Object-Oriented Data Warehousing

By

Joseph M. Firestone, Ph.D.

White Paper No. Five

August 7, 1997

Data Warehousing, Distributed Objects, and OOSE

Data warehousing has largely developed with little or no reference to Object- Oriented Software Engineering (OOSE) [1]. This is consistent with (a) its development out of two-tier client/server relational database methodology, and (b) its character as a kind of high-level systems integration, rather than software development, activity. Data Warehousing assembles components, rather than creating them. The initial top-down centralized data warehousing model with its single database server, limited middleware, and GUI front-end, could get by with a pragmatic "systems integration" orientation at the global level of component interaction, while restricting explicit object-orientation to the various GUI, middleware, analysis, web-enabling, data transformation, and reporting tools that comprise the data warehousing arsenal.

The days of two-tier client/server-based data warehouses are gone now. The dynamic of the data warehousing business and its own development of new technology, has caused centralized data warehouses to be increasingly supplemented with data marts, with data stores of diverse type and content, with specialized application servers, with design and metadata repositories, and with internet, intranet, and extranet front-ends. The two-tier client/server paradigm has given way to a multi-tier reality characterized by an increasing diversity of objects/components, and by an increasing complexity of object relations. Moreover this reality is one of physically distributed objects across an increasingly far-flung network architecture. Data warehouse development now requires the creation of a specific distributed object/component solution, and the evolutionary development of this solution over time, rather than the creation of a classical two-tier client/server application.

Methods used in the data warehousing field to arrive at this distributed object solution have been "pragmatic." There are check lists of steps to be followed in developing data warehouses and data marts [2]. There are assertions that Data Warehousing is data- rather than process-driven. There are characterizations of the "Decision Support Life Cycle." But there is no treatment of how to proceed from an OOSE standpoint, in spite of the fact that the developed system will be a distributed objects construct, whose components will be tied together by COM/DCOM, and/or CORBA compliant middleware, and whose behavior will be determined by processes of programmed interaction among these components, in response to user inputs.

Are the pragmatic methods of development used in data warehousing adequate to the challenges facing the field, or is it necessary or desirable to systematically apply OOSE to data warehouse development? The title of this paper is my answer to this rhetorical question. But what are the arguments for leaving behind the

"pragmatic" systems integration approaches and their associated methods, and moving to OOSE? Before I present these, it will be useful to provide an overview of an architectural approach as a baseline for thinking about OOSE.

Overview of an OOSE Approach to Data Warehousing

Jacobson characterized Object-Oriented Software Engineering (OOSE) as involving an architectural approach (foundation concepts and techniques), a method (step-by-step procedures to be followed in applying the approach), a process (providing for the scaling up of the method), and tools (to support the other three components). Here we need to provide only an overview of one architectural approach to OOSE. The overview relies heavily on the previous work of James Martin [3], Martin and James Odell [4], Ivar Jacobson et. al. [5], Stuart Frost [6], James Rumbaugh et. al. [7], and uses an Object-Oriented Information Engineering (OOIE) perspective in developing the OOSE approach. From this point on I will refer to the OOSE and OOIE approaches interchangeably as if they were one and the same. Keep in mind however that OOIE is only one type of OOSE approach, the type I have used here to make the discussion more specific. Many other OOSE approaches are available, and share with the OOIE approach the benefits and advantages over the systems integration approach to be specified later on. The issue of which OOSE approach is the best for data warehousing will not be covered in this paper.

Some OOSE Definitions and Concepts

Some definitions and characterizations of key concepts will help our exposition of this overview of an OOSE approach. In what follows we have tried to maintain consistency with the Unified Modeling Language (UML) version 1.0 draft [8] standard terminology for O-O development and OOSE.

An External User of a business system, is an individual or organization outside the logical boundary of the business area being modeled, who uses a business process or system. For example, a customer is a user external to the General Motors business system. In simply naming the user we imply no particular role or set of actions in connection with the business area. The user is not an abstraction except in the general sense that any concept is an abstraction.

A Business System Actor is a particular coherent cluster of activities adopted by a User in relation to a Business System or Process. These structured sets of activities, or roles played by users, are what we mean by Business System Actors. The Actor concept is an abstraction from the basic notion of User. And note that like the User concept, it also refers to a type of external entity.

A Business System or Process is a sequence of interrelated activities that transforms inputs into outputs. A Business Process may be supported by a subprocess or subsystem called an information system. Users who are external to the logical boundary of the information system and who play coherent roles in the business system are potential **Information System Actors.**

Note that some business system actors may never use the information system directly, but may always interface with others who in turn interface with the information system. The original business system actors are not information system actors because they don't directly use the system. On the other hand, the individuals they interface with may not be business system actors because they may play roles in the business system that place them inside its logical boundaries, e.g., they may be General Motors employees, a type of business object.

On the other hand, these "business objects" who interface with the information system on behalf of business system actors, are actors relative to the information system. That is, they are users who play definable roles in connection with the information system, and they are external to it, even if they are inside the business system.

The enterprise may be divided into business processes and **each business process into planning, acting,**

monitoring, and evaluating subprocesses. Planning means: setting goals, objective, and priorities, making forecasts, performing cost/benefit experiments, and revising or reengineering a business process. Acting means performing the business process. Monitoring means: tracking and describing the business process. Evaluating means assessing the performance of the business process as a value stream.

Strategic goals and objectives of an enterprise are defined in relation to processes and subprocesses. Thus, processes and subprocesses are value streams in that they are oriented toward producing value for the enterprise, and they produce outcomes that are positively or negatively valued. Within each of the subprocesses use cases may be distinguished to begin to specify the system requirements in more detail. Figure One shows how subprocesses interact.

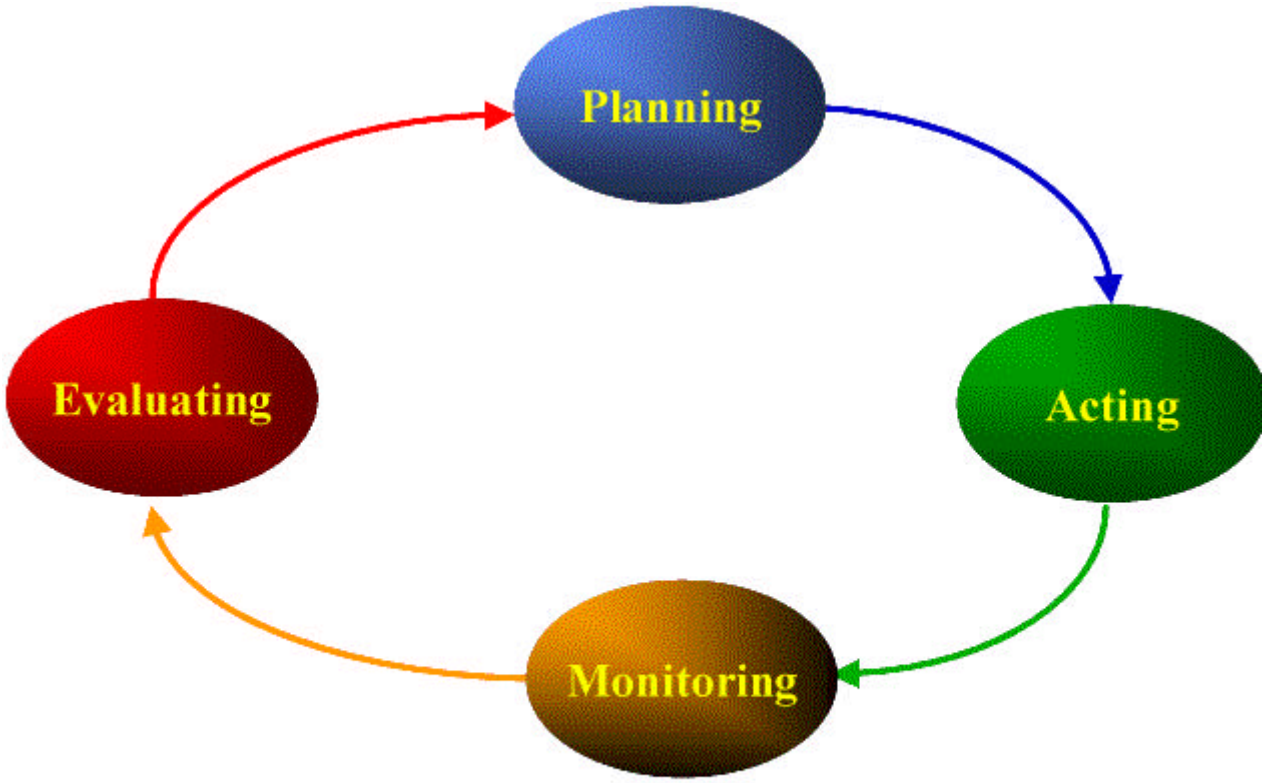

Figure One -- Business Subprocesses And Their Interaction

A Business System Use Case is defined by Jacobson as "A sequence of transactions in a system whose task is to yield a result of measurable value to an individual actor of the business system." [9] A behaviorally related set of bsuiness use cases, in turn, constitutes a business process, and therefore extends over the four subprocesses. Figure Two shows the relationships of business processes, subprocesses, and use cases to one another.

An Information System Use Case, in turn, is defined by Jacobson as "A behaviourally related sequence of transactions performed by an actor in a dialogue with the system to provide some measurable value to the actor." [10] A behaviorally related set of information system use cases, in turn, constitutes an information systems application supporting a business process through its

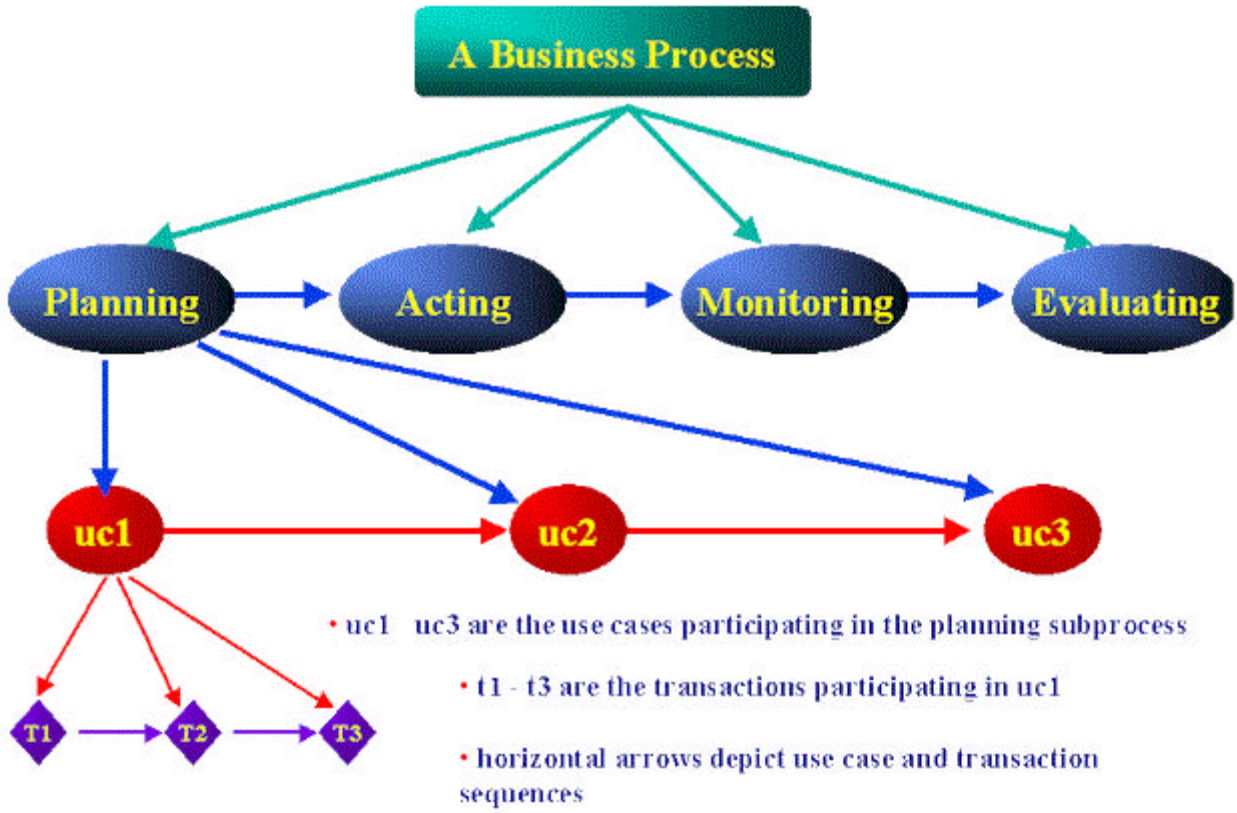

Figure Two -- The Hierarchy of Business Processes, Subprocesses, Use Cases And Transaction Sequences

use cases. This application may or may not extend over the four subprocesses, depending on its scope. Figure Three shows the relationships of business and information system use cases in an application.

