"TECHNOLOGIES AND POLICIES FOR THE DEVELOPMENT OF COMPOSITE INFORMATION SYSTEMS IN DECENTRALIZED ORGANIZATIONS"

by

BERTRAND RIGALDIES

Ingénieur de l'Ecole Centrale des Arts et Manufactures

(1988)

Submitted to the Department of Civil Engineering
in partial fulfillment of the requirements for
the Degree of

MASTER OF SCIENCE IN TECHNOLOGY AND POLICY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1990

Signature of Author _____
Department of Civil Engineering
Technology and Policy Program, May 12, 1990

Certified By _____
Professor Stuart E. Madnick
Thesis Supervisor

Accepted By _____
Professor Richard de Neufville, Chairman
Technology and Policy Program

Accepted By _____
Professor Ole S. Madsen, Chairman
Departmental Committee on Graduate Students
Department of Civil Engineering

ACKNOWLEDGEMENTS

I would like to express my gratitude to Dr. Stuart E. Madnick for his guidance throughout the work on this thesis. Working with him has been a unique experience. I would also like to thank Dr. Michael Siegel, for his assistance and helpful comments. Finally, I would like to thank the companies Citibank and Salomon Brothers, Inc. for their cooperations in this project.

This thesis is dedicated to everyone who has helped me to make it through the past two years, especially:

My family, for their love and support which made all of this possible;

Susie Doyle, for listening and cheering me on when I needed it most, and for reading the entire document;

Abyd, Alan, Bob, Brian, Guillaume, Mark, Robert, Steven, Toon King and all the good friends I met at MIT, for their moral support throughout this endeavour;

Bonnie Scurlock, for helping me during the software development.

This research was conducted at the Composite Information Systems Laboratory (CISL), Sloan School of Management, at MIT, and supported in part by the international Financial Services Research Center, Reuters, and AT&T.

TECHNOLOGIES AND POLICIES
FOR THE DEVELOPMENT OF COMPOSITE INFORMATION SYSTEMS
IN DECENTRALIZED ORGANIZATIONS

by

BERTRAND RIGALDIES

Submitted to the Department of Civil Engineering
in partial fulfillment of the requirements for
the Degree of Master of Science in Technology and Policy

ABSTRACT

This thesis describes the combination of a policy mechanism with a new technology as an implementation strategy to achieve information systems connectivity in an organization having a decentralized management structure. Information systems connectivity has emerged as being an important vehicle for achieving better coordination in large decentralized organizations. Better coordination has become a key requirement for the survival of the organization in an increasingly competitive and volatile environment. However, because the sub-units of a decentralized organization have traditionally been fairly autonomous, they have deployed information technology without a common corporate policy. This uncoordinated IT deployment has created in many decentralized organizations a heterogeneous information systems base.

This thesis explores various connectivity approaches that have been documented in the literature and tested by companies in the past. The thesis focus on one of these approaches known as *Focused Standards*, and provides recommendations for its implementation. The approach is based on the deployment of *Composite Information Systems* (CIS) coupled with the standardization of the definition of key corporate data elements. The recommendations address the questions of how to select the data elements whose definitions will be standardized across the sub-units, and how to enforce the standards. The selection process is based on a committee-based approach which gathers representatives from the main user groups. The enforcement process is based on a policy which gives a high degree of flexibility to the sub-units. This enforcement mechanism relies on the technology known as *semantics mapping*, and which allows organizational units to gradually shift their current internal data definitions to the corporate standards. The thesis also presents the design of such a semantics mapping tool.

The main conclusion of the thesis is that a decentralized organization should not attempt to adopt a complete company-wide standardization of its hardware, software, and data elements definitions, in order to gain connectivity. An organization should gradually move its IS base from a heterogeneous environment to a more standardized one in a self-interested manner, and based on the deployment of CIS systems and semantics mapping tools.

KEYWORDS
Information Systems Connectivity in a Decentralized Organization, Composite Information Systems, Focused Standards, IS Policies, Data Semantics Reconciliation.

Thesis Supervisor: Dr. Stuart E. Madnick
Title: John Norris Maguire Professor of Information Technology and Leaders for Manufacturing Professor of Management Science.

TABLE OF CONTENTS

List of Figures

-oOo-

Chapter I.

OVERVIEW

1.1 THESIS TOPIC

Increasing global competition and growing market expectations are confronting companies with accelerating change, change which is reshaping the way companies operate and compete. The role of information systems in meeting these business challenges is also changing from a service function to a true competitive weapon. Conventional computing strategies, however, cannot provide the capabilities needed for this transition. This thesis proposes the combination of a policy mechanism with a new technology as an implementation strategy to close the gap existing between what business needs and what conventional systems can offer.

We will study the implementation of this strategy in the framework of a decentralized organization. The goal of the strategy is to establish connectivity among the multitude of information systems that are typically scattered through a decentralized organization. The strategy is two-fold: (1) develop a policy mechanism in order to promote data representation standards in a decentralized organization; and (1) develop a new technology, called *semantics mapping*, which aims at facilitating the exchange of data among disparate information systems.

The potential audience of this thesis includes any large decentralized organization which is undertaking a major effort to improve the information-sharing and communications

capabilities of its IS infrastructure. Such an organization is aware of the tremendous strategic value of information systems connectivity, hence is more than willing to learn about the existing connectivity options and their pitfalls.

Figure 1.1 on the next page shows each step of the reasoning that will be followed in the thesis.

1.2 ORGANIZATIONAL FORCES

The case of a decentralized organization is particularly interesting since it is exposed to three potentially conflicting forces: autonomy, integration, and evolution.

The first force is inherent to the rationale behind a decentralized structure. Various organizational units are formed and equipped with a high degree of autonomy. Indeed, a sub-unit needs to be tailored to the local business needs in order to be responsive and innovative in regard to these needs.

The second force stems from increasing competition which requires organizations to continually improve their efficiency and effectiveness in order to sustain their competitive edges. For a decentralized organization, it means that better coordination and integration must be established among its various organizational sub-units. In addition to pushing toward more integration, the increasing competition also pushes the organizational units toward even more autonomy so that their responsiveness is increased.

The third force is inherent to the volatile nature of the organizational environment, which requires its structure to deal with continuous evolution. Large decentralized organizations are moving, for example, toward flatter structures linking a collection of smaller, highly independent and specialized units; in addition, organizations are exposed to the omnipresent risk of being merged with or acquired by another company.

Thus, as decentralized organizations increase their coordination and integration, an acceptable level of autonomy still has to be maintained, and the fragile balance which is established between the two forces has to be constantly adjusted.

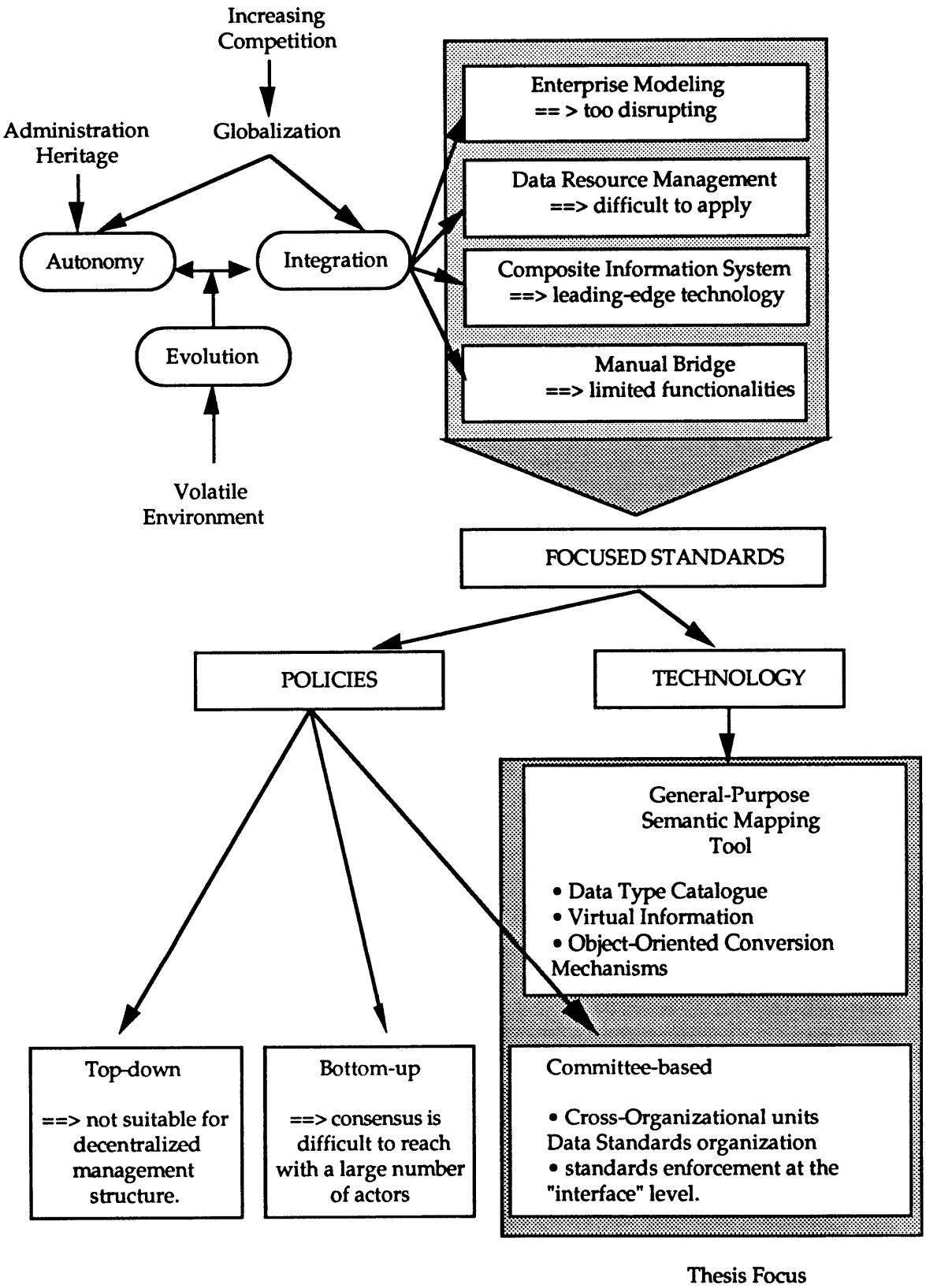

Figure 1.1: Thesis' contents and logic

Information Technology (IT) is playing a key role in decentralized organizations' quest for greater coordination. Due to dramatic improvements of distributed systems and communications networks technologies, both in terms of costs and performances, a firm can build a company-wide Information System (IS) infrastructure which links data bases, applications, knowledge bases, and people. Strategic applications, such as global risk management, global planning and control, and a variety of executive support systems requiring the integration of data existing in disparate and geographically-dispersed information systems, now become technically feasible. The building of such a company-wide information utility has been referred to as strategic connectivity in both the academic and business communities.[1]

However, as organizations begin to look at their collection of information systems on a more global point of view, they are discovering a very heterogeneous environment, especially in the case of decentralized organizations where sub-units have had their own policies to deploy information technology. The advent of end-user computing and computer proliferation due to cheaper hardware and software created a set of disparate, often incompatible, information systems distributed throughout the organization. The problem of decentralized IT deployment has led to discrepancies in data definition, accuracy differences, and timing and coordination problems. This resulting lack of accessible, quality data seriously hinders data integration.

1.3 CONNECTIVITY APPROACHES

Two types of methodologies to gain connectivity among heterogeneous information systems have been tested by organizations in the past: "proactive", and "reactive". Two different approaches will be considered within the first category: Enterprise Modelling, and Data Resource Management; and two other approaches will be considered within the second category: Manual Bridge, and Composite Information Systems.

The proactive stance advocates the planning of connectivity requirements ahead of time. Enterprise Modelling consists of redesigning and rebuilding the entire organization's IS infrastructure. Data Resource Management (DRM) consists of defining the entire set of corporate data elements, standardizing their definitions across all sub-units, and actively promoting adherence to the standards.

---

[1] All references used in this chapter are given in later chapters.

The reactive stance, on the other hand, does not plan for any connectivity requirements ahead of time, but advocates the building of linking devices (mainly software-based) on the top of the existing IS infrastructure. As opposed to the first stance that can be considered revolutionary, the reactive stance is more evolutionary. Manual Bridge constitutes an extreme version of this "reactive" paradigm, whereby connectivity among information systems is built like a "patchwork" where the different pieces (i.e., computers) are connected by bridges that are built by different organizational units at different times, and for different purposes. Composite Information System (CIS), in turn, is characterized by being "proactively reactive." CIS is a reactive approach since it advocates the building of applications on top of the existing IS architecture and given the current data element definitions, but also has the peculiarity of providing a systematic approach to connectivity that straightens up the "patchwork" and make it easier to build, maintain, and extend in the future.

A study of these methodologies will lead us to the following conclusions:

• Due to the magnitude of its endeavour, Enterprise modelling has never been able to achieve its final goal, that is, the implementation of a new IS structure.

• Because a company-wide standardization effort is arduous, DRM is also particularly difficult to apply in practice.

• The "patchwork" that Manual Bridge builds is appropriate for ad hoc integration needs but is not suitable to handle large volumes of transaction.

• CIS is promising, but has still to solve some non-trivial technical difficulties.

A new approach, called *Focused Standards*, has recently emerged and is combining some aspects of Data Resource Management and Composite Information System. The approach consists of defining critical corporate data elements that will be standardized throughout the organization in order to gradually shift the existing IS infrastructure toward a more standardized environment. Since the methodology starts from the existing *status quo*, the approach has the advantage of being evolutionary like the CIS methodology; and since it is aimed at gradually changing the heterogeneous environment into a more standardized one, the approach also has the potential of reaping the benefits of company-wide standards, which makes DRM attractive.

1.4 THESIS QUESTION

This thesis will focus on the following questions:

In a decentralized organization:
(1) How are the critical corporate data elements defined?
(2) What are the policy mechanisms which support a gradual shift toward standardization?
And
(3) What kinds of technologies enable an organization to quickly adopt new standards without disrupting its operations?

These questions will lead us to the following discussion:

1.4.1 Policy Issues

Critical corporate data elements which an organization intends to standardize may be defined using different approaches. The organization can choose to have its senior management define them in a top-down approach. On the other hand, the organization can choose to have a large number of user groups discuss and reach an agreement on a set of critical elements in a bottom-up approach. Additionally, the organization can choose a middle course approach between the above two and have representatives from the main user groups define the critical data elements in a committee-based approach.

Standards can be enforced through different policy mechanisms. They can be enforced by requiring all units to modify their existing systems before a certain date. They can be only promoted to the units without forcing these latter to comply. Finally, they can be enforced by placing the burden of implementation at the interface level of the supplier and receiver of a stream of data.

1.4.2 Technology Issues

In order to support a gradual company-wide standardization, a new technology must be developed to enable information systems to both send and receive data having different definitions from one computer to the other. We will see that a data element can be defined by two characteristics: representation and interpretation. The representation indicates whether the data item is, for example, an integer or a string of characters. The interpretation indicates

what the data items "mean"; it may indicate, for example, the accounting procedure on which the data element is based.

Computer scientists have traditionally called the technology able to convert data from one definition to another a *semantics mapping* tool. Such a tool can be built in different ways. One way consists of fully customizing the mapping mechanism to the needs of a particular information system in order to achieve fast execution time. A second way consists of building a general-purpose mapping mechanism which can be used by a variety of information systems, albeit at the price of a slower execution time.

1.5 THESIS HYPOTHESIS

In order to answer the three questions discussed in the previous section, this thesis propose the following three-faceted hypothesis:

First, critical data elements are determined through a coalition-building process among representatives of the main organization user groups.

Second, data standards are enforced by:

(a) distributing roles and responsibilities and establishing a cross-organizational data standards committee whose role is to assign controllership, ownership, and custodianship to the critical data elements; and

(b) establishing self-interested policies at the "interface level" among the organizational units, whereby a source of some of the key data elements is not forced to modify its IS base in order to comply with the standards, but should be able to provide the data in the standardized form if asked by another unit.

Third, because units are not forced to modify their existing IS base in order to comply with the new standards, they are provided with a general-purpose semantics mapping tool which allows them to either send or receive the critical data in their standardized forms.

1.6 DISCUSSION PLAN

The goal of the thesis is to provide technological and policy guidelines to a decentralized organization which is willing to embark on a Focused Standards methodology.

The data that will be used to reach this goal is based on a literature review, case study material,[2] and the author's personal technical contribution.

In chapter II, the existing theory on decentralized organizational structure and loosely-coupled organizations is reviewed in order to identify the main organizational forces that a firm faces.

In chapter III, the main information technology components which allow connectivity are described. The chapter will also present the main technical obstacles when building connectivity in a decentralized organization.

Chapter IV will describe and compare the proactive and reactive connectivity approaches. The chapter will finally introduce to the Focused Standards approach.

Chapter V will be entirely devoted to the study of Focused Standards. The chapter will examine the methodology, its advantages and disadvantages, and various implementation options. At the end of the chapter we finally recommend an implementation strategy based upon the coupling of a policy mechanism with the deployment of semantics mapping tools.

Chapter VI will present the functional requirements that a semantics mapping tool should satisfy. Chapter VII will describe an architecture to support these requirements, and chapter VIII will present and illustrate the main algorithms used by a semantics mapping tool.

Chapter IX will conclude the thesis.

-oOo-

[2] Two companies were interviewed: Citibank's department North American Investment Banking (NAIB) in May 1989, and Salomon Brothers, Inc. in November 1989.

Chapter II.

ORGANIZATIONAL FORCES

This chapter has three objectives. The first will be to examine the three fundamental and conflicting forces that a decentralized organization is exposed to: integration, autonomy, and evolution. We will also discuss how these three forces interact, and in particular, how an effort to integrate a decentralized organization is contingent on the required levels of autonomy and evolution. The second objective will be to examine how information technology allows an organization to integrate its various activities. Finally, we will discuss the technical and political issues that an integration project is likely to face.

2.1 INTEGRATION

The force of integration has been shaped by the pressing need that most decentralized organizations feel to better coordinate their activities. Indeed, an increasing competition requires organizations to continually improve their efficiency and effectiveness in order to sustain their competitive edges. For a decentralized organization, it means more than anything else that better coordination and integration must be established among its multiple components

in order to share and move resources, exchange important information, and increase the overall business knowledge. This strategy has sometimes been referred to as "globalization."

Competition increases for several reasons, the complete description of which are beyond the scope of this thesis. One reason, however, which is worth noting, is the advent of internationalization, whereby firms gain market shares outside of their national boundaries. Indeed, an increasing number of large and middle-size firms today are looking off-shore for their competition, market opportunities and capital.

Some firms have managed this trans-borders expansion by establishing decentralized units that can develop local expertise. However, as the competition on the international scene intensifies, the decentralized management approach by itself is not sufficient to assure a competitive level of productivity [Bartlett and Ghoshal, 1988]. The various subunits of the organization spread all around the world have to share resources (e.g., capital, human, machines), and information (e.g., data, information, expertise, wisdom).

Strategic goals, such as decreasing time to market, handling risk in a global manner, improving customer satisfaction, and driving costs down, all may be achieved if the organization can coordinate the exchange of information, people, and resources among its various units. Time-to-market refers to both the firm's ability to develop new products quickly, and to deliver the products in its current portfolio effectively. Improving customer service relies on management's ability to maintain an organization-wide knowledge of customers' and equipment's status and problems.

Globalization has emerged as a key strategy that large corporations are currently adopting to achieve the above strategic goals. The company Citibank, for example, is attempting to develop a global information system that will allow the bank to satisfy the demands of multinational corporate customers for real-time access to the financial status of their "global empires" [Layne, 1990]. The example is interesting since it shows that Citibank is adopting a global strategy because its customers are adopting a global strategy too.

Increasing customer satisfaction may also be dependent on the ability of a company to aggregate information. An increasing number of firms are reorganizing their structures around customer services, and product maintenance around accounts. The North American Investment Bank (NAIB) department at Citibank changed its approach from a functional separation by

product to a separation by customers.[3] In such a transition, the goal is to transform data aggregated at the product level into data aggregated at the client level. Figure 2.1 summarizes the dynamics leading to integration.

The management of a decentralized organization is a topic that management and organizational behaviour scientists have studied and documented. A recent study made at MIT explains that there exists a key principle for the achievement of the strategic goals mentioned earlier [Rockart and Short, 1988, p. 13]. This principle is the management of interdependencies among the various activities of the firm. This principle is not new, and in fact managing interdependencies is one of a firm's oldest organizational problems. Dr. Rockart explains that management ought now to think in terms of multi-function, multi-level, multi-organization in order to achieve an appropriate control over its activities and positions. The classic management techniques which optimize the business bottom line by functional departments, product line, and geographical regions are sub-optimal because they do not take into consideration the interdependencies among these organizational units. One reason explaining why organizations optimize each unit separately is that these units are fairly autonomous. In the next section I discuss this force of autonomy.

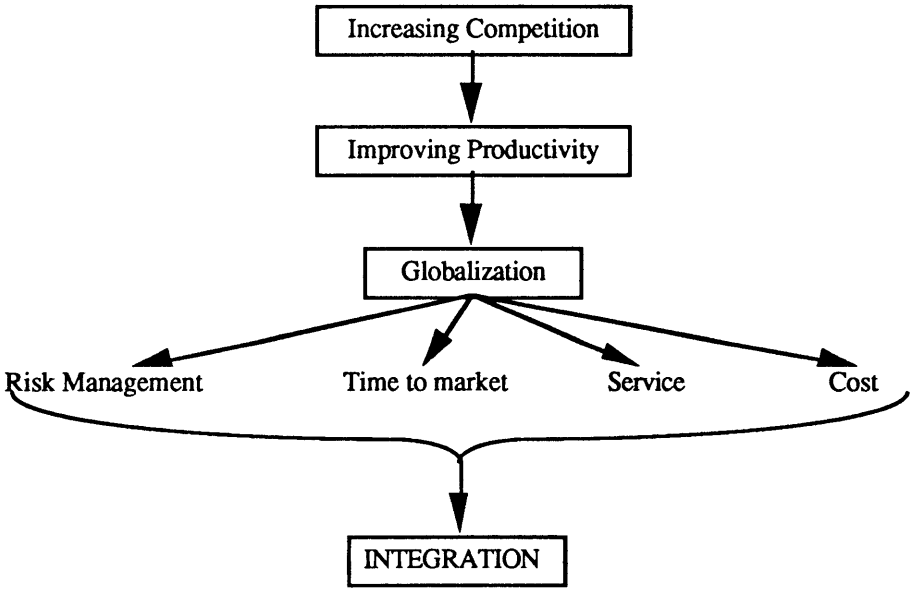

Figure 2.1: Causes of integration

[3] Interview with Judith Pesin, Integrated Database project manager at North America Investment Bank (NAIB), Citibank, New York, April 24, 1989.

2.2 AUTONOMY

A large organization involved in a wide range of products and/or services typically manages its tasks by creating independent units. Each unit can deal with a specific product or service and runs as a mini-company. The separation in a collection of units may be based on product types, regional areas, or countries in the case of a multinational firm. A unit may have its own strategies, technology deployment plans, personnel management, and marketing approaches that are most suitable for the local market. A subunit may be autonomous in terms of the magnitude of profit, and when the profit is made.

The decentralization can be quite extreme, as in the case of the company Citicorp. This company is an affiliation of small businesses which encourages and rewards entrepreneurship. Citicorp is only one example among a multitude of other decentralized firms where entrepreneurial actions are key to promotions. This kind of rewarding scheme gives incentives to subunit managers to develop a feeling of autonomy.

Organizational theorists argue that decentralization of managerial and technological resources allows each unit of the organization to develop local capabilities and initiative [Bartlett and Ghoshal, 1988, p. 8]. They argue that decentralization of assets also helps the units achieve local responsiveness and self-sufficiency. All these criterion, responsiveness, self-sufficiency, and initiative are aimed at giving each subunit the capability to fine-tune the organization's overall strategy to the local needs.

There is presently a clear tendency toward disintegration and distribution of responsibilities [Drucker, 1988, p. 47]. Hubert also argues that the increasing complexity of the world will shift firms toward a collection of very specialized subunits [Hubert, 1984, p. 931]. An increasing competition also requires organizations to have even more responsive and innovative units, which in turn requires even more autonomous units. Figure 2.2 summarizes the dynamics leading to autonomy.

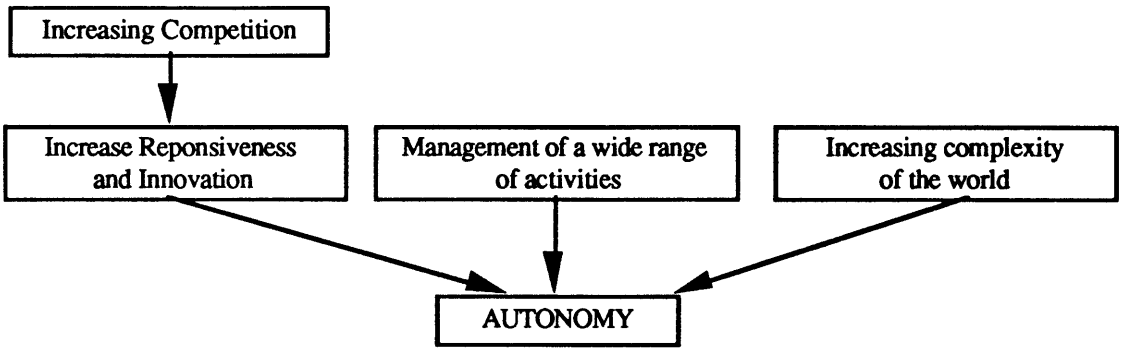

Figure 2.2: Causes of autonomy

2.3 EVOLUTION

An organization has to evolve in a continous manner due to the volatile nature of the environment in which it lives. A volatile environment implies that the location of an opportunity (or threat) is often different from where the company's appropriate response resources are situated. This is so because environmental opportunities and threats move from location to location, while organizational resources are not easily transferable even within the same company [Bartlett and Ghoshal, 1988, p. 12]. Hubert also explains that the need is not only to be able to adjust to the new situation, but also to be able to adjust on a continuous basis, since the external environment changes continuously too. Hence, organizations need to have flexible structures.

The omnipresent risk of mergers, acquisitions, and divestitures is one factor which causes volatility in the business environment. A recent survey of the U.S. electronics and computers industry showed that half of the CEOs expected their companies to be acquired or merged in the next five years [Betts, 1990]. When a company merges with another one, there is always a dilemma about which company's information system base has to be modified and integrated into the other one. An organization with a flexible structure is able to respond quickly to a merger or acquisition situation.

In order to respond to the changing environment, organizations' internal structures are changing toward a lighter and more flexible structure through a re-definition of roles, power, and hierarchy. Ultimately middle management will be eliminated and moved either up or down in the hierarchy [Drucker, 1988, p. 48]. Top managers are taking larger portions of the innovative, planning, and other creative functions to run the business. The main reason given for cutting middle management is to gain speed. Companies must be able to launch new products

quickly and alter existing ones for big customers. One solution that companies are currently trying out is to create a flatter organization in which information flows quickly from top to bottom and back again.

Team-based, problem-focused, often-changing work groups are also a trendy solution. "Collective entrepreneurship" with a few middle level managers and only modest differences between senior managers and junior employees is developing, leading to an organization composed of teams [Benjamin and Morton, 1986, p. 7]. The advent of "groupware", defined as better decision making aids, is becoming increasingly popular [Bullen and Johansen, 1988]. Business teams provide a very flexible organizational structure that allows a company to respond quickly to change. Companies form multi-department teams (e.g., six to twelve people), including engineers, manufacturers, and marketers, with the authority to make every decision on how products will work, look, be made, and cost [Dumaine, 1990]. Because the team doesn't usually need to send each decision up the line of approval, it can meet strict deadlines. Figure 2.3 summarizes the causes and effects of evolution.

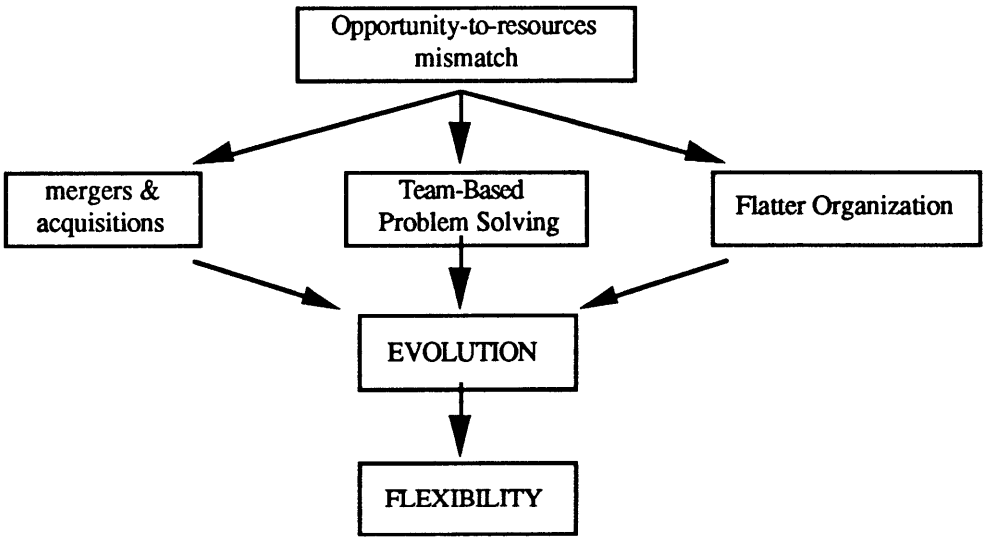

Figure 2.3: Evolution: causes and effects

2.4 INTERACTIONS OF THE THREE FORCES

The forces of integration and autonomy may conflict. Later in the thesis, we will show how the two forces can, in fact, be reconciled. Autonomy and local entrepreneurship can create a lack of global vision on the part of the subunits. The case study of an international bank gives the following statement from a top manager in regard to the introduction of a system measuring global risk [Massimo, 1987, pp. 113-114]:

"... most managers at the local level either don't have the time to or really don't consider it important to manage global risk. What they really care about is their own local risk. Given that this bank is so entrepreneurially independent, and that people are encouraged to really produce, in many cases the bank manager in Germany ... doesn't find it in his or her best interests to worry about what goes on in France."

The force of evolution further complicates the basic interaction between integration and autonomy by requiring the continuous adjustment of the fragile balance between them. Figure 2.4 is representative of these interactions.

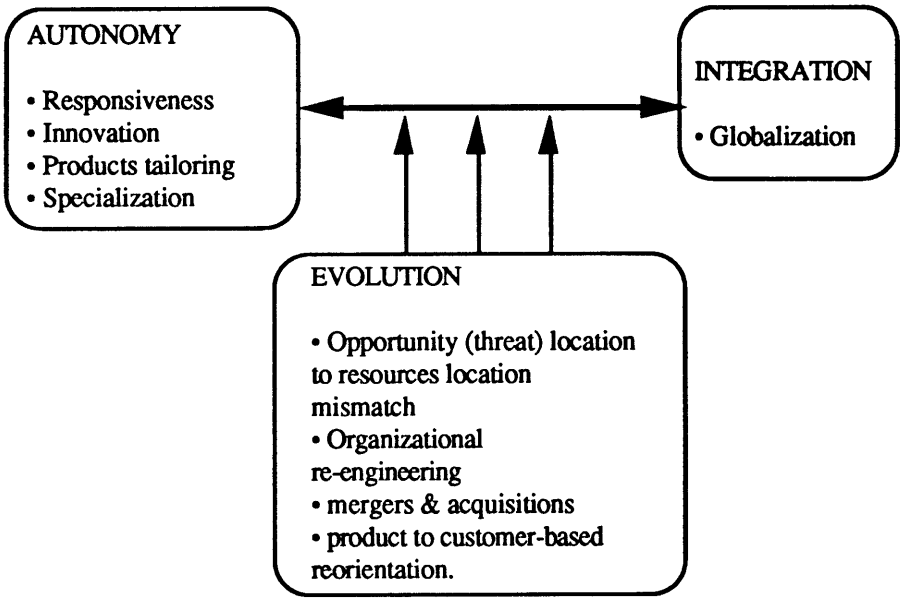

Figure 2.4: Interaction of the forces of integration, autonomy, and evolution

2.5 INFORMATION TECHNOLOGY-BASED INTEGRATION

2.5.1 The Advent of Connectivity

There is a consensus among management and organizational scientists which says that in a world where complexity increases, more relevant information must be acquired in order to reduce both uncertainty (the absence of information) and equivocality (lack of understanding).

Integrating various information sources can help managers access relevant information. Hubert argues that the expanded volume of information and the need for its integration will cause more formal integration to take place on both a routine and non-routine basis [p. 944]. He explains that information acquisition and distribution are keys to a good coordination, and that the ability to integrate relies upon internal and external data accessibility, information-

sharing, and communications capabilities. In the remainder of this thesis, this set of capabilities will be defined as connectivity.

Recent improvements in the fields of distributed systems and communication networks have allowed the advent of connectivity and integration. The quality of communications has improved in several ways: with respect to the cost of transmission, the purity of data, the speed of access, the type of data, and the availability of data. Strategic applications, such as global risk management, global planning and control, and a variety of executive support systems requiring the integration of data existing in disparate and geographically-dispersed information systems, now become technically feasible. Thus, IT has enabled the birth of a multitude of both inter- and intra-organizational systems [Barrett and Konsynski, 1982].

2.5.2 IT-Enabled Integration

Four types of integration enterprises have been identified [Rockart, p. 17]: (1) inter-functional integration across the value-added chain; (2) intra-functional integration; (3) integration for team-based structure; and (4) integration for planning and control management. This taxonomy is given within the framework of a single organization, however, the results can be easily extended to the case of linkage of processes between separately owned organizations.

2.5.2.1 Inter-Functional Integration Across the Value-Added Chain

An organization's value-added chain is the series of actions between a supplier who provides raw materials and a customer who buys the end product. Integration along this chain can create three major functions: product development, product delivery, customer & service management. This integration is shown in Figure 2.5.

This phenomenon has traditionally been seen in manufacturing companies through the advent of Computer Aided Design (CAD) systems, Computer Aided Manufacturing (CAM) systems, and very recently Computer Integrated Manufacturing (CIM) systems. Each segment is briefly described below.

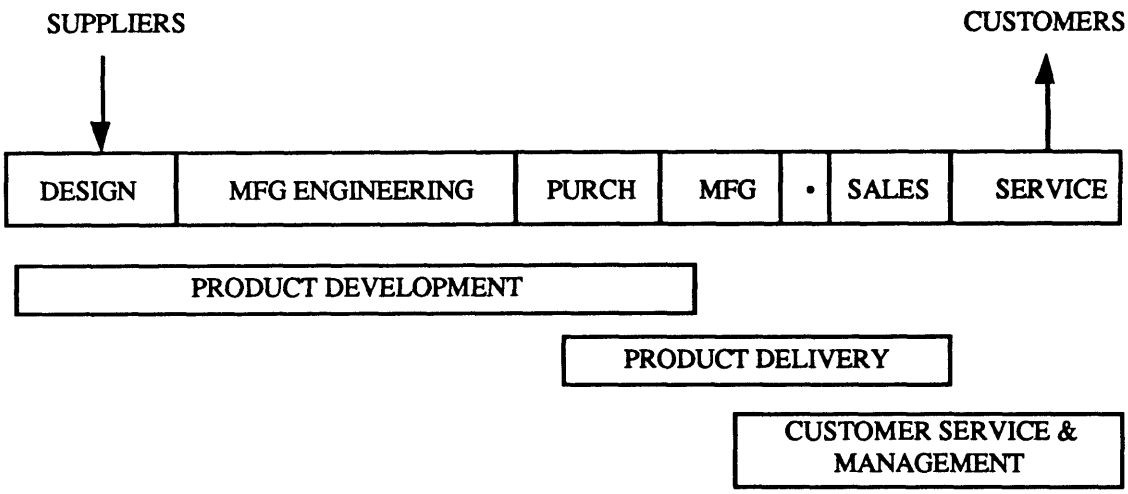

Figure 2.5: Integration along an organization's value-added chain

(i) Product Development

This segment is based on the use of CAD/CAM and CIM to provide integrated support for product designers, product engineers, materials purchasing, and manufacturing personnel involved in the design-to-production process. The goal of product development integration is to allow feed-back from the manufacturing designers to the product designers and vice versa, hence creating almost two parallel processes that once were purely sequential. The integration has allowed a more efficient utilization of the various departments capabilities, which has led to shorter product development time.

(ii) Customer Service & Management

This segment focuses on the customer side of the value-added chain. In most corporations the customer base is traditionally categorized around product type. By organizing the customer base around customers, as opposed to products, a firm can develop new marketing strategies and increase its bottom line. At Citicorp, the NAIB department recently decided to re-organize itself around customers. This new organization of the customer base will allow the bank to state at any point in time what products a given customer has either bought or sold to the bank, and what her aggregate profitability is. This *customer profitability* strategy, as the bank calls it, will allow the NAIB to categorize its customers by the magnitude of revenue generated and the type of products bought. Figure 2.6 is representative of the customer profitability approach.

(iii) Product Delivery

This segment attempts to integrate order entry, purchasing, materials resources planning, and distribution management systems. The goal of such an integration is to provide to the customer information on when an order will be completed, and to forecast and manage product shipment, outside suppliers, manufacturing and distribution processes.

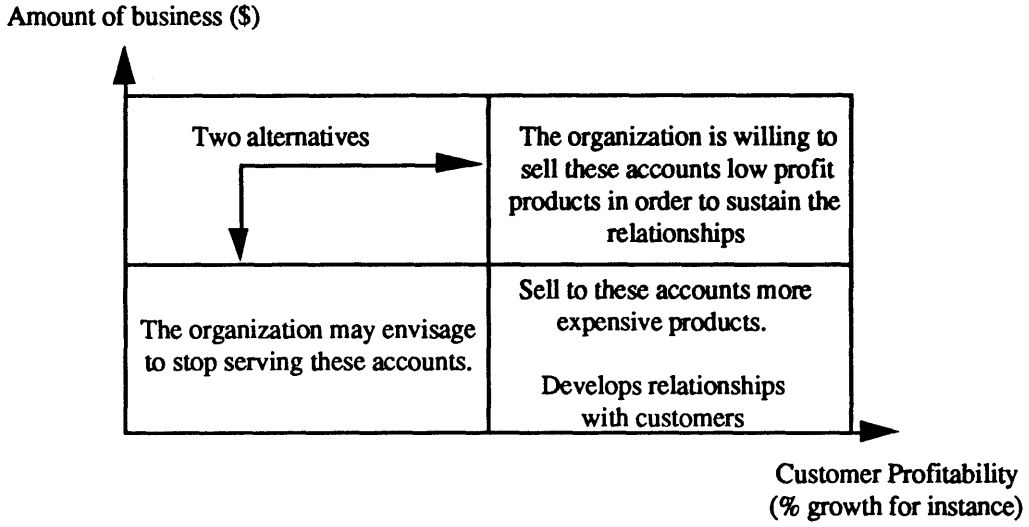

Figure 2.6: Customer profitability analysis

2.5.2.2 Intra-Functional Integration

The integration within a functional area consists of gathering the various organizational units involved in that area under a single unit. Departments may share, for example, the same customer base, the same products, and the same distributions channels. The integration aims at avoiding duplication of resources across the different units.

2.5.2.3 Integration for Team-Based Structure

The success of team-based approaches is based on a company's ability to integrate both people and the information they need to perform their job. The technologies of electronic mail, computer conferencing, and video conferencing systems help integrate people: members of a team who are geographically dispersed can now interact with the other members through the aid of the above systems. The technologies of distributed database management systems in turn help integrate the data: information existing in different computer systems and dispersed throughout the organization can be merged into a single information pool.

2.5.2.4 Integration For Planing and Control Management

The traditional managerial control process is a well-defined task that most corporations carry out in the same way. Before the new fiscal year, each subunit presents to top management the activities that it proposes to conduct for the upcoming year. The plans are discussed, and once agreed upon, they are monitored on a monthly basis through reporting to top management. Parallel to the formal planning and control activity is usually a process whereby senior management "keeps in touch" with the subunits to make sure the plans are respected. IT is now changing the planning and control process by allowing senior management to have on-line information regarding the activities of the subunits. Subunits may be required to submit their plans over the corporate network in a particular format. Having all the subunit plans in the same format enables the senior management to get to the key data faster, to perform comparisons among the reports more easily, and to allocate resources among the subunits more efficiently.

2.6 OBSTACLES

2.6.1 Technical Obstacles

All four integration types presented above attempt to integrate data from disparate computers scattered in the organization. Therefore, firms have begun to look at their collection of information systems with a more global point of view. In doing this, they have discovered a very heterogeneous environment, especially in the case of decentralized organizations where sub-units have had their own policies in terms of information technology deployment. This decentralized IT deployment has led to discrepancies in data definition, accuracy differences, and timing and coordination problems. The resulting lack of accessible, quality data seriously hinders data integration.

The advent of end-user computing and computer proliferation due to cheaper hardware and software has contributed to the creation of disparate, often incompatible, information systems distributed throughout an organization [Withington, 1980]. Along with the proliferation of equipment throughout the organization goes a proliferation of applications and approaches to the applications.

The good news with decentralized IT deployment is that local end-users can build applications that will fulfil their requirements. It is true that the computing needs are

different from one functional area to the other, hence different classes of computer hardware and software facilities are acquired. In addition, such decentralized IT deployment policies also allow end-users to have their data close to them, so that they can easily access and manipulate it. Notice that decentralized IT deployment goes hand in hand with the concept of autonomous units.

The bad news, however, is when the organization wishes to integrate some of its activities. Let us take, for example, the case of integration for customer service and management which we discussed earlier. The difficulty in successfully performing the integration is that a firm is traditionally organized around different product lines, and thus has built independent systems for each product division. When the firm tries to reorganize around customer segments, it may turn out to be difficult to pull together complete customer profiles because the entire IS base is hard-wired by product and lacks common customer numbers.

2.6.2 Political Obstacles

The re-engineering of a firm's structure is a highly political process due to organizational resistance. An organization has a certain "administrative heritage" which can have a strong inertia and can prevent any project entailing major organizational changes from succeeding. This heritage has been described by management scientists as the bundle of existing configuration assets, traditional distribution of responsibility, historical norms, values (or culture), and management style. In the case of the investment bank mentioned above, the heritage consisted of having autonomous traders who were only willing to take risk individually.

The dynamics among the various components of an organization's heritage have been studied by Dr. Harold Leavitt of Carnegie-Mellon University [Leavitt, 1965, chapter 27]. In his study, he considers an organization as being articulated around four main components: strategy, technology, roles, and organizational structure. The author states that one of management's key function is to adroitly maintain a "dynamic equilibrium" among these four components. In particular, Leavitt predicted that the advent of computers and management science would significantly change the structure and processes of most corporations. Figure 2.7 is representative of the dynamics between the various components.

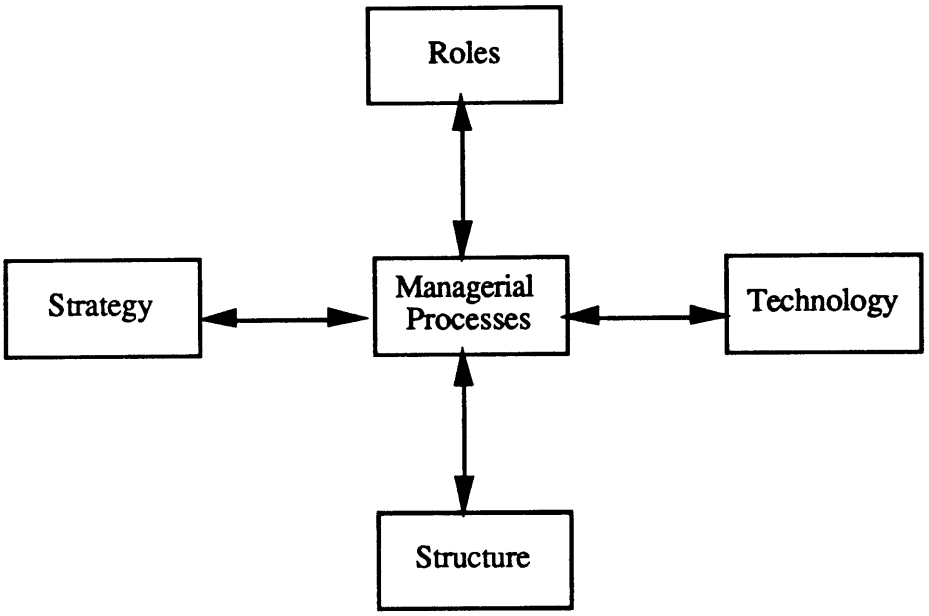

Figure 2.7: Adjusted Leavitt's organization model

Leavitt explains that external socio-economic environment and the technology are the two main driving forces that act on an organization's strategy, personnel, processes, structure, and technology. These two forces put the internal elements of an organization: technology, strategy, processes, people, and structure into motion. The resulting changes in any of the internal elements require re-equilibrating all the elements among each other in order to maintain the balance that a firm needs to be effective. The author also explains that people naturally tend to resist anything new that would change the *status quo*.

2.7 SUMMARY

We have identified three fundamental forces that a decentralized organization is continuously exposed to. These three forces are the following: (1) the force of integration which takes its root in the strategy of globalization that an increasing number of large firms are adopting in order to sustain competition; (2) the force of autonomy which gives organizational units the ability to be locally responsive and innovative; and (3) the force of evolution which is due to the highly volatile nature of the environment in which organizations live. We showed that integration and autonomy intrinsically conflict, and that the nature of this conflict and how to deal with it is made even more complex by the force of evolution.

We turned our attention to integration, and examined why information technology is key to the integration of various components of a decentralized organization. We also

presented four types of integration enterprises to illustrate our argument: (i) inter-functional integration along the value-added chain; (ii) intra-functional integration; (iii) integration for team-based structure; and (iv) integration for planning and control.

We presented two types of obstacles when conducting an integration project. The first type is related to the technology that is required to provide full integration of heterogeneous computer systems. The second type of obstacle that an integration enterprise will face concerns organizational resistance and the various political conflicts generated when modifying an organization's structure.

In summary, the implementation of a global strategy will require solving the problem of heterogeneous information systems. Chapter III will discuss the technical difficulties when attempting to integrate in an heterogeneous environment. Chapter IV and V will discuss the main connectivity approaches and examine how each approach deals with the technical and political obstacles.

-oOo-

Chapter III.

CONNECTIVITY: TECHNOLOGY, REQUIREMENTS, & APPROACHES

This chapter is divided into three parts. The first part describes the main information technology components for building connectivity. The second part describes the technical difficulties when building connectivity in an organization. Finally, the third part presents a few connectivity approaches that organizations have adopted in the past.

3.1 INFORMATION TECHNOLOGY COMPONENTS

In the previous chapter, we defined connectivity as being the ability to connect computer systems together for information sharing and exchange. Ideally, connectivity aims at building an organization-wide information systems architecture that links up all computer systems in a transparent manner.

We describe below the main IT components that lay the foundations for connectivity: (1) workstations; (2) shared distributed databases management systems; (3) communication networks; and (4) specialized processors [Madnick, 1989, pp. 10-16]. Figure 3.1 shows how these components are assembled.

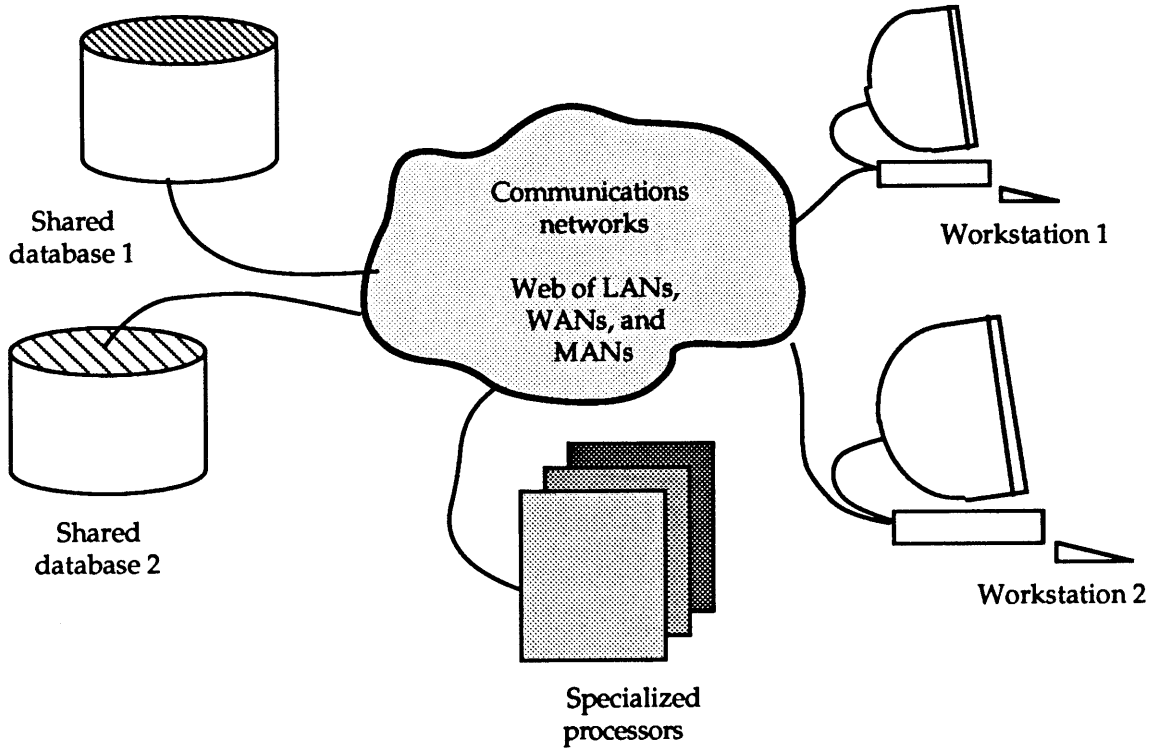

Figure 3.1: Main IT components

3.1.1 Workstations

The workstation is the entry point for the user (sometimes called terminal). It can be a personal computer that has local CPU power or a simple terminal that has no local CPU power. A workstation is typically connected to a communication network, and is thus linked to other workstations and computing resources (see Figure 3.1). One important characteristic of modern workstations is that they are specifically designed to be easily connected to other computers through a communications network.

3.1.2 Distributed Databases Management Systems

Shared access distributed database management systems are part of the building blocks for information-sharing capabilities within or among organizations. A distributed database is a collection of data that is physically spread over the sites of a computers network.

Only recently have distributed database management systems been available to organizations. Traditionally, a database resided on a single hardware machine with

associated secondary storage devices such as disks for on-line database storage and tapes for backup. It is true that such a database, called a centralized database system, can be accessed remotely via a terminal connected to the site; however, the data and database management system (DBMS) software reside principally at a single site. As the cost of on-line storage devices has decreased and distributed database management system (DDBMS) software has progressed, there has been a trend toward the distribution of computer systems over multiple sites that are connected together via a communications network. The following is a discussion of the terminology commonly used to explain the development of distributed databases:

(i) Nature of database applications

Many database applications are naturally distributed over different locations. For example, a company may have locations at different cities, or a bank may have multiple branches. It is natural for databases used in such circumstances to be distributed over these locations. A company may keep a database at each of its locations to store the data pertaining to that location. Many local users access only the data at this location, but other global users, such as company management, may require access to data stored at several of these locations.

(ii) Reliability and Availability

Increased reliability and availability are the most common potential benefits cited for distributed databases. Reliability is defined as the probability that the system will be running at a certain point in time, whereas availability is defined as the probability that the system will be continuously available during a given time interval. With the distribution of both software and data at multiple sites, one site may fail whereas the other sites continue their operations and their applications and data remain available to users. A smart duplication of software and data over the sites can increase both reliability and availability since one site can relay a failed site. In a centralized approach, the failure of a single site will bring the whole system down.

(iii) Custodianship

Distributed systems usually allow data sharing while maintaining some degree of local control over the software and data. Thus, distributed systems represent an interesting compromise between company-wide information-sharing and local ownership.

(iv) Performance

Performance can be improved by distributing a large database over multiple sites, with smaller databases existing at each site. Each site will have to process a smaller number of queries than

if it was a centralized DBMS, and thus will show better performance. In addition, due to the distribution, a query initiated by a user might be broken down into several sub-queries that are sent to the different nodes of the network and processed in parallel, which can reduce the transaction execution and response time.

3.1.3 Communication Networks

Communications networks fall into three categories: (i) local area networks; (ii) internal and external wide area networks; and (iii) metropolitan area networks.

(i) Local Area Network (LAN)

A LAN is the communications infrastructure of a single organization, such as a corporation, a hospital, or any other public institution. A LAN is limited in the distance it has to cover (floor, building, campus, or plant location). Within the same floor, two workstations would typically connect through a LAN.

(ii) Wide Area Network (WAN)

There are two types of WAN's: internal and external. An internal WAN links the various sites of a geographically dispersed organization. Such networks are usually built to carry voice (telephone conversations) and data (computer communications). The organization can lease telecommunications lines and customize the network for maximum efficiency and minimum cost. The resulting network is managed, controlled and owned by the organization. An external WAN is the traditional public network which is owned by a telephone company. Maintenance and services are provided by the telephone company which charges customers on a fee basis, be they residential users or large corporations. An end-user with a workstation would use an internal or external WAN in order to connect to a corporate database residing in a different building, different city, or different country.

(iii) Metropolitan Area Network (MAN)

A MAN is made of individual LAN's, hence providing high performance of data throughput across, for example, a city or a university campus. A MAN can be considered an internal service linking different buildings where the same organization has offices, or an external service providing high performance services to residential users as well as corporations on a fee basis, or an intermediate form such as a high performance network linking all financial institutions in a city.

The communication network sketched in Figure 3.1 is thus a complex web of LAN's, WAN's, and MAN's.

3.1.4 Specialized processors

Some computers on the network act as transaction processors and/or coordinators to check authorization when requests are sent to databases. Once a request is validated, the processor controlling the database executes a chain of actions to fulfil the request. Specialized processors may also be the high-speed and highly-reliable processors that are used for elaborate mathematical calculations or expert systems software.

***

A key feature of this IS infrastructure is that a user need not be aware of its underlying complexity. The questions of where the data is, where the software to process the data is, which processor the instructions are being executed on, and how the data and instructions (also data) are moved around in the network and are all taken care of by the "network operating systems software." Thus, the network provides a transparent computing environment.

3.2 CONNECTIVITY REQUIREMENTS

3.2.1 Physical versus Logical Connectivity

There are two types of connectivity: physical and logical [Madnick and Wang, 1988, pp. 15-18]. Physical connectivity aims at assuring the transmission of raw data, and is neutral as to what the contents of the transmitted messages are. Physical connectivity is centered around the communications network technologies presented above. Physical connectivity focuses on how to connect, for example, two different LAN's. There are different types of network interconnection equipment: *Bridges* are used to connect two LAN's of the same type (e.g., Ethernet-to-Ethernet or Token-Ring-to-Token-Ring), and *gateways* are used to connect two LAN's of a different type. *Routers* are used to store information about networks (LAN, WAN, or MAN) and network routes, even for networks to which they are not directly attached.

Logical connectivity focuses on the contents of a transmitted message in order to understand what the message means and act accordingly. Logical connectivity also takes into consideration where the data resides.

Whereas the problem of physical connectivity is relatively well understood and many vendors are providing equipment and services, the problem of logical connectivity is still unclear for most computer scientists and hence represents a field of new research.

Logical connectivity is further sub-divided into data connectivity and semantic connectivity. Data connectivity focuses on the location of data, and semantic connectivity focuses on the semantics of data.

3.2.2 Data Connectivity

Data connectivity is usually associated with the technologies of distributed database management systems. There are two types of distributed systems: homogeneous and heterogeneous. An homogeneous distributed database management system is characterized by the fact that the data must be distributed on the same or very similar database software [Gref, 1987]. Hardware equipment, however, need not be the same: data may be distributed on PCs, minicomputers, and mainframes as long as the software running on the computers has been specifically designed for sharing data.

A heterogeneous distributed database management system, however, makes no assumptions about the type of software existing on the different computer systems which are accessed. Each individual DBMS may be of a different type, such as flat files, network, hierarchy, or relational [Date, 1986, pp. 19-20].

Achieving transparency and independence are the key tasks of a distributed system. Transparency means that the user sees a single database schema and query language. Full transparency requires that the following requirements be met:

- *Retrieval transparency:* means that the same results are obtained regardless of the site from which the request was sent.
- *Update transparency:* means that the user can update any data item on any site from any other site.
- *Schema transparency:* means that the result of a schema modification command is visible at all sites, regardless of the site from which it was originated.
- *Performance transparency*: means that all sites have comparable performance for the same query. This requirement means that the system is using a global optimizer that determines the best route and best site, or series of sites, to execute the query; the optimization is based on

minimizing the amount of data transfer and maximizing the business transactions (e.g., a mainframe versus a personal computer).

• *Transaction transparency:* means that a single transaction composed of several updates is properly and efficiently executed against the correct sites automatically.

• *Copy transparency:* means that multiple redundant copies of data are supported and the system automatically maintains consistent values and efficiently makes use of these copies to optimize performance and recover from failures.

Independence means that the system is independent of external factors, such as (i) site crashes, (ii) recovery actions, (iii) communication network, (iv) hardware and OS, and (v) specific DBMS's used. Only a few database vendors have tackled the design of a full-blown homogeneous distributed databases management system. The companies Oracle and Ingres, for example, have some prototypes that have shown satisfactory results, even though they support only a few of the above features. There is still much work to be done in the domains of query optimization, updates, and the development of gateways into other systems.

In regard to the heterogeneous distributed database management systems, the situation is even less encouraging, as there is no available product to date. Prototypes do exist, but they are still in the development phase [Bhalla, S. et al, 1987]. Researchers in the field of heterogeneous distributed systems have narrowed down their task by abandoning the question of updates. The overall purpose of an heterogeneous distributed DBMS has thus been defined as to access, aggregate and update information maintained in existing, distributed heterogeneous DBMS's through a single uniform interface without changing existing database systems and without disturbing local operations. Such a system has to perform the following tasks in order to answer an end-user request:

(1) parse the query and generate a series of sub-queries expressed in the different languages supported by the various local DBMS's;
(2) formulate a plan for executing the sequence of subqueries and data movement operations;
(3) implement a plan for accessing data at individual local sites;
(4) resolve incompatibilities between the databases, such as differences in data types and conflicting schema names;
(5) resolve inconsistencies in copies of the same information that are stored in different databases; and
(6) combine pieces of retrieved data into a single response to the original request.

This series of steps helps distinguish between data connectivity and semantic connectivity. The first three steps focus on data connectivity, i.e. where the data is and how to access it. The last three steps focus on semantic connectivity, i.e. what the data looks like, means, and how it can be merged.

3.2.3 Semantic Connectivity

Semantic connectivity among computer systems consists of having these computers understand each other's data semantics. The semantics of a data element can be defined as a combination of three characteristics: (1) the data element's name; (2) the data element's representation; and (iii) the data element's interpretation. Figure 3.2 represents our view of the semantics of a data element.

