An Information Exchange Protocol for Constructed Facilities

by

M. K. Zamanian, S. J. Fenves

12-37-90 *C 0* 3

An Information Exchange Protocol for Constructed Facilities

M. Kiumarse Zamanian
Steven J. Fenves

April 1990
EDRC - Carnegie Mellon University

This work has been supported by the Engineering Design Research Center, a NSF Engineering Research Center.

Abstract

The principal motivation of this study is to provide an open architecture for the exchange of information among the many participants in the planning, design, construction, and management of constructed facilities. Since the participants are dispersed both geographically and organizationally, and are likely to use various computer systems and languages, the exchange protocol must be designed to be accessible to all. This document presents the preliminary findings of this research in the specific area of spatial decomposition and modeling of facilities. Conceptual models for representation of spaces and their organizations are discussed and two prototype programs, based on the non-manifold geometric modeling system NOODLES, are described. It is expected that non-geometric attributes can be readily augmented to the spatial representation during the remaining part of this work.

1 Overview

This document provides a progress report on the initial phase of research on a proposed information exchange protocol for constructed facilities. Two alternatives were originally proposed for the development of this protocol [1]: a description interface, in the form a high-level textual language, to express the facility data in an ASCII file; and a functional interface, consisting of a set of data definition and retrieval functions, to interact with an encapsulated data representation that contains the facility data. The primary goals of this report are to communicate some preliminary findings to the research community as well as to discuss more focused directions of the research proposed initially.

2 Research Activities

The activities to date have consisted of three areas:

- striving for a formal model to represent the relationships among the various components of a constructed facility;
- devising a general spatial decomposition scheme of constructed facilities; and
- developing two prototype modeling programs for defining and retrieving a wide variety of spaiial information about different abstractions and types of facility components.

These three areas are by no means disjoint; rather, they share fundamental modeling and reprcscmaiion concepts needed to satisfy the diverse requirements of various applications involved during the life-cycle of a facility. However, this organization is adopted to provide a more clear separation between the conceptual ideas and the implementation tasks of this work. The following sections briefly present these and other relevant areas.

2.1 A Graph-based Model of Facility Component Relationships

Components of a constructed facility are related in many different ways when viewed from different functional or descriptive views. These relationships can be of several types; for example, when two components are physically connected, these components are related via a "connected-to" relationship, or when a component is a composition of some lower-level components, the low-level components arc said to be "part-of" the higher level component In addition to these more general types of relationships, each application may require more specific types of relations between the components that it uses. For exam pic. a distribution system, such as HVAC, may require a relationship like "in-the-same-zone" in order to classi i y those spaces that are served by a particular mechanical system. Although these more specific relationships represent specialized versions of the more general types of relationships, they provide the specifit scmant K s needed by a particular application for a formal description of a facilty.

The relationship between two components is described with a triplet data structure, i.e., the identifiers of those components and the relation between them. Thus, it is possible to provide a general m<xkl for describing the relationships between various facility components regardless of the specific types oi relationships between them. A graph whose nodes represent various levels and abstractions of the components and whose branches indicate relationships between those components provides the structure of this general model. Each node of this graph can be further "expanded" to reveal a more detailed graph of the lower-level components that comprise the expanded node. Using this node expansion facility, on application can view the components of the facility at any desired level of granularity or abstraction, while maintaining the overall topology of the parent graphs. For example, consider a graph that represents the