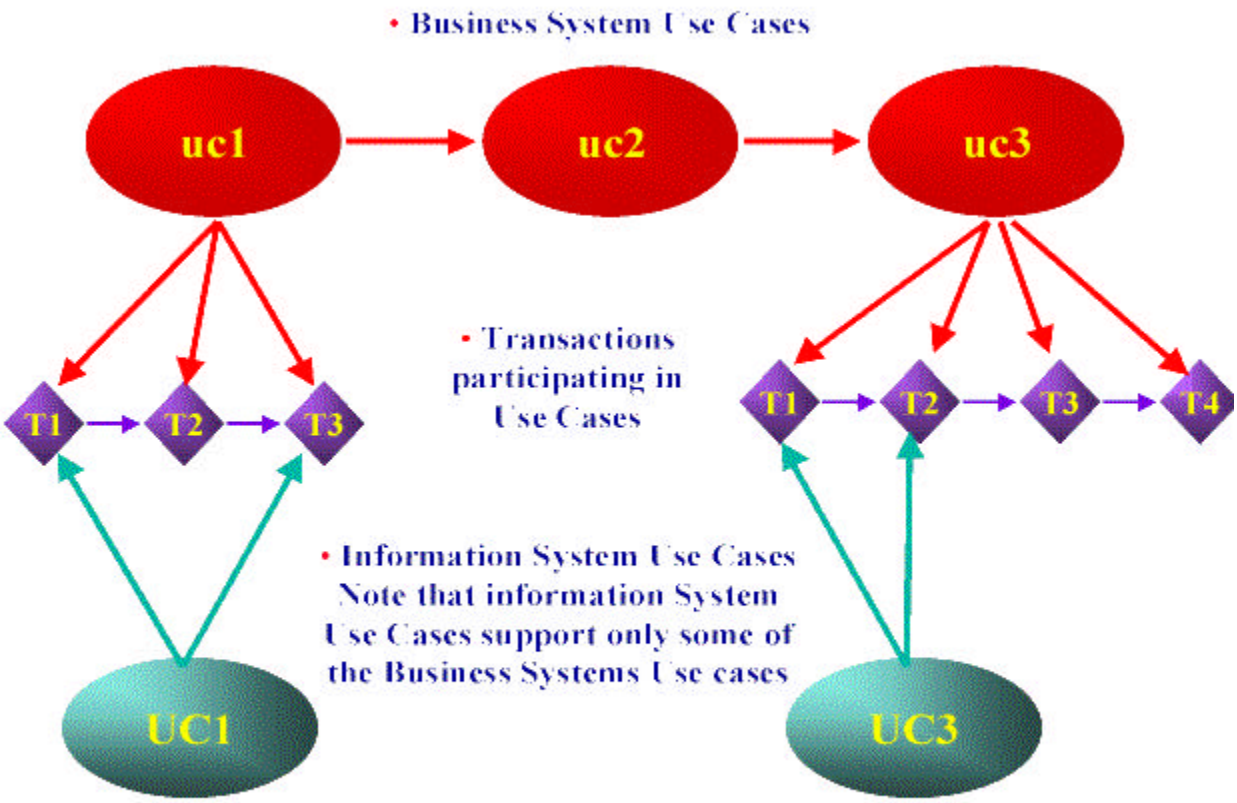

Figure Three -- Relationships of Business System Use Cases to Information System Use cases

A use case is intended to accomplish some tactical objective of an actor, or to aid in accomplishing a tactical objective. The use case concept focuses attention on the user's viewpoint about what the system is supposed to give back in response to the user's input. That is, it is supposed to give back a response or output, and that output will have value relative to a hierarchy of tactical and strategic objectives and goals. Figure Four illustrates the connection between a use case and a hierarchy of goals and objectives.

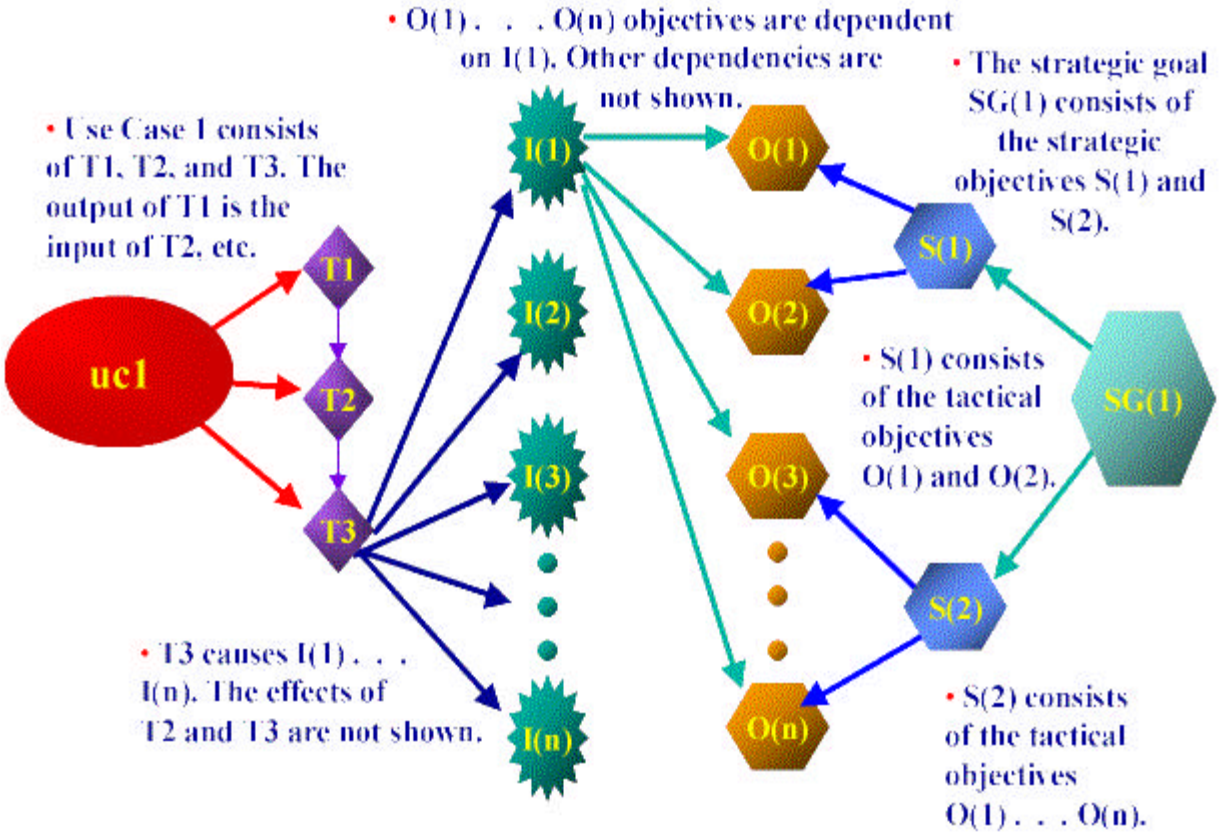

Figure Four -- Connecting a Use Case to Goals and Objectives

Thus, the use case provides an external view of the system from the viewpoint of its users, and their business purposes. ***If we build the system on a foundation of use cases specified by users, and let these lead us to objects and interactions, and given that objects defined at higher levels are traceable down to the level of implemented code, it follows that the whole system architecture will be determined by the use cases and ultimately by the users who specified them.*** If we want to change the system, to incrementally implement or improve it, to redesign it, or to re-engineer it, we can determine new requirements by asking users which use cases they want to see changed, deleted, or added, and we can determine the schedule of iterative development by asking the users about their priorities among the various use cases they have identified.

Use cases are a powerful starting point for re-engineering and they, in the context of JAD facilitation sessions to help formulate and validate them, are perhaps the ultimate measure of accountability to users in software development. Put simply, if it's not in the use cases it shouldn't be in the system, and if it is in the use cases it had better be in the system.

This formal OOSE approach specifies four major classes of models: The Use Case Model; The Object Model; The Object interaction Model and the Dynamic Model [11].

- **The Use Case Model** specifies each use case of the proposed application in terms of the actor performing the use case, the description of the use case, and the specification of the measurable value associated with the use case. Use Case Model diagrams correlate multiple actors to their use cases in the context of a single diagram relating to an application. The description of use cases is used by analysts to arrive at candidate business and information system objects. These objects are then used as material for developing the Object Model.
- **The Object Model** is developed through four sub-models: The Enterprise Business Object Sub-Model; The Local Business Object Sub-Model; The Interface Object Sub-Model; and the Storage Object Sub-Model.

- **The Enterprise Business Object Sub-Model** (EBOM) shows the various object classes in the enterprise system and their relations. An object is an identifiable unit of analysis of which properties may be predicated. The properties of an object are its attributes, its behavior or operations, and its methods. A relation between two objects is a property of the object pair, and is itself an object. Examples of relations are aggregation, or association, or inheritance.

  An enterprise level object is one that is not specific to a single business area. It freely participates in a variety of business area processes and use cases. Customer is such a transcendant object, as is Prospect.

  An object type is a description of a set of objects (instances) sharing the same attributes, operations, and relationships. Object types have subtypes. Also, Classes are implementations of types in software. So, objects are instances of classes as well as types.

  The objects mentioned above, therefore, when generalized, also are examples of object types, and, when implemented in software, of classes. Purchase order is another example of an object type, as is document. Automobile Purchase Order is a subtype of Purchase Order. A well-known abstract typology of objects is that of interface, control, and entity objects [12]. Interface objects are those that manage inputs to, and outputs from, an information system. Control Objects coordinate object interactions in one or more use cases. Entity Objects manage information, resources, and access to them within an information system. All of the examples of object types just given are entity object subtypes. Figure Five provides an illustration of an entity object.

