Issues in Designing an Information Model for Application Development

Gary H. Sockut, Helen P. Arzu, Robert W. Matthews, and David E. Shough[2]
IBM Santa Teresa Laboratory
P. O. Box 49023
San Jose, CA 95161-9023

ABSTRACT

IBM's[3] object-oriented information model lets a customer share data among various tools for application development. This paper discusses several issues in designing the information model, namely (1) techniques for diagrams (an essential part of communication between an information model designer and other designers or tool writers), (2) organization of the design of the information model (an essential step when many designers design anything large), and (3) technical content. These discussions of the experience of designing the information model should be valuable for further design of the information model and for other design efforts, e.g., involving other models or other integration of tools.

1. INTRODUCTION

This paper describes solutions to issues in designing IBM's object-oriented information model, which supports application development activities. The information model contains a set of object and relationship class definitions to describe and relate information about an enterprise and its data processing applications. Through the use of these definitions, different application development tools (perhaps produced by different vendors) can share this information; the output of one tool can be the input to another.

Specifically, we discuss issues in these aspects of the design:

1. *Techniques for diagrams* facilitate communication among designers, reviewers, and users of the model.
2. The *organization of the design process* enabled us to design a large model with a large number of designers and reviewers in a limited time.
3. The *technical content* is the result of the modeling effort; it enables sharing of information among application development tools.

We hope that these discussions of our experience can be valuable for further design of the information model and for other design efforts, e.g., involving other models or other integration of tools.

2. TECHNIQUES FOR DIAGRAMS

With a large team of designers (both within IBM and outside IBM), and with a large set of writers of application development tools, communication is essential. With a large set of constructs in the information model, diagrams are an essential part of that communication.

A combination of diagramming techniques is required to fully represent the information model. The techniques come from three diagram types:

- Entity-Relationship diagrams (ERD) show entity names, relationship verbs, and cardinality.
- Instance Diagrams show instances of entities (including attribute names and values) and of relationships.
- Inheritance Diagrams show the hierarchical layout of the model (superclasses and subclasses), including inherited relationships.

The first two diagram types are standard and commonly used in data modeling. There are organizations[4] and conferences that stimulate discussion of new developments and issues associated with the ER approach [March 1988, p. v].

In these gatherings, it is recognized that "although we (are) all practicing ERD modelers, each corporation represented and evolved their own unique methodology for information modeling. The terminology used and even the diagramming conventions adopted were often variations of the strict ERD approach, that had been adapted to meet the needs of the corporate environment being serviced" [Moriarty 1988, p. 25].

The need for adapting ER diagrams was exactly the situation in which our development team found itself. We needed to adapt ER diagrams in order to support an object-oriented model. We needed a diagramming technique that clearly reflected the hierarchical structure of the model.

We searched for a method of diagramming that was easy to read, manageable in size, and consistent with the best features of object-oriented diagrams already out in the field.

2.1 Techniques that were Considered

Currently, there is no diagramming standard for object-oriented development [Martin 1993, p. 54], so we looked at diagrams used in object-oriented design and data modeling.

2.1.1 Martin and Rumbaugh

James Martin, in his text, *Principles of Object-Oriented Design and Analysis*, acknowledges that there is a need for a diagramming standard that is easy for conventional systems people to learn. He goes on to make suggestions for standards. Martin's suggestions were based on *Recommended Diagramming Standards for Analysts and Programmers*, which he calls the "bible" for many application development vendors [Martin 1993, p. 55].

We looked at the diagramming Martin suggested and the diagramming technique used by James Rumbaugh et al. [Rumbaugh 1991]. We referred to Rumbaugh heavily during our architectural design.

[2] Correspondence about this paper should go to R. W. Matthews.

[3] AIX, DATABASE 2, DB2, IBM, IMS, OS/2, SQL/DS, and SystemView are trademarks of International Business Machines Corp.

[4] The ER Institute and the San Francisco Bay Area Entity-Relationship Diagramming (ERD) User's Group are such organizations.

