Distributed Knowledge Management Systems (DKMS):

The Next Wave in DSS

By

Joseph M. Firestone, Ph.D.

White Paper No. Six

August 23, 1997

From Data Warehousing to Distributed Knowledge Management

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 application, design, metadata, and universal 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.

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 [1] 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. It may be necessary to use a new term to describe a new phenomenon.

One possible new term is Distributed Data Warehousing (DDW). Another candidate is Object-Oriented Data Warehousing (OODW), which seems particularly appropriate if our quest is for distributed object/component solutions, and if the solutions will contain data warehouses and/or data marts. But while OODW has the advantage of being descriptive of the type of data warehousing and data marting involved in the activity, and is consistent with the continuing presence of data warehouses and data marts in the distributed object systems, it has two difficulties. First, though it may be an

appropriate term for those involved in software development, it is too methodological for business people. Second, because of its continuing association with data warehousing, it may be perceived as producing a read-only decision support system, since that has been an important distinguishing characteristic of data warehousing. For these and other reasons, I believe a better candidate to describe the new activity is Distributed Knowledge Management; and the applications associated with this activity I call Distributed Knowledge Management Systems (DKMS).

Defining and Characterizing the DKMS

A DKMS is a system that manages the integration of distributed objects into a functioning whole producing, maintaining, and enhancing a business knowledge base. A business knowledge base [2] is the set of data, validated models, metamodels, and software used for manipulating these, pertaining to the enterprise, produced either by using a DKMS, or imported from other sources upon creation of a DKMS. A DKMS, in this view, requires a knowledge base to begin operation. But it enhances its own knowledge base with the passage of time because it is a self-correcting system, subject to testing against experience.

The DKMS must not only manage data, but all of the objects, object models, process models, use case models, object interaction models, and dynamic models, used to process data and to interpret it to produce a business knowledge base. It is because of its role in managing and processing data, objects, and models to produce a knowledge base that the term Distributed Knowledge Management System is so appropriate.

Other reasons for adopting the term DKMS include:

- business knowledge production and management is what business intelligence is all about;
- DKMS plays off DBMS, and therefore capitalizes on a familar term while favorably contrasting with it, i.e. knowledge management is clearly better than mere data management;
- DKMS also highlights the point that data is not knowledge, but only a part of it;
- "DKMS" is a product/results-oriented name likely to appeal to business decision makers (that is, they get valuable and valid knowledge that they can use to gain control and produce results);

DKMSs are the "next wave" of enterprise-level "killer apps" that will encompass and integrate all of the components now being associated with distributed enterprise-level applications. The means of integration will be object modeling and universal repositories. It is the DKMS and not the data warehouse, that will ameliorate the islands of information problem in corporate enterprise. In fact, as distributed data marts proliferate in enterprises, they are filling short-term needs, but also creating a new islands of information problem.

This trend is a matter of heavy debate in data warehousing, an exploding field beset by an inconsistency between its core concept and the direction of its vigorous development. Data Warehousing, as originally formulated, was about data integration and centralization. But the movement toward data marts and multi-tier data warehousing architectures is a movement toward decentralization and disintegration. Current concepts of data warehousing don't handle this conflict well, and can't really transcend it. Data warehousing is therefore, just a way station between database management and distributed knowledge management, as the DBMS - based data warehouse is just a way station along the road to the DKMS.

DKMS Creation

DKMSs may be created from scratch, or from a base of varied applications and relatively unintegrated components. To create one from scratch is an expensive proposition; far more time and resource consuming than creating a data warehouse. DKMSs should therefore be created by integrating already existing components as much as possible, and by adding new components such as data warehouses and data marts where necessary. In addition, an *incremental approach to DKMS creation is best*, in order to get results more quickly and to build support for the longer term effort to create the DKMS.

An incremental approach to the DKMS should focus on segmenting the DKMS by process, subprocess, and use cases. Each increment then, would represent an implementation of a set of use cases found within a process, and subprocess, or perhaps spanning more than one subprocess. I will return to this point when use cases are discussed.

The next part of the discussion presents the tasks involved in developing a DKMS. The perspective taken is general, and can be applied to both creating the DKMS from scratch, and modifying existing legacy systems, including multi-tier systems containing data warehouses and data marts into full-fledged DKMS systems. The perspective covers Planning and Requirements Gathering, Requirements Analysis, System Design, and Construction activity; but a "waterfall" model of development is not being described. Again, the concept of development intended is incremental, with a focus on a limited number of use cases, on prototyping, on reusing and "snapping together" components, and on testing against use cases. The life cycle is one of Rapid Application Development (RAD) following a "spiral" pattern encompassing increasingly broader clusters of use cases as more and more system increments are developed.

Planning and Requirements Gathering

Specifying Enterprise-level Goals and Objectives

The strategic goals and objectives of an enterprise determine and are prior to its primary business processes, in the sense that the processes are value streams whose purpose is to accomplish those goals and objectives. The strategic and tactical goals and objectives may be expressed as an analytic hierarchy [3].

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, [4] 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 One includes 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.