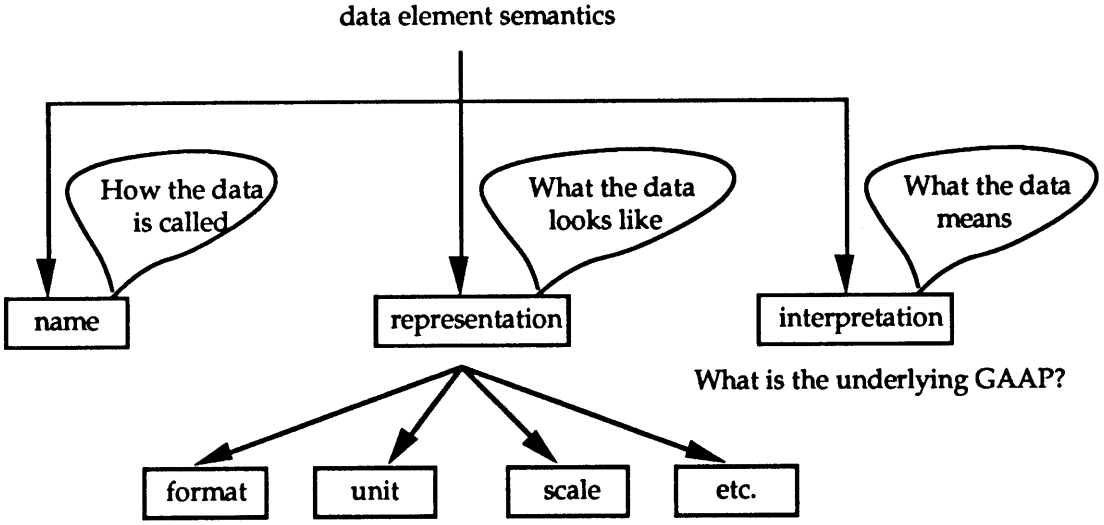

Figure 3.2: The semantics of a data element

In the next few pages we compare two computer systems in regard to the financial information they both provide for the same company. In doing so, we can illustrate a few data incompatibilities that are frequently encountered when exchanging data among computer systems, and that must be resolved in order to achieve semantic connectivity.

The two computer systems considered are the on-line financial services of Finsbury's Dataline and I.P. Sharp Disclosure.[4] We propose to compare financial information that the two services provide for Honda Motor Co.[5]

---

[4] I. P. Sharp Disclosure is a Toronto-based on-line service owned by Reuters Holdings PLC, London. The service provides financial information on a fee basis for about 12,000 companies which report to the Securities Exchange Commission in the U.S. Finsbury Dataline is a London-

	1985	1986	1987	1988	1989
CF	19,860,228	19,870,228	19,880,331	19,890,331	-
NI	146,502	83,689	56,676	97,299	-
NS	2,909,574	2,960,985	1,650,781	3,489,258	-
NRTAX	0.09	0.04	0.03	0.04	-
NRCEX	0.19	0.11	0.07	0.11	-

Table 1: I.P. Sharp Disclosure's output for Honda Motor Co.

HONDA MOTOR CO HOND JAPAN

INCOME STATEMENT Yen (m)

Period Ending	28-02-86	28-02-87	30-09-87	31-03-88	31-03-89
SALES	2909574.	2868305.	1778384.	1650781.	3489258.
EARNED FOR ORDINARY	146502.	83689.	50834.	56676.	97299.

ACCOUNTING RATIOS

Period Ending	28-02-86	28-02-87	30-09-87	31-03-88	31-03-89
RETURN ON SHAREHOLDERS EQUITY	19.57 %	11.32 %	6.63 %	7.27 %	10.79 %
RETURN ON CAPITAL EMPLOYED	30.99 %	17.86 %	9.79 %	8.89 %	12.70 %

Table 2: Finsbury Dataline's output for Honda Motor Co

The examination of the outputs provided by the Disclosure and Dataline databases shows that data incompatibility can occur on each characteristic of the semantics of a data element, i.e. on its name, its representation, and/or its interpretation.

Name incompatibility: the data item is called "NS" in Disclosure and "Total Sales" in Finsbury. This type of incompatibility is commonly referred to as *attribute naming*.

Representation incompatibility: the representation of a data element can be thought of as a collection of characteristics such as format, unit, and scale.

---

based financial service provided by Reuters Holdings PLC; it has financial statements from the previous five years and forecasting facilities for over 3000 corporations, covering primarily U.K. corporations as well as a number of other major European and Japanese firms.

[5] The outputs shown below correspond exactly to what a user sees on her screen.

format - the data elements' format in I.P. Sharp is characterized by the presence of a comma to indicate thousands (i.e., 2,909,574), whereas the figure has no comma in Dataline but a dot at the end of the string (i.e., 2909574.).

unit - the data elements' unit is not provided in Disclosure's output, and in fact there is no explicit attribute giving a company's currency in Disclosure. In Dataline, however, the unit is provided in the tabulated output (i.e., Yen).

scale - the scale is not provided in the Disclosure database. It can be obtained from the manual which is obtained when subscribing to the service; it turns out that financial information for all companies are given in their respective currencies with a scale factor of a thousand.

In Dataline, the scale is provided in the tabulated output (i.e., m). Note that there is still an issue of interpretation since the symbol "m" may mean different things to different people; for example, a natural reaction would be to think that "m" means "million". In reality, "m" is a scale of a thousand. The interpretation is further complicated by the fact that the scale notation is not consistent across companies; we have encountered, for example, the symbol "000s" meaning a scale of a thousand as well.

Interpretation Incompatibility: the interpretation of a data element refers to what the data means. In the case of financial information, for example, the interpretation may differ depending upon the General Accepted Accounting Procedures (GAAP) adopted in the system. In the case of Disclosure and Dataline, for example, the return on capital employed is 17.86% in Dataline and 0.03 in Disclosure in 1987. Ignoring the data representation difference, there is obviously a significant discrepancy between the two numbers (a factor of 6 between the two!). Neither number is more accurate than the other, they are just based on different accounting procedures that the two on-line financial services have chosen to adopt.

Another example of interpretation discrepancies was given by one manager of a company that provides on-line financial information. As this manager explained to us, a customer of theirs might believe mistakenly that the field *P/E* ratio provided for UK equities is calculated on a net basis.[6] However, as he pointed out, the fact that it is not is not a problem in itself, but the fact that "detective work is necessary to uncover the real meaning of the field

[6] The name of this company will not be disclosed due to the confidentiality of the information.

is a problem". Indeed most of the semantic connectivity mechanisms consist of carrying out detective work in order to find out what the data means.

3.3 CONNECTIVITY APPROACHES

The question that we are now examining is how organizations have approached the problem of data incompatibility across their information systems. Two types of approaches have been tested by organizations in the past.

The first type consists of rebuilding the complete organization's information system architecture and standardize hardware and software equipment, and data elements definitions throughout the organization. This approach, which has been qualified as "proactive" [Madnick, 1989, p 26], takes into consideration all the organization's connectivity requirements, and build a complete architecture accordingly. The clear advantage of this approach is that it fully eliminates the presence of data incompatibilities. The clear disadvantages with this approach, however, are that it causes a significant disruption of the organization's activities, and it is extremely time consuming and expensive.

Realizing that rebuilding the IS architecture is a difficult task whose outcome is even not guaranteed, organizations have adopted a second type of approach consisting of coping with the current information system architecture and its flaws! In doing so, they have built highly customized bridges among computer systems in order to satisfy ad hoc connectivity requirements. A customized bridge between two computer systems can solve the particular data incompatibility issues pertaining to the exchange of data between these two computers. If the representation of the date, for example, is "mm/dd/yyyy" in the first computer and "dd-mm-yy" in the second computer, a specific program is written to map the two formats, and these two formats only. In the remainder of the thesis, we will refer to such a bridge as a data semantics mapping tool.

Dr. Madnick has qualified this second type of approach as being "reactive". Such bridges are built to meet both physical and logical connectivity requirements, and are put together as quickly as possible in order to solve one particular problem or respond to one particular opportunity. The approach has the advantage of providing a quick and relatively cheap solution. The problem with this approach, however, was pointed out by a manager at Bankers Trust when he mentioned that developers and programmers are now becoming less and

less willing to spend time building such bridges.[7] He explained that the task is usually long and tedious, and has little visibility, and that the programmers involved feel that the task is one that must be dealt with over and over again. Indeed, such bridges are built almost manually, and with little use of the experience gained from previous situations.

It is important to point out that the above two approaches are somewhat extremes, and that there exists a wide array of connectivity approaches between the two. Thus, a less radical approach than the first one we mentioned, but still proactive, consists of focusing on the standardizing of the definition of data elements only. The approach aims at reaping the benefits of standardization, but makes its task more manageable by leaving the current IS architecture as such. It turns out, however, that a company-wide standardization of all data elements is also an arduous task.

Finally, a fourth avenue towards connectivity that organizations have explored consists of building bridges in a systematic way. This approach is based on the introduction of Composite Information Systems (CIS) which are based on a general-purpose integration mechanism. Any ad hoc integration need can thus be satisfied by the installation of a CIS system. The benefits of the CIS approach are numerous: it provides, for example, a quick way to achieve connectivity, the technology is relatively cheap since the same software kernel is used for all CIS systems, and it can accommodate with any computing environment. The problem with the CIS approach, however, stems from the fact that the technology it employs is still in a research phase and has some performance limitations.

We provided in this section a synopsis of the different connectivity approaches that an organization can adopt. We leave to chapter IV and V the task of comparing these approaches in more detail, and elaborate recommendations as to how to achieve connectivity.

3.4 SUMMARY

We started this chapter by describing the main IT components that are used to build information systems connectivity in an organization. We mentioned the workstations, shared access distributed database systems, communications networks, and specialized processors.

---

[7] Interview with Robert Phelan from Bankers' Trust, Vice President, Risk Management Technology, Center For International Financial Services Center (CIFSC), MIT, Mass., 28 February 1990.

Then, we defined connectivity in more specific terms. Connectivity was divided into two categories: physical and logical. We explained that physical connectivity focuses on how to transmit data, while logical connectivity focuses on where the data resides (data connectivity) and what it looks like and means (semantic connectivity). We presented a few examples of data semantics incompatibilities which must be solved to achieve semantic connectivity. In particular, we saw that incompatibility can occur on the data element's name, representation, and/or interpretation.

Finally, in the previous section we presented various approaches to physical and logical connectivity. Chapter IV will proceed in more detail with the description and comparison of these connectivity approaches.

-oOo-

Chapter IV.

CONNECTIVITY APPROACHES

We continue here the discussion that we began in chapter III on how to achieve connectivity within the framework of a decentralized organization. The discussion is articulated around three parts. The first part recalls the approaches that were identified in the previous chapter, and positions them along a connectivity approaches spectrum. The second part reviews each approach individually. For each one, we describe its methodology, and discuss its advantages and disadvantages. In the third part, we describe a methodology that we believe to be technologically, economically, and politically feasible. This third part also provides recommendations for the implementation of this methodology.

4.1 CONNECTIVITY APPROACHES SPECTRUM

Chapter III identified the following four connectivity approaches:

(i) "proactive";

(ii) "reactive";

(iii) Data Resource Management-based; and

(iv) Composite Information System-based.

The "proactive" approach advocates the rebuilding of the complete organization's information system infrastructure so that all connectivity requirements can be met. This approach will be referred to as "Enterprise Modelling".

The "reactive" approach advocates to stay with the current *status quo* and manually build bridges among computer systems when needed. This approach will be referred to as "Manual Bridges".

The Data Resource Management (DRM)-based approach advocates the standardization of the definition of all the corporation's data elements, and the storing of these definitions in a centrally maintained and controlled data dictionary.

Lastly, the Composite Information System (CIS)-based approach advocates the use of a general-purpose data integration mechanism. A CIS system is based on a series of bridges that are used to fetch data residing on disparate computers. These bridges must satisfy the same set of functions than those built through the Manual Bridges approach. In fact, any bridge previously built, be it under the category Manual Bridges or not, can be used by a CIS system. However, in addition to the data integration mechanism, the CIS approach also provides a methodology for building bridges. Hence, all bridges used by the CIS system are built from the same general-purpose architecture.

These four approaches can be positioned on a methodology spectrum starting from a "proactive" end, and ending at a "reactive" end (see Figure 4.1). We positioned Enterprise Modelling at one extreme of the spectrum for being the most proactive methodology. Then, comes DRM which is also proactive, but less radical than Enterprise Modelling. Then, comes the "NULL" approach which consists of not doing anything! It is the first time in the text that we mention this approach; indeed, it has little interests. Let us point out, however, that it has been the most commonly adopted approach in the past. Organizations have adopted this

ultimately conservative approach not because of a lack of desire to gain connectivity, but because of the risks inherent in each individual approach that we mentioned. Faced with the uncertainties of these latter, many organizations have decided to stay with the current IS base [Best, 1990]. To the right of the NULL approach on the spectrum lie the reactive methodologies, where Manual Bridges constitutes our extreme. CIS is positioned half-way between the NULL and the Manual Bridge approaches.

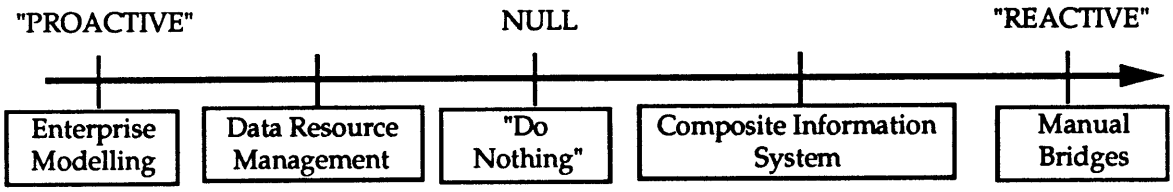

Figure 4.1: Connectivity Approaches Spectrum

In the following sections, we shall organize our discussion by distinguishing first, the proactive methodologies, and second, the reactive ones.

4.2 PROACTIVE APPROACHES

4.2.1 Enterprise Modelling

The enterprise modeling approach consists of building a corporate-wide strategic data model designed to serve as a blueprint for all future systems development. James Martin's Strategic Data Planning is one methodology which aims at building such a company-wide information systems architecture [Martin, 1982, chapter 1]. In building this architecture, the organization plans all the connectivity requirements in advance: choices in hardware, software, telecommunications, operating systems, programming languages, and data definitions are all thought through in advance so that the various units of the organization can exchange and share data as desired. Exactly how the company-wide architecture is established is described below.

4.2.1.1 Enterprise Modeling Methodology

Enterprise modeling is a top-down approach aimed at modeling data in the context of business functions. The methodology results in the specifications of the main databases to be

created and by indicating implementation priorities. The database implementation itself is conducted in a bottom-up approach. The diagram in Figure 4.2 illustrates this technique.

The top-down approach starts with the development of an enterprise model (an enterprise is a business or other organization such as a hospital, university, or government agency). The enterprise model shows the functional areas and processes that are necessary to run the business, and the links among these processes (box 1). The next step is to identify corporate data entities and link them to processes or activities (box 2). An entity can be something tangible, such as an employee, a part, or a customer, or it may be intangible, such as a profit center, a job title, or a purchase. Usually, the set of entities is represented as a chart of entities. Data requirements are mapped onto the enterprise model, leading to the identification of subject areas for which data bases need to be implemented (box 3).

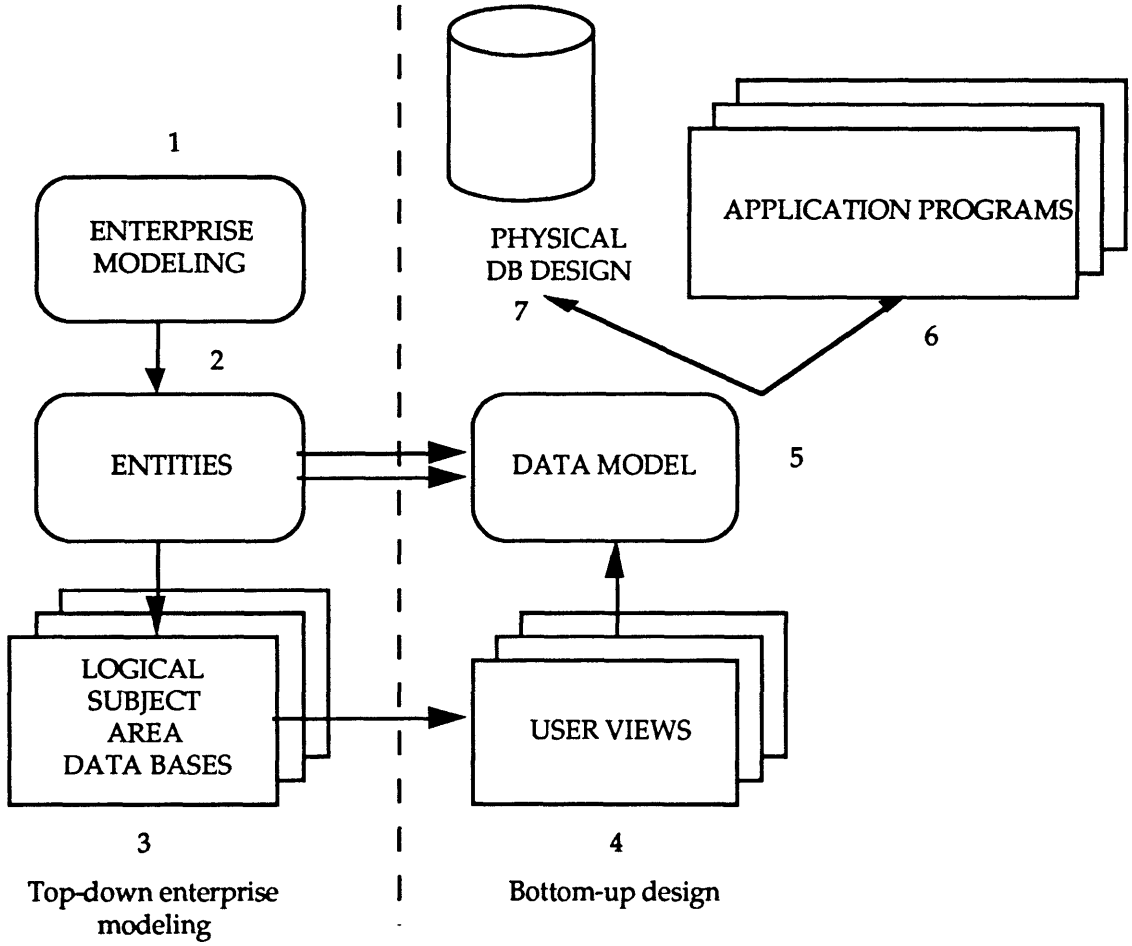

Figure 4.2: Enterprise modeling
(adapted from Martin's Strategic Data-Planning)

In most cases, only selected parts of the enterprise model and subject area databases are chosen for bottom-up design and implementation. The bottom-up design starts with a synthesis of detailed end user and management data views (box 4), and proceeds with the design of the data model (box 5). The data model is much more precise than the entities set, and eliminates some of the redundancies and unnecessary dependencies found in the entity chart. Finally, database design and subsequent design of application programs proceed from the data model design (boxes 6 & 7).

4.2.1.2 Enterprise Modeling Advantages

The methodology has several advantages. First, it provides an architectural foundation that leads to consistency of data and improved productivity in systems development and maintenance. Second, through the effort of modeling its information flows, the organization improves its understanding of the data and information needed to conduct the business. Third, the blueprint generated greatly facilitates the integration of data: when there is a need for inter-functional, intra-functional, team-based, or planning and control integration, the system can be put together easily because the blueprint has in theory eliminated the presence of all data incompatibility issues. The approach is particularly beneficial for organizations that need a high level of coordination.

4.2.1.3 Enterprise Modeling Disadvantages

On the other hand, the enterprise modelling approach has some disadvantages [Goodhue et al., 1987, pp. 15-19]. The enterprise modeling, done in detail and with a wide scope, can be very time consuming and expensive. Because there is a significant upfront effort needed, organizations face a longer and more expensive development process for the initial systems developed. IS managers, who have incentives to build systems quickly and cheaply, will resist the additional effort needed, especially if the organization has already invested a large amount in the existing IS base.

Since the approach attempts to coordinate the information needs on a management perspective across all units of the organization, some centralization of control is necessary. A decentralized organization may have some difficulty in providing this central control.

The methodology is based on the assumption that the planners know exactly the entire scope of the business. In the case of a large decentralized organization where virtually no one

has a global view of the whole set of activities, this assumption simply cannot be made. Building a model of the organization would require the cooperation of people from all units; and because these people have different motivations, conflicts about the definitions and uses of data are likely to arise.

The approach hampers autonomy because the data model is built in a top-down manner. Hence, some entity and data format choices pertaining to a subunit may not be in total agreement with what the subunit would have conceived independently.

Finally, the methodology does not recognize the ongoing nature of the organization's activities; thus, it is likely that the business will change while the plan is being developed and implemented. The resulting IS architecture will lack the flexibility necessary to quickly redeploy all available resources to take advantage of emerging opportunities.

4.2.1.4 Enterprise Modeling Summary

In summary, the Strategic Data Planning methodology results in the building of a corporate-wide data architecture which allows applications to easily integrate information; however, the architecture seems to lack the flexibility that an organization needs in order to be able to evolve with a constantly changing environment. In addition, the methodology is fraught with the following pitfalls: (i) it is time consuming and expensive; (ii) there is significant up front effort needed; (iii) it does not take into account the fact that the business is continuously changing; (iv) nor does it take into account that large investments have already been made in the existing systems; and finally (v) the payoffs are long term instead of immediate.

4.2.2 Data Resource Management

4.2.2.1 DRM Methodology

Whereas Enterprise Modeling addresses the problem of connectivity on a systems architecture perspective, DRM addresses it on a data perspective. The approach works from the organization's existing information systems infrastructure, and focuses on the management of data present in the various systems.

Organizations have attempted to manage their data to solve the inconsistencies that were created in the past and avoid them in the future. The management of data resources has been defined as "the establishment and enforcement of policies and procedures for managing the company's data as a corporate resource" [Kahn, 1983, p. 794].

In order to manage data, an organization typically creates a Data Administration (DA) group. The data administrator's goal is to develop policies, procedures, standards, and controls that are necessary for the management of the corporate data. DA's activities include implementing naming conventions, data definition standards, and logical and physical data base design standards and guidelines. The approach attempts to address the issues of accuracy, compatibility, security, redundancy, accessibility, and ownership. Each issue is described briefly below:

(1) *Accuracy* - inaccuracy stems from keying in the wrong value for a data element.

(2) *Compatibility* - James Martin explains that data can be incompatible in five ways:

(a) *field definition* - different parts of the organization do not agree on the definition and meaning of a field.
(b) *Field structure* - the same field is structured differently in different places (e.g., different length, binary vs. decimal, and so forth).
(c) *Record structure* - records with the same key are structured differently in different places.
(d) *Time of update* - data may be updated monthly, weekly, daily, or interactively in different systems, making different copies of data inconsistent at any point in time .
(5) *Inconsistency of update rules* - processing and update rules may be different for different copies of the data.

(3) *Security* - security issues include all of the ways in which data can be altered or destroyed.

(4) *Redundancy* - redundancy is created when the same data item is stored in several different computers.

(5) *Accessibility* - when an end-user desires to obtain some particular data, she needs to have a means for knowing that the data exists, what its definition is, and where it can be found. In

addition, the data must be technically accessible, and the user must have the appropriate authorization.

(6) *Ownership* - in a decentralized organization, sub-units may not readily share their data with other parts of the organization, feeling that outsiders may not have the understanding to interpret the data properly.

DRM usually involves the creation of a data dictionary [Ross, 1981, chapter 6]. A data dictionary is a tool for storing and updating information about the organization's data resources, including the definition, structure, and use of data. Typical information stored in a data dictionary includes standards for names, definitions, documentation, corporate owners of data, and names of departments sharing certain data items.

4.2.2.2 DRM Advantages

Data dictionary systems significantly improve operational control activities and data administration: control of systems maintenance and documentation, enforcement of systems and programming standards, control of the integrity and security of data, and assistance in systems design and development [Kahn and Lumsden, 1983, p. 30]. Documentation is improved by the use of data dictionaries since they include definition, owner, and physical location of data elements. Programming standards can also be enforced through the use of data dictionaries since they contains standards names and standard syntax for data descriptions. Data dictionaries assure data integrity by maintaining sound and secure descriptions of data. Finally, data dictionaries can assist in systems development by providing an easily accessible source of existing data descriptions. Corporate-wide establishment of standard data definitions is also ideal for the building of applications integrating data from disparate computer systems since it significantly reduces the number of logical connectivity issues.[8] In addition, a company-wide utilization of an on-line data dictionary by designers and developers will allow smooth evolution of the standards since any modification will be visible by all parties.

---

[8] See § 3.2.3 for a presentation of the different types of logical connectivity issues.

4.2.2.3 DRM Disadvantages

The task of introducing naming standards and resolving definition differences company-wide can be arduous and lengthy. Studies on the use of data dictionaries in organizations have illustrated this fact [Johnson, 1985, p. 31]. The main reason why more companies are not using data dictionaries is that the dictionary is made usable too slowly to justify the investment put into it. Because the task of analyzing and defining numerous data elements takes a long time, few data elements are available in the dictionary initially, and people are not interested in using it.

Properly managing a large dictionary is a tedious task too, and poor management will result in a low quality of data standardization due to a failure to detect and eliminate redundant definitions, and a failure to enforce the use of the dictionary on all projects.

Since DRM attempts to coordinate the definition of standards for all data elements of the company, as Johnson explains, some centralization of control is necessary. Indeed, the DA group which is created to coordinate all DRM activities is usually situated within the information systems group in the organization. The data administrator may well be a highly visible individual in the company, acting at the executive VP level, and given a separate budget. This paradigm of data administration is very likely to enter into conflict with a decentralized management philosophy. Studies have shown that a data administrator in a decentralized organization is weak, acting primarily through committee coordination, and having little enforcement power [Power, 1984, p. 108].

Data resource management is also the source of potential conflicts and political problems because it aims at setting company-wide data standards. Going back to the idea of sub-unit autonomy as being a strong force in a decentralized organization, company-wide standards may hinder innovation and entrepreneurship of organizational units that are used to carry out business independently from the rest of the company.

4.2.2.4 DRM Summary

Thus, DRM can greatly help an organization "clean" its IS infrastructure by standardizing the data and building easily accessible dictionaries containing the standard definition of each data element used by the organization. Company-wide data standards make the task of data integration significantly easier by eliminating all logical connectivity issues.

However, the task of establishing company-wide standards can be arduous and may conflict with a decentralized management culture.

4.3 REACTIVE APPROACHES

4.3.1 Manual Bridges

A reactive approach consists of letting each individual subunit define its data, design its databases, and acquire its own hardware and software to implement its databases. Each subunit can independently deploy the IT that it thinks is suitable for the local needs. Such an approach reflects the belief that no central organization can plan for the end-user computer of a whole company; each user is an individual with differing needs, and the sum of these needs is too big, too complex, and too diverse to be dealt with at the level of a central organization.

4.3.1.1 Methodology

Manual Bridges has virtually no integration methodology at all. Where there is a need for the integration of data, ad-hoc agreement are established among multiple subunits of the organization. Some subunits may have to acquire certain network interconnecting tools, and/or have to alter some of their data definitions if they accept to do so, and/or have to build bridges to physically and logically link with other computers. The integration approach is represented on Figure 4.3.

The term "manual" stems from the fact that a bridge may not be necessarily electronic. A bridge could be a tape (coined "floppy" bridge on Figure 4.4) which must be regularly downloaded into the computer, or a printed report generated by another computer (coined "sneaker" bridge on Figure 4.4) which must be rekeyed in the computer.

4.3.1.2 Manual Bridges Advantages

An ad hoc integration is much cheaper than a complete Enterprise Modelling endeavour, or a company-wide standardization of the data elements. Thus, building bridges may be worth considering for an organization where each unit has already heavily invested in IT equipment, and hence cannot afford to re-build or significantly modify its existing IS architecture for the sake of an immediate integration need.

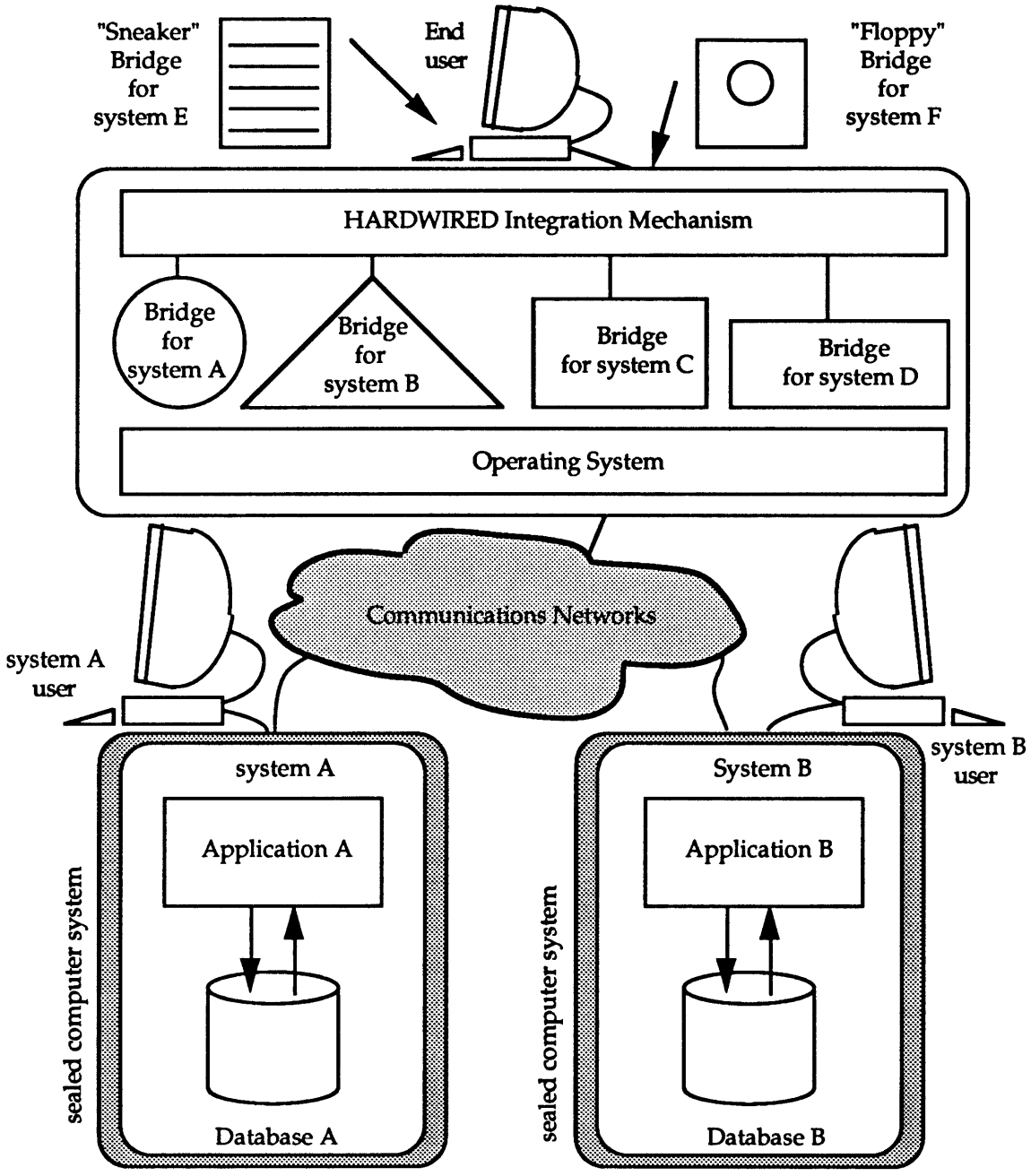

Figure 4.3: Manual Bridges Approach[9]

The approach can also help put an integration application together quickly. The capability of providing a quick solution to a strategic need is important. If the system shows

[9] Only systems A and B are represented; systems C and D are very likely to be of the same nature, i.e. sealed computers. Notice that systems E and F are not "hooked" on the corporate communications network, hence data is provided through printed reports, or at best tapes (or floppy disks).

satisfactory results, designers and developers will gain the upper management support necessary for the continuation of the project.

The approach has also the advantage of being non-intrusive. Each organizational unit not only keeps its existing IS base, but also can continue its regular operations while various bridges may be tapping in its databases as regular users [Madnick and Wang, 1988b, pp. 120-121].

The approach also respects the organization's culture, such as autonomy, because it does not alter the local processing of the computers that the CIS system accesses. Thus, ownership is not altered, not even put into question. A bridge accesses the computer systems as a regular user, hence a local unit is in the position of deciding what will be accessible to that "integration" user.

4.3.1.3 Manual Bridges Disadvantages

A bridge is fairly limited in its functions because it acts as a virtual driver. Such a technology of virtual driver is used because most of the computer systems accessed are sealed. By "sealed" we mean that the only access path to the data stored in the computer system is through the user interface with a terminal (providing, of course that the bridge is authorized to log in). Thus, through the use of virtual drivers, it will be difficult to access an outside database for purposes not supported through terminal command, and adding new functions and new types of data will be cumbersome. A virtual driver will also have difficulties handling large volumes of transactions.

The bridges are typically built one at a time. As we mentioned in chapter III, each bridge is designed to resolve the particular data incompatibility issues associated with the bridging to a given computer system. As a result, each bridge ends up being highly customized and optimized, but not usable for another purpose -- on Figure 4.3, we represented each bridge with a different shape to point out the uniqueness of its kind. Therefore, the approach may turn out to be more expensive than expected if new staff must be trained and new software must be written each time a new bridge must be built.

Finally, along with the lack of consistent methodology for building bridges, there is also a lack of a general-purpose data integration mechanism, which prevents the organization from creating economies of scale.

4.3.1.4 Manual Bridges Summary

In summary, building bridges that will satisfy specific connectivity requirements enables an organization to quickly put applications together, and at a moderate cost. As the number of bridges needed increases, however, the resulting IS architecture may become very complex and difficult to maintain, and to extend.

4.3.2 Composite Information System

4.3.2.1 CIS Methodology

The CIS methodology is based on the same principle as the previous approach: it builds upon the existing architecture and data element definitions. Compared with the Manual Bridges approach, however, CIS has two remarkable contributions which are:

(i) it provides a consistent architecture across the bridges; and
(ii) it provides a general-purpose data integration mechanism which constitutes a software foundation unto which applications can be built.

The approach is represented on Figure 4.4.

4.3.2.2 CIS Advantages

The CIS methodology has the same advantages as Manual Bridges, such as rapid application development, low cost, non-intrusiveness, and respect of subunits' autonomy.

CIS has also some advantages of its own. Because all the bridges used by the CIS system are built from a general-purpose architecture, it increases the ability of a CIS system to integrate data, and decreases time and resources necessary for the development of new bridges.

CIS also represents an evolving path. It does not ask explicitly for standardization of hardware, software, or data elements definitions, but uncovers incompatibility issues which may spark some attention. The organization may possibly realize that some standardization is necessary, and with the appearance of standards a CIS system could naturally evolve.

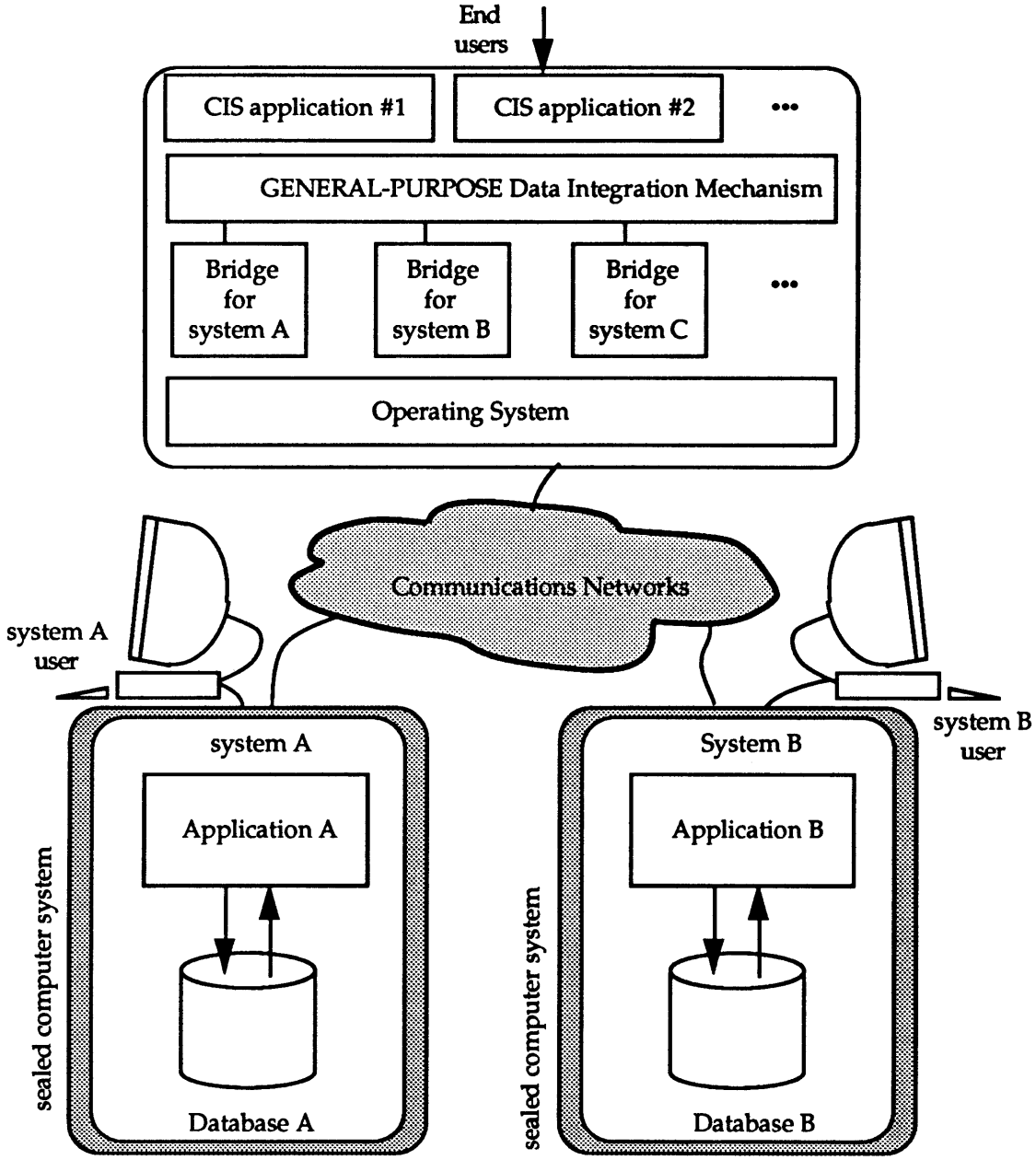

Figure 4.4: CIS methodology[10]

Finally, the CIS methodology is incremental by essence. When new computer systems must be accessed, the CIS system can quickly accommodate. Since most of the data incompatibilities are already being treated by existing software, the new system can be added to the CIS platform with a minimum of effort. Therefore, the organization can create economies of scale.

---

[10] Only systems A and B are represented; like Figure 4.4, the other systems are very likely to be sealed computers. Notice that if one system is not sealed, the bridge will be easier to built, and will have more interesting functionalities.

4.3.2.3 CIS Disadvantages

Since the bridges used by a CIS system remain based on the technology of virtual drivers, their performance may be severely limited (see § 4.3.1.3).

In addition, building a general-purpose architecture for a bridge is a particularly difficult task that very few organizations have begun to undertake. The conceptual difficulty stems from the fact that the architecture has to be able to accommodate a wide range of hardware and software environments, and a myriad of different data elements representations and interpretations. As a result, the field of semantics mapping is still in a research phase, and virtually no IT vendor has proposed yet such a technology. Thus, the success of a CIS system can be seriously jeopardized by the lack of a mature technology.

4.3.2.4 CIS Summary

In summary, CIS is the optimum approach of an organization if: (i) heavy investments have already been made in existing computer systems; (ii) company-wide standardization is not feasible; and (iii) the integration must be achieved as soon as possible. However, the resulting CIS system may be technically difficult to build, and for this reason may present some limitations, and may not provide all the functions it originally intended to.

4.4 ANALYSIS

4.4.1 Approaches Comparison

Because Enterprise Modeling completely re-engineers the organization's IS infrastructure, it represents an extreme solution. The methodology will obviously face major organizational resistance since many established rules and authority patterns will be affected [Keen, 1981].

DRM appears to be more moderate. It is similar to Enterprise Modeling in that it aims at establishing company-wide data standards; however, it starts from the existing IS infrastructure, and hence is more conciliatory than Enterprise Modelling. Unfortunately, the approach faces difficulties because it tries to establish standards on a large number of data elements. Whereas a centralized organization may be able to provide the appropriate level of

control to handle this broad standardization, a decentralized organization may have difficulties providing it.

A recent example was found at Citibank in its project of globalizing its various custody systems [Krishna, 1990]. In this endeavour, Citibank has chosen a DRM approach and aims at standardizing thousands of data elements. This approach will face two major difficulties: first, the process of gaining coalition among all the organizational units of the bank will be lengthy, and second, assuming that the bank does reach consensus on the definition of all its data elements, the manipulation and maintenance of such a large number of definitions will be particularly arduous. Krishna explains that the bank is currently relying on the use of CASE tools to manage the set of definitions. It is not clear, however, that CASE tools technology will be able to assure the whole set of functionalities that the bank has written down in its specifications.

A complete standardization of all data elements is not a viable solution anyhow in the context of a single organization maintaining multiple separate databases [Madnick, May 1989]. A database belonging to a group or division of a company is designed and evolves to help satisfy the objectives of that group or division. Any imposed standards inhibit evolution and serve as an indirect brake on business performance. Finally, another argument against the company-wide standardization in the DRM approach is the fact that more and more end-user applications are accessing external on-line service, whose data format and semantics are very likely not to coincide with the internal definitions, hence there will always be the necessity to deal with data incompatibilities.

On the other extreme of the spectrum, Manual Bridges decides to leave each unit intact and build "bridges" or virtual drivers among the systems in order to exchange data. As we explained in chapter III, such bridges have to perform complex semantic mapping between the unit's internal data representation and the rest of the organization's data representation. This approach is also extreme because it assumes no possibility of cooperation on the part of the organizational units and intends to fix the problem at the end of the pipe.

The CIS methodology is also based on the building of bridges, but goes one step further. It provides a general-purpose architecture for a bridge, and a software foundation as a data integration mechanism. The organization can thus create economies of scale. The problem that the CIS approach is still facing, however, is that the technologies used for both the bridges and the foundation are leading-edge technologies, and hence lack maturity.

Figure 4.5 summarizes the advantages and disadvantages of each connectivity approach.

4.4.2 Emerging Approach: Focused Standards

A new approach has recently emerged which combines the CIS and the DRM methodologies (see Figure 4.6). The approach consists of defining critical corporate data elements to be standardized throughout the organization in order to gradually shift the existing IS infrastructure toward a more standardized environment. Since it starts from the existing *status quo*, the approach has the advantage of being evolutionary like the CIS methodology; and since it is aimed at gradually changing the heterogeneous environment into a more standardized one, the approach also has the potential of reaping the benefits of company-wide standards, which is an advantage of the DRM methodology. This new approach has been called *Focused Standards* [Trice, 1987].

Enterprise Modelling	**Data Resource Management**	**Composite Information System**	**Manual Bridges**
Advantages	*Advantages*	*Advantages*	*Advantages*
• Provides data consistency • Greatly facilitates coordination • Eliminates all data incompatibilities • Improves mental model of the firm	• Provides data consistency • Improves coordination • Allows evolution • Is incremental	• Respects autonomy • Allows rapid integration • Allows evolution • Is incremental and cheap	• Respects autonomy • Allows rapid integration • Is relatively cheap

Disadvantages	*Disadvantages*	*Disadvantages*	*Disadvantages*
• Difficult to reach consensus • Time consuming • Very expensive • Does not allow any flexibility • Long-term pay-offs	• Difficult to reach consensus • Time consuming • Expensive • Difficult maintenance • Long-term pay-offs	• Technical difficulties • Difficult maintenance • Limited functionalities	• Technical difficulties • Very difficult maintenance • Limited functionalities • Allows little flexibility

Figure 4.5: Connectivity approaches advantages and disadvantages

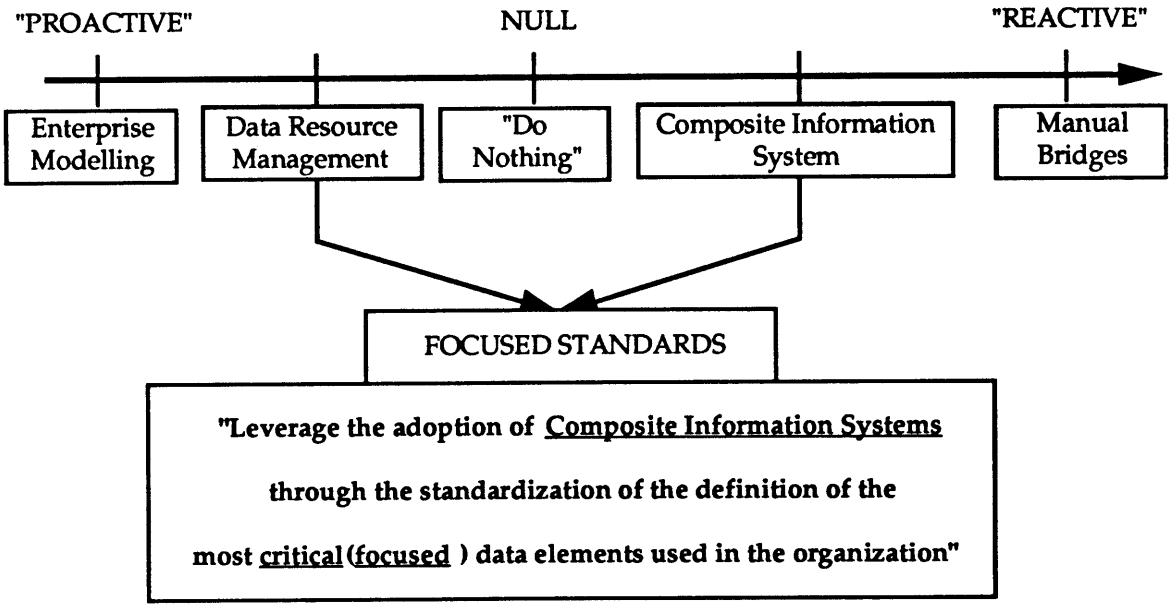

Figure 4.6: How the Focused Standards Emerged

We feel that Focused Standards is the best approach among all the connectivity approaches that were described in this chapter. Thus, the remainder of the thesis is devoted to the description of the methodology, how it can be implemented, and the technology involved.

-oOo-

Chapter V.

FOCUSED STANDARDS

In the previous chapter, we recommended the combination of the Data Resource Management and Composite Information System approaches for gaining connectivity. This combination was entitled Focused Standards because it was primarily based on the selection of a limited number of data elements to be standardized across all organizational units. Because Focused Standards involves the standardization and maintenance of the definition of data elements, it borrows some principles from DRM. The data integration approach in turn still relies on the installation of CIS systems. What Focused Standards gains compared with the CIS methodology used alone, however, is the increased ease in building bridges due to the introduction of standards on some of the organization's most critical data elements.

The Focused Standards approach has thus the following facets:

Policy - selection, and maintenance of the standard definitions;

Technology - deployment of CIS systems which implies:

(i) the development of a general-purpose data integration mechanism; and

(ii) the development of bridges to perform semantic mappings.

This chapter focuses on the "policy" facet. We present the methodology, its benefits, and the obstacles it must clear. Then, we spend some time on possible enforcement policies to deploy the focused standards, and finally propose the final policy recommendations. Chapters VI, VII, and VIII will focus on the development of bridges. We will not cover the development of a general-purpose data integration mechanism since it has been very well documented by Toon King Wong [Wong, 1989].

5.1 METHODOLOGY

The Focused Standards methodology consists of identifying the critical corporate data elements and standardizing their definitions throughout the corporation. Criterion that are used to decide whether a data element is crucial to the organization may vary from one company to the other, but we have recognized the following:
(1) *criticality* - a data element which describes groups of people, things, and rules that are fundamental to the organization's operations.
(2) *shareability* - a data element which is shared on a regular basis by several departments in the organization.
(3) *stability* - a data element whose value, name, representation, and interpretation do not change during operational processes.
(4) *read only* - a data element which is accessed in a read mode, as opposed to a write mode, for the majority of the accesses is standardized.
(5) *key and/or classifier* - a data element which is a key and/or classifier to major files.

The process of selecting focused standards among the multitude of data elements in an organization has sometimes been referred to as applying the "Critical Success Factors" (CSF) approach to standardization [Trice, 1987]. The CSF approach was designed to identify the few key areas where "things must go right" in order for the business to flourish [Rockart, 1979, p. 85]. The methodology leads top executives in an organization to define the critical data elements they need to conduct their business. Whereas the CSF approach focuses on management reporting data, the focused standards approach focuses on both management reporting and operational data.

It is not uncommon in an organization to manipulate thousands of data elements [Krishna, 1990]. Obviously, the standardization of such a large number of data elements will face difficulties. However, the order of magnitude for the number of data needed for management reporting at the headquarter level will usually be under 100 [Johnson, 1985].

The focused standards methodology is a mini-DRM approach since it standardizes a small amount of data, but it still needs mechanisms to define, enforce, and maintain the standards. Hence, the approach will have to articulate policies, procedures, standards, and controls that are necessary for the coordination of the standardization. The building of a data definitions dictionary is also part of the approach. This dictionary is more than a mere repository of the definitions for the chosen standards. If this were the case, considering the small number of data elements being standardized, a mere written copy distributed throughout the company would be more convenient and much cheaper. The data definitions dictionary also contains information about application programs, corporate terminology, users, and procedures.

5.2 BENEFITS

The following criterion are used to evaluate the Focused Standards approach:

(i) ability to integrate data;

(ii) fit to the decentralized management structure; and

(iii) degree of flexibility.

Any standardization, be it of hardware, software, or data definitions, greatly helps the integration of data across organizational units. Hence, the standardization of key data elements has an immediate impact on connectivity. In addition to the benefits of standardization per se, the fact that the standardized data elements were chosen on the basis of shareability reinforces their positive impact. Finally, most of the data which is integrated for the purpose of a strategic application correspond to key data elements, hence the standardization of these has also a positive impact on the building of strategic applications.

When we consider the second criterion set forth, we see that the approach builds upon the existing IS infrastructure of the organization, and is therefore likely to fit with the present company culture.

Finally, Focused Standards is a non-disruptive approach which aims at gradually moving the organization toward standardization. The approach encourages the organization to learn by doing. The experience gained from the standardization of a small number of data elements will serve as a basis for any future standardization of a larger number of data elements. This incremental process allows the organization to evolve with the information requirements.

5.3 METHODOLOGY REQUIREMENTS

The approach has to satisfy the following requirements:

1. choice of criterion to select the key data elements;
2. development of a technology to adjust the existing information systems to the new standards;
3. justification of the project to the top management; and
4. elaboration of a policy mechanism to enforce the standards.

5.3.1 Selection Criterion

We have mentioned in the previous section that in order for a data element to be standardized, it has to be identified as critical, shared, stable, read only, or key classifier. Apart from the shareability, all these criterion are difficult to judge. It is true that some techniques exist for identifying critical data elements in a organization, such as the CSF methodology, but they were developed for the information requirements at the level of the top executives in an organization. It is not obvious how these techniques can be used at the lowest level of an organization when examining operational data. Stability is also difficult to define: is a data element which is updated daily stable? Along the same line, is a data which is read only 98% of the time considered read only? The task of defining key data elements is further complicated by the fact that they are often moving targets.[11]

The nature of the personnel involved in the selection of the data elements to be standardized may vary from one company to another. The selection can be made by a top-down, a bottom-up, or committee-based approach. There is no recipe for a successful standards selection process; the appropriate approach depends upon the organization's management culture, and must be chosen carefully.

5.3.2 Technology Involved

One philosophy that the Focused Standards approach borrows from the CIS methodology is to build upon the existing IS infrastructure without disrupting it. Hence, the

---

11 This evolutionary character of the common data core in an organization was one of the main concerns of the company Salomon Brothers, Inc. that we interviewed.

standards are deployed while each unit continues its regular operations. This approach assumes that there exists a technology which allows an information system to adopt new data standards without having to take the system apart. In chapter III, we briefly presented the technology needed and pointed out that this technology is still being developed. Thus, one remaining issue in the focused standards approach is the fact that its success is highly contingent on some technical components. The last section of the thesis examines these technologies in depth and proposes an architecture fulfilling most of the requirements needed by the focused standards approach.

5.3.3 Cost justification

Since the approach is incremental by definition, the benefits may be realized slowly. This may create some difficulties when justifying the project to the organization's top management whose business practices usually favor quick returns. Top management commitment in the project is crucial for several reasons. First, since the approach intends to coordinate the standardization of some data elements used throughout the organization, top management support may significantly help the individual units cooperate. Second, data resources management in general tends to create conflicts and political tensions, which can only be resolved if top management has made its support of data administration clear. Finally, top management support is necessary to obtain sufficient resources in terms of dollars and people.

5.3.4 Enforcement Policies

The enforcement of standards in a decentralized organization is likely to meet with resistance. Independent units are often more concerned about having total control over their data than with standardizing key data elements throughout the firm. Data ownership is also a serious issue: the effort to standardize is likely to expose units' data. Departments will be reluctant to help standardize part of their data that, they claim, won't be understood by outsiders [Rockart and DeLong, 1988, chapter 10, p. 221]. Data ownership is an important part of an organization's culture. Like autonomy, data ownership motivates people and thus it is advisable not to alter it suddenly.

Hence, a successful policy enforcement mechanism will be one which gradually introduces new standards while maintaining the present patterns in which organizational units conduct their business. Different approaches may be adopted to enforce the standards:

authoritative, advocating, or self-interested. The nature of the policy that an organization should adopt depends upon its capacity to change.

5.4 SELECTION OF STANDARDS & ENFORCEMENT POLICIES

Roles and responsibilities must be established for two distinct phases in the process: the selection of the standards phase, and the enforcement phase. The two phases are not necessarily established sequentially. In fact, they are operated in parallel as new standards are proposed over time. Policies that are elaborated for the two phases are gathered under the focused standards policy.

A focused standards policy can be of three types: top-down; bottom-up; and committee-based. Each approach is discussed below.

5.4.1 Top Down

A "top-down" approach in data resource management consists of creating a single strong Data Administrator (DA) who takes on much of the responsibilities in data definitions, access control, data redundancy, performance management, data quality, and user coordination [Ross, 1981]. As mentioned in the previous chapter, the rationale is that having a central data resource management organization means coordinating better standardization across the departments. Ideally, Ross explains that data resources management should be a staff function at the top management level. This offers DRM the greatest leverage for achieving corporate data development objectives. This position provides a broad interdivisional perspective which facilitates a focus on information requirements. Senior staff has a broader view of the business, and hence is in the best position to define the most critical data elements.

(i) Selection of standards

The top-down approach in DRM and focused standards definition is a natural by-product of developing common systems. Once common systems have been developed, standard data definitions automatically exist. The data elements still have to be defined, but the definition process becomes part of the systems development process. The development of common systems facilitates access to the corporate data, which in turn drives accuracy since the data is visible to a large number of users and hence exposed to strong criticisms.

(ii) Enforcement policy

Enforcement responsibilities is given to the central DA. A Committee can be created to control the deployment of common systems and data standards [Rothschild, 1987]. As new systems are developed the committee reviews the design to ensure consistency with corporate standard data definitions. The high degree of centralization means that the committee contains most of the critical members of the information systems group. Because the members of the committee are not likely to be moved to a field location, a depth of experience is developed within the group. Top management support is inherent to the approach, and will function as a liaison with many different people across the organization.

(iii) Criticisms

In the context of a decentralized organization, the top-down approach is likely to face some difficulties. First, the philosophy of common systems is incompatible with a decentralized management culture, hence the organization will have to create the central DA group from scratch. Second, if the central DA imposes the corporate standards, individual units may complain that their business is different from what the DA perceived and did not fit the proposed standard definitions. Rothschild explains that this situation will require the intervention of an influential manager who knows the wheels of the organization, has a solid understanding of the technology, and is highly visible and regarded in the organization to negotiate the standards at the lowest levels.

Top-down does not build reciprocity among the various organizational units. Most of the time, one department will be expected to deliver data in a certain format, but will not see see what the benefits are for itself. It the reciprocity among the various groups involved is not obvious, then it will be unlikely that one group will go to the effort of making changes to support another group [Madnick, December 1989, and Osborn, 1987].

5.4.2 Bottom-Up

The "bottom-up" approach consists of letting the organizational units define the key data elements and having a large number of user groups discuss and reach an agreement. The rationale of the approach is that effective data management cannot be achieved just by the establishment of a central DA organization [Nolan, 1982]. The other organizations within the company need to assume supportive roles in order to make a transition into data resource management and focused standards.

(i) Selection of standards

Each organizational unit proposes data elements to standardize, along with their definitions. Definition sessions would be organized in order to review all the proposals and try to reach a consensus on a core of common data.

(ii) Enforcement policy

A bottom-up approach advocates only the deployment of data standards. However, connectivity can still be achieved in the organization with the presence of a contract between a sender and a receiver of a stream of data [Rothschild, 1987]. Among other things, the contract can stipulate: responsible parties, level of service, distribution media, frequency of data submission, approved data values, size of the data, reporting format, and security precautions. The contract does not have to comply to the standard definitions, hence the two parties concerned may keep the *status quo*, as long as connectivity is established.

(iii) Criticisms

The apparent problem with this approach is the inherent difficulty of reaching a consensus among a large number of personnel whose views are very likely to differ. Some companies have adopted this approach because they believed that everyone had a strong incentive to reach an agreement on data standards. But, as one company we interviewed pointed out:

> "No one is passionate about how long a customer number should be. People aren't passionate about what a network protocol should be. All they want is for someone to deliver effective network services."[12]

The enforcement of data standards according to a bottom-up approach will be difficult. Without some central control, the standardization effort is probably doomed to failure. Hence, even though the definition process may reach consensus in a bottom-up manner, the organization will still need to create some sort of central body for controlling compliance with standards. This leads us to the study of a third approach as a middle-course between the top-down and bottom-up approaches.

---

[12] Interview with Marc Sternfeld, managing director of the information technology division at Solomon Brothers Inc., New York, 13 November 1989.

5.4.3 Committee-Based

An organization can choose a middle course approach between top-down and bottom-up consisting of forming a committee which gathers representatives from the main user groups. This committee will be in charge of defining the company's focused standards and elaborating policies to enforce them. The committee-based approach combines the advantages of the previous two: it coordinates the standardization effort from a central place, but still heavily relies on the organizational units' participation to define the data elements.

(i) Selection of Standards

This committee can be formed as a task force first [Johnson, 1985]. Because each sub-unit is in the best position to identify the key data elements pertaining to its own activities, the task force includes representatives from the major sub-units in the organization. The committee must maintain a balance between technical and business members in order to assure the fit between business goals and information systems requirements. Once the task force has started the process and tried out different approaches, a more permanent Data Administration function can be established. In a manner similar to the bottom-up approach, the definition process is gradual, whereby the first standards are created by the task force. Then, after having learned by doing, the task force can formalize the process and elaborate some policy guidelines.

Like the bottom-up approach, the committee lets data elements "bubble up" from the lowest operational levels in the organization. For example, a procedure used in one company consisted of asking each line information systems group within the sub-unit to propose data elements candidates [Johnson, 1985]. In that company, each IS group could define the top 50 data elements for which they wanted to see corporate-wide definitions. Then, it was up to the committee to decide which data elements would be standardized. Hence, the final reviewing and decision making was done centrally.

In its project to integrate various custody systems, Citibank has adopted a committee-based approach [Krishna, 1990]. Through this project, the bank aims at building a global financial system (GFS) that will allow any customer to view the collection of all the securities that she holds worldwide under the "umbrella of a single account." One component of the project involves the management of the bank's data, and in particular the standardization of the definitions of a large number of data elements. Even though the approach is not purely Focused Standards since it aims at standardizing a large number of data elements, it is still

worthwhile studying the mechanisms of standards selection and maintenance that the bank has put in place.

This committee is formed with a global data administrator, regional data administrators, and executives, analysts, and developers from the Technology Group based in London. The position of global data administrator (GDA) was created because the bank felt that some level of central control was necessary for a project requiring the participation of all organizational units. In addition to the GDA, the positions of Regional Data Administrators (RDA) were established as well. As Krishna explains, the bank felt that the success of a corporate wide data administration relied on a good coordination between the headquarter in New York (i.e., the GDA), and the various sub-units (i.e., the RDAs).

These RDAs play the same role as what Johnson describes as being mini-task forces. A mini-task is formed for each sub-unit, and is in charge of identifying and developing standard definitions for the data elements pertaining to its sub-unit. It proposes data element candidates for standardization to the main task force, whose role is to coordinate the selection and publication of the final data elements. The interaction of the GDA with the RDAs at Citibank is of the same nature: the GDA controls the corporate data dictionary which is populated by definitions provided by the RDAs. The GDA makes sure, for example, that there is no duplicate, and that no other data element already entered in the corporate dictionary plays the same function as the data element under consideration.

