Using a CASE Based Repository for Systems Integration

Minder Chen
Department of Decision Sciences
School of Business Administration
George Mason University
Fairfax, VA 22030

Edgar H. Sibley
Information Systems & Systems Engineering Department
George Mason University
Fairfax, VA 22030

ABSTRACT

There is growing interest in frameworks and repositories. Both include databases that store information about resources (for systems development, particularly); however, a framework also implies some reference schema. Frameworks and repositories can be the foundation for data integration in a CASE environment. In this paper, the authors first discuss the objectives and functionalities of a generic systems development environment. A meta-architecture that incorporates these factors is proposed. The structures, contents, and architectures of repository system are then discussed, focusing on enterprise-wide systems for managing information resources. In this paper, the authors examine four types of repository system in terms of their scope and content; *viz*, data dictionaries, project dictionaries, information resources dictionary systems, and future composite repository systems. The authors propose the use of repository systems for the development of an integrated model of an organization and its information systems (IS). This model can be used for organization design/redesign, managing the alignment of IS with strategies and structures, and applications delivery. The managerial issues of setting up a corporate-wide repository system for information resource management are also addressed. Several standardization efforts and recent development are finally discussed.

1. Introduction

A system development environment is a system that generates a large amount of complicated information. Computer-Aided Software (or System) Engineering (CASE) technology involves the use of computer-based information systems (IS) to assist in elicitation, storage, management, control, and analysis of such data. The use of CASE technology is becoming a critical factor for the success of large scale systems projects. A CASE environment allows systems developers to "document and model an information system from its initial user requirements through design and implementation and let them apply tests for consistency, completeness, and conformity to standard" [Chikofsky and Rubenstein, 1988, p. 13].

Most CASE tools support various structured techniques, such as Data Flow Diagrams for structured analysis and Structure Charts for structured design. These structured techniques provide a set of consistent notations (in graphical, textual, and tabular formats) to model target IS. Structured techniques provide "common languages" to facilitate the communications among users, systems analysts, designers, and programmers in a system project. CASE technology enables system developers in applying structured techniques to large-scale systems projects.

First generation CASE tools (e.g., PSL/PSA [Teichroew and Hershey, 1977]) of the early 70s were generally mainframe based and non-graphic in nature, because of the cost of hardware at that time. They generated a great deal of interest in manual techniques for systems development, which resulted in several "methodologies" that survive today. It was then recognized that automated tools were necessary to support these methodologies [Sayani, 1989]. With the advent of graphics capabilities on personal computers (PC) and workstations, graphic-oriented front-end tools were developed to support structured systems analysis and design techniques; e..g, Data Flow Diagrams, Entity Relationship Diagrams, and Structure Charts. However, the information captured in these early graphic tools was unique to the particular tool and not generally transferable from one to another. For example, STRADIS/DRAW, one of the first PC-based CASE products, provides only structured graphics capabilities and does not have a design database for sharing design information among various tools [Gane, 1990].

The second generation CASE tools started to emerge at early 80s. These incorporated better analysis functions to enforce the principles and guidelines of the underlying structured techniques supported by the tools. Information captured in graphics is described by the systems developers in a project dictionary and it can be shared by other CASE tools in the same suite. However, this integration of CASE tools was limited to those from the same vendor and generally within the same project: data importing and exporting are used loosely to couple the project dictionary to add-on tools Excelerator from Index Technology was an example of such a CASE product, as it existed at its first release in 1984 [Index Technology, 1989].

Modern commercial CASE tools still generally do not allow interchange of software design data to other vendors' tools. Primitive download facilities are provided

0073-1129/91/0000/0578$01.00 © 1991 IEEE

by some (e.g., the dictionary import and export facilities in Excelerator), but these do little but capture a *picture* or allow the user (under special license) to import it to special (user written) facilities. In a few cases, vendors may link their tools by mutually-agreed-upon formats of the interchanged data. We shall term this type of exchange interface format a *local* or *limited reference model.* For example, Index Technology's CASE product, Excelerator, can now directly exchange information with TELON, an application generator from Pansophic.

1990's will be an era of systems integration. Business needs to develop integrated and strategic information systems to stay competitive. CASE tools keep IS organizations to deliver systems to meet business needs in time, but organizations are only just starting to adopt CASE tools into their system development processes. A system repository is a foundation that allows integration of CASE tools. Information stored in a repository will be increasingly expanded in scope and repository system will become more "intelligent" to provide effective management of the repository.

In this paper, the authors examine the systems integration problems in a CASE environment by discussing some similarities and differences in the objectives and functionalities of a generic systems development environment. The structures, contents, and architectures of repository system are discussed in Section 3. The focus is on enterprise-wide repository systems for managing information resources. Four types of repository system (data dictionaries, project dictionaries, information resources dictionary systems, and future composite repository systems) are compared. The authors propose the use of repository systems for the development of an integrated model of an organization and its IS. The use of such a model is articulated. The managerial issues of setting up a corporate-wide repository system for information resources management are addressed in Section 6. Several standardization efforts and recent development are discussed as a conclusion.

