

An Object-Oriented Approach To Distributed Data Management

M. P. Papazoglou and L. Marinos

Gesellschaft für Mathematik und Datenverarbeitung, (German National Research Center for Computer Science), Schloß Birlinghoven, 5205 St. Augustin I, West Germany

An object-oriented framework is presented for the development of an integrated environment in which a collection of independent heterogeneous data repositories are merged to form a loosely coupled data base environment. This framework promotes interoperability and extensibility of heterogeneous data managers as it facilitates the development of well-defined interfaces to a wide variety of existing information sources. To this end, the envisaged system creates the illusion of a single integrated data base that can be queried in a uniform and consistent manner.

1. INTRODUCTION

Distributed data base technology evolved from the need to integrate large volumes of corporate information, stored in pre-existing data repositories, to lower production and maintenance costs. These costs can be scaled down by incorporating the appropriate software tools aimed at the distribution of the sharable data. Users can manipulate the data contained in the various data bases without having to modify already existing data base applications and without causing any migration of data between data bases.

A distributed information system* typically consists of multiple computer systems (called *sites*) that are physically interconnected via a common communication network. Each of the distributed data base sites supports its own individual application developed on a locally supported *component data base* which revolved around a conceptual schema called the *component schema*. Such applications, called *local applications*, are normally maintained in differently structured data bases. On the other hand, a *distributed data base application*

requires access to multiple local applications whose data can be interpreted within a common semantic context. Consider for example a commercial application, relevant to a given enterprise or organization, which has been materialized on top of independently developed local applications possibly servicing the needs of diverse departments within the organization.

Distributed information systems rely on a *distributed data manager* which provides coordinated access to heterogeneous data stored in the disparate sites of the system. The distributed data manager also materializes the *distributed data model* whose prime purpose is to furnish the entire system with the appropriate structural and semantical capabilities so that data integration is made viable. One of the prime purposes of the distributed data model is to set up the environment which would facilitate the communication between the users of diverse and incompatible information systems, and assist in particular with the uniform representation and integration of heterogeneous data from one site to another.

Some of the contemporary distributed information systems partially lack the notion of the distributed data manager and thus comprise a number of disconnected data management subsystems patched together to provide an ad hoc system design (i.e., restricted to specific systems) with temporal functionality [1, 2]. Obviously, these systems are poorly engineered and thus unreliable and expensive to maintain, increment, or modify. Other distributed data information systems, called *logically centralized*, have tried to overcome these and related obstacles by promoting the notion of distributed data management through some tight form of data integration. However, tight data integration seems not be the appropriate solution as tightly coupled systems may become a performance bottleneck while complicating the addition and modification of component data bases.

The envisaged architectural framework tries to remedy this situation by suggesting that distributed data base

*We use the term *information system* in the ISO sense to denote any source of information consisting of a conceptual schema, an information base, which is a source of persistent data, and an information processor.

Address correspondence to M. P. Papazoglou, Australian National University, Department of Computer Science, GPO Box 4, Canberra ACT 2601, Australia.

management facilities should be the converging point in a distributed data-intensive application environment which supports a loose form of data integration. The long-term objective is to design a general architectural framework supporting independence from the physical distribution of the component data bases, while providing the users with a transparent view of the information that is scattered across the data base sites. The integration may be viewed as a loose integration, as our intention is not to bind pre-existing data bases into a fixed data base but rather to provide the user with uniform, integrated access to a set of data bases—some of which may contain semantically related and probably replicated information. Typically, these data base systems will be heterogeneous, that is, there will be a variety of incompatible component data base management systems (DBMSs) with different underlying data models and query languages. Obviously, the proposed architecture can, with slight adaptations, meet the requirements of homogeneous data bases as they comprise a subcase of heterogeneous information systems.

In particular, we will describe the requirements of a distributed object-oriented data management system (DOODMS) that is designed specifically to integrate heterogeneous information resources into a single object space. The DOODMS achieves this by mapping the data and processing resources of the entire system into a unique system wide object space, defined in terms of the unifying o-o data model, called the *distributed object data model* (DODM). Accordingly, the prime purpose of the DOODMS is to make the component data bases interoperable by providing a set of well-defined interfaces, implemented as extensions of already existing component DBMSs, while preserving their autonomy, i.e., avoiding any changes to the underlying data bases and procedures.

2. FUNCTIONAL REQUIREMENTS AND DESIGN ISSUES

The collective management of autonomous, cooperating heterogeneous information sources[†] relies on the existence of a single integrated environment facilitating their smooth symbiosis. Such an integrated environment is based on the assumption that all of the data used by a conglomeration of related, though distinct, applications should be first uniformized and then unified into a single accumulation of data.

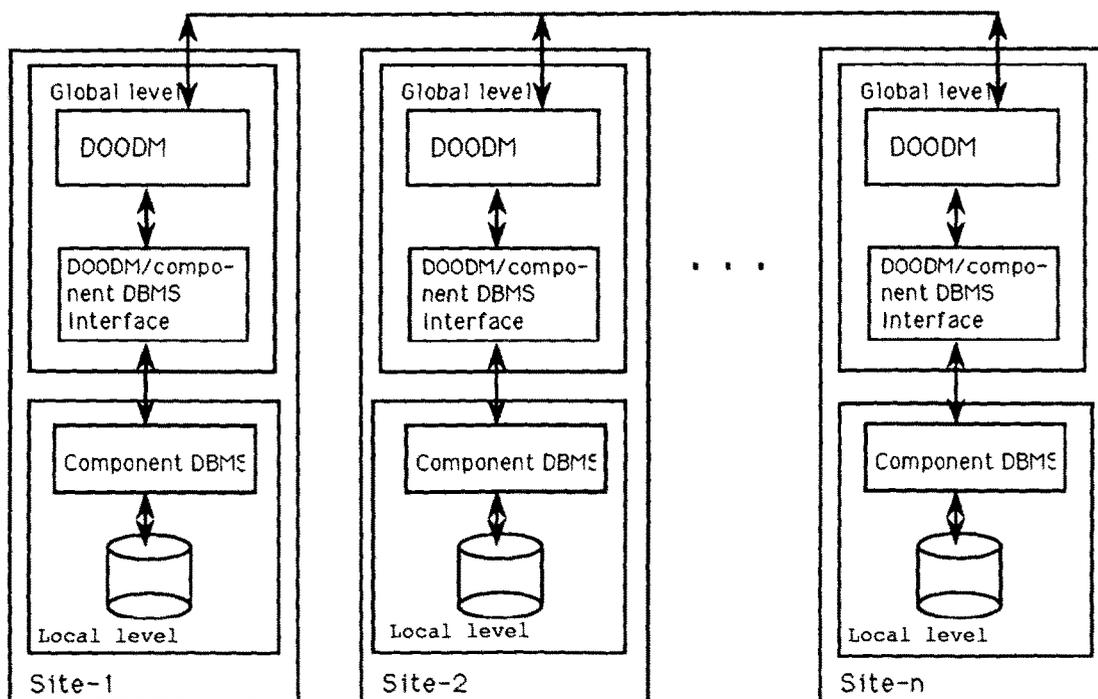
Retrieving and manipulating information in such an integrated environment represents quite a formidable task. The component data bases of such a system typ-

ically present differences in their underlying schemas, as they are expressed in terms of different data models, and have their own data manipulation languages which are supported by qualitatively different data base management systems (DBMSs). The idea of using a single semantically powerful data model to express a common unifying view of a variety of disparate data bases is quite appealing. Such a unifying view consists of a *distributed conceptual schema* (DCS) and mappings from DCS entities and properties to corresponding ones in the underlying conceptual schemas of the component data bases. The DCS is a highly logical view of the information content of the integrated system which does not require that the individual data bases are physically integrated. Rather, individual data bases tend to proliferate autonomously throughout this integrated environment while all global data base access, i.e., access to the distributed application, and manipulation operations are mediated through this new form of conceptual schema. However, this presumes the alleviation of a series of problems emerging from the distributed conceptual schema level [3, 4]. Among these problems one can distinguish such issues as data inconsistency, naming conflicts, naming equivalences, etc. These problems are additionally aggravated by the fact that constructs within a particular data model in the system may not exist in another data model (refer to Section 4.1).