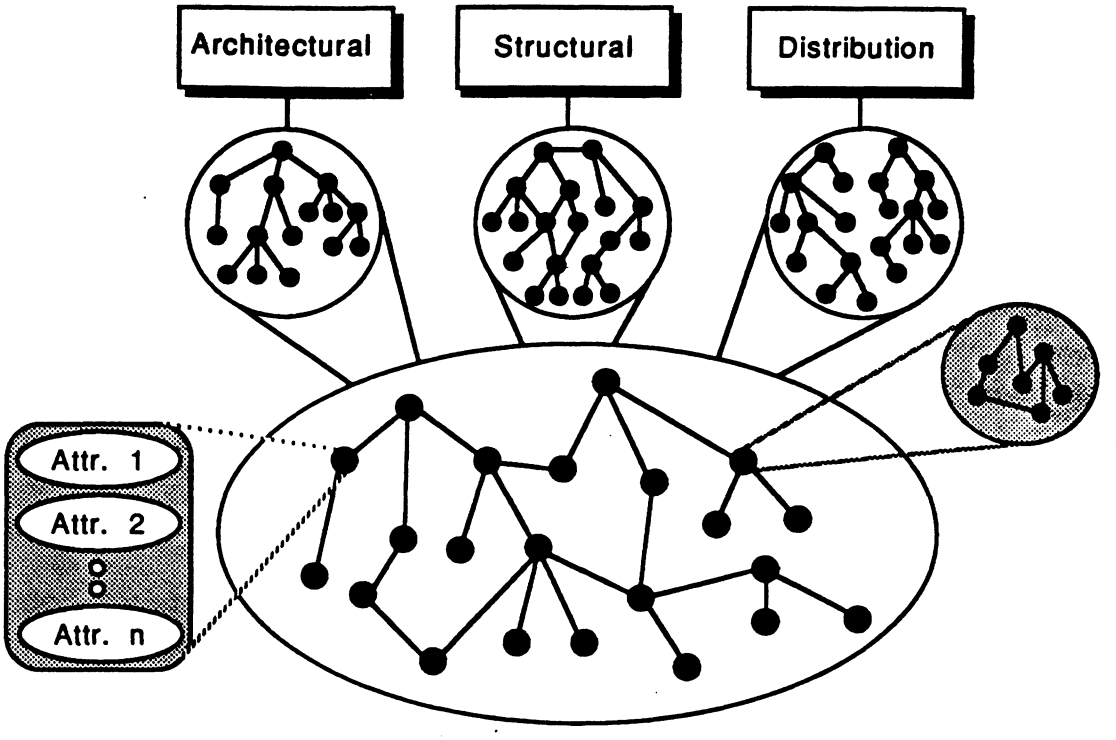

Figure 1: A Graph-based Model of Facility Components

relationships among several apartments and offices in a facility. Each apartment or office can in turn be expanded into a separate graph to show the relationships between its components, such as bedrooms, closets, or bathrooms, without altering the original graph.

Figure 1 illustrates the overall structure of the proposed model. This Figure reveals several important characteristics of this model which are summarized below:

- the underlying organization of facility components is represented by a general graph-based model, where a node identifies a component and a link corresponds to a particular relation between the two connected nodes;
- each application has a more specialized functional view of the components and their relationships to serve its particular needs;
- a node can be recursively expanded into a graph that describes that component's sub-parts and their relations; and
- in addition to the relationships represented by the links of the graphs, a number of non-spatial attributes, such as geometry and various application-dependent functional and behavioral attributes, are associated with each node.

The remaining part of this report is devoted primarily to the spatial modeling of constructed facilities using the general model presented above. Although Section 3.2 briefly discusses some high-level issues about the non-spatial attributes of the facility components, these issues will be considered in greater depth in the remaining portion of this work.