2. Systems Development Environments

There are several types of storage systems that have been used to capture meta-data about an IS environment. These include various data dictionary systems, some reference schema systems, and various transfer standards. However, before discussing particular technologies and environments, it is valuable to consider the overall objectives and needs of modern development environments.

2.1 Objectives of Systems Development Environments

First, our discussion starts by discussing the development of systems in general, though the ones of interest here are engineering-oriented. The first question of interest is: "What is needed for a modern systems development environment or *workbench*? " While opinions vary, many professionals performing systems development apparently want: "The provision of an architectural *framework* that has many useful tools to aid in the process (of systems development)."

There are some overall requirements that transcend the particular engineering discipline, such as: "To allow many *engineers* to cooperate on the design (including any simulation and prototyping), implementation, testing, and maintenance of the system." This implies secondary requirements: First, the design must be easily available to all authorized personnel. This must be so even when people are distributed geographically. Second, the information must be understandable to the recipient (i.e., couched in a well-defined syntax which has meaningful and unambiguous semantics). Moreover, if the transmission consists of both graphics and text, they must also have a well-defined and mutually agreed syntax and semantics.

In other words, it is necessary to provide a framework or structure that allows the recipient to use the information without having to ask for clarification or needing to reinterpret it. However, this is still a somewhat general definition that does not show what must and what must not be included in such a framework. The essence of this definition is, however, the need for a well-understood framework or workbench that contains tools that are useful in the particular environment. AD/Cycle [IBM, 1989], IBM's framework for developing and maintaining software applications, is one such effort. But, one may ask "Why don't you build a self-contained environment?"

One can imagine an excellent environment that is "solidly" built and absolutely self-contained; indeed, such workbenches actually exist in many engineering design and prototyping environments today, and they are very efficient in doing their specific function. However, the difficulty with them is a tendency for their having to be replaced ("thrown away") as they become obsolete. This is particular true when the business changes and the workbench no longer does its job efficiently or its activities are partly incorporated into other functions, etc. Thus, there is no reason why an excellent tool should remain "the best" as new ideas cause additions in the form of "external tools" that cannot be incorporated easily, though they are much cheaper to purchase and incorporate than to develop for (and add on to) the system.

The provision of a flexible framework has been addressed in the statement of objectives for the US Department of Defenses' EIS (Engineering Information System). It was defined by the Institute for Defense Analysis as having nine major objectives:

1. Provide a cost-effective tool integration framework,
2. Encourage portability of tools,
3. Encourage a uniform design environment,
4. Facilitate exchange of design information,
5. Support design management and reuse,
6. Achieve widespread acceptance of EIS,
7. Be adaptable to future engineering,
8. Be extensible to other disciplines, and
9. Be compatible for transition from existing design environments.

These involve dealing with a wide set of user perspectives; the system is to provide a highly flexible environment for passing information between tools and various classes of *engineering users* who need to exchange data, from management to engineer's assistant, and across a wide variation of *engineering* disciplines; e.g., mechanical, electronic, etc. The basic architecture of such a system should provide a model, if not a total environment for software definition and production, etc., One of the disciplines (objective #8) serviced by an EIS could be "Software Engineering".

The first five objectives could be rewritten slightly to describe such an environment by replacing the terms tools, design, module, and engineering by software tool, software design, software module, and software engineering respectively. Two of these objectives (#7 and #9) for a framework are intended to ensure that a system development environment based on the framework can adapt to future changes and needs. Achieving these two objectives have proved to be extremely difficult in any environment, and software engineering is no exception. The essence of the architecture of the EIS revolves around these two objectives, and the rest then seem to fold into the overall framework.

In summary, these objectives, when applied in the software development context, are driven by two essential needs of IS organizations: the need to employ a stable platform which can cope with changing information technologies, and the need to integrate systems development and maintenance tools.

2.2 Framework Requirements Groupings

The groupings of requirements of EIS were seen by the team of EIS contractors/developers (led by Honeywell) as a set of high level functions that provide [Honeywell, 1990]:

1. Tool and workstation integration
2. Data exchange between nodes containing EIS
3. Management and control of engineering projects
4. Information management
5. External systems interface
6. Object management system
7. Distributed operating system

