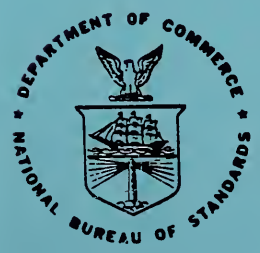
NBSIR 87-3529

The Use of Artificial Intelligence Programming Techniques for Communication between Incompatible Building Information Systems

William F. Danner

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
National Engineering Laboratory
Center for Building Technology
Gaithersburg, MD 20899

April 1987



U.S. DEPARTMENT OF COMMERCE
NATIONAL BUREAU OF STANDARDS

NBSIR 87-3529

THE USE OF ARTIFICIAL INTELLIGENCE PROGRAMMING TECHNIQUES FOR COMMUNICATION BETWEEN INCOMPATIBLE BUILDING INFORMATION SYSTEMS

William F. Danner

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
National Engineering Laboratory
Center for Building Technology
Gaithersburg, MD 20899

April 1987

U.S. DEPARTMENT OF COMMERCE, Malcolm Baldrige, *Secretary*
NATIONAL BUREAU OF STANDARDS, Ernest Ambler, *Director*

Abstract

A communication capability between incompatible information systems is presented. The purpose of the research reported here has been to develop an interface based on (1) a format for the exchange of knowledge needed by each system to understand the other, and (2) a format for the exchange of information in the context of that knowledge. Particular emphasis has been placed on developing protocols supporting the transfer of analytical data. These data are seen as comprising not only facts but also the semantics associated with those facts.

Two artificial intelligence programming techniques have been employed: (1) frame-based knowledge representation, and (2) object-oriented programming capabilities as an integral part of the frame-based representation. These techniques make self-descriptive formats possible that provide for a virtual extension of an information management system. Such an extension provides access to information without requiring a detailed understanding of specific system operations.

Keywords: artificial intelligence, buildings; communication; data; database; database management; information; information system; protocol.

Table of Contents

List of Tables

List of Figures

1. Introduction

Participants in the building process are not able to fully integrate their efforts because of incompatibilities between the computer systems they use to assist them in decision making. The design and construction phases of a building involve many interdependent processes and decisions among architects, engineers, and contractors that could benefit from the ability to communicate between dissimilar systems. Currently, the amount of effort required to understand one another's systems in order to share information and processing capabilities is prohibitive.

The scope of the research described in this report derives directly from the particular needs of the building industry but not limited to that industry. Participants in the building process have at least two quite distinct but related interests regarding communication between heterogeneous systems. One is the development of comprehensive integrated building information systems. This long term interest is exemplified by discussions within the Architecture Engineering and Construction Committee of the Initial Graphics Exchange Specification Organization concerning the next generation of data exchange specifications [1]. The second related interest is in the ability to establish ad hoc communication to serve the immediate needs of particular participants on a particular project.

Such ad hoc communication may involve, in the extreme case, a pair of project participants that have never worked together before nor have plans to work together on any future project. The information they wish to exchange may be a very small subset of the information contained in their respective systems. The development of an interface capability for use in the building industry must accommodate both short term needs for communication as well as the needs for comprehensive integrated systems.

This report presents an initial step toward the integration of incompatible systems. It provides a way to exchange essential knowledge about how a computer system organizes its information. It also provides a way to access information without knowing the operational details of the system. The approach taken employs techniques available from the field of artificial intelligence for representing and using knowledge. Protocols have been developed that establish a modular interface capability between incompatible systems. The modularity of the developed protocols allows for an adaptive interface capability responsive to the needs of the building community as well as to information systems generally.

The testbed environment used during the course of the research is shown in Figure 1. During the initial phase, a multitasking workstation and a micro-computer were used employing file transfer communications software over an RS/232 link.

Subsequent work has included a LISP machine network. Protocols providing a transparent interface for the transfer of information and cooperation between concurrently operating applications have been implemented.

In order to provide a context for the discussion of the interface protocols, this report begins with an assessment of underlying causes of system incompatibility in Section 2. This is followed by a discussion of approaches to developing communication interfaces between incompatible systems in Section 3. Section 4 presents the modular approach to representing how information is organized within a system as well as representing information consistent with that organization. The modular representation is extended in Section 5 to include knowledge about accessing information. Section 6 then describes the implementation of a modular knowledge-based interface. The final section presents conclusions drawn from experience with the interface.

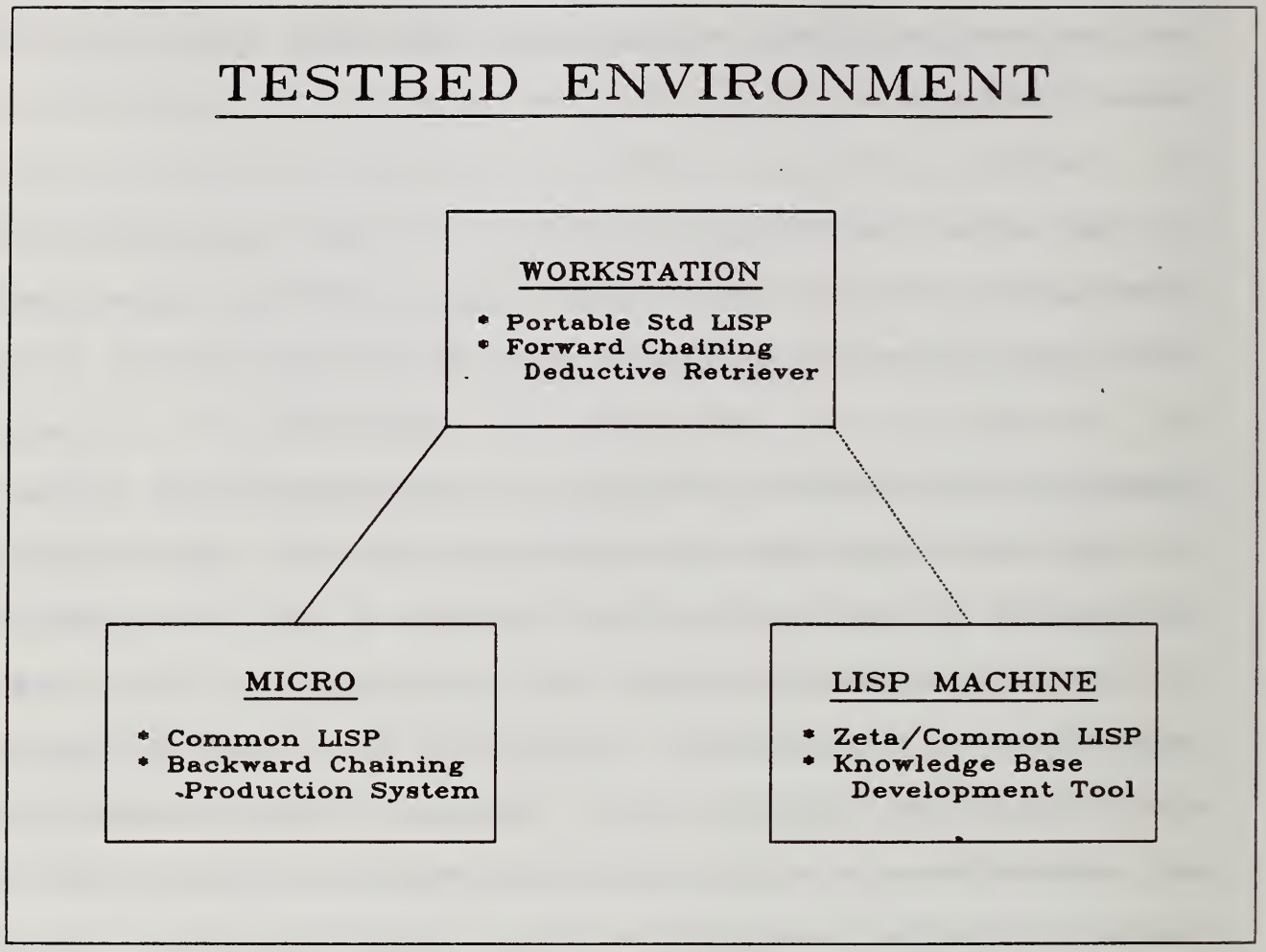


Figure 1. Testbed Environment

2. Incompatibility between Information Systems

The American National Standards Institute (ANSI) Committee on Computers and Information Processing (X3) Standards Planning and Requirements Committee (SPARC) framework for database management systems provides a useful context in which to discuss incompatible information systems [2]. A system's architecture can be thought of in terms of three levels. The external level is concerned with those views of information resulting from user-specific applications. The internal level is concerned with the way information is actually stored. Between these two levels is the conceptual level which corresponds to a common view of information underlying all external level applications.

The conceptual view of information (Figure 2) reflects the perspective provided by a specified logical organization, the conceptual schema. It is accessed using conceptual level operations in conjunction with mappings between levels. Incompatibility between information systems results from heterogeneity among the respective elements that provide the conceptual views of the information (i.e., the conceptual schemata and operations).

Conceptual level schemata and operations are developed in accordance with a particular theoretical data model. The data model specifies the structures used to develop the conceptual schema, as well as the operations that are allowed on those

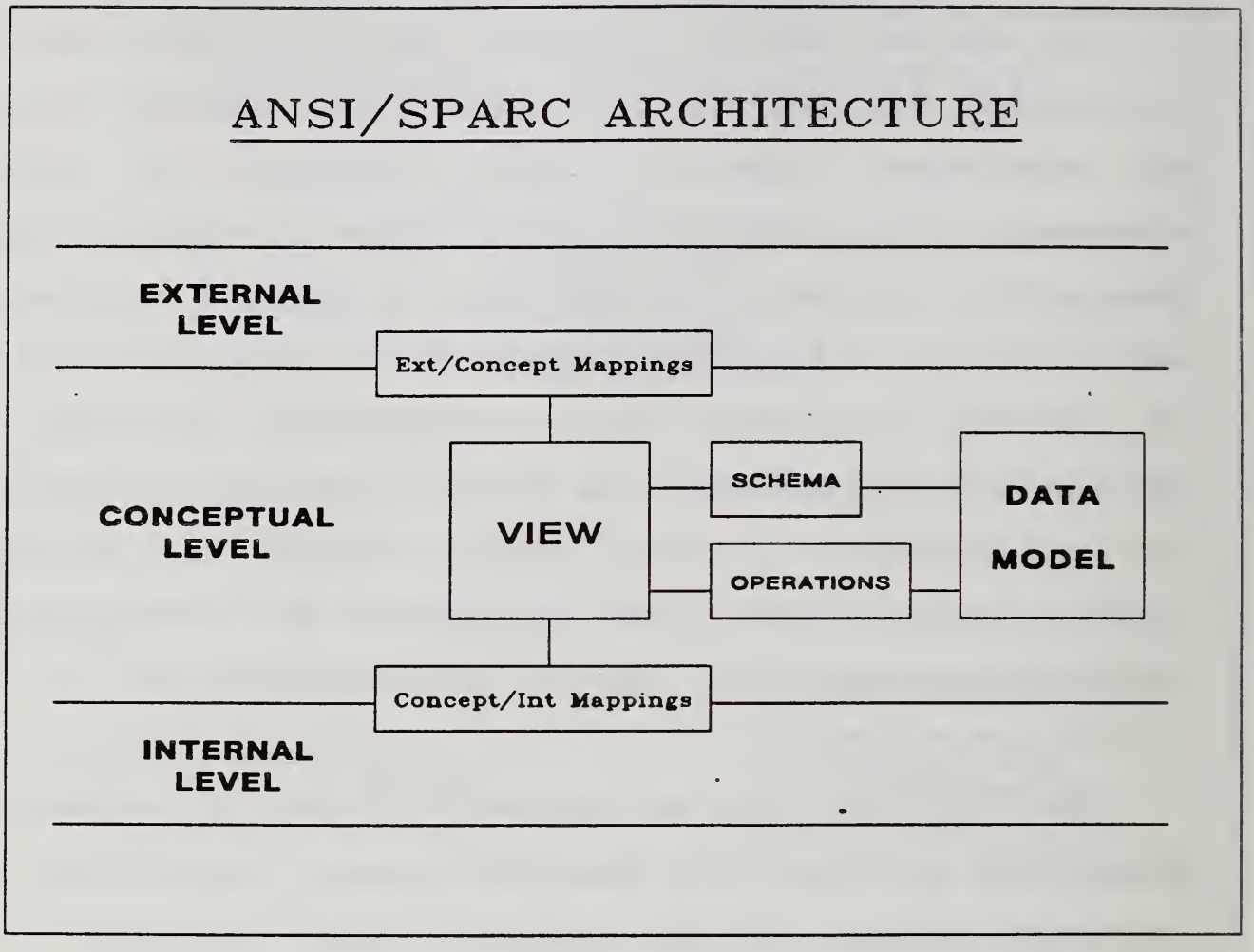


Figure 2. A Representation of the Levels of System Architecture Suggested by ANSI/SPARC.

structures. Therefore, a discussion of heterogeneity between information systems necessarily includes the topic of modeling.

2.1 The Underlying Theoretical Models

The field of information management, particularly in the area of modern database systems, has changed radically in recent years. Primitive file models first gave way to hierarchic, network, and relational models. These classical models differ predominantly in terms of the structures they use to model the organization of information. The structures in turn have implications for the ways in which information can be manipulated. As database systems became more sophisticated and knowledge-based systems began to appear, the classical models were extended to include ever increasing amounts of information about information (meta information). This trend has resulted in the development of what are often called semantic data models.

New models continue to appear, some representing considerable departures from the three classical approaches [3]. All however, are attempting to provide tools for dealing with the organization of information. Though they may differ greatly in their approaches, a common element remains, the underlying discipline-specific logical organization of information they are used to represent. Translation between models entails developing alternative representations of the same logical organization.

The incompatibility between information systems which results from dissimilar data models can be addressed by providing mappings between alternative representations. Translation must therefore include not only the information itself but also the semantics associated with that information if the logical organization of the information is to be understood.

2.2 The Logical Organization of Information and its Representation (Schema)

The logical organization of information derives predominantly from the discipline using the information. The tool (i.e., theoretical data model) used to model the information is selected on the basis of the appropriateness of the tool for the particular information, the ease with which the tool can implement an adequate representation, and the data manipulation capabilities that are to be provided.