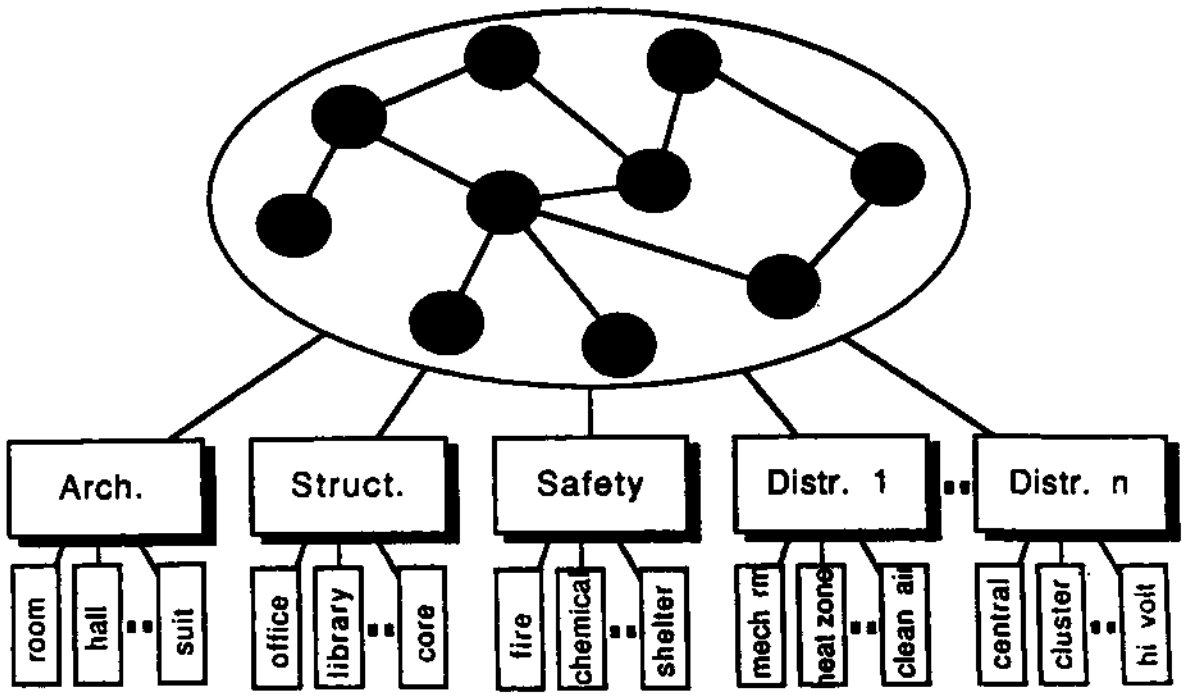

Figure 2: Spatial Decomposition of a Facility

2.2 Spatial Modeling of Constructed Facilities

Every application involved in a particular activity during the life-cycle of a constructed facility is concerned with some form of a spatial decomposition of that facility. Although the spatial decomposition will, in general, differ from one application to another, one can safely assert that each application uses the concept of a "space" that is separated from its neighboring spaces by one or more "partitions." For example, an apaitment is a residential space used by an architectural application, a library may be referred to as a space with a stated live load by a structural application, and a zone is a space used by an HVAC application. Using the graph-based model presented in the previous section, a node in the graph can similarly represent a space, while the branches of the graph indicate the spatial relationships, such as adjacency, between the spaces. Figure 2 illustrates this concept by showing the overall spatial decomposition of a facility.

Each application has its own spatial arrangements and decompositions of spaces that are suitable for representing its particular requirements. Usually an application has a more specific arrangement of spaces than the general graph-based model; however, the more specific arrangement is generally a subset of the general graph-based model. For example, consider the spatial decomposition of a residential apartment building from an architectural perspective. In this example, a hierarchical decomposition of spaces, such as a building being composed of a utility and access core and several floors, the floors consisting of apartments and common areas, and the apartments being made up of rooms, kitchen, etc., is a relatively simple but commonly used hierarchical model. One must note that the core, an apartment, or a kitchen are simply labeled spaces that are also used, either as the same space or as a subset other spaces, by applications other than architecture. Continuing with the above example, a fire zone may cover several apartments and common areas such as lobbies. Although the fire zone, which is a space used by a safety application, may or may not be explicitly composed of specific architectural spaces, it refers to a space that is certainly shared by the architectural, mechanical, and other applications. Thus, a particular space can be implicitly

or explicitly composed of many different types or abstractions of other spaces.

Although a space generally refers to a three-dimensional physical object, it is often represented by lower dimensionalities to simplify the representation of that object Such simplifications provide better abstractions by revealing only those attributes of a space that are important to particular applications and thus hide the unnecessary details. For example, the center-plane of a floor system or the center-line of a column generally provide sufficient spatial information about the three-dimensional spaces occupied by these objects during the preliminary architectural or structural design phases. Furthermore, it is common to have representations of mixed dimensionality in the same model of a facility; as is the case for one-dimensional columns and pipes, two-dimensional walls and floor systems, and three-dimensional rooms and fire zones. As a result, it is necessary to have a spatial model of a facility in which any application can represent the geometric attributes of and topological relations among spaces uniformly and consistently regardless of the level of dimensionality used for the representation of those spaces. This simply means that an application should be capable of defining and/or retrieving spatial information about a space (or an object in a general sense) at any level of dimensionality it desires.