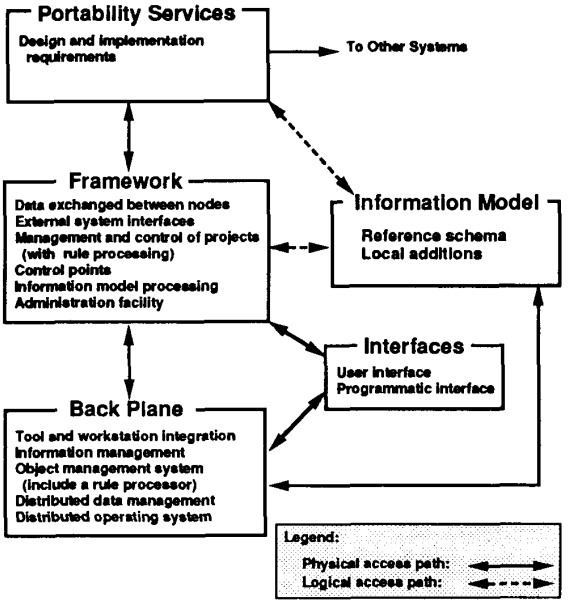

Figure 1: A Meta-Architecture of Systems Development Environments

8. Distributed data management facilities
9. Engineering information model and its processing
10. Rule processing
11. Control points
12. User interfaces
13. EIS Administration
14. Programmatic interfaces
15. Design and implementation requirements

It will be seen that these do not really map simply into a set of user functions, or even their needs and activities, but they are a good basis on which to form a "meta-architecture", i.e., an architecture that aided in the creation of the architecture of a framework. One way to of organizing them is shown in Figure 1.

A Back Plane is a set of data and control interfaces normally found within the interfaces and functions of modern operating systems and database facilities, extended to modern principles of distribution and object management. Thus these deal with requirements numbers 1, 4, 6, 7, 8, and the functionality of 10.

The major Framework consists of the additions that provide activities essential for implementing the Reference Model and to allow the local system administration to tailor the model to local needs (within the overall model). These services are the Engineering Environment Services in the EIS, but here we term them the Framework. The

Framework therefore is a provider of interface requirements numbers 2, 3, 5, 9, 11, 13, and some parts of 10.

The four other parts are: 1) User Interfaces (number 12) are intended to allow user needs to be satisfied by an extremely tailorable interface. It must use the modern concepts of human factors to give the user maximum flexibility in defining special I/O formatting for new graphical and other multi-media mechanisms. 2) Portability Services, which have the capabilities of some requirements number 5 and 7, but primarily those for number 15. These must also be general, and allow for translation of formats and methods between different environments. A high degree of flexibility of user interfaces may make portability hard to achieve. 3) Programmatic Interfaces (number 14) permeate all parts of the system, particularly via the back plane and framework facilities. These allow programmers to provide additional "functions" so that the local system can be more useful to its group of developers. Again, however, such additions make portability more difficult to achieve. 4) The Reference Schema, which is the key to all capabilities to integrate the services. This is the place where all the original *overall* tailoring is recorded. However, all local additions or modifications must be retained here to allow for easier (if no simple) translation between nodes.

For the purposes of this paper, the architecture will be slightly reduced. The Programmatic and User Interface will be considered to be a part of the Back Plane or Framework (depending on the degree of sophistication of the tool/system being considered). Thus the systems are analyzed here on the assumption that the architecture consists of three major portions: Back Plane, Framework, and Portability Services, with the Reference and Local Schema as receptacles of the major model definitions. It should be noted that an actual design (or set of specific system requirements) can be stored in the same object management system as its definition, though the implementer may resort to a similar one in order to keep the parts separate. Hence the term meta-architecture is used in our figure to show that this is an architecture to aid in building *other* architectures of IS.

3. The Structures, Contents, and Services of a Repository System

Various data generated in the Systems Development Life Cycle (SDLC) are inter-related, therefore, the structures of a repository are very important in managing the complexity. The content of a repository system is expanding in scope because of the increased functionalities that have been allocated to the repository. A set of service functions has been provided in advanced repository systems to assist the integration of CASE and other productivity tools within them. The details of structures, contents, and services of a repository system will now be discussed.

3.1 The Structure of a Repository System

A repository may be considered to be a specialized database management system for a system development environment. A meta-model is the database schema for such a repository. It describes the structure and meaning of data stored in the repository and therefore provides a basis for sharing data among CASE tools. A meta-meta-model will then be the syntactic and semantic constructs for defining meta-models [Lefkovits, Sibley, and Lefkovits, 1983; Welke, 1989]. Two major meta-meta-models used in existing CASE repositories are the Entity-Relationship-Attribute (ERA) model and the object-oriented model.

The ERA model was originally developed for conceptual data base design [Chen, 1976; Teory, Yang, and Fry, 1986]. An ERA model provides an intuitive way for describing a data model of data base applications, particularly for enterprise modeling. It has been used by many CASE vendors and standard committees; e.g., as a meta-meta-model to define the meta-model of a CASE repository. The Information Resource Dictionary Systems (IRDS) approved by American National Standard Institute (ANSI) on October 1988 also use the ERA model. Several commercial repository systems are currently supporting ANSI's IRDS. The structure of an IRDS consists of several layers that represent different levels of abstraction [Law, 1987; Goldfine and Konig, 1988; Welke 1989].

