. Tryfona, N., Busborg, F. and Cristiansen, J. G. B., "StarER: A Conceptual Model for Data Warehouse Design", In Proc. 1st DOLAP, Washington DC, USA, 1998.
18. Unified Modeling Language Home Page – www.uml.org.