CHAPTER 6

INTEGRATION APPROACHES VIA DBMS

6.1 INTRODUCTION

Existing engineering environments generally comprise incompatible tools and data servers linked together in a "Rube Goldberg" fashion to provide the semblance of integration. They rely on multiple translators, ad hoc methods of passing data between tools, and manual methods of ensuring consistency of the results. Often this requires the assistance of an expert for each tool, database, or hardware platform to provide expertise in using that component. Incompatibilities exist in data models, programming languages, hardware platforms, underlying data representations, and user interfaces.

Information interchange between engineering organizations is similarly hampered because of external incompatibilities between the tools and databases of the organizations. Although some syntactic problems are being addressed by standardization efforts, semantic incompatibilities remain. The ability to interchange information is based on agreements between the participants. Currently, there is no common representation of engineering objects, particularly of representations of behavior, nor is there agreement on the meanings of engineering objects or operations.

But, engineering organizations often have large investments in facilities hardware, user training and databases. In addition they have cultures and long-standing methodologies that would be costly or infeasible to change. The solution to the integration problem must preserve these investments as well as be extendible to incorporate new components. There is no guarantee that new components will be compatible with each other or with existing tools and databases.

The current lack of integration among engineering tools and databases is discussed in Section 5.2. Some efforts are addressing this situation, notably frameworks to integrate engineering tools being developed by EDA and Atherton Technologies and earlier efforts to integrate heterogeneous databases at INRIA, CCA, and Honeywell. The Air Force EIS project is also attempting to produce a framework that integrates engineering and administrative tools, DBMSs and file systems.
There are many deficiencies in existing models to support integration. For example, approaches that integrate existing tools and databases or extend environments to incorporate new tools and databases by modifying the tools or data are too costly. They are likely to result in a loss of vendor support for the components and the introduction of errors into the software or databases. What is needed are frameworks that integrate components without compromising their autonomy. It is believed that only "federated" systems in which the component tools and databases are left intact are practical for large engineering organizations.

The frameworks must also provide both conceptual models and integrating services. The conceptual models provide data and execution models for the federated systems that describe the structure and behavior of the components and of the federated system itself. They provide a homogeneous layer over heterogeneous underlying components. Integrating services must provide execution control (i.e., procedure invocation and management, transaction management) and facilities to tailor the resulting systems to the needs of specific organizations.

Recommended research in modeling methodology to support integration of heterogeneous engineering data modeling tools and databases include:

1. **Conceptual Models.** Research is needed to develop powerful meta-models for describing heterogeneous components of federated systems for engineering. They must describe not only engineering and administrative data but also engineering and management processes or procedures that comprise the resulting integrated systems.

2. **Execution Models.** This research would provide execution paradigms to support execution in a distributed, heterogeneous environment. The execution models must provide for parallel processing in support of collaborative transactions, as well as traditional database transactions. Moreover the research would result in optimization strategies over the heterogeneous implementations that make up the federated system.

3. ** Integrating Services.** This research would provide the design and prototype implementations of services to integrate heterogeneous software components and databases. Needed services include both mappings of requests for data access to the underlying database servers and file systems and execution services that include procedure invocation and execution control (including transaction management) in
distributed environments.

4. **Performance Measurement and Enhancement.** This research would provide metrics for measuring performance of federated systems to support engineering and extensible optimizers to improve performance where operations are implemented by heterogeneous procedures and data servers.

6.2 **INFORMATION INTEGRATION**

A measure of the effectiveness of information integration is whether a user can get the right data, at the right place, at the right time, in the right form, and at the right cost. The user's limitations are based only on his need to know, store, modify and otherwise access the data. Key elements of information integration include an infrastructure composed of:

* an information architecture with:
  - single logical database
  - location transparency
  - physical transparency
* an organization to control and manage the information and take advantage of the available integration tools.
* a software architecture and set of appropriate tools.
* a hardware architecture of appropriate levels of computers and peripherals.

The benefits of information integration are many, and include such areas as:

* cost reductions through
  - increased productivity
  - reduced time for locating data
  - minimize data redundancy
* reduced product development time.
* improved quality through
  - reduced development changes
  - improved change control
* streamlined operation.
* formalized communication.
This describes the characteristics and end results of integrating technical and business data, the process and metric of integration, and the techniques and tools of integration. The results are based on findings in ref. Fong.

Characteristics of the Data

The characteristics of technical data and business data differ in degree.

Data Characteristics: Technical and business data differ from one another only in a matter of degree.