The first layer (i.e., the highest level of abstraction) is the Information Resource Dictionary (IRD) Schema Description Layer; it defines a *meta-meta-model* of a repository. This layer defines four subtypes of a Meta Object Type: Meta Entity Type, Meta Relation Type, Meta Attribute Type, and Meta Attribute Group Type (i.e., a set of Meta Attribute Types). These Meta Entity Types are predefined in most of IRDS implementations and are used to describe and control the IRD Schema Layer, the second layer.

The IRD Schema Layer consists of a set of Entity Types (e.g., Data Item and Record), Relationship Types (e.g., Record-Structure-Relation), Attribute Types (e.g., Length), and Attribute Group Type (e.g., Value-Range). It defines the *meta-models* of an IRDS. A Relation Type consists of two or more than two sets of Entity Types (e.g., Record-Structure-Relation can be expressed as Record contains Record or Data Item). Attribute Types and Attribute Group Type can be attached to Entity Types and Relation Types (e.g., Data Item has Length). The IRD Schema Layer is used to describe and control the IRD Meta-Data Layer, the third layer. The IRD Meta-Data Layer consists of a set of Entities (e.g., Product ID, Product Name, Product Record),

Relationships (Product Record contains Product ID and Product Name), and Attributes and Attribute Groups (e.g., "Length: 8 Character" and "Value-Range: 0000000-ZZZZZZZ"), which are instances of the meta-model defined in the Schema Layer. The Meta-Data Layer are used to model the target application systems, the resulting models are called meta-data. The storage and manipulation of data in production IS defined by the meta-data usually are functions of a production data base management system.

Most objects created in a software development environment are composite objects. For instance, a data flow diagram (DFD) may consist of a set of data stores, information flows, processes, and external entities, as well as graphical specification of objects in the diagram. An object contained in a composite object may also be a composite object; for example, a process in a DFD diagram can be further decomposed into a detailed sub-level DFD diagram. The object-oriented approach supports encapsulation of data and procedures into a system viewed as a collection of objects interacting via messages. Message passing among objects implies dynamic relationships in the system, while a static relationship can be represented an instance of the relationship class which define the "schema" of the relationship. The creation and manipulation of static relationships can be implemented through a set of procedures (i.e., methods) associated with the relationship class. However, currently object-oriented development systems such as object-oriented languages and object-oriented database management systems are not really capable of supporting the development of a fully-integrated CASE environment. One major problem that the authors foresee in an object environment is the management of complicated relationships, which is critical in a CASE environment. Object-oriented systems do not have a standard set of operations for defining and manipulating such relations, especially those that are more than two-part. However, they do have many features that can make the development and integration of CASE tools easier; e.g., the inheritance feature encourages abstraction and reusability, which are basic principles of software engineering.

3.2 The Contents of a Repository

A repository could store all the information relevant to the management, development, operation, and maintenance of information systems resources in a corporation. Thus it is potentially enormous. The contents of a repository can be classified into two general categories.

The first category is development information generated from the systems development process, which includes include business models, logical models of information systems, physical models of the implementation systems, and system documentation. Examples of business models include business functions, business rules, organization structures, organization goals, and critical success factors. Both data and procedure specifications should be stored; however, currently there is no standard for specifying detailed processing procedures in an IS. Pseudo-code which may use structured English with predefined program control structures, and action diagrams are examples of popular ways of specifying procedures that have found limited use today.

The second category include information about development processes and methodologies. Project management information should be incorporated into the repository to assist in project management. Organization-wide system development process models and policies should also be included. A set of meta-CASE tools, such as the SEE System [The Delphi Group, 1989], can be employed; they use this information to control the development process, to enforce policies and standards, and to assist the invocation of appropriate tools for developers under special conditions. An intelligent repository should be able to represent and enforce development methodologies, which are usually defined as a part of a meta-model based on the discipline in a methodology using the semantic constructs provided in the meta-meta-model. For example, IBM's AD/Cycle [IBM, 1989], a facility called the *Process Manager* can select and trigger development activities based on a process model; i.e., a description of the system development methodology.

The actual contents of a repository are largely determined by the systems development methodologies and tools used by an IS organization. The information in a CASE repository can be elicited from the user and then represented to the user in various formats: graphics, tables, structured textual languages, and natural languages [Chen 1988; Poston, 1989]. Most information stored in the formats of graphics, tables, and structured text is interchangeable. The complete information to be transferred includes symbols, their characteristics, locations, and organizations [IEEE, 1989].

3.3 The Services of a Repository System

A repository system with an open architecture will provide a well-designed set of human interfaces, so that it can be easily used to extend the meta-model to integrate new tools. A repository should provide different types of services to serve various users; these again map into the set of functions in Section 2.2 and include:

1. *Import/export* facility will allow a repository manager to extract data to generate a flat-file to be exported and used by other systems in batch mode. Similarly, an importing facility is used to consolidate any data generated by and received from another system. The format for import/export is usually predefined, but it may sometimes

be tailor-made. This extension is achieved if the repository manager can define a set of transformation rules to convert the meta-data of one meta-model into that of another. However, there are limits to the degree of transformation allowed. Such a transformation mechanism can assist both forward and reverse engineering (i.e., re-engineering).