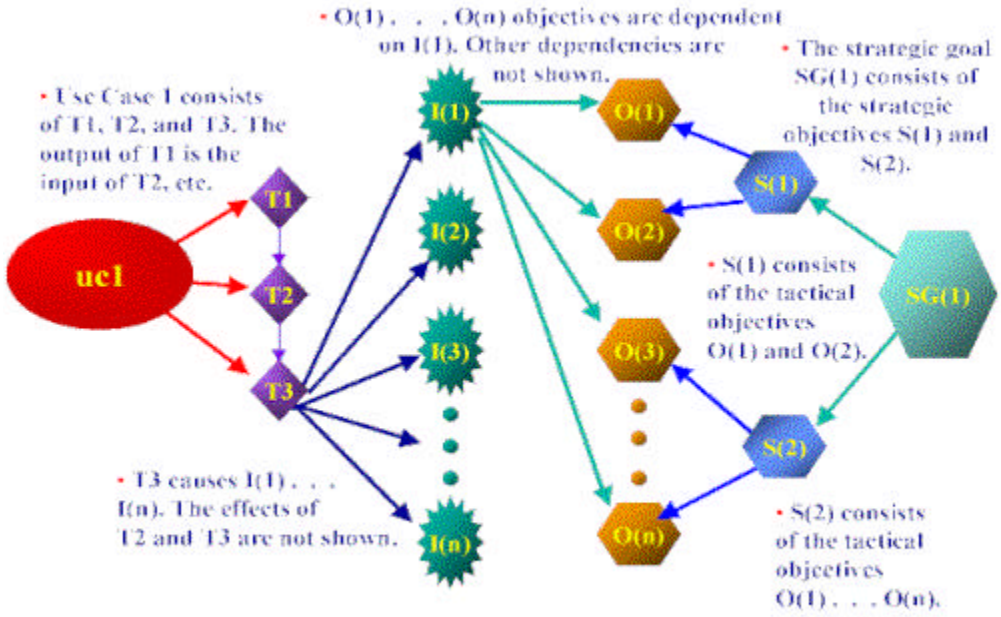

Figure One -- Connecting a Use Case to Goals and Objectives

Defining Major Subsystems

The subsystems of an enterprise are its organizational subdivisions. Primary business processes flow through such organizational subdivisions. An example is provided by the Insurance Industry. In it insurance enterprises normally have Home Offices, Affiliate Field Agencies, and individual agent representatives. All three subsystems are involved in selling an insurance policy.

Specifying Business Processes

A Business Process is a sequence of interrelated activities that transforms inputs into outputs. Processes are value streams in that they are oriented toward producing value for the enterprise, and they produce outcomes that are positively or negatively valued. Different enterprises will have different processes. Figure Two gives one classification of the primary business processes in the insurance industry.

Figure Two -- Business Processes of the Insurance System

Specifying Business Subprocesses

Each business process has four subprocesses: Planning, Acting; Monitoring; and Evaluating. 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. Figure Three lists Subprocesses and activities in the Sales Process of an enterprise.

- Planning
  - Develop or Reengineer Hierarchy of Strategic Selling Goals and Objectives and Tactical Selling Objectives
  - Plan specific sales effort to accomplish tactical objectives
    - Select leads, schedule and make contacts, schedule and make appointments, formulate scripts and select selling techniques considering alternative mixes in light of forecasts and forecast benefit/cost assessments
- Acting
  - Implement Sales Efforts by going to Sales Appointments, making telephone calls, selling at conferences, etc.
  - Gather and record data on sales progress, closes, and losses
  - Create, Reengineer or Enhance Sales Database of contacts, sales efforts and responses
- Monitoring
  - Describe sales responses and derive impact measurements
  - Summarize descriptions and measurements
- Evaluating
  - Assess past and forecast responses against tactical objectives
  - Assess past and forecast benefits VS. costs

Figure Three -- Selling

Specifying Actors

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 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.

Specifying Use Cases

A Business System Use Case is defined by Jacobson [5] 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." A behaviorally related set of bsuiness use cases, in turn, constitutes a business process, and therefore extends over the four subprocesses. Figure Four shows the relationships of business processes, subprocesses, and use cases to one another.

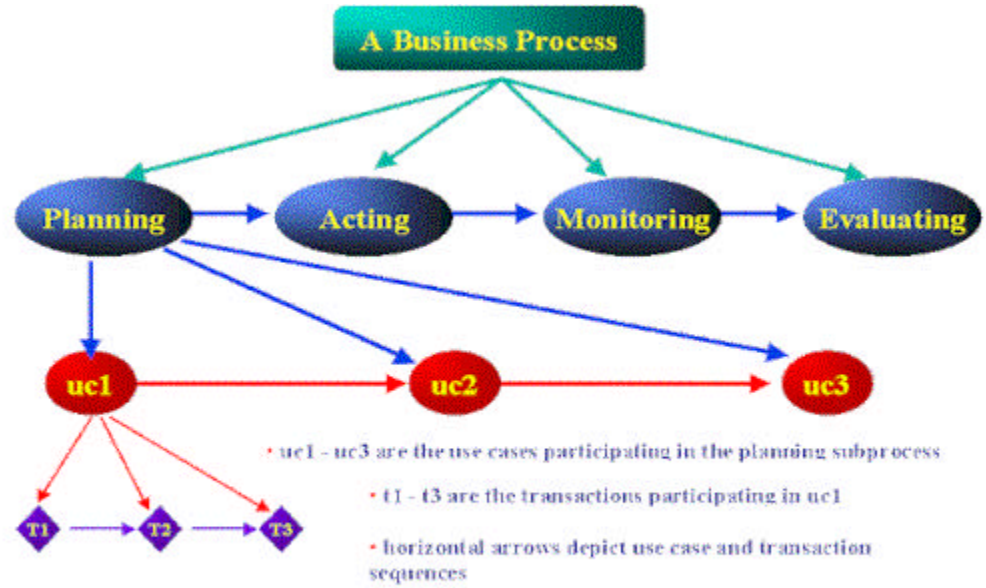

Figure Four -- The Hierarchy of Business Processes, Subprocesses, Use Cases And Transaction Sequences

An Information System Use Case, in turn, is defined by Jacobson [6] 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." A behaviorally related set of information system use cases, in turn, constitutes an information systems application supporting a business process through its use cases. This application may or may not extend over the four subprocesses, depending on its scope. Figure Five shows the relationships of business and information system use cases in an application.