Data Management: Technical data management requires mechanisms that are not required in most current business applications. These mechanisms include version and configuration control, data, access to the data at different levels of abstraction to support a variety of presentation capabilities, support of rigorous release procedures, and authentication of the data. However, some office data (for example, "what-if" analysis) seem to be more closely aligned in its data management requirements with technical data than with business data.

Data Transactions: Technical data and business data have markedly different transaction duration and rate requirements. Further, the consistency problem, as viewed by the application, can be tackled through different approaches. For example, version control in the technical case versus locking in the business case.

Database Management: The traditionally more demanding data management requirements of technical data (data types, multiple levels of abstraction, authentication, version control) will be of use in the advanced business application of the future. This will further lessen the perceived differences between business and technical data. It is questionable as to whether a single database management technology could satisfactorily meet the functional and performance needs of both business and technical data systems. If so, then perhaps the distinctions between technical and business data will be seen as insignificant and immaterial.

Organizational Considerations

Organizational considerations are significant determinants of the success of an integration project.

Eliminate Sub-Optimal (Parochial) Solutions: Typical integration efforts uncover
duplication of efforts, and recommend shifts in workloads and responsibilities to resolve the problems resulting from sub-optimization in the existing organizations. Each organization typically has been organized and automated without concern for the overall strategic (profit driven) goals of the corporation.

**Involve Users:** Early involvement of the users in the definition of the functions and data needs of the business (as-is and to-be) and in the future allocation of business functions in the to-be system is mandatory for the success and acceptance of an integration project. Integration must be viewed as a means to leverage existing business advantages of the corporation while lessening known shortcomings. Through integration, the organization is presented with the opportunity to reduce cost and cycle time, to improve quality, and to identify new products and market opportunities.

**Users Are Jointly Responsible For Success:** The responsibilities for the success of any integration project rest squarely with the user organizations acting as the primary system through the integration of their business functions. The integration of the information systems, supporting the newly redefined business functions into a cohesive whole, then becomes an implementation step of secondary concern.

**Roles of the Information Systems Organization:** The Information Systems organization should not be viewed as the primary beneficiary and sole proponent of integration. Instead, it should be viewed as a co-consultant, focused on solving a business rather than information system problem. The primary role of the Information Systems organization is:

* to facilitate the system analysis (as-is, to-be) of the user organization operations.
* to ensure that the newly integrated system complies with the corporate information and computer architectures.
* to define and evolve the information and computer architecture capable of supporting the overall business goals of the corporation.

**Top Management Support:** Top management support is a key contributor to the success of integration. This support is crucial in undertaking projects that transcend traditional organization boundaries, and that exceed, in scope and duration, typical Information Systems project norms. Top management support is also required to keep systems compliant with the corporate information and computer architectures as
systems evolve in response to the changing business and technical environments.

**Gaining Top Management Support:** Support from top management can only be expected when top management is convinced that integration serves the strategic (primary concern of top management) rather than the tactical goals of any one organization. In particular, integration must not appear to be a goal only of the information systems portion of the organization. Top management must understand that integration yields competitive advantages that competition cannot duplicate. This is as opposed to the ease with which stand-alone products can be acquired and deployed. This occurs because integration leverages the existing competitive position (market information, technical skills, installed assets) of the organization. Top management commitment is not simply measured by the resources (schedule, manpower) allocated to integration. It is mainly expressed by the willingness to make the organizational trade-offs promoting and maintaining inter-operable corporate systems. Top management must look at integration of the business functions as a significant corporate asset.

**Project Life Cycles**

The project life cycle is important in setting proper management directions. A prime issue related to the successful definition and implementation of integration in a corporation is the widely shared consensus that integration activities are difficult to justify to management. Management is viewed as being accustomed to evaluating the merits of a project through traditional financial measurements. Unlike hard automation projects, for which management has intuition, integration demands the commitment of resources to activities (e.g., functional and data modeling) that have no appeal to management. Further, benefits of these activities are difficult to foresee at the beginning of the investment period. Essential elements of a successful integration project are detailed below:

**Define An Open Computing and Information Architecture:** Integration efforts must begin with the definition of a computing environment (hardware, network, system software, system standards), and an information definition methodology (functional, data modeling methodologies, and integration techniques). It must also have the means of controlling the transition of existing applications and systems into an integrated
computing environment.

The architecture must be closely aligned with the major industry standardization efforts. This is necessary to allow for the orderly growth of the computing environment (new applications, new technologies, new standards), and to support unforeseen demands for inter-operability, both from within a corporation and from the marketplace (anticipation of more requirements like CALS in the defense and non-defense markets).