2. *User interface* includes a command language interface. It is allows users to define structured statements that interacts with a repository at both the Schema and Meta-Data Layers, either in batch or on-line mode. A menu- and form-driven interface is also currently being proposed as part of a future IRDS standard. Several standards are being developed to include windows and graphics in the repository user's interface, to make it possible to provide the same look-and-feel to all users.

3. *Application Program Interface (API)* is a set of functions made available to tools developers who are building CASE tools using the repository. With the API, CASE tools developers will be able to concentrate on the functionalities and interfaces of the tools instead of spending their effort implementing the database management functions of a repository. This type of service is sometimes called a tool service which is intended to provide *tool wrappers*. This, in general, means that the data structures used by and resulting from the tool operations must be defined to the underlying framework (e.g., an object-oriented database) along with a statement of any transformation needed. CASE tools and other productivity tools may use the API service of a repository system in such a way that it becomes an active component in the integrated systems development environment.

4. *Reports and query facilities* are provided by repository system to assist repository managers. Reports or queries can be entered in various formats and results provided in various forms, such as tables, structured statements, or hierarchical graphics. Statistics on the repository itself, such as number of objects and retained relationships, can be also generated. Reports which describe the links among entities in a repository can be used for impact analysis; e.g., when attempting to determine the effect of a change in some on extant systems containing compiled programs.

5. *Customization and methodology prototyping facility* (i.e., extensibility) will be a standard feature of future repository systems. For example, extensibility is part of the ANSI IRDS standard. Users can incorporate their own extended meta-models to the repository by adding new entity types, relation types, and attribute types into an existing meta-model. The major difficulty occurs in trying to add the semantics. A set of basic functions should be provided by the repository system so that repository managers can customize the repository to suit their own system development environments.

6. *Software metrics and project management tools* are indispensable for measuring system projects' progress and their impacts. The increasing use of CASE tools allows us to capture the system analysis and design information. This make the systems development processes more amendable to quantitative analysis through the calculation of various software metrics [Card and Glass, 1989]. Such software metrics can be used with project management tools for software project scheduling, quality control, and resource allocation.

3.4 Hardware Platforms of CASE Repository

Many existing CASE tools are using PCs to store project dictionaries. The support of teamwork in systems development relies heavily on the project manager's ability to replicate and divide relevant parts of a project dictionary into different subsets and export them to individual project members. Individual project dictionaries could be merged and consolidated into a complete project dictionary through an importing utility. Conflicting meta-data would then have to be resolved by project managers. Excelerator's XLDictionary is an example of a project dictionary system.

Several CASE vendors are moving their project dictionaries onto Local Area Networks (LAN). Concurrency control must then be available to support parallel development efforts. Several CASE tools have employed a two-tier or a three-tier architecture to manage an enterprise-wide repository. IBM's AD/Cycle [IBM, 1989] and Texas Instrument's IEF [TI, 1989] are examples. In a three-tier architecture, a centralized repository resides on a mainframe or minicomputer, a project dictionary is stored in a file server on a LAN, and individual working dictionaries (which may be optional) are stored on individual workstations. By using check-in and check-out procedures, repository information can be moved downward or upward between a centralized repository and project dictionaries, and between project dictionaries and individual dictionaries. However, for dynamic and distributed development environments, such as the aerospace industry, the employment of a distributed repository systems is a necessity.

4. Evolution of Repository Technology

The use of a repository for system integration is an evolving concept. The authors have classified this concept into four categories. The discussion here is based on the common characteristics of repository system in each category; of course, exception exist among actual implementations.

4.1 Data Dictionary Systems

A Data Dictionary System (DDS) contains an information model for a given reference schema that may only allow one or a few special database management systems (DBMS) or their add-on tools to interface with them. The meta-meta-model is often based on either a relational or ERA model.

A DDS can therefore be treated as a specialized database system for one or many DBMS. It contains information about the database schema (e.g., definition of columns, tables, views, and indexes in a relational database), access rights, and statistics of the databases. Sometimes it also contains rules for enforcing data integrity. Database statistics and index information can be used for query optimization. Authorization subsystem may also use the information about users' access rights to grant or deny service. In a relational database system that contains an internal data dictionary, information is usually stored in relational tables; thus users with appropriate access rights can access the dictionary information in exactly the same way as the database itself, e.g., through the SQL language [Sibley, 1986].