  More generally, object types encapsulate data and therefore have attributes. Objects are instances of object types whose attributes have values. Objects both perform and have operations performed on them e.g., products please peoples' taste buds, products are shipped. Objects also encapsulate methods that perform their operations, and these may be specified in code.

  Another special type of object is a component. A component is a large grained object. It may be composed of clusters of tightly-related small- or smaller-grained objects. Ot it may be a procedural legacy application that has been "wrapped," in a package providing an object interface and encapsulation to the legacy appplication. Components are now enjoying great popularity as the foundation of true rapid application development, and distributed data warehousing is often characterized as a component-based business application. Sometimes analysts are at pains to distinguish components from objects, because objects were associated with earlier difficult and slow mplementations of O-O systems, while components appear to have a very promising future in providing software reuse. In any case, it is clear that components are just large-grained objects, and concepts introduced here to describe objects and their interactions apply to components, as well.

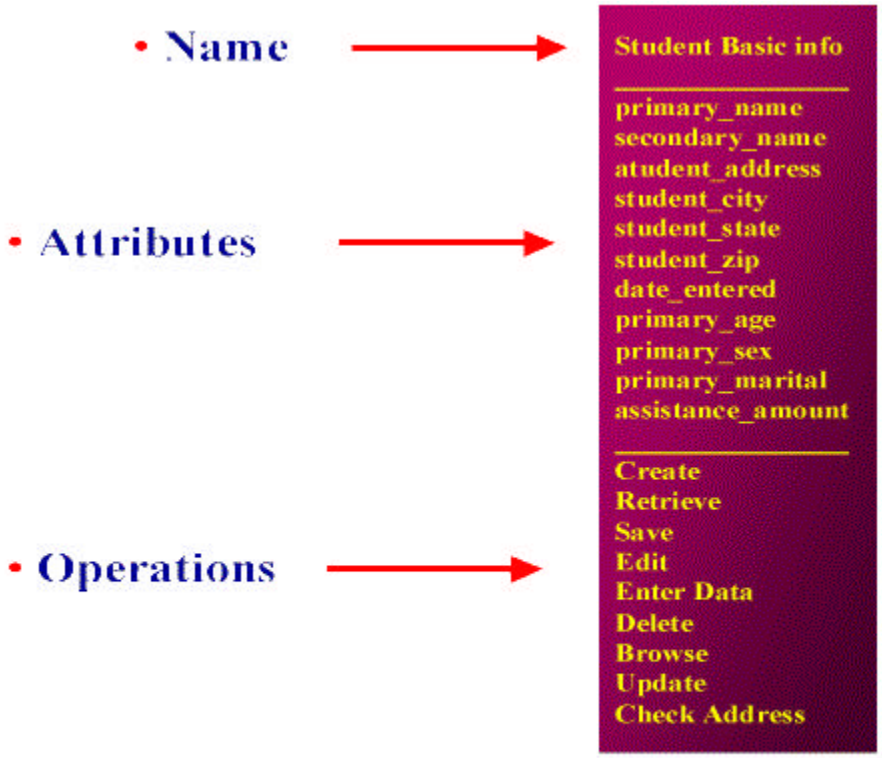

Figure Five -- An Entity Object with Name, Attributes, and Operations

- **A Local Business Object Sub-Model (LBOM)** shows the various objects in a business area system and their relationships. For example, an LBOM of the sales process would identify the objects participating in the sales business process, and name the relations between them. It incorporates the same kinds of relations as in the enterprisewide model, but is specific to the business system under consideration for development or re-engineering. An important aspect of this model is its business rules. An important subsystem of the LBOM is the Information System Object Model; in the example just referred to, it is the information systems sales object model.
- **The Interface Object Sub-Model (IOM)** specifies the forms, controls, input fields, buttons and other objects in the GUI environment that users interact with when they begin, work through and end use cases. It also specifies other interface mechanisms such as mice, keyboards, sensors, printers,etc. The IOM may be specified through diagrams, animated prototypes, or sometimes if the interface is not complex, its GUI portion could be directly constructed in a GUI development environment.
- **The Storage Object Sub-Model (SOM)** specifies how storage is provided for data used by all objects inside the information system. The model may be specified with E-R relationships, data dictionaries, DDL schema, data locations, and client/server transaction relations. Or, if an OODBMS is the means of physical storage, the storage object model is just a revised version of the LBOM, optimized for performance.
- **The Object Interaction Model (OIM)** specifies data and other transaction and communication flows, among the LBOM, EBOM, IOM, and SOM objects or external actors supporting a use case. The OIM is expressed when we specify use cases in detail, through Squence Dagrams correlating use case component tasks with meassage flows between objects. The OIM cross-references use cases and the objects supporting them. It is a key modeling construct linking use cases and the four object sub-models. Figure Six is an illustration of an OIM for a ‘batch cleaning and consolidation" use case, expressed in a Sequence Diagram. In the Sequence Diagram, the arrows represent the flow of task sequences from object-to-object. the blue vertical bars represent the duration of task sequences, while the thin vertical lines are objects suppporting the use case.

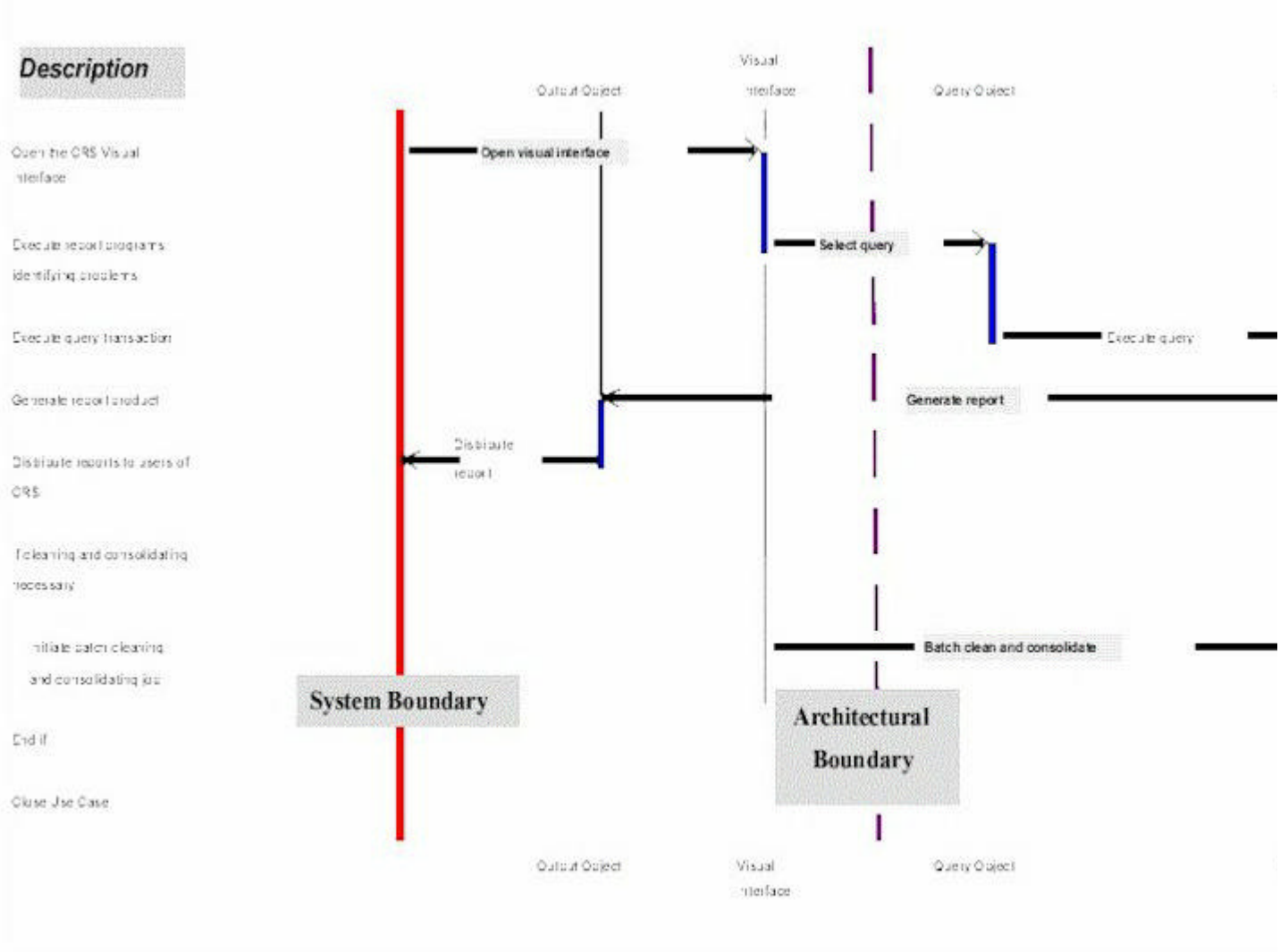

Figure Six -- An OIM Sequence Diagram for a Batch Cleaning and Consolidation Use Case

- **The Dynamic Model** specifies the responses business and information system objects make to events. It models "the Life Cycles" of objects. A set of State Transition Diagrams (STDs) are used to represent the dynamic model. "A state diagram is a network of states and events, just as an object diagram is a network of classes and relationships." [13] One STD is developed for each object type whose dynamic activity we think it is important to represent. Figure Seven presents an example of an STD.

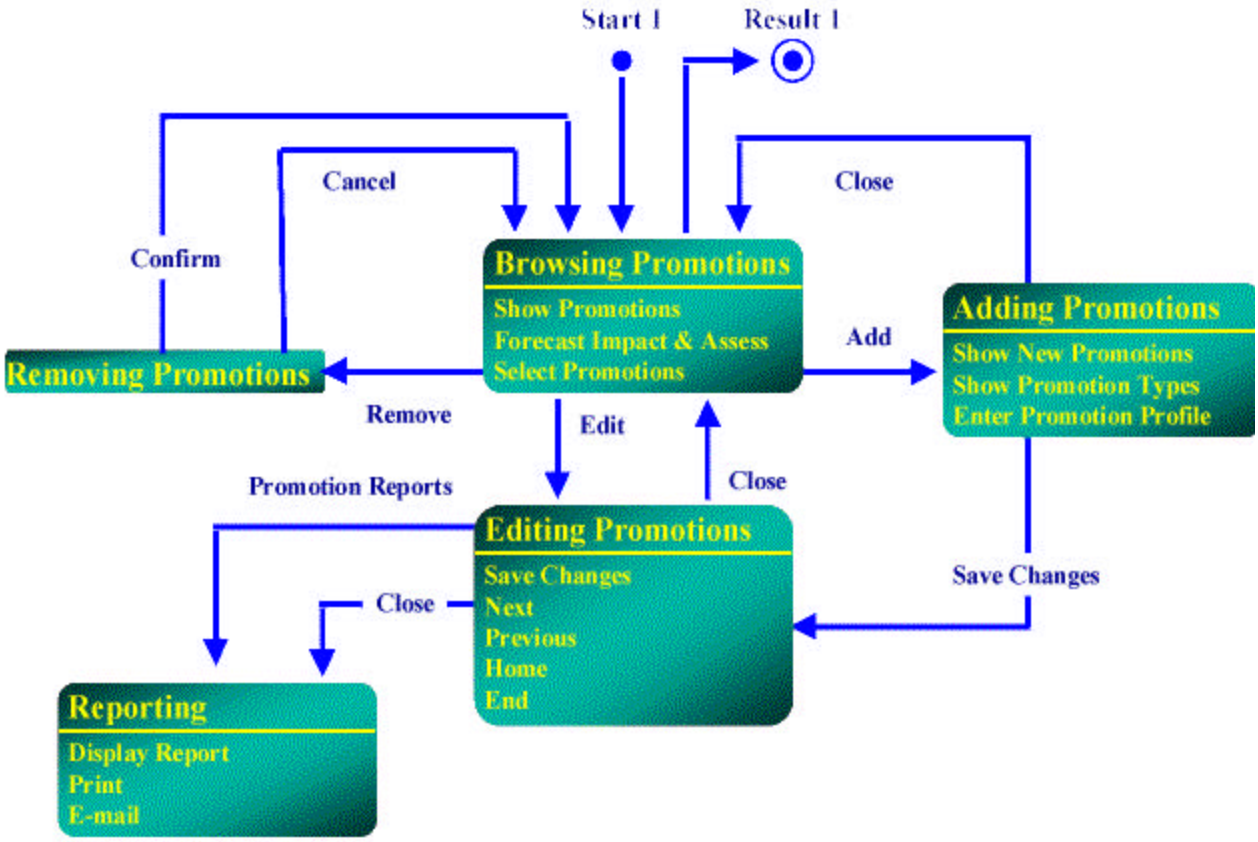

Figure Seven -- A Promotions Controller State Diagram

It shows the dynamics of a Promotions Controller Business Object, supporting data entry, editing, and reporting use cases [14]. The STD shows the various states of the object, and the events that bring transitions from one state to another.