As Krishna points out, the knowledge on data elements and how to standardize them really spring from the operational levels at Citibank. He reports an amendment to the Control Procedures in the January 1990 Data Administrators meeting held in New York specifying: "RDAs are CONSULTED, i.e. not advised."

Another characteristic of the committee-based approach, as Johnson explains, is the principle of controllership, whereby each data element is assigned a controller. A controller is only responsible for the definition of the data element; she is not the owner who in turn is responsible for controlling data access and security. A controller is chosen based on her knowledge of the business in which the data element is used; a controller is typically a key manager of the sub-unit representing the source of the data element.

The controller concept fits well with a decentralized management structure. Indeed, the decision-making authority in the definition process is pushed down to the business area

which is the source of a data element. Users who disagree with the definition of a data element resolve the conflict with the controller, not with the committee. In a decentralized structure, the committee may be geographically distant from the users, which would make the committee-to-users interaction difficult if there was no local controller.

(ii) Enforcement policy

Enforcement policies encountered in the case of the committee-based approach were a compromise between an authoritative and a contract-like approach. In one company, for example, data standards were enforced by placing the burden of implementation at the "interface" level of the supplier and receiver of a stream of data [Johnson, 1987]. This meant that a receiver of a data flow could request that the flow conform to the standard definition. The data stream sender, upon receival of the request for standards, had a finite number of days to produce a plan for converting the data into its standard form. In the reverse direction, the data sender was encouraged to provide data complying with the standard definitions. Upon receival of the sender's announcement to provide data in its standard form, the receiver also had a finite number of days to confirm its readiness to receive the data in its standard form as scheduled, or to propose a different plan.

This approach among the parties exchanging data has been termed "self-interested." Indeed, since the existence of standards greatly facilitates systems development, new systems are very likely to conform to these standards. Thus, standards will be gradually adopted as new systems are built. In regard to the already existing systems, however, a receiver and a sender are allowed by the policy to draft a contract which maintains the *status quo*. But, as the number of contracts increases for a single data provider, this latter is likely to adopt the standards in order to eliminate this administrative burden of writing a different contract each time around.

(iii) Criticisms

Finding a controller for each data element may be problematic. People in the line areas may not see the benefits that standard definitions will provide, and they may be unwilling to invest the time required to develop the definition.

It also takes a unique kind of person to be able to work well in a task force, and to deal with data across multiple functions. The desired profile is someone who understands the business first and the technology second, is influential and respected, and knows the wheels of

the organization. In the case of Citibank, for example, it seems that such a person was typically represented by the global data administrator.

As I mentioned earlier, the issue of ownership can be a major obstacle. There is a direct relationship between ownership of information and autonomy [Keen, 1981]. Thus, a focused standards policy must carefully resolve the issue of ownership in order to maintain autonomy, but at the same time allow access to the data. Even as a controller is designated for each data element, so can an owner be designated. The owner is responsible for the acquisition, updating, and accuracy of the physical data. The owner also manages the accounts on its computer systems so that she can control who accesses its data.

5.5 RECOMMENDATIONS

The approach based on a committee setting is most appropriate for a decentralized structure. We propose to recapitulate the main points of the approach as the set of recommendations that this thesis aims to provide.

Goal - the focused standards methodology aims at establishing procedures and schedules to ensure that complete, accurate, and timely data is distributed to users.

Principle - the methodology consists of selecting the critical corporate data elements and standardizing their definitions. Shareability and business criticality should be the two main criterion for selecting the data elements to be standardized.

Mechanics - the methodology's mechanics include the selection, enforcement, and maintenance of the standards. We recommend a committee-based approach which combines the characteristics of two approaches: top-down and bottom-up.

Key Data Task Force - we recommend that a key data task force be created to begin the process of defining data standards. The task force will move into a permanent Data Administration function once the process is well understood.

A mini-task force can be established at the level of each organizational unit so that data elements standards "bubble up." Data elements can be defined at the level of each unit, then at the level of each division, and finally at the headquarter level.

Data Standard Approval - we recommend that the process of standard approval be implemented in four steps: initiation, draft development, review, and approval. A standard can be proposed by anyone in the company, but must be sponsored by one member of the task force (or later, the Data Administration board). The sponsor develops a draft of the standard with the initiator and a working group. The draft is sent to the board members some time before a voting meeting. A simple majority of "yes" with no vetoes carries a standard to approval.

Data standard definition - the standard definition of a data element must be independent of an individual user's point of view, valid anywhere in the corporation, expressed in a suitable syntax, and complete and unique

For the syntax used in the formal writing of a standard definition, we recommend the use of a coding system based on metadata techniques. We believe that the use of metadata properties for a data element will ensure a uniform level of quality and completeness in definitions. There is no limitation on the number of properties a definition must contain, but the definition of a data element should at least include: "business name", "controller", "description", "internal representation", and "display format". A vocabulary of specific terms must be established for the fields' values. With such agreed upon fields and vocabulary of terms, members of the data force know exactly what information they need to obtain when writing a draft to propose a candidate.

We also recommend the use of a central dictionary to store, update, and access the standard data definitions. A local data dictionary should also be created at the level of each organizational unit. In doing so, local system developers will have quicker and easier access to the definitions. Along with the creation of the local data dictionary, we strongly recommend the elaboration of procedures for periodic synchronization checks between the central data dictionary and the local ones [Krishna, 1990, p. 82].

Controllership & Ownership - we recommend that a controller be designated for each data element. The controller is in charge of achieving a standard definition for the data element. She also serves as intermediary between a receiver and a sender of the data element who are setting up a protocol of exchange. A data element owner is also designated to be in charge of controlling authorized use of the element.

Enforcement Policy - we recommend an enforcement policy which places the burden of implementation at the interface level between the sender and receiver of the data element.

Upon publication of a standard definition, the sender/receiver is entitled to send/receive the new standardized data element in its new format. The two parties have a finite number of days to agree upon a protocol. We also recommend that new systems conform with the data standards.

Users are held responsible for analyzing differences between local definitions and standard definitions, and for resolving these differences; they can either convert existing applications, or develop new ones, or build bridges (i.e., semantics mapping tool). Four situations can thus arise:

Solution 1 - the sender builds a bridge to its system;
Solution 2 - the receiver builds a bridge to its system;
Solution 3 - both the receiver and sender build bridges; and
Solution 4 - the receiver and sender maintain the status quo.

Figure 5.1 on the next page represents the four alternatives.

Data Administration Organization - we recommend that the task force move into a permanent Data Administration function. This organization should act as a supervisor of the standardization effort, and as a point of contact for users in the organization that want access to data.

We suggest that the Data Administration organization be of the matrix type, so that it can report to the main organizational units. Figure 5.2 next page illustrates this type of DA organization.

The organization is made of a board, policy-making, and technical assistance groups. The board decides which data elements should be standardized. The policy-making group focuses on the elaboration of enforcement policy mechanisms. Finally, the technical assistance group provides support in the design and implementation of semantics mapping tools to the organizational units that have decided to install bridges.

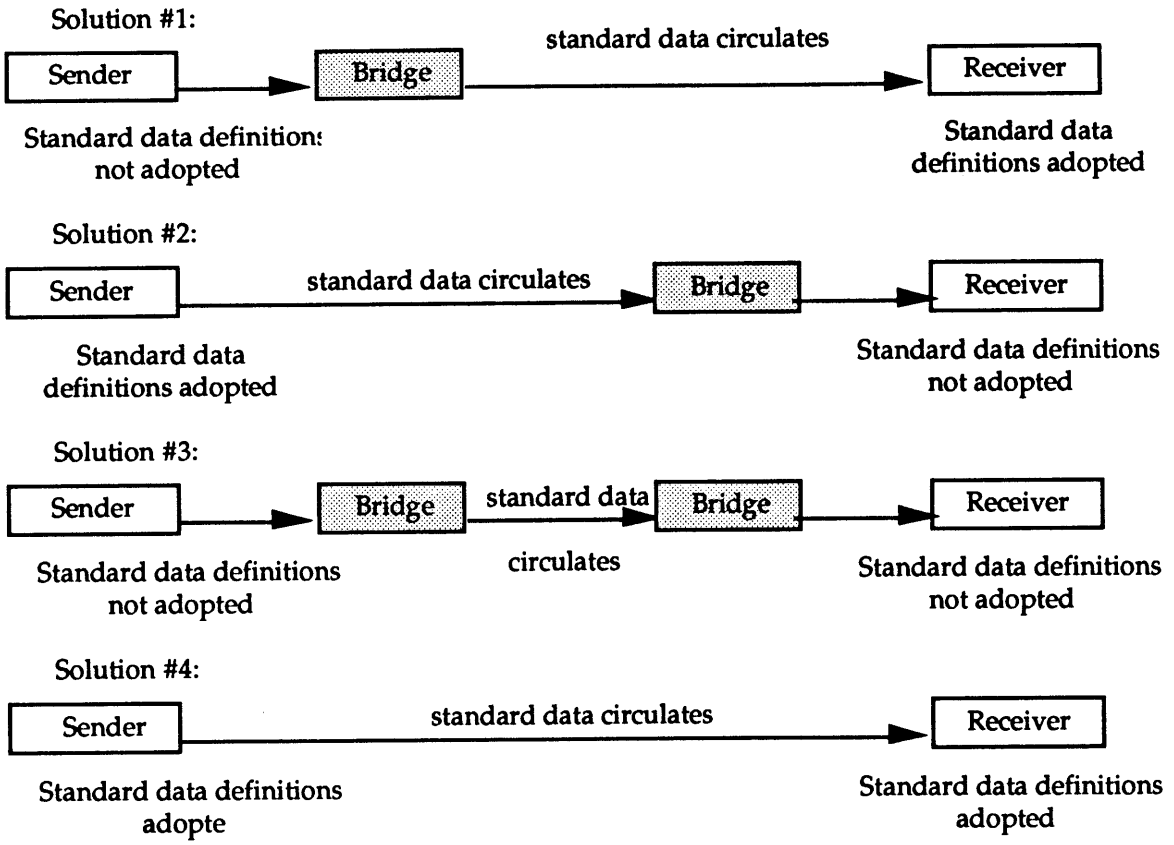

Figure 5.1: Sender-to-receiver protocol

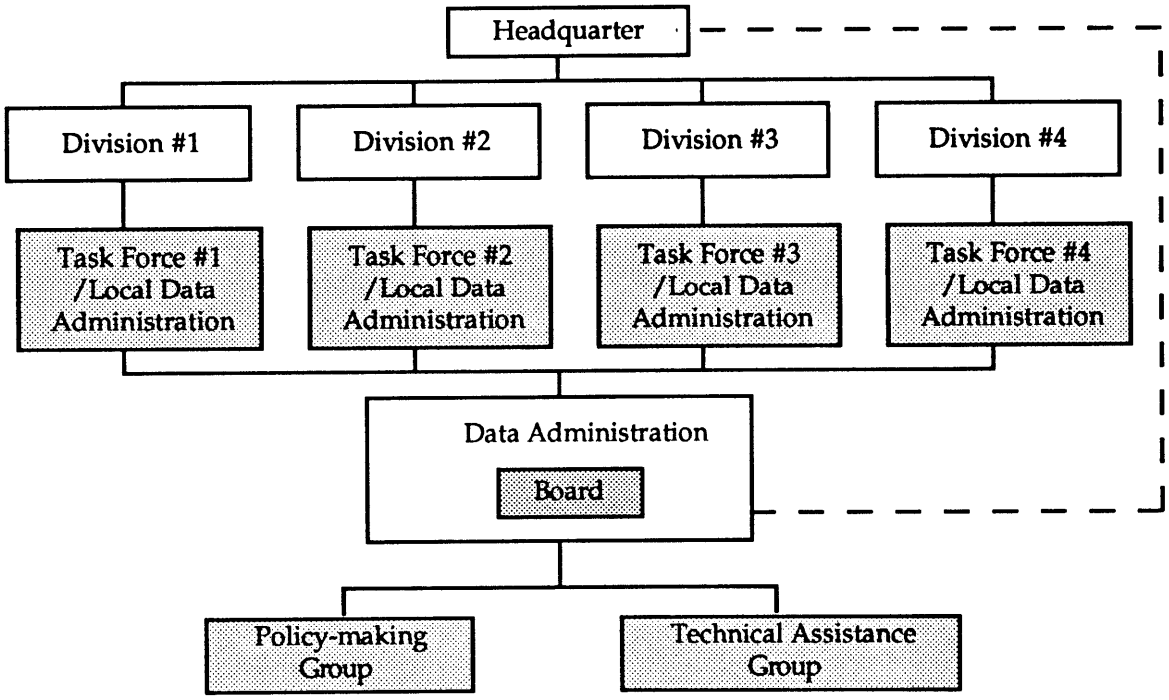

Figure 5.2: Data administration organization

5.6 SUMMARY

The focused standards methodology combines the advantages of Data Resource Management and Composite Information System approaches. Like DRM, it focuses on the management of data, and like CIS, it builds upon the existing organization's IS infrastructure.

Its contribution lies in the selection of key corporate data elements that are to be shared across the organization, and in the definition of clear roles and responsibilities in the deployment of data standards. The benefits are an improved delivery and access mechanism and improved communication of important information.

The methodology reconciles the conflicting forces of integration, autonomy, and evolution. It enables integration when needed because the key corporate data elements that are likely to be part of the integrations are being standardized. It enables autonomy by the nature of its data standards selection process and enforcement policies. Finally, the methodology enables evolution because the process of data standards selection is continuous and responds to different information requirements over time.

The focused standards methodology relies upon the availability of powerful semantics mapping tools. As we have seen in chapter III, such a tool is technically difficult to build, especially if data semantics change over time. Thus, a sub-unit should consider the installation of the semantics mapping tool as a short-term solution which allows the continuation of its activities, but should comply to all focused standards in the long-term.

Figure 5.3 on the next page summarizes our recommendations.

At this point, we have elaborated a policy for implementing the focused standards methodology. What is left is the design of a semantics mapping tool, or bridge, which constitutes a key aspect of the focused standards methodology. The three chapters which follow present the design of a semantics mapping tool. Chapter VI describes the functional requirements that the tool must satisfy. Chapter VII presents the architecture, and finally chapter VIII describes the main algorithms used by the tool.

-oOo-

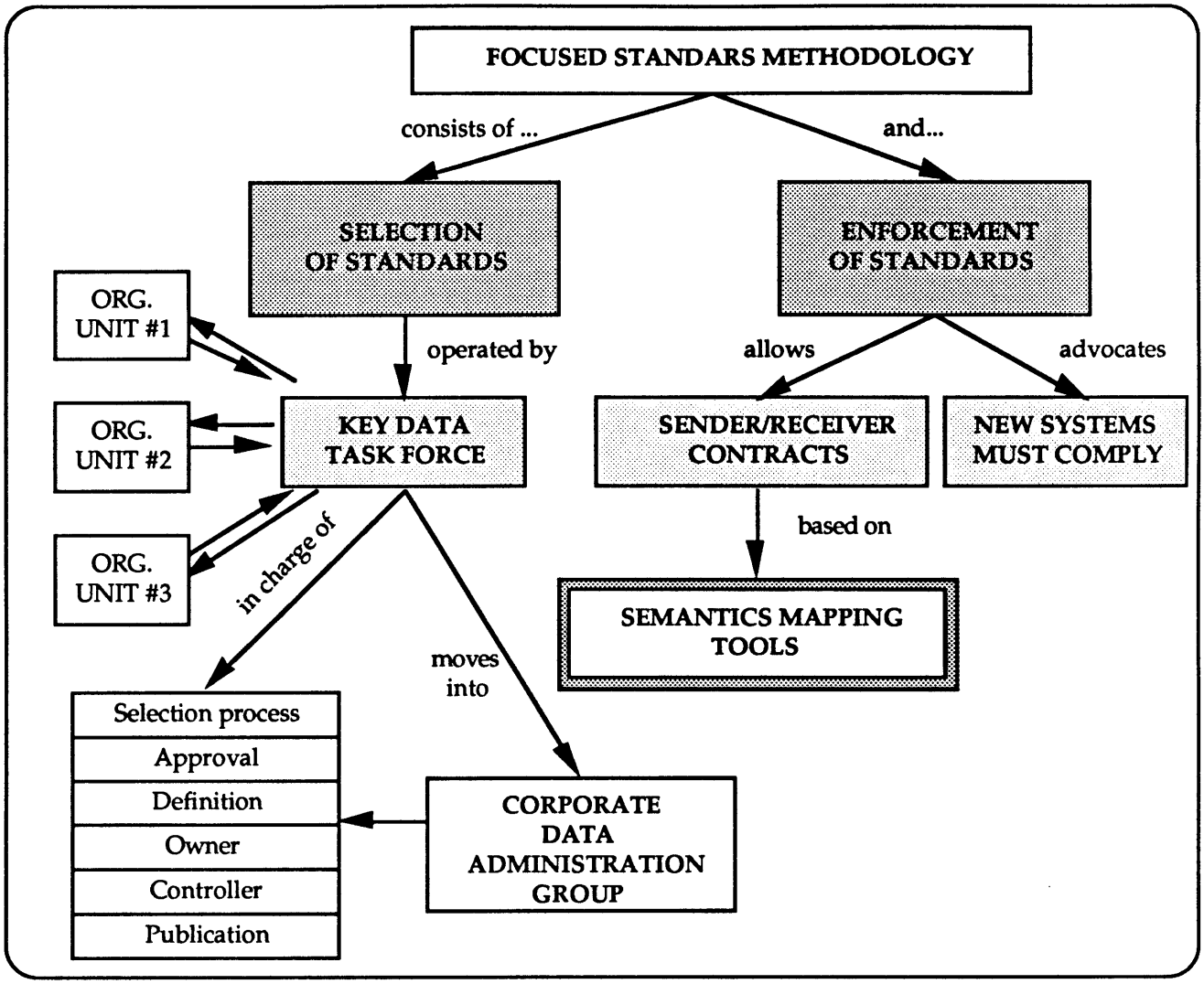

Figure 5.3: Recommendations

Chapter VI.

SEMANTICS MAPPING TOOL: FUNCTIONAL REQUIREMENTS

The Focused Standards policy described in the previous chapter encourages organizational units to use semantics mapping tools so that their various computer systems can send and receive the focused data in their standard definitions. In doing so, each organizational unit can still continue to run its application whose assumptions in terms of data definitions are not standard, and in the mean time exchange data in its standardized format with the exterior.

This chapter presents the functional requirements that a semantics mapping tool should satisfy in order to provide an acceptable level of semantic connectivity (see § 3.2.3 for a definition of semantic connectivity). The chapter has four parts. The first part presents the basic question that we are asking, and states the functionalities that the tool should satisfy. The second part examines in detail the different operations that must be carried out in order to satisfy an application's request. The third part reviews the existing literature in the domain of data semantics reconciliation. And finally, the fourth part lays out the strategy that we will adopt in the design.

6.1 FUNCTIONAL REQUIREMENTS

6.1.1 Questions Asked

We shall examine the situation where an application accesses data stored in a database (see Figure 6.1) -- which may reside on the same or a different computer from the application's -- and where the application's conventions in terms of data naming, representation, and interpretation differ from the database's. On figure 6.1, for example, the database manipulates financial figures in units of U.S. dollars only, whereas the database has its information stored in thousands of Japanese yen.

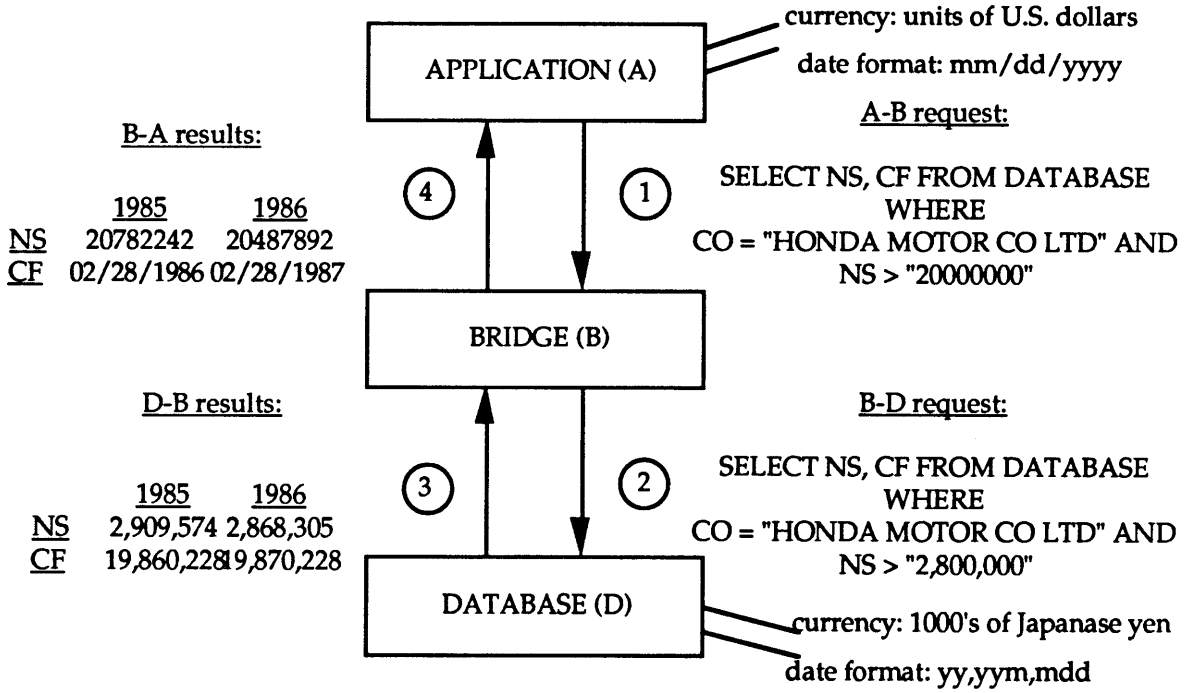

Figure 6.1: Application-to-database data conversion and vice versa

In order for the application to obtain data from the database, two essential steps must be performed:

Step 1 - the request sent by the application must be converted into the database's data naming, representation, and interpretation conventions (from (1) to (2) on Figure 6.1); and

Step 2 - the data obtained from the database must be converted into the application's data naming, representation, and interpretation conventions (from (3) to (4) on Figure 6.1).

In the remainder of the discussion, we will use the terms bridge or semantics mapping tool to designate the module which performs steps #1 and #2.

On figure 6.1, the first step converts 20 million U.S. dollars present in the A-B request into two billion and eight hundred million Japanese yen. The second step converts, for example, the "NS" data element "2,909,574" into "20782242".

The question that we are asking is the following:

"How can we carry out data conversion in a systematic way between an application and a database whose respective data naming, representation, and interpretation conventions differ?"

As we mentioned in chapter III, organizations have traditionally been dealing with this question by building highly customized bridges. Using this notion of bridge, our question is also equivalent to asking:

"how can we build a general-purpose data semantics bridge, or mapping tool?"

6.1.2 Design Goal

In chapter III we described the semantics of a data element as being the combination of three characteristics:
(i) the name of the data element;
(ii) the representation of the data element; and
(iii) the interpretation of the data element.

In this thesis, we will build a general-purpose semantics mapping tool to convert the representation of a data element.[13]

---

[13] For the conversion of data elements' names, see: Wong, T. K. "Data Connectivity in the Composite Information System/Tool Kit." B.S. Thesis, Massachusetts Institute of Technology (June 1989).
For the conversion of the interpretation of a data element, see: Siegel, M., and Madnick, S. (November 1989), "Identification and Reconciliation of Semantic Conflicts Using Metadata." CISL Working Paper No. 89-09, Sloan School Working Paper No. 3102-89 MSA.

6.1.3 Terminology

In the remainder of the thesis, we will use the following terminology (see Figure 6.1):

- the A-B request is the request sent by the application to the database whose data semantics are based on the application's conventions.
- the B-D request is the request sent by the bridge to the database whose data semantics are based on the database's conventions.
- the D-B results correspond to the data obtained from the database whose semantics are based on the database's conventions.
- the B-A results correspond to the data obtained from the database which has been converted into the application's data semantics conventions.
- we will use the terms attribute as well as data element to designate a data item.

6.1.4 Functionalities of a Data Semantics Mapping Tool

The data semantics mapping tool that we will design will support the following two functionalities:

(1) provide data representation transparency;
(2) allow the application to use selection operators in its requests which are not supported by the database (selection operators transparency).

Data representation transparency implies that the application can formulate requests based on its own data representation conventions without having to worry about whether or not the database will understand these conventions, and if not, how the conversion of data representation is operated.

Selection operators transparency implies that the application can formulate requests based on selection operators which are either supported or not supported by the database.

We shall now examine in more detail the A-B request given in Figure 6.1 in order to identify the main operations that must be carried out by the bridge.

6.2 SCENARIO ANALYSIS

6.2.1 Scenario Presentation

In this scenario, the database is the on-line financial service I.P. Sharp Disclosure introduced in chapter III. The application assumes that all financial information are expressed in U.S. dollars, whereas the database Disclosure has financial information expressed in different currencies depending upon the nationality of the company considered. In addition, the application has chosen to manipulate dates with the format *mm/dd/yyyy*, whereas the database's dates are float numbers (i.e., *yy,yym,mdd*) like any other financial figure.

The A-B request examined is the following:

```
SELECT ns, cf FROM disclosure
WHERE co = "HONDA MOTOR CO" and ns > "20000000"
```

This request asks for the financial reporting dates (called "CF" in the database) at which Honda Motor Co has net sales (called "NS" in the database) greater than twenty million U.S. dollars. Notice that the attribute names present in the request (e.g., NS, CF, and CO) are the names used by the database, hence it is assumed that attributes names mapping occurred in a previous step.

The request is given in the Structured Query Language (SQL) syntax. The syntax does not matter in our discussion; it is assumed that there exists a query syntax translator as part of the bridge's tool box. Regarding the terminology that will be used, a projection list corresponds to the information asked (e.g., NS and CF in the above request), and a constraint corresponds to the selection criterion (e.g., CO = "HONDA MOTOR CO" and NS > "20000000" in the above request).

6.2.2 Analysis

Chapter III introduced the term "data representation" and illustrated what the representation of a data element can be. Based on the examination of the on-line financial databases I.P. Sharp and Finsbury, we saw that a data element representation can be defined by the combination of the three characteristics (or fields): format, unit, and scale. However, this conception of data representation is somewhat empirical, thus we ought to ask ourselves the following questions:

"What makes up the representation of a data element? Can we build a representation framework that will be valid for all the data elements manipulated?"

The mere attempt to state in English what the A-B request is asking says a lot more than the request itself. The application assumes, for example, that the money amount "20000000" present in the constraint is in units of U.S. dollars. Since the information on data representation is not included in the query, we ask the following question:

"How does the application tell the bridge about the intended data representation used in the A-B request?"

Depending upon the currency of Honda Motor Co. in Disclosure, the money amount "20000000" may have to be converted. This brings up the following question:

"How does the bridge know the database's data representation choices?"

Let's proceed with the example and assume that the bridge has found out that financial information for Honda Motor Co. is provided in thousands of Japanese yen. Now, a currency conversion requires a date, and an exchange market at which the exchange rate will be obtained for the given date. Hence, initial and final currencies, a date, and an exchange market constitute the necessary arguments of a currency conversion. This brings up the following questions:

"What are the arguments of a given data representation conversion? How can the bridge know which arguments are needed?

"How can the values for these arguments be obtained at the time the conversion must be carried out (either during the A-B to B-D request conversion, or during the D-B to B-A results conversion)?"

Currency conversions can be operated in different ways. A first way consists of converting all data elements with today's (or most recent) exchange rates. This way of operating may appear very simplistic to some financial analysts. Depending on the purpose, a more proper way to adopt consists of converting each data element with the exchange rate pertaining to the date at which the data element was created. Converting a yearly net sales, for example, would thus require to break down the figure in the series of individual sales amounts that were made during the year. Then, each individual sales amount would be converted with the exchange rate obtained at the date the individual sales occurred. Unfortunately, this approach is likely to face some practical difficulties because the individual sales amount are not provided in most financial reports. A third way of performing

currency conversions that we will explore in our design consists of using the exchange rate obtained the day the financial figure was published. This last way of operating has the advantages of being practically feasible, since financial information is always provided with the financial reporting date, and of being relatively rigorous in its principle.

Let's proceed with the scenario, and adopt for the sake of simplicity the first way of performing currency conversions that we described above. Thus, let us imagine that we have picked up the spot price on the New York exchange market at today's date for the Japanese Yen and we obtained 0.7.[14] The A-B request can be converted into the B-D request as shown below:

```
SELECT ns, cf FROM disclosure
WHERE co = "HONDA MOTOR CO" and ns > "2,800,000"
```

Notice that several things occurred during the conversion of the data element "20000000" into "2,800,000": first, the format of the figure was converted (the new format has commas to mark thousands); second, the scale was converted (the new scale is in 1000's); and third, the currency was converted (the new currency is Yen).

The next question that we would like to ask concerns the ability of the remote database to support the operators used in the request. In our example, the predicate (> ns "20000000") contains the operator ">" that is not supported by all database systems. This brings up the following question:

"How can the bridge allow the application to use operators in the A-B request that are not supported by the database?"

Let's proceed with the example and assume that the Disclosure database supports the operators used in the B-D request. In that case, the B-D query can finally be converted into the Disclosure query language syntax and given to the database system.[15] The database thus receives the B-D request which is based on its own data representation choices. Notice that if we had not converted the figure "20000000" into its equivalent amount in yen, the database

---

[14] i.e., 1 ¥ = 0.7 U.S. cents (the I.P. Sharp Currency database provides all exchange rates in U.S. cents).

[15] Notice that syntax conversion does not entail data representation conversion; in Disclosure's syntax, our request becomes:
DISPLAY '(CO EQ HONDA MOTOR CO) AND (NS GT 20000000)' ADISCLOSURE 'CO, NI, CF'

would have interpreted it as being 20 billion yen instead of 2 billion and 8 hundred million yen! The following D-B results are obtained:

	1985	*1986*	
NS	*2,909,574*	*2,868,305*	<== *1000's of Yen*
CF	*19,860,228*	*19,870,228*	<== *float number notation*

Notice that all financial figures are in 1000's of yen, and the dates are expressed as float numbers. Because the application assumes that all financial figures are in U.S. dollars, the D-B results must be converted in units of U.S. dollars. The same way we operated for the A-B to B-D request conversion, today's exchange rate can be used to convert the D-B results. The dates have also to be converted from a float number notation to the application notation, i.e. "mm/dd/yyyy". Finally, the B-A results are:

	1985	*1986*	
NS	*20782242*	*20487892*	<== *units of U.S. dollars*
CF	*02/28/1986*	*02/28/1987*	<== *mm/dd/yyyy notation*

6.2.3 Recapitulation

In order to implement the two-step cycle composed of A-B to B-D request conversion and D-B to B-A results conversion, the following questions must be answered:

Question #1 - "What makes up the representation of a data element?
Question #2 - "What are the arguments of a given data representation conversion? How can the bridge know which arguments are needed?
Question #3 - "How does the application tell the bridge about the intended data representation used in the A-B request? How does the bridge know the database's data representation conventions?"
Question #4 - "How can the values for these arguments be obtained at the time the conversion must be carried out?"
Question #5 - "How can the bridge allow the application to use operators in the A-B request that are not supported by the database?"

We will address these questions in the next two chapters.

6.3 LITERATURE REVIEW

We shall now review the existing literature in the domain of data semantics reconciliation. This section studies two important concepts: the use of metadata to define data, and the use of a data catalog to store metadata.

6.3.1 Metadata

In the domain of data definition dictionaries, Symons and Tijmas have developed a systematic method for defining data elements using the metadata concept (Symons and Tijmas, 1982). The method consists of a data type classification system for data elements, syntax rules for the structure and completeness of formal definitions, and the use of a controlled vocabulary of permitted terms for formal definitions.

Data elements are divided into *type-classes*. Elements are classified as belonging to qualitative or quantitative data. For each class, there are certain standard aspects of the data which must be described, and for each aspect, there are a limited number of terms which are needed to represent the aspect. Thus, for each type-class, there is a List of Permitted Aspect-Terms, or LOPA terms. A formal definition is constructed by taking the one appropriate aspect-term for each relevant aspect from the LOPA, and stringing them together. Figure 6.2 represents Symons and Tijmas' data taxonomy.

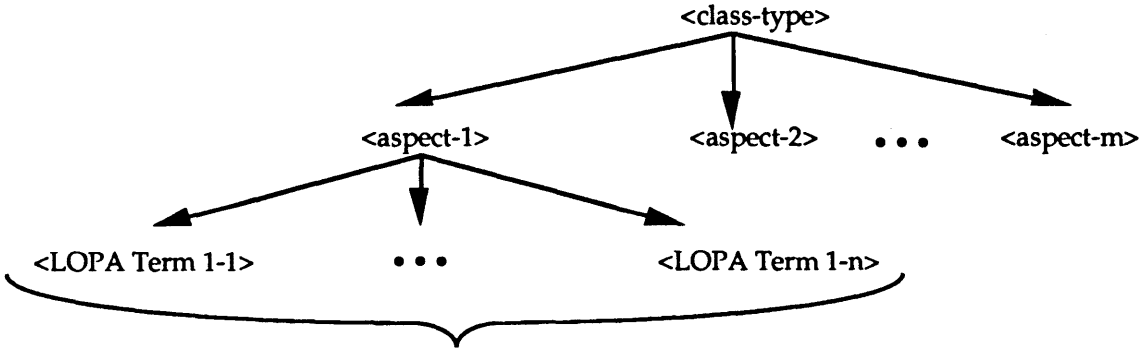

Figure 6.2: Symons and Tijmas' data taxonomy

Symons and Tijmas found that their method increased productivity ten-fold in the standardization of data elements, and in addition, improved the quality of the output. Their data representation schema is directly applicable to our requirements. Indeed, we could classify data elements by type-classes such as: `name`, `money-amount`, `date`, `year`, `ratio`, and

so forth. Then, for each data type, we also have to define standard "aspects", such as a `format`, `unit`, and `scale` for the type-class `money-amount`. Finally, for each aspect of a type-class, we have to define LOPA terms, such as `SCALE-1`, `SCALE-10`, `SCALE-100`, and so forth for the aspect scale of the type-class money amount (see Figure 6.3)

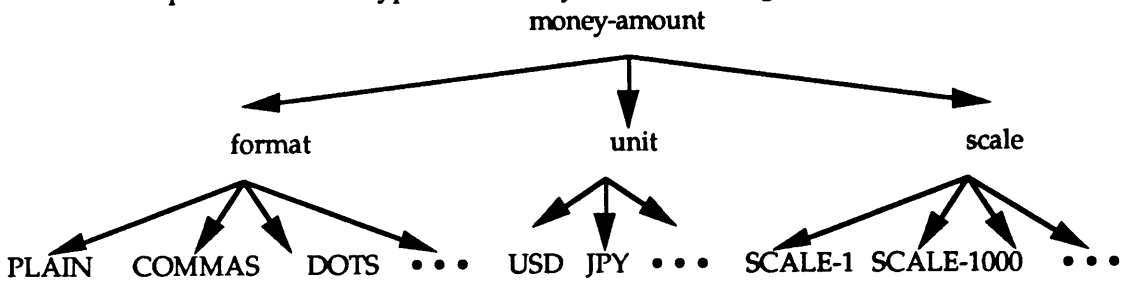

Figure 6.3: Money-amount data type

6.3.2 Data Dictionary

A data dictionary is one of the most important tool when managing data (also called the system catalog). The data dictionary may be regarded as a database in its own right whose contents are "data about data" (or metadata). It contains information concerning various objects that are of interest to the system itself. Examples of such objects are base tables, views, indexes, users, application plans, access privileges, and so on [Date, p. 39]. The great utility of the data dictionary is that it can be accessed like a regular table, hence the metadata can be obtained with ease.

A catalog structure will be explored as a support to store the metadata generated in the previous section. For example, the following catalog could be created:

application catalogue			
type-class	**aspect1**	**aspect2**	**aspect3**
money-amount	FIGURE-PLAIN	USD	SCALE-1
date	DATE-MM/DD/YYYY	-	-

The above data type catalog has one row for each data type defined by the semantics bridge, where each row has one column for each aspect of the representation of the given data type. At the intersection of a column and a row is a LOPA term. An alternative to this above catalog structure is given below:

application catalog

type-class	aspect
money-amount	PLAIN
money-amount	USD
money-amount	SCALE-1
date	DATE-YYYYMMDD
year	YEAR-YY
etc.	

The above structure eliminates the layer of aspects which is represented, for example, on Figure 6.2. It provides representation information on a given data type as a single block. Consider the following request of representation information for the class-type money-amount:

```
SELECT aspects FROM catalog
WHERE class-type = "money-amount"
```

which yields: PLAIN USD SCALE-1

The first catalog structure mentioned above, however, allows the selection to be more specific. The selection can thus be limited to certain aspects only of the representation:

```
SELECT aspect1, aspect2 FROM catalog
WHERE class-type = "money-amount"
```

which yields: PLAIN USD

When adopting the structure of the first table described above, one question that must be examined concerns the order in which the aspects characterizing a given data type have to be defined. As explained in more detail in the next chapter, this order will not matter.

6.4 DESIGN STEPS

We will conduct the design of a semantics mapping tool according to the steps given below (see Figure 6.4). Each step will answer one of the questions that we asked in § 6.2.3.

Functionality: data representation transparency

Step #1 - examine the data which is being exchanged between the application and the database, and based on this examination:

- identify the relevant data types, or classes;
- identify the relevant aspects of the representation of each data type.

(question #1).

Step #2 - define the necessary routines as well as their arguments to convert the representation of data (question #2).

Step #3 - build a data type catalog for the application and one for the database (questions #3).

Step #4 - design the bridge's strategy for converting data (question #4).

Functionality: selection operators transparency

Step #5 - design the bridge's strategy for processing operators not supported by the database (question #5).

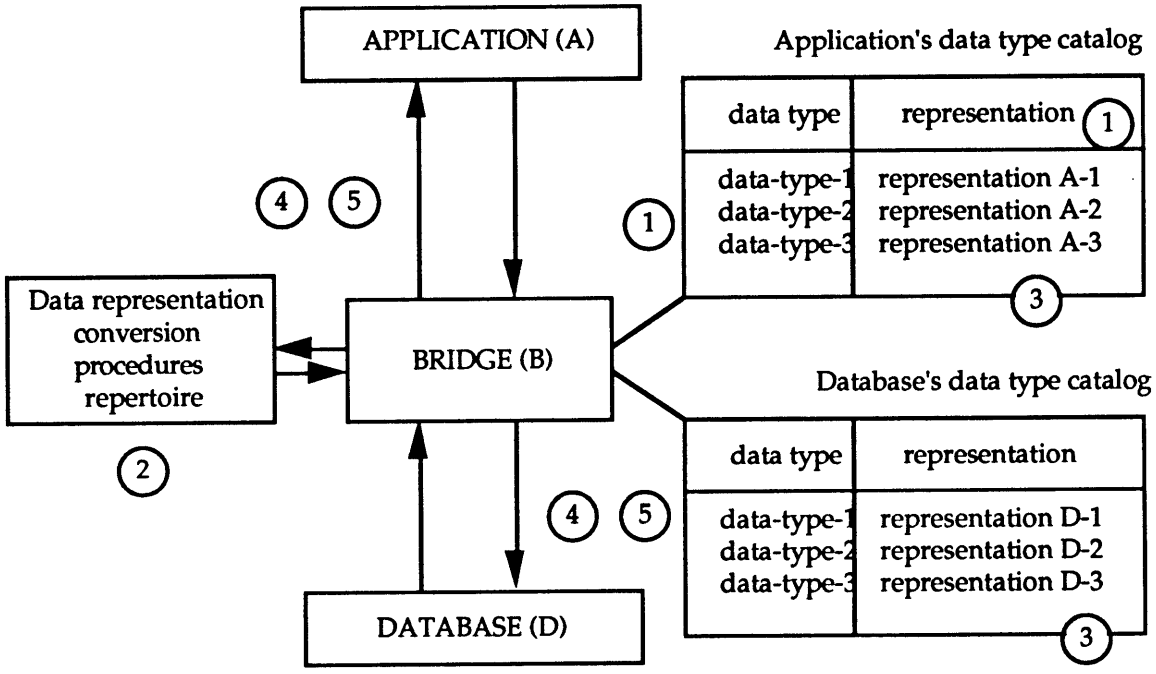

Figure 6.4: Design strategy

6.5 SUMMARY

The design will focus on the building of a semantics mapping tool for data representation conversion. The tool will be designed to support two essential functionalities:

1. provide data representation transparency to the application;
2. allow the application to use in its requests selection operators that are not supported by the database.

The next two chapters will present the tool's architecture and algorithms. Chapter VII will answer the first, second, and third questions we asked, and chapter VIII will answer the fourth and fifth questions.

-oOo-

Chapter VII.

SEMANTICS MAPPING TOOL: ARCHITECTURE

This chapter proposes an architecture of a semantics mapping tool satisfying the requirements described in the previous chapter. In regard to the design strategy provided in § 6.4, we will complete the first three design steps.

The chapter has three parts. The first part presents the notion of data type and data representation. The second part discusses the available options when designing a data conversion algorithms Two main algorithms are examined and compared. The third part presents how a catalog structure can be used to store the representation choices for each data type. We explain how this structure is the basis of a data representation protocol between the application and the database.

7.1 DATA TYPE, ASPECT, & LOPA TERM

7.1.1 Data Type Definition

As Symons and Tijmas suggested, we categorize data elements by classes, or domains, such as money amount, date, year, name, ratio, etc. The data types `money-amount`, `date`, and

year will be used as examples in the rest of the discussion. Money-amount is the class of all financial data elements which can be found in financial statements (e.g., balance sheet, income statement, and financial ratios). Date is the class of all data elements giving a date (e.g., today's date, financial reporting date, closing date, etc.). Finally, year is the class of all data elements giving a year.

A data type is created when different representations for that data type are encountered. The data type money-amount, for example, is defined because applications residing on different computers are likely to adopt different format, unit, and scale conventions for financial figures.

In addition, a final data type is defined in order to group all attributes whose representations do not vary. This data type, which we will call neutral in the rest of the discussion, is created in order to achieve a uniform treatment of all attributes. If, later in the utilization of the semantic mapping tool, it turns out that an attribute categorized under neutral may in fact be encountered with different representations, then a new data type is created. Examples of attributes that we classified under the data type neutral were various names (company names, addresses, etc.), telephone numbers, sic codes, etc. Notice that company names are very often encountered with different formats, or synonyms (e.g., Ford Motor Co versus THE FORD MOTOR CO), but we will not cover in this design the issue of synonyms matching and other names management techniques.[16]

7.1.2 Data Type Representation

The representation of a data element is composed of a collection of "aspects" [Symons and Tijmas, 1982]. A data representation can be defined by a variety of aspects, or fields, such as format, unit, scale, database source, geographic location, DBMS type, date of data creation, database credibility, confidence factor, etc. Data types are built so that all data elements belonging to a given data type have the same representation. Thus, the term "representation" will always refer to the representation of a data type. The following aspects are defined for the data types money-amount, date, and year (no aspects need to be defined for the data type neutral):

---

16 For more details on synonyms matching techniques, see Wong, T. K. (June 1989), "Data Connectivity in the Composite Information System/Tool Kit." B.S. Thesis, Massachusetts Institute of Technology.

- money-amount

  value: "1,000.5"

  format: uses a comma for thousands and a dot for decimals.

  unit: US dollars

  scale: thousand

- date

  value: "19,881,231"

  format: float number notation ("yy,yym,mdd")

- year

  value: "1990"

  format: 4-digit representation ("yyyy")

The semantics mapping tool that is being designed allows the definition of any number of aspects for a given data type. The list of the aspects that are used to describe the representation of a given data type will be called the representation template of this data type. Proceeding with the above examples, the representation template of the data type money-amount is (format unit scale). There is no particular order in which the aspects composing the template must be declared.

7.1.3 List of Permitted Aspects-Terms (LOPA Terms)

For each aspect of a data type representation, a list of permitted aspects-terms (or LOPA Terms) must be created in order to be able to designate different representations. For the data types defined earlier, for example, the following LOPA terms are created:

Data Type: money-amount

Aspect: format

LOPA terms: FIGURE-PLAIN, FIGURE-DISC, FIGURE-DATALN, etc.

FIGURE-PLAIN: no comma for thousands; a dot to mark decimals; e.g., "100000.5".

FIGURE-DISC: a comma to mark thousands; a dot to mark decimals; e.g., "100,000.5".[17]

FIGURE-DATALN: no comma for thousands; a dot to mark decimals or the end of the string when there is no decimals; e.g., "100000.".[18]

---

[17] Refers to the I.P. Sharp Disclosure database.

[18] Refers to the Finsbury Dataline database (the dot at the end of the string "100000." is wanted).

Data Type: money-amount

Aspect: unit

LOPA terms: USD, JPY, GBP, FRF, etc.

USD: US Dollars.

JPY: Japanese Yen.

GBP: Great Britain Pounds

FRF: French Francs.

etc...

Data Type: money-amount

Aspect: scale

LOPA terms: SCALE-1, SCALE-1000, etc.

SCALE-1: figures are expressed in units of their currencies.

SCALE-1000: figures are expressed in thousands of their currencies.

Data Type: date

Aspect: format

LOPA terms: DATE-MM/DD/YYYY, DATE-YY,YYM,MDD, etc.

DATE-MM/DD/YYYY: dates are expressed in the format "mm/dd/yyyy"; e.g., "03/23/1990".

DATE-YY,YYM,MDD: dates are expressed in the format "yy,yym,mdd"; e.g., "19,900,323".

Data Type: year

Aspect: format

LOPA: YEAR-YYYY, YEAR-YY

YEAR-YYYY: years are expressed in the four-digit format; e.g., "1990".

YEAR-YY: years are expressed in the two last-digit format; e.g., "90".

Thus, a particular representation of a data type is given by a collection of LOPA terms. This list of LOPA terms can be considered as an instance of the data type representation template, where each aspect takes its values from a limited number of terms. For the data type money-amount, for example, a representation could be: [FIGURE-PLAIN; USD; SCALE-

1000].[19] In the remainder of the text, a LOPA term associated with an aspect will sometimes be referred to as the aspect's identity.

7.2 CONVERSION ALGORITHMS

7.2.1 Assumption

There exist different types of mapping when converting data: one-to-one, one-to-many, many-to-one, and many-to-many mappings. An example of one-to-one mapping is converting a date format (e.g., from "19,900,407" to "04/07/1990"). An example of one-to-many mapping is converting a grade from a letter-based format into a figure-based format (e.g., from "A" to the range [90, 100]). Notice in this example that the image of "A" is a continuous range of figures. The mapping from region name to the state names that constitute the region is also a one-to-many mapping, but the image of a region is a discrete set of state names. An example of many-to-many mapping is the conversion of a given set of city names to the set of state names in which at least one city of the set lies. In this thesis, we shall focus on one-to-one mappings, and intra-data type representation conversion.

7.2.2 Algorithms

As we mentioned earlier in the chapter, the conversion tool is designed to handle one-to-one mappings. More specifically, the tool operates conversions from one representation of a data type to another representation of the same data type. In the scenario discussed in the previous chapter, for example, the net sales data item "20000000" was converted from the representation `[FIGURE-PLAIN; USD; SCALE-1]` to the representation `[FIGURE-DISC; JPY; SCALE-1000]`, hence becoming "2,800,000".

There are two a priori ways for designing the conversion procedures. The first consists of bundling some LOPA terms together and creating a finite number of so-generated "packages". Conversion procedures are written to convert data elements from one package to another. The second consists of allowing any combination of LOPA terms for the representation of a given data type. In this case, an independent conversion procedure must be written for each aspect of the data type representation.

---

[19] From now on, the notation [LOPA1; LOPA2; LOPA3; ...; LOPAn] will be used to designate a data type representation.

7.2.2.1 Package-to-Package Conversion

Let us take the data type `money-amount` is taken as an example, and assume that the following packages are created:

Package-1: format = `FIGURE-PLAIN`
unit = `USD`
scale = `SCALE-1`

Package-2: format = `FIGURE-DISC`
unit = `JPY`
scale = `SCALE-1000`

Package-3: format = `FIGURE-DATALN`
unit = `GBP`
scale = `SCALE-1`

Each package represents a particular association of LOPA terms, and is created to hide the details of its constituents. Indeed, if the above three representations are the only ones encountered, then the application, database, and bridge do not need to interact at the level of granularity provided by the individual aspects such as format, unit, scale, etc. The application can just indicate in its request the name of the package by which a given data element is represented.

One package must be chosen as the standard (e.g., package 1, but it does not matter which one is chosen) in order to reduce the number of conversion routines that are needed. Any conversion procedure converts the initial package to package-1, and package-1 to the desired package. The following procedure is representative of the package-to-package conversion algorithm:

```
Procedure: convert-package (value, initial-package, target-package)
Case (initial-package)
package-1:
   Case (target-package)
   package-1: done
   package-2: convert-package-1-to-2 (value)
   package-3: convert-package-1-to-3 (value)
   otherwise: signal error

package-2:
   Case (target-package)
   package-1: convert-package-2-to-1 (value)
   package-2: done
   package-3: convert-package-1-to-3 (convert-package-2-to-1 (value))
   otherwise: signal error.

package-3: ditto
```

Package-to-package conversion procedure

7.2.2.2 Aspect-By-Aspect Conversion

The second alternative allows any combination of LOPA terms for the representation of a data type, and provides conversion procedures for each aspect. A standard LOPA term is also defined for each aspect of the data type representation. As an example, we give below the aspect conversion procedure for the format of the data type `money-amount`:

Data Type:	`money-amount`
Aspect:	`format`
Standard Term:	`FIGURE-PLAIN`
Conversion routines:	`convert-money-amount-format`
	`convert-DISC-into-PLAIN`
	`convert-PLAIN-into-DISC`
	`convert-DATALN-into-PLAIN`
	`convert-PLAIN-into-DATALN`

The convert-money-amount-format procedure has exactly the same structure as the convert-package procedure which was defined earlier.

```
procedure: convert-money-amount-format (value, initial-format,
target-format)
Case (initial-format)
FIGURE-PLAIN:
   Case (target-format)
   FIGURE-PLAIN: done.
   FIGURE-DISC: convert-PLAIN-into-DISC (value)
   FIGURE-DATALN: convert-PLAIN-into-DATALN (value)
   otherwise: signal error

FIGURE-DISC:
   Case (target-format)
   FIGURE-PLAIN: convert-DISC-into-PLAIN (value)
   FIGURE-DISC: done.
   FIGURE-DATALN: convert-PLAIN-into-DATALN (convert-DISC-into-PLAIN
(value))
   otherwise: signal error

FIGURE-DATALN: ditto.
```

Aspect-by-aspect conversion procedure

The same description holds for the other aspects of the data type `money-amount` (i.e., unit and scale), and for all aspects of the other data types. The next section now compares the two approaches.

7.2.2.3 Comparison

In order to compare the two options, let us examine the conversion of the data item "20000000" from the representation `[FIGURE-PLAIN; USD; SCALE-1]` to the representation `[FIGURE-DISC; JPY; SCALE-1000]`, which corresponds to a conversion from package 1 to package 2. The necessary operations are the following:

- a conversion from FIGURE-PLAIN to FIGURE-DISC, which involves simple string manipulations in order to add commas;
- a multiplication by the exchange rate between US dollars and Japanese yen; and
- a division by 1000 for the scale conversion.

The input of the conversion is the string "20000000". Given the nature of the input and the operations to execute, adopting the package-to-package alternative will trigger the following steps:

```
(1) convert "20000000" into the float number 20000000;
(2) multiply the float number 20000000 by 140 and obtain
2800000000;[20]
(3) divide 2800000000 by 1000 and obtain 2800000;
(4) convert the float number 2800000 into the string "2800000"; and
(5) convert "2800000" into "2,800,000".
```

Package-to-package conversion

Adopting the second alternative will trigger a different series of steps (the output of one step is the output of the step following):

```
Separate format conversion: from FIGURE-PLAIN to FIGURE-DISC
input = ["20000000"; FIGURE-PLAIN; USD; SCALE-1]
(1) convert "20000000" into "20,000,000".
output = ["20,000,000"; FIGURE-DISC; USD, SCALE-1]

Separate unit conversion: from USD to JPY
input = ["20,000,000"; FIGURE-DISC; USD; SCALE-1]
(2) convert "20,000,000" into "20000000";
(3) convert the string "20000000" into the float number 20000000;
(4) multiply 20000000 by 140 and obtain 2800000000;
(5) convert the float number 2800000000 into the string "2800000000";
and
(6) convert "2800000000" into "2,800,000,000".
output = ["2,800,000,000"; FIGURE-DISC; JPY; SCALE-1]

Separate scale conversion: from SCALE-1 to SCALE-1000
input = ["2,800,000,000"; FIGURE-DISC; JPY; SCALE-1]
(7) convert "2,800,000,000" into "2800000000";
(8) convert the string "2800000000" into the float number 2800000000;
(9) divide 2800000000 by 1000 and obtain 2800000;
(10) convert the float number 2800000 into the string "2800000"; and
(11) convert "2800000" into "2,800,000".
output = ["2,800,000"; FIGURE-DISC; JPY; SCALE-1]
```

Aspect-by-aspect conversion (will be referred to as Algorithm[1])

Considering run time, the unit conversions for the `money-amount` data type are the sole expense, since they involve the fetching of exchange rates in a separate database. Because both approaches involve the same operations for unit conversion, the two alternatives are similar in performances.

However, the second option takes twice as many steps to reach the end result because the conversion of each aspect is activated separately. The format conversion which is first activated turns out to be useless since the unit conversion which follows has to convert the data item back to its initial format. Notice that the package-to-package conversion algorithm tracks down this type of inefficiencies and take advantage of the presence of short-circuits.

[20] We assume that 1 US $ = 140 ¥

Thus, each package-to-package conversion procedure is customized to minimize the number of operations.

The package-to-package conversion, however, suffers from an obvious lack of flexibility. When a new package is created from a new combination of LOPA terms, new software must be written. On the other hand, the second approach allows any combination of LOPA terms without the writing of new software. When a new aspect is added to a data type representation, the first approach requires the modification of the existing software in order to take into consideration the introduction of a new aspect. The second approach, however, requires only the writing of new software in a format which is well-defined (see the example of the procedure `convert-money-amount-format`).

Because extendability and flexibility are keys in our design, we believe that the second alternative is preferable.

Another strength of the aspect-by-aspect conversion algorithm is that it does not require any specific order in which the aspects of the representation are converted. Notice that conversions will be faster in some occasions due to a smaller number of steps to carry out, but the increased efficiency is not intended by the conversion algorithm. We consider this as being an advantage of the aspect-by-aspect paradigm because the introduction of a new aspect in a data type representation template does not require the updating of rules or any other sort of optimizer.[21]

7.2.3 Object-Oriented Implementation

An object-oriented support is chosen to implement the conversion tool because it supplies a simple way of organizing the knowledge on data representation conversions and also because it removes the complexity of the conversion routines [Levine, 1987].

An object is created for each data type defined. This object has an inferior object for each aspect of the data type representation. Finally, each aspect object has itself an inferior object for each LOPA term. Figure 7.1 shows the object tree so generated. All the data type object trees are then linked to a single object which acts as a root, hence creating a single

---

[21] The design of a conversion optimizer is in fact part of the enhancement of the tool that we discussed in the conclusion of chapter VIII.

conversion objects hierarchy. The knowledge on data representation conversions is distributed among the objects as follows:

- a `data-type` object has one slot containing the aspect names making up its representation (e.g., format, unit, scale, etc.).
- an `aspect` object has three slots to hold: (i) its LOPA terms; (ii) the standard term among the LOPA terms; and (iii) the arguments in the aspect conversion.

Notes: As we mentioned in the requirements analysis, an aspect conversion may require more arguments than the value to convert and the desired final LOPA term. A currency conversion, for example, requires a date and an exchange market as additional arguments.

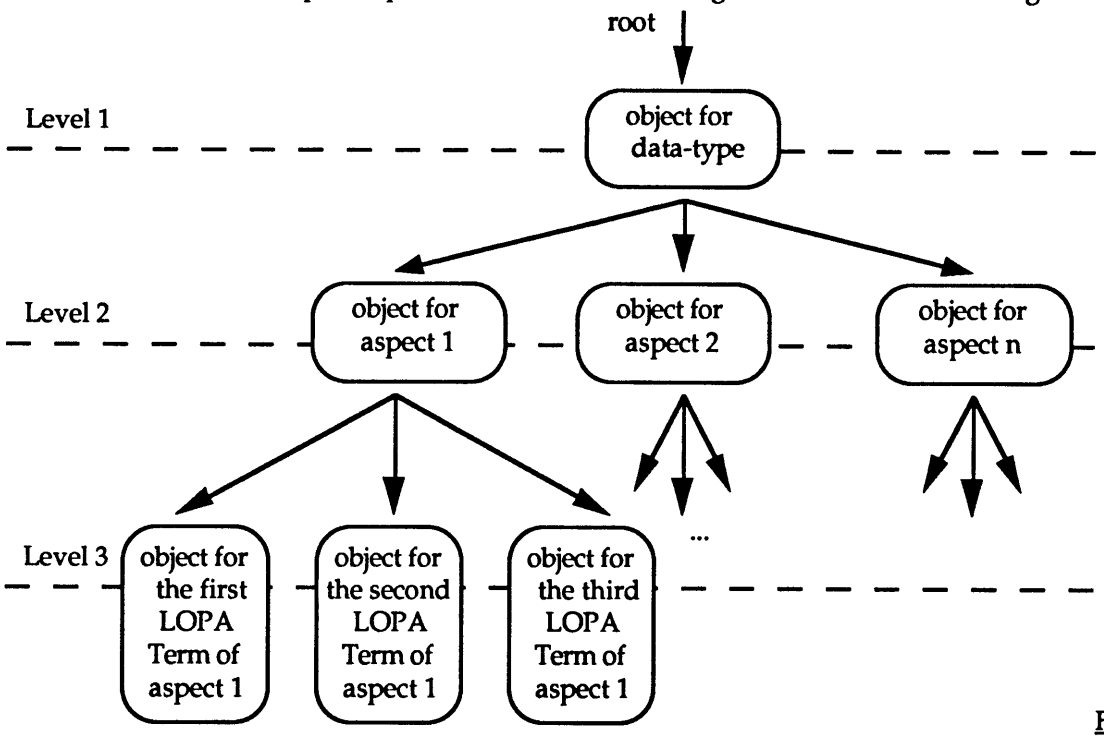

Figure 7.1: A data type's object tree

The conversion arguments are expressed as data types. For example, the unit conversion for the `money-amount` data type has two arguments (in addition to the value to be converted and the desired LOPA term): a `date` value and an `exchange-market` value, where `date` and `exchange-market` are two data types defined by the system.

- a `LOPA-Term` object has a "method" slot holding the name of the procedure that must be triggered in order to convert a value into the LOPA term.

The objects `money-amount`, `money-amount-unit`, and `USD` represent one branch in the conversion objects hierarchy:[22]

```
Object: MONEY-AMOUNT
    superiors: (value data-type)
    data-type: (value money-amount)
    aspects: (value (format unit scale))
    format: (value money-amount-format)
    unit: (value money-amount-unit)
    scale: (value money-amount-scale)

    methods: (value (:get-data-type-aspects :get-data-type-conversion-
    arguments))
```

```
Object: MONEY-AMOUNT-UNIT
    superiors: (value money-amount)
    aspect: (value unit)
    LOPA-Terms: (value (usd jpy gbp frf ...))
    Std-LOPA-Term: (value usd)
    arguments: (value (date exchange-market))
    currency-database: (value ip-currencydb)

    methods: (value :get-LOPA-Terms)
```

```
Object:
USD
    superior: (value money-amount-unit)

    methods: (value :convert-data-list-into-USD)
```

Notice the presence of the slot `Std-LOPA-Term` in the aspect object `money-amount-unit`. This slot holds the specific LOPA term which constitutes the standard ID for the aspect. Notice also the presence of the slot `currency-database` in the same object which holds the name of the bridge acting as the virtual driver to a database providing exchange rates.