The idea of active versus passive dictionaries is essentially determined by tight versus loose integration of such dictionaries with DBMSs [Lefkovits, Sibley, and Lefkovits, 1983]. In an active DDS, systems programmers can also use data dictionary information to develop add-on facilities, such as a form editor and a report generator. To ensure the integrity of the data dictionary, UPDATE, DELETE, and INSERT operations on a data dictionary are created automatically by the execution of the data definition languages (e.g., when initiating a CREATE table operation). The use of such types of integrated DDS are usually limited to database related applications; however, stand-alone dictionaries, which are not associated with any particular DBMS, can also be used as repositories for other information resources. Data Manager and several commercial products are of this nature [Van Duyn, 1982].

4.2 Project Dictionary Systems

A Project Dictionary System (PDS) is a key component of several CASE and productivity tools. Excelerator's XLDictionary [Index Technology, 1989] is an example of such a PDS. The information stored in a project dictionary is usually associated only with a specific application project. This dictionary is an active component in a CASE environment: information created through analysis and design tools, as well as screen and report painting facilities are directly stored in the project dictionary. Information on an object, once defined in the project, can be shared with other tools in the CASE environment throughout the project lifecycle.

The meta-meta-model of most project dictionaries are based on either the ERA or object-oriented model. The first is more appropriate for data processing applications, while the object-oriented model is popular for real-time, scientific, engineering, and embedded application environments. The meta-model of a project dictionary can be viewed as a conceptual schema of the database for system projects. A set of CASE tools (e.g., data flow diagramming and entity-relationship diagramming tools) from the same vendor is integrated through the PDS. The import/export facility can be potentially used with CASE tools from other vendors. A tighter interface (i.e., a bridge) to other CASE tools can be built using the API of the PDS. Several PDSs can be extended in this way via customization facilities (e.g., Customizer for Excelerator); moreover, the work of the CDIF as discussed below is expected to provide standard interfaces that will enhance this capability.

4.3 Information Resource Dictionary Systems

The Information Resource Dictionary Systems (IRDS) has a given reference schema and allows local additions (i.e., IRDS is extensible). The "Minimal Schema" is a meta-model which consists of a predefined schema structures and it must be provided by a complying implementation of the IRDS. An optional module, called the Basic Functional Schema, is also available [Law, 1988]. The facilities in the IRDS can be used to manage both the given schema and its local additions. Since additions may introduce discrepancies at the meta-model level, they may or may not be exported to other IRDS.

An IRDS can be used as a corporate-wide repository system to store information for various SDLC stages and for the management of information resources. It is designed to support multiple systems projects by providing a single point of control for the management of information about information resources. The current definition of the standard IRDS, however, only supports binary relationships.

4.4 Future Composite Repository Systems

In the future, repository systems will have extended functionality and content. Information about the IS development phases and its usage, in conjunction with all the related resources will be described in the repository. In addition to traditional data processing systems, future information systems will include development and management information for decision support systems (DSS), knowledge base systems, executive information systems, etc. Therefore, future repositories should store information about decision models and knowledge bases: i.e., "meta-knowledge" or knowledge-about-knowledge. Existing meta-models must then be extended to incorporate meta-knowledge. Future repository systems may therefore not only be used for integrating CASE environments, but also for other systems development environments, such as expert systems shells and DSS

generators, providing additional facilities to help users cope with this complexity.

Future repository systems will use an extended ERA model with n-ary relationships and will support inheritance. An object-oriented meta-meta-model will co-exist with the extended ERA model. Actually, the extended ERA meta-meta-model can be implemented in an object model [Chen, 1988]. The use of an object model will allow users to define new subtypes of the Meta Object Type to extend an existing meta-meta-model. However, the extension of a meta-meta-model should be performed with extra precaution, because it might introduce incompatibility among tools which utilize the repository.

5. Repositories for System Integration

A repository system, just like a common database system, may provide the basis for the integration of various application systems through a unified logical data base model and a shared data storage mechanism. A repository system can be used to help organizations achieve greater system integration through better development, management, and usage of information resources.

5.1 CASE Tools Integration

There are three dimensions of integration in a CASE environment: Presentation Integration, Control Integration, and Data Integration [Wasserman, 1989]. Each dimension can be further refined, providing different degrees of integration. Presentation integration includes the use of standard window systems to allow standardization of the look-and-feel of various tools. Control integration is the focus of many Integrated Project Support Environment (IPSE) implementations. The mechanisms for control integration are: explicit message, time-trigger coordination, trigger, and message server [Aranow, 1990]. The highest degree of control is in the use of the software process models to manage the invocation and management of various tools. Data integration is the foundation for the integration of CASE tools. There are four different methods of data integration [IEEE, 1989]: interacting directly between tools, exchanging data through files using import and export facilities, sharing a common data base though a repository system, and communicating through message passing under an open system architecture. A common repository provides a better mechanism for control and sharing of information.