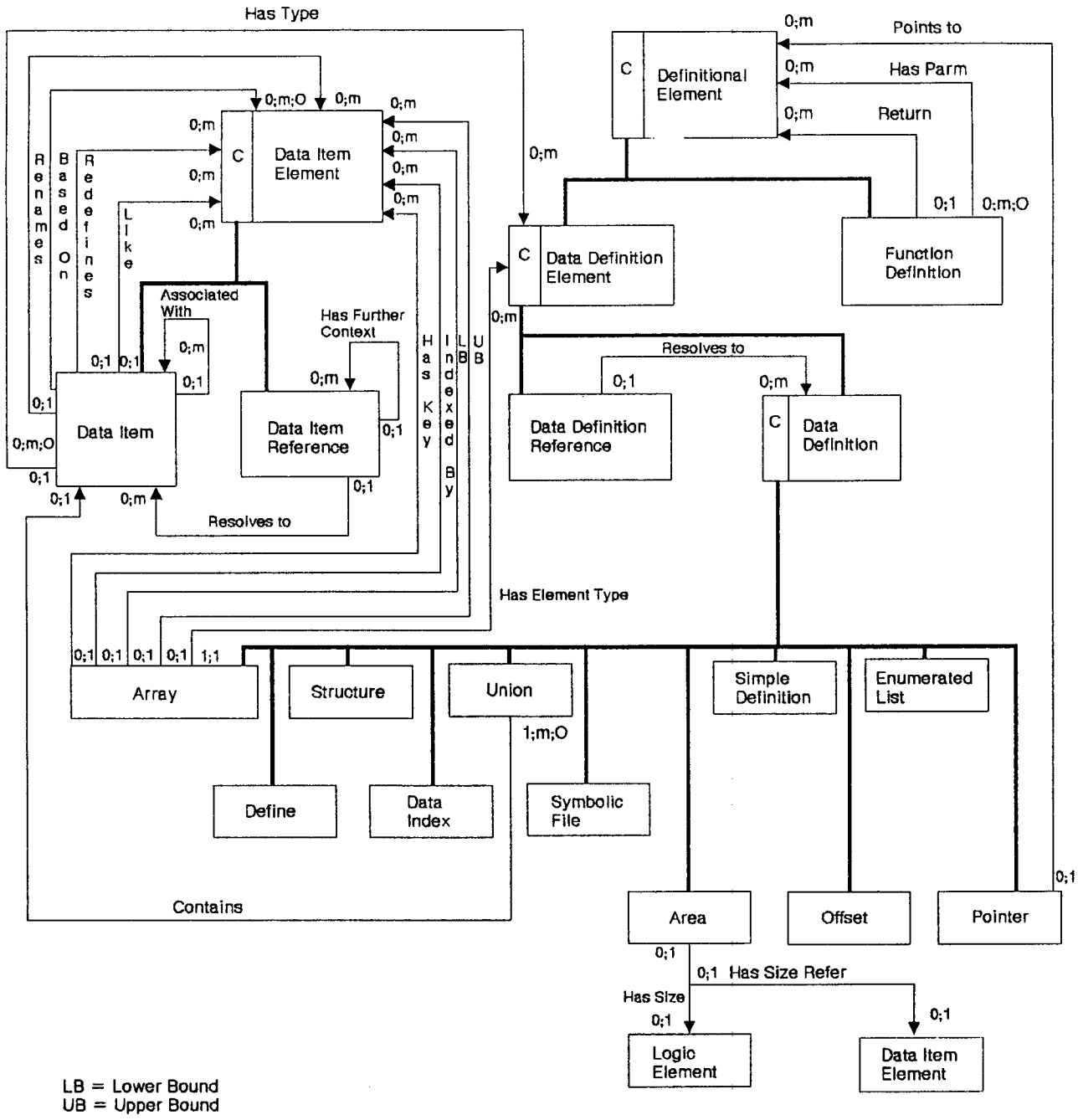

Figure 1. Horizontal Structure

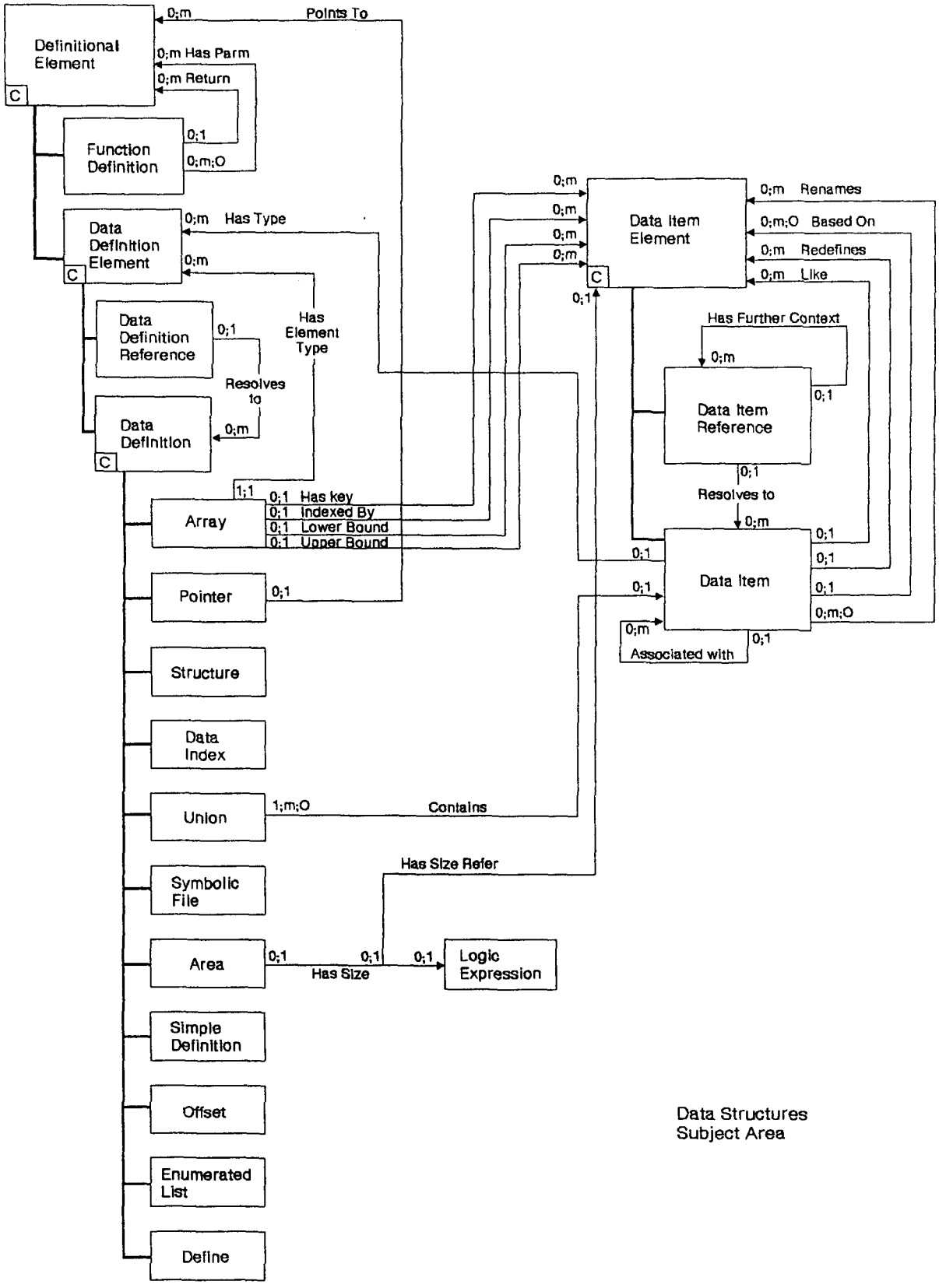

Figure 2. Vertical Structure

Upon further assessment, we did not fully adopt either Rumbaugh's or Martin's diagramming because:

- Rumbaugh's diagramming approach did not provide the detail required to document the information model.
- Martin's diagramming approach introduced several symbols to replace characters (strings) used in ER diagramming. We felt the characters currently used when diagramming our model were more straightforward and required less interpretation than symbols.

2.1.2 Conceptual Graphs

Another diagramming technique we considered was introduced by John Sowa [Sowa 1984]. Conceptual graphs are based on artificial intelligence semantics. The graphs form a knowledge representation language based on linguistics, psychology, and philosophy. In the graphs, concept nodes represent entities, attributes, states, and events; relation nodes show how the concepts are interconnected.

Conceptual graphs attracted our attention because we could: (1) diagram the precise semantic nature of information, which can be used to generate logic adhering to that information, (2) use logic to generate predicate calculus, (3) use the linguistic properties to aid in the development of stylized English directly from the model, (4) show all the semantic properties of the model, and (5) show partitioning and concurrency.

But the benefits of conceptual graphs were countered by characteristics of our model:

- We need to show model information content, not logic of how it was designed.
- The generated logic and predicate calculus were not requirements for our customers.
- It would be quite some time before the ability to generate stylized English would be considered an advantage in the market place.

Also, we found that conceptual graphs were complex, were difficult to read, and used a considerable amount of space on paper. Of course, this space reduced the amount of information per page. This reduced the portion of the model that could be visualized at one time, thus making it harder to understand the model.

2.1.3 SystemView's Technique