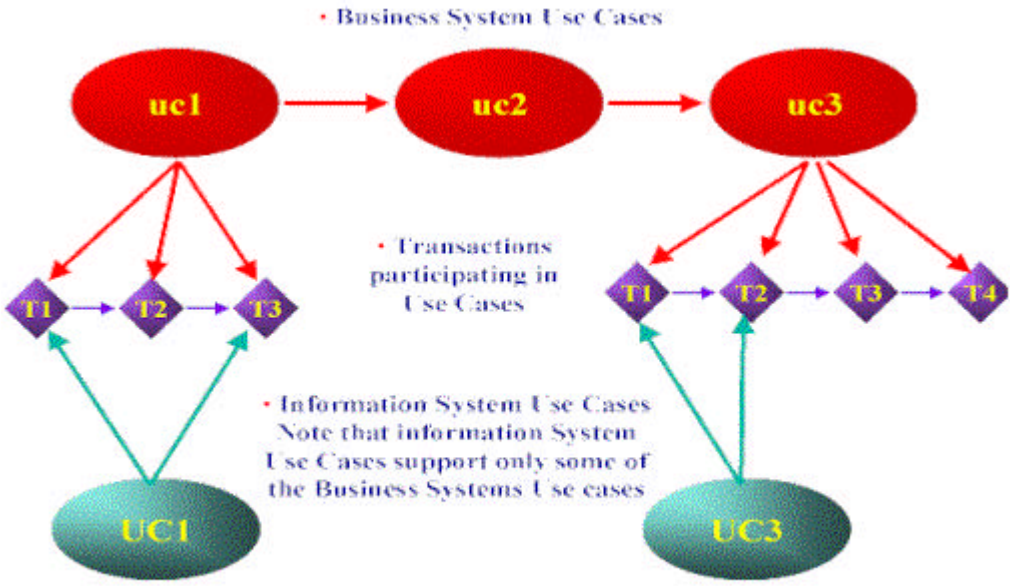

Figure Five -- Relationships of Business System Use Cases to Information System Use cases

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 One illustrated the connection between a use case and a hierarchy of 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.

The activities previously presented in Figure Three, are examples of Insurance Industry Business System Use Cases. A list of twenty-three closely related information system use cases for the Insurance Selling Process, classified by sub-process is presented just below. [7]

- **Planning**
-
- **Use Case One:** Develop, Revise Or Reengineer Hierarchy Of Sales Goals, Strategic Sales Objectives, Tactical Sales Goals, And Tactical Sales Objectives
- **Use Case Two:** Plan Specific Actions To Accomplish Tactical Sales Objectives
-
- **Acting**
-
- **Use Case Three:** Enter New Qualified Prospect Or Customer Information Into A Selling Database Interactively
- **Use Case Four**: Batch Enter Qualified Prospect Records And New Customer Information Into Agency Database
- **Use Case Five:** Review And Edit Qualified Prospect Records And New Customer Information At Agency
- **Use Case Six:** Batch Enter New Qualified Prospect And Customer Information Data Into Home Office Database
- **Use Case Seven:** Apply Scoring Models To Predict Likely Response To Sales Activity
- **Use Case Eight:** Home Office Notifies Agency And Agent Of Qualified Prospect And Customer Ratings Relative To Current Sales Opportunities

- **Use Case Nine:** Agent Selects Sales Prospect Lists
- **Use Case Ten:** Batch Update Records From External Lists
- **Use Case Eleven**: Batch Clean And Consolidate Server Databases
- **Use Case Twelve:** "Fix Bugs"
- **Use Case Thirteen**: Update And Enhance The Sales Productivity System
- **Use Case Fourteen**: Partition Applications And Balance Processing Load
- **Use Case Fifteen:** Tune Database Server Performance
-
- **Monitoring**
-
- **Use Case Sixteen:** Provide Secure Access
- **Use Case Seventeen**: Backup
- **Use Case Eighteen**: Produce Maintenance, Basic Information, And Complex Segmentation Reports In Response To Quick Count And Ad Hoc Queries Of Selling Activity Data
- **Use Case Nineteen:** Produce Reports In Response To Queries Whose Purpose Is Impact Analysis And/Or Forecasting Of The Properties And Outcomes Of Sales Activity
-
- **Evaluating**
-
- **Use Case Twenty:** Assess Actual Outcomes Of Selling Against Tactical Sales Objectives
- **Use Case Twenty-One:** Assess Forecast Outcomes Against Forecast Tactical Sales Objectives
- **Use Case Twenty-Two:** Assess Benefits And Costs Of Past And Current Selling Activities And Outcomes
- **Use Case Twenty-Three:** Assess Forecast Benefits And Costs Of Future Selling Activities And Outcomes

Use Cases, once again, are interrelated task sequences. An example of a use case with its task sequence and an accompanying comment including a description of the use case is given below.

- **Use Case Two:** Plan Specific Actions To Accomplish Tactical Sales Objectives

  - **Structure**

    (a) Retrieve tactical objectives specified in Use Case One

    (b) Enter action options being considered to help attain tactical objectives

    (c) Select the "backward-chain option to cause the system to choose the actions or action sequences likely to produce the tactical objectives; or alternatively specify rules relating actions to tactical objectives, and choose the actions based on the rules

    (d) Save any rules specified in (c)

    (e) specify a work-flow assembling individual actions into a plan constituting a sales effort, including planned activities to record the results of the sales effort.