**Identify and Tackle Visible Problem Areas That Will Benefit From Integration:** Integration can be implemented in a bottom-up fashion in visible areas of the business where cost/benefits, quality improvements, cost, and cycle time reduction can be realized and demonstrated to management. This bottom-up approach can be utilized as long as a well-thought-out information and computer architecture is available and is used to provide technical directions (e.g., standards, methods, tools) during implementation activities.

**Develop Metrics to Assess the Benefits of Integration:** Given the difficulty in quantifying benefits in advance, the anticipated benefits require that before integration, productivity metrics be acquired to evaluate the success of the integration project throughout its life cycle.

Likewise, it is important to establish reasonable expectations among the user organizations. These expectations must be documented in memoranda of understanding reflecting the commitments of the user organizations to the integration effort.

**Involve Users Early in the Project:** This is the key to a generation of truly useful ideas yielding significant business benefits. Justification of integration through savings in data processing costs is unlikely to be significant. This is because data processing costs amount to only a few percent of the design and manufacturing cost of a high technology product. The mechanisms recommended to obtain early user involvement are:

* tackling a pressing user business need through integration
* joint development of the as-is and to-be functional and data models of the user business.

**The Integration Problem**

The integration problem is defined across the following categories:

* differences in the nature of business and technical data
* differences in business and technical systems
* differences of maturity in business and technical methodologies
* organizational and cultural issues.

**Differences in the Nature of Business and Technical Data**

To assess the problem of integrating business and technical data, consider the difference in the basic nature of the two data domains. These potential differences are important because:

* Integration of the two domains will require data management facilities that can accommodate the data from both domains where necessary.
* Since virtually all commercial DBMSs were developed to support business applications and data, it follows that fundamental differences between technical and business data would render the current offering of DBMS tools useless for technical data, producing a serious impediment to integration of the two data domains.

In searching for fundamental differences, a list of fundamental data characteristics was assessed:

* Standard data types
* Extended data types
* Relationships and relationship types
* Schema characteristics
* Data volume
* Data integrity
* Data distribution
* Data semantics
* Data versioning and configuration management
* Paradigms of interoperability, interchangeability, and behavior.

The fundamental differences between technical and business data seem only to be quantitative in nature. A few of the important quantitative differences are discussed below.

* **Standard Data Types.** The "exotic" data types encountered in technical data are also arising in office automation concepts as office integration proceeds.
* **Extended Data Types.** The requirement for per-application data types, which originates in object-oriented programming concepts, is seen as defeating integration, which focuses on persistent objects coordinated under a global schema. That is, data types that cannot be recorded coordinated, and communicated cannot be shared.

* **Relationships and Relationship Types.** Both business and technical data require a fundamental capability in expressing relationships and relationship types. Technical data, however, tends to have a richer set of relationship types, and extremely complex relationships among data objects.

* **Schema Characteristics.** Technical data (particularly in a business such as electronics manufacturing) will have a richer vocabulary of objects and relationships than business data. Additionally, technical schemas will tend to change more frequently, tracking new technology and standards; whereas business schemas are more stable. However, as mentioned, this is a matter of degree.

* **Data Volume.** Both business and technical data involve large amounts of data. Technical data items tend to aggregate other data items into much larger objects than is customary for business data. In these early days of engineering information models, entire design "databases" are often tracked and managed as a single data item measuring from megabytes to terabytes in magnitude.

* **Data Versioning and Configuration Management.** Versioning is only modestly used in business data, but is an extremely important feature in technical data. Versioning is not well supported in current DBMSs.

• **Paradigms of Interoperability and Interchangeability and Behavior.** These paradigms illicit a strong feeling that technical and business data are fundamentally different. This is due to their requirement for extremely frequent application in the technical domain, and its occasional use in the business domain. In business, a purchase order is predictably relatable to a relatively small number of entity-types. The technical community (electronics design in particular) requires applying the paradigms heavily. An electrical part is relatable to a large number of entity-types, i.e., any types of electrical part. An electrical part, furthermore, also serves a role as a mechanical part that must be fit together with other physical (mechanical) entity-types. Once a design has been made (a relationship of electronic and mechanical objects), its behavior must be simulated. Therefore, the occurrence of a
part-type in a design must also contain its behavior. This type of complexity is the forte of object-oriented approaches seen increasingly and frequently in CAD tools today.