The information model group at IBM in the Research Triangle Park (RTP) worked with the group at the IBM Santa Teresa Laboratory on the information model project. The SystemView group at RTP developed another object-based data model. *SystemView* (a tool for managing system resources) used a vertical format to show relationships, but the diagrams did not show an inheritance hierarchy. The SystemView model had considerably fewer levels of hierarchy and fewer inheritance and constraint factors than does the information model. The information model group at RTP worked with SystemView on model convergence and consistency. The information model group at RTP expanded the SystemView diagramming format to reflect inheritance hierarchies.

2.1.4 Adjustments in Diagram Requirements

After reviewing these various object-oriented diagramming techniques, we determined that one technique could not cover all aspects of the model. We decided to combine (1) object-oriented diagramming techniques that show hierarchical structures with (2) our established ER and instance diagramming techniques for the finely grained level of the model.

The hierarchical structure needed to be similar to ER modeling diagramming because:

- We still needed to use ER diagrams for the most granular level and wanted a smooth visual transition from coarse to granular levels.
- Our target audience was familiar and comfortable with the current ER conventions.

2.2 Narrowing Down the Selections

We combined all the elements we liked about the diagrams we reviewed. We narrowed down our selections to what we called *horizontal diagramming* (based on ER diagramming and Martin and Rumbaugh diagramming) and *vertical diagramming* (based on the diagrams used by developers of the SystemView Data Model). Once we narrowed down our selection to two types, we identified common elements – symbols and characters that could be used regardless of whether the diagram was horizontal or vertical. We then did usability testing to determine which approach to use.

2.3 Testing the Selections

The approach we used for usability testing are shown in figures 1 and 2.

2.3.1 Usability Study: Summary of Results

The results of the test showed that 92% of the users (11 of 12 test participants) preferred Figure 1, the vertical diagram, over Figure 2, the horizontal one. Users consistently noted that the vertical diagram was easier to read and better clarified the flow of information.

2.3.2 Summary of Recommendations

Although most test participants preferred the vertical style, many of them recommended additional changes to improve its usability. Recommendations were to use standard object-oriented terminology and to provide more visual cues (e.g., boldface superclasses) to clarify the information in the diagram.