The prime difference between the DCS and the *global schema* used in logically centralized systems such as Multibase [5] or ADDS [6] is that a global schema is directly associated with logical centralization of all of the organizational information, possibly under the auspices of a centralized data processing element. In contrast to this, a DCS is a virtual schema for which there does not exist any corresponding physical data base; rather, mapping specifications are provided which define how the distributed conceptual schema is derived from the individual conceptual schemas maintained by the component data bases.

In envisaged architecture, the distributed conceptual schema is defined only once at the preintegration phase, resides in each individual data base site of the entire distributed complex, and is operated upon by the DOODMS. This type of conceptual schema has the ability to adjust itself to all potential user changes. All changes pertaining to the DCS are made locally and are automatically propagated to all of the sites in the system. Subsequently, these changes will be recorded at each individual DCS in the system so that at the conclusion of this phase all DCSs are identical. A DCS is free to change its logical structure without affecting the structure of the component schemas. This architecture introduces a *semi-decentralized* nature and its prime purpose is to merge a collection of pre-existing information sys-

[†]In this text, we use the term information source and data base interchangeably.



tems into a loosely coupled association in order to share and exchange semantically related information.

In this context, we perceive the composite information system as comprising of the union of two logical conceptual levels at each site: the *global* and the *local* levels. All operations that are issued locally and can be performed locally are handled at the local level by the component DBMS, whereas those operations requiring information or assistance from a "foreign" data base in the network are performed at the global level by the DOODMS; see Figure 1. This approach attempts to embed the advantages of logical centralization in a system presenting a semi-decentralized nature. Semi-decentralized systems are characterized by the lack of a central authority, e.g., a global data base administrator, and are based on equal and autonomous local data bases with all intersite communications being performed by well-defined schema-controlled interfaces.

This architecture takes the view that the universe of discourse in a distributed data base application should not be viewed through relations or functions operating on entities (as in the case of Multibase) but rather through both coarser-grained and finer-grained organizations of information than those offered by systems based either on the relational or on semantic data models. In sum, it focuses on distributed information management at an arbitrary level of granularity encountered only at the level of objects as offered by its unifying object data model, viz. the DODM, used to develop the DCS. In the following section, we will try to justify

Figure 1. Communicating sites in a loosely coupled distributed data base.

this thesis by arguing why object-oriented data models are considered to be superior than relational or semantic data models when distribution and heterogeneity issues are concerned.

3. ENHANCING THE EXPRESSIVENESS OF CONCEPTUAL SCHEMAS

The techniques and methodologies used to design a distributed information system are very specific to the nature of the DCS through which the distributed application is modeled. Until recently, a series of distributed heterogeneous data base systems have evolved around a global schema based on the relational paradigm [1, 7, 8]. Furthermore, it has been argued that the relational model is the model best suited for supporting distributed data base applications [9]. In our opinion, two of the most well-known virtues of the relational data model that could justify this choice by supporting data distribution are:

1. The provision of a uniform set-oriented query language for the manipulation of data. This means that there is no need to follow record-at-a-time navigational links which extend across the sites of the system.
2. A uniform value-based representation espoused for

interrecord relationships. Loosely speaking, it is possible to retrieve data from multiple relations only on the basis of common attribute values.

However, it is widely accepted that the data structures which are supported by the relational model do not adequately facilitate the design, evolution, and use of such complex structures as distributed heterogeneous information systems. The relational model presents significantly limited capabilities for properly expressing the semantics of data base entities and relating a data base to its natural application environment [10, 11]. This comes in contradistinction to a series of stringent requirements emanating from the essence of the heterogeneous distributed information complex. Furthermore, the relational model presents some notable drawbacks as far as heterogeneity and data distribution are concerned.

In the following, we mention succinctly the most striking limitations of the traditional relational systems with respect to heterogeneity and data distribution:

1. As all data in a relational system exist at a structural level, there exists no way to specify an abstract type in terms of abstract properties.
2. Relational systems make no provisions for extension in order to allow the underlying DBMS to represent a more general hierarchy or taxonomy of types, subtypes, and instances.
3. Relational systems are primarily value-based which means that tuples in a relational data base can be distinguished only on the basis of their attribute values.
4. Relational systems are unable to represent an entity both independently and in terms of a relationship in which it participates.
5. Relational systems record the data base semantics separately from the data (i.e., semantics are not readily apparent from the schema). Furthermore, semantics must be separately specified by the data base designer and consciously applied by the user [11].

From the above points, it becomes clear that the design of complex data base structures could be greatly ameliorated if we were able to represent naturally both general and specific application areas, whilst offering high-level data structuring and manipulation primitives that are oriented towards enhancing the expressiveness of the component schemas [4]. This requirement for advanced data modeling concepts and the ever-growing need to capture more meaning of the information stored in the data base lead to the emergence of semantic data models [12]. Semantic data base models attempt to embed the semantics of an application environment into the data base schema to make the data base useful and evolvable, thereby providing an increased degree of expressiveness to the modeler. Semantic data base modeling

concepts were successfully used in the Multibase distributed heterogeneous system which realizes a unified global schema based on the functional model. Such kind of semantic models facilitate the development of more complex data base structures while allowing a substantial amount of inferencing to be made on the basis of their structures. However, semantic models incorporate only rich structural aspects of the modeled real-world entities and cannot express behavioral aspects of application data bases.

When integrating distributed information sources, it is highly desirable to both increase the expressiveness of the data bases so that more information can reside in a manageable and usable form in the data base itself, and to also capture the dynamic properties (behavior) of distributed applications. Such dynamic properties can be readily expressed by *object-oriented (o-o) data models* which encapsulate the behavioral properties of modeled entities (objects) in methods which accompany the conventional application data. Consequently, a combination of the rich structural properties of semantic data models with the behavioral aspects of the o-o data models in a single model, henceforth referred to as a *high-level object-oriented* data model, is the obvious solution when dealing with distributed conceptual modeling issues.

The remainder of this paper presents an analysis of the basic principles and properties of high-level o-o data models that can be used to facilitate the uniformization and integration of distributed heterogeneous information systems in a semi-decentralized environment. In this paper, we are primarily concerned with data base integration concepts and how they are resolved in a semi-decentralized environment. Basic techniques and principles concerning such issues as intermodel and interlanguage transformations can be found in Refs. 13 and 14.

4. THE PROPOSED ARCHITECTURE

It is our view that the object-oriented approach deserves to be investigated as the foundation of a distributed data base model on account of its ability to be used as the powerful substrate for describing both the structure, the semantics, the processing, and the pattern of communication of remote incompatible sources of information. It is intended that the general nature of the candidate DODM provides a sound basis for allowing DOODMSs to serve as intelligent information processing agents in heterogeneous data base environment.

In general, the basic functional requirements for a DOODMS is to appropriately integrate the heterogeneous components in order to create the illusion of a single uniform system. This can only be achieved by

mapping the data and processing components of the individual sites in the system into a unique, system-wide object space defined in terms of the DODM. A site attached to the network can, through its DOODMS, interact with the resources of the entire network as if it were a single object space, ignoring the location of the individual data and the structural composition of objects (as their attributes and methods may reside in different locations). The basic idea is to equip the DOODMS at each site with a layer of software that implements a common interface, using the object-oriented methodology to encapsulate this software. This approach involves defining object methods that implement messages whose semantics are mutually understandable by the various sites in the network.