The LOPA term objects represent entry points in the conversion objects hierarchy. A request to convert the aspect of a data item from one LOPA term to another is sent to the LOPA term object corresponding to the final term. A message-passing mechanism is used for that purpose [Levine, 1987]:

```
(send-message 'Final-LOPA-Term :convert '(data-item))
```

---

[22] The slots names are underlined; the facet names are written in italic.

The `data-item` argument is a structure which carries all the necessary information needed for the conversion. This data structure contains:
(i) the value to be converted and its data type representation (i.e., list of LOPA terms); and
(ii) the values of the conversion arguments and their respective data type representations.

The general template of this data structure is given next page.

```
((data-type-of-data-item-to-convert
   (value "value of data item")
   (aspect11 LOPA-Term-11)
   (aspect12 LOPA-Term-12)
   ...)
(data-type-of-conversion-argument-1
   (value "value of the first
argument")
   (aspect21 LOPA-Term-21)
   ...)
...
(data-type-of-last-conversion-argument
   (value "value of the last argument")
   (aspect-n1 LOPA-Term-n1)
   (aspect-n2 LOPA-Term-n2)
...)
```

General template of the data structure passed as argument to a LOPA Term object

Notice that the values and representation information of conversion arguments are optional. The conversion of the aspect `scale`, for example, does not require additional arguments. Consider the following example:

```
(send-message 'SCALE-1000 :convert 'data-item)
```

with `data-item` being:

```
((money-amount
      (value "20,000,000")
      (format FIGURE-DISC)
      (unit USD)
      (scale SCALE-1)))
```

This scale conversion will output:

```
((money-amount
      (value "20,000")
      (format FIGURE-DISC)
      (unit USD)
      (scale SCALE-1000)))
```

A unit conversion, however, requires a date and an exchange market arguments. Consider the conversion of the same data item:

```
(send-message 'JPY :convert 'data-item)
```

with `data-item` being:

```
((money-amount
   (value "20,000,000")
   (format FIGURE-DISC)
   (unit USD)
   (scale SCALE-1))
(date
   (value "04/18/1990")
   (format DATE-MM/DD/YYYY))
(exchange-market
   (value "NY")
   (format INITIALS)))
```

This request asks the object JPY to convert the data item "20,000,000", which is currently expressed in units of U.S. dollars, into the equivalent sum in Japanese yen. In order to so, the unit conversion will use the exchange rate which was given on the New York Stock market on April 18, 1990. The database which is accessed to obtain this exchange rate is indicated in the slot `currency-database` in the aspect object `MONEY-AMOUNT-UNIT`. The unit conversion will output:[23]

```
((money-amount
   (value "2,800,000,000")
   (format FIGURE-DISC)
   (unit JPY)
   (scale SCALE-1))
(date
   (value "04/18/1990")
   (format DATE-MM/DD/YYYY))
(exchange-market
   (value "NY")
   (format INITIALS)))
```

It is important to point out that this data structure is built prior to sending a message to the LOPA term object. The role of an object is not to find out how to obtain the values of the arguments and their representations, but to carry out the conversion once all the information needed is found. Chapter VIII explains how the arguments values and their representations are determined.

---

[23] Chapter VIII presents in more detail how currency conversions are carried out.

7.3 DATA TYPE CATALOG

7.3.1 Basic Structure

Now that we have built a framework to manipulate data types and their representations, as well as to convert from one data type representation to another, we can begin to elaborate the data representation protocol between the application and the database.

This protocol is based on the consultation of catalogs which hold information on the data representation conventions adopted by each side, i.e. the application and the database. The general template of the catalog is shown below:

General template of a data type catalog

catalog-name				
data-type-name	**aspect-1**	**aspect-2**	**...**	**aspect-m**
data-type-1	Term-11	Term-12		Term-1m
data-type-2	Term-21	Term-22		Term-2m
...				
data-type-n	Term-n1	Term-n2		Term-nm

A data type catalog is created for the application as well as for each database that the application needs to access:

Application's catalog

application-catalog			
data-type	**format**	**unit**	**scale**
money-amount	FIGURE-PLAIN	USD	SCALE-1
date	DATE-MM/DD/YYYY	NIL	NIL
year	YEAR-YYYY	NIL	NIL
neutral	NIL	NIL	NIL

I.P. Sharp Disclosure's catalog

disclosure-catalog			
data-type	**format**	**unit**	**scale**
money-amount	FIGURE-DISC	N/A	SCALE-1000[24]
date	DATE-YYYYMMDD	NIL	NIL
year	YEAR-YYYY	NIL	NIL
neutral	NIL	NIL	NIL

Finsbury Dataline's catalog

dataline-catalog			
data-type	**format**	**unit**	**scale**
money-amount	FIGURE-DATALN	N/A	N/A
date	DATE-DD-MM-YY	NIL	NIL
year	YEAR-YY	NIL	NIL
neutral	NIL	NIL	NIL

The notation "NIL" indicates that the aspect is not relevant in the representation of the given data type. The notation "N/A" indicates that, although the aspect matters in the representation of the given data type, no constant LOPA term can be given because the identity of the aspect under consideration varies across the records in the database. In the Disclosure database, for example, no specific unit can be given for the `money-amount` data type because the database provides financial information for companies from different nationalities, hence in different currencies. The handling of this situation is treated in the next section.

A data type catalog is completed by an attribute catalog which indicates for each attribute used by the application or the database the data type to which it belongs. We give below examples of attribute catalogs:

[24] All financial information provided by I.P. Sharp Disclosure is always expressed in 1000's of the company's currency (see manual).

application-attribute-catalog	
attribute-name	**data-type**
company-name	neutral
sic-code	neutral
net-income	money-amount
net-sales	money-amount
report-date	date
year	year
etc.	

disclosure-attribute-catalog	
attribute-name	**data-type**
co	neutral
pc	neutral
ex	exchange-market
ni	money-amount
ns	money-amount
cf	date
year	year

dataline-attribute-catalog	
attribute-name	**data-type**
companyname	neutral
code	neutral
efo	money-amount
sales	money-amount
periodending	date
year	year
etc.	

Notice that in the attribute catalog `disclosure-attribute-catalog` the database attributes NS and NI were declared as belonging to the same data type `money-amount`. In doing so, it is assumed that NS and NI information for a given company are provided with the same representation: same format, same unit, and same scale. This assumption is indeed true for I.P. Sharp Disclosure. It is also true for Finsbury Dataline since there is one format, one unit, and one scale per tabulated output (see an example of Dataline's tabulated output in § 3.2.3). However, it might not be true for another financial database, and it may well be the case that NI data elements are provided in `[FIGURE-DISC; JPY; SCALE-1]` and NS data elements in `[FIGURE-DISC; JPY; SCALE-1000]`. In this case, the NS and NI attributes would be declared as belonging to two different data types: money-amount-1 and money-amount-2 whose representation would be:

money-amount-1: `[FIGURE-DISC; JPY; SCALE-1]`

money-amount-2: `[FIGURE-DISC; JPY; SCALE-1000]`

7.3.2 Data Type Catalog Extension

In the case of the Disclosure database the unit aspect of the data type `money-amount` could not be described by a constant LOPA term. More generally speaking, an aspect term which cannot be described by a constant LOPA term in the database's data type catalog can nevertheless be determined when obtaining data from the database. The database may have explicit attributes giving the aspect's identity. In Dataline, for example, a tabulated output always indicates both the currency and the scale, hence these two aspects can be treated as regular database attributes (see Dataline's output given in chapter III). If the database does not have explicit attributes for the aspects' identities, they may be deduced from available information. In I.P. Sharp Disclosure, for example, there is no attribute for the currency, but the `country-of-incorporation` attribute is a clear indication of the company's currency [Folinus et al, 1974, and Paget, 1989].

A representation aspect whose identity, or LOPA Term, cannot be described as a constant in the database's catalog is entered in this later with a notation indicating:

(1) which database attributes can be used to deduce the LOPA term; and

(2) which bridge-specific procedure has to be called up to obtain the LOPA term.

This notation is referred to as a "function call" and written as:

```
(<function-name> <data-element-1> ... <data-element-n>)
```

Based on this new formalism, the Disclosure and Dataline's catalogs become:

I.P. Sharp Disclosure's catalog

disclosure-catalog			
data-type	**format**	**unit**	**scale**
money-amount	FIGURE-DISC	(get-unit-disc IN)	SCALE-1000
date	DATE-YYYYDDMM	NIL	NIL
year	YEAR-YYYY	NIL	NIL
neutral	NIL	NIL	NIL

dataline-catalog			
data-type	**format**	**unit**	**scale**
money-amount	FIGURE-DATALN	**(get-unit-dataln CURRENCY)**	**(get-scale-dataln SCALE)**
date	DATE-DD-MM-YY	NIL	NIL
year	YEAR-YY	NIL	NIL
neutral	NIL	NIL	NIL

The functions `get-unit-disc`, `get-unit-dataln`, and `get-dataln-scale` are simple procedures that are used to interpret the values of some specific database attributes. The function `get-unit-disc`, for example, is given below:

```
procedure:   get-unit-disc
(IN)
Case(IN)
      FRANCE: return(FRF)
      UNITED      KINGDOM:
return(GBP)
      JAPAN: return(JPY)
      etc.
      Otherwise: return(USD)
EndOfCase
```

On an administration point of view, it is recommended that an application administrator be responsible for creating and maintaining all the catalogs. Notice that an application's end-user herself should be able to specify what data representation she wants, by having a personal data type catalog.

7.3.3 Catalog-Based Data Representation Protocol Between The Application and the Database

Let us assume that the application and the database interact through the use of messages as shown below:

```
(send-message 'Bridge-name :get-data 'application-data-type-catalog
                                    'query)
```

where:

- `Bridge-name` indicates to which bridge the message is sent. Recall that an application may require the utilization of multiple bridges if multiple outside computers are to be accessed.
- `:get-data` indicates that the application requests data (the application can also request general information on the database, table names, column names [Champlin, 1987]).
- `application-data-type-catalog` indicates the name of a data type catalog which describes the intended data representation conventions in the query;
- `query` is the request specifying which data to select and how. We assume that the attribute names in the query are those used by the outside computer [Wong, 1989].

Let us illustrate this data representation protocol with the query which was used in the requirements analysis (see § 6.2.1). In this illustration, we will assume for the sake of simplicity that no representation includes function calls -- this type of situation is examined in detail in the next chapter.

```
(send-message 'IPSharp-Disclosure
      :get-data
      'application-data-type-catalog
      '(SELECT ns, cf FROM disclosure
      WHERE co = "HONDA MOTOR CO" and ns > "20000000"))
```

As explained in the requirements analysis (see § 6.1.1), a bridge has to perform two steps:

Step 1: convert the query into the database's data representation conventions (A-B to B-D request conversion); and

Step 2: convert the data obtained from the database into the application's data representation conventions (D-B to B-A data conversion).

(i) Query conversion

As was explained in the scenario analysis, the predicate (`"ns > 20000000"`) must be converted into the database's data representation conventions. In order to do so, the data type catalogs of both the application and database are used as follows:

Step 1 - the database's attribute catalog indicates that NS belongs to the `money-amount` data type (see Disclosure's attribute catalog in § 7.3.1);

Step 2 - the application's data type catalog indicates that its `money-amount`'s representation is `[FIGURE-PLAIN; USD; SCALE-1]` (see the application's data type catalog in § 7.3.1);

Step 3 - the database's data type catalog indicates that its `money-amount's` representation is `[FIGURE-DISC; JPY; SCALE-1000]` (see Disclosure's data type catalog in § 7.3.1);[25]

step 4 - For each aspect where the LOPA terms do not match between the application and the database, a conversion request is sent to the desired LOPA object (according to Algorithm[1] given in § 7.2.3):

- the LOPA terms `FIGURE-PLAIN` and `FIGURE-DISC` do not match, hence a format conversion request is sent to the object `FIGURE-DISC`:

```
1> (send-message 'FIGURE-DISC :convert '(money-amount (value
"20000000") (format FIGURE-PLAIN) (unit USD) (scale SCALE-1))
<1 (money-amount (value "20,000,000") (format FIGURE-DISC)
(unit USD) (scale SCALE-1))
```

- the LOPA terms `USD` and `JPY` do not match, hence a unit conversion request is sent to the object `JPY`:

```
2> (send-message 'JPY :convert '(money-amount (value
"20,000,000") (format FIGURE-DISC) (unit USD) (scale SCALE-1)))
<2 (money-amount (value "2,800,000,000") (format FIGURE-DISC)
(unit JPY) (scale SCALE-1))
```

- the LOPA terms `SCALE-1` and `SCALE-1000` do not match, hence a scale conversion request is sent to the object `SCALE-1000`:

```
2> (send-message 'SCALE-1000 :convert '(money-amount (value
"2,800,000,000") (format FIGURE-DISC) (unit JPY) (scale SCALE-
1)))
<2 (money-amount (value "2,800,000") (format FIGURE-DISC) (unit
JPY) (scale SCALE-1000))
```

Notice that the order in which the above conversion is operated corresponds to the order in which the aspects are listed in the representation template. Recall that the representation template of a data type is given by the slot `aspects` in the data type object (cf § 7.2.3).

(ii) Data conversion

Let us assume that we obtain the following data from the database:

(("NS" "CF")
("2,909,574" "19,860,228")
("2,868,305" "19,870,228"))

---

[25] In reality, Disclosure's data type definition gives a "N/A" value for the unit LOPA term. We assume in this section that there exists a LOPA term, say JPY.

The conversion of each data element present in the above data set is operated the same way as presented in (i) -- in practice, the same conversion routine is called. The conversion of the data element "2,909,574", for example, is operated as follows:

Step 1 - the data element corresponds to the database attribute NS, whose data type is determined by looking up the database's attribute catalog: `money-amount;`

Step 2 - the database's data type catalog indicates that its `money-amount's` representation is `[FIGURE-DISC; JPY; SCALE-1000];`

Step 3 - the application's data type catalog indicates that its `money-amount's` representation is `[FIGURE-PLAIN; USD; SCALE-1];`

step 4 - For each aspect where the LOPA terms do not match between the database and the application, a conversion request is sent to the desired LOPA object:

- the LOPA terms `FIGURE-DISC` and `FIGURE-PLAIN` do not match, hence a format conversion request is sent to the object `FIGURE-PLAIN`:

```
1> (send-message 'FIGURE-PLAIN :convert '(money-amount (value
"2,909,574") (format FIGURE-DISC) (unit JPY) (scale SCALE-
1000))
<1 (money-amount (value "2909574") (format FIGURE-PLAIN) (unit
JPY) (scale SCALE-1000))
```

- the LOPA terms `JPY` and `USD` do not match, hence a unit conversion request is sent to the object `USD`:

```
2> (send-message 'USD :convert '(money-amount (value "2909574")
(format FIGURE-DISC) (unit USD) (scale SCALE-1)))
<2 (money-amount (value "20782,242") (format FIGURE-PLAIN) (unit
USD) (scale SCALE-1000))
```

- the LOPA terms `SCALE-1000` and `SCALE-1` do not match, hence a scale conversion request is sent to the object `SCALE-1`:

```
2> (send-message 'SCALE-1 :convert '(money-amount (value
"20782,242") (format FIGURE-PLAIN) (unit USD) (scale SCALE-
1000)))
<2 (money-amount (value "20782242") (format FIGURE-PLAIN) (unit
USD) (scale SCALE-1))
```

Figure 7.2 next page represents the flow chart of the two-step process.

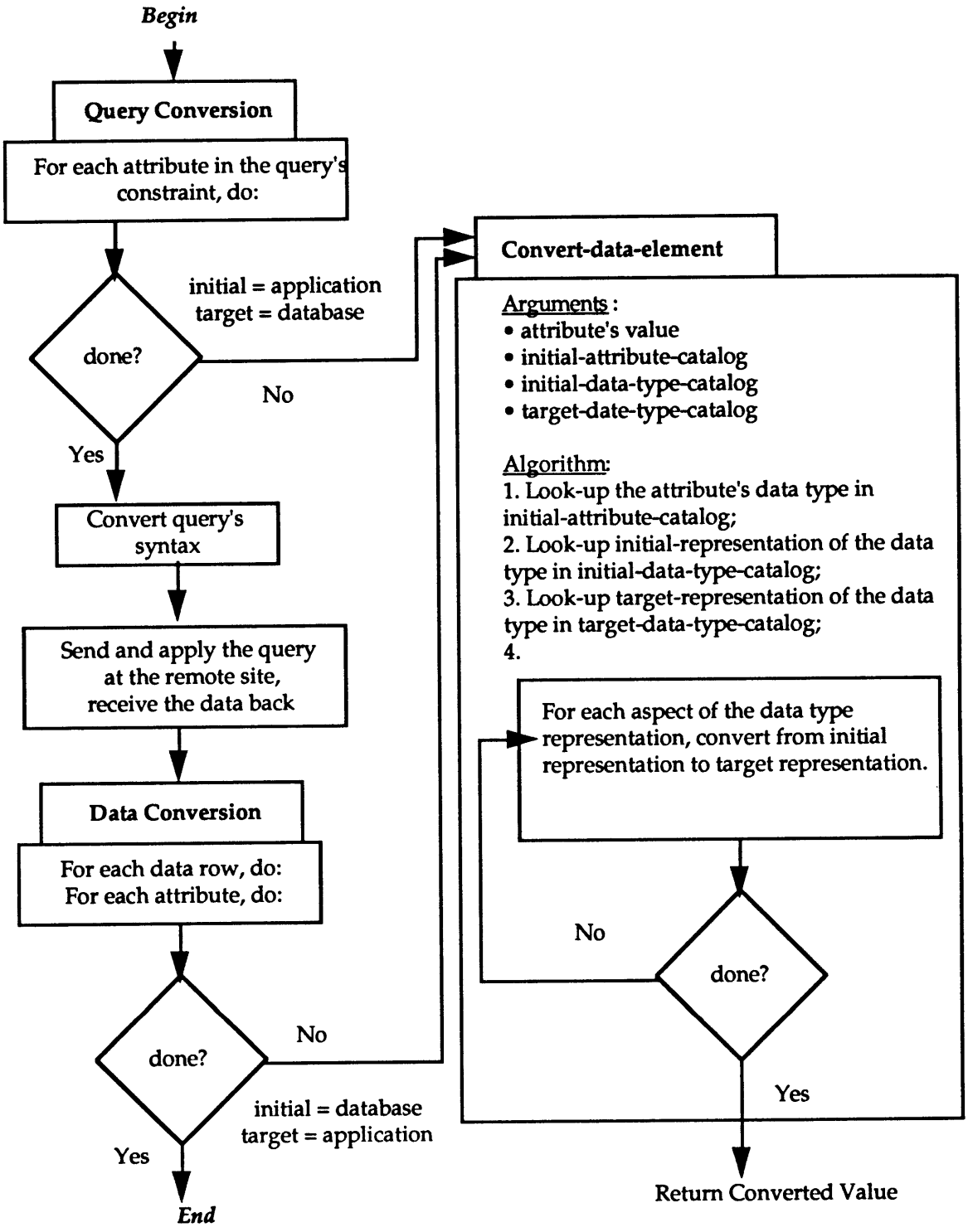

Figure 7.2: Two-step process: query conversion, and data conversion

7.4 SUMMARY

In summary, we have built an architecture centered around the following components:

(1) data types;

(2) data type representation templates as collections of aspects or fields;

(3) data type representations as lists of permitted terms;

(4) data definition conversions as series of separate conversion operations, each operation converting a specific aspect of the data type representation;

(5) data type catalogs as tables storing data type representations pertaining to a given party (application or database);

The next chapter tackles the last two steps of our design plan, i.e. the design of the bridge's strategies for converting data, and for processing operators not supported by the database.

-oOo-

Chapter VIII

SEMANTICS MAPPING TOOL: ALGORITHMS

The design of a semantics mapping tool is completed in this chapter. We address the fourth and fifth steps of the design plan elaborated in chapter VI (cf § 6.4).

The chapter begins by classifying the predicates that compose a query's constraint (see the definition of a query's constraint in § 6.2.1). The resulting taxonomy is then used to envisage various options when processing an application's request (A-B request). Two main options are

presented and compared, and the two main steps of the algorithm used by a semantics mapping tool are presented: query parsing and data filtering. Some details on the implementation are given; in particular, we describe how currency conversions are carried out. Finally, we conclude this design by suggesting future developments of the tool.

8.1 DEFINITIONS & TERMINOLOGY

8.1.1 Static versus dynamic data type representation

The representation of a data type is either static or dynamic. A representation is static if each aspect of the data type representation is given by a constant LOPA term across all records of the database. A representation of a data type is dynamic when the identity of at least one aspect of the representation varies across the records in the database, and has to be deduced from the data. In the previous section, a dynamic LOPA term was entered in a data type catalog as a function call with the function name and the names of its arguments (i.e., attribute names).

All the arguments of all the function calls of a dynamic data type representation are collected into a list of attributes that is called `infer-arguments`. The notation refers to the fact that these database attributes are used to determine the identity of some of the aspects of a dynamic data type representation. In the Dataline database, for example, the representation of the data type `money-amount` is given by `[FIGURE-DATALN; (get-unit-dataln CURRENCY); (get-scale-dataln SCALE)]`, hence its `infer-arguments` is the list `(CURRENCY SCALE)`. Using this new terminology, we conclude that a data type representation is static if its `infer-arguments` list is empty -- in the remainder of the design this property will be referred to as the static property.

8.1.2 Self-sufficient versus context-dependent conversion

The conversion of a data element from one representation to another is either self-sufficient or context-dependent. By self-sufficient, we mean that the conversion can be operated with the mere knowledge of the initial and desired representation of the data type. Converting the value "04/17/90" from `[DATE-MM/DD/YY]` to `[DATE-YYYYMMDD]` can be carried out without giving more than what is in this sentence. Converting the value "20000000" from `[FIGURE-PLAIN; USD; SCALE-1]` to `[FIGURE-DISC; JPY; SCALE-1000]`, on the other hand, needs additional information for the currency conversion, such as a date and an exchange

market. This last example illustrates what is called a context-dependent conversion. Notice that the conversion of each individual aspect can be qualified as either self-sufficient or context-dependent.

Within a given database, all conversion procedures' arguments for all aspects of the representation of a given data type are collected into a list of database attributes that is called `conversion-arguments`. In the Disclosure database, for example, the `conversion-arguments` list for the data type `money-amount` is given by `(CF EX)`. Let us show below how we arrive at this result:

The representation template of `money-amount` is `(format unit scale)`. The contents of the objects associated with these aspects are reproduced below:

Object: MONEY-AMOUNT-FORMAT
- superiors: (*value* money-amount)
- aspect: (*value* format)
- LOPA-Terms: (*value* (FIGURE-PLAIN FIGURE-DISC FIGURE-DATALN ...))
- Std-LOPA-Term: (*value* FIGURE-PLAIN)
- arguments: nil
- methods: (*value* :get-LOPA-Terms)

Object: MONEY-AMOUNT-UNIT
- superiors: (*value* money-amount)
- aspect: (*value* unit)
- LOPA-Terms: (*value* (usd jpy gbp frf ...))
- Std-LOPA-Term: (*value* usd)
- arguments: (*value* (date exchange-market))
- currency-database: (*value* ip-currencydb)
- methods: (*value* :get-LOPA-Terms)

Object: MONEY-AMOUNT-SCALE
- superiors: (*value* money-amount)
- aspect: (*value* scale)
- LOPA-Terms: (*value* (scale-1 scale-1000 scale-0.01 ...))
- Std-LOPA-Term: (*value* scale-1)
- arguments: nil
- methods: (*value* :get-LOPA-Terms)

The only aspect whose conversion requires additional arguments is the unit, since in both the objects `MONEY-AMOUNT-FORMAT` and `MONEY-AMOUNT-SCALE` the slot `arguments` has no value. The slot `arguments` in `MONEY-AMOUNT-UNIT` provides the

data type names `date` and `exchange-rate`. By looking up the database's attribute catalog for these two data types we finally obtain the database attributes `CF` and `EX`. (see Disclosure's attribute catalog in § 7.3.2).

Using this new terminology, we derive that a data type representation conversion if self-sufficient if the data type's `conversion-arguments` list is empty -- in the remainder of the design this property will be referred to as the self-sufficient property. Figure 8.1provides the flow chart of the algorithm generating the `infer-arguments` and `conversion-arguments` lists for a given data type. The static and self-sufficient properties are used in the next section to parse an application's request.

8.2 PREDICATE TYPES

The constraint of a query is composed of selection predicates. A selection predicate is defined by the following notation:

```
(<attribute> <operator> <value>)
with:
<operator>: =|>|<|<>...
<attribute>: any attribute supported by the database
<value>: any value consistent with the attribute's data type
```

With this notation, we will refer to the data type of a predicate as the data type of the predicate's attribute, and we will also talk about the conversion of a predicate as the conversion of the predicate's value.

A selection predicate can be characterized by two characteristics:
(1) the predicate can be either convertible into the database's data representation conventions or not; and
(2) the predicate's operator can be either applicable to the database or not.

In regard to the first characteristic, we have the following proposition:

Proposition: P(predicate)	
A selection predicate can be converted <==>	static (predicate) = True AND self-sufficient (predicate) = True.

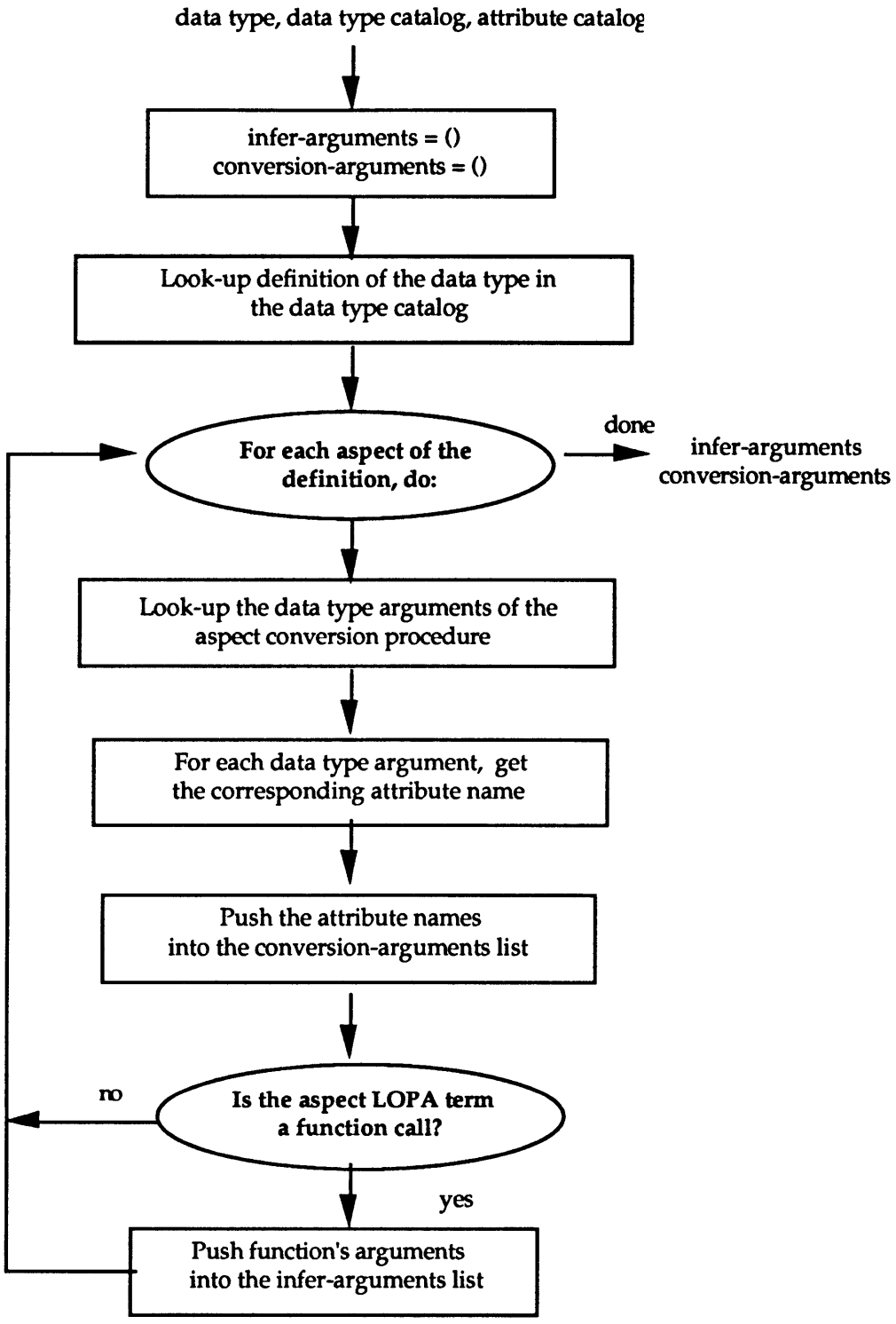

Figure 8.1: Infer-arguments and conversion-arguments generation

Let us examine this proposition closer. Proposition "P" says that a predicate can be converted into the database's data representation conventions if and only if it verifies the static and self-sufficient properties. Verifying the static property means that the predicate's

data type must have a static representation. Indeed, if it were not the case, then the identify of the dynamic aspect(s) would have to be deduced from the data, data that the bridge has not yet at the time the query is being converted. Verifying the self-sufficient property means that the conversion of the predicate's value is self-sufficient. Indeed, if it were not the case, then the conversion would need additional information that the bridge has not either at the time of the query is being converted. Thus, it is necessary that both properties be satisfied to be able to convert a predicate.

In regard to the second characteristic, each bridge is provided with a simple predicate filter. This filter examines whether the operator is supported by the database or not. The procedure given below, for example, is representative of the predicate filter for the Disclosure database:

```
procedure: Filter (predicate); returns True or False
operator = get-operator-from-predicate(predicate)
data-type = get-data-type-of-attribute(predicate's attribute)
Case (data-type)
money-amount, date, or year:
        Case(operator)
        =:              return(True)
        >, < or <>:     return (False)
        otherwise:      return (Syntax Error)
etc.
```

Predicate filter for Disclosure

The predicate `(> year "1987")`, for example, would not be applicable to the database Disclosure as the above filter indicates.

Based on these two characteristics, the selection predicates present in a query can be classified into three categories: first-class; second-class; and third-class.

First-class selection predicates:

A first-class selection predicate is a predicate whose value cannot be converted into the database's data representation conventions because either its data type has a dynamic representation, or the conversion into the database's data representation conventions

requires additional arguments. Hence, the following proposition holds ("P" refers to the proposition defined the previous page):

A predicate belongs to the first class <==> P(predicate) = False

Example: in the case of Dataline, the predicate `(> sales "20000000")` is not convertible because its data type (i.e. `money-amount`) has the following dynamic representation: `[FIGURE-DATALN; (get-unit-dataln CURRENCY); (get-scale-dataln SCALE)]`. The `infer-arguments` and `conversion-arguments` lists for `money-amount` in Dataline are `(CURRENCY SCALE)` and `(PERIODENDING)`, hence the predicate does not satisfy both the static and the self-sufficient properties.

Second-class selection predicates:

A second-class selection predicate is a predicate whose value can be converted into the database's data representation conventions, but the predicate's operator cannot be applied to the database. Hence, the following proposition holds:

A predicate belongs to the second class <==>	P(predicate) = True AND Filter(predicate) = False

Example: In the case of Dataline, the predicate *(> year "1988")* can be converted into the database's data representation conventions: the conversion from *YEAR-YYYY* to *YEAR-YY* is indeed self-sufficient. However, the operator ">" is not supported by the Dataline system, hence *Filter[(> year "1988")]* returns *False.*

Third-class selection predicates:

A third-class selection predicate is a predicate whose value can be converted into the database's data representation conventions, and whose operator can be applied to the database. Hence, the following proposition holds:

A predicate belongs to the third class <==>	P(predicate) = True AND Filter(predicate) = True

In the case of the Dataline database, the predicate *(= year "1988")* is convertible into *(= year "88")*, and applicable to the database (the operator "=" is applicable).

This taxonomy of predicates has helped us distinguishing various options to satisfy the application's request. These options are discussed in the next section.

8.3 QUERY PROCESSING OPTIONS

In the next three sections, we examine how to process each class of predicate. We start with the third-class predicates since they are the easiest to treat, proceed with the second-class predicates, and finally terminate with the first-class predicates which are the most difficult to process.

8.3.1 Application of Third-Class Predicates

Since the third class predicates can be converted into the database's data representation conventions and also be applicable to the database, the predicate is included in the B-D query.

8.3.2 Application of Second-Class Predicates

Since a second-class predicate cannot be applied to the database, the strategy that we have adopted consists of downloading the data obtained from the database into a local powerful database management system (DBMS), and then apply the predicate to the data in this DBMS. Notice that the data that is being downloaded in the DBMS is still in the database representation conventions, hence the predicate must be converted before it is applied to the DBMS.

The downloading of data in a local DBMS to perform further processing on the data was one characteristic of the system Multibase that we have thus borrowed for our design [Landers and Rosenberg, 1985].

8.3.3 Application of First-Class Predicates

In this design, we have examined two options for the application of third-class predicates. We will consider the query given below to illustrate these options (the query is sent to the bridge acting as a virtual driver to the Disclosure database):

```
SELECT co, ni, cf FROM disclosure
```

8.3.3 Application of First-Class Predicates

In this design, we have examined two options for the application of third-class predicates. We will consider the query given below to illustrate these options (the query is sent to the bridge acting as a virtual driver to the Disclosure database):

```
SELECT co, ni, cf FROM disclosure
WHERE (co = "HONDA MOTOR CO" and co = "JAGUAR PLC")
AND (ns > "20000000")
```

The request (or query) asks for the financial reporting dates (called "CF" in the database) at which Honda Motor Co and Jaguar PLC have net sales (called "NS" in the database) greater than twenty million U.S. dollars.

The predicate `(ns > "20000000")` is the only predicate whose intended data representation conventions differ from the database's. However, the predicate cannot be converted because the representation of the data type `money-amount` is dynamic. In order to apply this first-class predicate, the bridge can adopt one of the following strategies:

Strategy #1: convert the predicate by requesting specific information from the database, and then apply the converted predicate to the data obtained from the database.

Strategy #2: convert into the application's data representation conventions the data obtained from the database, and then apply the (non converted) predicate.

8.3.3.1 Strategy #1

Strategy #1's goal is to build a new query by converting the predicate `(ns > "20000000")`. Since the currencies in which the financial information for Honda and Jaguar provided by Disclosure differ, the predicate has to be converted for each separate company. In addition, financial statements may be available for several dates, say `date-1` and `date-2`. Thus, strategy #1 re-writes the query's constraint as shown below:

```
SELECT co, ni, cf FROM disclosure WHERE
  (co = "HONDA MOTOR CO" AND
  ((ns > "20000000 converted into Honda's currency for date-1's exchange
  rate") OR
  (ns > "20000000 converted into Honda's currency for date-2's exchange
  rate")))
OR
  (co = "JAGUAR PLC" AND
  ((ns > "20000000 converted into Jaguar's currency for date-1's
  exchange rate") OR
  (ns > "20000000 converted into Jaguar's currency for date-2's exchange
  rate")))
```

Strategy #1's algorithm is articulated below (the algorithm has also integrated the processing of the first- and second-class predicates):

Query Parsing

(1) access the database a first time to obtain Honda and Jaguar's currencies as well as all the dates at which financial statements are available for these two companies;

(2) for all the dates obtained in the first step, fetch the exchange rates for both Honda and Jaguar's currencies in respect to US dollars -- the exchange rates can be obtained from the same or another database;

(3) build a new query (as shown above);

(4) remove all second-class predicates from the new query;

Database Access

(5) apply to the database the query built in (4) (second database access);

Data Filtering

(6) locally apply the second-class predicates that were removed in (4); and

(5) convert the final set of data obtained in (6) into the application's data representation conventions.

8.3.3.2 Strategy #2

Strategy #2 has two goals: (i) minimize the number of accesses to the database; and (ii) leave the first-class predicates intact. To reach these two goals, strategy #2 adopts the following algorithm (the algorithm has also integrated the processing of the first- and second-class predicates):

Query Parsing

(1) parse the query's constraint in order to remove first- and second-class predicates;

Computer Access

(2) apply the resulting query to the database;

Data Filtering

(3) locally apply the second-class predicates to the data obtained in (2);

(4) convert the data obtained in (3) into the application's data representation conventions; and

(5) locally apply the first- predicates to the data obtained in (3).

8.3.3.3 Strategies Comparison and Design Choice

The first strategy is more efficient than the second because it converts the exact amount of data requested by the application, whereas the second strategy may convert more data than the application wants.

On a performance stand-point, the first strategy will take longer only because it needs to access the database twice. Since access to the database takes up most of the time to process an application's request, the first strategy is virtually twice as long as the first strategy. But, apart from the time taken to access the database, the first strategy is faster because a smaller number of data elements is converted. If access to the database can be speeded up, then the first strategy will clearly show better performance.

However, the first strategy is complex because it involves a significant query rewriting. We will explore the second more simple strategy, which is presented in detail in the remainder of the chapter.

Figure 8.2 represents three aspects of the design:
(1) the structure of the initial query (called A-B query in § 6.1.1) as a combination of the query being used to request data to the database (called B-D query in § 6.1.1), and the first- and second-class predicates. These latter are further decomposed in two queries: a query made of the second-class (converted) predicates, and a query made of the first-class (non converted) predicates.
(2) the application of different queries at different points in time and to different databases (see description of steps below);
(3) the manipulation of data during (i) data selection; (ii) downloading; and (iii) data conversion.

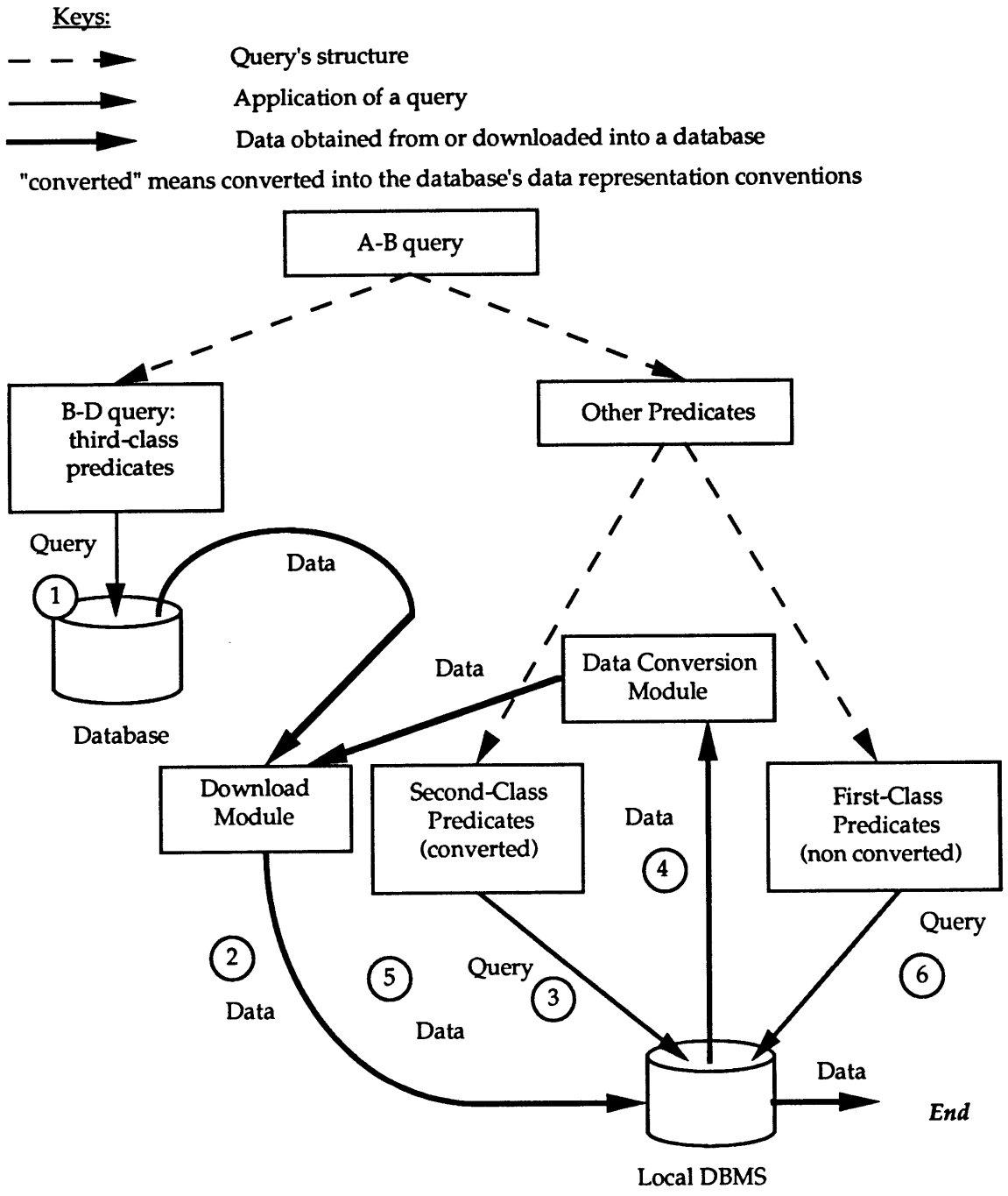

Figure 8.2: Strategy #2

Strategy #2 is composed of the 6 following steps:

Step 1 - query the database with the B-D query (recall that the B-D query is composed of third-class predicates only);

Step 2 - download the data obtained in step #1 into a local auxiliary table;[26]
Step 3 - apply to the auxiliary database the query made of the second-class (converted) predicates;
Step 4 - convert into the application's data representation conventions the data obtained in step #3.
Step 5 - download the data that was converted in step #4 into the auxiliary table; and
Step 6 - apply to the auxiliary table the query made of the first-class (non converted) predicates.

8.4 QUERY PARSING

8.4.1 Strategy #2 Optimization

As explained in the previous section, strategy #2 downloads data into the auxiliary table twice. We optimized strategy #2 by downloading into and selecting from the auxiliary table only once. Instead of creating two queries for the second- and first-class predicates, a single query combining both types of predicates is generated. This query (called `LocalQuery` on Figure 8.3 because locally applied) has the peculiarity of having the second-class predicates converted, and the first-class predicates non converted. To be able to apply this "mixed" query to the data obtained from the database, the data elements corresponding to the first-class predicates' attributes must be converted into the application's data representation conventions.

For example, if the predicate `(> ns "20000000")` has to be applied to the data set containing CO, NI, and NS data elements, then the NS data elements of the set must be converted into the application's data representation conventions (see a more detailed example in § 8.4). In doing so, the bridge converts less NI data elements than in the situation where the NS and NI data elements are converted at the same time, because the application of the predicate `(ns > "20000000")` supposedly decreases the number of records in the data set.

---

[26] In this design, the database which is created to hold the various auxiliary tables doesn't have any data per se. Once the data has been downloaded into the auxiliary table and the predicates have been applied, the table is emptied. Notice that an alternative to this is to cache the data [Tung, 1990].

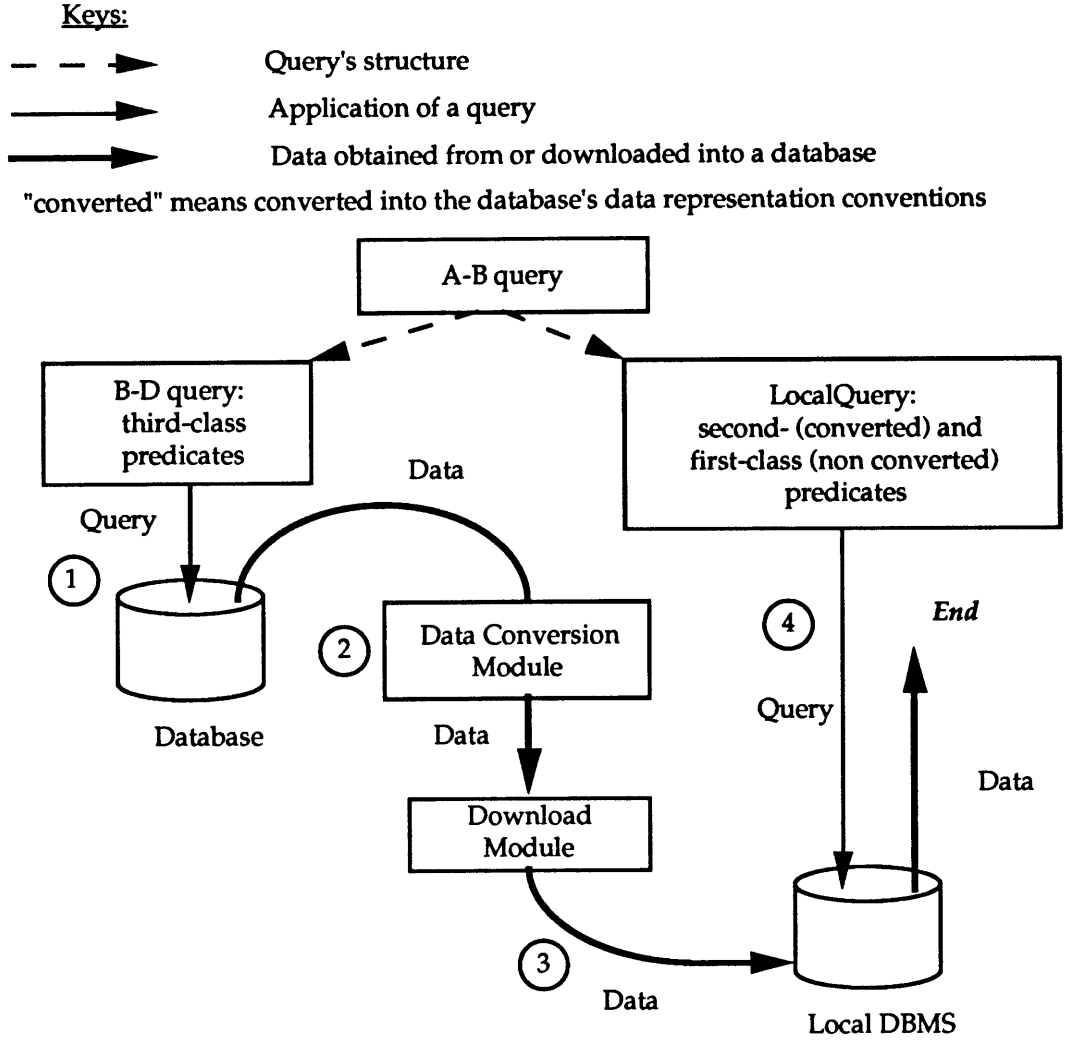

Figure 8.3: Optimized strategy #2

8.4.2 Query Parser Algorithm

8.4.2.1 Outputs

The query parser module builds two queries from the A-B query:
(1) the B-D query composed of the third-class (converted) predicates; and
(2) the `LocalQuery` composed of the second-class (converted) and first-class (non converted) predicates;

In addition to the above two queries, the query parser builds two lists of database attributes:

(1) the list of the first-class predicates' attributes (called `predicates-1`), so that we know which data elements to convert into the application's data representation conversion when obtaining the data from the database.
(2) the list of the second-class predicates' attributes (called `predicates-2`). This list plays the role of a flag to indicate the presence of second-class predicates in the initial A-B query. Thus, if `predicates-1` is empty (i.e., the initial A-B query does not contain first-class predicates), `predicates-2` is consulted to find out whether the data still needs to be downloaded (see § 8.4 for more detail).

8.4.2.2 Algorithm

The parser examines the initial A-B query's constraint to determine the class of each attribute. Second- and third-class predicates are converted into the database's data representation conventions. Second-class and first-class attributes are removed from the A-B query to build the B-D query. When a first- or second-class predicate is removed from the initial A-B query, its attribute is added to the projection list. In doing so, the `B-D` query will select data elements which will be the keys to apply the first- and second-class predicates.

The parser also assesses the information that will be needed when converting data into the application's data representation conventions. For each data type, this information is precisely contained in the `infer-arguments` and `conversion-arguments` lists. Thus, for each attribute involved in the initial A-B query (projection list and constraint), the `infer-arguments` and the `conversion-arguments` lists of the data type this attribute belongs to are determined (see Figure 8.1), and added to the projection list. Both `B-D` query and `LocalQuery` have the same projection list, that will be simply called the query's projection list. Figure 8.4 provides the flow chart of the query parsing algorithm.

8.4.2.3 Example

Let us examine the following query to illustrate the parsing of an application's request (the query is sent to the bridge acting as a virtual driver to I.P. Sharp Disclosure):

A-B query:

```
SELECT co, ni, cf FROM disclosure
WHERE
(year = "86" OR year = "87") AND
(co = "HONDA MOTOR CO" OR co = "JAGUAR PLC") AND
(ns > "20000000")
```

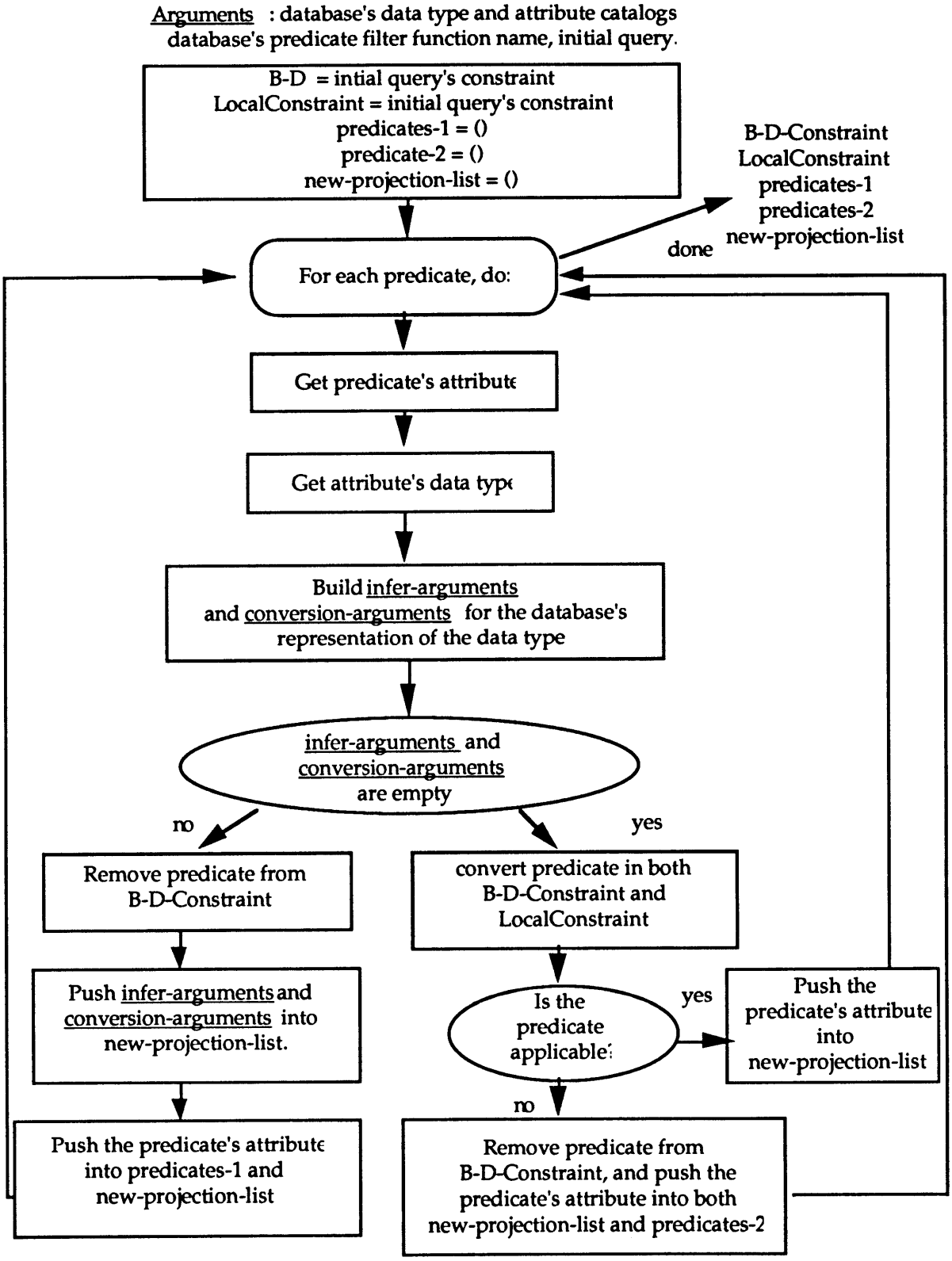

Figure 8.4: Query parsing algorithm

The application and database's data type and attribute catalogs used in this example are given below:

application-catalog			
data-type	**format**	**unit**	**scale**
money-amount	FIGURE-PLAIN	USD	SCALE-1
date	DATE-MM/DD/YYYY	NIL	NIL
year	YEAR-YY	NIL	NIL
neutral	NIL	NIL	NIL

disclosure-catalog			
data-type	**format**	**unit**	**scale**
money-amount	FIGURE-DISC	(get-unit-disc IN)	SCALE-1000
date	DATE-YYYYDDMM	NIL	NIL
year	YEAR-YYYY	NIL	NIL
neutral	NIL	NIL	NIL

disclosure-attribute-catalog	
attribute-name	**data-type**
co	neutral
sc	neutral
ni	money-amount
ns	money-amount
cf	date
year	year
ex	exchange-rate

(i) Parsing of the A-B query's constraint:

• `(year = "86")` - the predicate's attribute, i.e. YEAR, belongs to the data type `year`. The application's data representation for `year` is `[YEAR-YY]`, whereas the database's data representation is `[YEAR-YYYY]`. The conversion from `YEAR-YY` to `YEAR-YYYY` is self-sufficient, hence the predicate can be converted. Let us assume that the predicate belongs to the second class, i.e. it cannot be applied to the database. In this case, the predicate is removed from `B-D query`, and is converted into `(year = "1986")` in `LocalQuery`. The attribute YEAR is added to the projection list.

• (year = "87") - same as above.

• (co = "HONDA MOTOR CO") - the predicate's attribute, i.e. CO, belongs to the data type neutral, hence no conversion is required, and the predicate automatically belongs to the third class.

• (co = "JAGUAR PLC") - same as above.

• (ns > "20000000") - the predicate's attribute, i.e. NS, belongs to the data type money-amount. The application's representation for money-amount is [FIGURE-PLAIN; USD; SCALE-1], whereas the database's representation is [FIGURE-DISC; (get-unit-disc IN); SCALE-1000]. Infer-arguments and conversion-arguments are (IN) and (CF, EX), hence the predicate cannot be converted. Thus, the predicate is removed from B-D query but kept in LocalQuery. The attribute NS is added to the projection list.

(ii) Parsing of A-B query's projection list to assess the information needed for representation conversion:

• attribute = CO - the attribute's data type is neutral, hence infer-arguments and conversion-arguments are empty.

• attribute = NI - the attribute's data type is money-amount, and infer-arguments and conversion-arguments are given by (IN) and (CF, EX). Hence, the attributes IN, CF, and EX are added to the projection list.

• attribute = CF - the attribute's data type is date, and infer-arguments and conversion-arguments are both empty.

Finally, the parsing of the query yields:

B-D query:

```
SELECT co, ni, cf, in, ex, year, ns FROM disclosure
WHERE co = "HONDA MOTOR CO" OR co = "JAGUAR PLC"
```

LocalQuery:

```
SELECT co, ni, cf, in, ex, year, ns FROM disclosure
WHERE19
(year = "1986" OR year = "87") AND
(co = "HONDA MOTOR CO" OR co = "JAGUAR PLC") AND
AND (ns > "20000000")
```

predicates-1: (NS)

predicates-2: (YEAR)

8.5 DATA FILTERING

8.5.1 Goals

Data filtering operations occur after the data is obtained from the database. The goal of the data filter is two-fold: (1) apply `LocalQuery` to the data obtained from the database; and (2) convert into the application's data representation conventions the data obtained in step #1.

The filter follows the steps #2, #3, and #4 represented on Figure 8.2. Figure 8.3 represents the flow chart of the data filter algorithm.

8.5.2 Example

Let us proceed with the query used in § 8.3.2.3 to illustrate the data filtering. The following data set is obtained from the data base:[27]

(("CO" "NI" "CF" "IN" "EX" "YEAR" "NS")
("HONDA MOTOR CO" "146,502" "19,860,228" "JAPAN" "NY" "1985" "2,909,574")
("HONDA MOTOR CO" "83,689" "19,870,228" "JAPAN" "NY" "1986" "2,868,305")
("HONDA MOTOR CO" "56,676" "19,880,331" "JAPAN" "NY" "1987" "1,584,622")
("HONDA MOTOR CO" "N/A" "19,890,331" "JAPAN" "NY" "1988" "N/A")[28]
("JAGUAR PLC" "87,600" "19,851,231" "GREAT BRITAIN" "NY" "1985" "746,500")
("JAGUAR PLC" "83,400" "19,861,231" "GREAT BRITAIN" "NY" "1986" "830,400")
("JAGUAR PLC" "61,300" "19,871,231" "GREAT BRITAIN" "NY" "1987" "1,002,100")
("JAGUAR PLC" "28,400" "19,881,231" "GREAT BRITAIN" "NY" "1988" "1,075,500"))

Let us follow each step of the data filtering algorithm (refers to the steps represented on Figure 8.5):
Step 1 - the data elements corresponding to the attributes in the list `predicates-1` are converted into the application's data representation conventions. Since `predicates-1 = (NS)`, only the NS data elements are converted, and we obtain:

---

[27] The data was obtained from I.P. Sharp Disclosure.
[28] If data is not available for some records, the filter simply skips` them.