CASE integration can be also classified into: vertical integration, horizontal integration, and cross-project integration [TI, 1989]. Vertical integration ensures the integrity and consistency of information generated from various stages of the system development life cycle. Forward engineering, reverse engineering, and requirements tracing are mechanism to achieve vertical integration. Horizontal integration maintains integrity and consistency within each SDLC stage when multiple modeling methods (i.e., data, process, and object-oriented modeling) are used. A comprehensive meta-model which incorporates multiple modeling perspectives and the use of completeness and consistency checking rules across various model perspectives are ways to enforce horizontal integrity. Cross-project integration manages the sharing of information generated from many systems project via a centralized repository. Information model sharing and distribution, actual and trial model merging, version control, model downloading and uploading, and concurrency and security controls are examples of functions which facilitate enterprise wide cross-product integration [TI, 1990].

5.2 Systems Integration and Information Resource Management

A repository can be used for the integration of application systems and the management of information resources. Most existing repository systems deal with the system development process. A repository will also serve as a software back plane (also called a software bus) that allows systems developers to mix and match multiple CASE tools from multiple vendors. This provides greater portability of an application system.

ERA-based repository systems have been used to construct models for managing information resources and for strategic systems planning in an organization [Law, 1987]. Chen, Nunamaker, and Weber [1990] proposed the use of information in a repository for both building information systems, and delivering information to the end user. The information that system developers collected in systems planning and the requirement definition phases are basically user models of what target systems should do.

Potentially both business and information systems models will be captured and integrated in a repository system. The business models define a user perspective of the enterprise and various business functions and processes; these are linked to information systems that support the business. The access methods of various operational information systems are stored in the information system models so that users can trigger the stored access procedures to launch applications. Eventually, through the development of integrated models in the repository, end users will be able to access information technologies from a business perspective; i.e., without knowing the physical location of the resource or any specific commands. A high level of transparency to the user can thus be achieved.

6. Implementation of a Repository System

Who will need a repository? Companies who want to develop a sound IS infrastructure so that information can be developed and delivered to shorten the time-to-market of information-technology-based products or services. A good repository system will allow users to mix and match the best of the CASE tools for systems development. Multiple incompatible and inconsistent dictionaries systems can be eliminated. Sharing and reuse of system development data can be enhanced with an unified meta-model in the repository. However, it is necessary that the basic meta-model does not change radically from location to location or in time. Thus many organizations (e.g., the IRDS groups and EIS developers) are providing a basic reference model that may be modified to suit the user.

A repository management group should be established to be responsible to: 1) establish naming conventions, 2) manage version controls and configuration management, 3) establish standards or guidelines for systems development methodologies and tools at various stages of the system development life cycle, 4) define a meta-model for the organization through an evolutionary process, 5) work with the group in the system development processes, and 6) maintain an up-to-date business model by working closely with business managers.

The definition of business models should be customized to fit the characteristics of the organization, instead of using predefined business models. Managerial commitment and participation are important to ensure that the business models reflect all managers' views of the enterprise. The user involvement is critical to the success of CASE implementation [Friesen and Orlikowski, 1989]. Thus an extendible and customizable repository system is needed, where terminology and concepts of by business managers is employed; this will facilitate communication between business managers and system personnel.

The benefit of a repository is long-term. Top management support and commitment in allocating necessary resources for the purchase and implementation of a repository system and its related tools must be secured. Populating an empty repository is a tremendous effort; the use of conversion facilities that extract information from existing documents, programs, files, and dictionaries and place it into the repository is needed to assist in building the repository.

Currently, the cost of installing a repository system is very high [McClure, 1990]. An incremental approach to populating the repository may therefore prove better. Companies who do not have the expertise may start with the installation of a data dictionary system for their database environments and the employment of project dictionaries in a CASE environment for their systems development efforts. An enterprise-wide repository system can then be developed in a gradual manner, by extending the scope of the project dictionary to incorporate strategic planning and IS planning information, as well as integrating and consolidating all the existing project dictionaries.

7. Standards for Repositories

There are several standardization efforts that are related to repository systems and CASE tools. For example, as previously stated, the Electronic Industries Association (EIA) has a CDIF (CASE Data Interchange Format) committee with a large number of member organizations [EIA, 1990]. The purpose of CDIF is to provide a methodology-independent representation for the transfer of semantic and graphic information between CASE tools. these efforts focus on information exchange among CASE tools such that systems development information may be ported from one CASE environment to another. The CDIF committee's concern is therefore integration across different repository systems.

The *de facto* standard of repository systems for business applications is IBM's Repository Manager under its AD/Cycle, a framework for integrating a set of system development life cycle support tools. The mechanism for integrating these tools is through the Repository Manager. AD/Cycle contains an Information Model which incorporates a very detailed meta-model. Developing standards at the meta-model level makes it possible to integrate CASE tools from various vendors.

8. Conclusions

The contents of future repository will be much greater than that in today's dictionaries. Repository systems will become more "intelligent" to aid in the management of the repository and will provide more powerful facilities to integrate various CASE tools. Business will be able to capitalize on CASE technologies that help develop integrated systems for strategic purposes, because enterprise models now can be used to facilitate the alignment of information systems to business strategies. Systems development and maintenance productivity can be greatly improved by using high quality reusable components in the repository. IS organizations will then be able to deliver critical systems in time to meet business needs.