The result of using a given theoretical data model is a conceptual schema of the system. It is a representation of the logical organization of information in terms of the structures that are provided by the modeling tool. A major impediment to integrating dissimilar information systems results from the fact that each system can operate only in the context of its own schema. For meaningful communication to take place, an interface

must be developed between schemata. The interface must also be able to deal with differences in the way each system manipulates information.

2.3 The Operations used to Manipulate Information

The operations used to manipulate information within a computer system may adhere in varying degrees to manipulation languages that have developed in conjunction with theoretical data models. The Standard Query Language (SQL) developed for use with relational databases [4] is an example of attempts at conformity. However, differences between implementations of SQL have appeared even within the limited context of relational databases. Being faced with totally different data manipulation languages makes integration of systems even more problematic.

An interface between incompatible systems must therefore include a means of translating requests into the language being used by the system supplying information. It must also provide for translation of the response into a form that is meaningful to the requesting system. Two approaches to these requirements, as well as the resolution of incompatibilities between schemata will be discussed in the following section.

3. Integrating Heterogeneous Information Systems

In order for communication to take place between incompatible information systems, there must be a resolution of schematic and operational differences. The resolution of these differences is often subsumed under technology for distributed heterogeneous information management. As the name implies, one function of this technology is to provide an interface that not only allows communication but also manages information at various locations. The most common approach to such an interface views each system as an independent local component of an all encompassing global system.

3.1 The Global System Approach

Global system interfaces have been described as requiring three major elements [5]: a global data manager, a distributed transaction manager, and structured-data transfer protocols. Though the names of such elements are different among the various interfaces in the literature, the concepts are quite similar.

The global data manager is responsible for input and output operations. In this capacity it makes use of an integrated global schema (Figure 3). The integrated global schema is analogous to the conceptual schema of the ANSI/SPARC framework

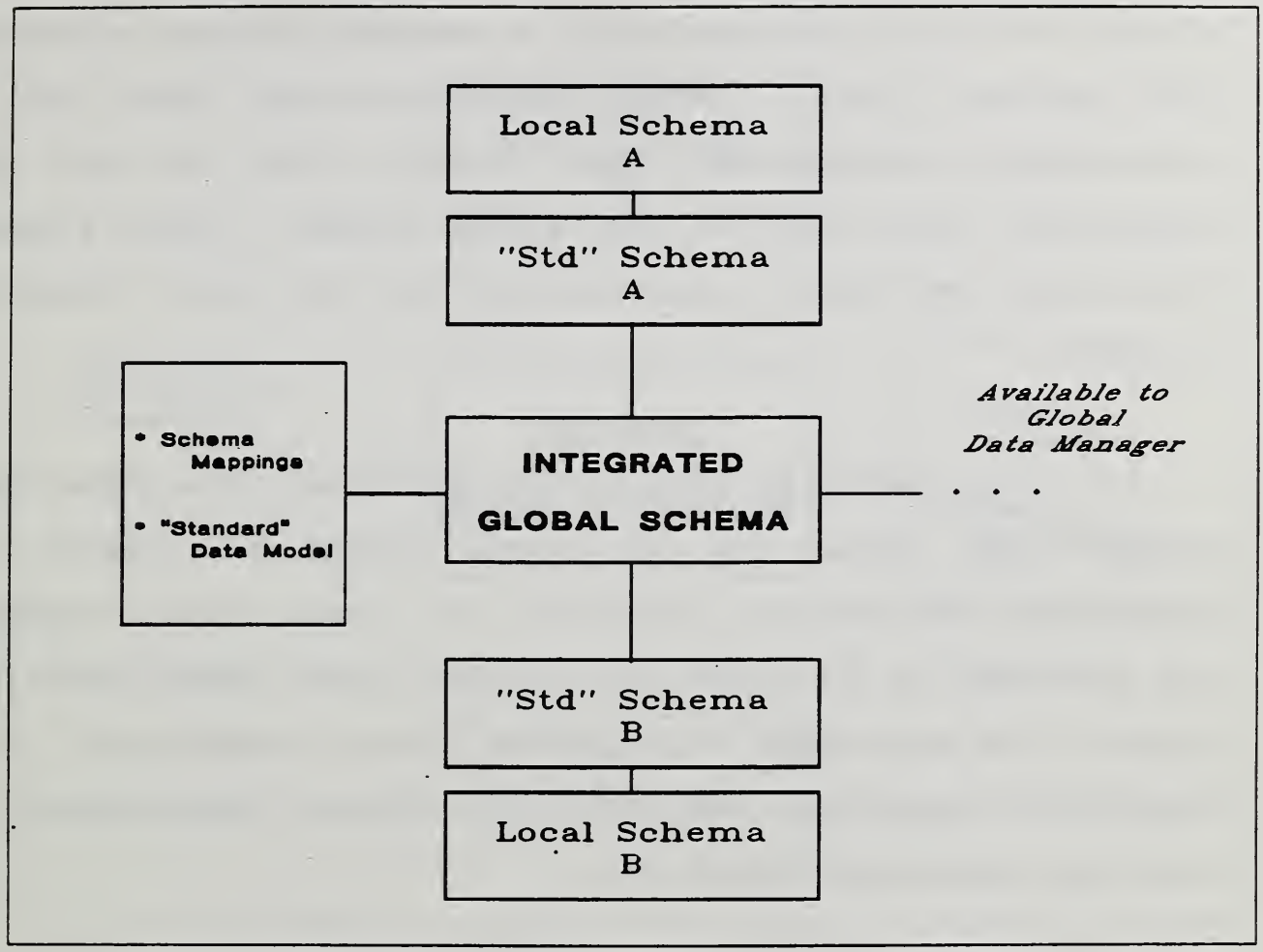


Figure 3. An Integrated Global Schema.

with the local schemata being analogous to external schemata. The global data manager also has access to mappings between global and local views analogous to mappings between conceptual and external views. Actual implementations, make use of intermediary standardized local schemata that use the same theoretical data model as the global schema. These elements facilitate the "global understanding" of the local conceptual schemata.

In its capacity to handle input and output, the global data manager also makes use of global operations (Figure 4). Translation of queries (operations) and local system responses are performed by the global data manager using capabilities that provide the equivalent of mappings between operations. This translation capability facilitates the "global understanding" of the local conceptual operations.

The data transaction manager, is responsible for consistency maintenance, including concurrency and recovery control. It is a necessary interface element for extending a distributed system from one capable of communication to one that is integrated. This element of global system interfaces is still largely at a research stage.

The structured-data transfer protocols are responsible, in combination with the global data manager, for the preservation of

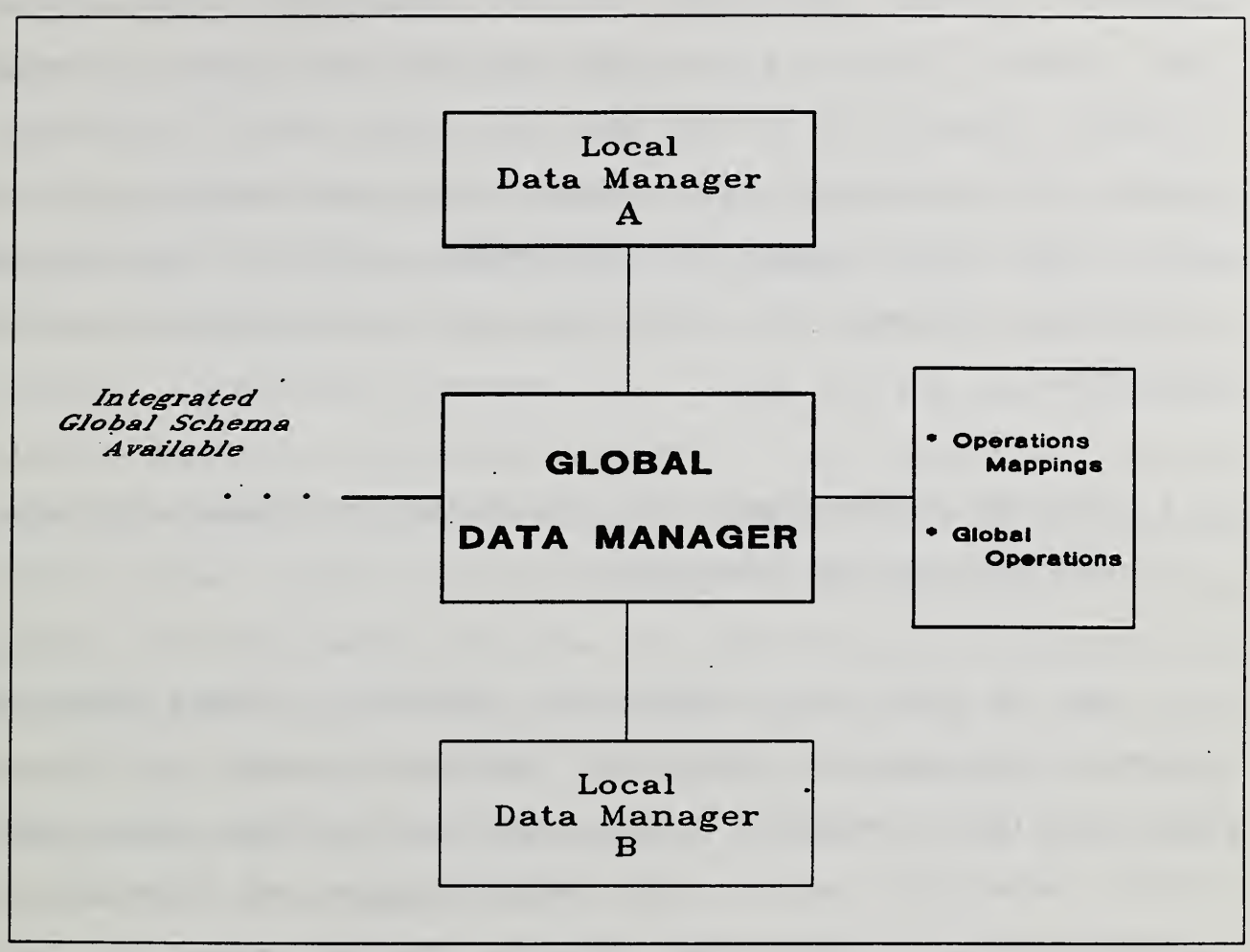


Figure 4. A Global Data Manager

data semantics. These protocols address issues of query and data transfer between components of the integrated global system. Data transfer protocols have not received much direct attention to date, largely due to the fact that each effort to develop a global system interface has proceeded as a comprehensive package, with its own internally consistent methods for passing information between its constituent parts and maintaining system semantics.

3.2 Information Management and Artificial Intelligence Efforts in Global System Development

One of the first efforts to develop a global interface between heterogeneous information systems appeared in 1978 [6]. Adiba and Portal proposed COSYS (cooperative systems) that used a binary relational model. In 1980 Cardenas and Pirahesh [7] presented their Distributed Database Testbed based on Chen's Entity-Relationship Model [8]. Smith et. al. [9] developed Multibase in 1981 using Shipman's Functional Model [10]. Later research into the use of global systems was carried out for the United States Air Force using a representation developed specifically for their Integrated Computer Aided Manufacturing projects [11]. More recently the Integrated Manufacturing Distributed Database Administration System [12] developed for the Automated Manufacturing Research Facility at the National Bureau of Standards has used Su's Semantic Association Model [13]. And

in a departure from the other efforts, Rehak and Howard [14] have proposed a knowledge-based global interface that is a current topic of research at Carnegie-Mellon University.

The global interfaces listed above constitute a great deal of research and development effort. Most have achieved some success at providing a communication link between heterogeneous systems. Providing integration of those systems has been more elusive however. One striking aspect of the above list is that each has chosen a different theoretical data model to develop its global schema. Each information system being connected to such a global interface must also have its local schema translated into a standardized local schema that uses the theoretical data model chosen for that interface. The global interface must then incorporate all the meta information of each of its constituent systems into its single global schema.

A similar situation exits for the operations aspect of the global system interface. Global operations are developed which act as an intermediary between requests and local operations. The knowledge about each new local system must be incorporated into global subsystems that perform such functions as schema analysis, query decomposition, translation, plan generation, and results integration [5] in order for communication to take place at the operations level.

Complex global system interfaces will undoubtedly one day provide not only communication between incompatible systems but also integration. Research on such systems is a continuing process. However, two observations can be made. The first is that though the potential for a global system having complete understanding of, access to, and ultimately control over all the information in distributed local systems is desirable, most needs for communication between incompatible systems are much less demanding. This is particularly true in the building industry. The second is that since each of the described global system interfaces uses a different theoretical data model and a different global operations capability, the problem of heterogeneity has not necessarily been solved. Rather it has been shifted to a higher, more complex level.

These observations have quite different implications depending on the purpose for which the interface is being used. In the case of a permanent fully integrated system a global approach is desirable. This is particularly true when demands for communication outside the global environment are relatively limited and of a predictable nature. However, if needs for communication are continually changing or if the required information is a limited subset of the total information available, a more adaptable approach is suggested.

3.3 The Role of Application Layer Protocols

Application layer protocols as used here refer to protocols that serve to establish a virtual communication path between heterogeneous distributed systems (Figure 5). They are at the seventh layer of the Open Systems Reference Model [15] of the International Organization for Standardization (ISO OSI Reference Model). Application layer protocols can be used to define standard canonical formats for representing both the schemata and the operations necessary for information access. They can also be used to define formats for transmitting requests and the information that constitutes a reply. A modular approach to representing system semantics both in terms of meta information and information has led to the development of Modular Semantic Application Layer Protocols (MS-ALPs).