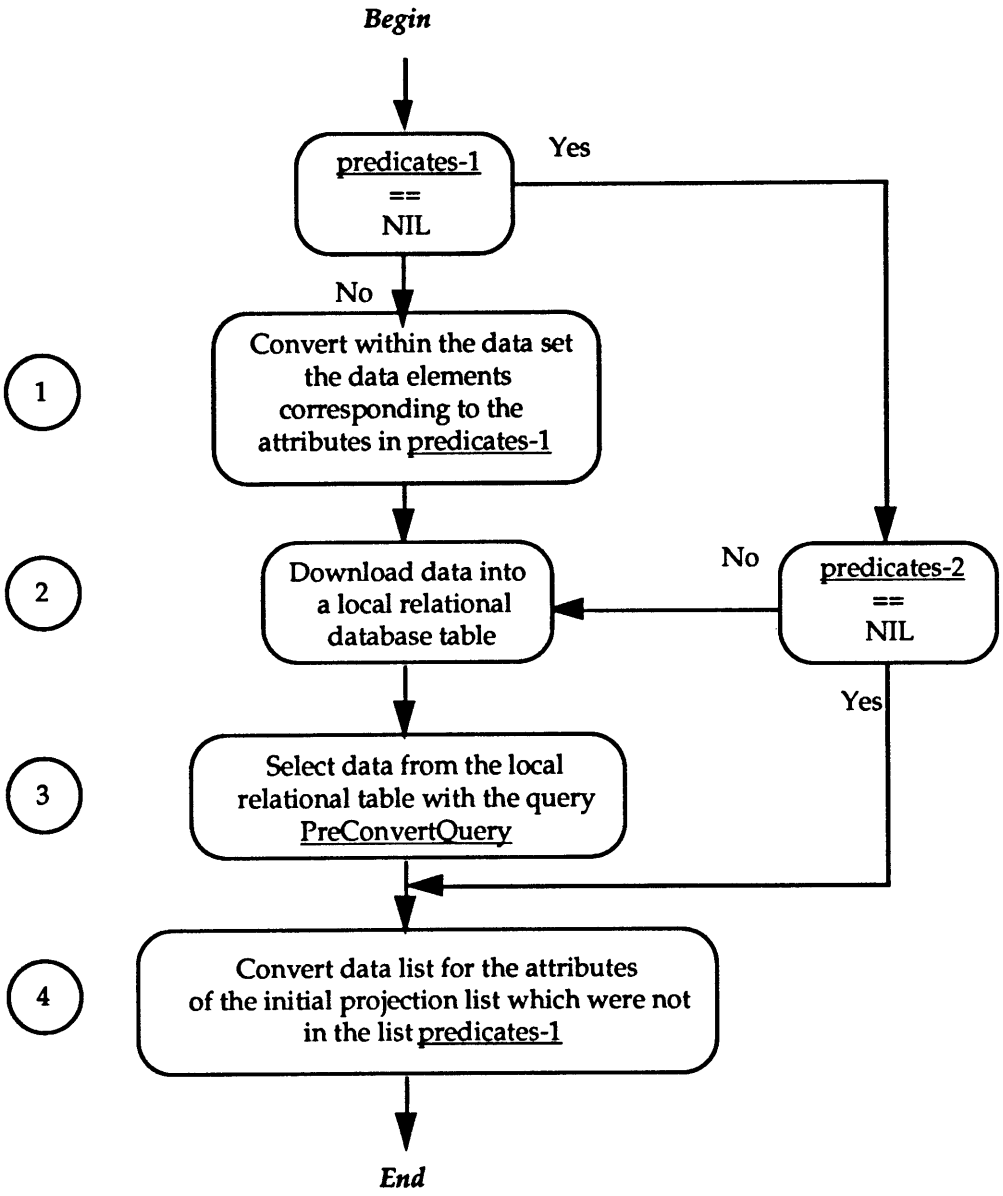

Figure 8.5: Flow chart of the data filter's algorithm

(("CO" "NI" "CF" "IN" "EX" "YEAR" "NS")
("HONDA MOTOR CO" "146,502" "19,860,228" "JAPAN" "NY" "1985" "20367018")
("HONDA MOTOR CO" "83,689" "19,870,228" "JAPAN" "NY" "1986" "20078135")
("HONDA MOTOR CO" "56,676" "19,880,331" "JAPAN" "NY" "1987" "11092354")
("HONDA MOTOR CO" "N/A" "19,890,331" "JAPAN" "NY" "1988" "N/A")
("JAGUAR PLC" "87,600" "19,851,231" "GREAT BRITAIN" "NY" "1985" "1381025000")
("JAGUAR PLC" "83,400" "19,861,231" "GREAT BRITAIN" "NY" "1986" "1536240000")
("JAGUAR PLC" "61,300" "19,871,231" "GREAT BRITAIN" "NY" "1987" "1853885000")
("JAGUAR PLC" "28,400" "19,881,231" "GREAT BRITAIN" "NY" "1988" "1989675000"))

Step 2 - the data resulting from (1) is downloaded into an auxiliary table, and `LocalQuery` is applied to this table:

```
SELECT co, ni, cf, in, ex, year, ns FROM disclosure
WHERE
(year = "1986" OR year = "1987") AND
(co = "HONDA MOTOR CO" OR co = "JAGUAR PLC") AND
(ns > "20000000")
```

The following data is thus obtained:

(("CO" "NI" "CF" "IN" "EX" "YEAR" "NS")
("HONDA MOTOR CO" "83,689" "19,870,228" "JAPAN" "NY" "1986" "20078135")
("JAGUAR PLC" "83,400" "19,861,231" "GREAT BRITAIN" "NY" "1986" "1536240000")
("JAGUAR PLC" "61,300" "19,871,231" "GREAT BRITAIN" "NY" "1987" "1853885000"))

Step 3 - the data elements corresponding to the initial projection list that were not converted in step #1 are now converted into the application's data representation conventions. Because the NS data elements only were converted in step #1 and the initial projection list is (CO NI CF), the CO, NI, CF data elements are converted.

(("CO" "NI" "CF" "IN" "EX" "YEAR" "NS")
("HONDA MOTOR CO" "597778" "02/28/1987" "JAPAN" "NY" "86" "20078135")
("JAGUAR PLC" "154290000" "12/31/1986" "GREAT BRITAIN" "NY" "86" "1536240000")
("JAGUAR PLC" "113405000" "12/31/1987" "GREAT BRITAIN" "NY" "87" "1853885000"))

Finally, only the data elements corresponding to the final projection list are returned to the application:

(("CO" "NI" "CF")
("HONDA MOTOR CO" "597778" "02/28/1987")
("JAGUAR PLC" "154290000" "12/31/1986")
("JAGUAR PLC" "113405000" "12/31/1987"))

The next section examines in detail how the data is converted.

8.5.3 Data Conversion Module

During the query parsing phase, when a predicate is removed from the initial query's constraint, both the `infer-arguments` and `conversion-arguments` lists are pushed into the query's projection list. In doing so, the information necessary to operate any future data representation conversions is available within the data set itself. Let us assume that the data set obtained from the database is in the following table-like format:

```
(("attribute-1" "attribute-2" ... "attribute-n")
("value-11" "value-12" ... "value-1n")
...
("value-p1" "value-p2" ... "value-pn"))
```

The conversion of the data set into the application's data representation conventions is operated data type by data type. From the projection list `("attribute-1" ... "attributes-n")`, the different data types involved and the corresponding attributes are identified. In the processing of the query examined in the previous section, for example, the data types involved and the corresponding attributes are the following:

data type `neutral`: attributes CO, IN

data type `money-amount`: attributes NI, NS

data type `exchange-rate`: attribute EX

data type `date`: attribute CF

data type `year`: attribute YEAR

Recall that a conversion request is in the form of a message sent to a LOPA term object, where the data item to be converted is contained in a special structure (see § 7.2.4). This structure contains the values to be converted, the representation of the data type to which the values belong, the conversion arguments' values, and the representations of the data types to which these arguments belong. This structure will be called a *data+metadata* structure. For each data type involved a *data+metadata* structure is created for each row of the data set. For the data type `money-amount`, for example, the *data+metadata* structure for the first row of the data set obtained in the example is given below:

```
((money-amount (
      (value "146,502" "2,909,574")
      (format FIGURE-DISC)
      (unit JPY)
      (scale SCALE-1000)))
(date (
      (value "19,860,228") (format DATE-YYYYMMDD)))
(exchange-market (
      (value "NY") (format INITIALS))))
```

The creation of the *data+metadata* structure is explained below. The attributes NI and NS correspond to the data type `money-amount`, hence the NI and NS data elements of the row are extracted, and we obtain: "146,502" and "2,909,574". These values are stored in the slot `value` of the *data+metadata* structure.

The values for the slots `format`, `unit`, and `scale` of `money-amount` are determined by examining the database's representation for the data type `money-amount`. In this example, the representation is `[FIGURE-DISC; (get-unit-disc IN); SCALE-1000]`.[29]

If the representation of one aspect is a function call as the above example shows, then the identity of the aspect is determined as follows:
(i) the values of the arguments of the function call are extracted from the row. With the aspect unit, for example, the data element IN is extracted from the row `("HONDA MOTOR CO" "146,502" "19,860,228" "JAPAN" "NY" "85" "2,909,574")`, hence obtaining `"JAPAN"`. The arguments' values are necessarily present in the row since the `conversion-arguments` list was added to the query projection list during the query parsing step.
(ii) the function is triggered with the arguments' values obtained from the row. The function is a bridge-specific procedure which interprets the values of its arguments. The function `get-unit-disc`, for example, "knows" that the value `"JAPAN"` is indicative of the unit LOPA term JPY (see § 7.3.2)

If the conversion of one aspect is context-dependent (i.e., requires additional arguments), then the values of these arguments are extracted from the row too. The aspect `unit` of the data type `money-amount`, for example, requires a date and an exchange rate. These two data types correspond to the local attributes CF and EX respectively, hence the values "19,860,228" and "NY" are extracted from the row `("HONDA MOTOR CO" "146,502" "19,860,228" "JAPAN" "NY" "85" "2,909,574")`. The data type representations of the conversion arguments are obtained from the database's data type catalog as well.

Once this data structure is created for each row of the data set, the conversion consists of treating one row at a time. For each row, the Algorithm[1] that we designed in §7.2.3 is used to convert the values. Figure 8.6 provides the flow chart of the complete data conversion algorithm.

---

[29] See Disclosure's data type catalog in § 7.3.

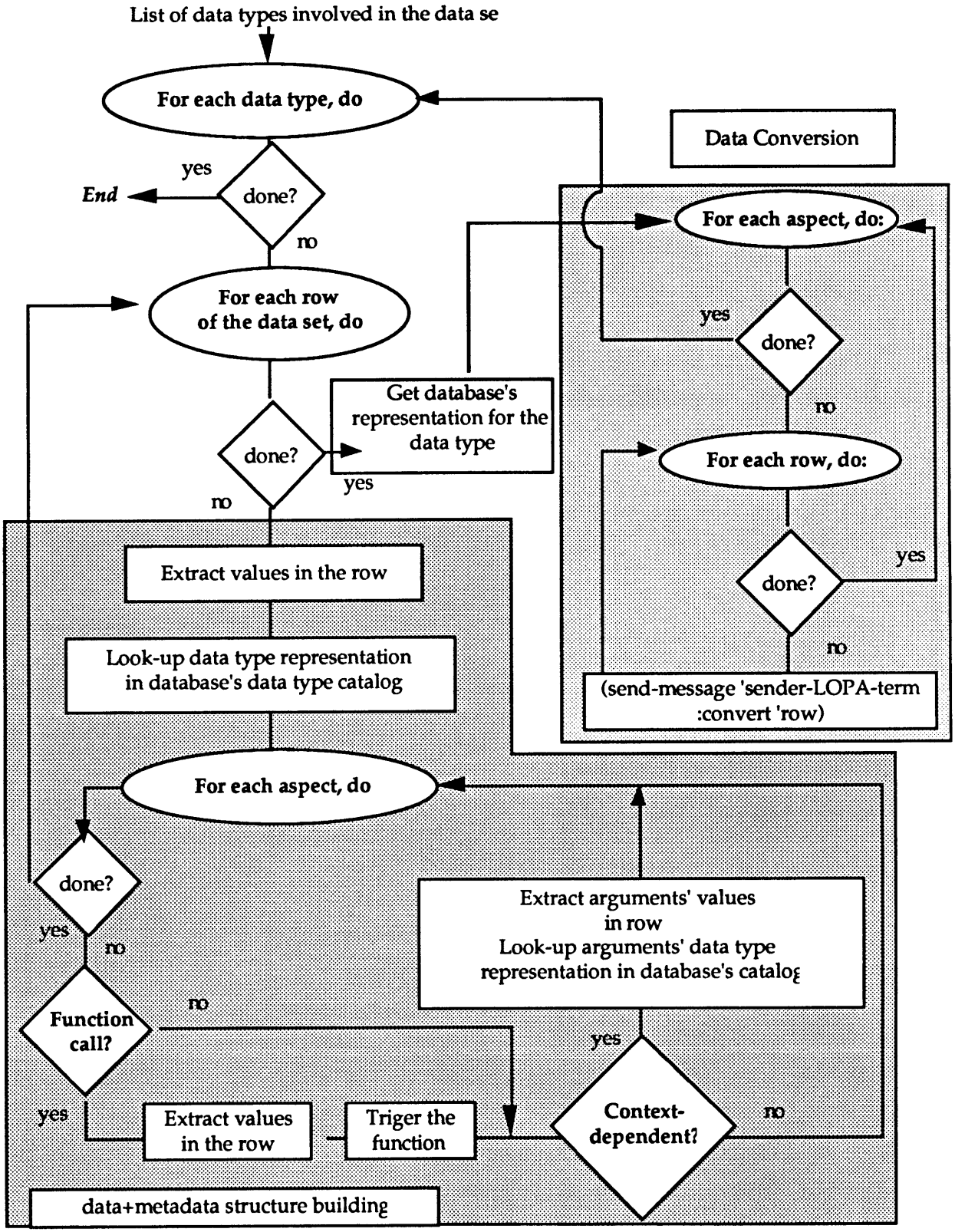

Figure 8.6: Flow chart of the data conversion algorithm

8.6 OVERALL BRIDGE ALGORITHM

We can now recapitulate the complete algorithm used to satisfy an application's request. The series of steps provided below synthesis the operations of query parsing, query's syntax translation, database access, and data filtering.

---

(1) Query Parsing:

(1.1) remove the predicates that cannot be converted;

(1.2) remove the predicates whose operators cannot be applied to the database;

(1.3) convert the second- and third-class predicates; and

(1.4) add to the projection list database attributes that will be used:

(1.3.1) in the identification of dynamic LOPA terms (inference purpose);

(1.3.2) in the conversion of certain data elements into the application's data representation conventions (conversion purpose); and

(1.3.3) in the filtering of the data set when applying the first- and second-class predicates (selection purpose).

(2) Query Syntax translation

(3) Database Access

(4) Data Filtering:

(4.1) convert into the application's representation conventions the data elements in the data set which correspond to the first-class predicates' attributes; and

(4.3) apply the first- (non converted) and second-class (converted) predicates to the data set;

(5) Return to the application the data elements which correspond to the initial projection list.

Figure 8.7 represents the flow chart of the complete algorithm.

Notice on Figure 8.7 that the data type catalog passed as argument to the bridge was entitled "sender-data-type-catalog". The term "sender" was used instead of "application" because the bridge can be used by another module than the application. A typical situation illustrating this fact is a currency conversion operation undertaken by the bridge, whereby the latter generates a data request and send it to another bridge, i.e. the one acting as a virtual driver to the currency database.

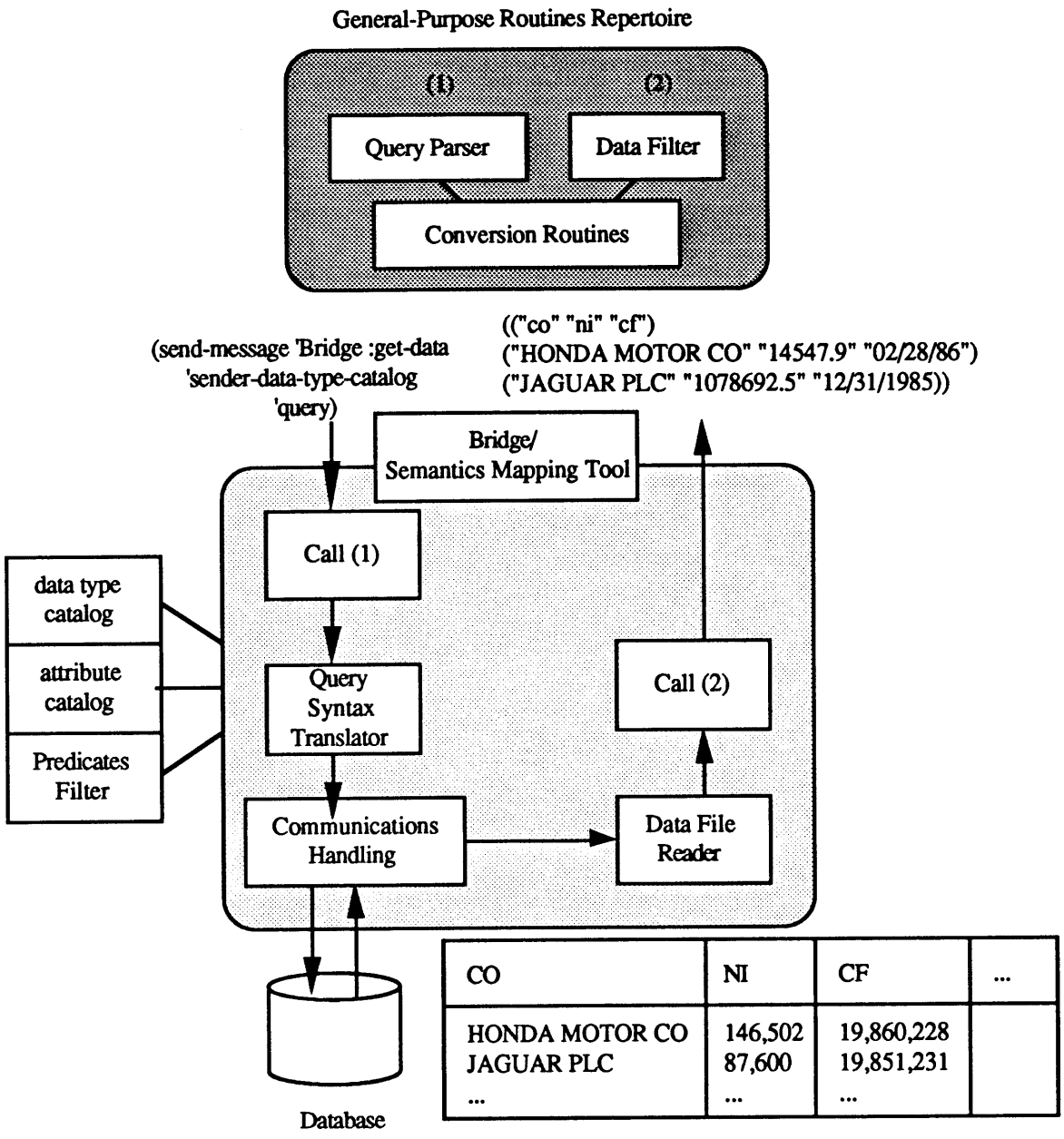

Figure 8.7: Semantics Mapping Tool

The query parser and data filter modules are general-purpose mechanisms that can be used for all bridges. Their algorithms are based on the comparison of two data type catalogs: the application and the database's, or the data request sender and the receiver's more generally speaking. These modules also use a set of conversion routines which are general-purpose routines as well. Hence, the installation of a new bridge involves the creation of a data type catalog, an attribute catalog, and the writing of the following software:

1) Query syntax translator;

2) Communications handler (e.g., communications script generator); and

3) Data file reader (i.e., page shredder).

Figure 8.8 represents the series of software layers within the application's computer system.

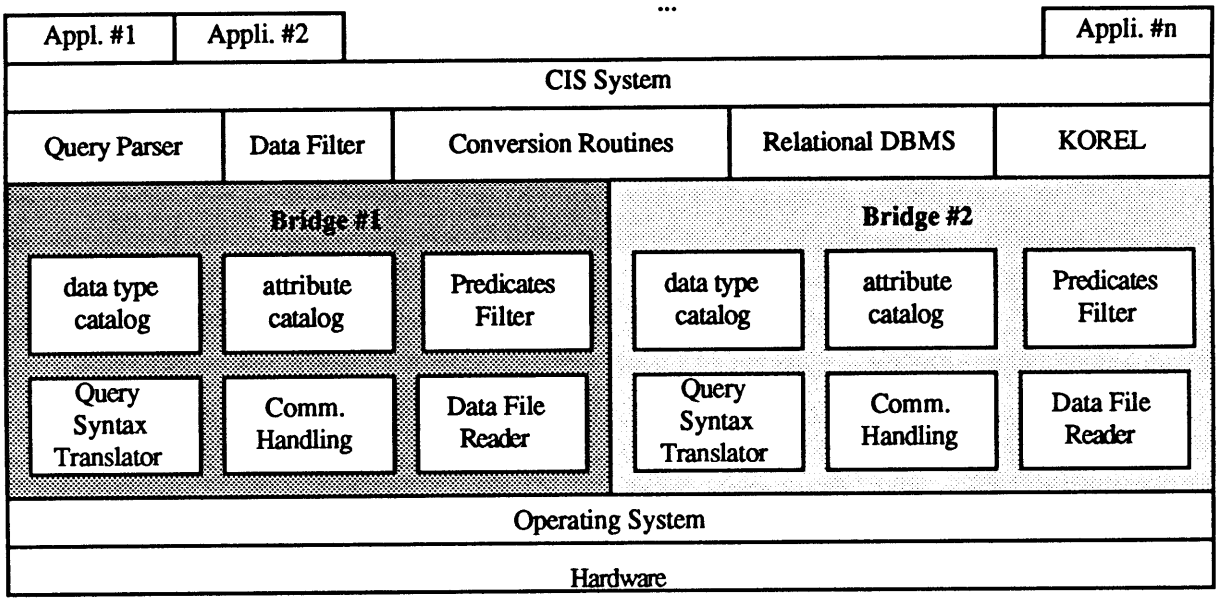

Figure 8.8: Software Layers

***

At this point, the description of the architecture and algorithms for the semantics mapping tool that we are proposing is completed. Before concluding the chapter, we shall give some additional details concerning the implementation of the tool.

8.7 IMPLEMENTATION

The architecture and algorithms that we described have been implemented on a Unix System V workstation,[30] and coded in Common Lisp. The architecture was integrated to a Composite Information System's prototype.[31] Two bridges were built: a first one for the I.P. Sharp Disclosure database, and a second one for the Finsbury Dataline database.

Two modules played an important role in the implementation of the tool: the data conversion strategy builder, and the currency conversion module. We chose to describe the

[30] Minicomputer AT&T 3b2/500.

[31] The CIS system prototype is being developed at the Composite Information Systems Laboratory (CISL), MIT, Sloan School of Management, Cambridge, USA.

module operating currency conversions in more detail because these latter represent both dynamic and context-dependent conversions. In addition, a currency conversion requires the fetching of exchange rates in a third party's computer.

8.7.1 Data Conversion Strategy Builder

When a bridge receives a data request from an application or any other sender, a special structure is built to hold information that will be used by both the query parser and the data filter. This special structure represents a simple and small-size repository holding information that is initially scattered across the conversion objects, and the application's and database's data type and attribute catalogs. Because this structure holds key information on the data representation conventions used by both the application and the database, it is called the data conversion strategy. A strategy is built for each data request received by the bridge.

The strategy holds the following information for each data type involved in the query:

(i) the application's representation;

(ii) the database's representation;

(iii) the `infer-arguments` list;

(iv) the `conversion-arguments` list;

Let consider the again the query used in § 8.4 and § 8.5:

```
SELECT co, ni, cf FROM disclosure
WHERE
(year = "86" OR year = "87") AND
(co = "HONDA MOTOR CO" OR co = "JAGUAR PLC") AND
(ns > "20000000")
```

The strategy which is generated for this query is detailed below:

```
((money-amount (
      (application-representation
            (format FIGURE-PLAIN)
            (unit USD)
            (scale SCALE-1))
      (database-representation
            (format FIGURE-DISC)
            (unit (get-unit-disc IN))
            (scale SCALE-1000))
      (infer-arguments (IN))
      (conversion-arguments ((date CF) (exchange-rate EX))))
(date ((application-representation (format DATE-MM/DD/YYYY))
      (database-representation (format DATE-YYYYDDMM))))
(year ((application-representation (format YEAR-YYYY))
      (database-representation (format YEAR-YY)))))
```

The algorithm generating a strategy is given below:

(1) Determine the data types involved in the query:
- For each attribute in the projection list, look-up its data type in the database's attribute catalog;
- Do the same for each predicate's attribute in the query's constraint;
- Return a list containing the data types involved.

(2) For each data type involved in (1), do the following:
- Look-up the application's representation in the application's data type catalog;
- Look-up the database's representation in the database's data type catalog;
- Determine the `infer-arguments` and `conversion-arguments` lists for that data type;
- Build a structure for the data type with the slots: `application-representation`, `database-representation`, `infer-arguments`, and `conversion-arguments`.

The data conversion strategy structure (simply called `strategy` herein) is used by the query parser and data filter as indicated below:

Query parser: for each data type examined, the `infer-arguments` and `conversion-arguments` lists are directly obtained from `strategy`, as opposed to building them from scratch.

Data filter: for each data type involved in the data set, the application's and database's representations for the data type are directly obtained from `strategy`, as opposed to obtaining them by sending a long series of messages to the conversion objects.

8.7.2 Currency Conversion

This section describes in detail how currency conversions are operated. We will use the same query as in § 8.3 and § 8.4 to illustrate the process. Recall that in the processing of this query, once the data set is obtained from the database, the data elements corresponding to the attribute NS must be converted into the application's representation conventions so that the first-class predicate `(> ns "20000000")` can be applied (step #1 of the data filtering algorithm on Figure 8.3). The following data was obtained from the Disclosure database:

```
(("CO" "NI" "CF" "IN" "EX" "YEAR" "NS")
("HONDA MOTOR CO" "146,502" "19,860,228" "JAPAN" "NY" "1985" "2,909,574")
("HONDA MOTOR CO" "83,689" "19,870,228" "JAPAN" "NY" "1986" "2,868,305")
("HONDA MOTOR CO" "56,676" "19,880,331" "JAPAN" "NY" "1987" "1,584,622")
("HONDA MOTOR CO" "N/A" "19,890,331" "JAPAN" "NY" "1988" "N/A")
```

("JAGUAR PLC" "87,600" "19,851,231" "GREAT BRITAIN" "NY" "1985" "746,500")
("JAGUAR PLC" "83,400" "19,861,231" "GREAT BRITAIN" "NY" "1986" "830,400")
("JAGUAR PLC" "61,300" "19,871,231" "GREAT BRITAIN" "NY" "1987" "1,002,100")
("JAGUAR PLC" "28,400" "19,881,231" "GREAT BRITAIN" "NY" "1988" "1,075,500"))

In order to convert the NS data elements, each row is transformed into a *data+metadata* structure. We thus obtain the following list of *data+metadata* structures:

(((money-amount ((value **"2,909,574"**) (format FIGURE-DISC) (unit JPY) (scale SCALE-1000))) (date ((value **"19,860,228"**) (format DATE-YYYYMMDD))) (exchange-rate ((value "NY") (format INITIALS))))
((money-amount ((value **"2,868,305"**) (format FIGURE-DISC) (unit JPY) (scale SCALE-1000))) (date ((value **"19,870,228"**) (format DATE-YYYYMMDD))) (exchange-rate ((value "NY") (format INITIALS))))
((money-amount ((value **"1,584,622"**) (format FIGURE-DISC) (unit JPY) (scale SCALE-1000))) (date ((value **"19,880,331"**) (format DATE-YYYYMMDD))) (exchange-rate ((value "NY") (format INITIALS))))
((money-amount ((value **"746,500"**) (format FIGURE-DISC) (unit GBP) (scale SCALE-1000))) (date ((value **"19,851,231"**) (format DATE-YYYYMMDD))) (exchange-rate ((value "NY") (format INITIALS))))
((money-amount ((value **"830,400"**) (format FIGURE-DISC) (unit GBP) (scale SCALE-1000))) (date ((value **"19,861,231"**) (format DATE-YYYYMMDD))) (exchange-rate ((value "NY") (format INITIALS))))
((money-amount ((value **"1,002,100"**) (format FIGURE-DISC) (unit GBP) (scale SCALE-1000))) (date ((value **"19,871,231"**) (format DATE-YYYYMMDD))) (exchange-rate ((value "NY") (format INITIALS))))
((money-amount ((value **"1,075,500"**) (format FIGURE-DISC) (unit GBP) (scale SCALE-1000))) (date ((value **"19,881,231"**) (format DATE-YYYYMMDD))) (exchange-rate ((value "NY") (format INITIALS))))

According to our conversion algorithm (see Algorithm[1] in § 7.2.3), each aspect of a `money-amount` representation must be converted separately. Thus, the `format`, `unit`, and then `scale` must be converted in sequence. let us assume that the format conversion has been operated and that our list of *data+metadata* structures is given by:

(((money-amount ((value **"2909574"**) (format FIGURE-PLAIN) (unit JPY) (scale SCALE-1000))) (date ((value "19,860,228") (format DATE-YYYYMMDD))) (exchange-rate ((value "NY") (format INITIALS))))
((money-amount ((value **"2868305"**) (format FIGURE-PLAIN) (unit JPY) (scale SCALE-1000))) (date ((value "19,870,228") (format DATE-YYYYMMDD))) (exchange-rate ((value "NY") (format INITIALS))))
((money-amount ((value **"1584622"**) (format FIGURE-PLAIN) (unit JPY) (scale SCALE-1000))) (date ((value "19,880,331") (format DATE-YYYYMMDD))) (exchange-rate ((value "NY") (format INITIALS))))
((money-amount ((value **"746500"**) (format FIGURE-PLAIN) (unit GBP) (scale SCALE-1000))) (date ((value "19,851,231") (format DATE-YYYYMMDD))) (exchange-rate ((value "NY") (format INITIALS))))

((money-amount ((value "830400") (format FIGURE-PLAIN) (unit GBP) (scale SCALE-1000))) (date ((value "19,861,231") (format DATE-YYYYMMDD))) (exchange-rate ((value "NY") (format INITIALS))))
((money-amount ((value "1002100") (format FIGURE-PLAIN) (unit GBP) (scale SCALE-1000))) (date ((value "19,871,231") (format DATE-YYYYMMDD))) (exchange-rate ((value "NY") (format INITIALS))))
((money-amount ((value "1075500") (format FIGURE-PLAIN) (unit GBP) (scale SCALE-1000))) (date ((value "19,881,231") (format DATE-YYYYMMDD))) (exchange-rate ((value "NY") (format INITIALS))))

From the above list of *data+metadata* structures, the following query requesting exchange rates is built, and sent to the semantics mapping tool which acts as the virtual driver to the currency database:

```
SELECT exchange-rate FROM currency
WHERE
(date = "19,860,228" AND code = "JPY") OR
(date = "19,870,228" AND code = "JPY") OR
(date = "19,880,331" AND code = "JPY") OR
(date = "19,851,231" AND code = "GBP") OR
(date = "19,861,231" AND code = "GBP") OR
(date = "19,871,231" AND code = "GBP") OR
(date = "19,881,231" AND code = "GBP")
```

The data type catalog provided below will be used as the catalog which describes the query's intended data representation conventions. Notice that we used the notation "sender-catalog" because the query was not generated by the application, but by the USD object itself.

sender-catalog			
data-type	**format**	**unit**	**scale**
money-amount	FIGURE-PLAIN	USD	SCALE-1
date	DATE-YYYYMMDD	NIL	NIL
year	YEAR-YYYY	NIL	NIL
exchange-market	INITIALS	NIL	NIL
currency-code	ISO	NIL	NIL
neutral	NIL	NIL	NIL

Let us assume that the data type and attribute catalogs for the currency database are given by:[32]

[32] The currency database catalogs are modelled after the I.P. Sharp Currency database.

currency-data-type-catalog			
data-type	**format**	**unit**	**scale**
money-amount	FIGURE-PLAIN	USD	SCALE-0.01
date	DATE-DD-MM-YY	NIL	NIL
currency-code	ISO	NIL	NIL
neutral	NIL	NIL	NIL

currency-attribute-catalog	
database-attribute	**data-type**
code	exchange-rate
date	date
exchange-rate	money-amount

The following data request is thus sent to the currency data base bridge through the message:

```
(send-message 'Currency :get-data 'sender-catalog 'query)
```

We show below the different steps in the processing of this data request, i.e. strategy building, query parsing, query syntax translation, data file reading, and data filtering.

(1) Strategy Building

```
((money-amount (
      (application-representation
            ((format FIGURE-PLAIN)
            (unit USD)
            (scale SCALE-1)))
      (database-representation
            ((format FIGURE-PLAIN)
            (unit USD)
            (scale SCALE-0.01)))))
(date (
      (application-representation ((format DATE-YYYYMMDD)))
      (database-representation ((format DATE-DD-MM-YY)))))
(currency-code (
      (application-representation ((format ISO)))
      (database-representation ((format CURRENCY-IPSHARP))))))
```

(2) Query Parsing

B-D query :

```
SELECT exchange-rate FROM currency
WHERE
(date = "28-02-86" AND code = "JPY") OR
(date = "28-02-87" AND code = "JPY") OR
(date = "31-03-88" AND code = "JPY") OR
(date = "31-12-85" AND code = "GBP") OR
(date = "31-12-86" AND code = "GBP") OR
(date = "31-12-87" AND code = "GBP") OR
(date = "31-12-88" AND code = "GBP")
```

LocalQuery : same as above.

predicates-1 and predicates-2 are empty.

(3) Query Syntax Translation

1 2 3 4 5 DAILY DATED AT 28 02 86 TO 28 02 86
CURRENCY 'NJPY'
1 2 3 4 5 DAILY DATED AT 28 02 87 TO 28 02 87
CURRENCY 'NJPY'
1 2 3 4 5 DAILY DATED AT 31 03 88 TO 31 03 88
CURRENCY 'NJPY'
1 2 3 4 5 DAILY DATED AT 31 12 85 TO 31 12 85
CURRENCY 'NGBP'
1 2 3 4 5 DAILY DATED AT 31 12 86 TO 31 12 86
CURRENCY 'NGBP'
1 2 3 4 5 DAILY DATED AT 31 12 87 TO 31 12 87
CURRENCY 'NGBP'
1 2 3 4 5 DAILY DATED AT 31 12 88 TO 31 12 88
CURRENCY 'NGBP'

(4) Data File Reading

```
(("exchange-rate") ("0.5") ("0.6") ("0.8") ("144.5") ("188.6")
("189") ("190"))
```

(5) Data Filtering

Only a scaling factor conversion is necessary; the initial exchange rates request thus receives the following list of exchange rates:

```
(("exchange-rate") ("0.005") ("0.006") ("0.008") ("1.445") ("1.886")
("1.89") ("1.90"))
```

We can now go back to the currency conversion operation per se, and multiply each NS data element with the appropriate exchange rate as shown below:

"2909574" by "0.005" yields "14547.9"
"2868305" by "0.006" yields "17209.83"
"1584622" by 0.008" yields "12676.9"
"746500" by "1.445 yields "1078692.5"

"830400" by "1.886" yields "1566134.4"
"1002100" by "1.89" yields "1993969"
"1075500" by "1.90" yields "2043450"

Finally, the conversion routine returns:

(("CO" "NI" "CF" "IN" "EX" "YEAR" "NS")
("HONDA MOTOR CO" "146,502" "19,860,228" "JAPAN" "NY" "85" "14547.9")
("HONDA MOTOR CO" "83,689" "19,870,228" "JAPAN" "NY" "86" "17209.83")
("HONDA MOTOR CO" "56,676" "19,880,331" "JAPAN" "NY" "87" "12676.9")
("HONDA MOTOR CO" "N/A" "19,890,331" "JAPAN" "NY" "88" "N/A")
("JAGUAR PLC" "87,600" "19,851,231" "GREAT BRITAIN" "NY" "85" "1078692.5")
("JAGUAR PLC" "83,400" "19,861,231" "GREAT BRITAIN" "NY" "86" "1566134.4")
("JAGUAR PLC" "61,300" "19,871,231" "GREAT BRITAIN" "NY" "87" "1993969")
("JAGUAR PLC" "28,400" "19,881,231" "GREAT BRITAIN" "NY" "88" "2043450"))

***

This concludes the presentation of the design of the semantics mapping tool. Appendix[1] provides a sample session showing the different steps followed by the tool to satisfy an application's request, and appendix[2] provides the code (in Common Lisp) which was written to implement the tool.

8.8 CONCLUSION & FUTURE RESEARCH

8.8.1 Design Summary

Chapters VI, VII, and VIII have presented the design of a data semantics mapping tool. This tool was a key component of a set of recommendations that we articulated in the conclusion of chapter V on how to gain information systems connectivity in an organization.

The tool is designed to support two essential functionalities:

1. Data Representation Transparency, whereby an application can express data requests based on data representation conventions that may differ from the conventions used in the database to which the requests are applied. Thus, the tool's first functionality consists of performing data representation mapping between the application and the database.
2. Selection Operator Transparency, whereby an application can express data requests based on selection operators that may not be supported by the database to which the requests are applied. Thus, the tool's second functionality consists of performing further data selection that the database cannot perform.

The architecture that was designed to support the above two functionalities is based on catalogs that hold information on the representation of data. The data being exchanged between the application and the database are classified into data types. A representation of a data type is defined by a series of permitted terms, each one designating a specific aspect of the representation.

One data type catalog is created for the application in order to describes its own data representation conventions. One catalog is also created for each database that the application needs to access, and describes the database's data representation conventions.

The architecture handles situations where the representations of certain data types are not constant across the records in the database. For such a representation, that we called dynamic in the text, we designed a mechanism to infer the dynamic aspects of this representation from the data returns by the request. This mechanism adds additional database attributes to the initial query's projection list, so that the values returned for these attributes can be used to infer the identities of the dynamic aspects. The specific database attributes that are used in this process are stored in the database's data type catalog .

The architecture also handles situations where the conversion of the representation of a given data type depends on the context. We designed a mechanism which provides the contextual information that this type of conversion needs. The mechanism also adds additional database attributes to the initial query's projection list. Then, the values returned for these attribute constitute the needed contextual information. The database attributes that represent contextual information are stored in a collection of objects that are accessible by the bridge through specific methods.

Finally, the overall bridge's algorithm is the following:

1. Query Parsing;
2. Query Syntax Translation;
3. Database Access;
4. Page Shredding; and
5. Data Filtering.

The query parser and data filter constitute the core of the data representation conventions reconciliation. The query parser examines the query in order to determine:

- which predicates can or cannot be applied to the database; and
- the information that will be needed in order to convert the data obtained from the database.

The data filter applies to the data the predicates that could not be applied to the database. The filter also sends conversion requests to the appropriate LOPA term objects in order to convert the data into the application's data representation conventions.

8.8.2 Tool's Enhancement

(i) Query enhancement

The application's request given to a bridge can be enhanced by directly including data representation information in it. Consider the example given below:

```
SELECT CO, NI [FIGURE-PLAIN; USD; SCALE-1000], CF [DD-MM-YY]
FROM Disclosure
WHERE
(CO = "HONDA MOTOR CO") AND (NS > "20000000")
```

The above query indicates for some attributes their intended data representation. For an attribute present in the query's projection list, the included data representation information indicates in which representation conventions the application expects to receive the data. For an attribute present in a selection predicate, the included data representation information indicates the representation conventions pertaining to the predicate's value. For an attribute whose representation is not indicated, the application's catalog can be consulted. This later thus describes the default data representation settings.

This new query structure provides more flexibility to the application when formulating a request, and also saves time since the application does not have to modify its data type catalog each time it wants to change its data representation conventions.

(ii) Currency conversion

The handling of currency conversions can be enhanced by providing the bridge with a choice in how to operate them. The following choices on which exchange rate to use could, for example, be considered:

Choice #1 - Today's (or most recent one)

Choice #2 - Financial reporting date's

In the current design we are not giving any choice to the bridge which adopts the second choice automatically. The query could be further enhanced by giving the application the ability to indicate its choice of the currency conversion method -- if no choice is indicated, then a default method is adopted. The exchange market is another parameter of a currency conversion operation that must be determined in the process. The application's request could also specify an exchange market, or else a default setting is used.

A last enhancement to the currency conversion module concerns the creation of a local auxiliary table to store the various exchange rates that are obtained over time. Data is not free, plus it is not going to change over time, hence the tool should keep the exchange rates around and use them for future conversions.

(iii) Conversion Optimizer

The number of steps involved in the conversion of a data element (see Algorithm[1] in § 7.2.2.3) can be minimized through the intervention of certain rules as we explain below.

A unit conversion, for example, converts the data element's format into FIGURE-PLAIN since the transformation of a string into the corresponding float number is easier if the string's format is FIGURE-PLAIN. The same pre-conversion occurs for a scale conversion. Therefore, it appears that the order in which the aspects of a representation are converted does matter. For the data type money-amount, for example, the unit and scale aspects should be converted prior to the conversion of the format aspect. In addition to the importance of this order, the conversion of one aspect is not independent from the conversion of a second aspect. Consider the case of a scale conversion following a unit conversion; since the scale conversion also has to generate a float number from the data element's string, the unit conversion should output the data element as a float number as opposed to as a string.

This set of rules could be captured in a knowledge base maintained by the bridge. This knowledge base would be a separate module within the bridge's architecture, so that the bridge could function without it. Separating the knowledge base from the bridge's kernel would facilitate extendability since new data types and aspects could be created without having to update the knowledge base immediately. Thus, the bridge developers can discover ways of optimizing conversions along the way and gradually extend the knowledge base.

The enhancement of the currency conversion module as well as the building of the conversion optimizer represent relatively easy tasks given the architectural foundation that was laid down in this thesis.

8.8.3 Future Research

The tool that we designed constitutes a foundation on which other data semantics resolvers can be built. Many-to-many mapping tools, rule-based translation, and attribute integration and disintegration are examples of mechanisms whose development can be facilitated by starting from the foundation laid down in this thesis. Many-to-many mappings were described in chapter VII. A rule-based translation mechanism is used to convert data elements in situations requiring a significant amount of knowledge. A example of such a situation is the translation of a yearly salary into a qualitative appreciation of this salary, such as *poor, average, competitive,* and *highly competitive.* Such a translation is contingent to several interrelated parameters, such as the industry type, location, and job position. Attribute integration and disintegration consists of converting one data element into two data elements and vice versa. The development of the data semantics resolvers mentioned above constitute our next step, and we believe that a bridge empowered by this repertoire of techniques can deal with the majority of logical connectivity issues.

-oOo-

Chapter IX.

CONCLUSION

9.1 PROBLEM

Large decentralized organizations are currently facing three potentially conflicting and complicating forces: autonomy, integration, and evolution. Most decentralized organizations have developed a culture of sub-units autonomy so that the sub-units can be highly responsive and innovative in regard to the local needs to which they are trying to respond. The force of integration may conflict with the force of autonomy in some occasions. Integration emerges as one solution for achieving better coordination among the various components of a decentralized organization. The strategic goals of globalization, risk management, time to market, service, and cost all rely on the ability to manage the interdependencies among organizational units. The force of evolution further complicates the potential conflict between autonomy and integration. This third force takes its source in the inherent volatility of the world in which organizations live. Due to this volatility, organizations have to evolve in a continuous manner in order to adjust to new opportunities and threats.

Information Technology is a key tool in an integration endeavour. This can be explained by remarkable improvements that have occurred in the fields of information-sharing and communications capabilities, both on a cost and performance point of views. Information Technology thus allows organizations to gain connectivity among the computer systems scattered across functions and geography.

However, tying computer systems together turns out to be technically and politically arduous. Technically, the presence of disparate information systems which are often

incompatible seriously hinders the integration effort. Politically, the presence in most decentralized organizations of a strong corporate culture favoring autonomy and local entrepreneurship conflicts with the desire that top management may have to integrate.

9.2 THESIS GOAL

The goal of the thesis was to provide recommendations to the top executives of a decentralized organization for the implementation of an integration project. We have aimed at designing an integration mechanism which respects the fundamental organizational forces of autonomy and evolution.

In order to reach our goal, we reviewed the existing connectivity approaches that have been documented in the literature and tested by various companies in the past. Four connectivity approaches were examined: Enterprise Modelling, Data Resource Management, Composite Information Systems, and Manual Bridges. We classified these approaches into two categories: proactive, and reactive approaches. Enterprise Modeling and Data Resource Management were considered as proactive, with Enterprise Modeling representing the most extreme of the two. Manual Bridge and Composite Information System were considered as reactive, with Manual Bridge representing the most extreme of the two.

9.3 ANALYSIS

The examination and comparison of the four connectivity approaches has led to the following conclusions:

(i) Proactive approaches

• Enterprise Modelling is a good exercise for an organization because it improves understanding of the business and the information requirements for conducting business in good conditions. However, the implementation phase of the approach is often too disruptive, too time-consuming, and too expensive to be feasible.

• Data Resource Management, like Enterprise Modelling, advocates an organization-wide standardization; in this case, the standardization of data element definitions. DRM envisages the standardization of the data element definitions given the present information systems architecture, hence the approach is less disrupting than Enterprise Modelling. However, DRM faces many of the same difficulties as Enterprise Modelling. These difficulties are inherent to the nature of a company-wide standardization endeavour: consensus is difficult to reach, it is

time-consuming and expensive, and the project is contradictory with the volatile nature of the environment in which the organization oughts to live.

(ii) Reactive approaches

• Manual Bridge is based on a totally different philosophy from Enterprise Modelling and DRM's philosophy. Manual Bridge refuses to embark on any standardization project, and thus lets each sub-unit deploy the IT it believes will help conducting the business. Then, when there is a need to integrate data across organizational units, an ad hoc agreement is established among the sub-units involved. Such an agreement typically binds an organizational unit to build a bridge to receive or send data in a particular semantics whose definition is also specified in the agreement. Manual Bridge has the advantage of being non-disruptive and a cheap (and dirty) solution to a connectivity requirements. However, the approach usually cannot handle complex connectivity requirements, or applications handling a large volume of transactions, reinvents the wheel rather than gains experience each time a new bridge is built, and is typically characterized by poor documentation.

• Composite Information Systems is based on the same philosophy of *laissez-faire* than Manual Bridge. What CIS does add to Manual Bridge, however, is a particular technology to integrate information systems. CIS thus provides general-purpose architectures for the fetching of data through bridges as well as for the integration of data. Through the use of CIS technology an organization can create economies of scale.

The CIS approach appears to be the only economically feasible solution if the organization has already heavily invested in computer systems, and if it is necessary for the integration to occur quickly and provide acceptable performances. However, the building of a general-purpose architecture for a bridge turns out to be technically difficult, which may significantly hamper the implementation of the CIS system.

Three important lessons can be learned from the above analysis:

Lesson #1

The presence of standards (on the hardware, software, or data element definitions) greatly facilitates information-sharing, communications and integration. A reference architecture that the Enterprise Modelling and Data Resource Management aim at building improves business responsiveness. Price, discount, and tax tables, for example, can be altered at one location and then propagated throughout the entire company. New business can thus be easily added and changes in market conditions can be more easily accommodated.

Lesson #2

Company-wide standardization is not a desirable solution, even if it were to be economically and politically feasible. The nature of the world is by essence diverse and will remain so. Any effort to suddenly rationalize and standardize an environment is thus doomed to failure. Corporations are not going to change, for example, their accounting procedures overnight. Each corporation's administrative heritage creates a strong inertia which has to be taken into consideration and with which any standardization impetus has to deal.

Lesson #3

Data semantics mapping is particularly difficult to perform. Focusing on the definition of the data elements used in an organization and broadcasting these definitions throughout all sub-units will significantly leverage the building of bridges and the implementation of CIS systems.

Based upon the above lessons, we have recommended the following connectivity approach called *Focused Standards*:

(i) adopt the CIS methodology;
(ii) standardize the definition of a limited number of critical corporate data elements; and
(iii) develop data semantics mapping techniques.

The Focused Standards' rationale is that if the organization builds computer systems allowing units to share some functions, such as customer billing calculations, without sharing others, such as customer service, then each organizational unit could have the flexibility to meet its own needs while gaining shared system economies.

9.4 HAS THE GOAL BEEN REACHED?

The Focused Standards approach respects the three organizational forces of integration, autonomy, and evolution. To that extent, we have reached our goal. We explain below how the approach deals with each force:

Integration - the approach adopts the CIS methodology, hence it will allow quick integrations. The approach is prototype-oriented and incremental, encourages learning by doing, and allows quick delivery.

Autonomy - autonomy is respected by both the data elements selection process and the standards enforcement policy.

The selection process of the data elements whose definitions will be standardized involves the participation of the sub-units. Each sub-unit is responsible for selecting and drafting the definition of the data elements pertaining to its local business. This selection process minimizes the sub-units' resistance since they will themselves define the data elements that they need.

The sender-to-receiver contract-based enforcement policy that we recommended is particularly flexible. With the aid of data semantics mapping techniques that were elaborated in chapters VI, VII, and VIII, an organizational unit is given complete freedom as to how to adopt the standards. Such an enforcement mechanism fosters connectivity while maintaining autonomy: as long as the departments build bridges to send and receive data in their standardized form, then the organization can maintain multiple definitions.

Evolution - the approach takes evolution into consideration in two respects. First, it recommends the establishment of a permanent committee gathering representatives from the main organizational units whose role is to review the propositions of standards made by the sub-units. Hence, the committee evolves with the organization's information requirements and elaborates standards in concordance. Second, the semantics mapping tool whose design has been documented in this thesis allows a computer system to gradually adopt the standards and adjust to modifications on already existing standards.

The semantics mapping tool is key in the implementation of the Focused Standards approach. It allows organizational units to exchange data based on definitions which differ from the ones used internally to each sub-unit. Not only does the tool allow the exchange of data based on different definitions, but also it allows it overnight.

We designed a general-purpose semantics mapping tool since it can be used for any type of application providing a minimum effort of installation and customization. However, optimization in terms of amount of processing and execution time was not my primary concerns.

A semantics mapping tool is a "biodegradable" piece of technology. As the computer system adopts the standard definitions of more and more data elements, the amount of processing that the tool must carry out decreases. Eventually, the tool reaches a minimum size for the sole purpose of receiving data from sources external to the organization. Figure 9.1 next page represents the evolution of a semantics mapping tool over time.

9.5 CONCLUDING REMARKS

The combination of the Composite Information Systems and Focused Standards approaches constitutes a technically, economically, and politically feasible solution for a decentralized organization willing to tie its information systems together. It short-term goal is to deliver an integrated computing environment quickly and allows the organization to implement strategic applications in an ad hoc manner. The combination of the two approaches also has the long-term objective to gradually shift the organization's information systems architecture towards a more standardized IS base. Therefore, the approach that is recommended in this thesis does not only fix the problem "at the end of the pipe", but also prepares for the future computing needs of the organization.

- FINIS -

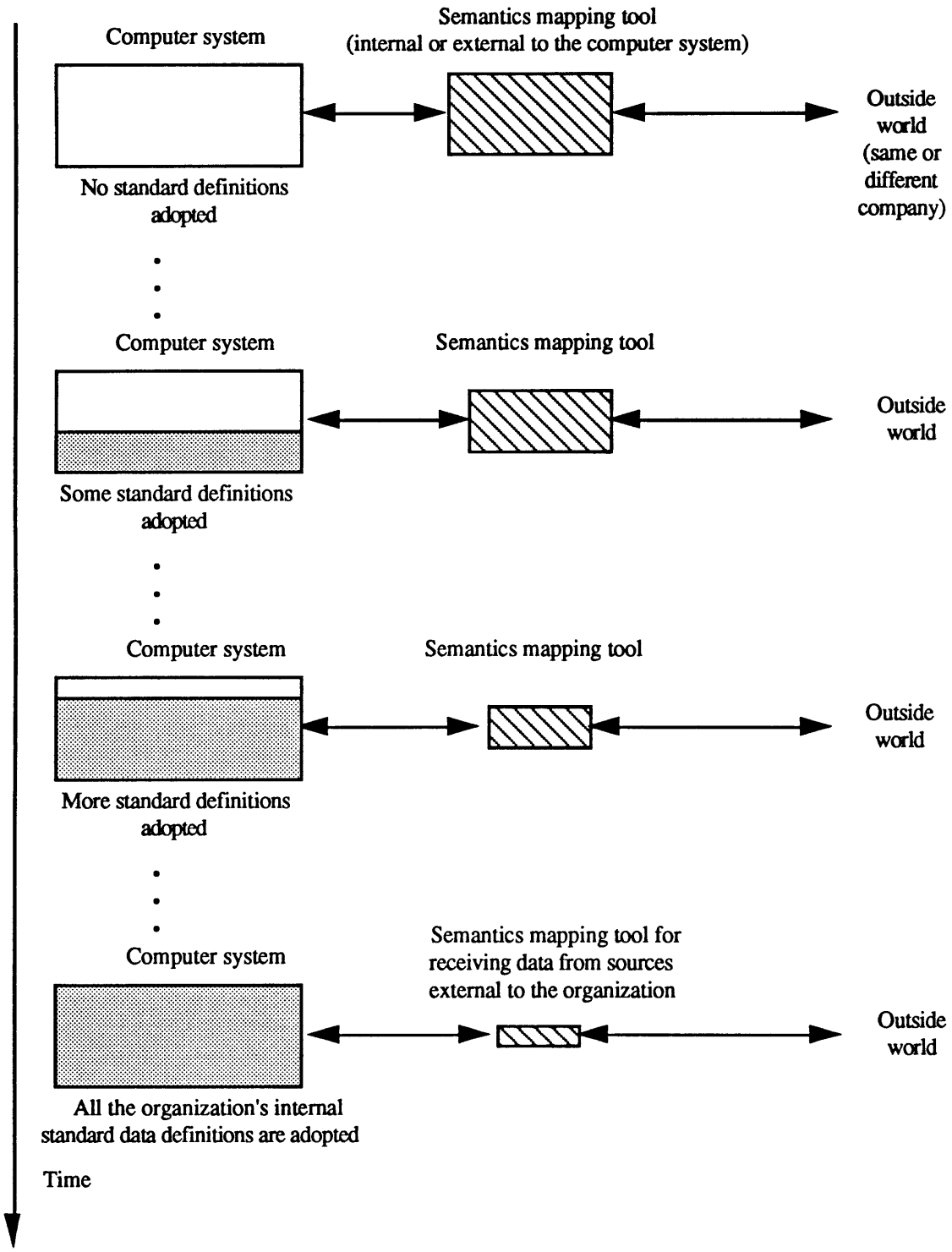

Figure 9.1: Evolution of a semantics mapping tool over time

REFERENCES

Barrett, S., and Konsynski, B. (1982) "Inter-Organizational Information Sharing Systems." MIS Quarterly/Special Issue, pp. 93-105.

Bartlett, A. C., and Ghoshal, S. (1988), "Organizing for Worldwide Effectiveness: The Transnational Solution." California Management Review, Volume 31, Number 1, Winner of the Third Annual Pacific Telesis Foundation Award.

Benjamin, R., and Scott Morton, M. (April 1986), "Information Technology, Integration, and Organizational Change." CISR Working Paper No. 138, Sloan School Working Paper No. 1769-86, 90's Working Paper No. 86-017, Sloan School of Management, Massachusetts Institute of technology.

Best, L. (26 March 1990), "Clearing the Dusty Decks." Computerworld, pp. 97-100.

Betts, M. (26 March 1990), "Going With Globalization Thrust." Computerworld, pp. 105-108.

Bhalla, S., Prasad, B., Gupta, A., and Madnick, S. (1987) "A framework and Comparative Study of Heterogeneous Distributed Database Management Systems." in Knowledge-Based Integrated Information System Engineering (KBIISE) Project, Volume 3. Ed. Amar Gupta and Stuart Madnick. Cambridge: Massachusetts Institute of Technology.

Brooks, R. (1985), Programming in Common Lisp. New York : John Wiley & Sons.

Bullen, C., and Johansen, R. (May1988), "Groupware: A Key to Managing Business Teams." CISR Working Paper No. 169, Sloan School Working Paper No. 2013-88, Sloan School of Management, Massachusetts Institute of technology.

Champlin, A. (May 1988), "Interfacing Multiple Remote Databases in an Object-Oriented Framework." B.S. Thesis. Massachusetts Institute of Technology.

Date, J. (1986) An Introduction to Databases Systems. Volume I, Fourth Edition. Reading Massachusetts: Addison-Wesley.

Dumaine, B. (13 February 1990), "How Managers Can Succeed Through Speed." Fortune, p. 54.

Drucker, P. (January-February 1988), "The Coming of the New Organization." Harvard Business Review.

Folinus, J., Madnick, S., and Schutzman, H. (July 1974), "Virtual Information In Data-Base Systems." Sloan School of Management Working Paper, No. 723-74, CISR Working Paper, No. 3.

Goodhue, D. L., Quillard, J. A., Rockart, J. F. (January 1987), "Managing The Data Resource: A Contingency Perspective." CISR Working Paper No. 150, Sloan School Working Paper No. 1859-87, 90s Working Paper No. 87-032.

Gref, W. (1987), "Distributed Homogeneous Database Systems: A Comparison Between Oracle and Ingres." in Knowledge-Based Integrated Information System Engineering (KBIISE) Project,

Volume 3. Ed. Amar Gupta and Stuart Madnick. Cambridge: Massachusetts Institute of Technology.

Hubert, G. (August 1984), "The Nature and Design of Post-Indutrial Organizations." Management Science, Vol. 30, No. 8.

Johnson, L. (May 1985), "Managing Data as a Corporate Resource: a Case Study." S.M. Thesis, Sloan School of Management, Massachuestts Institute of Technology.

Kahn, B. (October 1983), "Some Realities of Data Administration." Communications of the ACM, Vol. 26, No. 10.

Kahn, B., Lumsden, E. (Fall 1983), "A User-Oriented Framework for Data Dictionary Systems." Data Base.

Keen, P. (January 1981), "Information Systems and Organizational Change." Communications of the ACM, Volume 24, No. 1.

Kenney, J. (November 1989), "DuWayne Peterson." SLOAN CURRENTS, MIT Sloan School of Management.

Krishna, S. (May 1990) "Global Financial Systems." S. M. Thesis, Sloan School of Management, Massachuestts Institute of Technology.

Landers, T., and Rosenberg, R. (1985), "An Overview of Multibase." Computer Corporation of America, 575 Technology Square, Cambridge, MA 02139, U.S.A.

Layne, R. (2 March1990), "Global Link Of Accounts A Bold Risk For Citicorp." The American Banker, p. 1.

Leavitt, H. J. (1965), Applied Organizational Change in Industry, in Handbook of Organizations. Chicago: Rand McNally.

Levine, S. (May 1987), "Interfacing Objects to Databases." B.S. Thesis, Massachusetts Institute of technology.

Madnick, S. (4 December 1989), "Trip to Salomon Brothers and Banker's Trust in NYC." CISL Background Information File.

Madnick, S. (August 1989), "Information Technology Platform For The 1990's." Sloan School of Management Working Paper, No. 3061-89-MS.

Madnick, S. (May 1989), "Memo on the meeting with Reuters Holdings PLC, London." CISL library.

Madnick, S., and Wang, R. (1988a), "A Framework of Composite Information Systems for Strategic Advantage." in Connectivity Among Information Systems, Volume I. Ed. Richard Wang and Stuart Madnick. Cambridge: Massachusetts Institute of Technology, pp. 6-22.

Madnick, S., and Wang, R. (1988b), "Evolution Towards Strategic Applications of Databases Through Composite Information Systems." in Connectivity Among Information Systems, Volume I. Ed. Richard Wang and Stuart Madnick. Cambridge: Massachusetts Institute of Technology, pp. 112-126.

Martin, J. (1982), Strategic Data-Planning Methodologie. Englewood Cliffs: Prentice-Hall.

Massimo III, J.L. (1987), "Gaining Strategic Advantage Through Composite Information Systems." in Knowledge-Based Integrated Information System Engineering (KBIISE) Project, Volume 7. Ed. Amar Gupta and Stuart Madnick. Cambridge: Massachusetts Institute of Technology Press, pp. 113-114.

Milner, W. (1988), Common Lisp, a Tutorial. Englewood Cliffs New Jersey : Prentice Hall.

Nolan, R. (1982), Managing The Data Resource Function. Second edition. West Publishing Company.

Osborn, C. (1987), "Towards a CIS Model For Strategic Applications." in Knowledge-Based Integrated Information System Engineering (KBIISE) Project, Volume 6. Ed. Amar Gupta and Stuart Madnick. Cambridge: Massachusetts Institute of Technology Press, pp. 5-64.

Paget, M. (June 1989), "A Knowledge-Based Approach Toward Integrating International On-Line Financial Databases." S.M. Thesis in Technology and Policy, Massachusetts Institute of Technology, Cambridge.

Power, D. (September 1984), "The Impact of Information Systems on the Organization: Two Scenarios." MIS Quarterly.

Rockart, J., and DeLong, D. (1988), Executive Support Systems: The Emergence of Top Management Computer Use. Homewood Illinois: Dow Jones-Irwin.

Rockart, J., and Short, J. (September 1988), "Information Technology and the New Organization: Towards More Effective Management of Interdependence." CISR Working Paper No. 180, Sloan Working Paper No. 2076-88, 90's Working Paper No. 88-058, Sloan School of Management, Massachusetts Institute of technology.

Rockart, J. (March-April 1979), "Chief Executives Define Their Own Data Needs." Harvard Business Review.

Ross, R. (1981), Data Dictionaries and Data Administration: Concepts and Practices for Data Resource Management. New York: Amacon.

Rothschild, D. (June 1987), "Managing Data As A Corporate Resource: The Organizational Roles and Responsibilities." S.M. Thesis, Sloan School of Management, Massachusetts Institute of Technology.

Siegel, M., and Madnick, S. (November 1989), "Identification and Reconciliation of Semantic Conflicts Using Metadata." CISL Working Paper No. 89-09, Sloan School Working Paper No. 3102-89 MSA.

Steele, G. (1984), Common Lisp. Digital Press.

Symons, C., and Tijmas, P. (November 1982), "A Systematic and Practical Approach to the Definition of Data." The Computer Journal, British Computer Society, Vol. 25, No.4.