6.3 A FRAMEWORK FOR INFORMATION INTEGRATION

Manufacturing systems are becoming increasingly complex and often involve a wide range of functions that are multidisciplinary in nature. Integration efforts for these systems are often carried out in an ad-hoc fashion. This has been primarily due to lack of formalization and structured frameworks. Often there is a confusion between frameworks for integration of information and frameworks for process control. It is felt that the two should be kept separate. While a hierarchical structure seems to be suitable for control, this structure is not suitable for information flow.

This chapter discusses the initial framework used for the case studies, and a generic integration model (GIM). The experiences from these and the case studies were used to propose a next generation framework for information integration. The GIM formalizes the information-integration process and provides definitions and axioms to help achieve it. The framework builds upon the GIM to provide a structure through which integration may be achieved.

6.3.1 INITIAL FRAMEWORK FOR INFORMATION-INTEGRATION

Figure 6-1 shows an initial framework. The case studies discussed earlier utilize this framework. The basic idea behind this framework is to provide a shared information base for all the applications. This information base acts not only as a repository, but also as a communication medium for the applications. The following sections describe the system requirements that such a framework has to fulfill, and the components required to provide the necessary functionality.

6.3.1.1 SYSTEM REQUIREMENTS

This section describes the characteristics important to an integrated system, a generic software framework that provides those characteristics, and the system development methodology to realize the software framework. The most important factors considered in developing the integrated system configuration are the following:

Flexibility. Since material handling is a very diverse function, involving various
equipment from design through operation, there are a number of software packages designed to do specific tasks. The envisaged system should be flexible enough to accommodate the diversity in the software packages.

**Modularity.** The system should be modular in nature, to accomplish the addition and/or deletion of software packages with minimum effort.

**Consistency.** Various applications within the system should be consistent with one another. This will help the users to learn the system faster and to use it more efficiently. One way to accomplish this is to have the same front end interface for all the programs.

**Reusability.** Parts of or the whole of the system should be reusable for different applications. This becomes important due to shorter product life cycles.

**Maintainability.** The system should be easy to maintain. Since these systems and their requirements are continuously changing, their maintenance becomes an important issue.

**Multitasking.** The system should support multiple users as well as allow multiple tasks to be carried out simultaneously.

### 6.3.1.2 FRAMEWORK COMPONENTS

A conceptual sketch of the initial integrated system framework is given in figure 7-1. The user interacts with an executive, which controls the application programs within the system. This will ensure all the application programs to be highly consistent with each other by having the same user interface. Flexibility and modularity are achieved by interconnecting the applications loosely using a database manager. Note that the information framework can encompass tasks involved in the design of materials handling systems as well as tasks involved in the operation of material handling systems. While the specifics of these two activities may be different, the integration frameworks are quite similar. A shared information system framework should provide:

**A Shared Data Base:** to eliminate redundancies and inconsistencies. It must be noted that the shared data base implies a conceptual global schema. The data base can physically be located in different locations.

**A Data Base Management System:** to store, arrange, and retrieve data from the shared data base. The DBMS should provide a mechanism for data independence and integrity checks. Object-Oriented DBMSs may be incorporated as that technology becomes more
mature. Each physical database may have its own DBMS, providing similar capabilities at all locations. For the users and applications however the DBMS and database are single entities.

**Interfacing Programs:** to act as pre- or post-processors between the application programs and the shared data base. These processors are easier to develop and maintain if the applications conform the global schema. However the framework makes no such assumption.

**A Data Dictionary:** to describe the objects and their functions in the system. The data dictionary contains the metadata as well as the schema of the database.

**An Executive:** to manage the integrated material handling system. Besides providing a good user interface, an executive should manage multiple users and multiple tasks.

**A Geometric Modeler:** to provide geometric representation facilities. Geometric representation forms an integral part of most engineering applications and hence deserves special treatment.

**A Knowledge Based System:** to capture and use knowledge in a structured manner. The knowledge base would interact heavily with the data base and use the information stored in the DBMS to draw inferences.

### 6.3.1.3 FRAMEWORK EVALUATION

The integration infrastructure was evaluated by applying it to the integration of several Material Handling tasks. Three integration scenarios were used to test the concepts, one associated with the design of a Material Handling system, the second concerning integration of two knowledge sources, and the third associated with the manufacture of optical fiber products. The results of the evaluation with respect to the requirements are shown in figure 6-2. The framework is flexible as it can incorporate various functions as shown via case studies. The functions include software, expert systems, plant operators, manufacturing functions, etc. The framework is modular in that addition of new functionality does not affect the system. For example, addition of a new product manufacturing function had minimal impact on the rest of the system in case study 3. The framework provides a consistent user interface for a variety of applications as demonstrated by the three case studies. Reusability of components is demonstrated by using significant part of the system when a new product was introduced in the case study
3. Addition of foundation design function involved simply adding new rules to the rule base in case study 2. Large part of process model, and data model were reused, and so were the components like user interface, and the DBMS. The framework shows improved maintainability, as demonstrated by the little changes that had to be made to the design in the case of requirement changes and introduction of new products in case study 3. The framework was consistent with the requirement of multitasking as demonstrated by case study 3 where several users use the system concurrently for a variety of tasks. The capabilities required by the framework were demonstrated in the case studies where user interfaces, DBMS, applications, and a Knowledge based system coexist.