    (f) Communicate the plan to the rest of the sales team

  - **Comment**

    In this use case, the Sales Planner selects the plan of action designed to accomplish the tactical objectives expressed in the analytic hierarchy. Specific Selling activities include: selecting leads previously identified through marketing, scheduling and making contacts with them, further qualifying these prospects; scheduling and making sales appointments, formulating scripts for sales appointments, selecting selling techniques to be used for selling the customer and closing the sale.

    Outcomes of Selling activities include: numbers of qualified prospect leads, confirmed appointments, attendees at Selling seminars, successful sales, unsuccessful sales attempts, networked leads to further customers, up-sells, and cross-sells, as these outcomes are segmented by agency, by agent, by time, by geography, and/or by other segmenting factors that can add to one's

understanding of sales performance. A central task in this use case is relating action alternatives to likely outcomes through some visual interface. And since tactical objectives are expressed in terms of desired or required outcomes, the user needs an interface to select the outcomes first, and then needs to have the system "backward-chain" to the actions or action sequences that are likely to produce these outcomes.

This use case then, requires models and an inference engine that will relate actions to outcomes. In turn, to arrive at these models, and to keep them current, *a considerable amount of data mining activity* will need to support the application.

In additon to the capability to relate actions to outcomes, the use case must let the Sales Planner structure Selling activities into plans incorporating various Selling activities sequentially. In turn, this implies a work flow design capability in the application to assemble the individual actions leading to certain outcomes into a plan constituting a Sales effort, including planned activities to record results of the sales effort. Of, course there must be print, e-mail, fax, groupware, and perhaps other communication capabilities to allow the results of Sales efforts to be shared and published with the rest of the sales team.

Use case modeling is essential in constructing a DKMS, because it leads to identification of actors performing the use case and objects supporting it. Once the objects are identified, they, their relations, interactions, and dynamics can be modeled. Code can then be generated or written, and then tested against the use cases, until an acceptable application results. Use cases are also essential in arriving at increments of the DKMS, in the incremental approach.

For example, in the insurance selling process illustration, an increment could be scoped according to process (selling), subprocess (planning, acting, and monitoring), and use cases (all the use cases in the three subprocesses, and use cases Twenty and Twenty-One in evaluating). Thus, this increment would not contain cost/benefit analysis, and would only deal with the insurance selling process.

Requirements Analysis

Requirements analysis involves partitioning use cases into the various objects supporting them, and modeling these objects and their structural relations.

Modeling Objects

The Object Model is developed through four sub-models: [8] The Enterprise Business Object Sub-Model; The Local Business Object Sub-Model; The Interface Object Sub-Model; and the Storage Object Sub-Model. Here are definitions of key concepts, followed by descriptions object modeling activities for the sub-models.

- Enterprise Business Object Modeling

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 transcendent 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. [9] 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 Six provides an illustration of an entity object.

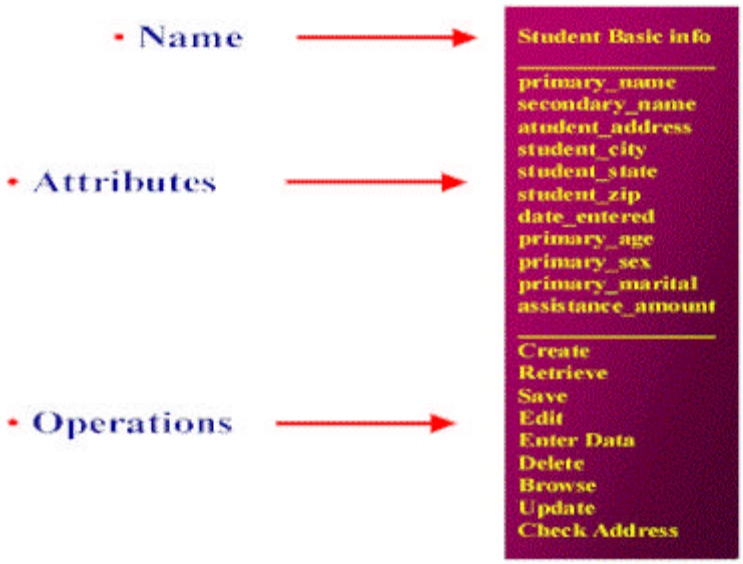

Figure Six -- An Entity Object with Name, Attributes, and Operations

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. Or it may be a procedural legacy application that has been "wrapped," in a package providing an object interface and encapsulation to the legacy application. 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 implementations 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.

Object modeling begins with identifying and specifying Enterprise Level Business Objects supporting the business processes, sub-processes 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, and enterprise-level business policies expressed as business rules.

Once the objects are specified the next step in object modeling is building 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 Seven illustrates a cause/effect relation among three object classes. The relation is defined by the association class called academic success, 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.