Specific comments included:

- Put the Parent classes in bold boxes with boldface text to more easily find superclasses.
- Offset or change the font of the relationships so you can visually separate them from the other text. Most everything looks the same – seems to blend together now.

- For inherited relationships, add a hollow triangle. For the other relationships, move the arrowhead to the middle of the line.
- Crossing lines are okay but extra "jogs" in the lines are confusing.
- Put inverse relationship names in parentheses at the other end. Put forward-direction relationship names at the source end of the relationship if possible.
- Use bold boxes for superclasses, or put the names of the superclasses in bold or uppercase.
- Use standard object-oriented notation like Booch's O-O notation to clearly show contains, uses, owns, kinds of relationships.
- Can the bold lines be eliminated? Superclasses are always above subclasses, so that is enough indication. Arrows denote relationships. Alternatively, use bold for the object class boxes to make them stand out more and normal thickness for connecting lines.

2.4 Further Definition of the Style Guidelines

The team responsible for information model documentation created more detailed guidelines for documenting and diagramming the model, concentrated on the more granular level of ER diagrams.

2.4.1 Object Class

Use the following rules when showing object classes: (1) Avoid spaces in object class names in text (following C++ conventions). (2) Use initial uppercase on each component word of a class name. (3) Avoid underscores in diagrams.

So a diagram looks like:

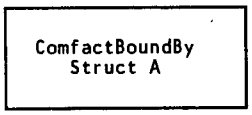

Figure 3. Example of object name in diagram

2.4.2 Relationship Class

Use the following rules when showing relationship classes: (1) Use initial uppercase on each component word of class name. (2) Omit source and target in relationship names. (3) In diagrams, show the verb for one direction of a relationship.

Here is an example:

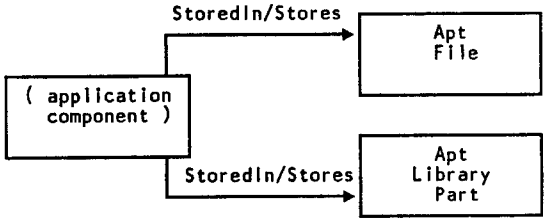

Figure 4. Example of relationship name in diagram

2.4.3 Instance Diagrams

Use the following rule when showing instance diagrams: Show primary (or necessarily specified) values within the box; show other values below it.

Use the following rules when showing attributes: (1) Avoid spaces between component words in text. (2) Use no emphasis in text. (3) Follow by "=" and attribute value in diagrams. See Figure 5.

When showing attribute values, use all lowercase (following "=" in diagram) unless some uppercase character is an intrinsic part of the value name.

Here is an example instance diagram:

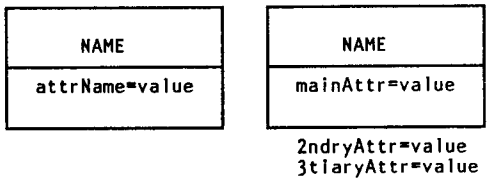

Figure 5. An instance diagram

3. ORGANIZATION OF THE DESIGN PROCESS

Now we turn from diagrams to the design process. When many people design a large specification, a well-organized process is essential. This section covers two aspects of organization of the design process, namely (1) techniques for dealing with a diverse group of modelers and (2) cost-effective, multisite design reviews.

3.1 Techniques for Dealing with a Diverse Group of Modelers

A central organization within IBM coordinated the design and performed most of the design, but the design required the expertise of other modelers within IBM and also some independent application development vendors that have a cooperative relationship with IBM. We needed such a large group of modelers and reviewers, because the information model should satisfy the modeling needs of a wide variety of tools. Here we will discuss the coordination among diverse modelers.

3.1 Establishing Ground Rules

When the modeling team first assembles, several topics must be discussed and agreed upon to expedite the modeling process. This section discusses most of them.

3.1.1.1 Methodology

Any experienced modeler has a preferred methodology or way of approaching a problem. When several modelers are united to address a given problem, they must agree on a single approach. Below are three common ways of addressing a modeling challenge based on the relationship between the model, processes, and existing technologies. The modeling team must agree on the best course of action for the problem it is addressing:

- Process driven – model the data used in existing defined processes.
- Data driven – model the data; then identify processes to manipulate the data.
- Technology driven – model the data found in existing technology (Database 2, etc.).

3.1.1.2 Base Assumptions
The modeling team must also establish some base assumptions:

- Will the basic modeling paradigm be entity-relationship or object-oriented? That is, will the modelers be only concerned with the data content of the model or will they also define the behaviors of the things that are modeled?
- Will the model use the concepts of subclassing and inheritance? If so:
  - Will the modelers be limited to single inheritance, or will multiple inheritance be allowed?
  - Will supertypes (superclasses) be concrete, abstract, or both (i.e., can the superclass be instantiated or not)?
- What naming conventions will be followed?
- With what recognized standards will the model comply?
- What diagram conventions will the modelers use?
- What rules shall the team follow for establishing a common terminology?
- Are there any modeling restrictions that are imposed by an intended implementation environment?
- What guidelines will the team follow with respect to normalization and redundancy of constructs in the model?