The overall objective of the envisaged architecture is twofold [15]. The first objective is to provide a standard view of remote data bases stored under different DBMSs and to permit access to all individual component data bases through their original query language. Here, emphasis is placed on the need to support multiple interfaces for data languages which rely on diverse storage techniques. The second objective is to provide an integrated view of the multiple heterogeneous data bases in the network and to provide the capability to access data from these different but related data sources by means of a high-level o-o data language. The basic difference between the suggested architecture and that adopted by logically centralized data base systems like Multibase is that whereas Multibase forces all users to use a common query language, DAPLEX, with a global schema defined using the functional data model, users of the semi-decentralized architecture have the option of using either the query language available at their own local site and view the composite data base in terms of the locally supported data models or alternatively use the high-level o-o query language for this purpose. Obviously, the high level o-o query language, henceforth called the *system data language*, can also be used in place of the component data languages to satisfy the conventional needs of local users.

To cope with the intricacies of the co-existence of multiple data languages, we introduce two layers of functionality in the system data language: the *o-o application layer* which actually supports the system data language used by end-users when interacting with the composite system, and the *interlanguage transformation layer* which maps system data language constructs into equivalent ones in the query languages supported by the disparate information sources in the network. It is our thesis that multiple data language support is best provided by a common low-level kernel which offers the functionality that is required to achieve interlanguage transformations. Consequently, it is the purpose of the

transformation layer of the o-o system to completely insulate users from the primitive model of objects provided by the *interlanguage transformer*. The interlanguage transformer provides a shared object space of resilient objects which are actually the homogenized counterparts of the heterogeneous data constructs in the system. The objects created by the interlanguage transformer implement virtual objects which are resilient in the sense that they survive hardware crashes but are not persistent since they are created to meet the temporal needs of users when posing global queries and thus do not outlive sessions. The application layer is implemented on top of the interlanguage transformation layer and augments this model by providing such concepts as typing, message passing, and inheritance. This is actually the model seen by the end-users and exists at a more abstract level than the primitive model provided by the interlanguage transformer.

Distribution places a wide spectrum of requirements that must be fulfilled by the designer of the system data language. The data language must closely mirror the distributed computation; it must allow for the definition and manipulation of entities both at the local and distributed levels by providing efficient mechanisms for both intra- and inter-site communication. Moreover, the semantics of computation should be consistent and readily apparent in both the local and remote cases.

According to the foregoing, it can be understood that the basic elements of the DOODMS architecture comprise logical front-end extensions which appropriately enhance the operational features of the existing component DBMS to provide the required spectrum of functionality. These front-end extensions consist of the following interface modules:

1. The system language components, i.e., the system language server and the interlanguage transformer.
2. The metadata data modules which provide information concerning the distributed application, needed for a conflict-free analysis of the DCS into its constituent entities and properties.
3. The global transaction module which is responsible for the issues of query decomposition and execution plan generation as well as for global concurrency control and recovery.

The DOODMS resides in each site and is sensitive only to global requests (see Figure 1), having not to deal with the user actions that are strictly performed locally. This form of integration entails a set of stringent requirements stemming from the individualities and concerning the behavior of the coupled information sources. To fulfill this wide range of requirements, the DOODMS has to promote inter-site communication by guaranteeing efficiency while providing a high degree of site autonomy.

4.1 Heterogeneity and Integration Issues

With the diversity of existing data models and DBMSs, the techniques of homogenization and of distributed schema development become key issues. Any system dealing with distributed heterogeneous data models presents not only the problem of translating between heterogeneous data representations but also the additional predicament of defining a suitable unifying conceptual framework in which equivalence properties between diverse component can be meaningfully established. Accordingly, one should provide a single unifying view of the DCS, expressed in terms of the diverse conceptual data models, to give the illusion of a nondistributed homogeneous system.

To support heterogeneity, the DODM should provide the appropriate structural and operational properties that allow it to represent the structure and simulate the operations of the diverse data models in the system. Furthermore, the structural capabilities of the DODM should be associated with a well-defined set of operations and integrity/consistency preserving rules to guarantee the co-existence of diverse data models. By advocating a single unifying data model, we are offering a concise architectural framework for heterogeneous systems that can evolve into equivalent homogeneous systems. The o-o concept provides a framework for the incorporation of data conversion procedures (methods), required in inter-object communication, to be incorporated in implementation classes. Therefore, given an abstract class in the DCS level, they may exist several implementation methods, one for each heterogeneous information source, that need to be accessed through this specific abstract class. In this way, a uniform interface is provided for the implementation of integrated, distributed applications.

Global requests expressed in terms of more than one component data models are homogenized by transforming appropriately their structural and semantical substance into equivalent DODM structures. Subsequently, data base integration can be effectuated by defining a DCS materializing the collective application needs of users over the homogenized component schemas. Any directives for the resolution of semantic or other differences between the component data bases are also expressed at the DCS definition level. This process assumes that the component schemas are already homogenized, i.e., translated into the DODM, so that users are shielded from differences between the original component data models and languages.

The integration methodology in distributed information systems presumes the resolution of discrepancies between the component data bases [16]. In many cases, the conceptual schema constructs in one component

schema may, for reasons of compatibility, have to be changed to correspond in a one-to-one basis to equivalent schema constructs in other component data bases. We can discriminate between the following broad integration problem categories:

1. Incompatibilities in naming conventions, which typically involve either semantically equivalent data items named differently in the different component information sources, or semantically different data items which have the same name in the different component data bases.
2. Incompatibilities in the representation of the same data item in the different component data bases. For example, a data item may be defined as a character string in one component data base and an integer in another.
3. Incompatibilities in the data structures in the different component data bases caused mainly by using diverse data models. This is a homogenization problem which can be resolved by choosing the appropriate methodology to transform the component data models into the unifying data model, viz. the DODM.

As explained, the DODM provides the rich structural and semantical facilities to resolve the integration issues in the heterogeneous distributed data base environment. The DCS is composed by first forming *property equivalence classes*. These classes are formed by specifying the equivalence amongst the properties (attributes and methods) of the homogenized object classes[†] and establishes meaningful relationships between the *local objects* (i.e., homogenized objects) in the component data bases. The information concerning inter-object equivalence forms the basis for identifying and describing the object classes and relationships which are established between diverse component schemas.

The DCS is composed with the aid of the necessary methodology required to establish assertions concerning local object properties. With this methodology, inter-schema assertions are required for schema integration and are used to impose strict naming correspondences between the various local objects and properties of the component schemas to be integrated. To cope with this kind of predicament, information pertinent to conflicting elements must be explicitly specified to the schema integration tools in forms of assertion methods. This is an interactive process which is accomplished only once, at the *preintegration phase*, by a continuous dialogue with the data base designer. This means that some integration/consistency rules must be specified in form of assertion methods at the global level, guaranteeing, thus, an accurate semantic transformation of the conflict-

[†] We assume that each object emanates from a single class.

ing information at the component data base level. Inter-schema assertion methods specify also consistency and integrity constraints which are to be explicitly stated and subsequently enforced by the system. Such constraints may express application dependent rules for consistency among several data objects and relationships. Therefore, they enable the DODMS to assess the dependent changes that must be made when updates occur to certain objects or relationships. In general, constraints assist in defining and delimiting the behavior of the entire information system as the component information sources evolve.