Trice, A. (1987), "The Use of Standard Data Definitions in Composite Information Systems." in Knowledge-Based Integrated Information System Engineering (KBIISE) Project, Volume 6, eds. Amar Gupta and Stuart Madnick. Cambridge: Massachusetts Institute of Technology, pp. 117-135.

Tung, M. Y. (June 1990), "Local Query Processor Manager for the Composite Information Systems/Tool Kit." B.S. Thesis, Massachusetts Institute of Technology.

Venkatakrishnan, V. (September 1983), "The Information Cycle." Datamation.

Winston, P., and Horn, P. (1989), Lisp. 3rd Edition. Readings Massachusetts: Addison-Wesley.

Withington, F. (May-June 1980), "Coping With Computer Proliferation." Havard Business Review, p. 152.

Wong, T. K. (June 1989), "Data Connectivity in the Composite Information Systems/Tool Kit System." B.S. Thesis, Massachusetts Institute of Technology.

-oOo-

APPENDICES

Appendix[1]: Sample session

```
1> (SEND-MESSAGE DISCLOSUREDB :GET-DATA GLOBAL-DATA-TYPE-CATALOGUE
        (DISCLOSURE (CO NI CF)
            (AND (OR (= CO "HONDA MOTOR CO LTD") (= CO "JAGUAR PLC"))
                 (> NS "20000000"))))
```

(application's request)

Entering Data Conversion Strategy Builder:(application's request)

```
-----------------------------------------------
No sub-strategy is required for the domain YEAR
No sub-strategy is required for the domain NAME
Strategy = ((MONEY-AMOUNT ((SENDER-REPRESENTATION
                    ((FORMAT FIGURE-PLAIN) (UNIT USD) (SCALE SCALE-1)))
                         (RECEIVER-REPRESENTATION
                    ((FORMAT FIGURE-DISC) (UNIT (GET-UNIT-DISC IN))
                            (SCALE SCALE-1000)))
                    (INFER-ARGUMENTS  (IN))
                    (CONVERSION-ARGUMENTS ((DATE CF)))))
            (DATE ((SENDER-REPRESENTATION  ((FORMAT DATE-MM/DD/YYYY)))
                   (RECEIVER-REPRESENTATION  ((FORMAT DATE-
YYYYMMDD))))))
```

(application-to-database strategy)

Entering the Query Parsing module:(application's request)

```
--------------------------------------
Begin parsing the query's projection list...
... query's projection list is parsed.
New projection list: (CO NI CF IN)

Begin parsing constraint...
... constraint parsed.
Initial Query = (DISCLOSURE (CO NI CF)
                    (AND (OR (= CO HONDA MOTOR CO LTD)
                             (= CO JAGUAR PLC))
                         (> NS 20000000)))
PreConvertQuery = (DISCLOSURE (CO NI CF IN NS)
                      (AND (OR (= CO HONDA MOTOR CO LTD)
                               (= CO JAGUAR PLC))
                           (> NS 20000000)))
RemoteQuery = (DISCLOSURE (CO NI CF IN NS)
                  (OR (= CO HONDA MOTOR CO LTD) (= CO JAGUAR PLC)))
predicates-1 = (NS)
predicates-2 = NIL
```

Database Access:(application's request)

```
-----------------
Connecting to disclosure on machine I.P.SHARP - N. American Financial
```

```
Connected

  7864) 1990-05-02 15:51:15 IPSA

SHARP APL SERVICE

SAVED  1990-05-02 15:48:28

      )LOAD 39 MAGIC

SAVED  1990-03-19 03:07:24

WIDTH 300
YEARLY DATED 85 TO 88
DISPLAY '((CO EQ HONDA MOTOR CO LTD)) OR ((CO EQ JAGUAR PLC))'
ADISCLOSURE 'CO'

HONDA MOTOR CO LTD
JAGUAR PLC

DISPLAY '((CO EQ HONDA MOTOR CO LTD)) OR ((CO EQ JAGUAR PLC))'
ADISCLOSURE 'IN'

JAPAN
UNITED KINGDOM

COLWIDTH 15
DISPLAY '((CO EQ HONDA MOTOR CO LTD)) OR ((CO EQ JAGUAR PLC))'
ADISCLOSURE 'NI,CF,NS'

                    1985           1986           1987           1988

LINE 1           146,502         83,689         56,676         97,299
LINE 2        19,860,228     19,870,228     19,880,331     19,890,331
LINE 3         2,909,574      2,960,985      1,650,781      3,489,258
LINE 4            87,600         83,400         61,300         28,400
LINE 5        19,851,231     19,861,231     19,871,231     19,881,231
LINE 6           746,500        830,400      1,002,100      1,075,500

Disconnected
```

Data Reading:(application's request)

```
---------------
Results from I.P.Sharp are being filtered.
Filtering is done.

data-list = ((CO NI CF IN NS)
             (HONDA MOTOR CO LTD 146,502 19,860,228 JAPAN 2,909,574)
             (HONDA MOTOR CO LTD 83,689 19,870,228 JAPAN 2,960,985)
             (HONDA MOTOR CO LTD 56,676 19,880,331 JAPAN 1,650,781)
             (HONDA MOTOR CO LTD 97,299 19,890,331 JAPAN 3,489,258)
             (JAGUAR PLC 87,600 19,851,231 UNITED KINGDOM 746,500)
             (JAGUAR PLC 83,400 19,861,231 UNITED KINGDOM 830,400)
             (JAGUAR PLC 61,300 19,871,231 UNITED KINGDOM 1,002,100)
             (JAGUAR PLC 28,400 19,881,231 UNITED KINGDOM 1,075,500))
```

(we will call this set of data "data-from-database")

Entering the Data Filtering Module:(application's request)

Data Filter Step #1:

```
First-class predicates = (NS)
Data is converted into the sender's data definitions for the attributes
in (NS)

Entering MONEY-AMOUNT FORMAT convertions...
    2> (SEND-MESSAGE FIGURE-PLAIN :CONVERT
            (((MONEY-AMOUNT ((VALUE "2,909,574") (FORMAT FIGURE-DISC)
                        (UNIT NJPY) (SCALE SCALE-1000)))
              (DATE ((VALUE "19,860,228") (FORMAT DATE-YYYYMMDD))))
             ((MONEY-AMOUNT ((VALUE "2,960,985") (FORMAT FIGURE-DISC)
                        (UNIT NJPY) (SCALE SCALE-1000)))
              (DATE ((VALUE "19,870,228") (FORMAT DATE-YYYYMMDD))))
             ((MONEY-AMOUNT ((VALUE "1,650,781") (FORMAT FIGURE-DISC)
                        (UNIT NJPY) (SCALE SCALE-1000)))
              (DATE ((VALUE "19,880,331") (FORMAT DATE-YYYYMMDD))))
             ((MONEY-AMOUNT ((VALUE "3,489,258") (FORMAT FIGURE-DISC)
                        (UNIT NJPY) (SCALE SCALE-1000)))
              (DATE ((VALUE "19,890,331") (FORMAT DATE-YYYYMMDD))))
             ((MONEY-AMOUNT ((VALUE "746,500") (FORMAT FIGURE-DISC)
                        (UNIT NGBP) (SCALE SCALE-1000)))
              (DATE ((VALUE "19,851,231") (FORMAT DATE-YYYYMMDD))))
             ((MONEY-AMOUNT ((VALUE "830,400") (FORMAT FIGURE-DISC)
                        (UNIT NGBP) (SCALE SCALE-1000)))
              (DATE ((VALUE "19,861,231") (FORMAT DATE-YYYYMMDD))))
             ((MONEY-AMOUNT ((VALUE "1,002,100") (FORMAT FIGURE-DISC)
                        (UNIT NGBP) (SCALE SCALE-1000)))
              (DATE ((VALUE "19,871,231") (FORMAT DATE-YYYYMMDD))))
             ((MONEY-AMOUNT ((VALUE "1,075,500") (FORMAT FIGURE-DISC)
                        (UNIT NGBP) (SCALE SCALE-1000)))
              (DATE ((VALUE "19,881,231") (FORMAT DATE-YYYYMMDD))))))

    <2 (SEND-MESSAGE
            (((MONEY-AMOUNT ((VALUE "2909574") (FORMAT FIGURE-PLAIN)
                         (UNIT NJPY) (SCALE SCALE-1000)))
              (DATE ((VALUE "19,860,228") (FORMAT DATE-YYYYMMDD))))
             ((MONEY-AMOUNT ((VALUE "2960985") (FORMAT FIGURE-PLAIN)
                         (UNIT NJPY) (SCALE SCALE-1000)))
              (DATE ((VALUE "19,870,228") (FORMAT DATE-YYYYMMDD))))
             ((MONEY-AMOUNT ((VALUE "1650781") (FORMAT FIGURE-PLAIN)
                         (UNIT NJPY) (SCALE SCALE-1000)))
              (DATE ((VALUE "19,880,331") (FORMAT DATE-YYYYMMDD))))
             ((MONEY-AMOUNT ((VALUE "3489258") (FORMAT FIGURE-PLAIN)
                         (UNIT NJPY) (SCALE SCALE-1000)))
              (DATE ((VALUE "19,890,331") (FORMAT DATE-YYYYMMDD))))
             ((MONEY-AMOUNT ((VALUE "746500") (FORMAT FIGURE-PLAIN)
                         (UNIT NGBP) (SCALE SCALE-1000)))
              (DATE ((VALUE "19,851,231") (FORMAT DATE-YYYYMMDD))))
             ((MONEY-AMOUNT ((VALUE "830400") (FORMAT FIGURE-PLAIN)
                         (UNIT NGBP) (SCALE SCALE-1000)))
              (DATE ((VALUE "19,861,231") (FORMAT DATE-YYYYMMDD))))
             ((MONEY-AMOUNT ((VALUE "1002100") (FORMAT FIGURE-PLAIN)
                         (UNIT NGBP) (SCALE SCALE-1000)))
              (DATE ((VALUE "19,871,231") (FORMAT DATE-YYYYMMDD))))
```

```
          ((MONEY-AMOUNT ((VALUE "1075500") (FORMAT FIGURE-PLAIN)
                          (UNIT NGBP) (SCALE SCALE-1000)))
           (DATE ((VALUE "19,881,231") (FORMAT DATE-YYYYMMDD))))))
MONEY-AMOUNT FORMAT convertions done.

Entering MONEY-AMOUNT UNIT conversions...(conversion of NS data elements)

    2> (SEND-MESSAGE USD :CONVERT
           (((MONEY-AMOUNT ((VALUE "2909574") (FORMAT FIGURE-PLAIN)
                            (UNIT NJPY) (SCALE SCALE-1000)))
             (DATE ((VALUE "19,860,228") (FORMAT DATE-YYYYMMDD))))
            ((MONEY-AMOUNT ((VALUE "2960985") (FORMAT FIGURE-PLAIN)
                            (UNIT NJPY) (SCALE SCALE-1000)))
             (DATE ((VALUE "19,870,228") (FORMAT DATE-YYYYMMDD))))
            ((MONEY-AMOUNT ((VALUE "1650781") (FORMAT FIGURE-PLAIN)
                            (UNIT NJPY) (SCALE SCALE-1000)))
             (DATE ((VALUE "19,880,331") (FORMAT DATE-YYYYMMDD))))
            ((MONEY-AMOUNT ((VALUE "3489258") (FORMAT FIGURE-PLAIN)
                            (UNIT NJPY) (SCALE SCALE-1000)))
             (DATE ((VALUE "19,890,331") (FORMAT DATE-YYYYMMDD))))
            ((MONEY-AMOUNT ((VALUE "746500") (FORMAT FIGURE-PLAIN)
                            (UNIT NGBP) (SCALE SCALE-1000)))
             (DATE ((VALUE "19,851,231") (FORMAT DATE-YYYYMMDD))))
            ((MONEY-AMOUNT ((VALUE "830400") (FORMAT FIGURE-PLAIN)
                            (UNIT NGBP) (SCALE SCALE-1000)))
             (DATE ((VALUE "19,861,231") (FORMAT DATE-YYYYMMDD))))
            ((MONEY-AMOUNT ((VALUE "1002100") (FORMAT FIGURE-PLAIN)
                            (UNIT NGBP) (SCALE SCALE-1000)))
             (DATE ((VALUE "19,871,231") (FORMAT DATE-YYYYMMDD))))
            ((MONEY-AMOUNT ((VALUE "1075500") (FORMAT FIGURE-PLAIN)
                            (UNIT NGBP) (SCALE SCALE-1000)))
             (DATE ((VALUE "19,881,231") (FORMAT DATE-YYYYMMDD))))))


Accessing the I.P. Sharp Currency database...

(currency conversions need exchange rates)

      3> (SEND-MESSAGE CURRENCYDB :GET-DATA FIGURE-UNIT-CATALOGUE
             (CURRENCY (EXCHANGE-RATE)
                 (OR (AND (= CODE "NJPY") (= DATE "19,860,228"))
                     (OR (AND (= CODE "NJPY") (= DATE "19,870,228"))
                         (OR (AND (= CODE "NJPY")
                                  (= DATE "19,880,331"))
                             (OR (AND (= CODE "NJPY")
                                      (= DATE "19,890,331"))
                                 (OR (AND (= CODE "NGBP")
                                          (= DATE "19,851,231"))
                                     (OR
                                      (AND (= CODE "NGBP")
                                       (= DATE "19,861,231"))
                                      (OR
                                       (AND (= CODE "NGBP")
                                        (= DATE "19,871,231"))
                                       (AND (= CODE "NGBP")
                                        (= DATE "19,881,231")))))))))))
```

Entering Data Conversion Strategy Builder:(ex. rates's request)

```
---------------------------------------------
No sub-strategy is required for the domain NAME
Strategy = ((MONEY-AMOUNT ((SENDER-REPRESENTATION
                           ((FORMAT FIGURE-PLAIN) (UNIT USD)
                            (SCALE SCALE-1)))
                       (RECEIVER-REPRESENTATION
                           ((FORMAT FIGURE-PLAIN) (UNIT USD)
                            (SCALE SCALE-0.01)))))
            (DATE ((SENDER-REPRESENTATION  ((FORMAT DATE-YYYYMMDD)))
                   (RECEIVER-REPRESENTATION  ((FORMAT DATE-DD-MM-
YY))))))
```

Entering the Query Parsing module:(ex. rates' request)

```
---------------------------------------
Begin parsing the query's projection list...
... query's projection list is parsed.
New projection list: (EXCHANGE-RATE)

Begin parsing constraint...

Predicate at hand: (= CODE NJPY)

Predicate at hand: (= DATE 19,860,228)
        4> (SEND-MESSAGE DATE-DD-MM-YY :CONVERT
                (((DATE ((VALUE "19,860,228") (FORMAT DATE-YYYYMMDD))))))

Entering DATE FORMAT convertions...
DATE FORMAT convertions done.
        <4 (SEND-MESSAGE
                (((DATE ((VALUE "28-02-86") (FORMAT DATE-DD-MM-YY))))))

Converted predicate = (= DATE 28-02-86)

etc...
... constraint parsed.
Initial Query:
(CURRENCY (EXCHANGE-RATE)
(OR (AND (= CODE NJPY) (= DATE 19,860,228))
    (OR (AND (= CODE NJPY) (= DATE 19,870,228))
        (OR (AND (= CODE NJPY) (= DATE 19,880,331))
            (OR (AND (= CODE NJPY) (= DATE 19,890,331))
                (OR (AND (= CODE NGBP) (= DATE 19,851,231))
                    (OR (AND (= CODE NGBP) (= DATE 19,861,231))
                        (OR (AND (= CODE NGBP) (= DATE 19,871,231))
                            (AND (= CODE NGBP) (= DATE
19,881,231)))))))))

PreConvertQuery:
(CURRENCY (EXCHANGE-RATE CODE DATE)
(OR (AND (= CODE NJPY) (= DATE 28-02-86))
    (OR (AND (= CODE NJPY) (= DATE 28-02-87))
        (OR (AND (= CODE NJPY) (= DATE 31-03-88))
            (OR (AND (= CODE NJPY) (= DATE 31-03-89))
                (OR (AND (= CODE NGBP) (= DATE 31-12-85))
                    (OR (AND (= CODE NGBP) (= DATE 31-12-86))
                        (OR (AND (= CODE NGBP) (= DATE 31-12-87))
```

```
                              (AND (= CODE NGBP) (= DATE 31-12-
88))))))))))

RemoteQuery:
(CURRENCY (EXCHANGE-RATE CODE DATE)
(OR (AND (= CODE NJPY) (= DATE 28-02-86))
    (OR (AND (= CODE NJPY) (= DATE 28-02-87))
        (OR (AND (= CODE NJPY) (= DATE 31-03-88))
            (OR (AND (= CODE NJPY) (= DATE 31-03-89))
                (OR (AND (= CODE NGBP) (= DATE 31-12-85))
                    (OR (AND (= CODE NGBP) (= DATE 31-12-86))
                        (OR (AND (= CODE NGBP) (= DATE 31-12-87))
                            (AND (= CODE NGBP) (= DATE 31-12-
88))))))))))

predicates-1 = NIL
predicates-2 = NIL
```

Database Access:(ex. rates' request)

```
------------------
Connecting to currency on machine CURRENCY - I.P. Sharp Currency
database...
Creating the communication script...
...communication script is done.

dialing CURRENCY - I.P. Sharp Currency database...
Connected

7909) 1990-05-02 15:53:11 IPSA

SHARP APL SERVICE

SAVED  1990-05-02 15:52:30

)LOAD 39 MAGIC

SAVED  1990-03-19 03:07:24

1 2 3 4 5 6 7 DAILY DATED 28 02 86 TO 6 3 86
CURRENCY 'NJPY'
0.5538631958 0            0            0.556637907  0.5586592179
0.5511160099 0.5577244841

1 2 3 4 5 6 7 DAILY DATED 28 02 87 TO 6 3 87
CURRENCY 'NJPY'
0            0            0.651465798  0.650829808  0.6505757595
0.6527415144 0.6515931452

1 2 3 4 5 6 7 DAILY DATED 31 03 88 TO 6 4 88
CURRENCY 'NJPY'
0.8058017728 0.8064516129 0            0            0.8038585209
0.7988496565 0.796812749

1 2 3 4 5 6 7 DAILY DATED 31 03 89 TO 6 4 89
CURRENCY 'NJPY'
0.7531821948 0            0            0.7572889057 0.7639419404
0.7587253414 0.7579777155
```

```
1 2 3 4 5 6 7 DAILY DATED 31 12 85 TO 6 1 86
CURRENCY 'NGBP'
144.5    0    145.05 143.8    0      0    143.55

1 2 3 4 5 6 7 DAILY DATED 31 12 86 TO 6 1 87
CURRENCY 'NGBP'
148.25   0    149      0      0    147.28 147.62

1 2 3 4 5 6 7 DAILY DATED 31 12 87 TO 6 1 88
CURRENCY 'NGBP'
188.6    0      0      0    187.73 183.05 180.6

1 2 3 4 5 6 7 DAILY DATED 31 12 88 TO 6 1 89
CURRENCY 'NGBP'
  0      0      0    182.25 180.7  179.7  178

Disconnected
```

Entering the Data File Reading Module:(ex. rates' request)

```
-------------------------------------------
data-list = ((EXCHANGE-RATE CODE DATE) (0.5538631958 NJPY 28-02-86)
             (0.651465798 NJPY 28-02-87) (0.8058017728 NJPY 31-03-88)
             (0.7531821948 NJPY 31-03-89) (144.5 NGBP 31-12-85)
             (148.25 NGBP 31-12-86) (188.6 NGBP 31-12-87)
             (182.25 NGBP 31-12-88))
```

Entering the Data Filter Module:(ex. rates' request)

```
-----------------------------------
predicates-1 and predicates-2 are both null.
```

(data filter steps 1 to 3 are skipped since both predicates-1 and predicates-2 are empty)

Data Filter Step #4:

```
converting data into the sender's data definitions.
No conversion is necessary for (FORMAT (FIGURE-PLAIN) (FIGURE-PLAIN))
No conversion is necessary for (UNIT (USD) (USD))

Entering MONEY-AMOUNT SCALE convertions...

        4> (SEND-MESSAGE SCALE-1 :CONVERT
          (((MONEY-AMOUNT ((VALUE "0.5538631958") (FORMAT FIGURE-PLAIN)
                           (UNIT USD) (SCALE SCALE-0.01))))
          ((MONEY-AMOUNT ((VALUE "0.651465798") (FORMAT FIGURE-PLAIN)
                           (UNIT USD) (SCALE SCALE-0.01))))
          ((MONEY-AMOUNT ((VALUE "0.8058017728") (FORMAT FIGURE-PLAIN)
                           (UNIT USD) (SCALE SCALE-0.01))))
          ((MONEY-AMOUNT ((VALUE "0.7531821948") (FORMAT FIGURE-PLAIN)
                           (UNIT USD) (SCALE SCALE-0.01))))
          ((MONEY-AMOUNT ((VALUE "144.5") (FORMAT FIGURE-PLAIN)
                           (UNIT USD) (SCALE SCALE-0.01))))
          ((MONEY-AMOUNT ((VALUE "148.25") (FORMAT FIGURE-PLAIN)
                           (UNIT USD) (SCALE SCALE-0.01))))
          ((MONEY-AMOUNT ((VALUE "188.6") (FORMAT FIGURE-PLAIN)
                           (UNIT USD) (SCALE SCALE-0.01))))
          ((MONEY-AMOUNT ((VALUE "182.25") (FORMAT FIGURE-PLAIN)
                           (UNIT USD) (SCALE SCALE-0.01))))))

        <4 (SEND-MESSAGE
```

```
                (((MONEY-AMOUNT ((VALUE "0.005538631958")
                                 (FORMAT FIGURE-PLAIN) (UNIT USD)
                                 (SCALE SCALE-1))))
                 ((MONEY-AMOUNT ((VALUE "0.006514657980000001")
                                 (FORMAT FIGURE-PLAIN) (UNIT USD)
                                 (SCALE SCALE-1))))
                 ((MONEY-AMOUNT ((VALUE "0.008058017727999999")
                                 (FORMAT FIGURE-PLAIN) (UNIT USD)
                                 (SCALE SCALE-1))))
                 ((MONEY-AMOUNT ((VALUE "0.007531821948000001")
                                 (FORMAT FIGURE-PLAIN) (UNIT USD)
                                 (SCALE SCALE-1))))
                 ((MONEY-AMOUNT ((VALUE "1.445") (FORMAT FIGURE-PLAIN)
                                 (UNIT USD) (SCALE SCALE-1))))
                 ((MONEY-AMOUNT ((VALUE "1.4825") (FORMAT FIGURE-PLAIN)
                                 (UNIT USD) (SCALE SCALE-1))))
                 ((MONEY-AMOUNT ((VALUE "1.886") (FORMAT FIGURE-PLAIN)
                                 (UNIT USD) (SCALE SCALE-1))))
                 ((MONEY-AMOUNT ((VALUE "1.8225") (FORMAT FIGURE-PLAIN)
                                 (UNIT USD) (SCALE SCALE-1))))))

MONEY-AMOUNT SCALE conversions done.

      <3 (SEND-MESSAGE
             (("EXCHANGE-RATE") ("0.005538631958")
              ("0.006514657980000001") ("0.008058017727999999")
              ("0.007531821948000001") ("1.445") ("1.4825") ("1.886")
              ("1.8225")))

Unit convertions into USD...
```

(we are still in the process of converting the NS elements)

```
    <2 (SEND-MESSAGE
       (((MONEY-AMOUNT ((VALUE "16115.05954056589") (FORMAT FIGURE-PLAIN)
                        (UNIT USD) (SCALE SCALE-1000)))
              (DATE ((VALUE "19,860,228") (FORMAT DATE-YYYYMMDD))))
        ((MONEY-AMOUNT ((VALUE "19289.80455891031") (FORMAT FIGURE-PLAIN)
                        (UNIT USD) (SCALE SCALE-1000)))
              (DATE ((VALUE "19,870,228") (FORMAT DATE-YYYYMMDD))))
        ((MONEY-AMOUNT ((VALUE "13302.02256304557") (FORMAT FIGURE-PLAIN)
                        (UNIT USD) (SCALE SCALE-1000)))
              (DATE ((VALUE "19,880,331") (FORMAT DATE-YYYYMMDD))))
        ((MONEY-AMOUNT ((VALUE "26280.46998663459") (FORMAT FIGURE-PLAIN)
                        (UNIT USD) (SCALE SCALE-1000)))
              (DATE ((VALUE "19,890,331") (FORMAT DATE-YYYYMMDD))))
        ((MONEY-AMOUNT ((VALUE "1078692.5") (FORMAT FIGURE-PLAIN)
                        (UNIT USD) (SCALE SCALE-1000)))
              (DATE ((VALUE "19,851,231") (FORMAT DATE-YYYYMMDD))))
        ((MONEY-AMOUNT ((VALUE "1231068.0") (FORMAT FIGURE-PLAIN)
                        (UNIT USD) (SCALE SCALE-1000)))
              (DATE ((VALUE "19,861,231") (FORMAT DATE-YYYYMMDD))))
        ((MONEY-AMOUNT ((VALUE "1889960.6") (FORMAT FIGURE-PLAIN)
                        (UNIT USD) (SCALE SCALE-1000)))
              (DATE ((VALUE "19,871,231") (FORMAT DATE-YYYYMMDD))))
        ((MONEY-AMOUNT ((VALUE "1960098.75") (FORMAT FIGURE-PLAIN)
                        (UNIT USD) (SCALE SCALE-1000)))
              (DATE ((VALUE "19,881,231") (FORMAT DATE-YYYYMMDD))))))
```

```
MONEY-AMOUNT UNIT conversions done.(of NS data elements)

Entering MONEY-AMOUNT SCALE convertions...(of NS data elements)

    2> (SEND-MESSAGE SCALE-1 :CONVERT
      (((MONEY-AMOUNT ((VALUE "16115.05954056589") (FORMAT FIGURE-PLAIN)
                       (UNIT USD) (SCALE SCALE-1000)))
              (DATE ((VALUE "19,860,228") (FORMAT DATE-YYYYMMDD))))
       ((MONEY-AMOUNT ((VALUE "19289.80455891031") (FORMAT FIGURE-PLAIN)
                       (UNIT USD) (SCALE SCALE-1000)))
              (DATE ((VALUE "19,870,228") (FORMAT DATE-YYYYMMDD))))
       ((MONEY-AMOUNT ((VALUE "13302.02256304557") (FORMAT FIGURE-PLAIN)
                       (UNIT USD) (SCALE SCALE-1000)))
              (DATE ((VALUE "19,880,331") (FORMAT DATE-YYYYMMDD))))
       ((MONEY-AMOUNT ((VALUE "26280.46998663459") (FORMAT FIGURE-PLAIN)
                       (UNIT USD) (SCALE SCALE-1000)))
              (DATE ((VALUE "19,890,331") (FORMAT DATE-YYYYMMDD))))
       ((MONEY-AMOUNT ((VALUE "1078692.5") (FORMAT FIGURE-PLAIN)
                       (UNIT USD) (SCALE SCALE-1000)))
              (DATE ((VALUE "19,851,231") (FORMAT DATE-YYYYMMDD))))
       ((MONEY-AMOUNT ((VALUE "1231068.0") (FORMAT FIGURE-PLAIN)
                       (UNIT USD) (SCALE SCALE-1000)))
              (DATE ((VALUE "19,861,231") (FORMAT DATE-YYYYMMDD))))
       ((MONEY-AMOUNT ((VALUE "1889960.6") (FORMAT FIGURE-PLAIN)
                       (UNIT USD) (SCALE SCALE-1000)))
              (DATE ((VALUE "19,871,231") (FORMAT DATE-YYYYMMDD))))
       ((MONEY-AMOUNT ((VALUE "1960098.75") (FORMAT FIGURE-PLAIN)
                       (UNIT USD) (SCALE SCALE-1000)))
              (DATE ((VALUE "19,881,231") (FORMAT DATE-YYYYMMDD))))))

    <2 (SEND-MESSAGE
      (((MONEY-AMOUNT ((VALUE "16115059.54056589") (FORMAT FIGURE-PLAIN)
                       (UNIT USD) (SCALE SCALE-1)))
              (DATE ((VALUE "19,860,228") (FORMAT DATE-YYYYMMDD))))
       ((MONEY-AMOUNT ((VALUE "19289804.55891031") (FORMAT FIGURE-PLAIN)
                       (UNIT USD) (SCALE SCALE-1)))
              (DATE ((VALUE "19,870,228") (FORMAT DATE-YYYYMMDD))))
       ((MONEY-AMOUNT ((VALUE "13302022.56304557") (FORMAT FIGURE-PLAIN)
                       (UNIT USD) (SCALE SCALE-1)))
              (DATE ((VALUE "19,880,331") (FORMAT DATE-YYYYMMDD))))
       ((MONEY-AMOUNT ((VALUE "26280469.98663459") (FORMAT FIGURE-PLAIN)
                       (UNIT USD) (SCALE SCALE-1)))
              (DATE ((VALUE "19,890,331") (FORMAT DATE-YYYYMMDD))))
       ((MONEY-AMOUNT ((VALUE "1078692500") (FORMAT FIGURE-PLAIN)
                       (UNIT USD) (SCALE SCALE-1)))
              (DATE ((VALUE "19,851,231") (FORMAT DATE-YYYYMMDD))))
       ((MONEY-AMOUNT ((VALUE "1231068000") (FORMAT FIGURE-PLAIN)
                       (UNIT USD) (SCALE SCALE-1)))
              (DATE ((VALUE "19,861,231") (FORMAT DATE-YYYYMMDD))))
       ((MONEY-AMOUNT ((VALUE "1889960600") (FORMAT FIGURE-PLAIN)
                       (UNIT USD) (SCALE SCALE-1)))
              (DATE ((VALUE "19,871,231") (FORMAT DATE-YYYYMMDD))))
       ((MONEY-AMOUNT ((VALUE "1960098750") (FORMAT FIGURE-PLAIN)
                       (UNIT USD) (SCALE SCALE-1)))
              (DATE ((VALUE "19,881,231") (FORMAT DATE-YYYYMMDD))))))

MONEY-AMOUNT SCALE conversions done.(of NS data elements)
```

(Now, we are going to download the data into an auxiliary table, and we must check whether the representation of the data we want to download is consistent with the representation expected by the auxiliary table. In order to do so, we check first the representation of the "predicates-1" data elements, and then the other data elements. We have to make this distinction because the "predicates-1" data elements have just been converted into the application's data representation conventions (see conversion of NS data elements above), whereas the other data elements are still in the database's data representation conventions)

("predicates-1" data elements)

```
Converting the data list into the auxiliary table's data definitions for
the attributes of (NS)

Entering Data Conversion Strategy Builder:
------------------------------------------------
```

(it is necessary to build a new strategy : the application-to-database strategy that was built were the initial query was sent to the bridge does not capture the information that we need here. For the "predicates-1" data elements, we are to build a application-to-auxiliary-table strategy)

```
No sub-strategy is required for the domain NAME
Strategy = ((MONEY-AMOUNT ((SENDER-REPRESENTATION
                           ((FORMAT FIGURE-PLAIN) (UNIT USD)
                            (SCALE SCALE-1)))
                          (RECEIVER-REPRESENTATION
                            ((FORMAT FIGURE-PLAIN) (UNIT NIL) (SCALE NIL)))))
            (DATE ((SENDER-REPRESENTATION  ((FORMAT DATE-MM/DD/YYYY)))
                   (RECEIVER-REPRESENTATION  ((FORMAT NIL)))))
            (YEAR ((SENDER-REPRESENTATION  ((FORMAT YEAR-YYYY)))
                   (RECEIVER-REPRESENTATION  ((FORMAT NIL))))))
```

(application-to-auxiliary-table strategy)

(conversion of NS data elements into the auxiliary table's data representation conventions)

```
MONEY-AMOUNT conversion:
No conversion is necessary for (FORMAT (FIGURE-PLAIN) (FIGURE-PLAIN))
No conversion is necessary for (UNIT (USD) (NIL))
No conversion is necessary for (SCALE (SCALE-1) (NIL))
```

(other data elements)

```
Converting the data list into the auxiliary table's data definitions for
the attributes in (CO NI CF IN)

Entering Data Conversion Strategy Builder:
------------------------------------------
No sub-strategy is required for the domain NAME
Strategy = ((MONEY-AMOUNT ((SENDER-REPRESENTATION
                           ((FORMAT FIGURE-DISC)
                            (UNIT (GET-UNIT-DISC IN)) (SCALE SCALE-1000)))
                          (RECEIVER-REPRESENTATION
                            ((FORMAT FIGURE-PLAIN) (UNIT NIL) (SCALE NIL)))))
            (DATE ((SENDER-REPRESENTATION  ((FORMAT DATE-YYYYMMDD)))
                   (RECEIVER-REPRESENTATION  ((FORMAT NIL))))))
```

```
          (YEAR ((SENDER-REPRESENTATION   ((FORMAT YEAR-YYYY)))
                 (RECEIVER-REPRESENTATION  ((FORMAT NIL))))))
```

(database-to-auxiliary-table strategy: notice that there is a mismatch on the format of the data type money-amount, hence a converion request is sent to the object FIGURE-PLAIN)

```
Entering MONEY-AMOUNT FORMAT convertions...
```

(the NI data elements within data-from-database are converted)

```
    2> (SEND-MESSAGE FIGURE-PLAIN :CONVERT
     (((MONEY-AMOUNT ((VALUE "146,502") (FORMAT FIGURE-DISC)
                      (UNIT NJPY) (SCALE SCALE-1000))))
     ((MONEY-AMOUNT ((VALUE "83,689") (FORMAT FIGURE-DISC) (UNIT NJPY)
                      (SCALE SCALE-1000))))
     ((MONEY-AMOUNT ((VALUE "56,676") (FORMAT FIGURE-DISC) (UNIT NJPY)
                      (SCALE SCALE-1000))))
     ((MONEY-AMOUNT ((VALUE "97,299") (FORMAT FIGURE-DISC) (UNIT NJPY)
                      (SCALE SCALE-1000))))
     ((MONEY-AMOUNT ((VALUE "87,600") (FORMAT FIGURE-DISC) (UNIT NGBP)
                      (SCALE SCALE-1000))))
     ((MONEY-AMOUNT ((VALUE "83,400") (FORMAT FIGURE-DISC) (UNIT NGBP)
                      (SCALE SCALE-1000))))
     ((MONEY-AMOUNT ((VALUE "61,300") (FORMAT FIGURE-DISC) (UNIT NGBP)
                      (SCALE SCALE-1000))))
     ((MONEY-AMOUNT ((VALUE "28,400") (FORMAT FIGURE-DISC) (UNIT NGBP)
                      (SCALE SCALE-1000))))))

    <2 (SEND-MESSAGE
     (((MONEY-AMOUNT ((VALUE "146502") (FORMAT FIGURE-PLAIN)
                      (UNIT NJPY) (SCALE SCALE-1000))))
     ((MONEY-AMOUNT ((VALUE "83689") (FORMAT FIGURE-PLAIN) (UNIT NJPY)
                      (SCALE SCALE-1000))))
     ((MONEY-AMOUNT ((VALUE "56676") (FORMAT FIGURE-PLAIN) (UNIT NJPY)
                      (SCALE SCALE-1000))))
     ((MONEY-AMOUNT ((VALUE "97299") (FORMAT FIGURE-PLAIN) (UNIT NJPY)
                      (SCALE SCALE-1000))))
     ((MONEY-AMOUNT ((VALUE "87600") (FORMAT FIGURE-PLAIN) (UNIT NGBP)
                      (SCALE SCALE-1000))))
     ((MONEY-AMOUNT ((VALUE "83400") (FORMAT FIGURE-PLAIN) (UNIT NGBP)
                      (SCALE SCALE-1000))))
     ((MONEY-AMOUNT ((VALUE "61300") (FORMAT FIGURE-PLAIN) (UNIT NGBP)
                      (SCALE SCALE-1000))))
     ((MONEY-AMOUNT ((VALUE "28400") (FORMAT FIGURE-PLAIN) (UNIT NGBP)
                      (SCALE SCALE-1000))))))

MONEY-AMOUNT FORMAT convertions done.

No conversion is necessary for (UNIT ((GET-UNIT-DISC IN)) (NIL))
No conversion is necessary for (SCALE (SCALE-1000) (NIL))
No conversion is necessary for (FORMAT (DATE-YYYYMMDD) (NIL))

Data Filter Step #2:(application's request)
-------------------
```

(Now that the data has been converted into the auxiliary table's data representation conventions, we can download it; ":download" is an object method)

```
2> (SEND-MESSAGE DISC-AUX-TABLE :DOWNLOAD
        (("CO" "NI" "CF" "INC" "NS")
         ("HONDA MOTOR CO LTD" "146502" "19,860,228" "JAPAN"
          "16115059.54056589")
         ("HONDA MOTOR CO LTD" "83689" "19,870,228" "JAPAN"
          "19289804.55891031")
         ("HONDA MOTOR CO LTD" "56676" "19,880,331" "JAPAN"
          "13302022.56304557")
         ("HONDA MOTOR CO LTD" "97299" "19,890,331" "JAPAN"
          "26280469.98663459")
         ("JAGUAR PLC" "87600" "19,851,231" "UNITED KINGDOM"
          "1078692500")
         ("JAGUAR PLC" "83400" "19,861,231" "UNITED KINGDOM"
          "1231068000")
         ("JAGUAR PLC" "61300" "19,871,231" "UNITED KINGDOM"
          "1889960600")
         ("JAGUAR PLC" "28400" "19,881,231" "UNITED KINGDOM"
          "1960098750"))
        DISC-AUX-DATA-TYPE-CATALOGUE
        (CO PC SC YR CF NI NS INC TE EM AD1) "/usr/cistk/database"
        "auxiliary" "ipsharp")

Downloading the data into a local auxiliary table...
```

(Notice that the "download" method will check whether the data is in the right representation conventions. In fact, we made sure that the data that was passed to the object DISC-AUX-TABLE was already in the auxiliary table's representation convention. Indeed, we could not have told the object that within the data set some data elements were in a first representation, and other data elements were in a second representation. A single data type catalog is passed as argument in the message, and the object assumes that all data elements satisfy the representation information present in the catalog)

```
Entering Data Conversion Strategy Builder:
------------------------------------------
No sub-strategy is required for the domain FIGURE
No sub-strategy is required for the domain DATE
No sub-strategy is required for the domain YEAR
No sub-strategy is required for the domain NAME
Strategy = NIL
The data is already in DISC-AUX-TABLE's data representation

Reordering (CO NI CF INC NS) into (CO PC SC YR CF NI NS INC TE EM
AD1)...
Downloading data in table ipsharp of database auxiliary...
Writing into /usr/cistk/brigaldi/Thesis/Work/Tmp/wn6dVj33G5:
((HONDA MOTOR CO LTD |    |    |    | 19,860,228 | 146502 |
  16115059.54056589 | JAPAN |    |    | )
 (HONDA MOTOR CO LTD |    |    |    | 19,870,228 | 83689 |
  19289804.55891031 | JAPAN |    |    | )
 (HONDA MOTOR CO LTD |    |    |    | 19,880,331 | 56676 |
  13302022.56304557 | JAPAN |    |    | )
 (HONDA MOTOR CO LTD |    |    |    | 19,890,331 | 97299 |
  26280469.98663459 | JAPAN |    |    | )
 (JAGUAR PLC |    |    |    | 19,851,231 | 87600 | 1078692500 |
  UNITED KINGDOM |    |    | )
 (JAGUAR PLC |    |    |    | 19,861,231 | 83400 | 1231068000 |
  UNITED KINGDOM |    |    | )
```

```
 (JAGUAR PLC |    |    |    | 19,871,231 | 61300 | 1889960600 |
  UNITED KINGDOM |    |    |  )
 (JAGUAR PLC |    |    |    | 19,881,231 | 28400 | 1960098750 |
  UNITED KINGDOM |    |    |  ))
8 row(s) loaded.

    <2 (SEND-MESSAGE 0)
```

Data Filter Step #3:(application's request)

(the query PreConvertQuery is applied to the auxiliary table)

```
Select data in the auxiliary table ipsharp with the query:
(DISCLOSURE (CO NI CF INC)
      (AND  (OR (= CO HONDA MOTOR CO LTD) (= CO JAGUAR PLC))
            (> NS 20000000)))

    2> (SEND-MESSAGE DISC-AUX-TABLE :GET-DATA
          GLOBAL-DATA-TYPE-CATALOGUE
          ("ipsharp" (CO NI CF INC)
           (AND (OR (= CO "HONDA MOTOR CO LTD") (= CO "JAGUAR PLC"))
                (> NS "20000000"))))
```

Entering Data Conversion Strategy Builder:(aux. table's request)

```
No sub-strategy is required for the domain NAME
Strategy = ((MONEY-AMOUNT ((SENDER-REPRESENTATION
                           ((FORMAT FIGURE-PLAIN) (UNIT USD)
                            (SCALE SCALE-1)))
                       (RECEIVER-REPRESENTATION
                           ((FORMAT FIGURE-PLAIN) (UNIT NIL) (SCALE
NIL)))))
            (DATE ((SENDER-REPRESENTATION  ((FORMAT DATE-MM/DD/YYYY)))
                   (RECEIVER-REPRESENTATION  ((FORMAT NIL)))))
            (YEAR ((SENDER-REPRESENTATION  ((FORMAT YEAR-YYYY)))
                   (RECEIVER-REPRESENTATION  ((FORMAT NIL))))))
```

Entering the Query Parsing module:(aux. table's request)

```
Begin parsing the query's projection list...
... query's projection list is parsed.
New projection list: (CO NI CF INC)

Begin parsing constraint...

Predicate at hand: (= CO HONDA MOTOR CO LTD)

Predicate at hand: (= CO JAGUAR PLC)

Predicate at hand: (> NS 20000000)
... constraint parsed.
Initial Query = (ipsharp (CO NI CF INC)
                 (AND (OR (= CO HONDA MOTOR CO LTD) (= CO JAGUAR PLC))
                      (> NS 20000000)))
PreConvertQuery = (ipsharp (CO NI CF INC NS)
                   (AND (OR (= CO HONDA MOTOR CO LTD)
                            (= CO JAGUAR PLC))
```

```
                        (> NS 20000000)))
RemoteQuery = (ipsharp (CO NI CF INC NS)
                (AND (OR (= CO HONDA MOTOR CO LTD) (= CO JAGUAR PLC))
                     (> NS 20000000)))
predicates-1 = NIL
predicates-2 = NIL

Connecting to auxiliary on machine mit2e...

Entering the Database Access Module:(aux. table's request)
------------------------------------------
dialing mit2e...

5 row(s) unloaded.

Entering the Data File Reading Module:(aux. table's request)
-------------------------------------------
Reading DBMS output file...Done.

data-list = ((CO NI CF INC NS)
             (JAGUAR PLC 28400.0 19,881,231 UNITED KINGDOM
              1960098750.0)
             (JAGUAR PLC 61300.0 19,871,231 UNITED KINGDOM
              1889960600.0)
             (JAGUAR PLC 83400.0 19,861,231 UNITED KINGDOM
              1231068000.0)
             (JAGUAR PLC 87600.0 19,851,231 UNITED KINGDOM
              1078692500.0)
             (HONDA MOTOR CO LTD 97299.0 19,890,331 JAPAN
              26280469.98663459))

Entering the Data Filter Reading Module:(aux. table's request)
-----------------------------------------------
predicates-1 and predicates-2 are both null.
No conversion is necessary for (FORMAT (FIGURE-PLAIN) (FIGURE-PLAIN))
No conversion is necessary for (UNIT (NIL) (USD))
No conversion is necessary for (SCALE (NIL) (SCALE-1))
No conversion is necessary for (FORMAT (NIL) (DATE-MM/DD/YYYY))
    <2 (SEND-MESSAGE
           (("CO" "NI" "CF" "INC")
            ("JAGUAR PLC" "28400.0" "19,881,231" "UNITED KINGDOM")
            ("JAGUAR PLC" "61300.0" "19,871,231" "UNITED KINGDOM")
            ("JAGUAR PLC" "83400.0" "19,861,231" "UNITED KINGDOM")
            ("JAGUAR PLC" "87600.0" "19,851,231" "UNITED KINGDOM")
            ("HONDA MOTOR CO LTD" "97299.0" "19,890,331" "JAPAN")))
    2> (SEND-MESSAGE DISC-AUX-TABLE :DELETE
           ("ipsharp" (CO NI CF INC)
            (OR (= CO "HONDA MOTOR CO LTD") (= CO "JAGUAR PLC")))
           GLOBAL-DATA-TYPE-CATALOGUE "/usr/cistk/database"
           "auxiliary")

Deleting data in the auxiliary table...

Deleting records in table ipsharp of database auxiliary according to the
constraint:
(CO = 'HONDA MOTOR CO LTD') OR (CO = 'JAGUAR PLC')

Entering Data Conversion Strategy Builder:(delete request)
```

```
----------------------------------------
No sub-strategy is required for the domain NAME
Strategy = ((MONEY-AMOUNT ((SENDER-REPRESENTATION
                           ((FORMAT FIGURE-PLAIN) (UNIT USD)
                            (SCALE SCALE-1)))
                          (RECEIVER-REPRESENTATION
                           ((FORMAT FIGURE-PLAIN) (UNIT NIL) (SCALE NIL)))))
            (DATE ((SENDER-REPRESENTATION  ((FORMAT DATE-MM/DD/YYYY)))
                   (RECEIVER-REPRESENTATION  ((FORMAT NIL)))))
            (YEAR ((SENDER-REPRESENTATION  ((FORMAT YEAR-YYYY)))
                   (RECEIVER-REPRESENTATION  ((FORMAT NIL))))))
```

Entering the Query Parsing module:(delete request)

```
----------------------------------------
Begin parsing the query's projection list...
... query's projection list is parsed.
New projection list: (CO NI CF INC)

Begin parsing constraint...

Predicate at hand: (= CO HONDA MOTOR CO LTD)

Predicate at hand: (= CO JAGUAR PLC)
... constraint parsed.
Initial Query = (ipsharp (CO NI CF INC)
                 (OR (= CO HONDA MOTOR CO LTD) (= CO JAGUAR PLC)))
PreConvertQuery = (ipsharp (CO NI CF INC)
                   (OR (= CO HONDA MOTOR CO LTD) (= CO JAGUAR PLC)))
RemoteQuery = (ipsharp (CO NI CF INC)
               (OR (= CO HONDA MOTOR CO LTD) (= CO JAGUAR PLC)))
predicates-1 = NIL
predicates-2 = NIL

8 row(s) deleted.

    <2 (SEND-MESSAGE 0)
```

Data Filter Step #4:(application's request)

```
-----------------------
```

(conversion of NI data elements in data-from-database)

```
No conversion is necessary for (FORMAT (FIGURE-PLAIN) (FIGURE-PLAIN))
    2> (SEND-MESSAGE USD :CONVERT
           (((MONEY-AMOUNT ((VALUE "28400.0") (FORMAT FIGURE-PLAIN)
                            (UNIT NGBP) (SCALE SCALE-1000)))
             (DATE ((VALUE "19,881,231") (FORMAT DATE-YYYYMMDD))))
            ((MONEY-AMOUNT ((VALUE "61300.0") (FORMAT FIGURE-PLAIN)
                            (UNIT NGBP) (SCALE SCALE-1000)))
             (DATE ((VALUE "19,871,231") (FORMAT DATE-YYYYMMDD))))
            ((MONEY-AMOUNT ((VALUE "83400.0") (FORMAT FIGURE-PLAIN)
                            (UNIT NGBP) (SCALE SCALE-1000)))
             (DATE ((VALUE "19,861,231") (FORMAT DATE-YYYYMMDD))))
            ((MONEY-AMOUNT ((VALUE "87600.0") (FORMAT FIGURE-PLAIN)
                            (UNIT NGBP) (SCALE SCALE-1000)))
             (DATE ((VALUE "19,851,231") (FORMAT DATE-YYYYMMDD))))
            ((MONEY-AMOUNT ((VALUE "97299.0") (FORMAT FIGURE-PLAIN)
                            (UNIT NJPY) (SCALE SCALE-1000)))
```

```
            (DATE ((VALUE "19,890,331") (FORMAT DATE-YYYYMMDD))))))

Entering MONEY-AMOUNT UNIT conversions...
(of NI data elements in data-from-database)

Accessing the I.P. Sharp Currency database...
      3> (SEND-MESSAGE CURRENCYDB :GET-DATA FIGURE-UNIT-CATALOGUE
              (CURRENCY (EXCHANGE-RATE)
                  (OR (AND (= CODE "NGBP") (= DATE "19,881,231"))
                      (OR (AND (= CODE "NGBP") (= DATE "19,871,231"))
                          (OR (AND (= CODE "NGBP")
                                   (= DATE "19,861,231"))
                              (OR (AND (= CODE "NGBP")
                                       (= DATE "19,851,231"))
                                  (AND (= CODE "NJPY")
                                       (= DATE "19,890,331")))))))))


Entering Data Conversion Strategy Builder:(ex. rates' request)
----------------------------------------------
No sub-strategy is required for the domain NAME
Strategy = ((MONEY-AMOUNT ((SENDER-REPRESENTATION
                           ((FORMAT FIGURE-PLAIN) (UNIT USD)
                            (SCALE SCALE-1)))
                       (RECEIVER-REPRESENTATION
                           ((FORMAT FIGURE-PLAIN) (UNIT USD)
                            (SCALE SCALE-0.01)))))
            (DATE ((SENDER-REPRESENTATION  ((FORMAT DATE-YYYYMMDD)))
                   (RECEIVER-REPRESENTATION  ((FORMAT DATE-DD-MM-
YY))))))

Entering the Query Parsing module:(ex. rates' request)
--------------------------------------
Begin parsing the query's projection list...
... query's projection list is parsed.
New projection list: (EXCHANGE-RATE)

Begin parsing constraint...

... constraint parsed.
Initial Query = (CURRENCY (EXCHANGE-RATE)
                      (OR (AND (= CODE NGBP) (= DATE 19,881,231))
                          (OR (AND (= CODE NGBP) (= DATE 19,871,231))
                              (OR (AND (= CODE NGBP) (= DATE 19,861,231))
                                  (OR (AND (= CODE NGBP)
                                           (= DATE 19,851,231))
                                      (AND (= CODE NJPY)
                                           (= DATE 19,890,331)))))))
PreConvertQuery = (CURRENCY (EXCHANGE-RATE CODE DATE)
                        (OR (AND (= CODE NGBP) (= DATE 31-12-88))
                            (OR (AND (= CODE NGBP) (= DATE 31-12-87))
                                (OR (AND (= CODE NGBP) (= DATE 31-12-86))
                                    (OR (AND (= CODE NGBP)
                                             (= DATE 31-12-85))
                                        (AND (= CODE NJPY)
                                             (= DATE 31-03-89)))))))
RemoteQuery = (CURRENCY (EXCHANGE-RATE CODE DATE)
                    (OR (AND (= CODE NGBP) (= DATE 31-12-88))
```

```
                    (OR (AND (= CODE NGBP) (= DATE 31-12-87))
                        (OR (AND (= CODE NGBP) (= DATE 31-12-86))
                            (OR (AND (= CODE NGBP) (= DATE 31-12-85))
                                (AND (= CODE NJPY) (= DATE 31-03-
89)))))))
predicates-1 = NIL
predicates-2 = NIL
```

Entering the Database Access Module:(ex. rates' request)

```
-----------------------------------------
Connecting to currency on machine CURRENCY - I.P. Sharp Currency
database...
Creating the communication script...
...communication script is done.
Connected

7955) 1990-05-02 15:55:40 IPSA

SHARP APL SERVICE

SAVED  1990-05-02 15:54:21

)LOAD 39 MAGIC

SAVED  1990-03-19 03:07:24

1 2 3 4 5 6 7 DAILY DATED 31 12 88 TO 6 1 89
CURRENCY 'NGBP'
  0      0      0    182.25 180.7  179.7  178

1 2 3 4 5 6 7 DAILY DATED 31 12 87 TO 6 1 88
CURRENCY 'NGBP'
188.6    0      0      0    187.73 183.05 180.6

1 2 3 4 5 6 7 DAILY DATED 31 12 86 TO 6 1 87
CURRENCY 'NGBP'
148.25   0    149      0      0    147.28 147.62

1 2 3 4 5 6 7 DAILY DATED 31 12 85 TO 6 1 86
CURRENCY 'NGBP'
144.5    0    145.05 143.8    0      0    143.55

1 2 3 4 5 6 7 DAILY DATED 31 03 89 TO 6 4 89
CURRENCY 'NJPY'
0.7531821948 0            0            0.7572889057 0.7639419404
0.7587253414 0.7579777155

Disconnected
```

Entering the Data File Reading Module:(ex. rates' request)

```
-----------------------------------------
data-list = ((EXCHANGE-RATE CODE DATE) (182.25 NGBP 31-12-88)
             (188.6 NGBP 31-12-87) (148.25 NGBP 31-12-86)
             (144.5 NGBP 31-12-85) (0.7531821948 NJPY 31-03-89))
predicates-1 and predicates-2 are both null.
```

Data Filter Step #4:(ex. rates' request)

```
----------------------
```

```
converting data into the sender's data definitions.
No conversion is necessary for (FORMAT (FIGURE-PLAIN) (FIGURE-PLAIN))
No conversion is necessary for (UNIT (USD) (USD))

Entering MONEY-AMOUNT SCALE convertions...

        4> (SEND-MESSAGE SCALE-1 :CONVERT
               (((MONEY-AMOUNT ((VALUE "182.25") (FORMAT FIGURE-PLAIN)
                           (UNIT USD) (SCALE SCALE-0.01))))
                ((MONEY-AMOUNT ((VALUE "188.6") (FORMAT FIGURE-PLAIN)
                           (UNIT USD) (SCALE SCALE-0.01))))
                ((MONEY-AMOUNT ((VALUE "148.25") (FORMAT FIGURE-PLAIN)
                           (UNIT USD) (SCALE SCALE-0.01))))
                ((MONEY-AMOUNT ((VALUE "144.5") (FORMAT FIGURE-PLAIN)
                           (UNIT USD) (SCALE SCALE-0.01))))
                ((MONEY-AMOUNT ((VALUE "0.7531821948") (FORMAT FIGURE-
PLAIN)
                           (UNIT USD) (SCALE SCALE-0.01))))))

        <4 (SEND-MESSAGE
               (((MONEY-AMOUNT ((VALUE "1.8225") (FORMAT FIGURE-PLAIN)
                           (UNIT USD) (SCALE SCALE-1))))
                ((MONEY-AMOUNT ((VALUE "1.886") (FORMAT FIGURE-PLAIN)
                           (UNIT USD) (SCALE SCALE-1))))
                ((MONEY-AMOUNT ((VALUE "1.4825") (FORMAT FIGURE-PLAIN)
                           (UNIT USD) (SCALE SCALE-1))))
                ((MONEY-AMOUNT ((VALUE "1.445") (FORMAT FIGURE-PLAIN)
                           (UNIT USD) (SCALE SCALE-1))))
                ((MONEY-AMOUNT ((VALUE "0.007531821948000001")
                           (FORMAT FIGURE-PLAIN) (UNIT USD)
                           (SCALE SCALE-1))))))
MONEY-AMOUNT SCALE conversions done.

      <3 (SEND-MESSAGE
             (("EXCHANGE-RATE") ("1.8225") ("1.886") ("1.4825")
              ("1.445") ("0.007531821948000001")))

Unit convertions into USD...(of NI data elements in data-from-database)

MONEY-AMOUNT UNIT conversions done.

    <2 (SEND-MESSAGE
           (((MONEY-AMOUNT ((VALUE "51759.0") (FORMAT FIGURE-PLAIN)
                       (UNIT USD) (SCALE SCALE-1000)))
             (DATE ((VALUE "19,881,231") (FORMAT DATE-YYYYMMDD))))
            ((MONEY-AMOUNT ((VALUE "115611.8") (FORMAT FIGURE-PLAIN)
                       (UNIT USD) (SCALE SCALE-1000)))
             (DATE ((VALUE "19,871,231") (FORMAT DATE-YYYYMMDD))))
            ((MONEY-AMOUNT ((VALUE "123640.5") (FORMAT FIGURE-PLAIN)
                       (UNIT USD) (SCALE SCALE-1000)))
             (DATE ((VALUE "19,861,231") (FORMAT DATE-YYYYMMDD))))
            ((MONEY-AMOUNT ((VALUE "126582.0") (FORMAT FIGURE-PLAIN)
                       (UNIT USD) (SCALE SCALE-1000)))
             (DATE ((VALUE "19,851,231") (FORMAT DATE-YYYYMMDD))))
            ((MONEY-AMOUNT ((VALUE "732.8387437184522") (FORMAT FIGURE-
PLAIN)
                       (UNIT USD) (SCALE SCALE-1000)))
             (DATE ((VALUE "19,890,331") (FORMAT DATE-YYYYMMDD))))))
```

```
Entering MONEY-AMOUNT UNIT conversions...(of NI data elements in data-from-
database)

Entering MONEY-AMOUNT SCALE conversions...(of NI data elements in data-from-
database)

    2> (SEND-MESSAGE SCALE-1 :CONVERT
     (((MONEY-AMOUNT ((VALUE "51759.0") (FORMAT FIGURE-PLAIN)
                      (UNIT USD) (SCALE SCALE-1000)))
             (DATE ((VALUE "19,881,231") (FORMAT DATE-YYYYMMDD))))
      ((MONEY-AMOUNT ((VALUE "115611.8") (FORMAT FIGURE-PLAIN)
                      (UNIT USD) (SCALE SCALE-1000)))
             (DATE ((VALUE "19,871,231") (FORMAT DATE-YYYYMMDD))))
      ((MONEY-AMOUNT ((VALUE "123640.5") (FORMAT FIGURE-PLAIN)
                      (UNIT USD) (SCALE SCALE-1000)))
             (DATE ((VALUE "19,861,231") (FORMAT DATE-YYYYMMDD))))
      ((MONEY-AMOUNT ((VALUE "126582.0") (FORMAT FIGURE-PLAIN)
                      (UNIT USD) (SCALE SCALE-1000)))
             (DATE ((VALUE "19,851,231") (FORMAT DATE-YYYYMMDD))))
      ((MONEY-AMOUNT ((VALUE "732.8387437184522") (FORMAT FIGURE-PLAIN)
                      (UNIT USD) (SCALE SCALE-1000)))
             (DATE ((VALUE "19,890,331") (FORMAT DATE-YYYYMMDD))))))

    <2 (SEND-MESSAGE
           (((MONEY-AMOUNT ((VALUE "51759000") (FORMAT FIGURE-PLAIN)
                      (UNIT USD) (SCALE SCALE-1)))
             (DATE ((VALUE "19,881,231") (FORMAT DATE-YYYYMMDD))))
            ((MONEY-AMOUNT ((VALUE "115611800") (FORMAT FIGURE-PLAIN)
                      (UNIT USD) (SCALE SCALE-1)))
             (DATE ((VALUE "19,871,231") (FORMAT DATE-YYYYMMDD))))
            ((MONEY-AMOUNT ((VALUE "123640500") (FORMAT FIGURE-PLAIN)
                      (UNIT USD) (SCALE SCALE-1)))
             (DATE ((VALUE "19,861,231") (FORMAT DATE-YYYYMMDD))))
            ((MONEY-AMOUNT ((VALUE "126582000") (FORMAT FIGURE-PLAIN)
                      (UNIT USD) (SCALE SCALE-1)))
             (DATE ((VALUE "19,851,231") (FORMAT DATE-YYYYMMDD))))
            ((MONEY-AMOUNT ((VALUE "732838.7437184522") (FORMAT FIGURE-
PLAIN)
                      (UNIT USD) (SCALE SCALE-1)))
             (DATE ((VALUE "19,890,331") (FORMAT DATE-YYYYMMDD))))))

MONEY-AMOUNT conversions done.(of NI data elements in data-from-database)

DATE FORMAT converions...(of CF data elements in data-from-database)

    2> (SEND-MESSAGE DATE-MM/DD/YYYY :CONVERT
           (((DATE ((VALUE "19,881,231") (FORMAT DATE-YYYYMMDD))))
            ((DATE ((VALUE "19,871,231") (FORMAT DATE-YYYYMMDD))))
            ((DATE ((VALUE "19,861,231") (FORMAT DATE-YYYYMMDD))))
            ((DATE ((VALUE "19,851,231") (FORMAT DATE-YYYYMMDD))))
            ((DATE ((VALUE "19,890,331") (FORMAT DATE-YYYYMMDD))))))

    <2 (SEND-MESSAGE
           (((DATE ((VALUE "12/31/1988") (FORMAT DATE-MM/DD/YYYY))))
            ((DATE ((VALUE "12/31/1987") (FORMAT DATE-MM/DD/YYYY))))
            ((DATE ((VALUE "12/31/1986") (FORMAT DATE-MM/DD/YYYY))))
            ((DATE ((VALUE "12/31/1985") (FORMAT DATE-MM/DD/YYYY))))
```

```
            ((DATE ((VALUE "03/31/1989") (FORMAT DATE-MM/DD/YYYY))))))
```