All of these and more are needed to establish the "playing field" where the modeling team will operate.

3.1.1.3 Model Scope
It is assumed that the "charter" that established the modeling team includes some indication at a gross level of the nature of the problem to be addressed. The modeling team must ensure that all members understand the scope within which they are to operate. Often the team will refine the scope to be more explicit and precise in defining the boundaries of the intended model. This refinement will describe the areas to be modeled and can also define the division of labor relative to the project.

The division of labor identifies the rules under which pieces of the model can be developed by sub-teams and later integrated into the whole. Two basic approaches are:

- Divide the overall model into *submodels*, each having a defined owner and where every model construct belongs to only one submodel.
- Identify overlapping *subject areas* in the overall model and assign ownership of the model constructs individually. A subject area is a domain of application development (e.g., enterprise modeling or relational database design) that the information model is to cover. A construct may appear in more than one subject area.

Both approaches rely on close teamwork and communication to ensure that "overlap" constructs address all requirements.

3.1.1.4 Modeling Tool(s)
The modeling team must identify the tool(s) it will use to produce the defined deliverables. The tool(s) must support most or all of several tasks (model definition, diagramming, documentation, and implementation assistance).

3.1.1.5 Project Management Procedures
Early on, several project related procedures must be defined so that the team's progress is not impeded unnecessarily. These procedures are mostly administrative and deal with issues that can and will arise during the life of the project. These include:

- Approval process – how are model proposals evaluated and approved?
- Change Control process – how are changes approved for previously approved parts of the model?
- Issue Resolution Procedures – how are general issues resolved? What is the escalation path for resolving major and minor issues?

3.1.2 Handling Requirements

Once the modeling team has discussed most or all of the topics listed in the sections above, it is ready to begin modeling. Or is it? Does the team know what it needs to model? Are there any itemized requirements that can drive the project? How will the team determine when it is done?

Initially, every group represented on the modeling team has some set of base requirements that it expects the product of the modeling effort to meet. The resulting model will undoubtedly differ, perhaps substantially so, from the model any one group would have produced independently. This is not necessarily bad. Frequently, such teams devise new and better ways to model the information so that the requirements of all groups represented are addressed.

As the modeling progresses, new requirements may surface. Some of these requirements come from the modeling team or its members. Others can come from other interested parties such as customers, vendors, and IBM tools.

Requirements can also arise from efforts to conform to various recognized standards. If the model will be implemented as a running system, frequently the target implementation environment may introduce additional requirements. Typically, these implementation requirements are in the form of restrictions of how things can be modeled, named, etc., not what can be modeled.

Regardless of the source of the requirements, the modeling team must establish and continually review priorities for each requirement. Careful tracking and recording of responses to each of the requirements will aid in reducing rework, promoting understanding, and compiling useful material for inclusion in user documentation.

3.2 Cost-Effective, Multi-Site Design Reviews

When faced with a need to conduct reviews of complex designs across multiple sites, the information model team developed an innovative, effective process which greatly reduced travel costs and inconvenience. This process consisted of an initial

video-conference meeting followed up by dialogue on the design specification.

The dialogue used *REVUFILE*, which is an internal IBM tool for reviewing documents. REVUFILE currently runs on the VM operating system. It provides all the function needed for inline annotation of text documentation at a line-by-line granularity. It does not offer the same granularity for graphics. Participants in the creation and review of a document can all see the document online (in a way that resembles its hard-copy appearance), enter comments in their appropriate contexts (e.g., directly under a sentence to which the comment applies), and see their comments (and other participants' comments) in these contexts. We used REVUFILE as a mechanism to gather discussion on a given topic. This was superior to using a forum (simple electronic bulletin board), because a forum offers only a chronological tracing of a discussion, not the segmentation by discussion topic that we achieved through REVUFILE.

3.2.1 Design Content

The DR2 (Design Review 2) was to be performed on the first "complete" version of the information model. It was complete in the sense that all of the major pieces were present but were not detailed in all aspects. In particular, the bulk of the material consisted of "pictures" of objects and their relationship to other objects. Subsequent design refinement would add attributes and methods to the objects and semantics to the relationships.

Previous experience had shown that it was crucial to get agreement and buy-in from the tool architects to this earliest version of the model. Most changes late in the model development cycle could be traced back to information present in this DR2-level model. Thus it was critical to get a thorough technical review of this model.

The entire model was too large to be reviewed as a whole. Also, not all tool architects cared about all portions of the model. Therefore, the model was divided into components. Each component had a lead designer and a list of tool architects interested in that portion of the model.

3.2.2 The Challenge

We recognized that we needed the synergy of multiple reviewers discussing the same issues. Each component of the information model required in-depth technical design reviews by tool architects at multiple sites. These sites spanned 8 time zones and included non-IBM vendors. We were on a very aggressive schedule that did not allow calendar time for extensive review-by-mail.

The classic single-meeting design review would have been ideal from the standpoint of technical interchange. But this would have required flying approximately 20 reviewers to IBM's Santa Teresa Lab from Europe and the East Coast of the U.S. Some would have to make multiple trips based on the design schedule of the components in which they had interest. This was clearly an unacceptable cost.