Object-Oriented Information Engineering (OOIE) for Data Warehousing

Martin has defined Information Engineering (IE) as: "the application of an interlocking set of techniques for the planning, analysis, design, construction, and maintenance of information systems *for a whole enterprise* or across a major sector of the enterprise." He further charcterizes IE in terms of characteristics that apply to both traditional and Object-Oriented Information Engineering. These are reproduced in Figure Eight, below.

Figure Eight -- Characteristics of Information Engineering That Apply to Both Traditional and O-O Information Engineering (From Martin, Principles . . ., 1993, P. 242)
IE applies modeling techniques to either the entire enterprise or a large sector of it, rather than on a projectwide basis.
IE provides a variety of development paths or task structures-each generally progressing from planning, through analysis, design, and construction to

cutover, with reiteration to earlier stages to reflect knowledge gained during later stages.
As it progresses through these stages, IE steadily populates a repository (encyclopedia) with knowledge about the enterprise, its system designs, and their implementation.
IE models cross-functional streams of operations (for example, O-O event diagrams) and provides the capability to redesign these streams.
IE creates a framework for developing a computerized enterprise. Separately developed systems fit into this framework. Within the framework, systems can be built and modified quickly.
There is no attempt to achieve central control of application development, but rather to facilitate faster development by decentralized groups who understand their own application needs. The separately developed systems use resources such as networks, data models, and class libraries which are supported centrally.
The enterprisewide approach makes it possible to achieve coordination among separately built systems and facilitates the maximum use of shared data and reusable designs, models, and classes.
IE involves end users strongly at each of the stages by facilitated workshop techniques (Chapter 14) and RAD (Rapid Application Development).
IE facilitates the long-term evolution of systems.
IE identifies how computing can best aid the strategic goals of the enterprise.
IE helps in redesigning the enterprise, business areas, or processes, to remove redundant or unnecessary operations and to take maximum advantage of networks, databases, and automation.

Prior to the 1990s, IE used structured techniques of modeling rather than O-O. Indeed, most of the IE toolsets available today are still oriented to structured analysis and design. According to Martin however, O-O techniques are much more powerful for modeling the enterprise than structured techniques, because they reflect the policies that business people want to use in running the enterprise and building systems that implement these policies.

Like I-E, OOIE approaches system development through the four levels of the information engineering pyramid: Enterprise Planning, Business Area Analysis, System Design, and System Construction. Each of these levels will be specified using the OOSE conceptual framework. The point of view governing development will be that of a comprehensive data warehousing effort whose objective is to construct a distributed processing system with both a data warehouse at the enterprise level and data marts at the level of the business area or

business process level.

Most real projects, however, are likely to be data marts aimed at providing decision support for particular business areas. In applying the framework to such projects the enterprise planning level activities would be truncated. In particular, the EBOM would be developed only to the extent necessary to specify the Object Interaction Model for the data mart's use cases in the course of the business area analysis. Here is the rest of the OOIE framework.

- **Enterprise Planning And Requirements Analysis**
  - Specify an enterprise analytic hierarchy [15] of goals and objectives -- An analytic hierarchy orders a number of disjoint sets of goals and objectives according to the absolute dominance of one set over another. In enterprise planning it is useful to first specify strategic and tactical goals and objectives, then group these objects into disjoint sets, and then order the disjoint sets according to their dominance relations. By using well-known ratio scaling methods [16], particular states of the objects in the analytic hierarchy may be given numerical, ratio-scaled values. Once an analytic hierarchy of this type has been constructed, business processes and use cases may be evaluated in terms of the quantitative contribution of their results to the goals and objectives in the hierarchy. Figure Four included a schematic of a three level analytic hierarchy in a broader context of its connection to use cases, transaction sequences, and their impact. But three level hierarchies, while convenient for illustrations are less than realistic. In an enterprisewide context, hierarchies will greatly exceed that number of levels, and will represent a detailed decomposition of an enterprise's structure of goals and objectives.
  - Create a high level overview of enterprise business processes directed toward achieving strategic enterprise goals -- Identify and describe these critical enterprise business processes, by specifying enterprise level use cases for the planning, acting, monitoring, and evaluating subprocesses. These use cases should be the focus of data warehouse application development, subject to feasibility considerations. They should identify not only the enterprise actors who will use the data warehouse, and describe the sequence of each use case, but also indicate the expected contribution to value of the use case in terms of its relationship to the analytic hierarchy of goals and objectives.
  - Identify and specify Enterprise Level Business Objects supporting the business processes, subprocesses and use cases -- These are objects relevant for more than one business area. Specification means identifying and describing object attributes, behavior (operations), and methods. Note that objects specified will include the goals and objectives of the analytic hierarchy, as these are linked to other objects through business processes and use cases. Incorporate enterprise business policies as business rules.
  - Build an Enterprise Business Object Sub-Model (EBOM) showing relations among the enterprise business objects -- The EBOM should include possible cause-effect relations (a type of association) among objects, as these will be relevant later in identifying causal dimensions of the data warehouse and creating necessary background for data mining activity. Figure Nine illustrates a cause/effect relation among three object classes. The relation is defined by the association class called academic sucess, a ternary relation among Student, Household, and (Student Assistance) Project. How do we know that this ternary relation is at least hypothetically causal in nature? Because academic success is being viewed as the result or effect of the combination of student, household, and project attributes. This hypothesis is the underlying reason for modeling such an association in the first place.

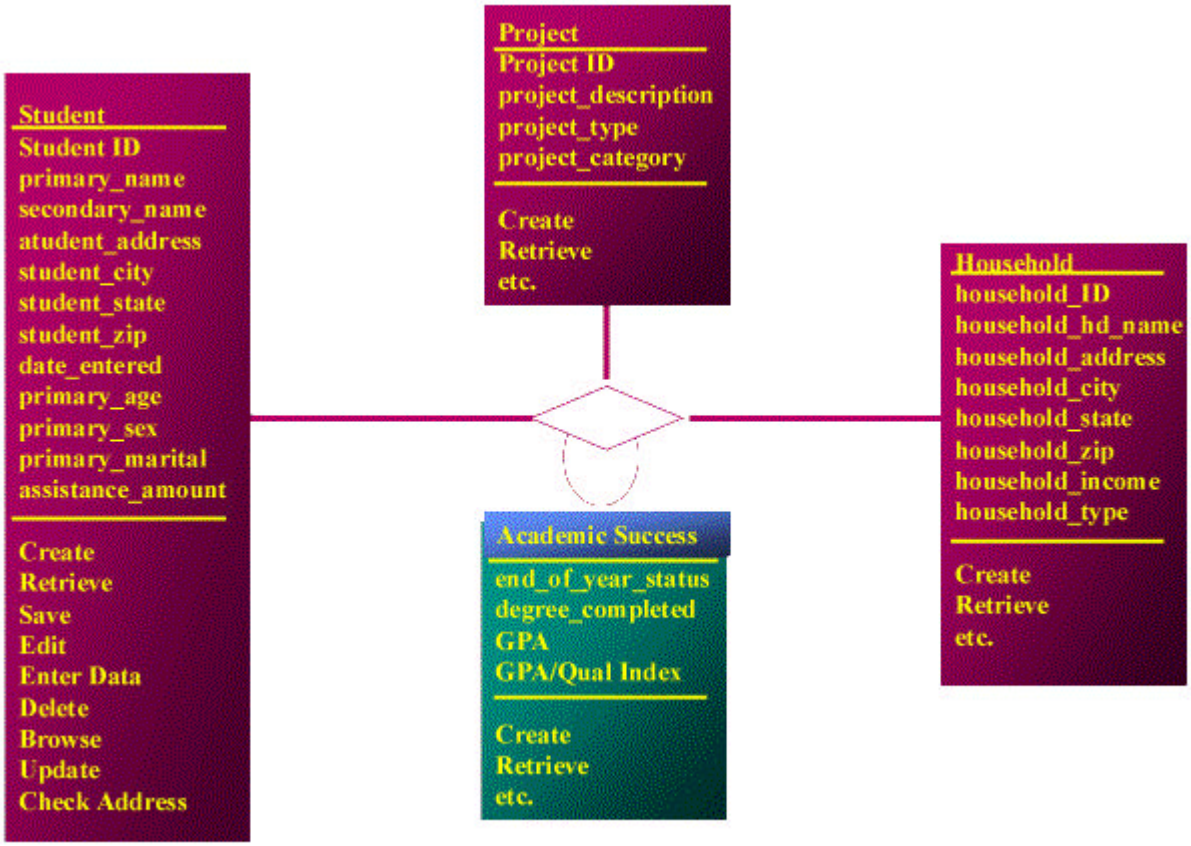

Figure Nine -- A Cause and Effect Association in an Object Model

- Build an Object Interaction Model specifying data and other transaction and communication flows, among the LBOM, EBOM, IOM, and SOM objects or external actors supporting a use case -- The OIM may include objects from the LBOM to support use cases where data or causal, measurement, and predictive models are communicated from the business area data marts to the data warehouse. Such connections support an evolutionary model of the data warehousing information system, in which business area revisions to data and to modeling constructs, might, with appropriate review and controls, be incorporated into the data warehouse. (See my "Data Warehouses and Data Marts: A Dynamic View," [17] for more detail.)
- Build an Interface Object Model (showing how actors interact with the system) -- Prototype the enterprise level IOM, after reviewing the use cases and the OIM. Data Warehouse reporting and OLAP tools are used here to prototype the IOM. JAD sessions may be employed for prototyping and checking consistency with use cases.
- Build the Storage Object Model (SOM) (shows how storage is provided for objects in the EBOM) -- Build a Dimensional Data Model (DDM) if the database(s) being used for the data warehouse is (are) relational, or extended relational. A DDM would also be used with Sybase IQ (a Vertical Technology, or VTDBMS product), because it supports DDMs executed in PowerDesigner 6.0's Warehouse Architect module. Otherwise, use an OODBMS or MDDBMS with a dimensionally-oriented SOM specified to support rapid query performance. Figure Ten depicts mappings from LBOMs to SOMs for various database servers.