During the process of schema integration, class hierarchies in the component data bases are merged into a single equivalence class (implemented as a virtual class encapsulating the operations which materialize it in terms of the component class objects) in the DCS. Such virtual classes are placed into a virtual-class lattice where some virtual classes represent generalizations of other virtual classes. The type lattice resulting from the integration of the virtual classes and inter-object relationships are merged to form the integrated DCS. This type of information is stored into a knowledge base, the metadata repository of the semi-decentralized architecture, which contains the information required to provide the user with a consistent data base view. This knowledge base is an o-o semantically enriched representation of the information required to merge two or more homogenized component schemas into a single DCS.

After the choice of the naming conventions and concepts at the DCS level, domains are provided for all local attributes, methods, object classes and inter-object relationships, and their substitutes at the DCS level are added to their original names. Operations on these information are obviously encapsulated within the equivalence classes in the class-lattice construct.

This method is in many respects superior to the conventional schema (view) mapping methodology. By allowing arbitrary procedures to be integrated into the DODM as object methods, the object-oriented approach provides a convenient way to provide increased functionality and view mapping capability. Although some portion of these facilities can be provided by more conventional approaches, they are often difficult to code and to handle, as conventional view mapping facilities can only be implemented by means of procedures unintelligibly scattered among the application programs. On the other hand, the uniform manipulation of data and metadata provides the means to support inferential capabilities so that view mapping decisions can be readily made on the contents of the metadata pertaining to the semantical and structural description of the component schemas. More importantly, with the aforementioned approach it is possible to allow for a smooth evolution of the DCS over

time to meet the ever changing information requirements of its users. Since the DCS is a generalization hierarchy (class lattice), schema changes essentially imply a possible reorganization of this generalization hierarchy.

DCS changes are propagated by means of a sequence of transactions of all of the sites in the network. Each transaction terminates when the DODMS of the destination source is instructed to perform a commit operation. During the transaction, the metadata object space is completely owned by the schema change session. The global commit operation has the effect of making the current state of metadata available to all sites in the network, reflecting therefore locally issued schema changes. In the face of potential hardware failures, commits are executed atomically, i.e., they either all run to completion or are not enforced at all. Consequently, the state of the shared metadata object space moves reliably in discrete steps from consistent state to another whenever a global commit is executed.

In retrospect, in distributed applications we cannot afford to have a static distributed conceptual schema for the component data bases in the network, nor can we foresee a system where changes at the global level are bestowed only through human intervention. Our position is that an information system environment should entail an evolvable DCS with such properties that guarantee its potential self-adjustment to changes affected by its views [17].

4.2 The Extended Schema Architecture

When integrating heterogeneous information sources that support diverse data representation models, one has to extend appropriately some traditional architectural concepts in the data base field. In fact the conventional three-level schema (ANSI/SPARC) architecture, motivated by the need for data independence and multiple view definition, must be augmented by introducing additional logical layers to meet the complex requirements imposed by the distributed data base architecture. These additional layers consist of the DCS together with its associated fragmentation schemas, which describe the logical structure and composition of information, respectively, in the entire data base network. In these additional layers, problems such as the identification of similar objects residing in more than one data bases have to be considered and solved.

The fragmented schemas comprise of a set of nonoverlapping virtual fragments representing the composition of the DCS from information stored in the component data bases. In other words, schema fragmentation performs the partitioning of the DCS over individual conceptual schemas on the remote sites of a distributed data base. Obviously, the fragmented schemas are derived from the DCS only after homogenization and conflict

resolution has been accomplished. Each fragment is a logical fraction of the DCS which has a corresponding physical image at one or several sites of the network. This mapping may be one to many, which implies that several physical images may correspond to the same fragment. Obviously, all of the physical images involved in this kind of mapping are semantically identical, i.e., they constitute copies of each other; therefore the mapping indicates that there exist replicated copies of critical information stored redundantly at more than one sites in the distributed information system. The objective of this methodology is to evaluate the utility of distributing objects into fragments both horizontally (collections of class instances, i.e., objects) and vertically (into collections of attributes and methods, i.e., object properties).

The implementation of the fragmentation schemas must also cater for the allocation of objects and properties found at the DCS level to corresponding physical constructs in their underlying component data bases. Accordingly, special system-defined allocation methods are used to map requests expressed in terms of abstractions at the DCS level, i.e., objects or messages, into concrete component data structures that can then be passed as parameters among the methods implementing the specified messages. These data structures serve as surrogates at the component data base object level and are passed at the fragmentation schemas as inputs or outputs among the methods that actually manipulate the state of an object.

Subsequently, the DOODMS determines, with the aid of fragmentation and allocation methods, the optimal allocation of fragments among the local data base schemas. Here, the optimal allocation of data should be determined on the basis of special optimization techniques or heuristic algorithms which ensure that the aggregate cost of transaction processing is minimized.

The issues of efficient handling of mappings between the DCS and the component schemas, as well as the secure communication and data exchange between the remote information sources, suggests the use of the universal extended schema architecture depicted in Figure 2. This extended schema supports implicitly reliability of operations in that each individual site would be able to function as a stand-alone site in case of a site/network failure. Subsequently, when the network becomes operational, all operations affecting the global state of the system are forwarded to their appropriate destination.

Figure 2 shows the extended schema as perceived from four interacting sites. Notice that a copy of the DCS resides in each site. It must be mentioned that the DCS is not a schema in the real sense; rather it is a virtual one. Virtuality lies in the fact that there exists no physical data that populate the DCS; rather, a specification is provided which describes how the DCS constructs

are materialized from data maintained by the individual component data base schemas. This means that the DCS contains the necessary metadata which support a consistent/coherent access to information originating from the remote data bases.

The prime implication of the previously described organization is that local data bases retain control over local operations while permitting adequate authority to be exercised by the DOODMS in order to coordinate the sharing and exchange of information between remote sites, and to ensure data compatibility and data consistency.

As previously explained, the overall architecture can be characterized as a semi-decentralized one, in contrast to another loosely coupled form of architecture known as *federated data base architecture*. Federated architectures [18] allow a loosely coupled union of the shared information by means of a collection of component data bases participating in the federation. Each component data base is associated with three schemas, each of them providing the users with views of information pertaining to the functionalities of their component data base. In particular, each component data base participating in the federation is supplied with a *private*, and *import*, and an *export* schema. The private schema describes locally contained information, whereas the import and export schemas describe the portions of a remote component that can be imported by this component, and the portions of this component that are allowed to be exported to other remote components. One can understand that federated data base architectures are a subcase of semi-decentralized architectures as import, export, and private schemas and their communication protocols are fully described and encompassed by the DCS. We believe that, for a very loose coupling of rather small data bases, a federated architecture may be an asset because of its simplicity. However, for very large data bases developed around complex schemas and requesting a rather complex communication pattern a semi-decentralized architecture can be of real use.

The purpose of the next section is to identify and summarize the properties of the high-level object used to describe the data in the semi-decentralized architecture. It is worth mentioning that due its nature the semi-decentralized architecture heavily depends upon the semantic and structural constructs offered by its underlying DODM.