The challenge then was in getting a synergistic, thorough review without incurring unacceptable travel costs. That is, how do we approximate a single meeting without holding such a meeting?

3.2.3 The Solution

We came up with a review process which relied on two key aspects: video-conferencing and REVUFILE. For each component, we would schedule a three-site video-conference to walk the tool architects through the proposed model and note their primary issues and concerns. This video-conference was taped and copies of the tape were sent out in overnight delivery to those locations without video-conference hook-ups (Europe and non-IBM vendors).

At the conclusion of the video-conference, a REVUFILE was created on a common conferencing disk. Since most of the review material was graphic in nature, a "table of contents" was put in the REVUFILE to organize the discussion. Tool architects added their comments to either the text block under discussion or to the contents entry for the picture of interest. Model designers would then answer the concerns directly in the REVUFILE. Other interested parties could see the interchange and add their comments as appropriate.

A deadline was established for inclusion of all comments, concerns, and issues. The model designers would then rework the model based on the issues accepted and changes indicated and redistribute it as the final DR2-level design. This was used as the basis for further design refinements.

3.2.4 Logistics

A considerable amount of detailed activity needs to take place for the process to run smoothly. Video-conferencing required significant lead-time for scheduling. We booked approximately the same time slot each week for six weeks. A schedule was published detailing which components would be reviewed at each meeting so that tool architects could plan to attend as appropriate.

The tapes were made at each location. This is easily done through most video-conferencing set-ups. Then we spent the afternoon copying the video-tapes for distribution to Europe and non-IBM vendors. Up to six copies had to be created for some sessions. This was an unavoidable, time-consuming process. Alternatives were considered such as using an outside service for copying. This idea was discarded due to turnaround time and security considerations.

The REVUFILEs were created on a common project conferencing disk. Reviewers did not need direct access to this disk; REVUFILE allows reviewers at remote locations to append comments. An output listing file was placed on the common disk and made available to those at remote sites as well for printing hard copies.

3.2.5 Summary of Process for Design Reviews

The information model design team achieved a cost-effective, quick-response means of reviewing complex designs across multiple sites. In some ways we received better responses

than in a classic single-meeting review, because reviewers had a chance to spend time thinking about an issue before raising it.

We did lose a little bit of synergy compared with a design review meeting. The discussions in REVUFILE tended to be briefer than a corresponding oral discussion. But this was made up for by the ability to bring in other sources and use off-line "think-time."

Handling the logistics of video-conferencing and tape distribution took more of the design team's time. This was necessary to get the level of participation desired. The travel savings were substantial. Clearly, we would not have the participation desired if we handled the design review in a more traditional manner. Even given optimal scheduling, the reviews would have taken at least twenty cross-country and transatlantic trips of one week's duration each. A conservative dollar estimate would be more than thirty thousand dollars.

4. TECHNICAL CONTENT

The last area we will discuss is the technical content. We will discuss technical issues that deal with the goals of accommodation of diversity, sharing of definitions, and support for impact analysis. The purpose of this section is to show the type of thinking and the style of design that we found appropriate for achieving these goals. Many of the issues and examples come from the relational database subject area. An introductory knowledge of database management will help in understanding this section. The examples are just a small subset of the entire information model. The goals and the style of design should apply to subject areas that may be added to the information model in the future.

4.1 Accommodation of Diversity

Different programming languages, different database languages, and even different versions of one language (e.g., SQL) have different features and use different techniques in representing data and its behavior. Since different environments for application development use many different languages, it is essential for the information model to accommodate the differences. The information model uses one set of constructs having enough flexibility to accommodate the differences. We believe that this flexibility gives the information model sufficient generality without making the model unnecessarily large. Of course, the well-known technique of an inheritance hierarchy (superclasses and subclasses) also helps to accommodate diversity.

4.1.1 Language-Independent Data Definition with Language-Specific Extensions

Some aspects of a data definition are language-independent (applicable to several languages). For example, many languages can each reflect the fact that each EMPLOYEE has a character NAME and a numeric SALARY. Some aspects are language-specific. For example, if a data item is the basis for an SQL column definition, a relational database comment can apply to the SQL column definition, but if a data item is instead (or additionally) the basis for a PL/1 variable definition, a relational comment *cannot* apply to the PL/1 variable definition. Also, different languages have different rules for naming data items.

To avoid cluttering language-independent constructs by including language-specific attributes, we relate language-independent constructs to language-specific extensions. For example, in Figure 6, the HasColumnType relationship class connects the language-independent DataElement object class to the language-specific ColumnDefinition object class, which contains attributes like relational comment and SQL name. If the name in the DataElement satisfies the naming rules for SQL, a database designer can implicitly use that name for a column definition by leaving a null value for the SQL name in ColumnDefinition. If the designer wants to use a different name for the column definition, or the name in the DataElement does not satisfy the naming rules for SQL, the designer can specify a nonnull value for the SQL name in ColumnDefinition; this overrides the name in the DataElement.