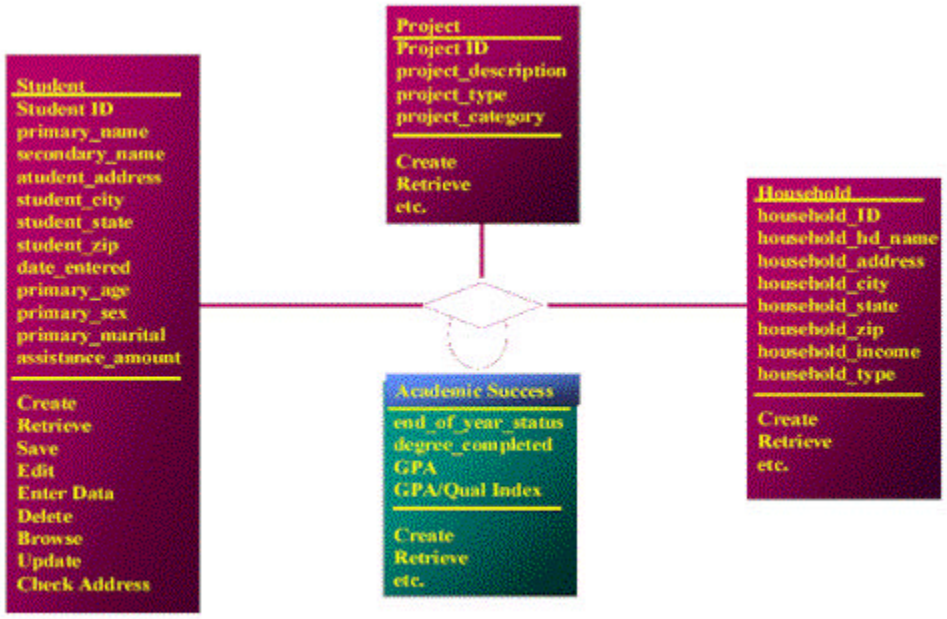

Figure Seven -- A Cause and Effect Association in an Object Model

- Local Business Object Modeling

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.

Developing the LBOM begins with identifying and specifying LBOs supporting the business processes, sub-processes 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; and also the business area policies expressed as business rules.

Next, build an LBOM showing relations among the LBOs. 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.

- Interface Object Modeling

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 is 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. 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.

- Storage Object Modeling

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.

If the database(s) being used for the data warehouse is (are) relational, or extended relational, model the SOM as a Dimensional Data Model (DDM). 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 Eight depicts mappings from LBOMs to SOMs for various database servers.

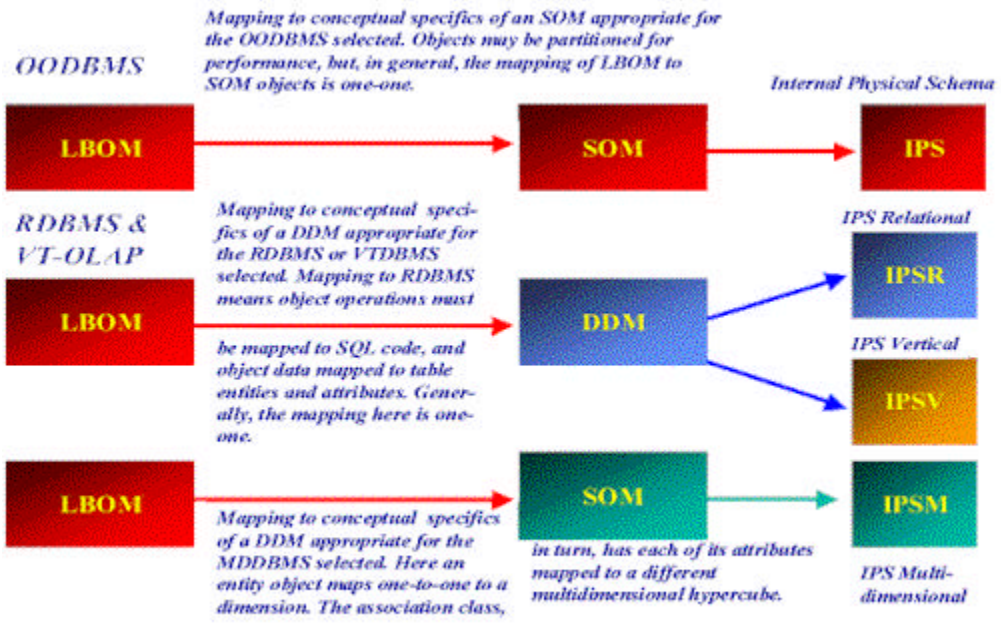

Figure Eight -- LBOM to SOM Mappings

An Object-Oriented Software Engineering (OOSE) [10] approach with its associated object modeling provides a particularly systematic viewpoint for both conventional E-R and dimensional data modeling. (See the discussion on the use of object modeling in dimensional data modeling in my paper on "Object-Oriented Data Warehousing.") 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 Seven.

System Design

System Design is the process of adapting the requirements analysis to the specific physical environment of the DKMS application. This begins with Object Interaction Modeling, continues with dynamic modeling of the life cycle of important objects, includes detailed design of multi-tier client/server architecture, more detailed specification of previous models developed, including physical specification of database models, and selecting tools for system construction.

Object Interaction Modeling

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 Sequence 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 Nine 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 supporting the use case.