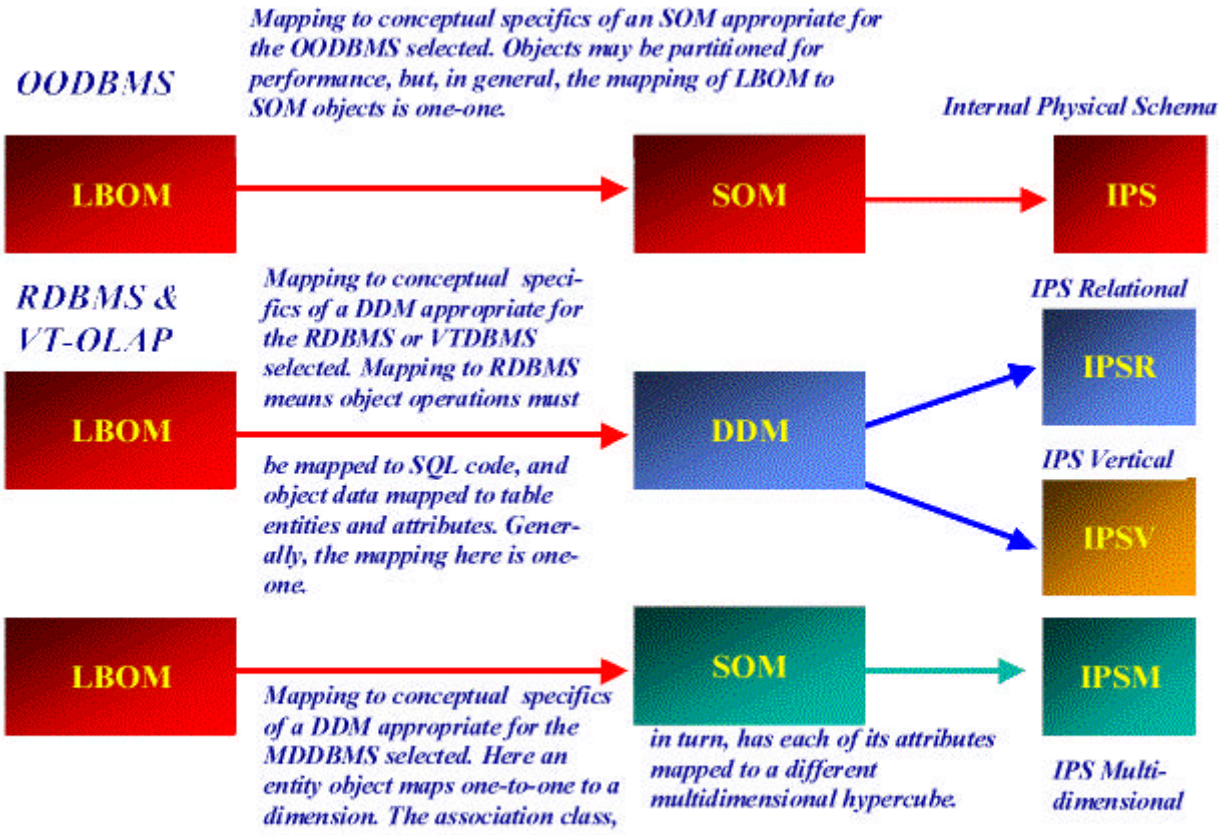

Figure Ten -- LBOM to SOM Mappings

An OOSE approach with its associated object modeling provides a particularly systematic viewpoint for both conventional E-R and dimensional data modeling. (See the later discussion on the use of object modeling in dimensional data modeling.) Some relations developed in the object model provide a specific guide to the tables and relationships in a conventional E-R model, other relations can provide a guide to the tables and relationships in a dimensional data model.

Identification of causal dimensions in DDMs is particularly facilitated by object modeling, because tracing use cases to objects supporting them naturally leads to identifying the entity objects whose attributes may possibly determine the impact of transaction sequences on tactical objectives. The attributes measuring or describing tactical objectives are those of the association classes linking causal dimensions, such as the Academic Success Class of Figure Nine.

- Build a dynamic model (shows how objects respond to events that occur in the course of using the system) -- While the object model describes the statics of a system, such as a data warehousing information system at a point in time, the dynamic model describes changes in objects over time in response to events (messages transmitted from one object to another). ***Causal relationships identified in the Object Model may be explicated in the Dynamic Model by specifying causal events and depicting the changes in state they produce.*** The causal relationships with the most significant effect attributes viewed from a business standpoint, are the ones that are most important to explicate in the dynamic model. We can pick these out by looking at the analytic hierarchy and evaluating candidate relationships or hypotheses, according to whether they express goals and objectives in the hierarchy as effects.

- **Business Area Planning and Requirements Analysis**
  - Starting from the enterprise analytic hierarchy, specify an extension of the analytic hierarchy of

goals and objectives for a business area or process -- The process of doing this is the same as for the enterprise, but this analytic hierarchy is focused on a specific business process such as sales, marketing, or recruiting.

- Create an overview of a business area or process directed toward achieving strategic business process goals -- Identify and describe these critical "local" business processes, by specifying business area level use cases for the planning, acting, monitoring, and evaluating subprocesses. These use cases should be the focus of data mart application development, subject to feasibility considerations. They should identify not only the actors who will use the data mart, and describe the sequence of each use case, but also indicate the expected contribution to value of the use case in terms of its relationship to the now extended analytic hierarchy of goals and objectives.
- Identify and Specify Local Business Objects supporting the business processes, subprocesses and use cases -- These are objects specific to the business area being analyzed for the data mart. Specification means identifying and describing object attributes, behavior (operations), and methods. Note that objects specified, will include the goals and objectives of the extended analytic hierarchy as these are linked to other objects through business processes and use cases. Incorporate business area policies as business rules.
- Build a Local Business Object Sub-Model (LBOM) showing relations among the objects -- These should include possible cause-effect relations between pairs of objects, as these will be relevant later in identifying causal dimensions of the data mart and creating necessary background for data mining activity.
- Build an Object Interaction Model specifying data and other transaction and communication flows, among the LBOM, EBOM, IOM, and SOM objects or external actors supporting a use case -- The OIM may include objects from the EBOM to support use cases where data or models are communicated from the data warehouse to the business area data marts. Such connections would support an evolutionary model of the data mart/data warehousing information system, in which enterprise-level revisions to data and to modeling constructs, might, with appropriate review and controls, be incorporated into the data mart. (Again see my Data Warehouses and Data Marts . . ." for more detail.)
- Build an Interface Object Model (showing how actors interact with the system) -- Prototype the Local business area IOM, after reviewing the use cases and the OIM. Data Warehouse/data mart reporting and OLAP tools are used to prototype the IOM. JAD sessions may be employed for prototyping and checking consistency with use cases.
- Build the Storage Object Model (SOM) (shows how storage is provided for objects in the LBOM) -- The remarks made for the enterprise-level apply here, as well.
- Build a dynamic model (shows how objects respond to events that occur in the course of using the system) -- The remarks made for the enterprise-level apply here, as well.

- **System Design --** Specify an enterprise-level or local business area O-O Model of a specific system in greater detail and develop its physical design
  - Perform detailed design of object types, relationships and interactions -- specify EBOM and/or LBOMs, and OIMs in greater detail. Utilize other design techniques such as Use Case Maps [18] (a visual notation for use cases to aid in expressing and reasoning about large-grained behavior patterns in systems and to provide a high level design tool to develop a better basis for specifying OIDs), and collaboration graphs, to further specify the detail of use cases.
  - Create physical SOM database dictionaries and schema, specifying the physical dimensional data model, or object model (if an OODBMS is used).
  - Create more detailed IOM Prototypes for user validation of the various IOMs defined in enterprise level and/or business area analysis
  - Create more detailed Dynamic Models. Review Use Case Maps and Sequence Diagrams as a foundation for extending dynamic models through more detailed STDs. Included in the STDs should be the dynamics of the process of extracting, transforming, and transporting data to the data warehouse, data mart, and application (such as data mining) servers that will be part of the system. STD design of these dynamics should be followed with low level detailed design using

data warehousing tools such as those produced by Prism, Carleton, ETI, Sagent, and Informatica.
- Perform detailed design of muti-tier client/server distributed processing network architecture.
- Select tools for systems construction including RAD, web client, reporting, multidimensional client and server, data extraction, transformation, and transportation, web server, data mining, data visualization, system management and monitoring, component creation, connectivity/gateway, transaction servers, Object Request Brokers (ORBs), TP Monitors, and other necessary tools. Note that Repository, Object and Data Modeling, RDBMS, and prototyping RAD tools would have been selected for earlier tasks.

- **Construction -- Includes Building, Testing and Installing the System**
  - Extract, Transform and Transport (ETT) data specified in the data model to the system's database server(s), and automate the ETT process for maintenance.
  - Implement ROLAP, MDOLAP, VT-OLAP, OODBMS database server(s). Tune the database server(s) for rapid ad hoc query response.
  - Implement necessary middleware connectivity software such as Transaction Servers, TP Monitors, Object Request Brokers (ORBs), ODBC and JDBC, gateways, etc.
  - Partition the application and balance the processing load across the distributed processing network.
  - Perform data mining and validation activities to arrive at measurement, predictive and forecasting models for the application, and produce derived scores for the data warehouse and/or data mart. A broad scale approach to data mining involving use of a range of diverse data mining tools is recommended. Validation of findings should be comparative across tool categories, and should adhere to explicit criteria for fair comparison.
  - Review the Use Cases, and Analysis and Design results, and implement the set of standard dialogues and DSS reports and analyses based on these and the IOM prototypes. Though the makeup of the dialogue and report set will depend on use case analysis, data warehouses and data marts will need an increasingly wide range of dialogues and/or reports once the capabilities of data mining and data visualization are better integrated into the data warehousing field. A list of categories of standard dialogues and reports could include:
    - analytic hierarchies of goals and objectives;
    - planning scenarios connecting action options to tactical objectives, and indirectly to strategic objectives;
    - work flows assembling action sequences into strategic and tactical plans;
    - model applications to derive new predictions from data using existing models;
    - maintenance, basic information, and complex segmentation responding to quick count and ad hoc queries;
    - impact analysis and/or forecasting of the properties and outcomes of previous activity;
    - assessing outcomes against tactical objectives;
    - assessing forecast outcomes against tactical objectives and forecast tactical objectives;
    - assessing benefits and costs of past and current activities and outcomes;
    - assessing forecast benefits and costs of future activities and outcomes;
    - batch data maintenance and updates;
    - batch cleaning and consolidation of server databases;
    - application partitioning and load balancing; and
    - security and access.
  - Implement the IOM on the web platform, if necessary.
  - Test the application (unit, system, integration, regression, and acceptance testing). The utility of use cases in testing should be noted. In OOSE, use cases are used for integration testing, because the sequence of events specific to each use case involves the participation of objects and components. If a use case works, the objects and components supporting it are working together as required. Use Cases also become acceptance test cases, since they record what the user expects from the system, and ensure traceability of user requirements from analysis through acceptance. Use case testing includes database validation, as well as vaidation of all the object and dynamic

models underlying the application.
- Implement Cutover planning. Includes planning of: physical installation and testing of the LAN/WAN/Web infrastructure, installing the application at the user's site, training, user support organization, maintenance procedures for the application including backup, procedures for expanding and modifying the application.
- Implement Cutover.