The framework seems to provide the desired functionality to a certain extent, and the prototypes work, but soon we start discovering a set of problems: how to manage the constraints on the shared information, how to manage the exceptions being generated by the applications, what if the DBMS does not support multiple views on the information, how to manage differences in schemas supported by different applications and the shared database, how to manage different types of events being generated by the system in a consistent manner. These problems suggested certain updates to the current framework and are discussed in section 6.5. A constraint manager implementation is described in case study 2 where a rule based system is used to enforce all the constraints on the information.

6.4 AN INFORMATION-ORIENTED GENERIC INTEGRATION MODEL (GIM)

There is a need for a methodology to study the interactions between different data sets in complex environments and for characterizing the nature of the data, the constraints that propagate between different tasks, and the exceptions it encounters. A generic model is proposed here which is the basis for the methodology employed to identify and classify these common data sets, constraint functions, and exceptions. Such an approach is imperative for defining data entities that play an important role in the design of databases and in identifying data dependencies that facilitate consistency checking and inferencing. The nature of the ideal system inherently defines three axioms that are typical of any integration effort. These axioms provide some guidance in defining the methodology for efficient data integration.

Conditions necessary for data integration, constraint implementation, and exceptions
handling within the framework of this generic model have been identified. Such a basis is essential if any level of integration is to be achieved between the different serial and parallel functions [CIM3]. Characteristics of a real system are contrasted to that of an ideal system under this framework. The purpose of any model is to provide an analytical basis to characterize reality. To illustrate the usefulness of this model, three integration efforts were carried out using the methodology outlined by the model. The primary focus in these case studies involves the integration of interrelated data entities that are used in different tasks. An earlier attempt for the integration model is discussed in [CIM3]. The model presented here builds on the earlier model.

6.4.1 Assumptions
1). Engineering functions (software, algorithms, decision making agents, etc.) can be represented as operators on data.
2). Pertinent data can be captured using protocol analysis.
3). Accurate and timely information availability will significantly improve the productivity of a system.
4). Any system will be subjected to exceptional events/conditions.

Definition 1: Information-Integration. A system has achieved information-integration if it can provide correct, secure, and manageable information at the right time and in the right format in response to a request.

6.4.2 Model Description
The data interactions that exist with regard to any particular task, ‘a’, may be modeled by considering its relations to all other tasks with which ‘a’ interacts. In the model presented herein these interacting tasks are: supplier tasks, represented by $s_i$, and customer tasks represented by $c_i$. Realizing that a particular task may satisfy the role of both a supplier as well as a customer, it becomes obvious that the nature of these interactions is extremely complex. Figure 6-3 characterizes this situation. Thus for any function, “a”, $s_i$ may be regarded as a corresponding upstream function, and $c_i$ may be regarded as the corresponding downstream function. The function “a” is represented by the operator $F_a$ that essentially transforms supplier data into customer data. On a broad
scale $F_a$ is a combination of analytical tools, CAD programs, Machine tools, qualitative decision making, and intelligence. In order to ensure that the operator's ($F_a$) operation is consistent with both the upstream and the downstream functions, it is necessary to impose a constraint function $Y_a$ that incorporates the interdependencies that exist between the upstream functions, $s_i$, the current function, “a”, and the downstream functions, $c_j$. In a real life situation things sometimes don't work as planned, i.e., normal flow of activities can be disrupted by exceptions. Exceptions are events that are either unusual, unplanned, can disrupt normal process, or happen infrequently. Exceptions are conditions that require special attention, and are not necessarily disruptive. It is our experience that this important aspect of systems is often neglected and hence there is a lack of formal methods of exception management. Thus there is a need to explicitly model exceptions for the system activities, i.e., an operator $F_a$ can raise a set of exceptions $g_a$. A simple model describing these requirements is shown in figure 6-4. The data inputs and outputs for operator $F_a$ are $D_i$ and $D_j$.