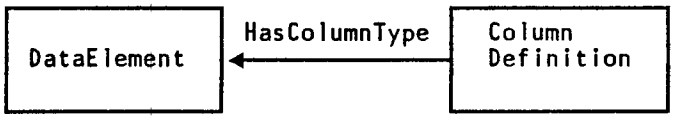

Figure 6. Language-independent DataElement and SQL-specific ColumnDefinition

4.1.2 DBMS-Independent Part and DBMS-Specific Part of Relational Area

The relational database subject area consists of the following parts:

1. The *DBMS-independent part* contains constructs that model features that conceivably could apply to a variety of DBMSs (database management systems), each of which implements SQL.
2. Each *DBMS-specific part* contains constructs that model features that are specific to the implementation (or operating system) of a DBMS. These features are unlikely to apply to other DBMSs. For example, the constructs might model physical storage or information from the DBMS catalog. Each DBMS can have a DBMS-specific part of the relational database subject area. In the current design of the information model, the only DBMSs for which IBM supplies a DBMS-specific part are Database 2 (DB2) for MVS and Database 2 for OS/2. Relationship types connect constructs in a DBMS-specific part and constructs in the DBMS-independent part.

To store all the information needed to generate SQL statements for a particular DBMS, it is usually necessary to populate instances of constructs in the DBMS-independent part and instances of constructs in the part that is specific to that DBMS. For example, in Figure 7, the ColumnDefinition object class contains column attributes that apply to all relational DBMSs, while the MRDColumnDefExt object class contains column attributes that are specific to

Database 2 for MVS; the name "MRD" comes from "MVS Relational Database." Instances of some constructs in the DBMS-independent part can be related to instances of constructs in any number of different DBMS-specific parts.

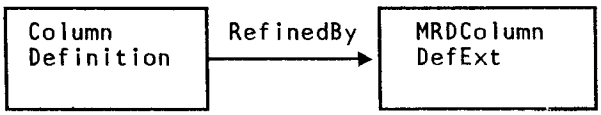

Figure 7. DBMS-independent and DBMS-specific constructs

SQL has many versions, and the DBMS-independent part reflects some of the features that appear in one or more of these versions of SQL:

- The Entry level (and, in a few cases, the full level) of "Database Language SQL," X3.135-1992, American National Standards Institute (ANSI), 1992
- IBM SQL (a product-independent specification within IBM)
- Database 2 for MVS (DB2 for MVS)
- Database 2 for AIX/6000 (DB2/6000)
- Database 2 for OS/2 (DB2/2)
- SQL/Data System (SQL/DS)
- Structured Query Language/400 (SQL/400)

A DBMS might support only a subset of the features that appear in the DBMS-independent part. Therefore, when a tool performs validation (checking) of the DBMS-independent part or generates SQL statements based on the DBMS-independent part, some aspects of the tool's actions might be specific to the DBMS.

4.1.3 Places to Specify Uniqueness (Table vs. Index)

An instance of the KeyDefinition object class represents the definition of a key in a table definition that forms the basis for any number of SQL tables. An SQL key is an ordered set of columns from a table. An instance of a key can serve as the basis for any combination of several possible uses for the key's set of columns, namely a unique key (perhaps the primary key), a foreign key, and an index key.

Different relational DBMSs can have different ways to specify uniqueness. For example, IBM SQL and the ANSI standard for SQL specify a UNIQUE clause in a CREATE TABLE statement. Some old releases of DB2 for MVS only specify a UNIQUE keyword in a CREATE INDEX statement. The current release of DB2 has both ways; a table with a UNIQUE (or PRIMARY KEY) clause but no unique index is valid, but it cannot be populated until a unique index is created. SQL/DS automatically creates a unique index if a CREATE TABLE statement includes a UNIQUE clause. Therefore, if a key definition is unique, a tool that generates SQL statements should specify uniqueness in a way that is appropriate for the target DBMS.

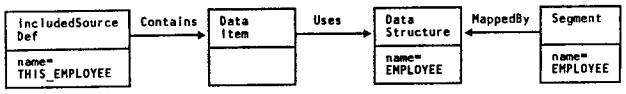

Figure 8. An instance diagram that shows sharing of a structure

4.1.4 Scopes of Name Uniqueness for Relational Database Constraints

SQL/DS requires uniqueness of nonnull names of unique constraints within a *table definition.* IBM SQL requires uniqueness of nonnull names of referential constraints within a *table definition.* Full ANSI SQL is stricter; it requires uniqueness of nonnull names of such constraints within a *schema.*

Accordingly, a constraint assures uniqueness of nonnull names of such constraints in a *table definition.* For a stricter DBMS, a tool can also check uniqueness within a *schema.*

4.2 Sharing of Definitions

The information model supports sharing of definitions by many uses of those definitions.

Sharing is similar but not identical to accommodation of diversity:

- Accommodation of diversity means that one set of constructs (object classes and relationship classes) can represent any of several languages. For example, a structure (which contains data items) can represent a structure in a programming language and/or a record type in a database language.
- Sharing means that one set of *instances* of constructs can apply to any of several uses. For example, a structure instance can apply to several PL/1 variables and also to several IMS segment definitions.

Accommodation of diversity is a prerequisite for sharing of definitions by uses (of definitions) that involve different languages.