DATE FORMAT conversions done. (of CF data elements in data-from-database)

```
  <1 (SEND-MESSAGE
        ((("CO" "NI" "CF") ("JAGUAR PLC" "51759000" "12/31/1988")
         ("JAGUAR PLC" "115611800" "12/31/1987")
         ("JAGUAR PLC" "123640500" "12/31/1986")
         ("JAGUAR PLC" "126582000" "12/31/1985")
         ("HONDA MOTOR CO LTD" "732838.7437184522" "03/31/1989")))
```

FINAL RESULTS:

```
(("CO" "NI" "CF")
 ("JAGUAR PLC" "51759000" "12/31/1988"
 ("JAGUAR PLC" "115611800" "12/31/1987")
 ("JAGUAR PLC" "123640500" "12/31/1986")
 ("JAGUAR PLC" "126582000" "12/31/1985")
 ("HONDA MOTOR CO LTD" "732838.7437184522"
"03/31/1989"))
```

Appendix[2]: Common Lisp Code

```
;;; DATA-TYPE & ATTRIBUTE CATALOGUES

;;; root object CATALOGUE
(make-object 'catalogue)

;;; APPLICATION
;;; data type catalog
(make-object 'global-data-type-catalogue
      ('superiors 'catalogue)
      ('data-types t 'multiple-value-f)
      ('data-types 'figure)
      ('data-types 'date)
      ('data-types 'year)
      ('data-types 'name)
      ('figure 'FIGURE-PLAIN 'format)
      ('figure 'USD 'unit)
      ('figure 'SCALE-1 'scale)
      ('date 'DATE-MM/DD/YYYY 'format)
      ('year 'YEAR-YYYY 'format)
      ('methods t 'multiple-value-f)
      ('methods '(:get-data-type get-database-data-types)))

;;; DATABASE: I.P. SHARP DISCLOSURE
;;; data type catalog
(make-object 'disc-data-type-catalogue
      ('superiors 'catalogue)
      ('data-types t 'multiple-value-f)
      ('data-types 'figure)
      ('data-types 'date)
      ('data-types 'year)
      ('data-types 'name)
      ('figure 'FIGURE-DISC 'format)
      ('figure '(get-unit-disc IN) 'unit)
      ('figure 'SCALE-1000 'scale)
      ('date 'DATE-YYYYMMDD 'format)
      ('year 'YEAR-YYYY 'format)
      ('attributes-catalogue 'disc-attributes-catalogue)
      ('methods t 'multiple-value-f)
      ('methods '(:get-data-type get-database-data-types))
      ('methods '(:get-attributes-of-data-type get-attributes-of-data-
type)))

;;; Attribute catalogue
(make-object 'disc-attributes-catalogue
      ('superiors 'disc-data-type-catalogue)
      ('attributes t 'multiple-value-f)
      ('attributes 'co) ; company name
      ('attributes 'pc) ; primary sic code
      ('attributes 'sc) ; list of sic codes
      ('attributes 'year) ; year (YYYY)
      ('attributes 'cf) ; financial reporting date (YYYYMMDD)
      ('attributes 'ni) ; net income
      ('attributes 'ns) ; net sales
      ('attributes 'in) ; country of incorporation (DE, JAPAN)
      ('attributes 'te) ; telephone number
      ('attributes 'em) ; number of employee
```

```
      ('attributes 'ad1) ; address
      ('co 'name) ; now match each attribute to its data-type
      ('pc 'name)
      ('sc 'name)
      ('year 'year)
      ('cf 'date)
      ('ni 'figure)
      ('ns 'figure)
      ('in 'name)
      ('in 'inc 'aux-attribute)
      ('te 'name)
      ('em 'name)
      ('ad1 'name)
      ('methods t 'multiple-value-f))

;;; AUXILIARY I.P. SHARP DISCLOSURE
;;; Data type catalogue
(make-object 'disc-aux-data-type-catalogue
      ('superiors 'catalogue)
      ('data-types t 'multiple-value-f)
      ('data-types 'figure)
      ('data-types 'date)
      ('data-types 'year)
      ('data-types 'name)
      ('figure 'FIGURE-PLAIN 'format)
      ('attributes-catalogue 'disc-aux-attributes-catalogue)
      ('methods t 'multiple-value-f)
      ('methods '(:get-data-type get-database-data-types))
      ('methods '(:get-attributes-of-data-type get-attributes-of-data-
type)))

;;; Attribute catalogue
(make-object 'disc-aux-attributes-catalogue
      ('superiors 'disc-aux-data-type-catalogue)
      ('superiors 'disc-data-type-catalogue)
      ('attributes t 'multiple-value-f)
      ('attributes 'co) ; company name
      ('attributes 'pc) ; primary sic code
      ('attributes 'sc) ; list of sic codes
      ('attributes 'yr) ; yr (YYYY)
      ('attributes 'cf) ; financial reporting date (YYYYMMDD)
      ('attributes 'ni) ; net income
      ('attributes 'ns) ; net sales
      ('attributes 'inc) ; country of incorporation (DE, JAPAN)
      ('attributes 'te) ; telephone number
      ('attributes 'em) ; number of employee
      ('attributes 'ad1) ; address
      ('co 'name) ; now match each attribute to its data-type
      ('pc 'name)
      ('sc 'name)
      ('yr 'year)
      ('cf 'date)
      ('ni 'figure)
      ('ns 'figure)
      ('in 'name)
      ('inc 'in 'aux-attribute)
      ('te 'name)
      ('em 'name)
```

```
      ('adl 'name)
      ('methods t 'multiple-value-f))

;;; ============================ DATABASE: DATALINE FINSBURY
=============
;;; data type catalog
(make-object 'dataln-data-type-catalogue
      ('superiors 'catalogue)
      ('data-types t 'multiple-value-f)
      ('data-types 'figure)
      ('data-types 'date)
      ('data-types 'year)
      ('data-types 'name)
      ('figure 'FIGURE-DATALN 'format)
      ('figure '(get-unit-dataln CURRENCY) 'unit)
      ('figure '(get-scale-dataln SCALE) 'scale)
      ('date 'DATE-DD-MM-YY 'format)
      ('year 'YEAR-YY 'format)
      ('attributes-catalogue 'dataln-attributes-catalogue)
      ('methods t 'multiple-value-f)
      ('methods '(:get-data-type get-database-data-types))
      ('methods '(:get-attributes-of-data-type get-attributes-of-data-
type)))

;;; Attribute catalogue
(make-object 'dataln-attributes-catalogue
      ('superiors 'dataln-data-type-catalogue)
      ('attributes t 'multiple-value-f)
      ('attributes 'companyname) ; company name
      ('attributes 'code) ; company code
      ('attributes 'yr) ; year (YY)
      ('attributes 'periodending) ; financial reporting date (DD-MM-YY)
      ('attributes 'efo) ; net income
      ('attributes 'sales) ; net sales
      ('attributes 'country) ; country of incorporation (e.g., JAPAN)
      ('attributes 'currency) ; currency (e.g., Yen)
      ('attributes 'scale) ; scale (e.g., (000s))
      ('companyname 'name) ; now match each attribute to its data-type
      ('code 'name)
      ('yr 'year)
      ('periodending 'date)
      ('efo 'figure)
      ('sales 'figure)
      ('country 'name)
      ('currency 'name)
      ('scale 'name)
      ('methods t 'multiple-value-f))

;;; DATABASE: IPSHARP CURRENCY
;;; data type catalog
(make-object 'currency-data-type-catalogue
      ('superiors 'catalogue)
      ('data-types t 'multiple-value-f)
      ('data-types 'figure)
      ('data-types 'date)
      ('data-types 'name)
      ('figure 'FIGURE-PLAIN 'format)
      ('figure 'USD 'unit)
```

```
      ('figure 'SCALE-0.01 'scale)
      ('date 'DATE-DD-MM-YY 'format)
      ('attributes-catalogue 'currency-attributes-catalogue)
      ('methods t 'multiple-value-f)
      ('methods '(:get-data-type get-database-data-types))
      ('methods '(:get-attributes-of-data-type get-attributes-of-data-
type)))

;;; Attribute catalogue
(make-object 'currency-attributes-catalogue
      ('superiors 'currency-data-type-catalogue)
      ('attributes t 'multiple-value-f)
      ('attributes 'code) ; e.g., NJPY
      ('attributes 'exchange-rate) ; e.g, actual value
      ('attributes 'date)
      ('code 'name)
      ('exchange-rate 'figure)
      ('date 'date)
      ('methods t 'multiple-value-f))

;;; Conversion Objects

;;; root object DATA-TYPE
(make-object 'data-type)

;;; DATA TYPE: MONEY-AMOUNT
(make-object 'money-amount
           ('superiors 'data-type)
           ('data-type 'money-amount)
           ('components t 'multiple-value-f)
           ('components 'format)
           ('components 'unit)
           ('components 'scale)
           ('format 'figure-format)
           ('unit 'figure-unit)
           ('scale 'figure-scale)
           ('methods t 'multiple-value-f)
           ('methods '(:get-components get-components))
           ('methods '(:get-conv-arguments get-conv-arguments)))

(make-object 'figure-format
           ('superiors 'money-amount)
           ('component 'format)
           ('representations t 'multiple-value-f)
           ('representations 'figure-plain) ; component IDs
           ('representations 'figure-disc)
           ('representations 'figure-dataln)
           ('reference 'figure-plain)
           ('arguments t 'multiple-value-f)
           ('methods t 'multiple-value-f)
           ('methods '(:get-representations get-representation)))

(make-object 'figure-plain
           ('superiors 'figure-format)
           ('methods t 'multiple-value-f)
           ('methods '(:convert data-list-convert-into-FIGURE-PLAIN)))

(make-object 'figure-disc
```

```
          ('superiors 'figure-format)
          ('methods t 'multiple-value-f)
          ('methods '(:convert data-list-convert-into-FIGURE-DISC)))

(make-object 'figure-dataln
          ('superiors 'figure-format)
          ('methods t 'multiple-value-f)
          ('methods '(:convert data-list-convert-into-FIGURE-DATALN)))

(make-object 'figure-unit
          ('superiors 'money-amount)
          ('component 'unit)
          ('representations t 'multiple-value-f)
          ('representations 'usd)
          ('representations 'jpy)
          ('representations 'gbp)
          ('representations 'frf)
          ('reference 'usd)
          ('arguments t 'multiple-value-f)
          ('arguments 'date)
          ; data type catalogue used to prepare the query sent to the
          ; LQP I.P. Sharp Currency
          ('data-type-catalogue 'figure-unit-catalogue)
          ; database from which exchange-rates can be obtained
          ('database 'currencydb)
          ('methods t 'multiple-value-f)
          ('methods '(:get-representations get-representations)))

(make-object 'figure-unit-catalogue
          ('superiors t 'multiple-value-f)
          ('superiors 'catalogue)
          ('superiors 'figure-unit)
          ('datatypes t 'multiple-value-f)
          ('datatypes 'figure)
          ('datatypes 'date)
          ('datatypes 'name)
          ; this IDs are changed for each request sent to the currency
LQP
          ('figure 'FIGURE-PLAIN 'format)
          ('figure 'USD 'unit)
          ('figure 'SCALE-1 'scale)
          ('date 'DATE-YYYYMMDD 'format))

(make-object 'USD
          ('superiors 'figure-unit)
          ('methods t 'multiple-value-f)
          ('methods '(:convert data-list-convert-into-USD)))

(make-object 'GBP
          ('superiors 'figure-unit)
          ('methods t 'multiple-value-f)
          ('methods '(:convert data-list-convert-into-GBP)))

(make-object 'JPY
          ('superiors 'figure-unit)
          ('methods t 'multiple-value-f)
          ('methods '(:convert data-list-convert-into-JPY)))
```

```
(make-object 'FRF
           ('superiors 'figure-unit)
           ('methods t 'multiple-value-f)
           ('methods '(:convert data-list-convert-into-FRF)))

(make-object 'figure-scale
           ('superiors 'money-amount)
           ('component 'scale)
           ('representations t 'multiple-value-f)
           ('representations 'scale-1)
           ('representations 'scale-1000)
           ('representations 'scale-0.01)
           ('reference 'scale-1)
           ('arguments t 'multiple-value-f)
           ('methods t 'multiple-value-f)
           ('methods '(:get-representations get-representations)))

(make-object 'SCALE-1
           ('superiors 'figure-scale)
           ('methods t 'multiple-value-f)
           ('methods '(:convert data-list-convert-into-SCALE-1)))

(make-object 'SCALE-1000
           ('superiors 'figure-scale)
           ('methods t 'multiple-value-f)
           ('methods '(:convert data-list-convert-into-SCALE-1000)))

(make-object 'SCALE-0.01
           ('superiors 'figure-scale)
           ('methods t 'multiple-value-f)
           ('methods '(:convert data-list-convert-into-SCALE-0.01)))

;;; DATA TYPE: DATE
(make-object 'date
           ('superiors 'data-type)
           ('data-type 'date)
           ('components t 'multiple-value-f)
           ('components 'format) ; representation components of DATE
           ('format 'date-format) ; date-format is an object name
           ('methods t 'multiple-value-f)
           ('methods '(:get-components get-components))
           ('methods '(:get-conv-arguments get-conv-arguments)))

(make-object 'DATE-FORMAT
           ('superiors 'date)
           ('component 'format)
           ('representations t 'multiple-value-f)
           ('representations 'date-mm/dd/yyyy)
           ('representations 'date-dd-mm-yy)
           ('representations 'date-yyyymmdd)
           ('representations 'date-ddsmmsyy)
           ('reference 'date-mm/dd/yyyy)
           ('arguments t 'multiple-value-f)
           ('methods t 'multiple-value-f)
           ('methods '(:get-representations get-representations)))

(make-object 'date-mm/dd/yyyy
           ('superiors 'date-format)
```

```
          ('methods t 'multiple-value-f)
          ('methods '(:convert data-list-convert-into-DATE-
MM/DD/YYYY)))

(make-object 'date-dd-mm-yy
          ('superiors 'date-format)
          ('methods t 'multiple-value-f)
          ('methods '(:convert data-list-convert-into-DATE-DD-MM-YY)))

;;; e.g., "24 03 90"
(make-object 'date-ddsmmsyy
          ('superiors 'date-format)
          ('methods t 'multiple-value-f)
          ('methods '(:convert data-list-convert-into-DATE-DDsMMsYY)))

(make-object 'date-YYYYMMDD
          ('superiors 'date-format)
          ('methods t 'multiple-value-f)
          ('methods '(:convert data-list-convert-into-DATE-YYYYMMDD)))


;;; DATA TYPE: YEAR
(make-object 'year
          ('superiors 'data-type)
          ('data-type 'year)
          ('components t 'multiple-value-f)
          ('components 'format) ; representation components of YEAR
          ('format 'year-format) ; year-format is an object name
          ('methods t 'multiple-value-f)
          ('methods '(:get-components get-components))
          ('methods '(:get-conv-arguments get-conv-arguments)))

(make-object 'YEAR-FORMAT
          ('superiors 'year)
          ('component 'format)
          ('representations t 'multiple-value-f)
          ('representations 'year-yyyy)
          ('representations 'year-yy)
          ('reference 'year-yyyy)
          ('arguments t 'multiple-value-f)
          ('methods t 'multiple-value-f)
          ('methods '(:get-representations get-representations)))

(make-object 'year-yyyy
          ('superiors 'year-format)
          ('methods t 'multiple-value-f)
          ('methods '(:convert data-list-convert-into-YEAR-YYYY)))


(make-object 'year-yy
          ('superiors 'year-format)
          ('methods t 'multiple-value-f)
          ('methods '(:convert data-list-convert-into-YEAR-YY)))
```

***

```
;;; Database objects

;;; DEFINITION OF BRIDGE OBJECT: LOCAL DBMS
```

```
(make-object 'informix-2e
           ('machine-name "mit2e")
           ('type-of-DBMS "informix")
           ('local-DBMS?  t)
           ('database-directory "/usr/cistk/database" 'default)
           ('database "localdb" 'default)
           ('bridge-directory
"/usr/cistk/brigaldi/Thesis/Work/Informix")
           ('communications-script "2EFETCHn")
           ('efficient-comm-script "2EFETCHe")
           ('account  "IRRELEVANT")
           ('password "IRRELEVANT")
           ('data-file-reader 'read-standard-table)
           ('download-script "d-script")
           ('delete-script "del-script")
           ('methods t 'multiple-value-f)
           ('methods '(:self-info display-INFORMIX2E-self-info))
           ('methods '(:get-tables get-INFORMIX2E-tables))
           ('methods '(:get-columns get-INFORMIX2E-columns))
           ('methods '(:get-data get-INFORMIX2E-data))
           ('methods '(:download download-INFORMIX2E-data))
           ('methods '(:delete delete-INFORMIX2E-data)))

;;; INSTANCE OBJECT FROM INFORMIX-2E: AUXILIARY IPSHARP CURRENCY
(create-instance 'informix-2e 'currency-aux-table)
(put-object 'currency-aux-table 'database "auxiliary")
(put-object 'currency-aux-table 'table "curr_aux")
(put-object 'currency-aux-table 'database-directory
"/usr/cistk/database")
(put-object 'currency-aux-table 'switch-conversion-facility 'on)
(put-object 'currency-aux-table 'data-type-catalogue 'curr-aux-data-
type-catalogue)
(put-object 'currency-aux-table 'debug-id "auxcurr")
(put-object 'currency-aux-table 'query-syntax-translator 'form-SQL)
(put-object 'currency-aux-table 'script-writer "IRRELEVANT")
(put-object 'currency-aux-table 'communication-server "2EFETCHe")
(put-object 'currency-aux-table 'predicate-filter 'informix-predicate-
filter)

;;; INSTANCE OBJECT FROM INFORMIX-2E: AUXILIARY FINSBURY DATALINE
(create-instance 'informix-2e 'dataln-aux-table)
(put-object 'dataln-aux-table 'database "auxiliary")
(put-object 'dataln-aux-table 'table "dataline")
(put-object 'dataln-aux-table 'database-directory "/usr/cistk/database")
(put-object 'dataln-aux-table 'switch-conversion-facility 'on)
(put-object 'dataln-aux-table 'data-type-catalogue 'dataln-aux-data-
type-catalogue)
(put-object 'dataln-aux-table 'debug-id "auxdataln")
(put-object 'dataln-aux-table 'query-syntax-translator 'form-SQL)
(put-object 'dataln-aux-table 'script-writer "IRRELEVANT")
(put-object 'dataln-aux-table 'communication-server "2EFETCHe")
(put-object 'dataln-aux-table 'predicate-filter 'informix-predicate-
filter)

;; INSTANCE OBJECT FROM INFORMIX-2E: AUXILIARY IPSHARP DISCLOSURE
(create-instance 'informix-2e 'disc-aux-table)
(put-object 'disc-aux-table 'database "auxiliary")
(put-object 'disc-aux-table 'table "ipsharp")
```

```
(put-object 'disc-aux-table 'database-directory "/usr/cistk/database")
(put-object 'disc-aux-table 'switch-conversion-facility 'on)
(put-object 'disc-aux-table 'data-type-catalogue 'disc-aux-data-type-
catalogue)
(put-object 'disc-aux-table 'debug-id "auxdisc")
(put-object 'disc-aux-table 'query-syntax-translator 'form-SQL)
(put-object 'disc-aux-table 'script-writer "IRRELEVANT")
(put-object 'disc-aux-table 'communication-server "2EFETCHe")
(put-object 'disc-aux-table 'predicate-filter 'informix-predicate-
filter)

;;; DEFINITION OF BRIDGE OBJECT: FINSBURY DATALINE
(reset-frame 'finsbury)
(make-object 'finsbury
           ('machine-name "FINSBURY - EEC Financial Database")
           ('type-of-DBMS "menu")
           ('local-DBMS?  nil)
           ('database-directory "IRRELEVANT" 'default)
           ('database "datalinedb" 'default)
           ('bridge-directory
"/usr/cistk/brigaldi/Thesis/Work/Dataline")
           ('communications-script "communicate")
           ('efficient-comm-script "irrelevant")
           ('phone-number "92920662" 'default)
           ('account  "c202430" 'default)
           ('password "rna *325250" 'default)
           ('methods t 'multiple-value-f)
           ('methods '(:self-info display-DATALINE-self-info))
           ('methods '(:get-tables get-DATALINE-tables))
           ('methods '(:get-columns get-DATALINE-columns))
           ('methods '(:get-data get-DATALINE-data)))

;;; INSTANCE OBJECT FROM FINSBURY: FINSBURY DATALINE
(create-instance 'finsbury 'datalinedb)
(put-object 'datalinedb 'debug-id "dataln")
(put-object 'datalinedb 'database "dataline")
(put-object 'datalinedb 'database-directory "IRRELEVANT")
(put-object 'datalinedb 'switch-conversion-facility 'on)
(put-object 'datalinedb 'data-type-catalogue 'dataln-data-type-
catalogue)
(put-object 'datalinedb 'auxiliary-table 'dataln-aux-table)
(put-object 'disc-aux-table 'communication-server "2EFETCHe")
(put-object 'datalinedb 'predicate-filter 'dataln-predicate-filter)

;;; DEFINITION OF LQP OBJECT CLASS IPSHARP DISCLOSURE
(make-object 'ipsharp-disc
           ('machine-name "I.P.SHARP - N. American Financial Database")
           ('type-of-DBMS "APL")
           ('local-DBMS?  nil)
           ('database-directory "IRRELEVANT" 'default)
           ('database "disclosure" 'default)
           ('bridge-directory
"/usr/cistk/brigaldi/Thesis/Work/Disclosure")
           ('communications-script "lqpdriver")
           ('efficient-comm-script "lqpdriver")
           ('account  ")1769739:SLOAN" 'default)
           ('password ")LOAD 39 MAGIC" 'default)
           ('terminal-type "IRRELEVANT" 'default)
```

```
            ('methods t 'multiple-value-f)
            ('methods '(:self-info display-IPSHARP-self-info))
            ('methods '(:get-tables get-IPSHARP-tables))
            ('methods '(:get-columns get-IPSHARP-columns))
            ('methods '(:get-data get-IPSHARP-data)))

;;; INSTANCE OBJECT FROM IPSHARP-DISC: IPSHARP DISCLOSURE
(create-instance 'ipsharp-disc 'disclosuredb)
(put-object 'disclosuredb 'debug-id "disc")
(put-object 'disclosuredb 'lqp-directory
"/usr/cistk/brigaldi/Thesis/Work/Disclosure/")
(put-object 'disclosuredb 'query-syntax-translator 'form-SQL)
(put-object 'disclosuredb 'script-writer "lqpdriver")
(put-object 'disclosuredb 'switch-conversion-facility 'on)
(put-object 'disclosuredb 'data-type-catalogue 'disc-data-type-
catalogue)
(put-object 'disclosuredb 'auxiliary-table 'disc-aux-table)
(put-object 'disclosuredb 'delete-keys t 'multiple-value-f)
(put-object 'disclosuredb 'delete-keys 'co)
(put-object 'disclosuredb 'predicate-filter 'disc-predicate-filter)

;;; DEFINITION OF BRIDGE OBJECT: IPSHARP CURRENCY
(make-object 'ipsharp-curr
            ('machine-name "CURRENCY - I.P. Sharp Currency database")
            ('type-of-DBMS "APL")
            ('local-DBMS? nil)
            ('database-directory "IRRELEVANT" 'default)
            ('database "currency" 'default)
            ('bridge-directory
"/usr/cistk/brigaldi/Thesis/Work/Currency")
            ('communications-script "csgen")
            ('efficient-comm-script "irrelevant")
            ('phone-number "97236715" 'default)
            ('account  ")1769739:SLOAN" 'default)
            ('password ")LOAD 39 MAGIC" 'default)
            ('terminal-type "IRRELEVANT" 'default)
            ('methods t 'multiple-value-f)
            ('methods '(:self-info display-CURRENCY-self-info))
            ('methods '(:get-tables get-CURRENCY-tables))
            ('methods '(:get-columns get-CURRENCY-columns))
            ('methods '(:get-data get-CURRENCY-data)))

;;; INSTANCE OBJECT FROM IPSHARP-CURR: IPSHARP CURRENCY
(create-instance 'ipsharp-curr 'currencydb)
(put-object 'currencydb 'debug-id "curr")
(put-object 'currencydb 'lqp-directory
"/usr/cistk/brigaldi/Thesis/Work/Currency/")
(put-object 'currencydb 'query-syntax-translator 'ipcurr-syntax-
translate)
(put-object 'currencydb 'script-writer "csgen")
(put-object 'currencydb 'communication-server "thesis1")
(put-object 'currencydb 'pre-read-filter 'ip-pre-read-filter)
(put-object 'currencydb 'data-file-reader 'read-ip-currency)
(put-object 'currencydb 'switch-conversion-facility 'on)
(put-object 'currencydb 'data-type-catalogue 'currency-data-type-
catalogue)
(put-object 'currencydb 'auxiliary-table 'currency-aux-table)
(put-object 'currencydb 'predicate-filter 'currency-predicate-filter)
```

```
                                    ***
;;; Object methods

;;; Update an object (KOREL extension)
(defun update-object (catalogue-name slot facet value)
(let ((old-value (get-object catalogue-name slot facet)))
  (cond
   ((null old-value) ; no old value
    (remove-object catalogue-name slot old-value facet)
    (put-object catalogue-name slot value facet))
   (T ; there was already a value
    (remove-object catalogue-name slot old-value facet)
    (put-object catalogue-name slot value facet)))))

;;; Method: get-database-data-types
;;; give the list of data-types supported by the database
(defun get-database-data-types ()
(let ((object-name (get-current-object)))
  (get-object object-name 'data-types)))

;;; Method: get-attribute-of-data-type
;;; Returns a list of the database attributes belonging to a given data-
type.
(defun get-attributes-of-data-type (data-type)
;; need to check if the data-type exists.
(let* ((object-name (get-current-object))
       (att-catalg (get-object object-name 'attributes-catalogue))
       (attributes-list (get-object att-catalg 'attributes))
       (returned-attributes 'NIL))
  (dolist (attribute attributes-list returned-attributes)
        (let ((data-type-at-hand (get-object att-catalg attribute)))
          (if (eq data-type-at-hand data-type)
            (pushnew attribute returned-attributes))))
  (reverse returned-attributes)))

;;; Method: get-components
;;; return a list of all components desribing the data types
(defun get-components ()
(let ((object-name (get-current-object)))
  (get-object object-name 'components)))

;;; Method: get-conv-arguments
;;; get the arguments required for the conversion of a given component
(defun get-conv-arguments (component)
(let* ((object-name (get-current-object))
       (component-object (get-object object-name component)))
  (cond
   ((null component-object) (format T "~%Error: the data type ~A doesn't
support the component ~A." object-name component))
   (T (get-object component-object 'arguments)))))

;;; Method: get-representations returns the list of representation IDs
of the
;;; object.
(defun get-representations ()
(let ((object-name (get-current-object)))
  (get-object object-name 'representations)))
```

```
                                        ***
;;; Data Conversion Strategy Building

;;; procedure: build-lqp-data-conversion-strategy

;;; inputs:
;;; 1) sender-dom-catalg
;;; 2) receiver-dom-catalg
;;; assumptions: domains supported by the sender are also supported by
the
;;; receiver.

(defun build-lqp-data-conversion-strategy (sender-dom-catalg
receiver-dom-catalg)
(let (
      (domains-list (get-object receiver-dom-catalg 'data-types))
      (strategy 'NIL))

; domains-list = (figure date year name)
; strategy is initialized to NIL
; process each domain one at a time to build the corresponding sub-
strategy

(dolist (domain domains-list strategy)
  (lqp-print 'verbose "~%Domain examined = ~A" domain)
  (let ((sub-strategy 'NIL) ; initialization
      (components-list (get-object domain 'components))
      (sender-rep 'NIL) ; working var. to host sender representation.
      (receiver-rep 'NIL) ; ditto for receiver
      (ID-arguments 'NIL)
      (conv-arguments 'NIL))

    (dolist (component components-list) ; one component at a time
      (let ((sender-ID (get-object sender-dom-catalg domain component))
            (receiver-ID (get-object receiver-dom-catalg domain
component))
            (conv-args (send-message domain :get-conv-arguments
component)))
        ; sender-ID = USD, or (get-unit-disc IN)
        ; receiver-ID = JPY or (get-unit-disc IN)
        ; conv-args = (date location)

        (setq sender-rep (append sender-rep (list (list component
sender-ID))))
        (setq receiver-rep (append receiver-rep (list (list component
receiver-ID))))
        (when (and (listp receiver-ID) receiver-ID) ; if receiver-
ID=NIL,skip
            (setf ID-arguments (append ID-arguments (cdr receiver-ID))))

        ; add the conversion arguments only if there is a need to, that
is
        ; when the sender-rep and receiver-rep don't match, and the
        ; receiver-ID is not NIL
        (if (null receiver-ID)
            (setq conv-args 'nil))
```

```
        (when (and conv-args
                   (and (not (equal sender-rep receiver-rep))
                      receiver-ID))
              ; conv-args = (date location)
              ; need to process each domain given in conv-args
              (dolist (arg-domain conv-args)
                    (let ((arg-attribute (first (send-message receiver-
dom-catalg :get-attributes-of-data-type arg-domain))))
              ; :get-attributes-of-data-type might give more than one
              ; attribute, we take the first one.
                      (setq conv-arguments (append conv-arguments (list
(list arg-domain arg-attribute)))))))))

; at this point:
; sender-rep = ((format format-plain) (unit usd) (scale scale-1))
; receiver-rep = ((format format-disc) (unit (get-unit-disc IN)) (scale
scale-1000))
; ID-arguments = (IN)
; conv-arguments = (date CF)

    ; now we build the sub-strategy
    (cond ; for sub-strategy building
     ((equal sender-rep receiver-rep) sub-strategy) ; returns NIL if
match
     (T
      (setq sender-rep (list 'sender-semantics sender-rep))
      (setq receiver-rep (list 'receiver-semantics receiver-rep))
      (if ID-arguments
          (setq ID-arguments (list 'ID-arguments ID-arguments)))
      (if conv-arguments
          (setq conv-arguments (list 'conversion-arguments conv-
arguments)))

      (cond ; on ID-arguments and conv-arguments
       ((and ID-arguments conv-arguments)
      (setq sub-strategy (list domain (list sender-rep receiver-rep ID-
arguments conv-arguments))))
       ((and ID-arguments (null conv-arguments))
      (setq sub-strategy (list domain (list sender-rep receiver-rep ID-
arguments))))
       ((and (null ID-arguments) conv-arguments)
      (setq sub-strategy (list domain (list sender-rep receiver-rep
conv-arguments))))
       ((and (null ID-arguments) (null conv-arguments))
      (setq sub-strategy (list domain (list sender-rep receiver-rep))))
       (T (format t "Error: unexpected event occured when examining ID-
arguments and conv-arguments")))))

  (cond ; strategy building
   (sub-strategy (setq strategy (append strategy (list sub-strategy))))
   ((null sub-strategy) (format t "~%No sub-strategy is required for the
domain ~A" domain))
   (T (format t "~%Error: unexpected value for sub-strategy")))))
  (lqp-print 'terse "~%Strategy = ~A" strategy)
  strategy
  ))

;;; STRATEGY readers and writers
```

```
;;; 'domain' and 'data-type' refers to the same concept; they should be
;;; taken as synonyms


;;; READER
(defun get-sub-strategy (strategy domain)
  (assoc domain strategy))

;;; READER
;;; Extract information from domain strategy: read slot
(defun get-domain-strategy-slot (domain-strategy slot)
  (second (assoc slot (second domain-strategy))))

;;; WRITER
;;; modify sender-semantics or receiver-semantics
(defun write-semantics (domain-strategy which? rep-component value)
(cond
 ((or (eq which? 'sender-semantics) (eq which? 'receiver-semantics))
  (rplacd (assoc rep-component (get-domain-strategy-slot domain-strategy
which?)) (list value)))
 (T (format T "You gave a wrong slot name: re-enter with sender-
semantics or receiver-semantics"))))

;;; WRITER
;;; modify ID-arguments or conversion-arguments
(defun add-arguments (domain-strategy which? value)
(cond
 ((or (eq which? 'ID-arguments) (eq which? 'conversion-arguments))
  (let (
      (old-value-in-slot (second (get-domain-strategy-slot domain-
strategy which?))))
  (setf (second (get-domain-strategy-slot domain-strategy which?))
(append old-value-in-slot (list value)))))
 (T (format T "You gave a wrong slot name: Re-enter with ID-arguments or
conversion-arguments"))))
```

***

```
;;; Query Parser Module

;;; PROCEDURE: parse a query's projection list.
;;; Inputs:
;;; 1) LQP object name: lqp-object-name
;;; 2) projection list, e.g. (co ni cf): projection-list
;;; 3) a list of domains (see parse-constraint for more details)
;;; 4) lqp-strategy
;;; Output: projection list + some attributes pushed in.

(defun parse-projection-list (data-type-catalogue projection-list
domain-list lqp-strategy)

  (let ((new-projection-list projection-list))
    ; initialize new-projection-list to projection-list
    (lqp-print 'terse "~%~%Begin parsing the query's projection
list...")
    ; treat one column at a time
    (dolist (attribute projection-list new-projection-list)
       (let*
```

```
        ((attributes-catalogue (get-object data-type-catalogue
'attributes-catalogue))
         (domain (get-object attributes-catalogue attribute))
         (domain-strategy (get-sub-strategy lqp-strategy domain))
         (ID-arguments (get-domain-strategy-slot domain-strategy 'ID-
arguments))
         (conv-arguments (mapcar #'(lambda (couple) (second couple))
(get-domain-strategy-slot domain-strategy 'conversion-arguments))))
       ; conversion-arguments is originally ((date CF) (name AD) ...)
       ; the mapcar makes it as (CF AD ...)

       ; push ID-arguments and conv-arguments into new-projection-list
       (when (member domain domain-list)
             (setf new-projection-list (pushnewlist ID-arguments new-
projection-list))
             (setf new-projection-list (pushnewlist conv-arguments new-
projection-list)))))
    (lqp-print 'terse "~%... query's projection list is parsed.~%New
projection list: ~A" new-projection-list)
    new-projection-list))


;;; PROCEDURE: examine a single predicate.
;;; Inputs: 4 arguments
;;; 1) lqp object name: lqp-object-name
;;; 2) a predicate: (operator attribute value)
;;; 3) domain-list (see domain-list in the function 'parse-constraint'
;;; 4) lqp-strategy: strategy built by 'build-lqp-data-conversion-
strategy
;;; Returns four values:
;;; 1) predicate converted if possible
;;;    (for PreConvertQuery)
;;; 2) predicate converted or else T if the predicate can't be converted
;;;    or can be converted but cannot be applied at the remote site
;;;    (for RemoteQuery)
;;; 3) list of columns to push into the list predicates-category-1
;;; 4) list of columns to push into the list predicates-category-2
;;; 5) list of columns to push into the query's projection list
;;; returns NIL if an unexpected event occurs.

(defun process-predicate (data-type-catalogue predicate domain-list
predicate-filter target lqp-strategy)

(let* ((attribute (second predicate))
       (predicate-value (third predicate))
       (attributes-catalogue (get-object data-type-catalogue
'attributes-catalogue))
       (domain (get-object attributes-catalogue attribute))
       (domain-strategy (get-sub-strategy lqp-strategy domain))
       (sender-semantics (get-domain-strategy-slot domain-strategy
'sender-semantics))
       (receiver-semantics (get-domain-strategy-slot domain-strategy
'receiver-semantics))
       (ID-arguments (get-domain-strategy-slot domain-strategy 'ID-
arguments))
       (conv-arguments (mapcar #'(lambda (element) (second element))
(get-domain-strategy-slot domain-strategy 'conversion-arguments)))
       (predicate-status (funcall predicate-filter predicate domain))
```

```
       (predicate-1 'NIL) ; predicate of first category
       (predicate-2 'NIL) ; predicate of second category
       (projection-list 'NIL)) ; for attributes to be pushed in column
list

(cond ; outer 'cond'
 ((null domain)
  (format t "~%~%Error: The attribute ~A is not known by ~A" attribute
data-type-catalogue))

 ; if the predicate's domain is not in the list, then skip it
 ; and just return values as if the predicate was a 3rd category one.
 ; or if sender-semantics and receiver-semantics are identical
 ((or (not (member domain domain-list)) (equal sender-semantics
receiver-semantics))
  (lqp-print 'verbose "~%~%The predicate ~A does not belong to the list
of domains indicated ==> treated as a 3rd category predicate" predicate)
  (values predicate predicate 'NIL 'NIL (list attribute))) ; exits the
routine

 ((not (or (equal predicate-status '1) (equal predicate-status '0)))
  (format t "~%~%Syntax error in ~A" predicate))
 ; if there is a syntax error, i.e. neither "1" nor "0", then stop.
 ; else:
 (T ; predicate-status is valid
  (cond ; both ID-arguments and conversion-arguments are NIL
   ((and (null ID-arguments) (null conv-arguments))
    ; <==> the predicate can be converted.
    ; the predicate is converted if the domain-strategy exists
    ; Recall that if domain-strategy exists, then there is mismatch
    (if domain-strategy ; there is a strategy for this domain
      (lqp-print 'verbose "~%~%The predicate ~A can be converted because
both ID-arguments and conversion-arguments lists are empty for the
domain ~A." predicate domain)
      (lqp-print 'terse "...does not require conversion."))

    ; push attribute in projection list:
    (setq projection-list (pushnewlist attribute projection-list))

    (when domain-strategy
        (let ((prepared-data-list (list (list (format 'nil "~A"
attribute)) (list predicate-value))))
          (data-list-convert prepared-data-list target (list attribute)
data-type-catalogue lqp-strategy 'no)
          (setq converted-value (car (second prepared-data-list)))
          (setq converted-predicate (list (first predicate) attribute
converted-value))
          (setq predicate converted-predicate)

          (when (equal predicate-status '0) ;  2nd category predicate
              (setq predicate-2 (pushnewlist attribute predicate-2))))))

   (T
    ; ID-arguments and/or conversion-arguments are not empty
    (lqp-print 'verbose "~%The predicate can't be converted because ID-
arguments and/or conversion-arguments aren't empty.")
    (setq projection-list (pushnewlist attribute projection-list))
    (setq projection-list (pushnewlist ID-arguments projection-list))
```

```
    (setq projection-list (pushnewlist conv-arguments projection-list))
    (setq predicate-1 (pushnewlist attribute predicate-1))))

  (values predicate ; for PreConvertQuery
        ; the T  means yes indeed, the predicate must be removed
        ; from RemoteQuery.
        (if (or predicate-1 predicate-2) 'T predicate)
        predicate-1
        predicate-2
        projection-list)))))

;;; parse constraint
;;; Inputs: 7 arguments
;;; 1) lqp object name: lqp-object-name
;;; 2) constraint to parse: constraint
;;; 3) list of domains to consider in the constraint, i.e. only the
predicates
;;;    corresponding to these domains should be considered. We made this
design
;;;    to be as flexible as possible (see data filter).
;;; 4) predicate filter function name
;;; 5) lqp strategy: lqp-strategy
;;; 6) 1st category predicates: predicates-category-1
;;; 7) 2nd category predicates: predicates-category-2
;;; 7) list of attributes to add to the projection list; projection-list
;;; Outputs: 5
;;; 1) constraint-1 = predicates that can't be converted and/or
;;;    can't be applied at the remote site are removed.
;;;    (part of PreConvertConstraint)
;;; 2) constraint-2 = no predicate is removed.
;;; 3) predicates-1
;;; 4) predicates-2
;;; 5) projections-list

(defun parse-constraint (data-type-catalogue constraint domain-list
predicate-filter target lqp-strategy predicates-category-1 predicates-
category-2 projections-list)

  (cond
   ((predicate-p constraint) ; if the query is a predicate
    (multiple-value-bind
     (constraint-1 constraint-2 predicates-1 predicates-2 projections-
to-add)
     (process-predicate data-type-catalogue constraint domain-list
predicate-filter target lqp-strategy)

     (setq predicates-category-1 (pushnewlist predicates-1 predicates-
category-1))
     (setq predicates-category-2 (pushnewlist predicates-2 predicates-
category-2))
     (setf projections-list (pushnewlist projections-to-add projections-
list))

     (values constraint-1 constraint-2 predicates-category-1 predicates-
category-2 projections-list)))

   (T ; the constraint is a conjunction of ORs and ANDs
    (let ((conjunction (first constraint)) ; AND or OR
```

```
        (left-tree (second constraint))
        (right-tree (third constraint)))
      (cond
       ((and (predicate-p left-tree) (predicate-p right-tree))
      ; the two branches of the tree are predicates
      (multiple-value-bind ; examine left tree.
       (l-constraint-1 l-constraint-2 l-predicates-1 l-predicates-2 l-
projections-to-add)
       (process-predicate data-type-catalogue left-tree domain-list
predicate-filter target lqp-strategy)

       (setq predicates-category-1 (pushnewlist l-predicates-1
predicates-category-1))
       (setq predicates-category-2 (pushnewlist l-predicates-2
predicates-category-2))
       (setf projections-list (pushnewlist l-projections-to-add
projections-list))

       (multiple-value-bind ;within the first multiple-value-bind, right
tree
        (r-constraint-1 r-constraint-2 r-predicates-1 r-predicates-2 r-
projections-to-add)
        (process-predicate data-type-catalogue right-tree domain-list
predicate-filter target lqp-strategy)

        (setq predicates-category-1 (pushnewlist r-predicates-1
predicates-category-1))
        (setq predicates-category-2 (pushnewlist r-predicates-2
predicates-category-2))
        (setf projections-list (pushnewlist r-projections-to-add
projections-list))
        (values
         (list conjunction l-constraint-1 r-constraint-1)
         (cond
          ((and (equal l-constraint-2 'T) (equal r-constraint-2 'T)) 'T)
          ((equal l-constraint-2 'T) r-constraint-2)
          ((equal r-constraint-2 'T) l-constraint-2)
          (T (list conjunction l-constraint-2 r-constraint-2)))
         predicates-category-1
         predicates-category-2
         projections-list))))

       ((and (not (predicate-p left-tree)) (predicate-p right-tree))
      ; left tree is a conjunction of OR and AND
      ; right tree is a predicate
      (multiple-value-bind ; examine left tree.
       (l-constraint-1 l-constraint-2 l-predicates-1 l-predicates-2 l-
projections-to-add)
       (parse-constraint data-type-catalogue left-tree domain-list
predicate-filter target lqp-strategy predicates-category-1 predicates-
category-2 projections-list)

       (setq predicates-category-1 (pushnewlist l-predicates-1
predicates-category-1))
       (setq predicates-category-2 (pushnewlist l-predicates-2
predicates-category-2))
       (setq projections-list (pushnewlist l-projections-to-add
projections-list))
```

```
       (multiple-value-bind ;within the first multiple-value-bind, right
tree
        (r-constraint-1 r-constraint-2 r-predicates-1 r-predicates-2 r-
projections-to-add)
        (process-predicate data-type-catalogue right-tree domain-list
predicate-filter target lqp-strategy)

        (setq predicates-category-1 (pushnewlist r-predicates-1
predicates-category-1))
        (setq predicates-category-2 (pushnewlist r-predicates-2
predicates-category-2))
        (setf projections-list (pushnewlist r-projections-to-add
projections-list))
        (values
         (list conjunction l-constraint-1 r-constraint-1)
         (cond
          ((and (equal l-constraint-2 'T) (equal r-constraint-2 'T)) 'T)
          ((equal l-constraint-2 'T) r-constraint-2)
          ((equal r-constraint-2 'T) l-constraint-2)
          (T (list conjunction l-constraint-2 r-constraint-2)))
         predicates-category-1
         predicates-category-2
         projections-list))))

       ((and (predicate-p left-tree) (not (predicate-p right-tree)))
      ; left tree is a predicate
      ; right tree is a conjunction of OR and AND
      (multiple-value-bind ; examine left tree.
       (l-constraint-1 l-constraint-2 l-predicates-1 l-predicates-2 l-
projections-to-add)
       (process-predicate data-type-catalogue left-tree domain-list
predicate-filter target lqp-strategy)

       (setq predicates-category-1 (pushnewlist l-predicates-1
predicates-category-1))
       (setq predicates-category-2 (pushnewlist l-predicates-2
predicates-category-2))
       (setq projections-list (pushnewlist l-projections-to-add
projections-list))

       (multiple-value-bind ;within the first multiple-value-bind, right
tree
        (r-constraint-1 r-constraint-2 r-predicates-1 r-predicates-2 r-
projections-to-add)
        (parse-constraint data-type-catalogue right-tree domain-list
predicate-filter target lqp-strategy predicates-category-1 predicates-
category-2 projections-list)

        (setq predicates-category-1 (pushnewlist r-predicates-1
predicates-category-1))
        (setq predicates-category-2 (pushnewlist r-predicates-2
predicates-category-2))
        (setq projections-list (pushnewlist r-projections-to-add
projections-list))
        (values
         (list conjunction l-constraint-1 r-constraint-1)
         (cond
```

```
          ((and (equal l-constraint-2 'T) (equal r-constraint-2 'T)) 'T)
          ((equal l-constraint-2 'T) r-constraint-2)
          ((equal r-constraint-2 'T) l-constraint-2)
          (T (list conjunction l-constraint-2 r-constraint-2)))
         predicates-category-1
         predicates-category-2
         projections-list))))

       ((and (not (predicate-p left-tree)) (not (predicate-p right-
tree)))
      ; left tree is a conjunction of OR and AND
      ; right tree is a conjunction of OR and AND
      (multiple-value-bind ; examine left tree.
       (l-constraint-1 l-constraint-2 l-predicates-1 l-predicates-2 l-
projections-to-add)
        (parse-constraint data-type-catalogue left-tree domain-list
predicate-filter target lqp-strategy predicates-category-1 predicates-
category-2 projections-list)

       (setq predicates-category-1 (pushnewlist l-predicates-1
predicates-category-1))
       (setq predicates-category-2 (pushnewlist l-predicates-2
predicates-category-2))
       (setq projections-list (pushnewlist l-projections-to-add
projections-list))

       (multiple-value-bind ;within the first multiple-value-bind, right
tree
        (r-constraint-1 r-constraint-2 r-predicates-1 r-predicates-2 r-
projections-to-add)
        (parse-constraint data-type-catalogue right-tree domain-list
predicate-filter target lqp-strategy predicates-category-1 predicates-
category-2 projections-list)

        (setq predicates-category-1 (pushnewlist r-predicates-1
predicates-category-1))
        (setq predicates-category-2 (pushnewlist r-predicates-2
predicates-category-2))
        (setq projections-list (pushnewlist r-projections-to-add
projections-list))
        (values
         (list conjunction l-constraint-1 r-constraint-1)
         (cond
          ((and (equal l-constraint-2 'T) (equal r-constraint-2 'T)) 'T)
          ((equal l-constraint-2 'T) r-constraint-2)
          ((equal r-constraint-2 'T) l-constraint-2)
          (T (list conjunction l-constraint-2 r-constraint-2)))
         predicates-category-1
         predicates-category-2
         projections-list))))

       (T (lqp-print 'terse "~%Error: unexpected event occured")))))))

;; test whether an atom is an operator
;; returns T or NIL
(defun operator-p (operator)
(case operator
      ('> 'T)
```

```
      ('< 'T)
      ('= 'T)
      (otherwise 'NIL)))

;; test whether the list at hand is a predicate
(defun predicate-p (constraint)
(operator-p (first constraint)))

;;; Query Parser: final function
;;; Input: 4
;;; 1) lqp object name: data-type-catalogue
;;; 2) query to parse: query
;;; 3) list of domains to consider
;;; 4) predicate filter function name
;;; 5) lqp data conversion strategy
;;; Output: 4
;;; 1) PreConvertQuery
;;; 2) RemoteQuery
;;; 3) predicates-category-1
;;; 4) predicates-category-2

(defun parse-query (data-type-catalogue query domain-list predicate-
filter target lqp-strategy)

(cond ; check if domain-list is nil
 ((null domain-list) (values query query 'nil 'nil))
 ((not (equal '3 (length query))) (lqp-print 'terse "~%Error: query's
syntax isn't correct."))

 (T ; domain-list is not empty
  (let ; outer let
      ; parse out the projection list
      ((projections-list (parse-projection-list data-type-catalogue
(second query) domain-list lqp-strategy))
       (constraint (third query)))

    (lqp-print 'normal "~%Projection list after parsing the query's
projection list= ~A" projections-list)

    (lqp-print 'terse "~%~%Begin parsing constraint...")
    (multiple-value-bind
     (PreConvertQuery RemoteQuery predicates-category-1 predicates-
category-2 more-projections)
     (parse-constraint data-type-catalogue constraint domain-list
predicate-filter target lqp-strategy 'nil 'nil 'nil)

     (setq projections-list (pushnewlist more-projections projections-
list))

     (cond ; test if RemoteQuery is nil
      ((null PreConvertQuery) (lqp-print 'terse "~%Error in parse-query:
the constraint parser was not successful"))
      ((null RemoteQuery) (lqp-print 'terse "~%Error in parse-query:
RemoteQuery hasn't any constraint!"))
      (T
       ; (first query) is the table name
       (setq PreConvertQuery (list (first query) projections-list
PreConvertQuery))
```

```
      (setq RemoteQuery (list (first query) projections-list
RemoteQuery))

      (values PreConvertQuery RemoteQuery predicates-category-1
predicates-category-2))))))))

                                    ***
;;; Data+MetaData  Structure  Building

;;; 'domain' and 'data-type' refers to the same concept.

;;; Read a slot
(defun read-data-slot (data-structure slot-name)
(cdr (assoc slot-name (second data-structure))))

;;; Write into a slot
(defun write-data-slot (data-structure slot-name stuff-to-write)
(cond
 ((null stuff-to-write)
  (format t "~%Warning: the data writer is asked to write NIL in a data
structure slot! Are you sure? The writing is done anyway.")
  (setf (cdr (assoc slot-name (second data-structure))) stuff-to-write))
 ((listp stuff-to-write)
  (setf (cdr (assoc slot-name (second data-structure))) stuff-to-write))
 ((atom stuff-to-write)
  (setf (cdr (assoc slot-name (second data-structure))) (list stuff-to-
write)))))

;;; Build a data structure
(defun build-data-structure (domain-name values components-list)
; a member of components-list is under the form (component-name
component-ID)
; e.g., (format FORMAT-DISC)

  (let ((data-structure (list (append (list 'value) values)))
      (comp-list (copy-alist components-list)))
    (cond
     ((null comp-list) (list domain-name data-structure))
     (T
      (dolist (component comp-list data-structure)
       (setq data-structure (append data-structure (list component))))
      (setq data-structure (list domain-name data-structure))))))

;;; Description: the procedure reads values in a data row pertaining to
;;; given attributes. It also read the representation information in the
;;; data conversion strategy for the domain of that attribute. Finally
;;; it builds the data structure as shown above.
;;; Inputs:
;;; 1) data-row
;;; e.g., ((company-name "HONDA") (net-sales "200") ...)
;;; 2) list of attributes: attributes-list
;;; e.g., (NS NI)
;;; 3) the domain of the attributes
;;; 4) lqp data conversion strategy
;;; Output: 1
;;; 1) data+metadata row
```

```
(defun build-data-row (initial-data-row target attributes-list domain
strategy)

(let* ((domain-strategy (get-sub-strategy strategy domain))
       (values (get-values-in-row initial-data-row attributes-list))
       (which-semantics? (if (equal target 'receiver-semantics)
                             'sender-semantics
                           'receiver-semantics))
       (semantics (get-domain-strategy-slot domain-strategy which-
semantics?))
       ; ((FORMAT FIGURE-DISC) (UNIT (GET-UNIT-SUZE CURRENCY EX-MARKET))
       ; (SCALE (GET-SCALE-SUZE SCALE)))
       (conversion-arguments (get-domain-strategy-slot domain-strategy
'conversion-arguments))
       ; conversion-arguments = ((DATE REPORT-DATE) (NAME EX-MARKET))
       (deduced-semantics 'nil)
       (final-data-row 'nil))

  ; process each component of the representation to build the data
structure
  (if (not semantics) (return)
    (dolist (component semantics deduced-semantics)
       (let ((aspect (first component))
             (ID? (second component)))
       (cond
        ((null ID?) (lqp-print 'verbose "~%Build-data-row: ~A is
irrelevant" (first component)))
        ((listp ID?) ; the ID is a function call
         (setq component (deduce-component-ID component initial-data-
row))
         ;(if (not component) ; deduce-component-ID returned NIL
         ;    (setq deduced-semantics (append deduced-semantics (list
(list aspect 'nil)))))
         (setq deduced-semantics (append deduced-semantics (list
component))))
        ((atom ID?)
         (setq deduced-semantics (append deduced-semantics (list
component))))
        (T (format t "~%Error: build-data-row doesn't understand ~A"
component))))))

  ; deduced-semantics = ((FORMAT FIGURE-DISC) (UNIT NJPY) (SCALE SCALE-
1000))

  (cond
   ; for the two first conditions, the function stops and returns NIL
   ((null values) (format t "~%build-data-row stopped: values is
NIL~%"))
   ((not (listp values)) (format t "~%Error in build-data-row: values is
not a list!"))
   (T
    (setq final-data-row (list (build-data-structure domain values
deduced-semantics)))
    ; final-data-row:
    ; (FIGURE ((VALUE "200" "300") (FORMAT FIGURE-DISC) (UNIT NJPY)
    ; (SCALE SCALE-1000)))

    ; now we have to treat the conversion arguments, one after the other
```

```
    (dolist (conv-arg conversion-arguments final-data-row)
          ; conv-arg = (date REPORT-DATE)

      (let* ((domain-name (first conv-arg))
             ; date
             (attribute-strategy (get-sub-strategy strategy domain-
name))
             (attribute (second conv-arg))
             (attribute-value (get-values-in-row initial-data-row (list
attribute)))
             (attribute-semantics
            (if attribute-strategy
                (get-domain-strategy-slot attribute-strategy which-
semantics?)
             'nil))
             ; ((FORMAT DATE-YYYYMMDD))
             (attribute-structure (build-data-structure domain-name
attribute-value attribute-semantics)))
        (setq final-data-row (append final-data-row (list attribute-
structure)))))))))


;;; Enhance a whole data list
;;; Input: 4
;;; 1) a data-list
;;; e.g., (("co" "ni" "ns" "cf") ("honda" "200" "300" "19882131") ...)
;;; 2) list of attributes to convert
;;; 3) domain to which the attributes belong
;;; 4) lqp data conversion strategy
;;; Outputs: 1
;;; 1) enhanced data-list

(defun prepare-data-list (data-list target attributes-list domain
lqp-strategy)
(let* ((columns-list (first data-list))
       (row-list (cdr data-list))
       (data-list-tmp 'nil)
       (new-data-list 'nil))

; build a tmp data list of the form:
; ((CO "honda") (NET-SALES "200"))
(dolist (row row-list data-list-tmp)
   (setq data-list-tmp (append
                 data-list-tmp
                 (list (mapcar #'(lambda (elt1 elt2) (list (read-from-
string elt1) elt2)) columns-list row)))))

; prepare data row for each row of data list
(dolist (row data-list-tmp new-data-list)
   (let ((data-row (build-data-row row target attributes-list domain
lqp-strategy)))
     (if (not data-row) ; data-row is NIL
       (return (format t "~%Prepare-data-list stopped: build-data-row
returned NIL~%"))
       (setq new-data-list (append new-data-list (list data-row)))))))))

;;; get list of values in a rwo given a list of attributes
(defun get-values-in-row (row attributes-list)
```

```
(let ((result 'nil))
  (dolist (attribute attributes-list result)
        (setf result (pushnewlist (second (assoc attribute row))
result)))
  (if (not (equal (length result) (length attributes-list)))
      (format t "~%get-values-in-row returned NIL: some attributes of ~A
weren't found in row provided:~%~S~%" attributes-list row)
    result)))
```