The format that is used to represent the schematic and operational knowledge is the Meta Information Format. Application of this format to a given system results in a standardized Meta Information Representation (MIR) as depicted in Figure 6. An MIR is comprised of modules each of which contains knowledge about particular information stored in an information system. Knowledge about the information, how it fits into the overall organization of information, and methods for accessing that particular information are present. Knowledge about system operations is not, however, represented in a form that is

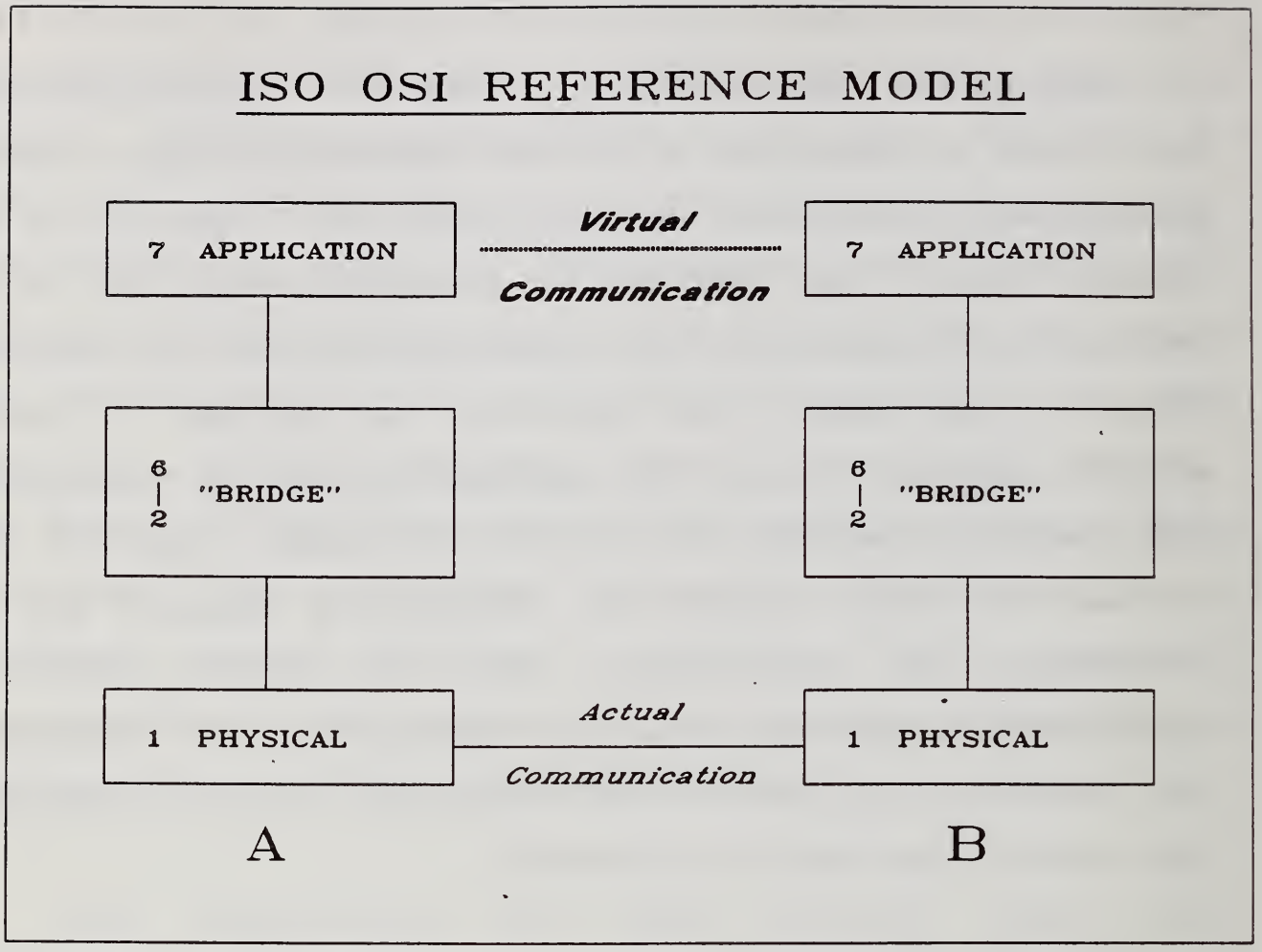


Figure 5. Simplified Representation of Layers 1-7 of the ISO OSI Reference Model.

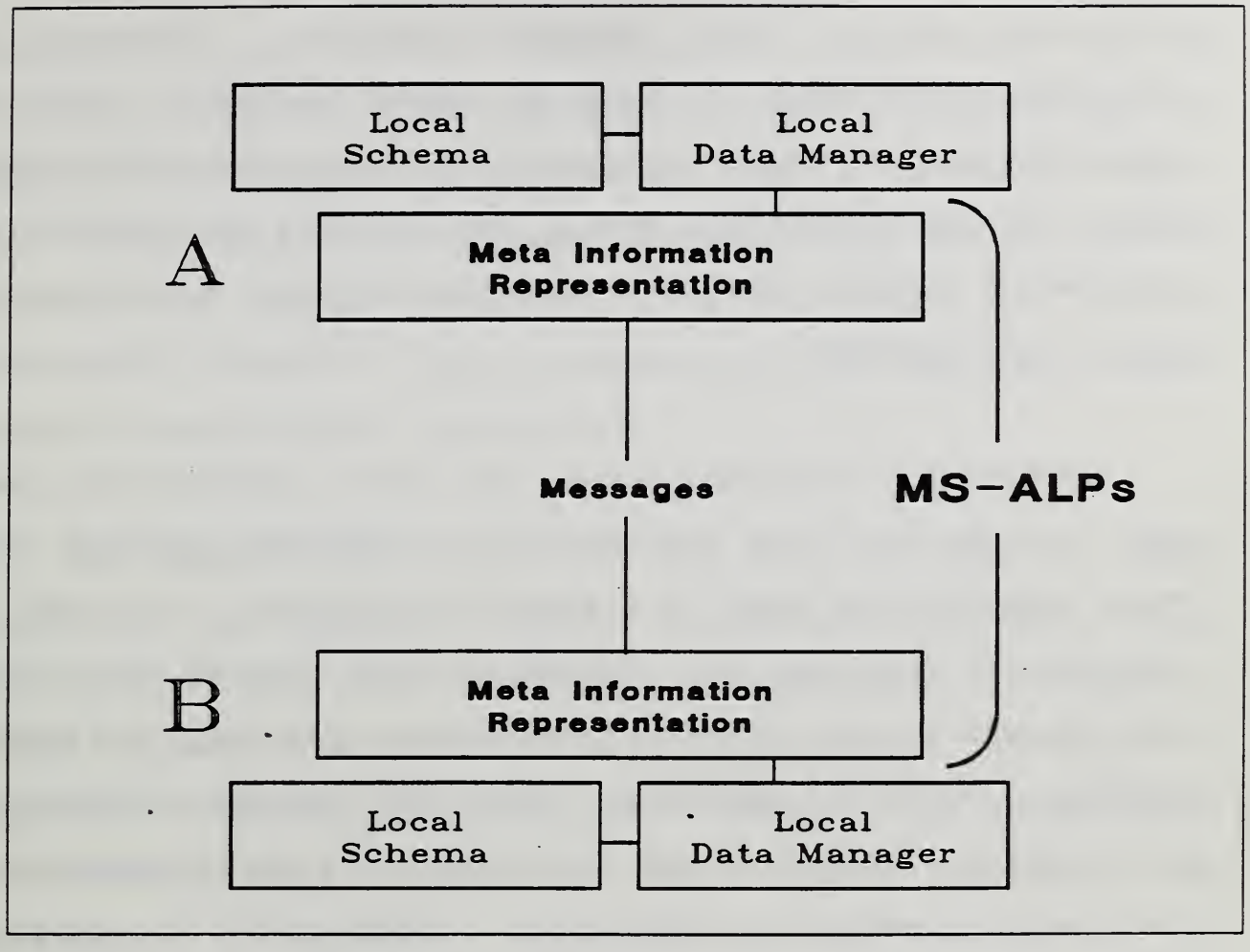


Figure 6. The MS-ALPs Approach.

necessarily understandable by remote users. Rather, methods are present for accessing specific information in response to a particular set of MIR supported messages. Requests for information are made in terms of these messages. Once the representation of needed information is understood, the methods stored in an appropriate module can be used to retrieve the information without the user understanding the details of how this is accomplished.

Information retrieval makes use of an information format that provides for the transmission of messages between MIRs. These messages are used in transmitting requests to an MIR and subsequently returning the information which constitutes a reply. This process is also referred to as message passing. For message passing to occur automatically, additional methods are developed on a remote system that are responsible for making requests in response to needs for information. Such needs are initiated either by a user making requests in terms of a local query language or more directly by applications. These methods make use of the messages supported by the MIR modules with which they are communicating. Methods are also developed for entering information into the requesting information system upon receiving a reply. Once these methods are in place, requests and replies for information between systems are exchanged in an automatic and transparent fashion.

An interface capability based on application layer protocols as described above makes it relatively easy for systems to be connected to one another. Requirements include local MIRs and the ability to develop the additional methods for making specific requests and entering the information that is received in reply. The perspectives and additional methods are developed between systems that respond precisely to actual needs for specific information transfer. This characteristic provides for a highly adaptive communication capability.

Application layer protocols are also of use in those situations that suggest a global information management system approach. Since meta information is available in an MIR, a global system interface can make use of the knowledge it contains. Expectations as to the format of this knowledge is useful when adding systems to a global system interface. The development of automated processes for the addition of new systems is possible in this context.

Application layer protocols therefore can provide communication capabilities that address both short and long term needs of the building industry. The sections which follow present meta information and information formats for application layer protocols and an implementation of an interface based on their use.

4. Representing System Semantics in Modules for Selective Network Communication of Information: Frames

Information system semantics that constitute the meta information of a system can be divided into three categories: declarative, compositional, and procedural [16,17,18]. Declarative semantics derive from the definition of a conceptual schema. This is typically accomplished through the use of a data definition language that embodies a particular underlying approach to the organization of information (the theoretical data model). Declarative semantics often become a major part of a system's data dictionary.

Compositional semantics derive from syntactic conventions of an information system. The meaning resulting from the position of symbols in specifying a particular operation or query is an example.

Procedural semantics derive directly from the conceptual level operations that are provided by a system. That is, the meaning of the system content (stored values) is established by the operations of that system. Stated another way, values have meaning as a result of the operations which serve to assign, retrieve, or manipulate those values.

The specification of application layer protocols for meta

information involves first the identification of the semantic concepts that can serve to encapsulate the desired knowledge, and second, the selection of symbolic elements that serve to identify those concepts. The protocols must be eclectic in the sense that the fundamental concepts of the various theoretical data models should be accommodated to allow for translation between what are to be essentially alternative representations. The identification of semantic concepts should also consider the nature of the discipline in which a protocol is to be used to determine if there are unique requirements or characteristics specific to that discipline.

The frame-based knowledge representation developed in the field of artificial intelligence [19,20] provides a means of storing the declarative, compositional, and procedural semantics that constitute the meta information of a system. Simply stated, a frame is a collection of properties, applicable to all instances of an identified class of objects. (An "object" in this context refers to a uniquely identifiable thing, event, or concept.) A frame can be thought of as a template or stereotype reflecting certain expectations regarding properties of an object based on the class to which it belongs.

A frame-based knowledge representation is essentially modular in nature. Each frame captures knowledge about a particular object class. This may include a description of the

object class, the characteristics of the object class, and the relations that exist between this and other object classes. These constitute necessary declarative semantic concepts for encoding a conceptual schema. A frame-based representation can also capture knowledge concerning operations that are of relevance to a given object class (compositional and procedural semantics).

The use of a frame-based knowledge representation reflects an orientation considered appropriate to the building industry. That is object classes like doors, windows, hallways, or, for that matter, buildings serve as an intuitive basis for organizing building information. Further, the modularity that a frame-based representation provides is well suited for the establishment of communications capabilities that address the particular needs of the participants that wish to communicate.

The symbolic elements used to describe the Modular Semantic Application Layer Protocols are described in the sections that follow.

4.1 Meta Information Modules

The entity-type module has been chosen as the basic unit of the Meta Information Format. An entity-type module is a

collection of properties (i.e., a frame) applicable to all object class identifiers of a conceptual schema (i.e., of the class "entity-type"). That portion of the meta information directly related to a given entity-type is represented as the contents of its module properties.

Table 1 presents definitions of the properties comprising an entity-type module. They include a description of the entity-type, the relationships involving the module entity-type and other related entity-types, key-attributes of the module entity-type, non-key attributes, and interpreters which capture compositional and procedural semantics unique to the module entity-type.

An entity-type module in tabular form is presented as Table 2. The module entity-type is the identifier for which the entity-type module is established. The description property contains text that defines or otherwise describes the entity-type of a given module. (This as well as all other module properties are optional.)

The relationships property is used to represent the logical connections between object classes of a conceptual schema. A relationship includes the module entity-type, a pair of roles that define a relation, and one or more related entity-types. The first role is assigned to the module entity-type, while

Table 1. The Entity-Type Module.

entity-type module	A collection of properties applicable to all entity-types of a conceptual schema (i.e., to all instances of the class "entity-type").
Module Entity-Type (MET)	The conceptual schema object class for which an entity-type module is created.
description	A property containing Description of the module entity-type.
Description	Text which specifies the conceptual schema object class represented by the module entity-type.
relationships	A property comprising a Relation, an ordered pair of Roles, and Related Entity-Types.
Relation	A logical connection between entity-types.
Role	A term used to identify the part played by an entity-type in a Relationship.
Related Entity-Type (RET)	A conceptual schema object class related to the MET by a given Relation

Table 1. (continued)

attributes	A property comprising an Attribute, Domain,and Domain-Values, (or Constraints).
Attribute	A conceptual schema identifier used to represent a characteristic of the conceptual schema object class represented by the MET.
Domain	A schema identifier which represents the set of possible values of an Attribute.
Constraint	A list of values or one or more rules governing allowable values that make up the Domain (i.e., that defines the acceptable Attribute-Values.
key-attributes	A property comprising a Key-Attribute, Domain, Domain-Values and Constraints.
Key-Attribute	A conceptual schema identifier used to represent a key characteristic of a conceptual schema object class, the value of which serves to identify an instance of that object class.
interpreters	A property containing Selectors to which an interpreter can respond and the associated methods which represent compositional and procedural semantics specific to the entity-type for which the module is established.
Selector	A message identifier used to communicate the type of information desired.
Method	An expression which initiates a conceptual level operation.