A realization of such an elegant representation involves numerous prerequisites. Figure 6-5 depicts ideal system data characteristics during the design-production life cycle. Graph 1 of figure 6-5 shows that the available data must always be accessible for maximum benefit. This essentially requires that the designers or engineers must be in a position to track any information required for their task without any retrieval time delay. Further, graph 2 of figure 6-5 implies that as the constraint implementation increases along with the design cycle, there is no time wasted due to redesign- which might otherwise add additional design freedom and disrupt the design cycle. In contrast to this representation, Figure 6-6 depicts the characteristics of real systems which typically deviate from the ideal system and exhibit data and constraint oriented deficiencies. Graph 1 of figure 6-6 demonstrates the fall in productivity resulting from a gap between the available data and the accessible data. Thus in order to obtain the available data that is not in an easily accessible form, a severe overhead is expended within the operator, $F_a$. This generally translates into a delay by stretching the design/production cycle time. The interdependencies that exist between the different upstream and downstream functions as represented by $Y_a$, if not fully imposed will result in local peaks. This implies an enhanced freedom (less constraint satisfaction) due to changes that need to be incorporated resulting in unproductive design activity. Thus graph 1 of figure 6-6
measures the level of data integration available in any organization whereas graph 2 of figure 6-6 measures the level of integration between functions and the level of constraint implementation available. By ensuring that both these requirements exist between the serial and parallel tasks (defined in [CIM3]), a "Concurrent engineering" paradigm results. For the sake of consistency, three terms are defined within the framework:

(1) Data integration.
(2) Constraint function integration.
(3) Exceptional event integration.

6.4.2.1 Data Integration

Consider the system shown in Figure 6-3. Define data sets associated with each function (task), i.e., let “A” be the data set associated with function “a”, S\textsubscript{i} be the data set associated with function s\textsubscript{i}, and C\textsubscript{i} be the data set associated with function c\textsubscript{i}. The common data set associated with function “a” can be defined as

\[
D_{i} = (\bigcap x \hat{1} A) (\bigcap P_{i}(x) \bigcup s_{i}(x))
\]

(6.1)

where \(\hat{1}\) is the union operator, \(\cap\) is the intersection operator, and

\[
P_{i}(x) = \bigcap_{i=1}^{n} P_{i}(x),
\]

and

\[
P_{i}(x) = x \hat{1} (A \cap C_{i})
\]

\[
s_{i}(x) = \bigcup_{i=1}^{m} s_{i}(x),
\]

and

\[
s_{i}(x) = x \hat{1} (A \cap S_{i})
\]

and \((A \cap X)\Rightarrow (y \hat{1} X)\) for any data set X, i.e., a data element in “A” is functionally related to a data element in X by an arbitrary function \(\hat{1}\). n is the number of downstream functions for “a”, and \(m\) is the number of upstream functions for “a”.

Thus if "D" is the universal design data set then, \((A \cap C_{i})\) is the extension of \(P_{i}\) in
D, and \(( A \subseteq S_i )\) is the extension of \(s_i\) in \(D\). Hence for complete data integration of the system -

\[
D = \bigcup_{i=1}^{l} D_i = D_1 \cup D_2 \cup \ldots \cup D_l
\]

This definition by virtue of the union operator will intrinsically eliminate redundancy in the common data set. Thus \(D\) provides the necessary and sufficient condition for the contents of the common database.

### 6.4.2.2 Constraint Function Integration

Consider the data set associated with functions “a”, \(c_i\), and \(s_i\) as shown in figure 6-4. We can define a constraint function \(Y_i\), which relates the data sets in different functions and provides a basis for defining the interdependencies between tasks. \(Y_a\) is the set of all the constraint functions \(Y_i\) for the function “a”. This is a prerequisite for incorporating any level of concurrence into the design/production life cycle, and may be viewed as a generic DFX (Design for Manufacture /Assembly /Testing/ etc.) prerequisite. Thus

\[
( \$ x \hat{\in} A ) ( \$ y \hat{\in} C_i ) ( \$ z \hat{\in} S_i ) (Y_i(x, y, z))
\]

\[
Y_a = \{ Y_i \mid i = 1, 2, \ldots \}
\]

Thus by incorporating all the constraint functions \((Y_a)\) within the system, the need for changes and design feedback is reduced. However, under certain circumstances, design iterations may be impossible to avoid. This is similar to the nature of non-linear systems which are inherently iterative. We may thus classify constraints as linear and non-linear constraints. A linear constraint may be implemented without any feedback or iterations, whereas a non-linear constraint can only be implemented indirectly and would involve iterations and feedback. Although figure 6-4 represents an ideal system it may not be realizable if the constraints are non-linear. Under such situations, there may be a need to linearize these constraints in order to reduce design feedback.