5. HIGH-LEVEL O-O DATA MODELING CONCEPTS

As previously explained, the o-o paradigm offers the natural substrate for use when integrating disparate information sources. Viewing the data components to be integrated as objects allows a common design method-

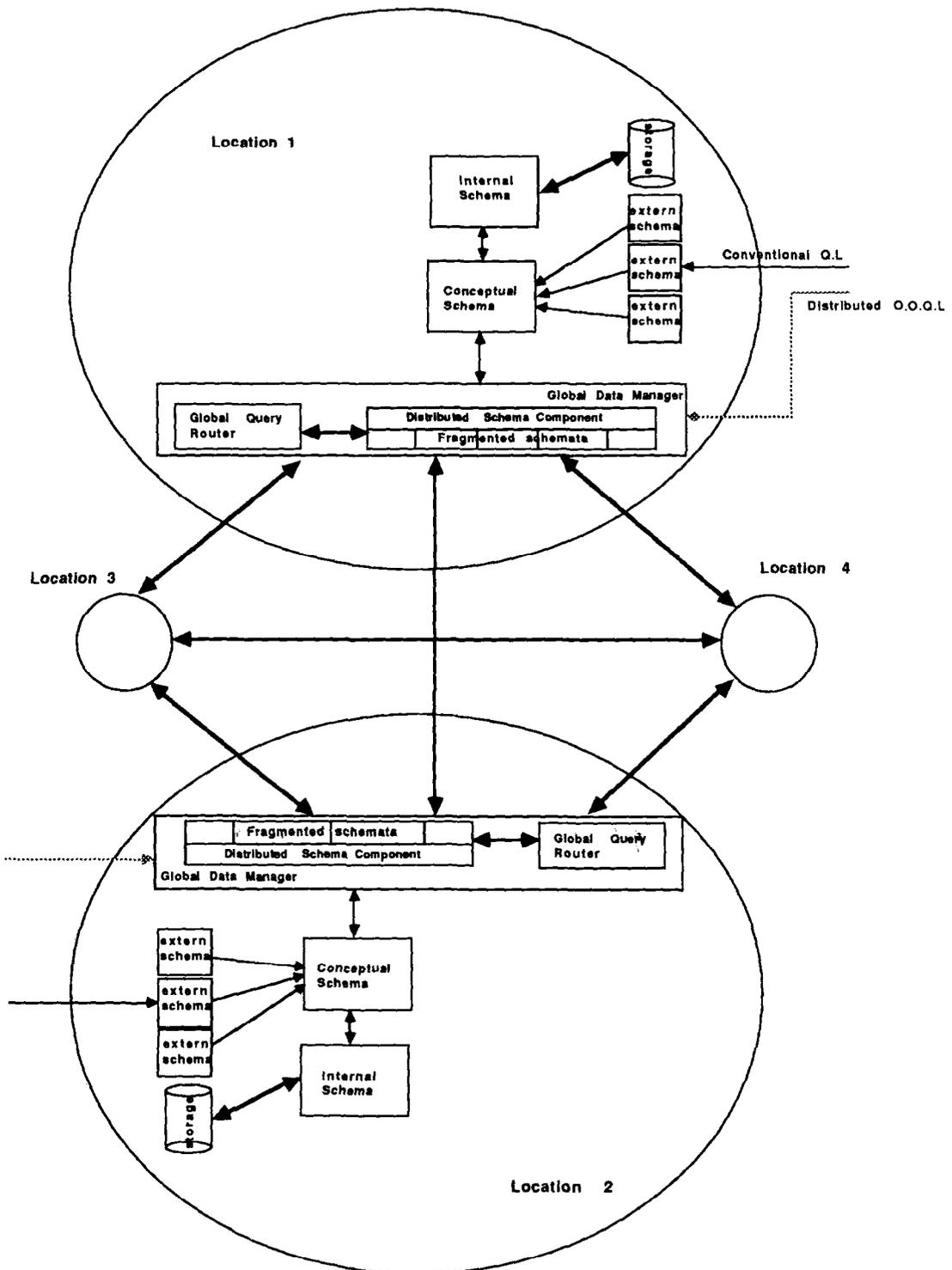


Figure 2. The semi-decentralized architecture involving four interacting sites.

ology to be applied to objects at all levels of granularity. With this approach we direct our attention to the definition of message protocols that are mutually understandable so that objects can communicate and ignore the syntactic differences in the data constructs of their underlying component data bases.

High-level o-o data models for semi-decentralized information sources must equip the system with a substantial degree of modeling power when compared with their conventional record-oriented or semantic (e.g., the functional data model in Multibase) data modeling counterparts. This additional modeling power entails the support of modeling *complex objects* (i.e., nested objects of arbitrary complexity), the ability to define relationships between these objects, and to organize classes of objects into an inheritance hierarchy. A concise entity in the data base should be modeled as a single object of arbitrary granularity and not as multiple tuples in multiple relations. Consequently, properties of entities need not assume simple data values, rather they can be other entities of arbitrary complexity.

The homogenized composite data base counterpart of a set of loosely coupled information sources should consist of collections of objects, the basic building blocks provided by the internal executional data model, on the basis of which the DCS is materialized.

The primary requirement for a high-level o-o data model for a semi-decentralized architecture is that individual entities in the application domain can survive as non-atomic entities. This comes in contradiction with conventional record-oriented data models (e.g., relational, network, hierarchical models), where the only non-atomic entities in a data base are the records themselves. In record-oriented models, all record constituents (i.e., attributes) are necessarily atomic entities (e.g., strings, integers, reals, and booleans), which draw their values from the corresponding predefined domains. On the other hand, o-o data models point decisively towards an efficient representation of non-atomic entities called "objects," and can thus view the data base as a collection of abstract objects, rather than a set of flat files.

Objects in a semi-decentralized architecture must attain two types: they must either be *primitive* or *non-primitive*. A primitive object is a non-decomposable atomic value, such as a string or integer. A nonprimitive object has an internal state which is made up of a collection of attributes. Attributes are given labels and thus their order of appearance in a nonprimitive object is immaterial. On the other hand, o-o data models[§] support the modeling of the behavior of real-world en-

tities and not only their structure. The state of an object is accessible only through its message interface which controls the behavior of objects. The behavior of an object is encapsulated in methods which comprise the code that manipulates or returns the state of an object. In an o-o data model, methods are organized along with the structural descriptions of objects. By contrast, in a conventional data base system a common procedure would typically reside at a file external to the data base, rather than being part of the data base.

The methods implementing a message can execute any number of data base queries or updates with the additional advantage of code reliability, as every operation that requires a specific data base operation (e.g., the debiting of an account) uses the very same method(s). Moreover, distributed applications are more concise as the sending of a single message may take the place of a series of conventional data base operations.

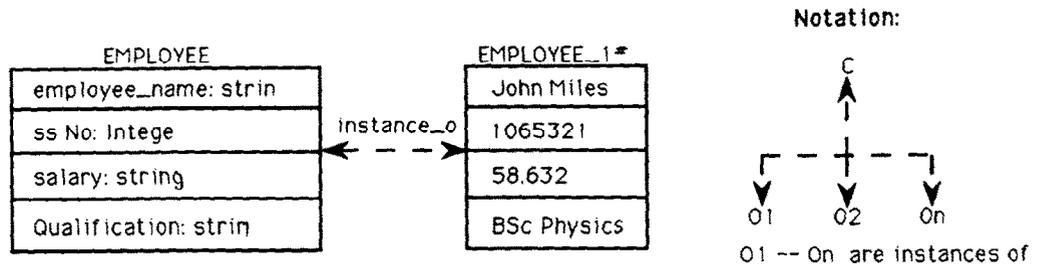
Object-oriented data modeling techniques provide for the concept of data abstraction where layers of abstraction may be used to develop a hierarchy of data. The topmost layer represents the most general view of data with all intermediate views progressively representing specialized views of data. Moving down the hierarchy reveals not only more specialized but also more complex views of data.

Furthermore, an o-o data model should guarantee the provision of powerful mechanisms for the formulation of integrity constraints (normally based on first-order predicate languages which isolate objects on the basis of their attribute values) and should also offer extensible metadata facilities to promote the transformation between different organizations of data into a common data representation. In Ref. 19, we describe how a high-level o-o data model can be used in a heterogeneous distributed data base system as an intermediary agent to bridge the semantical gap existing between the source (translated) and the target data models.