Table 2. Elements of an Entity-Type Module in Tabular Form.

Module Entity-Type (MET)	Description
relationships	
MET Role (RET Role)	: Related Entity-Type ...
.	
.	
.	
key-attributes	
Key-Attribute	: Domain Constraint
.	
.	
.	
attributes	
Attribute	: Domain Constraint
.	
.	
.	

the second (the inverse) is assigned to the one or more related entity-types. A relationship establishes the place of the module entity-type within the organizational "structure" of the conceptual schema. This is more easily visualized when the entity-type module is portrayed in a semantic network form (Figures 7 and 8).

Only those other entity-types directly related to the module entity-type are visible from a given module. Therefore, no assumptions are made about the overall structure of a conceptual schema. The schema structure is revealed layer-by-layer from an arbitrary starting point (the module entity-type) as the modules of successive related entity-types are accessed.

It should be noted that the term "relationship" in the Meta Information Format can be defined on more than two entity-types. Further, relations are used exclusively to represent logical interconnections between entity-types. That is, relations are not allowed to have attributes. Further, relations that may exist conceptually between attributes are represented by establishing a module for those attributes. That is, they are represented as entity-types.

A pair of roles that define a relation capture semantics associated with the conceptual schema structure. Examples of relations and the corresponding roles useful in the

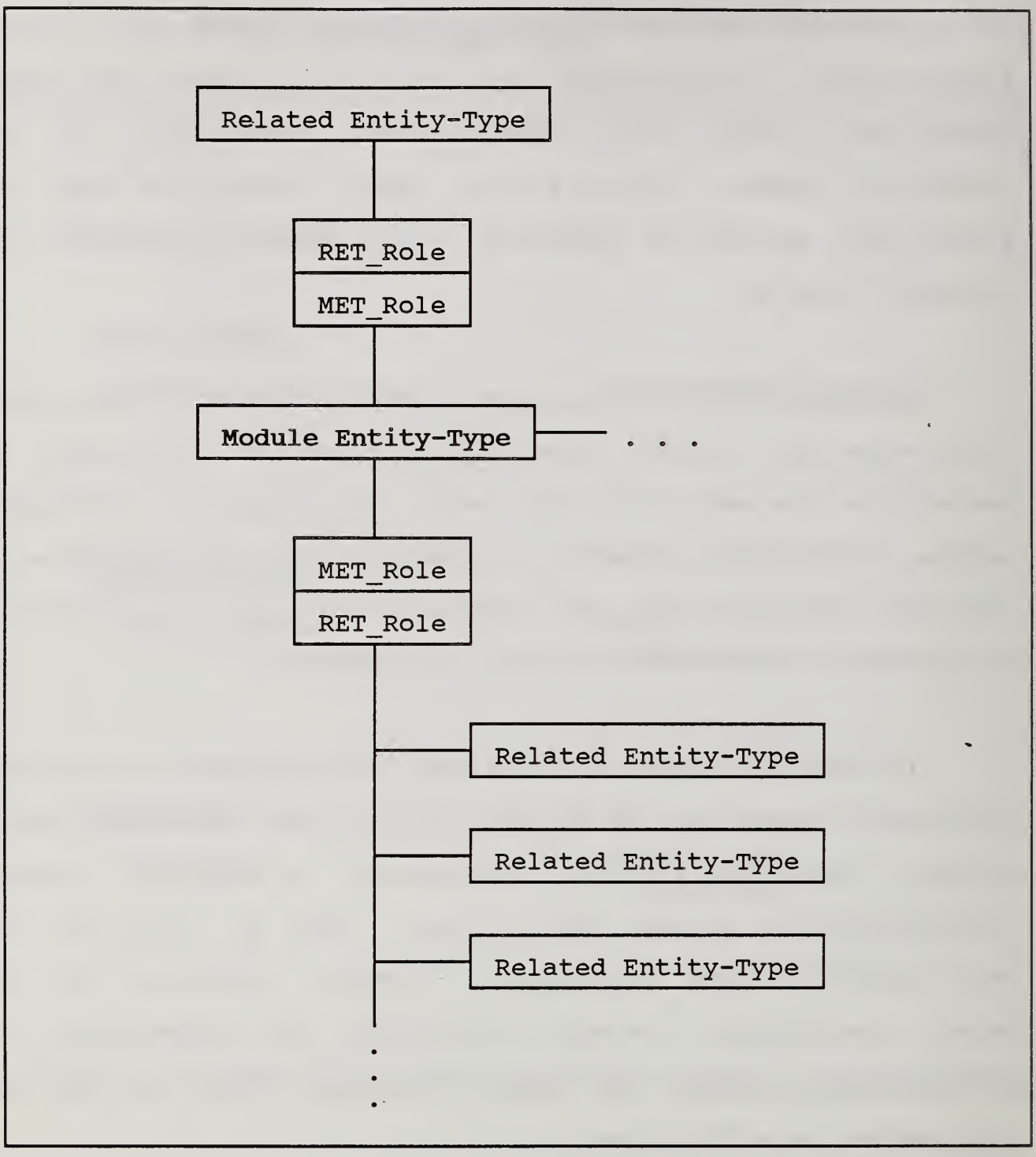


Figure 7. Entity-Type Relationships in Semantic Network Form.

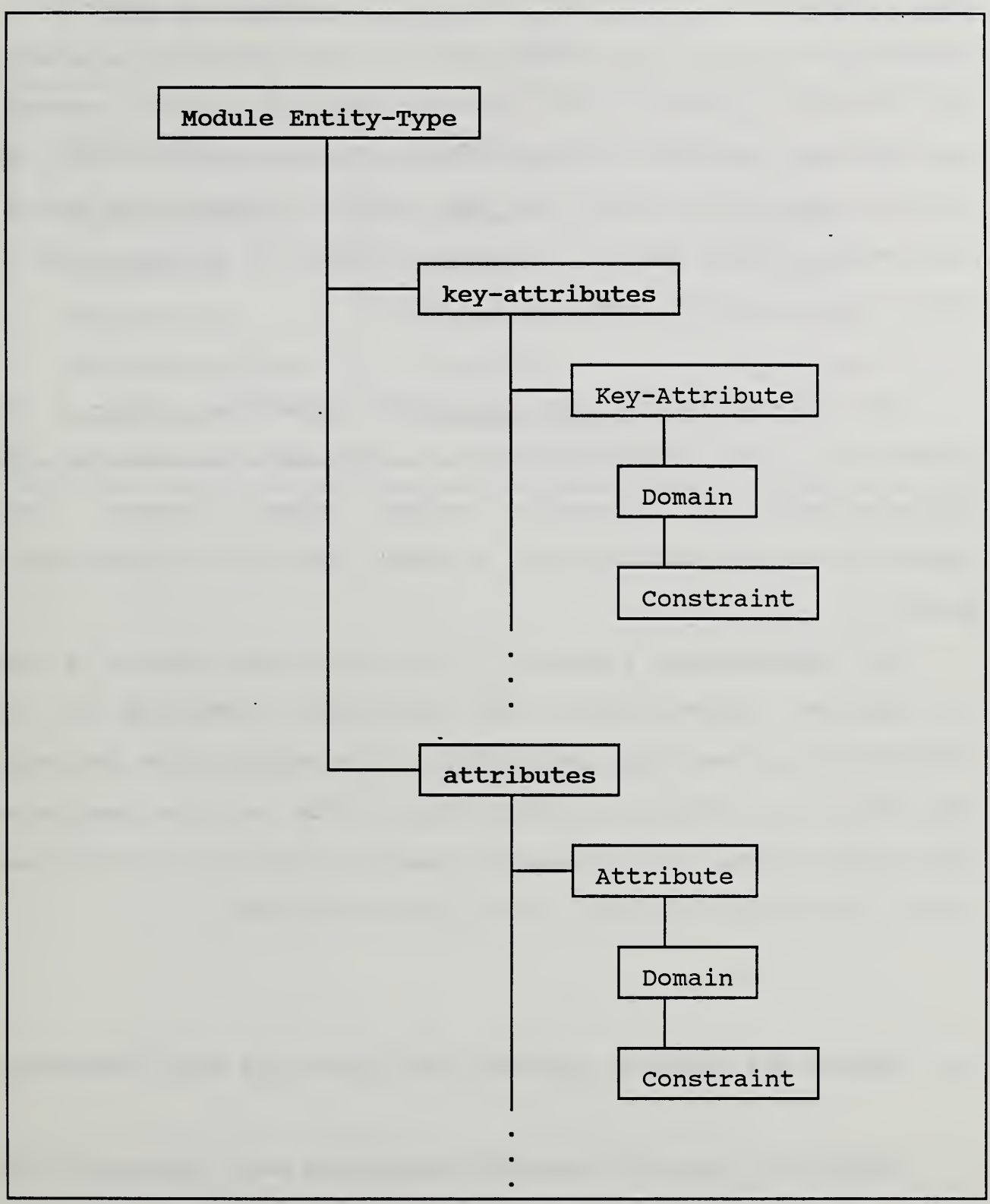


Figure 8. Entity-Type Attributes in Semantic Network Form.

Meta Information Format are presented in Table 3. The use of relations defined by an ordered pair of roles provides a flexible and extensible system for the representation of semantic concepts that are the hallmark of many "semantic" data models (e.g., the Semantic Association Model and the Semantic Hierarchical Model). Discussions of the semantic concepts embodied in the relations of Table 3 can be found elsewhere (21,22,23).

The key-attributes and attributes properties (Table 2 and Figure 8) of an entity-type module are used to represent the characteristics of conceptual schema object classes. Each key-attribute and attribute has a domain from which values can be drawn.

The interpreters property of an entity-type module is used to capture compositional and procedural semantics of an information system that are unique to the entity-type for which the entity-type module is established. This is to be contrasted with compositional and procedural semantics that are of relevance to all entity-type modules, to be discussed later.

4.2 Tables and Semantic Networks for Displaying Meta Information

Tables and semantic networks displaying meta information are often essential for understanding the schema of an information system. The previous section gave examples of how meta

Table 3. Common Relations and Roles used in Relationships.

RELATIONS:	ROLES:	
Generalization	type-of	has-type
Aggregation	part-of	has-part
Association	member-of	has-member
Set Association	subset-of	has-subset
"relation"	(schema defined roles)	

information can be represented using the abstract symbolic elements. Tabular and semantic network forms were presented in Table 2 and Figures 7 and 8.

An example entity-type module in both the tabular and semantic network form is presented in Table 4 and Figures 9 and 10. This module is from a schema used by a hypothetical application program that assists a mechanical engineer in the design of heating, ventilating, and air conditioning (HVAC) systems. (This example will be followed throughout subsequent sections to illustrate the substitution of information system elements for abstract elements in the formats used to communicate selected information.)

A hypothetical building economics application is to carry out a life cycle cost analysis of two alternative heating systems that have been determined to be capable of providing essentially equivalent performance by the HVAC application. The building economics application requests certain specific information about the alternatives from the information system used by the HVAC application. The example begins with the task of understanding the schema of the information system supporting the HVAC application. For the purpose of an example the schema has been limited to a single entity-type module.

The entity-type is HS (heating system). HS has all five

Table 4. Example of an Entity-Type Module in Tabular Form.

HS	Heating System: An HVAC subsystem providing heat to one or more zones of a building	
relationships		
PART-OF (HAS-PART)	:	HVAC
HAS-PART (PART-OF)	:	COMPONENT CONTROL PATH
key-attributes		
HS-ID	:	BUILDING-NUMBER "B-integer&-character"
attributes		
TYPE	:	FUEL ELECTRIC GAS OIL SOLAR
EFFICIENCY	:	PERCENT "integer"
INITIAL-COST	:	DOLLARS "integer"
LOAD	:	MMBTU "real"

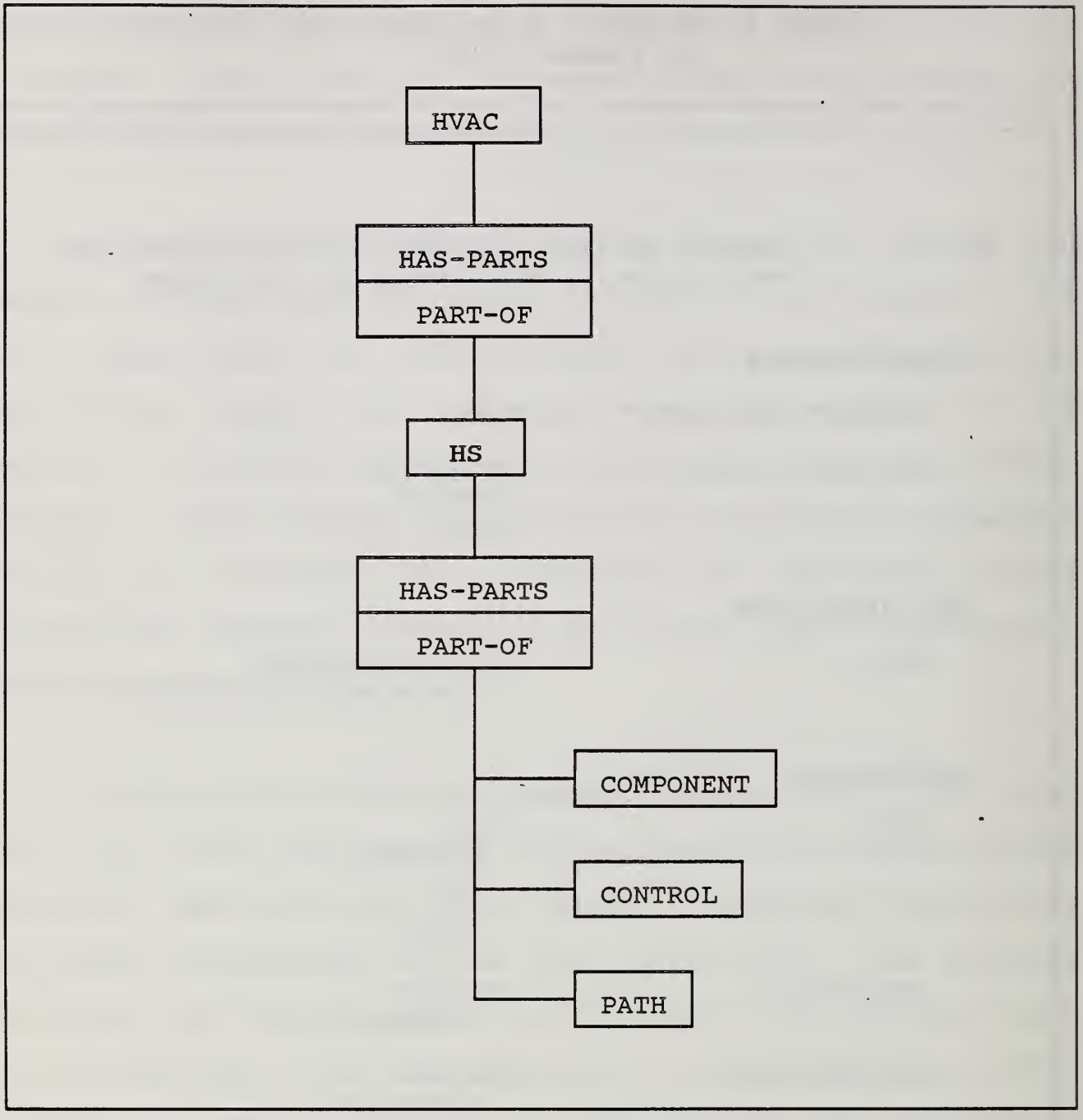


Figure 9. Example Entity-Type Relationships in Semantic Network Form.