In the foreseeable future, a new alternative to building an application in-house or purchasing an application package will be to acquire the meta-data of an application from a vendor. After revising the meta-data to conform to the specific user's business policies and procedures, the user can load the meta-data to an integrated CASE environment to generate a customized business information system. This alternative is appropriate when the user does not have enough expertise in the business

domain and when changes in the domain in future are very likely. The ramifications of repository technology to CASE vendors and IS organizations have just started to surface. Currently the authors are working on the use of repository systems for integrated organization and information modeling. Using integrated organization and information models as a basis for systems development, we can ensure the concordance of business and information systems.

8. References

Aranow, E., "Is CASE Too Immature for Real Integration," *Software Magazine*, May 19, 1990, pp. 89-96.

Card, D. N., with Glass, R. L., *Measuring Software Design Quality,* Prentice-Hall, Inc., 1990.

Chen, M., *The Integration of Organization and Information Systems Modeling: A Metasystem Approach to the Generation of Group Decision Support Systems and Computer-Aided Software Engineering,* unpublished doctoral dissertation, The University of Arizona, 1988.

Chen, M., Nunamaker, J. F., and Weber, E. S., "The Use of Integrated Organization and Information Systems Models in Building and Delivering Business Application Systems," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 1., No. 3, 1989, pp. 406-409.

Chen, P. P., "The ER Model Toward a Unified View of Data," *ACM TODS,* Vol. 1, No. 1, 1976, pp. 9-36.

EIA, *CDIF Organization and Procedure Manual*, CDIF-DOC-N2-V1, January 1990.

Chikofsky, E. J. and Rubenstein, R. L., "CASE: Reliability Engineering for Information Systems," *IEEE Software*, March 1988, pp. 11-16.

The Delphi Group, *SEE System Overview,* San Diego, CA, 1989.

Friesen, M. E. and Orlikowski, W. J., "Assimilating CASE Tools in Organizations: An Empirical Study of the Process and Context of CASE Tools," *CISR WP* No. 199, Massachusetts Institute of Technology, 1989.

Gane, C., *Computer-Aided Software Engineering*, Englewood Cliffs, NJ: Prentice-Hall Inc., 1990.

Goldfine, A. and Konig, P., *A Technical Overview of the Information Resource Dictionary System*, 2nd ed., National Bureau of Standards, NBSIR 88-3700, January 1988.

Honeywell, "Underlying Framework of PREIS", *EIS Update*, Vol. 3, No. 1, August, 1990, pp. 1-3.

IBM, *AD/Cycle Concepts*, GC26-4531-0.

IEEE, *A Standard Reference model for Computing System Tool Interconnections - Draft,* IEEE Computer Society's Task Force on Professional Computing Tools, June 1989.

Index Technology, Co., *Excelerator Facilities and Functions Reference Guide,* Release 1.9, 1989.

Law, M. H., *Guide to Information Resource Dictionary System Applications: General Concepts and Strategic Systems Planning*, NBS Special Publication 500-152, 1988.

Lefkovits, H. C., Sibley, E. H., and Lefkovits, S. L., *Information Resource/Data DIctionary Systems,* Q.E.D. Information Sciences, Inc., 1983.

McClure, C., "Preparing for Repositories," *System Builder*, August/September 1990, pp. 35-39.

Poston, B., "Proposed Standard Eases Tool Interconnection," *IEEE Software,* Nov. 1989, pp. 69-70.

Sayani, H. H., "Repository As Unifier in an Adaptive CASE Environment," working paper, Advanced Systems Technology Corporation (ASTEC), Greenbelt, MD, 1989.

Sibley, H. E., "An Expert Database System Architecture Based on an Active and Extensible Dictionary System," *Expert Database Systems: Proceedings from the First International Workshop,* The Benjamin/Cummings Publishing Co., Inc., pp. 401-422.

Teichroew, D. and Hershey, E. A., III, "PS/PSA: A COmputer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems," *IEEE Transactions of Software Engineering,* Vol. SE-3, No. 1, January 1997, pp. 41-48.

Texas Instruments, *Texas Instruments Information Engineering Facility (IEF)*, 1989.

Texas Instruments, *Information Engineering Facility: Methodology Overview*, second edition, January 1990.

Teory, T. J., Yang, D., and Fry, J. P., "A Logical Design Methodology for Relational Database Using the Extended Entity-Relationship Model," *ACM Computing Survey,* Vol. 18, No. 2, June 1986, pp. 197-222.

Van Duyn, J., *Developing a Data Dictionary System,* NJ: Prentice-Hall, 1982.

Wasserman, A., "The Architecture of CASE Environment," *CASE Outlook*, Vol. 89, No. 2., 1989.

Welke, R. J., "Meta Systems on Meta Models," *CASE Outlook*, Dec. 1989, pp. 35-44.