In addition to the notion of object, two other principles are highly desirable in managing the complexity of distributed data base design: the principle of *data abstraction* and the principle of *data localization*. The principle of data abstraction suggests the suppression of any irrelevant details in favor of emphasizing and representing more appropriate detail. Data abstraction refers to composition rules used to compose higher-level data objects from their constituents. The principle of data localization states that each property of an entity should be modeled as independently as possible (i.e., it should be localized); subsequently, all properties should be integrated appropriately to produce the overall design.

As previously suggested, high-level o-o modeling relates directly to analogous concepts developed for abstract data types. We can postulate five forms of ab-

[§] We use the unqualified term o-o data models to stand for high-level o-o data models in the rest of this paper.



straction and inter-object relationships useful in building such data hierarchies in distributed heterogeneous information environments: classification (or typing), aggregation (or type composition), generalization (or subtyping), association (or membership), and relativism[¶] (where an n-ary relation is interpreted as an entity). The purpose of these forms of abstraction and inter-object relationships is not only to provide a desired representation of the application, which closely mirrors the user's perception of the intended distributed application, but also to facilitate integration between the homogenized conceptual schemas of the local applications and the DCS. These forms of data abstraction are fundamental to the notion of o-o data modeling; thus, they are outlined here briefly.

The concept of *classification* collects objects (entities) in a data base having common properties (attributes and methods) to form object instances. Classification can be perceived as a simple form of abstraction in which a collection of objects with common properties is considered as a higher-level object-type, called a *class* or *type*. If an object possesses the set of properties defined by a certain type, then the former is an instance of the latter. In general, classification establishes an **instance-of** relationship between a generic object-type (e.g., Employee in Figure 3) in a schema and an object (e.g., John Miles in Figure 3) in the data base content.

When a collection of related objects combine to form a higher-level object, an *aggregation-abstraction* is defined between the higher-level object (the *aggregate object*) and the collection of its constituent or *component* objects. When considering the aggregate object, specific details of the constituent objects can be suppressed. In general, aggregation establishes an **is-part-of** relationship between the component objects and the aggregation objects. Loosely speaking, the whole (here, the instance of the higher-level object Employee) is the sum (aggregate) of its component parts (the related objects Name, Age, Address in Figure 4).

Many distributed applications require the ability to define and manipulate a set of objects as a single entity,

Figure 3. The concept of classification.

called a *complex object*. A complex object with a hierarchy of exclusive component objects is referred to as a *composite object* and the hierarchies of classes to which the components belong is referred to as the *composite object hierarchy*. A composite object is a special case of an aggregate object which adds the notion of dependency to the features of the **is-part-of** relationship. A dependent object is one whose existence depends on the existence of another object and is owed by exactly one such object [17].

Generalization refers to a mode of abstraction in which a set of similar objects, the *category objects*, is regarded as a generic object [20]. The generic object includes the common characteristics of the entire group of similar objects. The similar objects can then be referred to as in a generic manner through the higher-level objects; see Figure 5. In sum, generalization enables the coalescence of category objects into generic objects, in terms of common properties. The constituent objects are regarded as a *specialization* of the generic object. This establishes an **is-a** relationship between category objects and their corresponding generic object.

An **is-a** hierarchy may also be thought of as defining relationships between parent objects and their offsprings which are said to *inherit* all about their parents. In general, the term inheritance describes the sharing of information between adjacent levels in the abstraction hierarchy. The most general form of inheritance in distributed applications, however, is *multiple inheritance*. When multiple inheritance is employed, an object may be considered as being a part of a set of objects that is generalized by a more generic object. In this case, the hierarchy may resemble a *lattice* (an acyclic hierarchy of types), see Figure 5, in which an object descends from one or more parents.

The concepts of generalization and **is-a** relationships assume great importance for data base integration in multi-data base systems. Research conducted in the direction of distributed schema integration motivate for introducing different kinds of **is-a** relationships between objects [21]. For example, **is-a** relationships can be used to represent one or more possibly overlapping types of

[¶] Relativism is considered to be a property of both data abstraction and data localization.

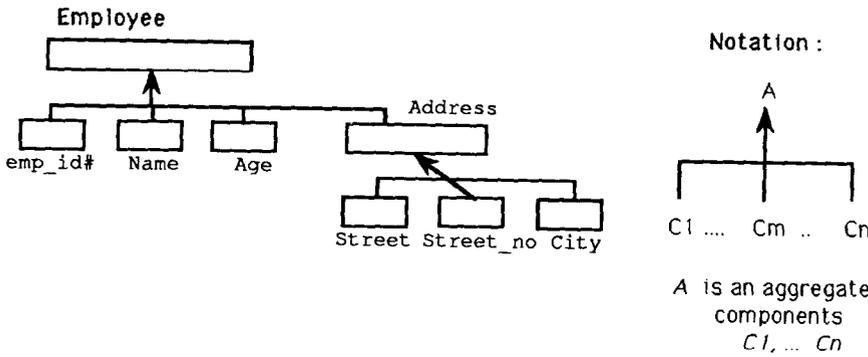


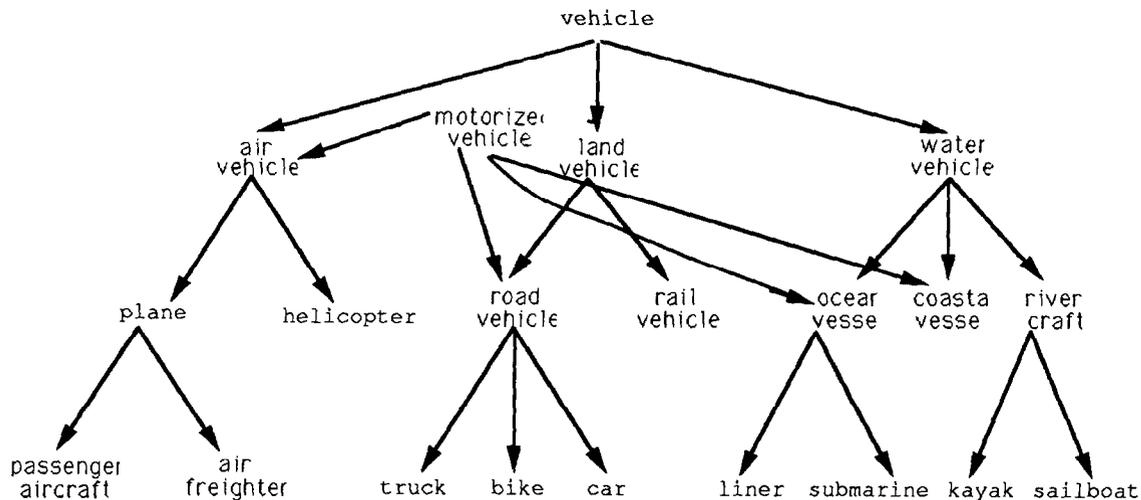
Figure 4. The concept of aggregation.

a supertype, or to construct a supertype from the union of subtypes already defined in component schemas.

Combining generalization and aggregation produces data abstraction hierarchies where a well-structured information model can evolve from successive abstraction applications. The information content at each level of the data hierarchy is controlled by appropriately choosing the abstraction method to apply when moving between levels of abstraction. Each successive level in the hierarchy will contain a more specific view of the system data than the preceding level.

Suppose, for example, that we wish to model the employees in a certain enterprise. Assume that the company employs only three different types of employees: drivers, engineers, and secretaries, as shown in Figure 6. Now, assume that we are interested in the affiliation of employee types with trade unions. We then form an abstract type called *affiliation* which is an aggregate of employee and trade union types, and this lead to the complex situation illustrated in Figure 7. Notice that in both Figures 6 and 7 higher-level abstract objects are shaded.

Figure 5. The concept of generalization.