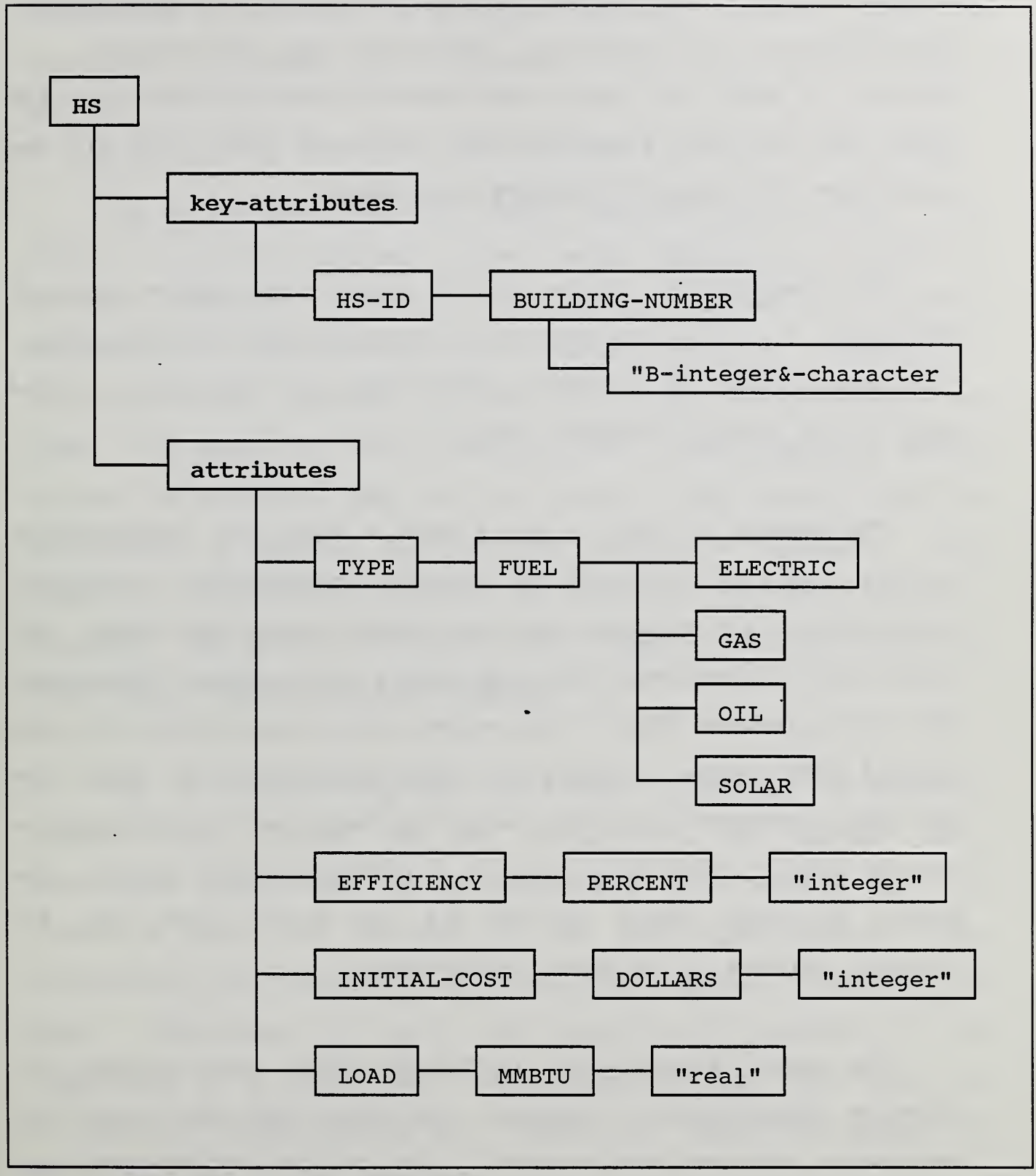


Figure 10. Example Entity-Type Attributes in Semantic Network Form.

properties of an entity-type module. They are its description, relationships, key-attributes, attributes, and interpreters. However, in both the tabular and semantic network forms only the first four of these properties are displayed since they are the properties that serve to specify the schema.

The description of the module entity-type states that HS represents "... an HVAC subsystem providing heat to a building." (The description of a module entity-type is typically omitted when in the semantic network form.)

The module in Table 4 and Figure 9 shows two relationships for HS. The first involves the relation aggregation. It has the roles part-of and has-part and the related entity-type HVAC. The first role is associated with the module entity-type. Therefore, "HS [is a] part-of HVAC." The second role is associated with the related entity-type. Therefore, "HVAC has-part HS." Note that the HVAC aggregate has other parts but they are not displayed in the HS module since they are not directly related to HS. The module for HVAC would include all the parts since they are directly related to the HVAC entity-type.

The second relationship in the HS module also involves the relation aggregation. However, in this case the roles are reversed: has-part and part-of. The related entity-types are component, control, and path. Therefore, the relationship is

understood to be that "HS has-part component, control, path" or conversely that each part "component, control, path [is a] part-of HS."

The HS module has the single key-attribute HS-ID. Its domain is building-number. The domain values are limited to integers preceded by "B" and optionally followed by a letter representing alternatives for a given building. Additional attributes of HS include its type, efficiency, initial-cost, and load. The domain of the HS type is fuel. The acceptable domain values are electric, gas, oil, and solar. The domain of the HS efficiency is percent with values limited to integers. The domain of the initial-cost is dollars with values limited to integers. The domain of the HS load is mmBTU (millions of BTU's) with values limited to real numbers.

The HS module displayed in either a tabular or semantic network form allows for the decision as to whether this particular entity-type is of importance to a remote user and therefore should be included in that user's perspective. The perspective can include or omit this module either in whole or in part. Therefore, if only the initial-cost attribute is of importance to a given user, the perspective for that user can exclude the other attributes. Similarly, selection among the relationships is also possible.

4.3 Perspectives and the Resulting Views of Information

The entire conceptual schema in the Meta Information Format constitutes the most comprehensive perspective that can be taken regarding an information system. It is in fact the Meta Information Representation (MIR) of the information system (Table 5). Subschemata comprise the alternative perspectives that are listed in the MIR perspectives property. A subschema is a subset of the conceptual schema. The subset includes only entity-type modules and elements of those entity-type modules that are of interest to a particular user or set of users.

Perspectives provide a means of grouping together a number of entity-type modules including the compositional and procedural semantics of an information system that are of relevance to those entity-type modules. Table 6 presents the definitions of the elements of a perspective. A perspective is a collection of properties applicable to all instances of the class "perspective," each of which is identified by a perspective-ID.

The entity-types property of a perspective identifies those entity-types whose entity-type modules are included in the perspective. The messages property identifies those messages to which a perspective can respond. The translators property of a perspective is used to capture compositional and procedural semantics of an information system that are relevant to

Table 5. The MIR

Meta-Information Representation (MIR)	A collection of properties applicable to a MS-ALP representation (i.e., instances of the class "MIR").
System-ID	An identifier used to designate an instance of the class "MIR."
perspectives	A property of an MIR which identifies its constituent perspectives.

Table 6. The Perspective

perspective	A collection of properties applicable to the conceptual schema and all subschemata that are a subset of the conceptual schema (i.e., to all instances of the class "perspective").
Perspective-ID	An identifier used to designate an instance of the class "perspective."
entity-types	A property of a perspective which identifies its constituent entity-types.
messages	A property containing the Selector which will be used to activate methods and the arguments that are required.
translators	A property containing Selectors to which a translator can respond and associated methods which represent compositional and procedural semantics that are of relevance to all entity-types of a given perspective.

the included entity-type modules.

In the HVAC example, a remote user identifies his or her perspective by assigning a perspective-ID such as building-economist-perspective. The hypothetical application needs access to the attributes of the HS module and only the relationship indicating the hvac system of which the heating system is a part.

A view is a representation of the contents of an information system accessed via the selected perspective. Access is provided by the perspective translators in combination with entity-type interpreters as needed. The action of these properties is essentially transparent to the user or application communicating with the system. The amount of information that can be accessed is limited by the scope of the perspective through which an authorized user or application is viewing the system contents. Therefore, in the example of the building-economist-perspective, a building-economist-view will be limited to information concerning HS type, efficiency, initial-cost, and load.

4.4 Information Modules

The basic conceptual unit of the Information Format (though not transfer protocol unit, see section 6.3) is the entity module. Table 7 presents the definition of the elements of an entity module. The entity module is a collection of properties applicable to the objects of an information system (i.e., of the class "entity"). An entity is represented by following the entity-type with the one or more keys (i.e., values of key attributes) necessary to uniquely identify an information system object (e.g., entity-type (key)). It represents an instance of a given entity-type.

In the HVAC example, there are three HS systems with HS-IDs from the domain building-number of B1A, B1B and B2. The building-numbers containing the "A" and "B" are alternative HS systems that are being considered for the same building (B1). The corresponding entities are HS (B1A), HS (B1B) and HS (B2).

Since an entity represents an instance of an entity-type, an entity module can be thought of as an instance of an entity-type module. The properties of the entity module are limited however to relationships and attributes. The relationships and attributes of the entity-type modules included in a given perspective define the scope of the information that may be viewed.

Table 7. The Entity Module.

entity module	A collection of properties applicable to all objects of an information system (i.e., to all instances of the class "entity").
Module Entity	An instance of the conceptual schema object class (entity-type_key) for which an entity module is created.
Key	A member of the set specified by the Key-Attribute Domain used to identify an instance of an entity-type (entity).
relationship	
Entity-Role	A term used to identify the part played by an Entity in a Relationship.
Related-Entity	An information system object related to the Module Entity.
attribute	
Attribute	An identifier used to represent a characteristic of an Entity which is defined by an attribute of the Entity-Type of which the Entity is an instance.
Domain	An identifier which represents the set of possible values of an Attribute as defined by the Entity-Type of which the Entity is an instance.
Value	A member of the set specified by the Attribute Domain of the corresponding entity-type.

Information system contents are represented within the entity module properties. The elements of the entity module properties emphasize the role of the entity in a relationship. Therefore, the elements of the Information Format concerning relationships includes the entity, its role in the relationship and the related entities (instances of the related entity-types). Correspondingly the elements concerning attributes include the attribute, its domain, and value.

For the example HS entity-type module, there are three entity modules; HS (B1A), HS (B1B), and HS (B2). An example relationship would be that an entity such as HS (B1A) is part-of the HVAC system for building one (HVAC (B1)). As for the attributes, an entity like HS (B1A) may have an oil fuel type.

The inclusion of these modules and their contents depends upon the perspective from which the information is viewed. One or more entity modules or parts thereof, accessed via a given perspective, make up a view. Table 8 presents the definition of the elements of a view. A view is a collection of properties applicable to all instances of the class "view," each of which is identified by the perspective-ID which identifies the perspective from which the view is derived.

The entities HS (B1A) and HS (B1B) viewed from the building-economist-perspective constitute the building-economist view.

Table 8. The View

view	A collection of properties applicable to the conceptual view and all "subviews" that are a subset of the conceptual view (i.e., to all instances of the class "view").
Perspective-ID	An identifier indicating the perspective from which the view is derived.

This perspective omits the second relationship involving aggregation but includes the first relationship and all four attributes of the HS entity-type module. The entity modules are also similarly limited. Therefore, hypothetical information such as that shown in Table 9 can be accessed for each entity.

A view of an information system is obtained via a perspective by using the message passing capabilities of the Meta Information and Information Formats. Messages are provided for accessing individual relationships or attributes of an entity module or for accessing entire modules. Therefore, a message can ask that the type of HS (B1A) be supplied so long as that information is part of the authorized perspective. Alternatively, everything that is known about the HS (B1A) entity module from a given perspective may be requested.

A list of supported messages is a part of the Meta Information Representation. The context in which these messages act includes the perspective translators and entity-type module interpreters. The translators and interpreters contain the knowledge necessary to retrieve requested information. Object-oriented programming is the technique by which messages are passed among appropriate translators and interpreters of the Meta Information Format.

Table 9. Example Information.

	Entities:	
	HS (B1A)	HS (B1B)
Relationships:		
PART-OF	HVAC (B1)	HVAC (B1)
Attributes:		
TYPE (FUEL)	OIL	ELECTRIC
EFFICIENCY (PERCENT)	75	100
INITIAL-COST (DOLLARS)	3500	2000
LOAD (mmBTU)	50.155	50.155

5. MS-ALPs Approach to Local Communication Interfaces: Object Oriented Programing

Object-oriented programing provides a technique for making use of the compositional and procedural semantics captured within perspective translators and entity-type module interpreters of the Meta Information Format. These properties have as their contents a number of message identifiers (selectors) to which they are able to respond and associated procedures called methods. When a recognizable message is passed to a translator or interpreter, the corresponding method is initiated.