The efficient implementation of the constraint function is dictated to a large extent by
the level of data integration that pre-exists this effort. In many cases it becomes difficult to even conceptualize the nature of Y unless the relevant tasks have been integrated from a data perspective.

6.4.2.3 Exceptional Event Integration

As mentioned earlier exception management is very important to avoid lengthy delays and shutdowns in case of an exception or breakdown. If an exception occurs, the function “a” will raise the appropriate exception \( g_i \). Let \( g_a \) be the set of all the exceptions the function “a” can raise.

\[
g_a = \{ g_i \mid i = 1, 2, \ldots \}
\]

An exception will influence the function “a” that raised it, as well as other functions as defined in the system. The behavior of exceptions and how they are handled is discussed in a subsequent chapter.

6.4.3 Perturbations in \( F \) and \( Y \)

Any system oriented model should facilitate dynamic adjustments. Thus a design/manufacturing organization modeled using the methodology presented herein should be capable of accommodating changes resulting from the induction of new technologies and functionalities. These changes typically result from changes in the operator \( F \) or a change in the constraint function \( Y \).

Changes in \( F \) are indicative of a change in the associated data sets, \( D_i \). This may result from the introduction of new tools (software or hardware), new technologies, or different levels of expertise. To study how such changes affect the model, we again consider the generic model. Figure 6-7 shows how data is transformed according to the GIM, in this figure:

- \( F^* \) is the processor that transforms the data values: \( d_{ai} \rightarrow d_{ao} \)
- \( F_i \) is the pre-processor that transforms the data format: \( d_{ni} \rightarrow d_{ai} \)
- \( F_o \) is the post-processor that transforms the data format: \( d_{ao} \rightarrow d_{no} \)
Thus correlating this form to equation 6.1 -

\[ F_A = F_i F^* F_o \]

and

\[ \frac{d n_i}{d o} = D_i \]

For facilitating this transition, it is necessary to standardize the operators \( F_i , F_o \). It is obvious that these operators will be easy to develop if the input and output data formats match, i.e., applications conform to the database schema. This is consistent with the statement made in section 6.1.2. Several database management systems provide a flexible data manipulation and data definition language that satisfies this requirement (e.g., ORACLE’s SQL, INGRES’s QUEL, and VBASE’s TDL).

### 6.4.4 Modularity and Flexibility

Changes in \( F^* \) may be prompted by changes in software or hardware tools, changes in technology, or by general changes in operation and methodology. In order to absorb such changes within the affected function, \( A \), without affecting other functions in the system, the pre and post processors may be changed. Thus, changes in \( F^* \) to \( F^{*'} \) can be neutralized by changing \( F_i , F_o \) to \( F'_i , F'_o \) such that

\[ F_A = F_i F^* F_o = F'_i F'^* F'_o = F_A \]

Similarly, \( \frac{d n_i}{d o} \) may change to \( \frac{d n'_i}{d o'} \) due to changes in technology (e.g., Relational model \( \rightarrow \) Object data model). These changes may also be absorbed by \( F_i \) and \( F_o \) to keep \( F^* \) unchanged.

Equation 2 does not impose any restrictions on the implementation of the constraint function, \( Y \). The constraint function, \( Y \), may therefore be incorporated entirely within \( F^* \) or distributed between \( F_i , F_o , \) and \( F^* \).

Typically two classes of constraints exist
1) Allowable values: $x \in A$.

2) Function verification, i.e., $Y(x,y,z)$ is true.

Type 1 is generally used for serial functions (design, manufacturing, etc.), while type 2 is generally used for parallel functions (multidisciplinary design). It is usually possible to implement type 1 constraints within $F_i$ and $F_o$, whereas $F^*$ is more amenable to constraints of type 2. These constraint usually exist in natural hierarchies, which are explored in subsequent chapters.

6.4.5 Axioms for Information Integration

In the previous sections a generic integration model has been developed. The development of this model implies certain a priori principles that serve as necessary and sufficient conditions for effective system integration in the design/manufacturing areas.

There is a need to generalize the observations in explicit terms so as to eliminate the ad hoc nature of the integration process. Axioms enable us to generalize the observed behavior in terms of explicit statements, and should contribute to the development of the field [6]. In light of the generic model certain generalizations can be made and are presented herein in the form of axioms that must be satisfied in any integration effort:

* **AXIOM 1** 
  A flexible, modular integrated system requires complete data integration, complete constraint integration, and complete exception integration.