Association is a form of abstraction in which a relationship between similar *member objects* is considered to be a higher-level *set object*. The details of a member object are suppressed and properties of the set object are emphasized. An instance of a set object can be decomposed into a set of instances of member objects. This represents the **member-of** relationship between a member and its corresponding set; see Figure 8.

It is important to distinguish between association and aggregation. Both are means of abstraction forms where a higher-level object is constructed from lower-level objects. With aggregation, a higher-level object is composed of a combination of two or more primitive or nonprimitive objects, whereas association selects a subset of already existing member objects from a prespecified class of similar objects in the data base.

Finally, the principle of *relativism* states that an object can be considered both independently and in terms of any relationship in which it participates. This suggests that a relationship between objects can be considered as an object itself. In supporting relativism, a given DCS supports both the principles of abstraction and localization. Furthermore, this kind of duality between objects and relationships allows for a uniform treatment of both data and metadata.

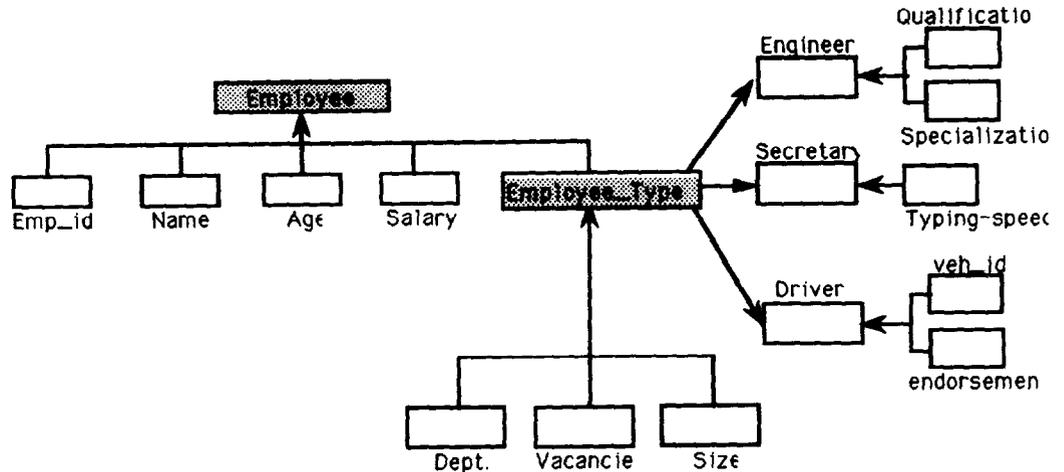


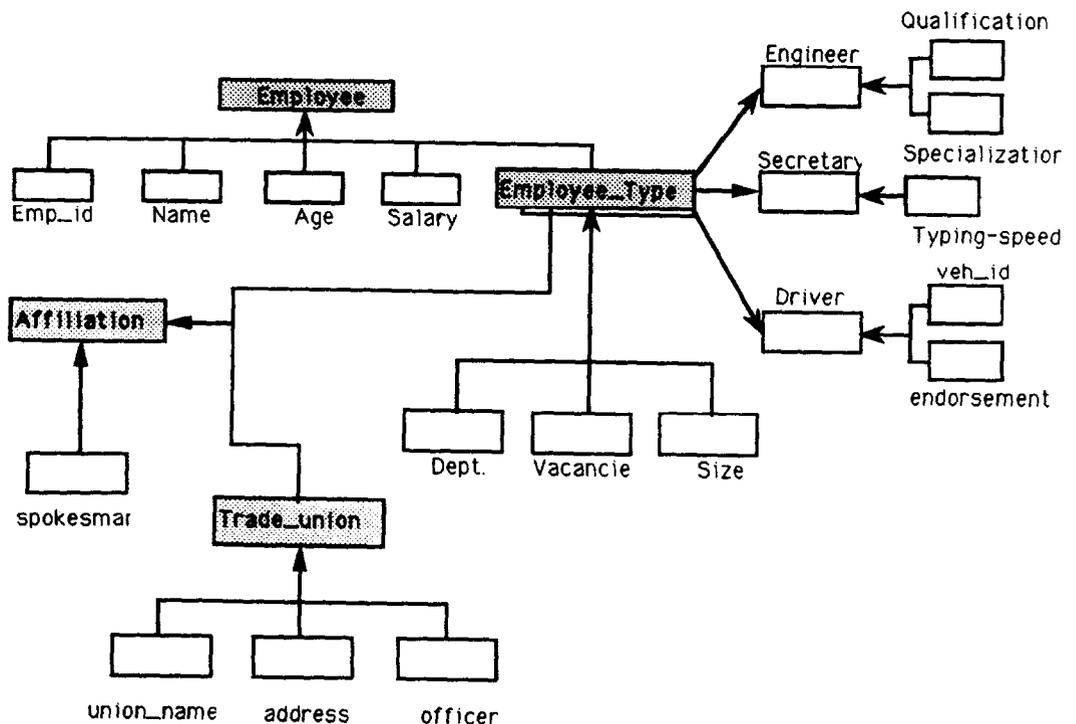
Figure 6. Combining generalization with aggregation: before.

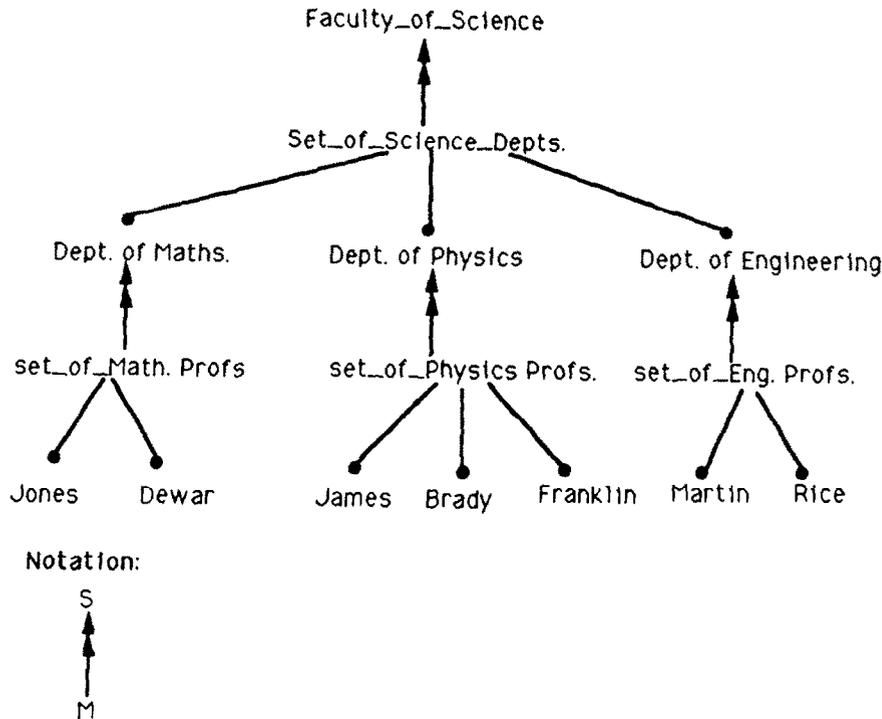
The above forms of data abstraction localization represent a cross-fertilization of ideas and concepts met in data base and AI research. The aforementioned concepts contribute to such techniques for structure modeling as composition/decomposition, association/membership, and generalization/specialization which all take advantage of property inheritance [22]. These techniques are highly desirable for modeling purposes at the user level while being extremely useful for the schema integration methodology. Finally, localization plays an important role in the design of abstraction hierarchies for distributed data base applications which is quite a formidable task. This is where the principle of localization assists in designing these hierarchies in a stepwise manner.