The translators and interpreters of the Meta Information Format provide a self contained access capability to a local information system or application. Such a capability makes communication between incompatible information systems possible. It can also be of use to global system interfaces. The process of passing messages and information is essentially the same whether a local system is connected to a remote terminal, another information system, an application, or a global system interface.

5.1 Translators and Interpreters for Obtaining a Particular View of Information from a Given Perspective

Passing a message to a translator or an interpreter is

equivalent to delegating control over what is done with that message. If a message contains a request for information from outside an information system the message can be passed between various translators and interpreters that contain the necessary knowledge to respond to the request. Appropriate system operations are initiated and entity modules or parts thereof making up a view can be obtained.

In the case of the HVAC example, the building economics application is to perform a life-cycle cost comparison between two alternative heating systems for building one. The values of the HS type, its efficiency, initial-cost, and load can be retrieved for each alternative by sending a message requesting the entity modules for HS (B1A) and HS (B1B) from the building-economist-perspective.

The request in the form of a message supported by the perspective is delegated to the appropriate translator for analysis. Methods are used to consult entity-type modules for relevant schema information. This is accomplished by delegating control to the entity-type interpreter.

The entity-type interpreter contains knowledge concerning operations that may be specific to the particular entity-type. If a response unique to that entity-type is required, the interpreter will initiate local information system operations

either directly or indirectly through perspective translators. Alternatively, if a response generally applicable to entity-types of the perspective is sufficient the interpreter will delegate the message to an appropriate perspective translator.

In the case of unique requirements for the requested information, the interpreter can take appropriate action not only in terms of initiating information system operations but also in terms of analyzing the response it receives from the information system. The result of such an analysis determines the content of the reply subsequently passed to perspective translators. The translators are then responsible for encoding the reply in the correct form for transmission over the communication channel.

In the case of an acceptable general solution, the entity-type interpreter delegates the message to appropriate perspective translators that initiate information system operations to retrieve the requested information. These translators then pass the reply on to other translators that provide for transmission of the reply.

In the case of the building-economist-perspective, a perspective translator receives a message requesting the entity modules for HS (B1A) and HS (B1B). This translator passes the message on to the HS entity-type module. The HS module determines whether a specialized or general approach to

fulfilling the message is appropriate.

The entity-type module knows that the HS type, efficiency, and initial cost are contained in an HVAC-database of the engineering information system. The message is therefore delegated to a translator of the building-economist perspective with the name of the appropriate database added. The load however is contained in a thermal-analysis database. The message in this case is delegated to the translator with this database name (see Appendix).

Once the view of the HS type, efficiency, initial-cost, and load has been obtained for HS (B1) and HS (B2), it is passed to a perspective translator that prepares the view for transmission in standard form. The actual transmission of the view as well as the initial delivery of the request to the building-economist-perspective is the function of a messenger.

5.2 The Use of a Messenger to Deliver Requests for Information and Transmit Replies

The messenger is responsible for initiating the process that ultimately makes a current view of the information available to the user or application. Table 10 presents the definition of the messenger. The messenger is sent with a message to a

Table 10. The Messenger

messenger	The means by which a request for a view from an available perspective is delivered and a reply is returned.

perspective. It delegates the message to the appropriate perspective translator, and later receives a view as a reply. The view can then be transmitted to the message origination point.

The message passing process is diagrammed in Figure 11. A flowchart of the delegation of the message at each stage in the process, keyed to that diagram, is presented in Figure 12.

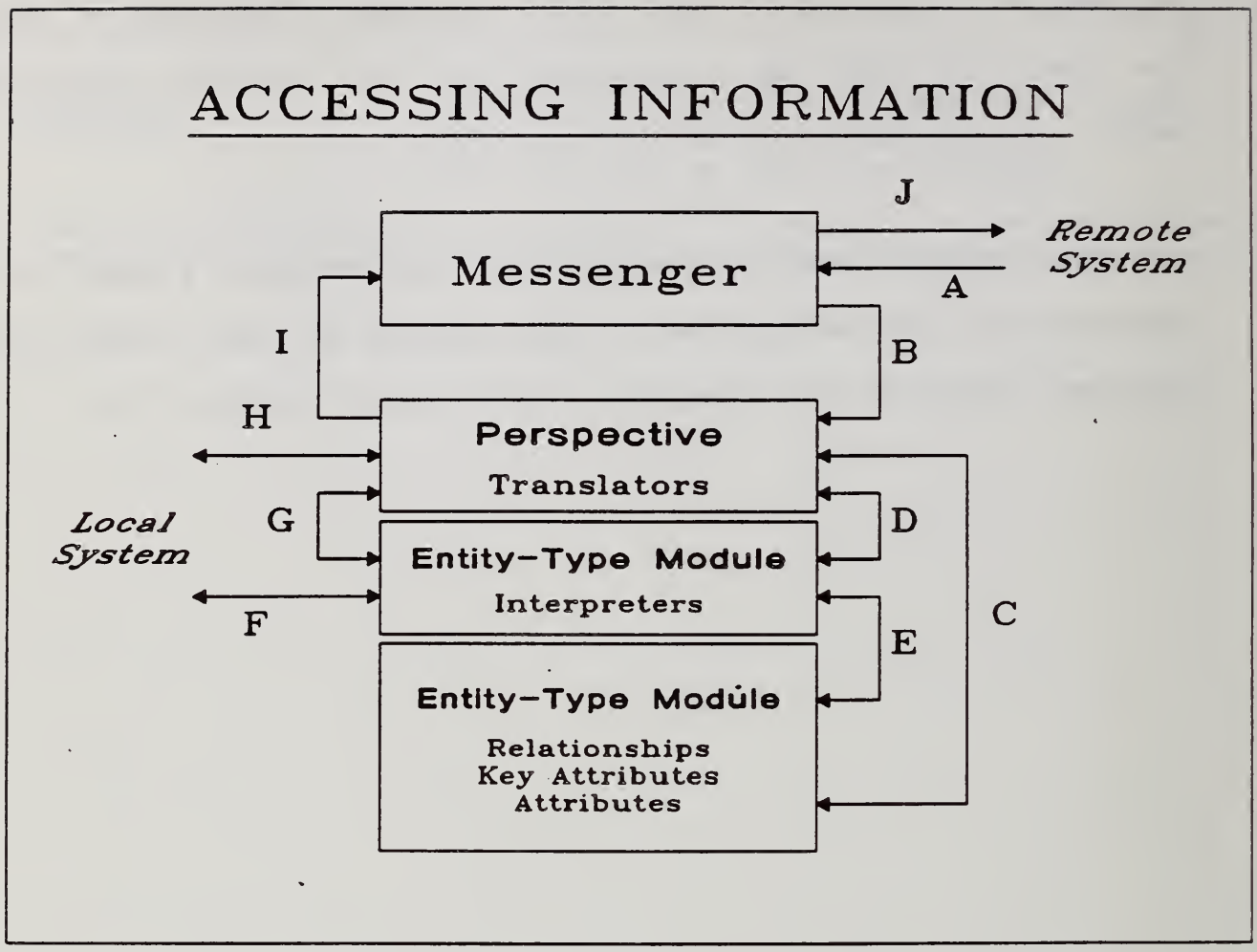


Figure 11. Accessing Information Using a Messenger

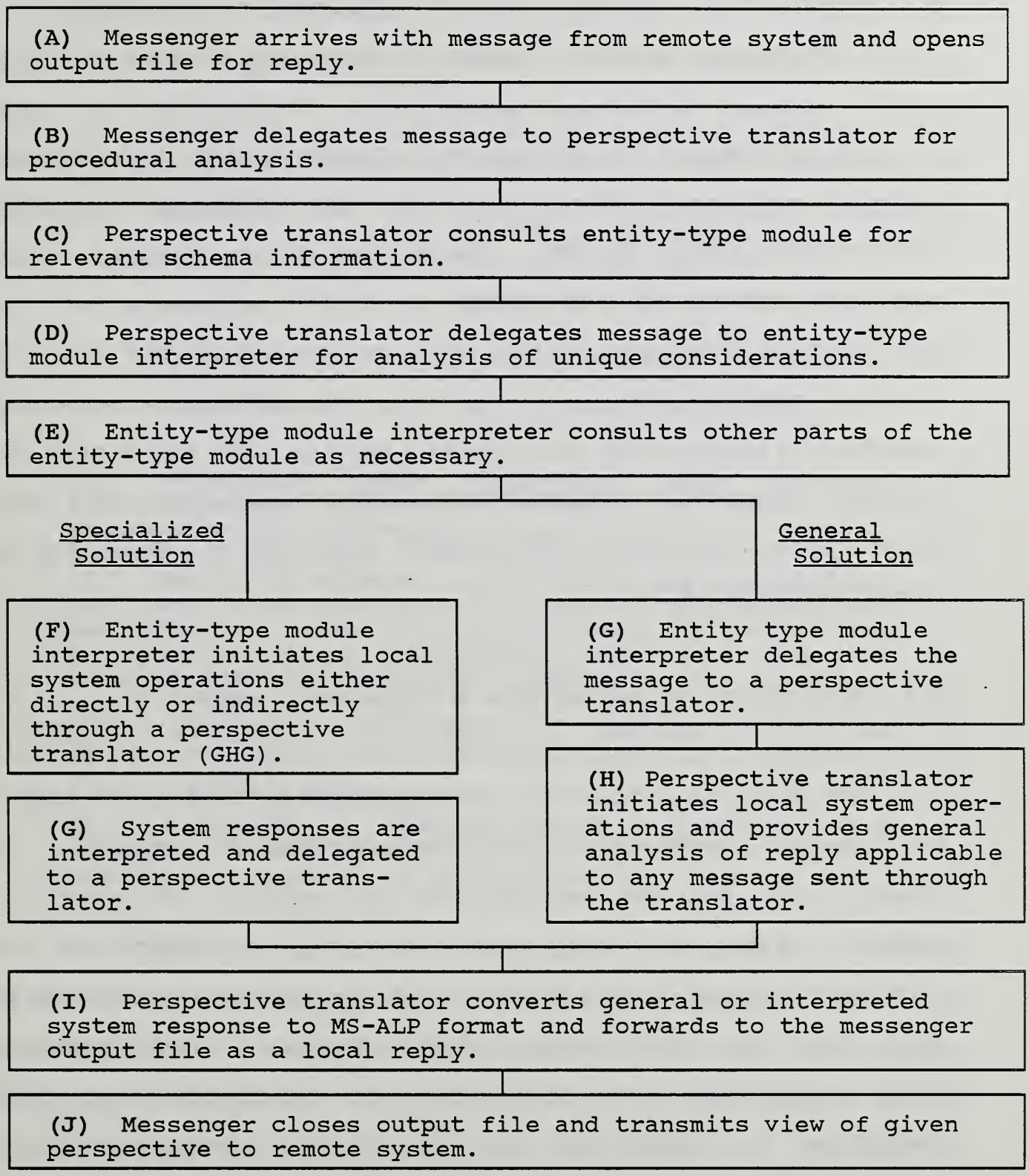


Figure 12. Sequence of Events in Accessing Information.

6. Implementation of MS-ALPs: The Perspectives Available to a Messenger and the Transfer of Appropriate Views

An implementation of the Meta Information and Information Formats has been undertaken within the Computer Integrated Construction Group of CBT. Though the frame elements of the developed formats are not limited to a LISP implementation, they were in fact developed within that context during the course of research reported here. The formats are therefore most easily amenable to programming environments such as LISP and Prolog that provide for symbolic manipulation. Other languages, such as C, would require additional development including a parser for the forms used by MS-ALPs.

6.1 Integration of Declarative & Procedural Semantics

The field of artificial intelligence has long found LISP to be a useful language for the representation of knowledge. Its capabilities for the manipulation of symbols and groups of symbols (lists) have been used extensively in efforts to make computers perform intelligently. Of particular interest is the fact that List Programming (LISP) provides an environment in which both data and procedures are represented as lists. Therefore, an integration of declarative, compositional, and procedural semantics is possible.

The LISP environment also provides the capabilities necessary to implement many of the programming concepts currently used in artificial intelligence including property lists, structures, methods, objects, and others. These features are directly applicable to the use of a frame-based knowledge representation as discussed in the previous sections.

The implementation of formats for meta information and information makes use of four basic programming concepts:

(1) Developing **frames** (structures) using the Meta Information and Information Formats to encode meta information and information respectively.

(2) Developing **methods** (lambda forms, i.e., procedures as lists) that capture compositional and procedural semantics of a given information system and provide for an object-oriented programming capability that can interact with the information system.

(3) Defining a **messenger** (a function) which can deliver messages to an available perspective and return an appropriate view.

(4) Developing a **message-passing** capability that delegates control over what is done with a message between translators and interpreters.

The development of a frame-based information protocol provides for the representation of static elements of both the Meta Information and Information Formats (Table 11). The formats are used to encode the conceptual schema of an information system (Meta Information Format) and views of information (Information Format). The dynamic elements of the Meta Information and Information Formats are encoded using methods, a messenger, and a

Table 11. Static and Dynamic Elements of MS-ALPs.

	META INFORMATION FORMAT	INFORMATION FORMAT
STATIC	MIR System-ID perspectives Perspective Perspective-ID description entity-types messages Entity-Type Module Entity-Type description relationships attributes key-attributes	View Perspective-ID Entity entity-roles attributes
DYNAMIC	Perspective Perspective-ID translators Entity-Type Module Entity-Type interpreters	Messenger View-ID messages

means of delegating responsibility for messages. Methods retrieve views from an information system or application in response to messages. The messenger in conjunction with a message-passing capability provides the means by which messages and views are handled.

Providing for a communication capability using MS-ALPs begins ideally during the initial development of an information system or application. A system MIR is developed which captures the knowledge required to establish an understanding of the system. A messenger is also developed that is capable of performing system input and output as well as delegating responsibility for messages.