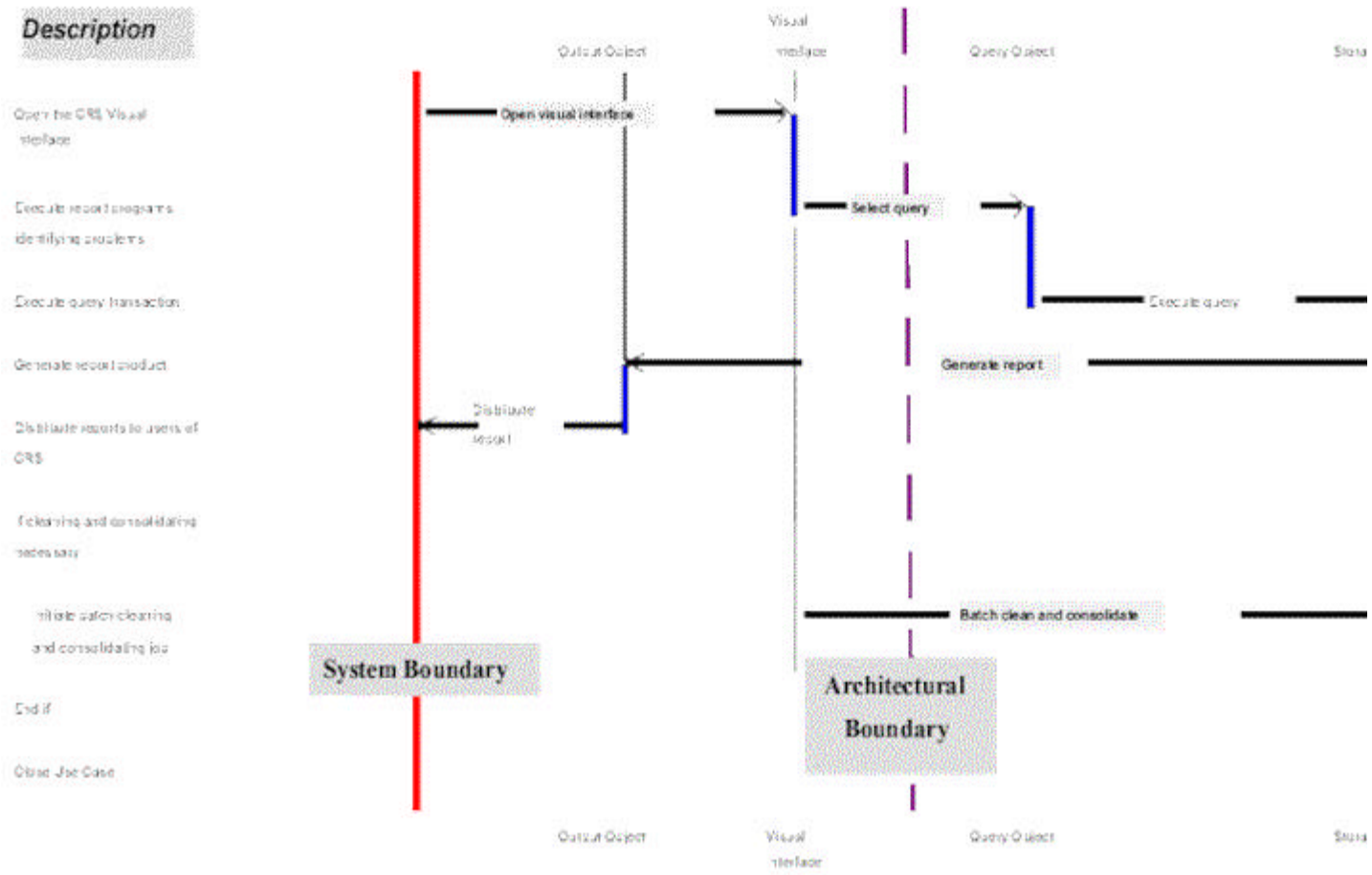

Figure Nine -- An OIM Sequence Diagram for A Batch Cleaning and Consolidation Use Case

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,"[11] for more detail.)

Dynamic Modeling

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." [12] One STD is developed for each object type whose dynamic activity we think it is important to represent. Figure Ten presents an example of an STD.

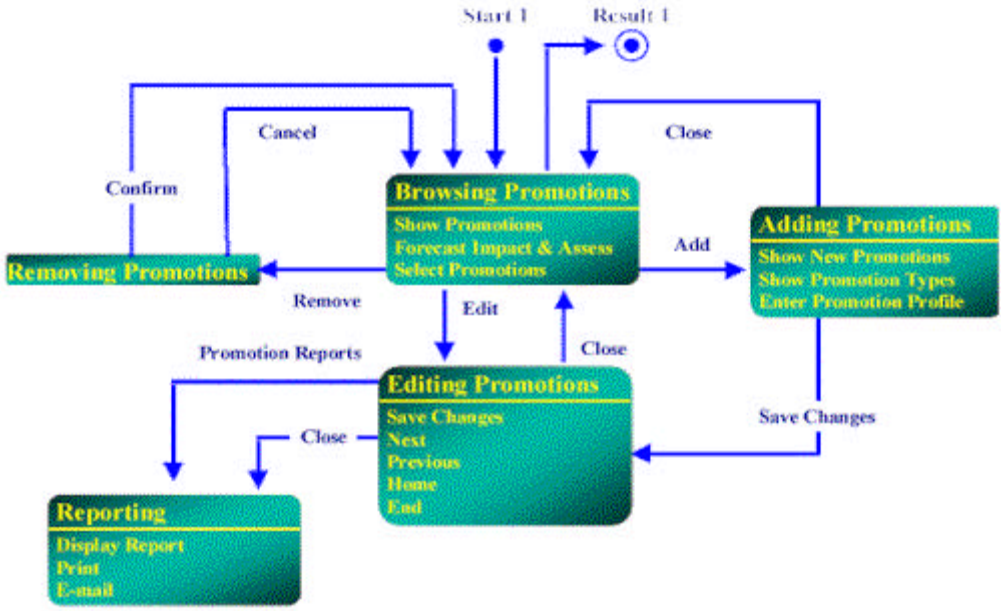

Figure Ten -- A Promotions Controller State Diagram

It shows the dynamics of a Promotions Controller Business Object, supporting data entry, editing, and reporting use cases. [13] The STD shows the various states of the object, and the events that bring transitions from one state to another.