Why OOSE for Data Warehousing?

Data Warehousing, like other areas of Information Technology, is a field in the midst of change. The current systems integration approach is associated with the objective of creating a centralized operational data store and DSS read-only server-based application. To meet this objective, it is necessary to extract, transform, and transport data from isolated islands of information to such centralized repositories, and then to retrieve information efficiently and effectively through query and reporting tools. To perform multidimensional analysis, and to meet performance criteria, special methods and tools associated with On-Line Analytical Processing (OLAP) are employed. Multidimensional client, multidimensional server (MOLAP or MDOLAP), Relational OLAP (ROLAP) servers, and most recently, Vertical Technology OLAP (VT-OLAP) servers, are used to help performance in the query and reporting process.

To implement this image of data warehousing is often challenging, and never trivial, but it is essentially a matter of finding and applying the right tools for *modeling the data in the data warehouse*, implementing the front-end process of populating it with clean and reliable data, and applying the right OLAP tool, performance tuning techniques, and reporting tools to both achieve the required performance and extract knowledge from this invaluable enterprise asset. In other words, it is a matter of implementing a good systems integration approach in the context of a good systems integration practice, to link together various data warehousing components and produce applications.

This data warehousing paradigm and its related systems integration approach, is server-centric. It restricts formal requirements modeling to data modeling, only. In viewing data warehousing as data-driven, it rejects process modeling as a necessary activity in requirements analysis. It does not envision the process-based development of data warehouses, data marts and multi-tier architecture, and the continuing evolution of relations among data marts, application servers, and centralized data stores.

A new paradigm, based on distributed processing in enterprisewide networks is needed to effectively conceptualize the dynamics of this process and these relations, and to include them in data warehousing designs and applications. This paradigm is inconsistent with the static systems integration approach oriented only toward a step-by-step view of construction of server-based applications characterized by clean, reliable data and good reporting.

In place of the systems integration approach, the enterprisewide distributed processing paradigm of data warehousing requires an approach characterized *by the use of more formal, systems (not just data) modeling* techniques, better adapted than the systems integration view to handling dynamics in complex systems. The kind of approach needed is a variant of OOSE. Here are more specific reasons why an OOSE -based approach is preferable to the systems integration approach for handling dynamics in complex data warehousing systems.

Capability of OOSE Approaches in Modeling Complex Systems

If we assume that enterprisewide data warehousing systems are increasingly likely to be subject to a process of continuing evolution, necessitating continuing development, it follows that we must choose an approach to data warehousing that is better than the systems integration approach at achieving key change-related goals. Some of these goals and the reasons why the systems integration approach cannot meet them are:

- creating models that suggest how we can manage continuous change in enterprise-level decision support

systems, such as data warehouses;

Systems integration approaches cannot meet this goal because they do not provide a conceptual foundation for process modeling. Change is a process, and so requires process modeling. OOSE as developed earlier, requires Use Case, Object interaction, and Dynamic Models to handle process, and to handle transactions between the Enterprise Level Object Models and Local Business Object Models. This is a rich foundation for modeling and managing continuous change in distributed data warehousing information systems, and stands in sharp contrast to the ad hoc nature of the systems integration approach;

- facilitating the rapid redesign of decision support business processes with maximum reuse of previously developed software technology to reduce both new development and maintenance costs;

  Systems integration approaches are less effective at meeting this goal because they make no explicit provision for cataloging and reusing object/components. Of course, a specific systems integration approach, or data warehousing practice method, may provide for reuse of components. But in OOSE reuse is fundamental, and the development of software technology over time leads to increasingly comprehensive design, application, and metadata repositories for guiding the data warehousing process.

- Coordinating the development of centralized data stores, data marts, and application servers through use of a business process-centric enterprise-level model;

  Systems integration approaches are oriented toward producing a centralized data warehouse, or a special purpose data mart. They are not oriented toward producing a system containing both data warehouses and data marts in interaction. OOSE, on the other hand, is immediately oriented towards viewing data warehouses and data marts as part of an enterprisewide system. The question of their interaction then, is fundamental to an OOSE approach, and modeling this interaction through the Object Interaction and Dynamic Models distinguished earlier, is a necessary step in describing data and information flows between data warehouses and data marts, and in integrating them;

- Supporting rapid application development and maintenance by building both changed and new applications out of existing repository-based objects and components;

Again, the systems integration approach does not explicitly support repositories. In particular, even though specific components integrated by this approach may utilize metadata or design repositories, there are no repositories that contain data about the systems model integrating the multi-tier data warehousing application, and all its components. OOSE approaches, in contrast, lead to the conceptualization and eventual creation of universal repositories containing the various objects, components and models created in the course of implementing an OOSE application. Eventually these repositories will become the center of adaptive and organizational intelligence in data warehousing systems.

OOSE Approaches, Use Cases and Data Warehousing Requirements Analysis

OOSE approaches are better at specifying user requirements than Systems Integration ones. In particular, systems integration approaches seem to move from business process identification to data modeling without specifying the details of the identified business processes. They do not employ Use Case specification and analysis to get at requirements, while this is a central aspect of OOSE.

Use cases are an effective means of getting at requirements. They provide a link, between (a) specification of the goals of the data warehouse, the business processes through which the goals will be fulfilled, the roles or job responsibilities of its users, and (b) the information system that is intended to fulfill their needs. Without this

link, one cannot trace the details of the desired interaction between system actors and the data warehousing information system. Without these details, how can one arrive at a satisfactory model of the data warehousing information system?

In neglecting OOSE approaches and use case specification and analysis, practitioners of data warehousing have therefore, also neglected an important technique for specifying user requirements. In its place, there is no generally agreed upon systematic approach to requirements analysis in data warehousing. Each practitioner has a unique, often informal way of determining data warehousing requirements. These approaches are informative and often very creative; but almost invariably they do not deal with process.

Object Modeling and Dimensional Data Modeling

OOSE approaches also offer advantages over systems integration approaches in supporting Dimensional Data Modeling (DDM) of data warehouses and data marts. The current approach to making the basic decisions in producing a DDM is again, a pragmatic one. The pragmatic approach has had considerable commercial success, but it still makes tight coupling of strategic goals and objectives to the DDM result a matter of art, rather than a product of an explicit method or procedure. While following an OOSE approach will not entirely remove the element of art from the process, the traceability of requirements from strategic goals and objectives, through business processes, through sub-processes, through use cases, and through entity objects to the key decisions in constructing a DDM, is likely to greatly improve the quality of practice in the DDM area. Let’s examine the nature of DDM, develop the argument for tight coupling of strategic goals and objectives to the DDM through OOSE, and see how an OOSE approach to DDM can be implemented.

Dimensional Data Modeling

DDM is the favorite modeling technique in data warehousing. In DDM, a model of tables and relations is constituted *with the purpose of optimizing decision support query performance in relational databases, relative to a measurement or set of measurements of the outcome(s) of the business process being modeled. In contrast, conventional E-R models are constituted to (a) remove redundancy in the data model, (b) facilitate retrieval of individual records having certain critical identifiers, and (c) therefore, optimize On-line Transaction Processing (OLTP) performance.*

Practitioners of DDM have approached developing a logical data model by selecting the business process to be modeled and then deciding what each individual low level record in the "fact table" (the grain of the fact table) will mean. The fact table is the focus of dimensional analysis. It is the table *dimensional queries segment* in the process of producing solution sets. The criteria for segmentation are contained in one or more "dimension tables" whose single part primary keys become foreign keys of the related fact table in DDM designs. The foreign keys in a related fact table constitute a multipart primary key for that fact table, which, in turn, expresses a many-to-many relationship.

In a DDM further, the grain of the fact table is usually a quantitative measurement of the outcome of the business process being analyzed. While the dimension tables are generally composed of attributes measured on some discrete category scale that describe, qualify, locate, or constrain the fact table quantitative measurements.

The decision about the grain of the fact table is crucial to DDM, because once it is selected the dimensions that will relate to it can usually be easily specified, their attributes identified, and their respective grains determined. So it is difficult to overestimate the importance of selecting the fact table grain, or of making this selection on the basis of a procedure *providing for a tight coupling between grain selection and user requirements.*

This tight coupling can be achieved by selecting a fact table grain and associated numerical measurement attribute(s) that will facilitate planning, monitoring and evaluation of the extent to which past, present, planned,

and forecast business outcomes will fail, meet, or exceed strategic and tactical goals and objectives. That is, the choice of fact table grain should flow from strategic and tactical goals and objectives and the need to measure actual outcomes and business performance against them.

OOSE and Dimensional Object Modeling

According to the OOSE approach outlined earlier, business processes produce value streams, and sub-processes, and use cases contribute to value and to strategic and tactical goals and objectives within the analytic hierarchy. Also, use cases are performed by entity, control, and interface objects, and object modeling formulates relationships between objects in the Object Model.

Focusing on entity objects alone, an OOSE approach to data warehousing and data marts utilizing dimensional analysis, would proceed from Use Case Modeling to identification and modeling of entity objects containing attributes specifying quantitative measures of the outcome of the actions taken to implement business use cases. If the use case modeling is done carefully, it helps the analyst to arrive at quantitative measures that are indicated by, and closely coupled with, the use cases, business sub-processes, and business processes that are, in turn, tied to the analytic hierarchy. So, the analyst would emerge from object specification with a set of classes identifying data warehouse and/or data mart outcome measures. Figure Eleven shows the chain of analysis that precedes selection of the fact table grain in OOSE.

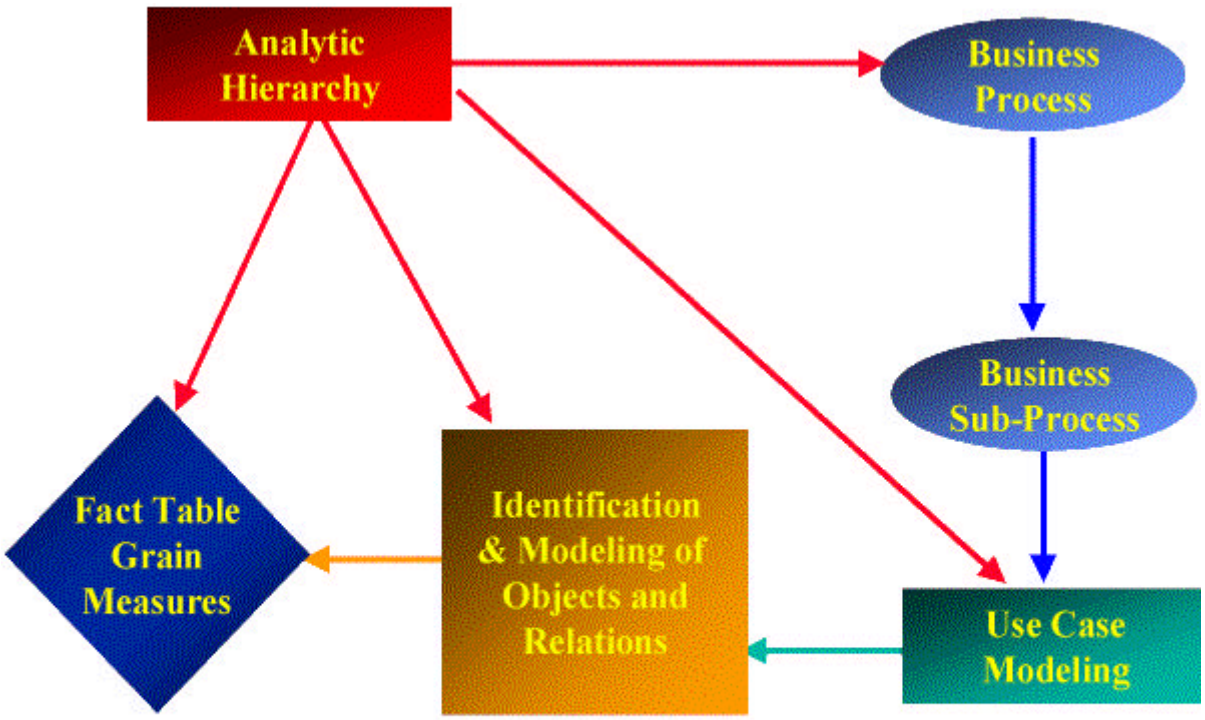

Figure Eleven -- The Chain of Analysis Preceding Selection of the Grain of the Fact Table