The system MIR comprises both the static and dynamic elements of the Meta Information Format. This requires first a translation of the conceptual schema from the data model used by the system to the Meta Information Format. It then requires decisions regarding what system operations are to be made available through methods and the subsequent development of those methods. Implicit in this process is a selection of the messages for which methods are to be provided. Messages involving both access and input are possible. For the purpose of this discussion only access will be considered.

Once an MIR is developed, the static elements and the

messenger can be transmitted to any number of external users. An external user (most likely a system or network developer) determines that part of the MIR that is of interest. A user perspective is thereby established. The user perspective is returned to the information system and becomes a resident feature of the system.

Once a user perspective has been established, the messenger can be used to pass messages from the external user to the resident user perspective. The translators of that user perspective in combination with its entity-type module interpreters then provide a view in accordance with the received messages. The messenger makes this view available to the external user. The user must provide for translation between the Information Format representation and a form suitable for local applications.

Establishing communication between two incompatible information systems requires that the above process be completed for each system. Communication employing Meta Information and Information Formats has been established to date within the context of distributed information systems operating within LISP environments. This reflects both the preliminary phase of this research, in which a homogeneous programming environment is desirable, and the recognition of the increasing importance of knowledge-based systems within the building industry.

6.2 The LISP Interpreter as Working Memory

Within a LISP environment, the interpreter serves as a working memory into which both data and functions (procedures) can be entered. LISP functions provide a means of manipulating the contents of working memory. Meta information can be loaded into this environment and subsequently consulted interactively. A user can gain an understanding of an information system schema and in the context of that understanding access information through the use of appropriate message passing capabilities.

The Meta Information Format as implemented at CBT uses LISP functions to include meta information in an MIR and subsequently display the meta information on request. Customized functions can be developed within a particular programming environment which make use of the unique input and output characteristics of that environment. This is in fact necessary in the case of the methods used to access information. Functions may also be desirable to provide report generation in an end user oriented form. These specialized functions, however, are not part of the Meta Information Format.

6.3 The Meta Information Format

The Meta Information Format is presented in Table 12. The basic unit of meta information is a property of a frame. A set of properties serve to define the frame. The properties that define an entity-type module include its description, relationships, key-attributes, attributes, and interpreters. Similarly, a perspective is encoded in terms of its properties. These include the entity-types, messages, and translators. The general form applicable to the properties of the perspective and entity-type modules is:

```
(<property> <frame-identifier>
     <property-information>)
```

Enclosure in "<>" indicates that an appropriate element or list of elements is to be substituted. The property-information consists of one or more elements that capture the meta information relevant to the given property.

In the case of the example building-economist-perspective, the entity-type property-information is a single element, the entity-type that is being included in the perspective. An example entity type follows: (Note: For portability, the use of upper case characters is advisable.)

```
(ENTITY-TYPE BUILDING-ECONOMIST-PERSPECTIVE HS)
```

Table 12. The Forms of the Meta Information Format.

MIR Module: System-ID

```
(MIR System-ID)

(PERSPECTIVE System-ID Perspective-ID)
```

Perspective Module: Perspective-ID

```
(ENTITY-TYPE Perspective-ID Entity-Type)

(MESSAGE Perspective-ID (Selector (Argument ...)))

(TRANSLATOR Perspective-ID (Selector (Method)))
```

Entity-Type Module: Module-Entity-Type (MET)

```
(DESCRIPTION Module-Entity-Type ("Description" ...)

(RELATIONSHIP Module-Entity-Type
     (MET-Role (RET-Role (Related-ET ...))))

(KEY-ATTRIBUTE Module-Entity-Type
     (Key-Attribute (Domain (Constraint ...))))

(ATTRIBUTE Module-Entity-Type
     (Attribute (Domain (Constraint ...))))

(INTERPRETER Module-Entity-Type
     (Selector (Method)))
```

The property-information of the message property is in the form of an association list. That is, the property-information is in the form of a list that contains two associated elements. The first element is the message identifier or selector. The second is a list of arguments that are to be supplied when sending a message with that selector. The following is an example:

```
(MESSAGE BUILDING-ECONOMIST-PERSPECTIVE
     (RETRIEVE-ATTRIBUTE
      ((ENTITY-TYPE (KEYS)) ATTRIBUTE)))
```

The property-information of the translator property is also in the form of an association list. The first element is the message identifier to which the translator responds (the selector). The second is a list containing a Lambda form, that is, a LISP function that can be evaluated directly without the need for a function name (see Appendix). The Lambda form is the method used to respond to a message containing an element that matches a translator selector. The translator property takes the form:

```
(TRANSLATOR BUILDING-ECONOMIST-PERSPECTIVE
     (SEND-ATTRIBUTE (<method>))

(TRANSLATOR BUILDING-ECONOMIST-PERSPECTIVE
     (RETRIEVE-ATTRIBUTE (<method>))
```

Selectors beginning with SEND are used to respond to

requests from an external user while those beginning with RETRIEVE are used by translators and interpreters to obtain information from the resident information system or application.

The general form for the properties of a frame also applies to the entity-type modules of a given perspective. The precise nature of the property-information depends upon the property of which it is a part. The description property has property-information in the form of a simple list of strings (i.e., phrase-like structures enclosed in double quotation marks that taken together constitute sentences).

```
(DESCRIPTION HS
     ("Heating System:  An HVAC subsystem providing "
      "heat to a building."))
```

Relationships, key-attributes, and attributes of an entity-type module have property-information in the form of nested lists. The top level in the case of a relationship contains the module entity-type role. The second level contains the related entity-type role. The third contains a list of one or more related entity types.

```
(RELATIONSHIP HS
     (PART-OF (HAS-PART (HVAC)))

(RELATIONSHIP HS
     (HAS-PART (PART-OF (COMPONENT CONTROL PATH)))
```

In the case of the key-attributes and the attributes, the top level contains the key-attribute or attribute. The second level contains the domain. The third contains a list of acceptable domain values or other constraints that apply to domain values. (Note: The values of a key attribute, i.e., the keys, begin with a letter since they are used as symbols to represent entities. A constraint in double quotation marks indicates a proper form for a value. Anything listed after "&" is optional.)

```
(KEY-ATTRIBUTE HS
     (HS-ID (BUILDING-NUMBER ("B-integer&-character")))

(ATTRIBUTE HS
      (TYPE (FUEL (ELECTRIC GAS OIL SOLAR)))

(ATTRIBUTE HS
       (EFFICIENCY (PERCENT ("integer"))))
```

The form for an entity-type interpreter is similar to that of a perspective translator. That is, an association list made up of a selector and a list containing a method.

```
(INTERPRETER HS
     (SEND-ATTRIBUTE (<method>)))

(INTERPRETER HS
     (RETRIEVE-ATTRIBUTE (<method>)))
```

Displaying the meta information once the properties have been entered into working memory involves the use of display functions.

The general form of a display function is:

```
(MIR-DISPLAY <system-ID>
  <property> <frame-identifier>)
```

The LISP interpreter returns the available meta information for the requested property. This approach to obtaining meta information applies to both perspectives and entity-type modules. An example using the attributes property follows:

```
(MIR-DISPLAY ENGINEERING-INFORMATION-SYSTEM
  ATTRIBUTES HS)
```

This form upon evaluation returns:

```
(TYPE (FUEL (ELECTRIC GAS OIL SOLAR)))
(EFFICIENCY (PERCENT ("integer")))
(INITIAL-COST (DOLLARS ("integer")))
(LOAD (MMBTU ("real")))
```

By successively querying the working memory, an understanding of an information system schema is achieved. As described previously, customized functions within a given programming environment are highly desirable. The Information Format for encoding information also makes use of functions that manipulate properties.

6.4 The Information Format

The Information Format is presented in Table 13. The Information Format is centered around a single frame of interest to the external user, the view. It is at a level conceptually that corresponds to a perspective in the Meta Information Format.

A view is identified by the view property. The view property contains the perspective-id which indicates the perspective from which the view has been derived. The use of this property in transfering information between systems is not compulsory. The following shows the general case for the view:

(VIEW <perspective-ID>)

The example building-economist view therefore begins with the following forms:

(VIEW BUILDING-ECONOMIST-PERSPECTIVE)

The properties that contain information about an entity are its relationships with other entities and its attributes that have values. Unlike entity-type modules of a perspective, entity information is not represented in frames identified by the entity but rather as individual constituent parts therof referenced by the view to which they belong. The entity information is

Table 13. The Forms of the Information Format.

View Module: Perspective-ID

```
(VIEW Perspective-ID)

(VIEW-RELATIONSHIP Perspective-ID
      ((Module-Entity-Type (Key ...))
       Module-Entity-Role
       (Related-Entity-Type (Key ...))))

(VIEW-ATTRIBUTE Perspective-ID
      ((Entity-Type (Key ...))
       Attribute
       (Domain (Value ...))))
```

Messenger: Messenger

```
(Messenger Perspective-ID
     (Selector
      (Argument ...)))
```

represented as a data triplet containing the entity, an information identifier, and an information list. The data triplet provides for flexible manipulation of information and translation to other representations. The grouping of information by entity can be accomplished by a given information system in the context of the perspective from which it is viewed or its own internal represention as appropriate. The general form is:

```
(VIEW-<property> <frame-identifier>
     (<entity>
      <information-identifier
      <information-list>))
```

The form for a relationship is:

```
(VIEW-RELATIONSHIP Perspective-ID
     ((Entity-Type (Key ...))
      Entity-Role
      ((Related-Entity-Type (Key ...)) ...)))
```

The form for an attribute is:

```
(VIEW-ATTRIBUTE Perspective-ID
     ((Entity-Type (Key ...))
      Attribute
      (Domain (Value ...))))
```

The forms differ in terms of the elements that are supplied in the positions of the data triplet. The entity is the first element. It is represented as a nested list containing an

entity-type at the top level and one or more keys at the second level. Either an attribute of the entity or the roles played by the entities of a relationship constitute the second element. The third element is a nested list containing information about either the attribute or the related entities of the referenced relationship.

In the case of an attribute, the third element contains the domain in the top level of nesting and the attribute value (or values) in the second. An example follows:

```
(VIEW-ATTRIBUTE BUILDING-ECONOMIST-PERSPECTIVE
     ((HS (B1A))
      TYPE
      (FUEL (OIL))))
```

In the case of a relationship, the third element consists of a list of the related entities. The entities are stored in terms of their constituent parts. Therefore, the top level of the list contains the entity type while the second level contains a list of the one or more keys that when considered in conjunction with the entity type identify an entity. An example for the building-economist-view takes the form:

```
(VIEW-RELATIONSHIP BUILDING-ECONOMIST-PERSPECTIVE
     ((HS (B1A))
      PART-OF
      ((HVAC (B1)))))
```

It is the messenger that provides information in the Information Format to a remote user. The messenger uses available methods in response to messages requesting information to construct the appropriate view.

6.5 The Messenger and its Messages

Each MIR has a messenger developed to work specifically with the information system of which it is a part. The messenger is a function written to provide input and output for messages and subsequent views. It also provides for the initial delegation of responsibility for a message to an appropriate perspective translator. The actual code used for a messenger is specific not only to the information system of which it is a part but also the particular programming environment used (see Appendix).

The general form by which a message is sent via a messenger is as follows:

```
(<messenger-ID> <perspective-ID>
     (<selector>
      (<argument> ...)))
```

The messenger-ID is the name of the function which usually identifies the information system of which it is a part (e.g.,

engineering-system-messenger). The second element is the perspective-ID which identifies the perspective from which the information is being sought. This is followed by a message. The message in the form of an association list has as its first element the selector which specifies the kind of information being sought. The associated element is a list of arguments required by the selector. The following is an example of a request for information from the building-economist-perspective that is part of the engineering system MIR:

```
(ENGINEERING-SYSTEM-MESSENGER
     BUILDING-ECONOMIST-PERSPECTIVE
     (SEND-ATTRIBUTE
      ((HS (B1A)) TYPE)))
```

Table 14 presents four example messages (selectors and the corresponding arguments) that can be used in writing messages. The first is a message that requests the keys (key-attribute values) that exist for a given entity-type. Once the keys are known, information concerning a given entity can be requested. The other two messages request information regarding relationships and attributes. Selectors and the associated methods can be developed to be responsive to the way a given information system is to be used.

Table 14. Example Messages: Selectors and Arguments.

Selector	Arguments
(SEND-KEYS	Entity-Type)
(SEND-RELATIONSHIP	((Entity-Type (Key ...)) Entity-Role))
(SEND-ATTRIBUTE	((Entity-Type (Key ...)) Attribute))

7. Conclusions

Modular Semantic Application Layer Protocols have been presented as a means of transmitting both meta information and information between incompatible information management systems. As a first step toward the development of such protocols, Meta Information and Information Formats have been presented which (1) make use of modular frame-based knowledge representations to capture declarative, compositional, and procedural semantics of an information system, and (2) provide an object-oriented programing capability for intercommunication with the information system or application.

7.1 The Relations Implicit in MS-ALPs Formats

The use of a frame-based representation for meta information includes two implicit relations (Table 15). The first is classification with the roles instance-of and has-instance. Classification is essentially the relation upon which the concept of a frame is based. Therefore in the MS-ALPs Format, an entity-type identifies a class of which an entity is an instance. (Entity-type has-instance entity and entity [is-an] instance-of entity-type.) The fact that knowledge about a class is applicable to all instances of that class is to say that classification is a relation that involves inheritance.

Table 15. The Relations and Roles Implicit in MS-ALPs.

RELATIONS:	ROLES:	
Classification	instance-of	has-instance
Characterization	attribute-of	has-attribute

The MS-ALPs approach to representing meta information includes inheritance only in its frame-based representation (i.e., the relation of classification implicit in frames). Generalization often involves inheritance between one entity-type and another as well. And, of course, user defined relations may also be employed that involve inheritance. MS-ALPs supports inheritance only in terms of its use of frames. Where inheritance is used by an information system for relations other than classification, each MS-ALP module involved must repeat explicitly the relationships and attributes without reference to any other module (i.e., the modules are completely self contained). This approach maintains the modularity of the system such that perspectives and views can be constructed without regard to the issue of inheritance. This decision will need review as MS-ALPs is subjected to further testing and use.