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.

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.

Detailed Design Of Object Types, Relationships And Interactions

Specify EBOM and/or LBOMs, OIMs, and Dynamic Models in greater detail. Utilize other design techniques such as Use Case Maps [14] (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 more detailed Dynamic Models. Review Use Case Maps and Sequence Diagrams as a foundation for extending dynamic models through more detailed STDs.

SOM database dictionaries and schema

Create physical SOM database dictionaries and schema, specifying the physical dimensional data model, or object model (if an OODBMS is used).

Detailed IOM Prototypes

Create more detailed IOM Prototypes for user validation of the various IOMs defined in enterprise level and/or business area analysis. As detailed IOM prototypes are implemented, the prototype begins to approach the GUI implementation.

Detailed Design Of Client/Server Architecture

Perform detailed design of muti-tier client/server distributed processing network architecture.

Selecting Tools

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.

Constructing the System

System Construction covers a variety of code generation, coding, systems integration, data manipulation and processing, database tuning, testing, installing and validating tasks. These are listed below with brief comments.

Extracting, Transforming, and Transporting Data

Extract, Transform and Transport (ETT) data specified in the data model(s) to the system's database server(s), and automate the ETT process for maintenance. The DKMS is not merely a data warehouse or data mart application. But the ETT tasks characteristic of these applications are necessary for the DKMS, as well.

Implementing Database Servers

Implement ROLAP, MDOLAP, VT-OLAP [15], OODBMS database server(s). Tune the database server(s) for rapid ad hoc query response.

Implementing Middleware

Implement necessary middleware connectivity software such as Transaction Servers, TP Monitors, Object Request Brokers (ORBs), ODBC and JDBC, gateways, etc.

Partitioning and Load Balancing

Partition the application and balance the processing load across the distributed processing network, to improve performance

Knowledge Discovery in Databases (KDD)

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 Modeling Results and Implement Dialogues and Reports

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, DKMSs 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 DKMS 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.

Implementing on the Web

Implement the IOM on the World-Wide Web platform, if necessary.

Testing

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 validation of all the object and dynamic models underlying the application.

Planning Cutover

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

Complete installation of DKMS at user site.

DKMS Integration Tools

No software vendor yet offers a DKMS application generator, analogous to a DBMS, and it is likely that this will remain the situation for the foreseeable future. To create a DKMS one must employ a variety of software tools and integrate a variety of components, and in a real world situation many of these components are likely to be legacy systems whose productivity needs to be conserved or enhanced. The objective in seeking tools to help create DKMSs, is to find tools that allow us to systematically model and treat the integration problem that is at the core of the DKMS. The tools we seek must be development tools, but in addition they must provide the "glue" that will hold the variety of DKMS components together. In short, the tools involved are DKMS integration tools, rather than application generators. To produce DKMSs they require the assistance of a variety of tools and components. It will be useful to have a list of these in mind when reviewing DKMS integration tools and their capabilities.

- Universal Repositories
  - Application Repositories
  - Design Repositories
  - Metadata Repositories
- Data Warehouse (ROLAP Servers)
- Data Marts (ROLAP, MD-OLAP, or VT-OLAP Servers
- Data Mining Servers
- Web Servers
- Mail and Fax Servers
- Other Application Servers
- Wide Area Networking Servers
- Transaction Servers
- OLAP Clients
- Distributed Processing Middleware
- Transaction Servers
- Reporting Tools
- Data Visualization Tools
- Intelligent Agents
- Object Request Brokers
- System Monitoring and Management Tools
- TP Monitors
- Data Extraction, Transformation and Transportation Tools

Considering the characteristics of the DKMS described earlier, a set of criteria for DKMS integration tools can be specified. To support the approach to DKMS construction specified above, DKMS integration tools would need to support:

- Object Modeling
  - Enterprise Business Object Modeling
  - Local Business Object Modeling
  - Interface Object Modeling
  - Storage Object Modeling
- Object Interaction Modeling
- Dynamic Modeling
- Physical Class and Code Generation, and GUI prototyping and development
- Integration of External Software into Object Sub-Models
- Distribution and Deployment of Components Across Networks
- Web implementation capability
- Programming Capability with Generation of O-O Language Source (C++, Smalltalk, Java)

Tools available currently don't fulfill all of the criteria. The tool that comes the closest is perhaps Template Software's SNAP 8.0 development environment, supplemented by its Workflow, and Web Templates. Tempate's suite of tools is weak in Object Interaction and Dynamic Modeling (though not in development in these areas), but has excellent capability in other areas. To strengthen its modeling capabilities, Template's suite could be integrated with Select Enterprise or Rational Rose, or another good O-O CASE tools which provides OIM, and Dynamic Modeling capability.

Forte is very good at development and deployment tasks, but it is not an object, object interaction, or dynamic modeling tool. To work for DKMSs it needs to be supplemented by a strong O-O CASE tool it can be integrated with, such as Select Enterprise.

The combination of Riverton's HOW 1.0 and Powersoft's PowerDesigner and PowerBuilder may provide another good suite for DKMSs, as HOW provides an Object Modeling and round-trip engineering front-end for Powersoft's tools. But at this time HOW is not yet integrated with the latest versions (6.0) of Powersoft's products, and it is also a very new product itself. Finally, HOW doesn't support Object Interaction or Dynamic Modeling.