5.1 Distributed Data Language Properties

As previously explained, the DODM should form the basis for the realization of high-level data manipulation languages in which the data manipulation facilities are integrated neatly with general-purpose computation facilities. In this sense, distributed data base applications require the existence of a computationally complete data base programming language (such as Rigel [23], or Gemstone [24]) and not just a data manipulation language. The data language should be computationally complete in the sense that it can naturally support all of

Figure 7. Combining generalization with aggregation: after.





S is a set of members from the class M

Figure 8. The concept of association.

the computation load required in a distributed application.

Completeness helps to avoid the problem of *impedance mismatch*, arising when information should pass between two languages that are both structurally and semantically wide apart (such as an SQL-like declarative data sublanguage and an imperative general-purpose programming language, e.g., PL/I into which SQL declarations are embedded). This guarantees the provision of a self-contained programming environment while alleviating the obstacle of impedance mismatch between the programming language used to develop an application and the data manipulation language used to access the data base [25]. The benefits which accrue from this approach are immense:

1. Type checking can now be performed with ease in contrast with the conventional approach, where there always exists the burden of performing type checking across the language boundaries.
2. Constraints are expressed in terms of the data model, and are not any more represented as procedures written in the application programming language.
3. Efficiency and speed of execution are improved as no trade-off between compilation and interpretation has to be considered.
4. Finally, the inaesthetic combination of procedural

query languages with general-purpose procedural languages disappears leaving in its place a uniform programming interface at the disposal of its user/application programmer.

These are in fact very appealing properties because amongst others they guarantee that the metadata facilities can be uniformly accessed by the system data language and have enhanced structural and semantical capabilities when compared with their conventional counterparts which are endowed with limited and of static nature structural capabilities. The idea is that metadata repositories are asked to play a more dynamic and active role in the collective management of component data bases. In fact, one should not be able to distinguish between data and metadata; there should be a continuous spectrum of objects ranging from data base object instances to object instances representing knowledge. This approach is similar to the one adopted for knowledge bases where data and metadata are completely integrated and thus made co-resident in the same "knowledge base" [26]. Actually, the provision of objects and a complete programming environment at the global level guarantees that metadata can be queried and manipulated in the same manner as application data.

CONCLUSION

In this paper, we have outlined the architectural requirements for a heterogeneous DOODMS that copes with the

intricacies of information distribution across a network of geographically dispersed data bases. We have argued in favor of the introduction of semantically enhanced o-o modeling facilities which are in a position to manipulate information about the required application and can thus contribute to the retrieval of information which need to be deduced from explicit facts and cognition about the collective application domain.

Although there currently exists no prototype version of the semi-decentralized architecture, it is nevertheless possible to assess how the original objectives are satisfied by the topology described herein. Moreover, it is also possible to understand that the architecture is an "open" one, meaning that it can be expanded by the introduction of new components, if necessary, or refined by appropriately adjusting the functionality of noncompletely successful components. It is important to note that this kind of modular approach to distributed data base management is in accordance with most modern approaches to the engineering of large and complex software systems. Furthermore, the explicit control of information through the use of opaque or semi-opaque interfaces, as suggested herein, facilitates the design and construction of reliable and flexible systems.

ACKNOWLEDGMENTS

We are deeply indebted to the anonymous referees who instigated major improvements in an earlier version of this paper.

REFERENCES

1. P. M. Stocker, A Complete Specification for a Model Independent Heterogeneous Distributed Database Systems, in *Conference in Distributed Data Base Systems*, Commission of European Communities, Brussels, 1987.
2. M. P. Atkinson et al., The PROTEUS Distributed Data Base System, *Proceedings of the Third British National Conference on Data Bases*, 1984.
3. M. Mannino and C. Karle, An Extension of the General Entity Manipulator Language for Global View Definition, *Data and Knowledge Engineering*, 2(1), 305-326 (1986).
4. M. P. Papazoglou and C. Hoffmann, Towards Versatile Object-Oriented Query Languages, *IEEE Workshop on LFA*, 99-103 (1987).
5. J. Smith et al., Multibase-Integrating Heterogeneous Distributed Database Systems, in *Proceedings AFIPS NCC*, pp. 487-499, 1981.
6. Y. Brietbart and L. Tieman, ADDS-Heterogeneous Distributed Data Base System, *Conference on Distributed Data Sharing Systems*, 7-23 (1985).
7. D. Bril and M. Templeton, Distributed Query Processing Strategies, in MERMAID, a Front-End to Data Management Systems, *Conference on Data Engineering*, pp. 211-218, 1984.
8. Y. Brietbart, P. Olson, and G. R. Thompson, Data Base Integration in a Distributed Heterogeneous Data Base System, *Conference on Data Engineering*, 301-310 (1986).
9. F. E. Codd, Recent Investigations in Relational Data Base Systems, in *IFIP Congress 1974*, pp. 1017-1021, 1974.
10. W. Kent, Limitations of Record-Based Information Models, *ACM Trans. Data Bases*, 4(1), 107-131 (1979).
11. M. Hammer and D. McLeod, Data Base Description with SDM: A Semantic Data Base Model, *ACM Trans. Data Bases*, 6(3), 351-386 (1981).
12. R. Hull and R. King, Semantic Data Base Modeling: Survey, Applications, and Research Issues, *Comput. Surv.* 19(3), 201-260 (1987).
13. L. Marinos, M. P. Papazoglou, and M. Norrie, Towards the Design of an Integrated Environment for Distributed Data Bases, in *Proceedings of the International Conference on Parallel Processing and Applications*, pp. 349-358, L'Aquila, 1987.
14. S. Demurjian and D. Hsiao, Towards a Better Understanding of Data Models Through the Multilingual Data Base System, *IEEE Trans. Software Eng.*, SE-14(7), 946-956 (1988).
15. L. Marinos and M. P. Papazoglou, Integrating Heterogeneous Data, in *Proceedings of the EURINFO'88 Conference*, pp. 757-757, Athens, 1988.
16. S. Navathe, R. ElMasri, and J. Larson, Integrating User Views in Data Base Design, *IEEE Computer*, 19(1), 50-62 (1986).
17. J. Banerjee and et al., Data Model Issues for Object-Oriented Applications, *ACM Trans. Office Inform. Syst.* 5(1), 3-26 (1987).
18. D. McLeod and D. Heiminger, A Federated Architecture for Data Base Systems, *Proc. AFIPS NCC*, pp. 283-289, 1980.
19. L. Marinos and M. P. Papazoglou, Enhancing Intermodel Transformations in a Distributed Object-Oriented Data Base System, in *Proceedings of the EUROMICRO'88 Conference*, pp. 249-256, Zurich, 1988.
20. J. M. Smith and D. C. P. Smith, Data Base Abstractions: Aggregation and Generalization, *ACM Trans. Data Bases*, 2(2), 105-133 (1977).
21. U. Dayal and K. Y. Hwang, View Definition and Generalization for Data Base Integration in a Multi-Data Base System, *IEEE Trans. Software Eng.* SE-10(6), 628-645 (1984).
22. M. Brodie, The Design and Specification of Database Transactions, *On Conceptual Modelling: Perspectives from Artificial Intelligence*, Springer Verlag, 1984.
23. L. Rowe and K. Shoens, Data Abstraction Views and Updates in Rigel, *SIGMOD Conference 79*, 491-503 (1979).
24. D. Maier and J. Stein, Development of an Object-Oriented Dbms. *OOPSLA 86 Proceedings*, pp. 472-482, Sept. 1986.
25. P. Buneman and M. Atkinson, Inheritance and Persistence in Data Base Programming Languages, *SIGMOD Record*, 15(2), 4-15 (1986).
26. C. Zaniolo et al., *Object-Oriented Database Systems and Knowledge Systems, 1st Workshop on Expert Database Systems*, Benjamin/Cummings Publ. Co., 1986.