The second relation implicit in the modular approach of MS-ALPs can be referred to as characterization. Characterization has the roles attribute-of and has-attribute. Since attributes and key attributes are properties of both meta information and information modules there is typically no need to include this relation in the relationships property of the modules. The possible exception to this rule is in those cases where relationships may exist between attributes, the attributes would themselves need to appear as entity-types and therefore would be explicitly listed using the characterization relation.

The implicit inclusion of the relations classification and characterization in the frame-based representation used by MS-ALPs appears to be reasonable within the context of the building industry. Alternative representations, notably the Semantic Association Model [13], require explicit use of the characterization relation. MS-ALPs, however, has been developed as a frame-based approach to information modelling which implicitly includes characterization to provide a convenient mechanism for not only the representation of declarative semantics in a way that is intuitively familiar to the user but also compositional and procedural semantics through its use of object oriented programming.

7.2 Current Implementation

Intercommunication between incompatible information systems and applications using MS-ALPs makes use of message passing capabilities that initiate sytem operations without the necessity of a detailed understanding of those operations. This constitutes a knowledge-based approach to a "virtual extension" of the system operations. This extension occurs within the context of the Meta Information and Information Formats. Users that do not have a detailed understanding of system operations are able to make use of these "self-descriptive" formats to access information required by their applications and to further adapt the contents of these formats to their particular needs.

This type of intercommunication between incompatible systems has been accomplished within LISP environments. Interest within the building industry in knowledge-based systems, such as information systems having deductive capabilities and expert systems, will continue to focus the development of MS-ALPs within this environment.

The current implementation of MS-ALPs within the Computer Integrated Construction Group demonstrates communications capabilities operating simultaneously (in background) with knowledge-based systems on both multitasking workstations and LISP machines. The implementation provides for concurrent and cooperative operation of applications interacting directly via MS-ALPs in order to accomplish joint goals. It provides a prototype for the development of MS-ALPs capabilities on other systems as well as a demonstration of a complete working implementation of an interface using MS-ALPs.

References

[1] Palmer, M.E.; The Current Ability of the Architecture, Engineering, and Construction Industry to Exchange CAD Data Sets Digitally, NBSIR 86-3476, National Bureau of Standards, U.S. Dept. of Commerce, Gaithersburg, MD, 1986.

[2] Tsichritzis, D.C. and Lochovsky, F.H.; Data Base Management Systems, Academic Press, NY, 1977.

[3] Brodie, M., Mylopoulos, J., and Schmidt, J.; On Conceptual Modeling, Springer-Verlag, NY, 1984.

[4] Date, C.J.; An Introduction to Database Systems, Vols. I & II, Addison-Wesely Publishing Comapany, Reading, Mass, 1986.

[5] Gligor, V.D. and Luckenbaugh, G.L.; Interconnecting heterogeneous database management systems, in Tutorial: Distributed Database Management, J.A. Larson and S. Rahimi (eds), IEEE Computer Society EH0222-0, IEEE, NY, 13-23, 1985.

[6] Adiba, M. and Portal, D.; A cooperation system for heterogeneous data base management systems, Inform. Systems, 3, 209-215, 1978.

[7] Cardenas, A.F. and Pirahesh, M.H.; Data base communication in a heterogeneous data base management system network, Inform. Systems, 5, 55-79, 1980.

[8] Chen, P.P.; The entity-relationship model - Toward a unified view of data, in Tutorial: Centralized and Distributed Data Base Systems, W.W. Chen and P.P. Chen (eds), IEEE Computer Society #EHO 154-5, IEEE, NY, 166-193, 1979.

[9] Smith, J.M., Bernstein, P.A., Dayal, U., Goodman, N., Landers, T., Lin K.W.T., and Wong, E.; Multibase -- Integrating heterogeneous distributed database systems, National Computer Conference, 487-499, 1981.

[10] Shipman, D.W.; The Functional Data Model and the data language Daplex, ACM Transactions on Database Systems, 6(1), 140-173, 1981.

[11] Integrated Computer-Aided Manufacturing (ICAM) Information Modeling Manual (IDEF1), UM 110231200, Air Force Wright Aeronautical Laboratories, Wright-Patterson Air Force Basce, OH, 1981.

[12] Alashqur, A. and Su S.Y.W.; An Interactive Data Dictionary System for Computer Integrated Manufacturing Applications, DBRDC-NBS-84-04, Database Systems Research and Development Center, Univ. of Florida, FL, 1984.

[13] Su, S.Y.W.; Modeling integrated manufacturing data with SAM*, Computer, 34-49, Jan. 1986.

[14] Rehak, D.R. and Howard H.C.; Interfacing expert systems with design databases in integrated CAD systems, Computer-Aided Design, 7(9), 443-454, 1985.

[15] Tanenbaum, A.S.; Network Protocols, in Tutorial: Distributed Database Management, J.A. Larson and S. Rahimi (eds), IEEE Computer Society EH0222-0, IEEE, NY, 433-469, 1985.

[16] Barr, A. and Feigenbaum, E.A.; The Handbook of Artificial Intelligence, Vol. 1, HeurisTech Press, Stanford, CA., 1981.

[17] Charniak, E. and McDermott, D.; Introduction to Artificial Intelligence, Addison-Wesley Publishing Company, Reading, MA, 1985.

[18] Rich, E.; Artificial Intelligence, McGraw-Hill Book Company, NY, 1983.

[19] Minsky, M.A.; A framework for representing knowledge, In P. Winston (ed.), The Psychology of Computer Vision, McGraw-Hill Book Company, NY, 1975.

[20] Winston, P.H. and Horn, B.K.P.; LISP, Second Edition, Addison-Wesley Publishing Company, Reading, MA, 1984.

[21] Smith, J.M. and Smith, D.C.; Database Abstraction: Aggregation and Generalization, ACM Trans. Database Systems, 2(2), 105-133, 1977.

[22] Borkin, S.A.; Data Models: A Semantic Approach for Datbase Systems, The MIT Press, Cambridge, MA, 1980.

[23] Brodie, M.L. and Mylopoulos, J.; On Knowledge Base Management Systems, Springer-Verlag, NY, 1986.

[24] Wilensky, R.; LISPcraft, W.W. Norton & Company, NY, 1984.

Appendix

Demonstration Messenger, Translators, and Interpreters

ENGINEERING-SYSTEM-MESSENGER

A messenger function that delivers a message requesting the value of a specific attribute of an entity.

DELEGATE-TO

A function that transfers control of program execution from one object to another (i.e., translators or interpreters).

BUILDING-ECONOMIST-PERSPECTIVE TRANSLATORS

Perspective translators that receive messages and delegate them to module interpreters, access the local information system, and send replies to a requesting system.

HS INTERPRETERS

Module interpreters that supply entity type specific meta information of use in accessing information.

This appendix presents messenger and delegate-to functions as well as several translators and an interpreter. These constitute examples of the dynamic elements of the MS-ALPs Formats. Since the dynamic elements of MS-ALPs necessarily involve code that is system as well as programmer specific, their presentation serves to demonstrate the principles rather than to recommend a particular approach to their implementation. The dialect of LISP used in these examples is a version of Standard Portable LISP. The demonstration continues the example used throughout the document involving an engineering information system and a building-economist-perspective that has been developed as an extension to that system.

For communication to be possible using MS-ALPs, an information system must provide a messenger, in this case the engineering-system-messenger. Table 16 presents a simplified LISP function for this purpose. The function takes two arguments, the perspective-ID which identifies the perspective being used to view the data and a message. The message is in turn comprised of a selector and its arguments. An example using this messenger function to access the value of an attribute follows:

```
(ENGINEERING-SYSTEM-MESSENGER
     BUILDING-ECONOMIST-PERSPECTIVE
     ((SEND-ATTRIBUTE
       (((HS (B1A)) TYPE))))
```

Table 16. Example Messenger and Delegate-to Functions.

```
(de ENGINEERING-SYSTEM-MESSENGER (perspective-id message)

        (open "output/reply.lsp" 'output)
        (let ((selector (caar message))
              (arguments (cadar message)))
          (delegate-to perspective-id 'translators
                       selector arguments))
        (close 7))
```

```
(de DELEGATE-TO (frame property selector arguments)

        (apply
          (cadr
            (assoc selector
                   (retrieve property frame)))
          arguments))
```

The communication requests that the engineering-system-messenger send the type of the B1A heating system using the building-economist-perspective. The process by which the request is fulfilled is initiated by the messenger. First, the engineering-system-messenger (Table 16) opens a file in the first line. (To demonstrate the basic principles involved, the messenger writes a reply to file but does not transfer the file to requesting system translators.) In the second and third lines, the messenger identifies salient aspects of the message (the selector and its arguments). These are then delegated-to the translators of the appropriate perspective as specified by the perspective-ID. After a reply has been generated by perspective translators and module interpreters, the messenger will close the output file.

Delegating responsibility for a message is performed by the delegate-to function (Table 16). This function is used to advance the object-oriented program. It causes the method associated with a given selector to be evaluated. Lines 2-4 are used to return the associated method. The method is applied using the list of arguments supplied.

In the case of the first use of the delegate-to function by the messenger, the selector and its arguments are delegated to the perspective-ID (the frame) translators (the property). Therefore, the method associated with the send-attribute selector

will be returned from the translators of the building-economist-perspective. The send-attribute method will then be applied using the arguments for the entity-type, key, and attribute specified (HS, B1A, and type).

Table 17 presents example translators for the building-economist-perspective. The first translator has a selector which matches that of the request, and therefore it is the method associated with this selector that is applied. The lambda form (i.e., the method) first identifies the salient elements of the arguments. It then concatenates the entity-type and key to form the entity for which an attribute value is sought (HS (B1A)). The delegate-to function is then used to transfer control to the appropriate interpreters (those of the HS module).

Table 18 presents the interpreters property. The example interpreter has a selector which matches that of the message delegated to the interpreters. There the associated method is applied. The interpreter proceeds differently depending on the exact nature of the request. In those cases where the attribute being accessed is either the type, efficiency, or the initial-cost, the message is delegated to the retrieve-attribute method with an added argument indicating that retrieval should be from an hvac-database. If the attribute being accessed is the load, retrieval must be from a thermal-analysis-database and the message is delegated with that knowledge supplied. If the

Table 17. Example Translators.

```
(TRANSLATORS IN BUILDING-ECONOMIST-PERSPECTIVE

   ((SEND-ATTRIBUTE                          %  Selector

     (lambda (arguments)                     %  Lambda Form
        (let*
           (entity-type (car arguments))
           (key (car (cadr arguments)))
           (attribute (car (cadadr arguments)))
           (entity (implode (append (explode entity-type)
                                    (append (explode '_)
                                      (explode key)))))
           (value nil))
         (delegate-to entity-type
                      'interpreters
                      'send-attribute
                      (list entity attribute value))))


    (RETRIEVE-ATTRIBUTE                      %  Selector

      (lambda (database                      %  Lambda Form
               entity attribute)
        (let* ((value (cadaar
                        (backward-chain
                          '(,attribute ,entity ?x)
                          database))))
          (delegate-to 'building-economist-perspective
                       'translators
                       'print-attribute-to-file
                       (list entity attribute value)))))


    (PRINT-ATTRIBUTE-TO-FILE                 %  Selector

      (lambda (entity attribute value)       %  Lambda Form
        (channelprintf 7
          "%n   (VIEW-ATTRIBUTE (%w %w %w)%n"
          entity attribute value)))))
```

Table 18. Example Interpreter.

```
(INTERPRETERS IN HS

   ((SEND-ATTRIBUTE                          %  Selector

      (lambda (entity attribute)             %  Lambda Form
        (cond

          ((or (equal attribute 'type)
               (equal attribute 'efficiency)
               (equal attribute 'initial-cost))
           (delegate-to 'building-economist-perspective
                        'translators
                        'retrieve-attribute
                        (list 'hvac-database
                              entity attribute)))

          ((equal attribute 'load)
           (delegate-to 'building-economist-perspective
                        'translators
                        'retrieve-attribute
                        (list 'thermal-analysis-database
                              entity attribute)))
          (t
           (delegate-to 'building-economist-perspective
                        'translators
                        'print-attribute-to-file
                        (list entity attribute
                         "Not in perspective!")))))))))
```

attribute is not any of the four listed, the message is delegated to the translator of the building-economist-perspective with a print-attribute-to-file selector with a value of "Not in perspective!" added to the argument list.

The delegation of control to the retrieve-attribute translator of the building-economist-perspective causes the associated method to be applied to the message. The lambda form invokes the backward chaining mechanism of a deductive retriever [24] within the specified database of the engineering-information-system. Once the value has been retrieved control is delegated to the lambda form responsible for printing the attribute value to the output file. Upon completion, the messenger closes the file. Transmission of the file to appropriate MIR translators at the requesting site would then perform necessary conversions from MS-ALPs format to that appropriate to the remote information system.

NBS-114A (REV. 2-8C)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET (See instructions)	1. PUBLICATION OR REPORT NO.	2. Performing Organ. Report No.	3. Publication Date

4. TITLE AND SUBTITLE

The Use of Artificial Intelligence Programming Techniques for Communication Between Incompatible Building Information Systems

5. AUTHOR(S)

William F. Danner

6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions)

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, D.C. 20234

7. Contract/Grant No.

8. Type of Report & Period Covered

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (Street, City, State, ZIP)

10. SUPPLEMENTARY NOTES

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)

A communication capability between incompatible information systems is presented. The purpose of the research reported here has been to develop an interface based on (1) a format for the exchange of knowledge needed by each system to understand the other, and (2) a format for the exchange of information in the context of that knowledge. Particular emphasis has been placed on developing protocols supporting the transfer of analytical data. These data are seen as comprising not only facts but also the semantics associated with those facts.

Two artificial intelligence programming techniques have been employed: (1) frame-based knowledge representation, and (2) object-oriented programming capabilities as an integral part of the frame-based representation. These techniques make self-descriptive formats possible that provide for a virtual extension of an information management system. Such an extension provides access to information without requiring a detailed understanding of specific system operations.

12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)

artificial intelligence; building; communication; data; database; database management system; information; information system; protocol

13. AVAILABILITY

☐ Unlimited

☐ For Official Distribution. Do Not Release to NTIS

☐ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.

☐ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES

15. Price