Another alternative worth examining for DKMS integration functionality is NatSystems NatStar. NatStar has strengths in both and object modeling and development. In addition it can integrate with Select Enterprise, Rational Rose and Paradigms Plus, as well as with development tools such as Visual Basic and PowerBuilder. NatStar, however, produces C source, rather than C++, Smalltalk, or Java.

Conclusion

The future of software application development will involve familiar activities and components. But these will be encompassed in a broader "adaptive component architecture" and in a more encompassing application. This "unifying template," [16] the DKMS, will not be data-driven. It will be process-driven, with data playing a vital role.

While the distinction between the traditional Data Warehouse and the DKMS should be clear from the previous description, it is important to distinguish the DKMS from the Object-Oriented Data Warehouse (OODW). Unlike the OODW the DKMS departs from the defining characteristic of the data warehouse as a read-only system. In data warehousing it is an article of faith that the application is read-only. "Twinkling databases," [17] produced by On-Line Transaction Processing (OLTP) are not allowed in data warehouse architecture.

Part of the reason for this is the desire for superior performance. If the data warehouse must support transactions as well as querying, it will not deliver the necessary query performance. An even more important reason for the read-only data warehouse is the desire for stability of results. If updates were to occur between ad hoc queries the resulting "temporal inconsistency," would destroy the integrity of the analysis embodied in a chain of queries.

In the DKMS, the relational database server is not the primary focus of attention, as it is in the traditional data warehouse. The presence of multiple database servers and application servers, allows DSS, Batch and OLTP processing in the same DKMS. The data warehouse and data marts must still avoid the twinkling database in order to produce temporal consistency during query chaining. Nevertheless, the shift from server-centricity to the network as computer in the DKMS makes possible applications that can provide for all three classical categories of processing simultaneously.

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, read-only systems will not be providing Decision Support. Instead, multifunctional DKMSs will be. Data entry and transaction processing tiers of processing will coexist along with data analysis and DSS tiers within the same DKMS. Data gathering and data entry, after all, are also aspects of distributed knowledge 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.

References

[1] "What is a Data Warehouse?" Prism Tech Topic, Vol. 1, No. 1, 1995.

[2] This is a broader, and I think more justifiable, definition of knowledge base than popularized by the field of AI. There knowledge base refers to the part of the domain knowledge that an expert system must contain, and includes rules, procedures, criteria, and formalized expertise. See Fatemeh Zahedi, Intelligent Systems for Business (Belmont, CA: Wadsworth, 1993).

[3] 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.

[4] 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),

[5] Ivar Jacobson, Maria Ericsson and Agneta Jacobson., The Object Advantage: Business Process Reengineering with Object Technology (Reading, MA: Addison-Wesley, 1995), P. 343

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

[7] A detailed treatment is in my "Use case Analysis of the Insurance Selling Process," (Wilmington, DE: Executive Information Systems, Inc., June 25, 1997).

[8] The development from here on relies heavily on the work of Stuart Frost, The Select Perspective: Developing Enterprise Systems Using Object Technology (Santa Ana, CA: Select Software Tools, Inc. 1995), James Martin and James O'Dell, Object-Oriented Methods: Pragmatic Considerations (Englewood Cliffs, N.J.: Prenctice-Hall, 1996), Jacobsen, et.al, Ivar Jacobson, Magnus Christerson, Patrik Jonsson, and Gunnar Overgaard (Reading, MA: Addison-Wesley, 1992), James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen, Object-Oriented Modeling and Design (Englewood Cliffs, N.J.: Prenctice-Hall, 1991), and Grady Booch, James Rumbaugh, and Ivar Jacobson in Rational Software's Unified Modeling Language version 1.0, available at http://www.rational.com., as is an alpha version of UML 1.1.

[9] Jacobson et.al. , 1992, Pp. 169-187.

[10] Object-Oriented Software Engineering (OOSE). The term is used here in a general sense, not in the specialized meaning of Jacobson et.al., 1992.

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

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

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

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

[15] Refers to the column-oriented query technology implemented in Sybase's IQ Product. See, Joseph M. Firestone, "Evaluating OLAP Alternatives," (Wilmington, DE: Exeutive Information Systems White Paper No. Four, March 28, 1997).

[16] The term was suggested by Ron Goodes.

[17] This is Ralph Kimball's expression. See The Data Warehouse Toolkit (New York, NY: John Wiley & Sons, 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 System (DKMS) as an organizing framework for the next business "killer app." You can e-mail Joe at eisai@home.com.

[Up] [KMBenefitEstimation.PDF] [MethodologyKIv1n2.pdf] [EKPwtawtdKI11.pdf] [KMFAMrev1.PDF]
[EKPebussol1.PDF] [The EKP Revisited] [Information on "Approaching Enterprise Information Portals"]
[Benefits of Enterprise Information Portals and Corporate Goals] [Defining the Enterprise Information Portal]
[Enterprise Integration, Data Federation And The DKMS: A Commentary]
[Enterprise Information Portals and Enterprise Knowledge Portals] [The Metaprise, The AKMS, and The EKP]
[The KBMS and Knowledge Warehouse] [The AKM Standard]
[Business Process Engines in Distributed Knowledge Management Systems]
[Software Agents in Distributed Knowledge Management Systems]
[Prophecy: META Group and the Future of Knowledge Management] [Accelerating Innovation and KM Impact]
[Enterprise Knowledge Management Modeling and the DKMS] [Knowledge Management Metrics Development]
[Basic Concepts of Knowledge Management]
[Distributed Knowledge Management Systems (DKMS): The Next Wave in DSS]