If the purpose of the application is to provide a high performance decision support system, object modeling would proceed to model the associations of classes having outcome attributes, with other (dimensional) classes that locate, identify, describe, classify, subset, or otherwise characterize the instances of the outcome classes. This activity produces an entity object model in which outcome entity objects (classes) are related to dimensional entity objects -- in other words, a Dimensional Object Model (DOM). In Figure Twelve, the outcome entity is the association class used in Figure Nine containing educational outcome attributes. Status,

School, and Time, have been added to the three dimensions included in the earlier example of a ternary causal association, in order to complete this DOM.

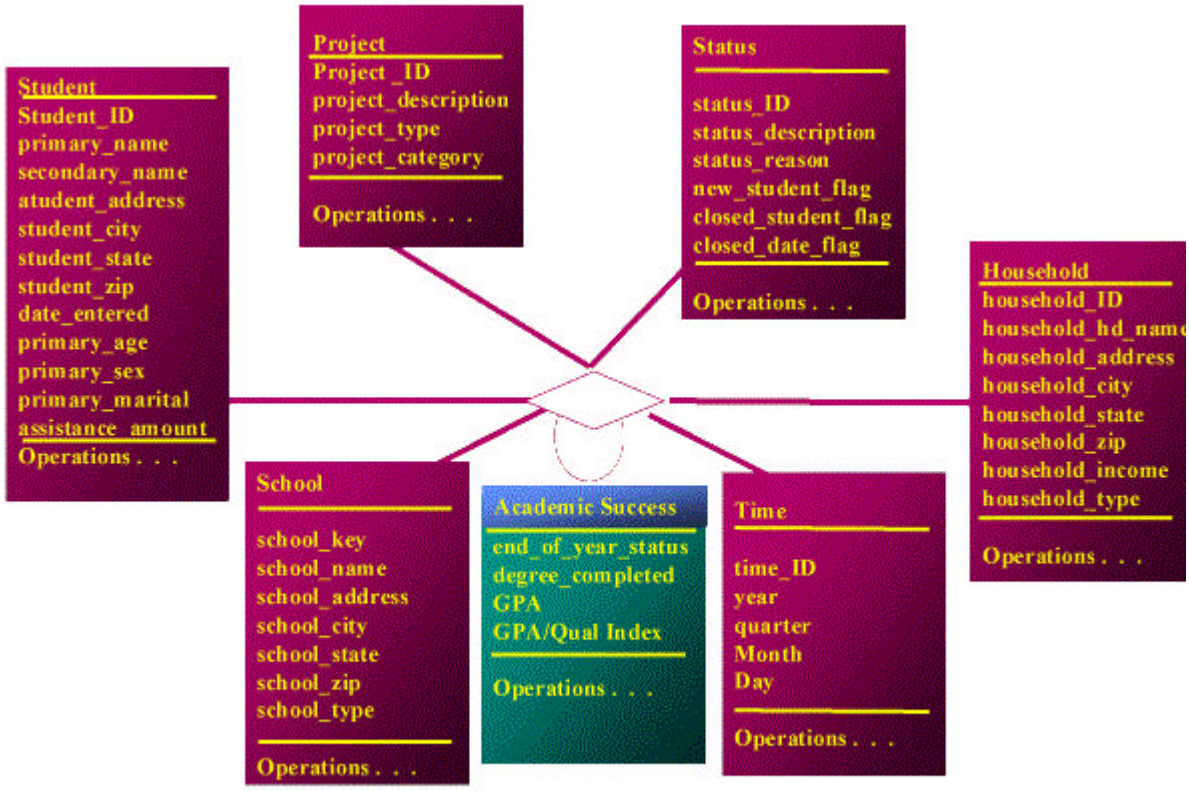

Figure Twelve -- A Dimensional Object Model

But entity object types in OOSE are not equivalent to entities in an E-R model. Nor is a DOM equivalent to a DDM. Entity objects are characterized by behavior providing for access to data, and they encapsulate the methods that produce this behavior in response to external stimuli. Relational entities, on the other hand, represent tables, purely passive containers for data, and since they are not objects, are independent of behavior (operations) and methods.

Still, entity object classes are similar to entities in that they are associated with instances (objects) that have attributes with values, and entity object classes can be mapped onto the entities of a DDM, while object relations can be mapped onto DDM relations, by implementing the required foreign keys and retaining unique object entity IDs in corresponding DDM entities. In short, OOSE suggests an approach to DDM through use cases, and dimensional object modeling of object types and relations, followed by a mapping of the resulting DOM onto a DDM to specify the OOSE Storage Object Model. Figure Thirteen depicts mapping a DOM onto a DDM.

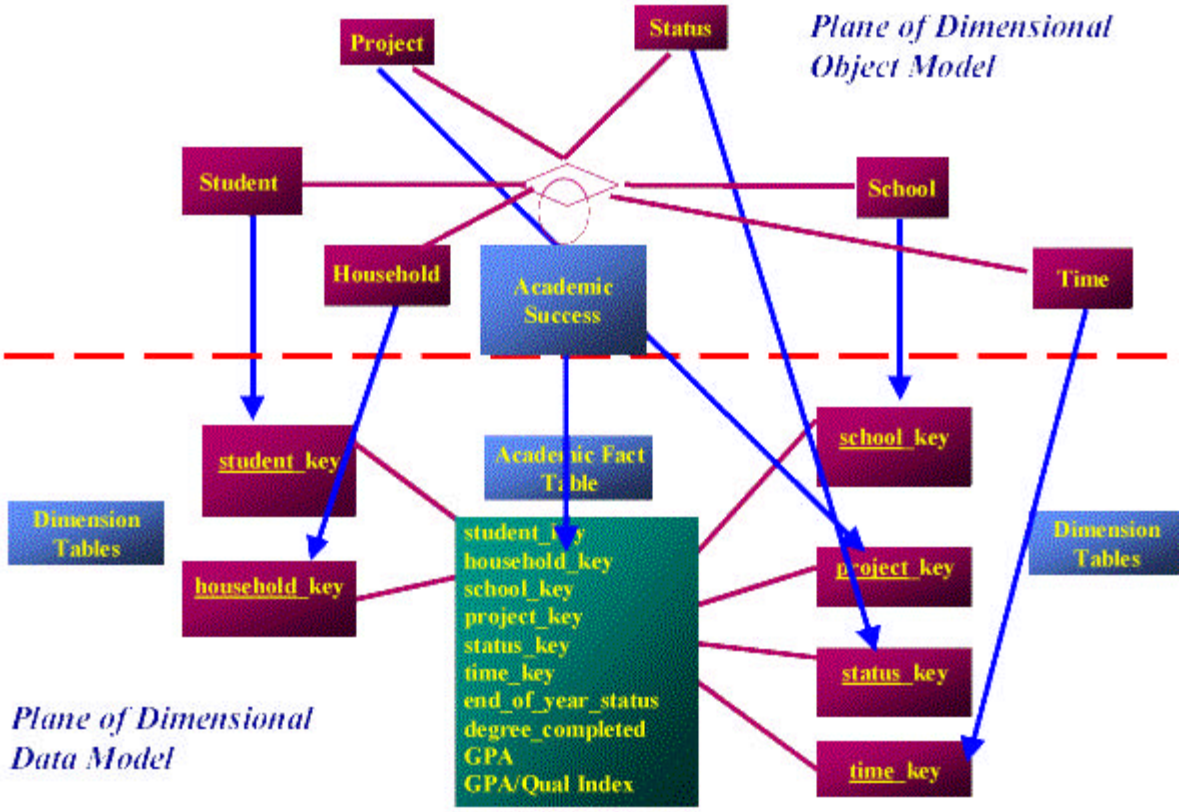

Figure Thirteen -- Mapping a DOM onto a DDM

In the figure the arrows from each entity object to each relational entity represent a one-one mapping of the data aspect of entity objects to corresponding entities. That is, classes identify entities, and their attributes correspond one-to-one with entity attributes. Dimensional Object IDs are mapped directly to Dimensional Entity IDs. The Association Class in the DOM is mapped to the Fact Table in the DDM, which is at the center of the star schema. Finally, the Object IDs become the candidate foreign keys comprising the composite primary key of the Fact Table.

The example should make clear that one can formulate dimensional data models by working through an OOSE approach, arriving at Dimensional Object Models and mapping them onto entities to arrive at Dimensional Data Models. This procedure for arriving at DDMs through OOSE, is an alternative to the pragmatic approaches currently being employed, and ought to be preferred because ot its systematic character and connection to the enterprise's hierarchy of goals and objectives.

OOSE and The Components of Data Warehousing Solutions

The fourth class of specific reasons for an OOSE approach to data warehousing is conceptual consistency with the various components of a data warehousing solution. Though data warehousing solutions have not benefited from OOSE approaches at the global level of the distributed data warehousing information system, the tools used to arrive at these solutions are increasingly object-oriented.

For example, data extraction, transformation and transportation (ETT), tools from Sagent, Informatica, Carleton, ETI, VMARK and others strongly reflect the conceptual outlook of object technology. The same is true of reporting tools such as Impromptu, Business Objects, Brio, DSS Agent, and Forest and Trees; OLAP tools such as PowerPlay, Essbase, Acumate, InfoBeacon, DSS Server, and Express Server; administrative tools such as DSS Administrator, DSS Architect, Intelligent Warehouse, and Universal Directory; and other data warehousing components.

Two partial holdouts from the overall trend toward object orientation at the component level of data warehousing solutions are relational databases and data modeling tools. But these holdouts are also illustrative of the trend. The major relational database vendors Oracle, IBM, Sybase, Informix, and Microsoft have all begun to add object extensions to their relational database products, or to incorporate object-orientation into component-base architecture. While Informix, IBM, and Oracle, are incorporating object extensions into "Universal Servers," Microsoft and Sybase are following a component-based approach to enterprise-level computing. Both have developed Transaction Servers to facilitate inter-component communication in distributed processing systems, and Microsoft, in collaboration with Texas Instruments, has developed a universal repository to store design, application, and metadata components of distributed object/component systems.