* **AXIOM 2** 
  Complete data integration is a prerequisite for complete constraint integration.

* **AXIOM 3** 
  Complete constraint integration is a prerequisite for complete exception integration.

Axiom 1 states that in order for a system to be integrated, all data flows must be understood and integrated as per equation 6.1, all the constraints should be imposed on the data as per equation 6.2, and all the exceptions should be handled by the system. Thus, an intelligent, information driven inferencing mechanism must allow for data, constraint,
and exception integration. Reverse may not be true. This axiom provides the necessary components to achieve information-integration as per Definition 1. Axiom 2 states that complete constraint implementation is not possible if complete data integration has not been achieved. In the absence of data integration it is difficult to conceptualize the constraints (or even be aware of the existence of some constraints). Axiom 3 states that complete exception integration is not achievable unless complete constraint integration has been achieved. It must be noted that complete exception integration can never be achieved according to the assumption 4 of the model, since there will always be events that the system will fail to handle. The objective thus is to be able to handle as many exceptions as possible, and continue learning from new exceptions.

6.4.6 Evaluations and Observations

The GIM was evaluated with respect to the requirements using the case studies as shown in figure 6-8. The experiences from the case studies confirm that the assumptions for the model are realistic. The functions (software, algorithms, humans, decision making agents) encountered can be viewed as operators on data sets. The data sets have to conform to constraints and the functions are subjected to unplanned or infrequent events. The case studies confirm that common data can be exchanged by functions via a shared database. The case studies also validate the methodology for designing and implementing the shared information base. As pointed out in the third case study, the database design remained unaltered, during the application design and redesign, i.e., as the application changed, the required information was made available by changes in the pre- and post-processing software. The case studies demonstrated the need for constraint implementation and exception handling along with data integration to provide better information integration. As described in case study 3, constraint checking was implemented using the shared database, and it is only after implementing process and data constraints, the issue of exceptions was addressed. This validates the three axioms proposed in this chapter.

6.5 PROPOSED FRAMEWORK FOR INFORMATION INTEGRATION

Based on the experiences in case studies and the GIM, the original framework was
found deficient in certain respects. In this section an improved framework is presented, that overcomes some of the problems. The requirements were assumed to be the same as those of the initial framework.

6.5.1 FRAMEWORK COMPONENTS

A conceptual sketch of the proposed integrated system configuration is given in figure 6-9. The user interacts with an executive, which controls the application programs within the system. This will ensure all the application programs to be highly consistent with each other by having the same user interface. Flexibility and modularity are achieved by interconnecting the applications loosely using a database manager. This framework retains the components of the earlier framework which are discussed in section 6.1.2, in addition the framework should provide a set of managers for the interacting applications to use as tools. The utilities that these managers provide could be part of the DBMS. In case the DBMS doesn't provide these utilities, these have to be provided externally. We envision future information management products providing all these as standard utilities. The descriptions provided here are conceptual in nature and the comments on implementations of these managers have been deliberately avoided to ensure that the content is implementation independent. Various managers identified are:

A Constraint Manager: to keep the information in a consistent state. It is the responsibility of the constraint manager to enforce right constraints on the right data at the appropriate time. Constraint violations are referred to the exception manager for handling.

An Exception Manager: is responsible for managing all the exceptions in the system that could not be handled by their originator. The exception manager provides tools and an uniform way of handling these exceptions. An exception may result in relaxation of some of the constraints. The exception manager notifies the constraint manager in such cases.

A Change Manager: is responsible for managing changes in the shared information base. It is the responsibility of the change manager to notify applications that may be affected by a change [CE23].

A View Manager: provides different views on the shared information as per the needs of an application. Relational DBMSs currently provide these utilities. The ability to provide different views on the common information is very crucial for information-
An Object Manager: manages the objects in the system. An object manager is a standard part of an OODBMS, but systems with non OODBMSs might need to provide these utilities to interact with applications that are based on the O-O paradigm.

An Event Manager: manages the events generated by the users, the applications, and the managers. Any action taken by the system is usually in response to an event. Events may be normal or exceptional. Exceptional events are forwarded to the exception manager for proper handling. Other events are handled by their target applications or managers.

A Schema Manager: manages the schema using the data dictionary. It is the responsibility of the schema manager to maintain the global schema and the local schemas of different applications, and provide utilities to translate between global and local schemas. The schema manager also makes sure that various schemas are consistent with each other.