Sharing is important, because it lets a customer:

- Establish a central point of control for the definition of some aspect of data, e.g., the fact that an employee has a name, an employee number, a social security number, a salary, a telephone number, etc.
- Ease application developers' use of such a definition.
- Minimize redundancy among definitions.

4.2.1 Sharing of Structures

A *structure* is an ordered set of components. For example, many programming languages (e.g., COBOL, PL/1, and C) allow data items to be structures, which contain other data items. Similarly, IMS segments are structures that contain fields.

A team of application designers might want to record centrally (and with minimal redundancy) that EMPLOYEE contains NAME, SALARY, MANAGER, DEPARTMENT, and other components. Several applications and languages will use this central definition. To accomplish this, instances of constructs in several languages can all share a structure for EMPLOYEE, and that structure can be related to constructs for NAME, SALARY, etc. For example, an IMS segment definition and a PL/1 variable definition can share one EMPLOYEE structure. Figure 8 is an instance diagram that shows a PL/1 variable definition named THIS_EMPLOYEE (depicted by an instance of the IncludedSourceDef object class) and an IMS segment definition named EMPLOYEE (depicted by an instance of the Segment object class) that share a structure named EMPLOYEE. The DataStructure object class has relationships (not shown in the figure) to object classes that represent language-specific information. As an additional level of sharing *within* one language, several PL/1 data items, for example, might all use one structure.

4.2.2 Sharing of Relational Database Definitions Across Database Sites

The information model lets a user define a set of database structures (for tables, views, indexes, constraints, etc.) and instantiate that set for each of several database sites. This sharing minimizes redundancy if a user installs an application and its database structures at each of several database sites. For example, Figure 9 shows a many-to-one relationship class (HasDefinition) between Table (a use of the TableDefinition) and TableDefinition.

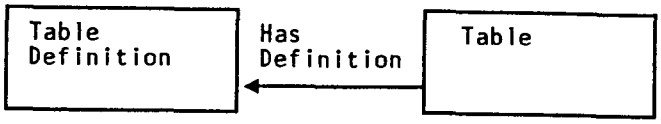

Figure 9. Sharing across database sites

The table definition contains a suggested table name, and the table contains the actual table name at the site. Therefore, information that can refer to names (search conditions in check constraints and selects in views) can appear abstractly (using suggested names) in definitions and concretely (using real names) in site-specific uses.

4.3 Support for Impact Analysis

If an application developer changes the definition of some information, that change can cause changes in other definitions that depend on the first definition. For example, eliminating a column of a table or view can affect views that come from the table or view, which in turn can affect views that come from the affected view. When a developer considers a change in a definition, *impact analysis* is an activity in which the developer finds the proposed change's effects on other definitions. Impact analysis has these benefits:

- The likely impact of a proposed change can be known before the change is made. This can influence a decision on whether to make the change, and it can help in planning for the change.
- If a change requires another change, there is a reduced likelihood that an application developer will forget to make the second change.

The information model uses relationships between objects to support impact analysis. For example, in Figure 10, the ResolvesInPRDB relationship class connects the ViewDefinition and PersistentRelDef object classes. A *persistent relation* is a superclass of table and view. A instance of the ResolvesInPRDB relationship class represents the fact that a view definition comes from a persistent relation definition in the view definition's SELECT clause. A tool for impact analysis can follow this instance to help determine the impact of changes in the definition of a table or view.

Figure 10. Support for impact analysis

5. SUMMARY

We have described the use of IBM's object-oriented information model, which lets a customer share data among various application development tools. We discussed (1) techniques for diagrams (an essential part of communication between a designer and other designers or tool writers), (2) organization of the design of the information model (an essential step when many designers design anything large), and (3) technical content. We hope that these discussions of our experience can be valuable for further design of the information model and for other design efforts, e.g., involving other models or other integration of tools.

ACKNOWLEDGEMENTS

Kevin Cattell, Bill Cromer, Ed Downing, John Garth, Kathie Goldberg, Ken Hardy, Dave Hubbard, Eric Lin, Carole Mayrhofer, Bruce Neuchterlein, Becky Nin, Ahmad Nouri, Chris Porter, and Tom Potok also designed parts of the information model. We thank Kevin Cattell, Leslie Gornig, Kevin Haga, Eric Lin, and Mary Vinopal for reviewing an earlier draft of this paper.

REFERENCES

March, S. T. (Ed.), *Entity-Relationship Approach*, Elsevier Science Publishers, Neth., 1988 (Proc. 6th Intl. Conf. Entity-Relationship Approach, Nov. 1987).

Martin, J., *Principles of Object-Oriented Analysis and Design*, Prentice-Hall, Englewood Cliffs, NJ, 1993.

Moriarty, T. et al., "Which is the 'Right' Data Model for a Given Problem?," in March, S. T. (Ed.), *Entity-Relationship Approach*, Elsevier Science Publishers, Neth., 1988 (Proc. 6th Intl. Conf. Entity-Relationship Approach, Nov. 1987).

Rumbaugh, J. et al., *Object-Oriented Modeling and Design*, Prentice-Hall, Englewood Cliffs, NJ, 1991.

Sowa, J., *Conceptual Structures: Information Processing in Mind and Machine*, Addison-Wesley, Reading, MA, 1984.