In the area of CASE modeling tools, vendors have incorporated various degrees of support for object modeling, ranging from such pure object modeling tools as Rational Rose, Select Enterprise, Select Component Factory, and Paradigms Plus, through tools such as System Architect, and EasyER that provide both Object Modeling and E-R support, to more traditional primarily E-R modeling tools such as ERwin, and PowerDesigner (incidentally, the leaders in providing support for dimensional data modeling and data warehousing). Clearly, the trend here is also toward object-orientation and it is hard to believe that ERwin and PowerDesigner will not provide support for Object Modeling soon, especially given the Sybase/Powersoft emphasis on adaptive component architecture, and their release of various products supporting this distributed processing architecture.

In brief, the components of data warehousing solutions are increasingly object-oriented in their conceptual frameworks, and are increasingly based on tools that require the user to manipulate objects to get results. But when we transcend the component level of analysis and address the data warehousing solution at the system level of analysis, there is currently a conceptual disconnect in the movement from an object-oriented to a systems integration perspective. By adopting an OOSE approach to data warehousing we avoid this disconnect and extend the component level O-O perspective to the data warehousing system viewed as a whole. In doing so, we will create the means to more easily model and design multi-tier data warehousing information systems. We will be able to manage the evolutionary development of data warehouses and data marts in the context of a process-driven, enterprise-level, distributed data warehousing information system.

If OOSE Is So Good, Why Ain’t It Rich?

If OOSE is so much better suited to data warehousing than the Systems Integration approaches, why have OOSE approaches to data warehousing been largely absent from the literature? The answer to this question lies in the similarity of the systems integration approach to OOSE, in the fact of changing data warehouse paradigms, and in the absence of readily available O-O modeling and design tools that would support the DDM schema.

There is an important surface similarity between the systems integration approach and OOSE, which makes the need for OOSE harder to see. Systems Integration, after all, is a process of integrating components through interfaces to construct an application. It is therefore a process that implicitly views components as isolable objects with interfaces. But it is not an approach that relies on abstract modeling of objects and interfaces to gain an understanding of a system and full control over its behavior, but a more pragmatic process of snapping together components into a working application.

As long as data warehousing was viewed as a more-or-less two-tiered server-centered application, implementations could proceed according to a pragmatic systems integration approach having as its objective, production of a stationary data warehousing solution. But now that we increasingly view a data warehousing system as a multi-tier distributed processing system in constant change and evolution, spawning new data marts and application logic servers, we need a more formal OOSE approach which will allow us to view a data

warehousing system as a moving target, an adaptive, intelligent decision support system conforming to the precepts of object-oriented change and learning.

The absence of O-O modeling and design tools supporting DDM may also be a source of problems in applying OOSE to data warehousing. While this situation is not yet resolved, O-O CASE tool designers are beginning to create links with metadata repositories of popular modeling tools supporting DDM. For example, Riverton, Inc.'s HOW 1.0 is a comprehensive Object Modeling tool that interfaces with PowerDesigner, and supports round-trip engineering with that tool. Currently, Riverton supports PowerDesigner 5.0 and the mapping of object models onto E-R models, but not 6.0's Warehouse Architect. it will shortly support 6.0 however, and when it does, the capability to map DOMs onto DDMs will be present in a convenient form.

At present Select Software has a version of its enterprise tool that can export its repository information to Logic Works's ERwin, a CASE tool with support for DDMs. This combination of tools then, can already support dimensional data modeling through an OOSE approach.

DKMS: The Future of OOSE-based Data Warehousing

Recently, I went to Sybases's "Impact Now" seminar, the "roll-out" of their new line of products, and listened enrapt, as they outlined their adaptive component architecture and its relation to data warehousing. One of the things that really interested me about their presentation was their elastic use of the term data warehousing. They used the term a lot. But, on closer examination, they talked mostly about data marts, distributed processing and adaptive component architecture, all under the umbrella of "interactive data warehousing." Were they using the term as a banner to sell products; or was their use of "data warehousing," the verb, a justifiable extension in light of the rapid evolution of data marts and multi-tier architecture?

Those in data warehousing have a semantic, and, at the same time, an economic decision to make. When Bill Inmon defined the data warehouse, he had in mind a centralized repository of enterprisewide data whose purpose was to provide decision support. Decision Support Systems were in bad odor in the corporate world at the time, and a fresh slogan or banner was needed to get corporate sponsorship for comprehensive decision support systems of the kind Inmon envisioned.

As banners useful for attracting corporate support, "data warehouse," the noun, and "data warehousing," the verb, have been remarkably successful. Most large businesses are reportedly either constructing or planning to construct data warehouses or data marts. And the results of the data warehousing movement are all around us in the form of new companies growing at 50 to 100 percent per year, new data warehousing consulting practices at most of the major consulting companies, and new respect for the results and possibilities inherent in decision support processing. This success means there is a natural tendency to want to continue to use the "data warehousing" banner to describe the activity of enterprisewide distributed processing decision support. But this activity may be sufficiently evolved from Inmon's data warehouse concept that the old banner may not stretch far enough to encompass the new distributed processing reality.

As OOSE is increasingly employed in developing distributed process decision support systems, it will become increasingly apparent that we are not developing data warehouses or data marts as much as we are creating Distributed Knowledge Management Systems (DKMSs), the next wave in DSS. DKMSs, the applications corresponding to the adaptive component architecture, are enterprise level "killer apps" that will encompass and integrate all of the components (data warehouses, data marts, data mines, web servers, applications servers, etc.) now being associated with data warehousing.

The means of integration will be object modeling and universal repositories (the prototype of which is the Microsoft Repostory). It is the DKMS, and not the data warehouse, that will ameliorate the islands of information problem in corporate enterprise. Data warehousing is just a way station between database management and distributed knowldge management, as the DBMS -based data warehouse is just a way station

along the road to the DKMS.

Recently, Judith Hurwitz predicted that "The artificial distinction between the DSS and OLTP systems is going to become increasingly blurred during the next five years, as organizations move into what I call the Hyper-Tier environment." I agree. In five years, DKMSs will not be read-only systems for providing Decision Support. Their enterprisewide and distributed processing nature means that data entry and transaction processing tiers of processing could coexist along with data analysis and DSS tiers within the same DKMS. Data gathering and data entry, after all, are also aspects of distributed knowlege management. The goal of On-Line Complex Processing (OLCP), the unification of OLTP, DSS and Batch processing, would finally be realized. Not through the Universal Server, which is too limiting a concept, but through the DKMS and its corporate networking platform.

In short, the future of OOSE -based data warehousing will involve familiar data warehousing activities and components. But these activities and components will be encompassed in a broader adaptive component architecture and in a more encompassing application. This application, the DKMS, will not be data-driven. It will be process-driven, with data playing a vital role. The DKMS will not be produced by traditional software manufacturing vendors, as is the DBMS. No vendor currently has the resources to offer a template for building a DKMS using only their products.

Instead, DKMSs will be custom built by consulting organizations, including the consulting arms of software vendors. OOSE-based modeling, component management, and RAD capabilities will be the enabling technologies for the DKMS. Development environments such as Template Software's SNAP, and Forte Software's, Forte, will move to center stage, as the means of integrating its diverse components. But the continuing motivation for the DKMS will be the persistent drive of corporate and government managers to gain control of their data, and to manage it to produce knowledge-supported decisions.

References

[1] The title of a well-known book by Ivar Jacobson, Magnus Christerson, Patrik Jonsson, and Gunnar Overgaard (Reading, MA: Addison-Wesley, 1992). Jacobson, et.al.'s version of OOSE is a simplified version of their much more detailed Objectory method of software engineering. Our reference here though, is to neither method, but to a more abstract type of conceptual approach to be defined later. The type includes both Jacobson's approach (associated with his method) and many others that may share its assumptions and outlook.

[2] See, for example,Vidette Poe, with Laura L. Reeves, Building a Data Warehouse for Decision Support (Upper Saddle River, NJ: Prentice-Hall, 1997).

[3] Principles Of Object-Oriented Analysis and Design (Englewood Cliffs, N.J.: Prenctice-Hall, 1993).

[4] James Martin and James Odell, Object-Oriented Methods: Pragmatic Considerations (Englewood Cliffs, N.J.: Prenctice-Hall, 1996).

[5] Object-Oriented Software . . ., and Ivar Jacobson, Maria Ericsson, and Agneta Jacobson, The Object Advantage: Business Process Reengineering with Object Technology (Reading, MA: Addison-Wesley, 1995).

[6] The Select Perspective: Developing Enterprise Systems Using Object Technology (Santa Ana, CA: Select Software Tools, Inc., 1995).

[7] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen,

Object-Oriented Modeling and Design (Englewood Cliffs, N.J.: Prenctice-Hall, 1991).

[8] The UML 1.0 is available at http://www.rational.com., as is an alpha version of UML 1.1.

[9] Jacobson, et. al., 1995, P. 343

[10] Jacobson, et. al., 1995, P. 343

[11] Stuart Frost, The Select Perspective . . . Pp. 3-7.

[12] Jacobson et.al. , 1993, Pp. 169-187.

[13] Rumbaugh et.al, Object-Oriented . . ., P. 85

[14] The example is closely modeled on one offerred by Frost, The Select . . ., Pp. 91-96, but the subject matter of this example, is a litle more consonant with data warehousing

[15] Since the early seventies the definitive work on analytical hierarchies has been performed by Professor Thomas L. Saaty, his colleagues and students. With Profesor Ernest Foreman, Saaty developed a software package called Expert Choice, which has gone through many iterations since the early 1980s, and has been widely applied in industry and government both in the US and internationally. Current and background information on the both the Analytic Hierarchy Process, and Expert Choice software is at http://www.expertchoice.com.

[16] Again, see Saaty's, Foreman's and related work at **expertchoice.com**, and also Thomas L. Saaty, The Analytic Hierarchy Process (Pittsburgh, PA: RWS Publications, 1990), and Thomas L. Saaty, Decision Making For Leaders (Pittsburgh, PA: RWS Publications, 1990),

[17] Joseph M. Firestone, "Data Warehouses and Data Marts: A Dynamic View," White Paper prepared for Execuitve Information systems, Inc. Wilmington, DE, March 1997.

[18] See R. J. A. Buhr and R. S. Casselman, Use CASE Maps for Object-Oriented Systems (Upper Saddle River, NJ: Prentice Hall, 1996).

Biography

Joseph M. Firestone is an independent Information Technology consultant working in the areas of Decision Support (especially Data Marts and Data Mining), Business Process Reengineering and Database Marketing. He formulated and is developing the idea of Market Systems Reengineering (MSR). In addition, he is developing an integrated data mining approach incorporating a fair comparison methodology for evaluating data mining results. Finally, he is formulating the concept of Distributed Knowledge Management Systems (DKMS) as an organizing framework for the next business "killer app." You can e-mail Joe at eisai@home.com**.**

[Up] [Data Warehouses and Data Marts: New Definitions and New Conceptions]
[Is Data Staging Relational: A Comment]
[DKMA and The Data Warehouse Bus Architecture]
[The Corporate Information Factory or the Corporate Knowledge Factory]
[Architectural Evolution in Data Warehousing]
[Dimensional Modeling and E-R Modeling in the Data Warehouse]
[Dimensional Object Modeling] [Evaluating OLAP Alternatives]
[Data Mining and KDD: A Shifting Mosaic]
[Data Warehouses and Data Marts: A Dynamic View]
[A Systems Approach to Dimensional Modeling in Data Marts]
[Object-Oriented Data Warehousing]