***

;;; Deduction of dynamic representation aspects' identities deduction from data

```
;;; Infer the representation IDs if necessary (general-purpose routine)
;;; inputs: 2
;;; 1) component
;;; e.g., (UNIT (GET-UNIT-SUZE CURRENCY EX-MARKET))
;;; 2) row
;;; e.g., ((co "honda") (ni "200") ...)
;;; output: 1
;;; Deduced representation ID: (unit JPY)

(defun deduce-component-ID (component row)
(let* ((component-name (first component))
       (ID-to-deduce (second component))
       (function-call (first ID-to-deduce))
       (arguments (cdr ID-to-deduce))) ; e.g., (CURRENCY EX-MARKET)

(cond
 ((not (listp ID-to-deduce)) component)
 ((null ID-to-deduce) (format t "~%Error: no component ID exists in ~A!"
component))
 (T ; the component is a function call
  (let* ((values (get-values-in-row row arguments))
       (ID-deduced (if values
                     (eval `(,function-call ,@values))
                     values)))
    (if values
      (list component-name ID-deduced) ; return ID-deduced
      (progn
      (format t "~%deduce-component-ID returned NIL: VALUES was bound to
NIL")
      (list component-name 'nil)))))))

;;; Finsbury Dataline: infer the scale
(defun get-scale-dataln (scale)
(cond
 ((equal scale '"000s") 'SCALE-1000)
 ((equal scale '"m") 'SCALE-1000)
 (T (format t "~%Error: the scale ~A is not known by get-scale-suze yet"
scale))))

;;; Finsbury Dataline: infer the currency
(defun get-unit-dataln (currency)
(cond
 ((equal currency "JAPAN") 'NJPY)
 ((equal currency "UNITED KINGDOM") 'NGBP)
 ((equal currency "FRANCE") 'NFRF)
```

```
 (T
  ((format t "~%Error in get-unit-dataln: ~A not recognized"
currency)))))

;;; I.P. Sharp Disclosure: infer the currency
(defun get-unit-disc (country)
(cond
 ((equal country "JAPAN") 'NJPY)
 ((equal country "UNITED KINGDOM") 'NGBP)
 ((equal country "FRANCE") 'NFRF)
 (T
  (if (equal (length country) 2) ; e.g., DE
      'NUSD
    (format t "~%Error in get-unit-disc: ~A not recognized" country)))))

                                    ***
;;; Data Convert Module

;;; Inputs:
;;; 1) Data list such as:
;;; (("company-name" "net-sales" "net-income" "report-date" "currency"
;;;   "ex-market" "scale")
;;;  ("honda" "500,000.4" "250,300.2" "19,881,231" "Yen" "New York"
"000s")
;;;  ("Jaguar" "1,000.67" "500.4" "19,870,228" "Pounds" "London" "000s")
;;;  ("Ford Motor Co" "20,000" "13,000" "19,890,318" "US Dollars"
;;;   "New York" "m"))
;;; 2) list of attributes to convert, e.g. (NI)
;;; 3) Bridge data type catalogue
;;; 4) Bridge data conversion strategy
;;; outputs: same data list with data converted.

(defun data-list-convert (data-list target attributes-list data-type-
catalogue lqp-strategy update-strategy?)

(let ((attributes-catalogue (get-object data-type-catalogue 'attributes-
catalogue))
      (conversion-tasks 'nil)) ; outer let

; first thing to do is to create a list of conversion tasks:
; if attributes-list = (NS NI CF YEAR)
; ==> ((figure NS NI) (date CF) (year CF))
; the following let does that...
(let ((data-types-list (get-object data-type-catalogue 'data-types)))

  ; for each data type, get its attributes in attributes-list
  (dolist (data-type data-types-list)
     (let ((attributes-of-data-type (list data-type)))
       (dolist (attribute attributes-list attributes-of-data-type)
        (let* ((attr (if (stringp attribute)
                         (read-from-string attribute)
                       attribute))
               (data-type-of-attribute
                (get-object
                 attributes-catalogue
                 attr)))
          (if (equal data-type-of-attribute data-type)
```

```
            (setq attributes-of-data-type (append attributes-of-data-
type (list attr))))))
        (if (second attributes-of-data-type) ; there is at least 1
attribute
          (setq conversion-tasks (append conversion-tasks (list
attributes-of-data-type)))))))

; conversion-tasks:
; ((FIGURE NET-SALES NET-INCOME) (DATE REPORT-DATE) (YEAR YEAR))
; for each conversion task, we prepare the data list and convert it;

(lqp-print 'verbose "~%List of conversion tasks by data types: ~A"
conversion-tasks)
(dolist (conv-task conversion-tasks)
   (lqp-print 'verbose "~%Conversion task for the data type ~A:~A"
(first conv-task) conv-task)
   (let* ((init-semantics (if (equal target 'sender-semantics)
                          'receiver-semantics
                        'sender-semantics))
          (data-type (first conv-task))
          (data-type-strategy (get-sub-strategy lqp-strategy data-type))

          ;enhanced-data-list is NIL if the domain does not require
conversions
          (enhanced-data-list
           (if data-type-strategy
               (prepare-data-list data-list target (cdr conv-task) data-
type lqp-strategy)))
          (target-representation (get-domain-strategy-slot data-type-
strategy target))
          (init-representation (get-domain-strategy-slot data-type-
strategy init-semantics))
          (conversion-plan (mapcar #'(lambda (init target)
                                       (list (first init) (cdr init) (cdr
target)))
                                   init-representation
                                   target-representation)))

     ; conversion-plan looks like:
     ; ((FORMAT (FIGURE-DISC) (FIGURE-PLAIN)) (UNIT (JPY) (USD))
     ; (SCALE (SCALE-1000) (SCALE-1)))
     ; convert for each component of the data type
     (when (and data-type-strategy enhanced-data-list)
         (dolist (conversion conversion-plan)
            (let ((rep-component (first conversion))
                (init-ID (first (second conversion)))
                (target-ID (first (third conversion))))

              (if (and (not (equal init-ID target-ID)) (and init-ID
target-ID))
                  ; conversion is necessary
                  ; note that it target-ID is NIL, then no conversion
                  ; is necessary
                  ; ditto if init-ID is NIL.
                  (progn
                    (send-message target-ID :convert enhanced-data-list)
                    ; update the lqp data type catalogue
                    (if (equal update-strategy? 'yes)
```

```
                    (progn
                      (lqp-print 'verbose "~%strategy was updated.")
                      (write-semantics data-type-strategy init-semantics
rep-component target-ID))))

                  (lqp-print 'terse "~%No conversion is necessary for ~A"
conversion))))
          ; update the data list with the new values
          (update-reg-data-list data-list (build-reg-data-list-from-
enhanced-one enhanced-data-list (cdr conv-task) data-type)))))
data-list))

;;; Representation conversion of the data type MONEY-AMOUNT

;;; Conversion of the aspect FORMAT

;;; ============= Special Lisp dependent processing ====================
;;; From 3.828328398329383E12 to 3828328398329.38298298
;;; 3.8398E3 is sometimes sent back by Lisp after calculations and when
;;; using 'read-from-string'.  This format is not appropriate in the
;;; conversion to DISC or others, hence we need to convert this notation
;;; back to the PLAIN representation.
;;; Note: 'read-from-string' truncates numbers:
;;; >(read-from-string '"247474747284242.83928283")
;;; 2.474747472842428E14
;;; 24

;; test whether the value is of the form
;; returns NIl if not of the type or an integer indicating what
;; 'Val' and 'n' are.

(defun Val-E-n-p (value)
(let* ((string-value (if (stringp value)
                         value
                         (format nil "~A" value)))
       (str-length (length string-value))
       (exponent (make-string '0))
       (Val (make-string -0))
       (flag 'NIL))
  (dotimes (index str-length)
     (let ((character (char string-value index)))
       (cond
       ((not flag) ; 'E' hasn't been encountered yet
        (if (equal character '#\E) ; this is it.
            (setq flag 'T)
          (setq Val (concatenate 'string Val (list character)))))
       (flag ; E was encountered
        (setq exponent (concatenate 'string exponent (list
character)))))))
  (if flag
      (values Val exponent)
    'NIL)))

;; Function that does the actual conversion
(defun convert-Val-E-n-to-PLAIN (value)
(multiple-value-bind
 (mantisse exp)
 (Val-E-n-p (if (stringp value) value (format nil "~A" value)))
```

```
 ; body
(if (not mantisse) ; the value is not of the type ValEn
    (let* ((val (format nil "~A" value))
           (long (length val)))
      (if (and (equal (char val (- long 2)) '#\.)
               (equal (char val (1- long)) '#\0))
        (let ((new-string (make-string '0)))
          (dotimes (index (- long 2) new-string)
               (let ((character (char val index)))
              (setq new-string (concatenate 'string new-string (list
character))))))
      val))
  ; else
  (let ((str-length (length mantisse))
      (exponent (read-from-string exp))
      (new-string (make-string '0))
      (flag 'NIL) ; to notify when the "." is encountered
      (counter '0))
    (dotimes (index str-length new-string)
       (let ((character (char mantisse index)))
       (cond
        ((not flag) ; "." was not met yet
         (if (equal character '#\.)
             (setq flag 'T)
           (setq new-string (concatenate 'string new-string (list
character)))))
        (flag ; "." was already encountered
         (setq counter (1+ counter)) ; increment 'counter'
         (if (and (equal counter exponent) (< index (1- str-length)))
             (setq new-string (concatenate 'string new-string (list
character '#\.)))

           (setq new-string (concatenate 'string new-string (list
character)))))))))
    (if (< counter exponent)
      (dotimes (index (- exponent counter) new-string)
           (setq new-string (concatenate 'string new-string (list
'#\0))))
      new-string)))))

;; Conversion routine from DISC to PLAIN
(defun convert-DISC-to-PLAIN (value)
; value is a string
; the filtering consists of removing all the ","
(let ((new-value (make-string '0))
      (max-index (length value)))
  (dotimes (index max-index new-value)
     (let ((character (char value index)))
       (when (not (equal character '#\,))
           (setq new-value (concatenate 'string new-value (list
character)))))))


;; Conversion routine From PLAIN to DISC
(defun convert-PLAIN-to-DISC (value)
; value is 200000.54
; we split into 200000 and .54
```

```
(let ((left-part (make-string '0))
      (right-part (make-string '0))
      (max-index (length value))
      (flag 'NIL)) ; to indicate when the "." was found
  (dotimes (index max-index)
     (let ((character (char value index)))
       (cond
      ((not flag) ; "." was not encountered yet
       (if (equal character '#\.) ; if this is it
           (progn (setf flag 'T)
                (setf right-part (concatenate 'string right-part (list
character))))
         ; else, complete left-part
         (setf left-part (concatenate 'string left-part (list
character)))))
      (flag ; if "." was already encountered
       (setf right-part (concatenate 'string right-part (list
character)))))))

  ; we examine the left-part
  ; "200000"
  (let ((max (length left-part))
      (new-value (make-string '0))
      (counter '1))
    (dotimes (index max new-value)
       (let ((character (char left-part (- (1- max) index))))
       (if (equal counter '3)
           (progn
             (if (< index (1- max))
               (setf new-value (concatenate 'string new-value (list
character '#\,)))
             (setf new-value (concatenate 'string new-value (list
character))))

               (setf counter '1))
         (progn (setf new-value (concatenate 'string new-value (list
character)))
              (setf counter (1+ counter))))))
  (concatenate 'string (reverse new-value) right-part))))


;; Conversion routine from DATALN to PLAIN
(defun convert-DATALN-to-PLAIN (value)
  (let ((last (char value (1- (length value)))))
    (if (equal last '#\.)
      (string-trim (list last) value)
      value)))


;; Conversion routine from PLAIN to DATALN
(defun convert-PLAIN-to-DATALN (value)
(let ((max (length value))
      (flag 'NIL)) ; to indicate that a "." was encountered
  (dotimes (index max)
     (let ((character (char value index)))
       (if (equal character '#\.)
         (progn (setf flag 'T) (return)))))
  (if flag
```

```
      value
    (concatenate 'string value (list '#\.)))))


;;; Function: convert a value from init-ID to target-ID
;;; reference ID: PLAIN
(defun convert-figure-format (init-ID target-ID value)
  (case init-ID
   (FIGURE-PLAIN
    (case target-ID
     (FIGURE-PLAIN value)
     (FIGURE-DISC (convert-PLAIN-to-DISC value))
     (FIGURE-DATALN (convert-PLAIN-to-DATALN value))
     (otherwise (format t "~%Error: ~A is not understood!" target-ID))))
   (FIGURE-DISC
    (case target-ID
     (FIGURE-PLAIN (convert-DISC-to-PLAIN value))
     (FIGURE-DISC value)
     (FIGURE-DATALN (convert-PLAIN-to-DATALN (convert-DISC-to-PLAIN
value)))
     (otherwise (format t "~%Error: ~A is not understood!" target-ID))))
   (FIGURE-DATALN
    (case target-ID
     (FIGURE-PLAIN (convert-DATALN-to-PLAIN value))
     (FIGURE-DISC (convert-PLAIN-to-DISC (convert-DATALN-to-PLAIN
value)))
     (FIGURE-DATALN value)
     (otherwise (format t "~%Error: ~A is not understood!" target-ID))))
   (otherwise (format t "~%Error: ~A is not understood!" init-ID))))

;;; KOREL object methods:
;;; :data-list-convert-into-FIGURE-PLAIN
;;; :data-list-convert-into-FIGURE-DISC
;;; :data-list-convert-into-FIGURE-DATALN
;;; Input: data+metadata
(defun data-list-convert-into-FIGURE-PLAIN (data-list)
  (let ((object-name (get-current-object)))
    (data-list-convert-figure-format object-name data-list)))

(defun data-list-convert-into-FIGURE-DISC (data-list)
  (let ((object-name (get-current-object)))
    (data-list-convert-figure-format object-name data-list)))

(defun data-list-convert-into-FIGURE-DATALN (data-list)
  (let ((object-name (get-current-object)))
    (data-list-convert-figure-format object-name data-list)))


;; main function: data-list-convert-figure-format
(defun data-list-convert-figure-format (target-ID datalist)
  (let* ((domain (get-object target-ID 'data-type))
         (rep-component (get-object target-ID 'component))
         (new-values 'nil))
    (dolist (row datalist)
          (let* ((data-str (assoc domain row))
                 (init-ID (first (read-data-slot data-str rep-component)))
                 (values (read-data-slot data-str 'value)))
            (when (not (equal target-ID init-ID))
```

```
                (setq new-values (mapcar #'(lambda (value) (convert-
figure-format init-ID target-ID value)) values))
                (write-data-slot data-str 'value new-values)
                (write-data-slot data-str rep-component target-ID))))
    datalist))

;;; Conversion of the aspect UNIT (MONEY-AMOUNT)

;;; Series of list surgery function to prepare the query to send to the
;;; currency LQP.
;;; When receiving the data-list, the object updates
;;; the figure-unit-catalogue
;;; inputs:
;;; 1) row (a single row is examined since we assume that all rows have
;;; the same data semantics).
;;; 2) the object name (e.g., USD, etc...)

(defun update-figure-unit-catalogue (row curr-object)

(let ((catalogue-name (get-object curr-object 'data-type-catalogue))
      ; only the date format interests us
      ; obtain (DATE-YYYYMMDD)
      (date-format (first (read-data-slot (assoc 'date row) 'format))))
  (update-object catalogue-name 'date 'format date-format)))

;;; Build the following list:
;;; ((code-1.1 code-1.2 date-1)
;;;  (code-2.1 code-2.2 date-2)
;;;  ...
;;;  (code-n.1 code-n.2 date-n))
;;; Input:
;;; 1) data-list
;;; 2) target-currency, e.g., USD (which is the object name)
;;; Output: above list

(defun build-currency-requests-list (data-list curr-object)

; target-currency is USD, hence an N must be prefixed to obtain NUSD
(let ((target-currency (read-from-string (format nil "N~A" curr-
object))))

(mapcar #'(lambda (data-row)
          (let ((init-currency (first (read-data-slot (assoc 'figure
data-row) 'unit)))
                (date-value (first (read-data-slot (assoc 'date data-row)
'value))))
            (list init-currency target-currency date-value))) data-
list)))

;;; 1. Transform each item of the list built by build-currency-requests-
list
;;; the following manner:
;;; (same-code same-code date) becomes (1 1 date)
;;; (NUSD code date) becomes (1 code date)
;;; (code NUSD date) becomes (code 1 date)
;;; 2. remove from the list any item which has a "1"
;;; (for query to be sent to the currency Bridge)
```

```
;;; 3. keep track of the position in the initial list of the items that
were
;;; not removed from step #2.

;;; input: list of (code-1 code-2 date)
;;; output:3
;;; 1) list of (code-1 code-2 date) with possibly code-1 == 1 and/or
;;; code-2 == 1.
;;; (list-for-divisions)
;;; this list will serve for figuring out, in the case of (1 code date)
;;; or (code 1 date):
;;; (1 code date) ==> (1/code date)
;;; (code 1 date) ==> (code date)
;;; 2) list of (code-1 code-2 date) or (code-1 date)
;;; where code-1 != code-2 != USD
;;; (list-for-query)

;;; 3) list of positions in the initial list of the item of the second
output.

(defun filter-currency-requests-list (requests-list)
(let* ((list-for-divisions (mapcar #'(lambda (request)
                           (let ((init (first request))
                               (target (second request))
                               (date (third request)))
                             (cond
                              ((equal init target)
                               (list '1 '1 date))
                              (T
                             (list (if (equal init 'NUSD)
                                       '1
                                     init)
                                   (if (equal target 'NUSD)
                                       '1
                                     target)
                                   date)))))
                         requests-list))

       (tmp-list '())
       (list-for-query (dolist (request list-for-divisions tmp-list)
                         (let ((init (first request))
                             (target (second request))
                             (date (third request)))
                           (cond
                            ((equal init '1)
                             (if (not (equal target '1))
                               (setf tmp-list (append tmp-list (list
(list target date))))))
                            (T
                             (if (not (equal target '1))
                               (setf tmp-list (append tmp-list (list
(list init target date))))
                             (setf tmp-list (append tmp-list (list
(list init date)))))))))
      (tmp-list '())
      (position '0)
      (list-to-keep-track (dolist (request list-for-divisions tmp-list)
                          (let ((init (first request))
```

```
                          (target (second request)))
                        (if (not (and (equal init '1) (equal target '1)))
                          (setf tmp-list (append tmp-list (list
position))))
                        (setf position (1+ position))))))

  (values list-for-divisions list-for-query list-to-keep-track)))

;;; Build the query from list-for-query
;;; Input: list of (code-1 code-2 date) or (code-1 date)
(defun build-currency-query (requests-list)
(let ((total-constraint 'NIL))
  (dolist (request requests-list total-constraint)
     ; request = (code-1 code-2 date) or (code date)
     ; we have to form (and (or (= code "code-1") (= code "code-2"))
     ;                      (= date date))
     ; or (and (= code "code") (= date date))
     (cond
      ((equal (length request) '3) ; of the kind (code-1 code-2 date)
       (let* ((code-1 (list '= 'code (format nil "~A" (first request))))
            (code-2 (list '= 'code (format nil "~A" (second request))))
            (or-codes (list 'or code-1 code-2))
            (date (list '= 'date (third request)))
            (constraint (list 'and or-codes date)))
       (setf total-constraint (add-date/codes-predicate total-constraint
constraint))))
      ((equal (length request) '2) ; of the kind (code-1 date)
       (let* ((code-1 (list '= 'code (format nil "~A" (first request))))
            (date (list '= 'date (second request)))
            (constraint (list 'and code-1 date)))
       (setf total-constraint (add-date/codes-predicate total-constraint
constraint))))
      (T (format t "~%Error in build-currency-query: the request ~A is
not understood~%" request))))

  (list 'currency (list 'exchange-rate) total-constraint)))

;;; Add a predicate to an already existing constraint
(defun add-date/codes-predicate (constraint predicate)
; the predicate is (and (or (= code "code-1") (= code "code-2"))
;                       (= date "date"))
(let ((operator (first constraint))
      (last-pred (third constraint)))
  (cond
   ((null operator) predicate)
   ((not (equal operator 'or)) ; the constraint is a single date/codes
pred.
    (list 'or constraint predicate))
   ((equal operator 'or) ; the constraint is already a conjunction
    (list 'or (second constraint) (add-date/codes-predicate
                          (third constraint)
                          predicate)))
   (T (format t "~%Error in add-date/codes-predicate: operator ~A is not
understood!~%" operator)))))

;;; Function which reads the output from the Currency database
;;; Recall: filter-currency-requests outputs:
;;; ((NJPY NFRF "19,881,231") (NGBP NFRF "19,870,228")
```

```
;;;  (1 NFRF "19,890,318"))
;;; ((NJPY NFRF "19,881,231") (NGBP NFRF "19,870,228") (NFRF
"19,890,318"))
;;; (0 1 2)
;;; Inputs:
;;; 1) Initial list returned by filter-currency-requests
;;; 2) positions list returned by filter-currency-requests
;;; 3) list of exchange rates obtained from the database
;;; Output:
;;; 1) list of exchange rates that will be applied onto the data row
;;;    directly.
(defun map-exchange-rates-to-requests-list
  (init-requests
   positions-list
   exchange-rates-list)
(let ((max (length init-requests))
      (new-ex-rates-list 'nil))
  (dotimes (index max new-ex-rates-list)
     (let* ((row (nth index init-requests))
          (code-1 (first row))
          (code-2 (second row)))
       ; row = (code-1 code-2 date)
       (cond
      ((and (equal code-1 '1) (equal code-2 '1)) ; row = (1 1 date)
       (setq new-ex-rates-list (append new-ex-rates-list (list '"1"))))
      ((and (not (equal code-1 '1)) (equal code-2 '1)) ; row = (code 1
date)
       (setq exchange-rate (first (pop exchange-rates-list)))
       (setq new-ex-rates-list (append new-ex-rates-list (list exchange-
rate))))
      ((and (equal code-1 '1) (not (equal code-2 '1))) ; row = (1 code
date)
       (setq exchange-rate (first (pop exchange-rates-list)))
       (setq exchange-rate
             (format nil "~A"
                   (coerce
                  (multiple-value-bind
                   (expression)
                   (read-from-string
                    (format nil "~A" (list '/ '"1" exchange-rate)))
                   (eval expression))
                  'long-float)))
       (setq new-ex-rates-list (append new-ex-rates-list (list exchange-
rate))))
      ; row = (code-1 code-2 date)
      ((and (not (equal code-1 '1)) (not (equal code-2 '1)))
       (setq exchange-rate-1 (first (pop exchange-rates-list)))
       (setq exchange-rate-2 (first (pop exchange-rates-list)))
       (setq exchange-rate
             (format nil "~A"
                   (coerce
                  (multiple-value-bind
                   (expression)
                   (read-from-string
                    (format nil "~A" (list '/ exchange-rate-1 exchange-
rate-2)))
                   (eval expression))
                  'long-float)))
```

```
      (setq new-ex-rates-list (append new-ex-rates-list (list exchange-
rate))))
      (T (format t "~%Error in map-exchange-rates-to-requests-list: ~A
is not understood!" row)
         (return)))))))

;;; KOREL methods
(defun data-list-convert-into-USD (data-list)
(data-list-convert-figure-unit 'USD data-list))

(defun data-list-convert-into-JPY (data-list)
(data-list-convert-figure-unit 'JPY data-list))

(defun data-list-convert-into-FRF (data-list)
(data-list-convert-figure-unit 'FRF data-list))

(defun data-list-convert-into-GBP (data-list)
(data-list-convert-figure-unit 'GBP data-list))


;;; Conversion routine data-list-convert-figure-unit
(defun data-list-convert-figure-unit (target-ID data-list)
  (let* ((domain (get-object target-ID 'data-type))
       (rep-component (get-object target-ID 'component))
       (new-values 'T)
       (figure-format (first (read-data-slot
                        (assoc domain (first data-list))
                        'format)))
       (date-format (first (read-data-slot
                        (assoc 'date (first data-list))
                        'format)))
       (plain-flag 'T) ; to indicate whether the format is FIGURE-PLAIN
       (requests-list (build-currency-requests-list data-list target-
ID))
       (currency-db-LQP (get-object target-ID 'database))
       (figure-unit-catalogue (get-object target-ID 'data-type-
catalogue)))

    ; update the data type catalogue of the object
    (update-object figure-unit-catalogue 'date 'format date-format)

    (multiple-value-bind
     (initial-requests filtered-requests positions-list)
     (filter-currency-requests-list requests-list)
     (let* ((currency-query (build-currency-query filtered-requests))
          (exchange-rates-list-first
           (cdr (send-message currency-db-LQP ; take cdr to take out col
list
                     :get-data
                     figure-unit-catalogue
                     currency-query)))
          (exchange-rates-list (map-exchange-rates-to-requests-list
                          initial-requests
                          positions-list
                          exchange-rates-list-first))

          ; just to check if we have the same number of row and ex.
rates
```

```
          (row-nb (length data-list))
          (ex-rates-nb (length exchange-rates-list)))

       (if (not (equal row-nb ex-rates-nb))
         (progn
           (format t "~%Error in data-list-convert-figure-unit: the
number of exchange rates provided does not match the number of rows to
be converted.")
           (return)))

       ; if the format is not the reference format FIGURE-PLAIN, then
convert
       ; the data list into FIGURE-PLAIN

       (if (not (equal figure-format 'FIGURE-PLAIN))
         (progn (format t "~%Format conversion is required from ~A to
~A:" figure-format 'figure-PLAIN)
                (setq plain-flag 'nil)
                (data-list-convert-figure-format 'FIGURE-PLAIN data-
list)))

       ; multiply the records by the exchange-rates
       (multiply-data-list-by-ex-rates data-list exchange-rates-list
domain)

       ; update the semantics of the data-list: the unit becomes Target-
ID
       (dolist (row data-list)
             (let* ((data-str (assoc domain row)))
             (write-data-slot data-str rep-component target-ID)))

       ; if it was not PLAIN originally, convert back into the original
format
       (if (not plain-flag)
         (progn (format t "~%Converting the format back to ~A" figure-
format)
                (data-list-convert-figure-format figure-format data-
list)))
       (lqp-print 'terse "~%FIGURE UNIT conversions done.")
       data-list))))

;; Function that multiply an enhanced data-list by a list of exchange
rates
;; Inputs:
;; 1)
;; (((FIGURE ((VALUE "500,000.4" "250,300.2") (FORMAT FIGURE-DISC)
;;           (UNIT NJPY) (SCALE SCALE-1000)))
;;  (DATE ((VALUE "19,881,231"))))
;; ((FIGURE ((VALUE "1,000.67" "500.4") (FORMAT FIGURE-DISC) (UNIT NGBP)
;;           (SCALE SCALE-1000)))
;;  (DATE ((VALUE "19,870,228"))))
;; ((FIGURE ((VALUE "20,000" "13,000") (FORMAT FIGURE-DISC) (UNIT NUSD)
;;           (SCALE SCALE-1000)))
;;  (DATE ((VALUE "19,890,318")))))
;; 2) exchange rates list ("0.0069" "15.67" etc.)
;; 3) domain name
(defun multiply-data-list-by-ex-rates (data-list exchange-rates-
list domain)
```

```
(mapcar #'(lambda (row exchange-rate)
          (let* ((data-str (assoc domain row))
               (values-list (read-data-slot
                         data-str
                         'value))
              ; values-list = ("20,000" "13,000")
               (new-values-list (mapcar #'(lambda (value)
                                  (operate-two-strings
                                   value
                                   exchange-rate
                                   '*))
                                values-list)))
            (write-data-slot data-str 'value new-values-list)))
      data-list exchange-rates-list))

;; Function that multiply two strings
(defun operate-two-strings (string-1 string-2 operator)
(format nil "~A"
       (coerce
        (multiple-value-bind
         (expression)
         (read-from-string
          (format nil "~A" (list operator string-1 string-2)))
         (eval expression))
        'long-float)))

;;; Conversion of the aspect SCALE (MONEY-AMOUNT)

;;; Function to get the scale factor from the scale ID
;;; example: ID = SCALE-0.01 ==> scale factor = 0.01
(defun get-scale-factor-from-ID (scale-ID)
; scale-ID = SCALE-0.01
(let* ((scale-string (format nil "~A" scale-ID))
       ; scale-string = "SCALE-0.01"
       (scale-factor (make-string '0)) ; will be converted into float
after
       (str-length (length scale-string))
       (flag 'NIL)) ; to notify when '-' is encountered
  (dotimes (index str-length scale-factor)
     (let ((character (char scale-string index)))
       (cond
      ((not flag) ; "-" was not encountered yet
       (if (equal character '#\-) ; if this is it
           (setq flag 'T)))
      (flag ; if "-" was already encountered
       (setf scale-factor (concatenate 'string scale-factor (list
character)))))))
  (read-from-string scale-factor)))

;;; Function that converts the scale on a FIGURE-PLAIN figure
;;; The scale convertions are only made on FIGURE-PLAIN; if a figure is
;;; given in a different format ID, the ID is converted first.
(defun convert-scale-on-PLAIN (init-scale target-scale string-value)
(let* ((coef1 (get-scale-factor-from-ID target-scale))
       (coef2 (get-scale-factor-from-ID init-scale))
       (final-coef (/ coef2 coef1))
       (value (if (stringp string-value)
              (read-from-string string-value)
```

```
            string-value)))
  (convert-val-E-n-to-PLAIN (* value final-coef))))

;;; Conversion objects methods
(defun data-list-convert-into-SCALE-1 (data-list)
  (data-list-convert-figure-scale 'SCALE-1 data-list))

(defun data-list-convert-into-SCALE-1000 (data-list)
  (data-list-convert-figure-scale 'SCALE-1000 data-list))

(defun data-list-convert-into-SCALE-0.01 (data-list)
  (data-list-convert-figure-scale 'SCALE-0.01 data-list))

(defun data-list-convert-into-SCALE-100 (data-list)
  (data-list-convert-figure-scale 'SCALE-100 data-list))

(defun data-list-convert-into-SCALE-10 (data-list)
  (data-list-convert-figure-scale 'SCALE-10 data-list))

;;; Function that process an enhanced data-list
(defun data-list-convert-figure-scale (target-ID data-list)
  (let* ((domain (get-object target-ID 'data-type))
       (rep-component (get-object target-ID 'component))
       (new-values 'T)
       ; looks up the first row to see what the figure format of data is
       (figure-format (first (read-data-slot
                       (assoc domain (first data-list))
                       'format)))
       (plain-flag 'T)) ; to indicate whether the format is FIGURE-PLAIN

    ; if the format is not the reference format FIGURE-PLAIN, then
convert
    ; the data list into FIGURE-PLAIN

    (if (not (equal figure-format 'FIGURE-PLAIN))
      (progn (format t "~%Format conversion is required from ~A to ~A:"
figure-format 'figure-PLAIN)
             (setq plain-flag 'nil)
             (data-list-convert-figure-format 'FIGURE-PLAIN data-list)))

    ; scale convertion row by row
    (format t "~%Entering scale convertions...")
    (dolist (row data-list)
          (let* ((data-str (assoc domain row))
                (init-ID (first (read-data-slot data-str rep-component)))
                (values (read-data-slot data-str 'value)))
            (setq new-values (mapcar #'(lambda (value) (convert-scale-
on-PLAIN init-ID target-ID value)) values))
            (write-data-slot data-str 'value new-values)
            (write-data-slot data-str rep-component target-ID)))

    ; if it was not PLAIN originally, convert back into the original
format
    (if (not plain-flag)
      (progn (format t "~%Converting the format back to ~A" figure-
format)
             (data-list-convert-figure-format figure-format data-list)))
    (format t "~%Scale conversions done.")
```

```
    data-list))

;;; Representation conversion of the data type DATE

;;; Conversion of the representation aspect FORMAT

;;; we support the following date formats:
;;; date-YYYYMMDD, e.g., "19,900,204"
;;; date-mm/dd/yyyy, e.g., "04/02/1990"
;;; date-ddsmmsyy, e.g., "02 04 90"
;;; date-dd-mm-yy, e.g., "02-04-90"
;;; date-mm-dd-yy, e.g., "09-04-90"
;;; Reference format ID is date-mm/dd/yyyy

;;; Conversion routine to convert YY,YYM,MDD into MM/DD/YYYY
(defun convert-yyyymmdd-into-mm/dd/yyyy (date-value)
; the date looks like "19,900,402"
(let ((tmp-date date-value)
      (new-date (make-string '0)))
  (setq new-date (concatenate 'string new-date (list (char tmp-date
'5))))
  (setq new-date (concatenate 'string new-date (list (char tmp-date
'7))))
  (setq new-date (concatenate 'string new-date (list '#\/)))
  (setq new-date (concatenate 'string new-date (list (char tmp-date
'8))))
  (setq new-date (concatenate 'string new-date (list (char tmp-date
'9))))
  (setq new-date (concatenate 'string new-date (list '#\/)))
  (setq new-date (concatenate 'string new-date (list (char tmp-date
'0))))
  (setq new-date (concatenate 'string new-date (list (char tmp-date
'1))))
  (setq new-date (concatenate 'string new-date (list (char tmp-date
'3))))
  (setq new-date (concatenate 'string new-date (list (char tmp-date
'4))))))

;;; Conversion routine to convert MM/DD/YYYY into YY,YYM,MDD
(defun convert-mm/dd/yyyy-into-yyyymmdd (date-value)
; date-value looks like "04/02/1990"
(let ((tmp-date date-value)
      (new-date (make-string '0)))
  (setq new-date (concatenate 'string new-date (list (char tmp-date
'6))))
  (setq new-date (concatenate 'string new-date (list (char tmp-date
'7))))
  (setq new-date (concatenate 'string new-date (list '#\,)))
  (setq new-date (concatenate 'string new-date (list (char tmp-date
'8))))
  (setq new-date (concatenate 'string new-date (list (char tmp-date
'9))))
  (setq new-date (concatenate 'string new-date (list (char tmp-date
'0))))
  (setq new-date (concatenate 'string new-date (list '#\,)))
  (setq new-date (concatenate 'string new-date (list (char tmp-date
'1))))
```

```
  (setq new-date (concatenate 'string new-date (list (char tmp-date
'3))))
  (setq new-date (concatenate 'string new-date (list (char tmp-date
'4))))))

;;; Conversion routine to convert DD-MM-YY into MM/DD/YYYY
(defun convert-dd-mm-yy-into-mm/dd/yyyy (date-value)
; date-value looks like "02-04-90"
(let ((tmp-date date-value)
      (new-date (make-string '0)))
  (setq new-date (concatenate 'string new-date (list (char tmp-date
'3))))
  (setq new-date (concatenate 'string new-date (list (char tmp-date
'4))))
  (setq new-date (concatenate 'string new-date (list '#\/)))
  (setq new-date (concatenate 'string new-date (list (char tmp-date
'0))))
  (setq new-date (concatenate 'string new-date (list (char tmp-date
'1))))
  (setq new-date (concatenate 'string new-date (list '#\/)))
  (setq new-date (concatenate 'string new-date (list '#\1)))
  (setq new-date (concatenate 'string new-date (list '#\9)))
  (setq new-date (concatenate 'string new-date (list (char tmp-date
'6))))
  (setq new-date (concatenate 'string new-date (list (char tmp-date
'7))))))

;;; Conversion routine to convert MM/DD/YYYY into DD-MM-YY
(defun convert-mm/dd/yyyy-into-dd-mm-yy (date-value)
; date-value looks like "04/02/1990"
; returns "02-04-90"
(let ((tmp-date date-value)
      (new-date (make-string '0)))
  (setq new-date (concatenate 'string new-date (list (char tmp-date
'3))))
  (setq new-date (concatenate 'string new-date (list (char tmp-date
'4))))
  (setq new-date (concatenate 'string new-date (list '#\-)))
  (setq new-date (concatenate 'string new-date (list (char tmp-date
'0))))
  (setq new-date (concatenate 'string new-date (list (char tmp-date
'1))))
  (setq new-date (concatenate 'string new-date (list '#\-)))
  (setq new-date (concatenate 'string new-date (list (char tmp-date
'8))))
  (setq new-date (concatenate 'string new-date (list (char tmp-date
'9))))))


;;; Conversion routine to convert DDsMMsYY into MM/DD/YYYY
(defun convert-ddsmmsyy-into-mm/dd/yyyy (date-value)
(convert-dd-mm-yy-into-mm/dd/yyyy date-value))

;;; Conversion routine to convert MM/DD/YYYY into DDsMMsYY
(defun convert-mm/dd/yyyy-into-ddsmmsyy (date-value)
; date-value looks like "04/02/1990"
; returns "02 04 90"
(let ((tmp-date date-value)
```

```
      (new-date (make-string '0)))
  (setq new-date (concatenate 'string new-date (list (char tmp-date
'3))))
  (setq new-date (concatenate 'string new-date (list (char tmp-date
'4))))
  (setq new-date (concatenate 'string new-date '" "))
  (setq new-date (concatenate 'string new-date (list (char tmp-date
'0))))
  (setq new-date (concatenate 'string new-date (list (char tmp-date
'1))))
  (setq new-date (concatenate 'string new-date '" "))
  (setq new-date (concatenate 'string new-date (list (char tmp-date
'8))))
  (setq new-date (concatenate 'string new-date (list (char tmp-date
'9))))))


;;; Function: convert a value from init-ID to target-ID
;;; Reference ID: DATE-MM/DD/YYYY
(defun convert-date-format (init-ID target-ID date-value)
  (case init-ID
   (DATE-MM/DD/YYYY
    (case target-ID
     (DATE-MM/DD/YYYY date-value)
     (DATE-YYYYMMDD (convert-MM/DD/YYYY-into-YYYYMMDD date-value))
     (DATE-DD-MM-YY (convert-MM/DD/YYYY-into-DD-MM-YY date-value))
     (DATE-DDsMMsYY (convert-MM/DD/YYYY-into-DDsMMsYY date-value))
     (otherwise (format t "~%Error: ~A is not understood!" target-ID))))
   (DATE-YYYYMMDD
    (case target-ID
     (DATE-MM/DD/YYYY (convert-YYYYMMDD-into-MM/DD/YYYY date-value))
     (DATE-YYYYMMDD date-value)
     (DATE-DD-MM-YY (convert-MM/DD/YYYY-into-DD-MM-YY (convert-YYYYMMDD-
into-MM/DD/YYYY date-value)))
     (DATE-DDsMMsYY (convert-MM/DD/YYYY-into-DDsMMsYY (convert-YYYYMMDD-
into-MM/DD/YYYY date-value)))
     (otherwise (format t "~%Error: ~A is not understood!" target-ID))))
   (DATE-DD-MM-YY
    (case target-ID
     (DATE-MM/DD/YYYY (convert-DD-MM-YY-into-MM/DD/YYYY date-value))
     (DATE-YYYYMMDD (convert-MM/DD/YYYY-into-YYYYMMDD (convert-DD-MM-YY-
into-MM/DD/YYYY date-value)))
     (DATE-DD-MM-YY date-value)
     (DATE-DDsMMsYY (convert-MM/DD/YYYY-into-DDsMMsYY (convert-DD-MM-YY-
into-MM/DD/YYYY date-value)))
     (otherwise (format t "~%Error: ~A is not understood!" target-ID))))
   (DATE-DDsMMsYY
    (case target-ID
     (DATE-MM/DD/YYYY (convert-DDsMMsYY-into-MM/DD/YYYY date-value))
     (DATE-YYYYMMDD (convert-MM/DD/YYYY-into-YYYYMMDD (convert-DDsMMsYY-
into-MM/DD/YYYY date-value)))
     (DATE-DD-MM-YY (convert-MM/DD/YYYY-into-DD-MM-YY (convert-DDsMMsYY-
into-MM/DD/YYYY date-value)))
     (DATE-DDsMMsYY date-value)
     (otherwise (format t "~%Error: ~A is not understood!" target-ID))))
   (otherwise (format t "~%Error: ~A is not understood!" init-ID))))

;;; KOREL object methods:
```

```
;;; :data-list-convert-into-DATE-MM/DD/YYYY
;;; :data-list-convert-into-DATE-YYYYMMDD
;;; :data-list-convert-into-DATE-DD-MM-YY
;;; :data-list-convert-into-DATE-DDsMMsYY
;;; Input: enhanced data list.

(defun data-list-convert-into-DATE-MM/DD/YYYY (data-list)
  (let ((object-name (get-current-object)))
    (data-list-convert-date-format object-name data-list)))

(defun data-list-convert-into-DATE-YYYYMMDD (data-list)
  (let ((object-name (get-current-object)))
    (data-list-convert-date-format object-name data-list)))

(defun data-list-convert-into-DATE-DD-MM-YY (data-list)
  (let ((object-name (get-current-object)))
    (data-list-convert-date-format object-name data-list)))

(defun data-list-convert-into-DATE-DDsMMsYY (data-list)
  (let ((object-name (get-current-object)))
    (data-list-convert-date-format object-name data-list)))

;; main function: data-list-convert-date-format
(defun data-list-convert-date-format (target-ID datalist)
  (let* ((domain (get-object target-ID 'data-type))
       (rep-component (get-object target-ID 'component))
       (new-values 'nil))
    (format t "~%Entering DATE FORMAT convertions...")
    (dolist (row datalist)
          (let* ((data-str (assoc domain row))
                (init-ID (first (read-data-slot data-str rep-component)))
                (values (read-data-slot data-str 'value)))
            (setq new-values (mapcar #'(lambda (value) (convert-date-
format init-ID target-ID value)) values))
            (write-data-slot data-str 'value new-values)
            (write-data-slot data-str rep-component target-ID)))
      (format t "~%DATE FORMAT convertions done.")
      datalist))

;;; Representation conversion of the data type YEAR

;;; Conversion of the representation aspect FORMAT

(defun convert-yyyy-into-yy (year-value)
(let ((new-date (make-string '0)))
  (setq new-date (concatenate 'string new-date (list (char year-value
'2))))
  (setq new-date (concatenate 'string new-date (list (char year-value
'3))))))

(defun convert-yy-into-yyyy (year-value)
; YEARS < 2000
(let ((new-date (make-string '0)))
  (setq new-date (concatenate 'string new-date (list '#\1)))
  (setq new-date (concatenate 'string new-date (list '#\9)))
  (setq new-date (concatenate 'string new-date (list (char year-value
'0))))
```

```
  (setq new-date (concatenate 'string new-date (list (char year-value
'1))))))

;; main function
(defun convert-year-format (init-ID target-ID year-value)
  (case init-ID
   (YEAR-YYYY
    (case target-ID
     (YEAR-YYYY year-value)
     (YEAR-YY (convert-yyyy-into-yy year-value))
     (otherwise (format t "~%Error: ~A is not understood!" target-ID))))

   (YEAR-YY
    (case target-ID
     (YEAR-YYYY (convert-yy-into-yyyy year-value))
     (otherwise (format t "~%Error: ~A is not understood!" target-ID))))

   (otherwise (format t "~%Error: ~A is not understood!" init-ID))))

;;; KOREL object methods:
;;; :data-list-convert-into-YEAR-YYYY
;;; :data-list-convert-into-YEAR-YY
;;; Input: enhanced data list.

(defun data-list-convert-into-YEAR-YYYY (data-list)
  (let ((object-name (get-current-object)))
    (data-list-convert-year-format object-name data-list)))

(defun data-list-convert-into-YEAR-YY (data-list)
  (let ((object-name (get-current-object)))
    (data-list-convert-year-format object-name data-list)))

;; main function: data-list-convert-date-format
(defun data-list-convert-year-format (target-ID datalist)
  (let* ((domain (get-object target-ID 'data-type))
       (rep-component (get-object target-ID 'component))
       (new-values 'nil))
    (lqp-print 'normal "~%~%Entering YEAR FORMAT convertions...")
    (lqp-print 'normal "~%Input = ~A" datalist)
    (dolist (row datalist)
          (let* ((data-str (assoc domain row))
               (init-ID (first (read-data-slot data-str rep-component)))
               (values (read-data-slot data-str 'value)))
            (setq new-values (mapcar #'(lambda (value) (convert-year-
format init-ID target-ID value)) values))
            (write-data-slot data-str 'value new-values)
            (write-data-slot data-str rep-component target-ID)))
      (lqp-print 'normal "~%YEAR FORMAT convertions done.")
      (lqp-print 'normal "~%Output = ~A" datalist)
      datalist))
```

***

```
;;; Data Filter Module

(defun data-filter (data-list LQP-object-name PreConvertQuery
RemoteQuery predicates-1 predicates-2 sender-data-type-catalogue
receiver-data-type-catalogue lqp-strategy initial-columns)
```

```
(let ((tmp-data-list))
  (cond ; initial cond
   (predicates-1 ; predicates-1 is not nil
    (lqp-print 'terse "~%~%Data Filter Step #1:")
    (lqp-print 'terse "~%First-class predicates = ~A" predicates-1)
    (lqp-print 'terse "~%Data is converted into the sender's data
definitions for the attributes in ~A" predicates-1)

    ; Convert the predicates-1 attributes
    (data-list-convert data-list 'sender-semantics predicates-1
receiver-data-type-catalogue lqp-strategy 'no)

    ; Conversion of the data elements in the data set in order to
    ; download in the auxiliary table.
    (let* ((LQP-attributes-catalogue (get-object receiver-data-type-
catalogue 'attributes-catalogue))
          (aux-table-object (get-object LQP-object-name 'auxiliary-
table))
          (aux-table-data-type-catalogue (get-object aux-table-object
'data-type-catalogue))
          (aux-table-attributes-catalogue (get-object aux-table-data-
type-catalogue 'attributes-catalogue))
          (not-predicates-1 (list1-list2 (second RemoteQuery) predicates-
1))
          (all-columns-in-order (get-object (get-object aux-table-data-
type-catalogue 'attributes-catalogue) 'attributes))
          (aux-table-directory (get-object aux-table-object 'database-
directory))
          (database-name (get-object aux-table-object 'database))
          (table-name (get-object aux-table-object 'table))
          (strategy-for-aux-table)
          (tmp-query))

      ; Conversion of data elts corresponding to predicates-1
      ; Build strategy
      (setq strategy-for-aux-table (build-lqp-data-conversion-strategy
sender-data-type-catalogue aux-table-data-type-catalogue))

      ; Convert
      (data-list-convert data-list 'receiver-semantics predicates-1 aux-
table-data-type-catalogue strategy-for-aux-table 'no)

      ; Conversion of data elts corresponding to other than predicates-1
      ; Build strategy
      (setq strategy-for-aux-table (build-lqp-data-conversion-strategy
receiver-data-type-catalogue aux-table-data-type-catalogue))

      ; Convert data-list
      (data-list-convert data-list 'receiver-semantics not-predicates-1
aux-table-data-type-catalogue strategy-for-aux-table 'no)

      ; Download data-list
      (download-data data-list LQP-attributes-catalogue aux-table-object
all-columns-in-order aux-table-directory database-name table-name)

      ; Map database to aux attributes in PreConvertQuery
      (setq tmp-query (map-real-to-aux-attributes-in-query
PreConvertQuery LQP-attributes-catalogue))
```

```
      ; Select data with tmp-query
      (lqp-print 'terse "~%~%Data Filter Step #3:")
      (setq tmp-data-list (send-message aux-table-object :get-data
sender-data-type-catalogue (list table-name (second tmp-query) (third
tmp-query))))

      ; Delete the data put in the table
      (setq tmp-query (map-real-to-aux-attributes-in-query RemoteQuery
LQP-attributes-catalogue))

      (send-message aux-table-object :delete (list table-name (second
tmp-query) (third tmp-query)) sender-data-type-catalogue aux-table-
directory database-name)

      ; Update the strategy, since the data obtained from the auxiliary
table
      ; is in FIGURE-PLAIN
      (write-semantics (get-sub-strategy lqp-strategy 'figure)
'receiver-semantics 'format 'FIGURE-PLAIN)

      ; Map aux to database attributes in tmp-data-list1
      (setq tmp-data-list (map-aux-to-real-attributes-in-data-set tmp-
data-list aux-table-attributes-catalogue))

      ; Convert the data for the not-predicates-1 attributes
      (data-list-convert tmp-data-list 'sender-semantics not-predicates-
1 receiver-data-type-catalogue lqp-strategy 'no))
    ) ; end of predicates-1 is not nil

   (T ; predicates-1 is nil
    (cond
     (predicates-2 ; predicates-2 is not nil
      )
     (T ; predicates-2 is nil
      (data-list-convert
       data-list
       'sender-semantics
       (second RemoteQuery)
       receiver-data-type-catalogue
       lqp-strategy
       'no)))))
  (extract-subset-of-datalist tmp-data-list initial-columns)))

                                    ***
;;; Local DBMS utilities

;; GET-DATA method (for local DBMS)
(defun get-INFORMIX2E-data (sender-data-type-catalogue abstract-
local-query)
(let* ((initial-columns (second abstract-local-query))
       (LQP-object-name (get-current-object))
       (switch-conversion-facility (get-object LQP-object-name 'switch-
conversion-facility)))
(if (eq switch-conversion-facility 'on)
    (let* ((receiver-data-type-catalogue (get-object LQP-object-name
'data-type-catalogue))
```

```
        (lqp-strategy (build-lqp-data-conversion-strategy sender-data-
type-catalogue receiver-data-type-catalogue)) ; build the strategy
        (data-types-list (get-object receiver-data-type-catalogue
'data-types))
        (predicate-filter (get-object LQP-object-name 'predicate-
filter)))
      ; parse the query:
      (multiple-value-bind
       (PreConvertQuery
      RemoteQuery
      predicates-1
      predicates-2)
       (parse-query receiver-data-type-catalogue abstract-local-query
data-types-list predicate-filter 'receiver-semantics lqp-strategy)

       ;; fetch the data
       (let ((data-list (get-data-LQP-main LQP-object-name
RemoteQuery)))
       (cond ; initial cond
        (predicates-1 ; predicates-1 is not nil
         (lqp-print 'terse "~%predicates-1 is not null.")
         )
        (T ; predicates-1 is nil
         (cond
          (predicates-2 ; predicates-2 is not nil
           (lqp-print 'terse "~%Predicates-2 is not null.")
           )
          (T ; predicates-2 is nil
           (format t "~%predicates-1 and predicates-2 are both null.")
           (data-list-convert
            data-list
            'sender-semantics
            (second RemoteQuery)
            receiver-data-type-catalogue
            lqp-strategy
            'yes)))))
       ; returns initial columns
       (extract-subset-of-datalist data-list initial-columns))))

  ; if switch-conversion-facility = OFF
  (get-data-LQP-main LQP-object-name abstract-local-query))))

;;; Data Downloading Module

;;; Function that downloads data in a local auxiliary relational
database table
;;; with informix DBMS package on the computer mit2e.
(defun download-INFORMIX2E-data (data-list sender-data-type-
catalogue columns-list-in-order database-directory database-name table-
name)
  (let* ((LQP-object-name (get-current-object))
       (receiver-data-type-catalogue (get-object LQP-object-name 'data-
type-catalogue))
       (lqp-strategy (build-lqp-data-conversion-strategy sender-data-
type-catalogue receiver-data-type-catalogue)))
    (if lqp-strategy
      (progn
```

```
        (data-list-convert data-list 'receiver-semantics (first data-
list) receiver-data-type-catalogue lqp-strategy 'no)
        (lqp-print 'terse"~%The data needs to be converted into ~A's
data representation" table-name))
      (lqp-print 'terse "~%The data is already in ~A's data
representation" LQP-object-name))
    (let* ((data-list-in-order (reorder-data-list data-list columns-
list-in-order))
         (unique-file-name (unique-name))
         (data-file-name (concatenate 'string *tmp-files-directory*
unique-file-name))
         (download-script (get-object LQP-object-name 'download-script))
         (script-file (concatenate 'string *comm-server-directory*
download-script)))
      (write-into-file data-file-name data-list-in-order)
      (system (unix-format "~A ~A ~A ~A ~A" script-file data-file-name
database-directory database-name table-name)))))

;;; Function to reorder the columns in data list
(defun reorder-data-list (d-list c-list)
; Initializations
(let ((my-column-list (first d-list))
      (d-list (cdr d-list))
      (new-d-list '())
      (max (length c-list)))
; For every element in the c-list
(dolist (d-row d-list new-d-list)
   (let ((new-d-row '())
       (counter '0))
     (dolist (c-element c-list)
      (let* (;(c-element (if (not (stringp c-element)) (format nil "~A"
c-element) c-element))
             (position (search-column c-element c-list '0))
             (is-it-there? (search-column c-element my-column-list '0)))
        (setq counter (1+ counter))
        (if is-it-there?
            (setq new-d-row (append new-d-row (list (nth is-it-there? d-
row))))
          (setq new-d-row (append new-d-row (list '" "))))
        (if (not (= counter max))
            (setq new-d-row (append new-d-row (list '"|")))))))
     (setq new-d-list (append new-d-list (list new-d-row)))))))

;;; Write-into-file turns a data-list into a file.
(defun write-into-file (file-name data-list)
(with-open-file (stream file-name :direction :output)
      (dolist (row data-list)
            (dolist (element row)
                  (princ element stream))
            (terpri stream))))

;;; Downloading module
;;; ------------------
(defun download-data (data-list LQP-attributes-catalogue aux-table-
object all-columns-in-order aux-table-directory database-name table-
name)
(let* ((tmp-data-list)
```

```
       (aux-table-data-type-catalogue (get-object aux-table-object
'data-type-catalogue)))
  ; Check whether some mapping between database column names
  ; and auxiliary table is necessary.
  (setq tmp-data-list (map-real-to-aux-attributes-in-data-set data-list
LQP-attributes-catalogue))

  ; Send message to INFORMIX2E object
  (send-message aux-table-object :download tmp-data-list aux-table-data-
type-catalogue all-columns-in-order aux-table-directory database-name
table-name)))

;;; Data Deleting Module

;;; Function to delete records in a table in the Informix DBMS
(defun delete-INFORMIX2E-data (query sender-data-type-catalogue
database-directory database-name)
; query: the query that will delete the records
; sender-data-type-catalogue indicates the semantics of the query
(let* ((LQP-object-name (get-current-object))
       (table-name (first query))
       (delete-constraint (parse-SQL-conds (third query)))
       (switch-conversion-facility (get-object LQP-object-name 'switch-
conversion-facility))
       (del-script-file (concatenate 'string *comm-server-directory*
                                (get-object LQP-object-name 'delete-
script))))
(if (eq switch-conversion-facility 'on) ; If the conversion facility is
"ON"
    (let* ((receiver-data-type-catalogue (get-object LQP-object-name
'data-type-catalogue))
         (lqp-strategy (build-lqp-data-conversion-strategy sender-data-
type-catalogue receiver-data-type-catalogue)) ; build the strategy
         (data-types-list (get-object receiver-data-type-catalogue
'data-types))
         (predicate-filter (get-object LQP-object-name 'predicate-
filter)))
      (if lqp-strategy
        ; if the strategy is not null, then conversion is needed
        ; parse the query:
        (multiple-value-bind
         (PreConvertQuery
          RemoteQuery
          predicates-1
          predicates-2)
         (parse-query receiver-data-type-catalogue query data-types-list
predicate-filter 'receiver-semantics lqp-strategy)
         (if (not (equal PreConvertQuery RemoteQuery))
             (progn
             (format t "~%Error in delete-INFORMIX2E-data:
PreConvertQuery and RemoteQuery are not identical")
             (return))
           (let ((delete-constraint (parse-SQL-conds (third
RemoteQuery))))
             (system (unix-format "~A ~A ~A ~A ~A" del-script-file
delete-constraint database-directory database-name table-name)))))
     (system (unix-format "~A ~A ~A ~A ~A" del-script-file delete-
constraint database-directory database-name table-name))))
```

```
  (system (unix-format "~A ~A ~A ~A ~A" del-script-file delete-
constraint database-directory database-name table-name)))))

                                    ***
;;; I.P. Sharp Disclosure

;;; Main function
(defun get-IPSHARP-data (sender-data-type-catalogue abstract-local-
query)
;; sender-data-type-catalogue: data type catalogue name of the sender
;; abstract-local-query: query.
(let* ((initial-columns (second abstract-local-query))
       (LQP-object-name (get-current-object))
       (switch-conversion-facility (get-object LQP-object-name 'switch-
conversion-facility)))
(if (eq switch-conversion-facility 'on)
    (let* ((receiver-data-type-catalogue (get-object LQP-object-name
'data-type-catalogue))
         (lqp-strategy (build-lqp-data-conversion-strategy sender-data-
type-catalogue receiver-data-type-catalogue)) ; build the strategy
         (data-types-list (get-object receiver-data-type-catalogue
'data-types))
         (predicate-filter (get-object LQP-object-name 'predicate-
filter)))
      ; parse the query:
      (multiple-value-bind
       (PreConvertQuery
      RemoteQuery
      predicates-1
      predicates-2)
       (parse-query receiver-data-type-catalogue abstract-local-query
data-types-list predicate-filter 'receiver-semantics lqp-strategy)
       ;; fetch the data !!!
       (let ((data-list (fetch-IPSHARP-data LQP-object-name
RemoteQuery)))
       (data-filter data-list LQP-object-name PreConvertQuery
RemoteQuery predicates-1 predicates-2 sender-data-type-catalogue
receiver-data-type-catalogue lqp-strategy initial-columns))))))))

;;; Function for fetching the data with RemoteQuery
(defun fetch-IPSHARP-data (LQP-object-name abs_local_query)
  (multiple-value-bind (SQL columns) (form-SQL abs_local_query)
      (lqp-print 'normal "SQL query to be sent to DBMS....~%~A~%" SQL)
      (lqp-print 'verbose "Columns reported by FORM-SQL...~%~A~%"
columns)
      (let ((data-file (connect LQP-object-name SQL 'EFFICIENT)))
        (lqp-print-file 'normal data-file)
        (cons columns (read-filter-ipsharp-output data-file *common-lqp-
directory*)))))

;;; Data File Reader for I.P. Sharp Disclosure

(defun read-filter-ipsharp-output (file comdir &aux tmp info)
  (if (probe-file file)
      (progn (lqp-print 'terse "~%Reading awk filter output file...")
           ;;(system (format nil "~A/preREAD ~A" comdir file))
           (with-open-file (data file :direction :input)
            (loop (let ((line (read-line data nil 'EOF)))
```

```
                (cond ((equal line 'EOF)
                       (lqp-print 'terse "Done.~%")
                       (return (remove-if #'null (reverse info))))
                      ((equal line "")
                       (setq info (cons (reverse tmp) info))
                       (setq tmp '()))
                      (t (setq tmp (cons line tmp))))))))
      (lqp-print 'terse "READ-STANDARD-TABLE -- File '~A' not found!"
file)))
```

***

```
;;; I.P. Sharp Currency
(defun get-CURRENCY-data (sender-data-type-catalogue abstract-local-
query)
;; sender-data-type-catalogue: data type catalogue name of the sender
;; abstract-local-query: query.
(let* ((initial-columns (second abstract-local-query))
       (LQP-object-name (get-current-object))
       (switch-conversion-facility (get-object LQP-object-name 'switch-
conversion-facility)))
(if (eq switch-conversion-facility 'on)
    (let* ((receiver-data-type-catalogue (get-object LQP-object-name
'data-type-catalogue))
         (lqp-strategy (build-lqp-data-conversion-strategy sender-data-
type-catalogue receiver-data-type-catalogue)) ; build the strategy
         (data-types-list (get-object receiver-data-type-catalogue
'data-types))
         (predicate-filter (get-object LQP-object-name 'predicate-
filter)))
      ; parse the query:
      (multiple-value-bind
       (PreConvertQuery
      RemoteQuery
      predicates-1
      predicates-2)
       (parse-query receiver-data-type-catalogue abstract-local-query
data-types-list predicate-filter 'receiver-semantics lqp-strategy)
       ;; fetch the data
       (let ((data-list (get-data-LQP-main LQP-object-name
RemoteQuery)))
        (cond ; initial cond
         (predicates-1 ; predicates-1 is not nil
          (format t "~%predicates-1 is not null.")
          )
         (T ; predicates-1 is nil
          (cond
           (predicates-2 ; predicates-2 is not nil
            (format t "~%Predicates-2 is not null.")
            )
           (T ; predicates-2 is nil
            (data-list-convert
             data-list
             'sender-semantics
             (second RemoteQuery)
             receiver-data-type-catalogue
             lqp-strategy
             'yes)))))
       ; returns initial columns
```

```
      (extract-subset-of-datalist data-list initial-columns)))))))

;;; Data File Reader for I.P. Sharp Currency

;;; Pre-Read-Filter for I.P. Sharp
;;; File cleaner
(defun ip-pre-read-filter (init-data-file lqpdir)
; all the file names are assumed to be with a directory path name
(system (unix-format "cat ~A | ~Acurcleaner > ~Atmp-file" init-data-file
lqpdir lqpdir))
(system (unix-format "mv ~Atmp-file ~A" lqpdir init-data-file)))

;;; File Reader
(defun read-ip-currency (columns file-name currency-request-in-
string)
(let ((data-list 'nil)
      (dates-and-codes (read-token-in-currency-request currency-request-
in-string))
      (dates-list)
      (codes-list)
      (final-data-list)
      (found-token 'T))
(with-open-file (data-stream file-name :direction :input)
   (do ((row (read data-stream nil)
             (read data-stream nil)))
       ((not row) data-list)
       ; (lqp-print 'normal "~%Row at hand: ~A" row)
       (if
       (and (and (not (equal row '*****BEGINNING))
                 (and (not (equal row 'MARKER*****))
                      (and (not (equal row '********END))
                           (not (equal row 'MARKER********))))))
         ; it is not a marker
         (if (and (not (equal row '0)) (not found-token))
             ; the token has not been found yet, and the one at hand is
not 0
             (progn
             (setf data-list (append data-list (list (list (format nil
"~A" row)))))
             (setq found-token 'T)))
        ; it is a marker
        (if (equal row '*****BEGINNING) ; when starting a new search
            (if found-token ; if the token was found in the previous line
              (setq found-token 'NIL) ; start a new token search
              ; if it was not found, put a N/A tag
              (setf data-list (append data-list (list (list
"N/A"))))))))))
; dates-and-codes
; ("25-04-90" "NJPY" "25-04-90" "NFRF" "26-04-90" "NGBP" "26-04-90"
;  "NFRF" "27-04-90" "NJPY" "27-04-90" "NFRF")
; columns = (exchange-rate code date)
; create list of dates, and list of codes
(let ((max-index (/ (length dates-and-codes) 2)))
  (dotimes (index max-index)
         (setq dates-list (append dates-list (list (pop dates-and-
codes))))
         (setq codes-list (append codes-list (list (pop dates-and-
codes))))))
```

```
;; add dates and codes data
(dolist (row data-list final-data-list)
      (setq final-data-list (append final-data-list (list (append row
(list (pop codes-list)) (list (pop dates-list)))))))
(append (list
       (mapcar #'(lambda (column) (format nil "~A" column))
             columns))
      final-data-list)))

;;; Read token one after the other
(defun read-token-in-currency-request (string)
(let ((new-token (make-string '0))
      (max-index (length string))
      (token-list 'NIL)
      (first-found 'NIL)
      (second-found 'NIL))
  (dotimes (index max-index token-list)
     (let ((character (char string index)))
       (cond
      ((equal character #\") ; found a \"
       (if first-found ; if already found the first one, then it is the
2nd.
           (setq second-found 'T)
         (setq first-found 'T)))
      (T ; is is not a \"
       (if first-found ; i'm in the middle of the token
           (setq new-token (concatenate 'string new-token (list
character))))))
       (if (and first-found second-found)
         (progn
           ; put token in token-list
           (setq token-list (append token-list (list new-token)))
           ; re-initialize the variables
           (setq new-token (make-string '0))
           (setq first-found 'nil) (setq second-found 'nil)))))))
```