Some of these issues and others related to the specification and the retrieval of spatial information have been investigated in the initial phase of this work, and the results are briefly discussed in the remaining part of this report.

Spatial information of a facility can be defined as the topological relations among and the geometric attributes of abstractions of spaces in that facility, where a space corresponds to an approximation of the region occupied by a physical or a virtual component of the facility. A space can be either void, such as an empty closet, or occupied (partially or fully), such as a pipe filled with fluid. As discussed earlier, spaces can be recursively decomposed until a desired level of granularity is obtained, and furthermore, spaces are separated from one another by means of partitions. A partition may or may not have a physical realization; moreover, the representation of a partition may correspond to an abstraction of another space. For example, the kitchen and the living room in an apartment correspond to two separate spaces and are separated by either a physical (a complete or a partial wall) or a virtual partition. A physical partition, such as a wall, can be further treated as either a space whose two sides separate it from its neighboring spaces or a simple planar face that is shared by the two adjacent spaces. Thus, depending on the level of granularity of spatial information available or needed about a facility, spaces and partitions are represented by different abstractions of geometric entities. For example, a space representing a room can be a three-dimensional cuboid or a two-dimensional rectangle corresponding to a projection of that room onto a specific plane. Similarly, a wall can be a complete three-dimensional block, or a two dimensional rectangle (and even a one-dimensional line) representing the center-plane of that wall.

3 Objectives for Information Exchange Protocol

3.1 Spatial Information

The applications involved throughout the life-cycle of a constnicted facility define and retrieve the spaiial information of the facility's spaces (or objects) in various fashions. Therefore, an information exchange protocol must be able to accommodate these applications with the most appropriate data specification and retrieval facilities suited for their needs. To meet this challenge, a number of objectives have been identi (led:

Object definition. Allow any application to define the spatial information of objects (spaces) by any combination of:

1. independent geometrical definition with inferred topological relations; or
2. definition in terms of the topological relations of objects among which at least one is defined geometrically.

Object identification. Provide a naming mechanism to label unique objects in a facility such that they can be accessed from any application. These identifiers will also be used as access keys to database(s) of the non-spatial attributes of objects.

Compositional polymorphism. Exploit any level of spatial composition of the objects regardless of the level of dimensionality and the type of geometry involved.

Topological polymorphism. Make topological relations invariant with respect to dimensionality of objects involved.

The following paragraphs briefly elaborate on these objectives.

Object Definition. The spatial information of an object is defined in terms of its geometric and topological attributes. These attributes are either defined explicitly by an application or specified in terms of the attributes of other objects. For example, an HVAC zone may be defined either explicitly by its shape, dimensions, and orientation in space, or in terms of existing spaces, such as rooms or corridors, that are defined by other applications. Furthermore, it is often desirable to use only parts of other objects together to construct a new object. For example, an architectural application may create a new wall from the intersection of two adjacent rooms or model a hallway by the region of space enclosed by the walls and the floor systems of a building. Regardless of how the spatial information of an object is defined, the topological relationships between the various objects of a facility must be readily available to any application's request.

Object Identification. Each application must be able to define and retrieve the spatial information of an object by an application-defined name. Two spatially equivalent objects, i.e., having the same geometry and topology, can have different names, so long as they belong to different "classes" of objects. Therefore, in order to identify a unique object, a name and a class are needed. The class attribute of an object also provides a mechanism to reduce the search space when only spatial information about objects of a certain class (or certain classes) are desired. For example, an application may require the names of all objects of class "pipes" that are within objects of class "rooms" and within the object "zone-1" of class "fire-zone."

The name and the class of an object can be used directly in an external information management system, such as a database or a hypermedia, to define or access the non-spatial attributes of that object. The proposed information exchange protocol is a functional interface to this information management facility that uses the non-manifold geometric modeling system NOODLES [2] for storing and retrieving all spatial information of the facility components.

Compositional Polymorphism. The ability to compose various components or systems of components from objects of different dimensionality was briefly discussed above and is further emphasized here. The existing geometric modeling systems used in the domain of constructed facilities are of two general types: systems that are based on the wire-frame technology and used primarily for simple computer-aided drafting, design, and/or analysis tasks; and more sophisticated systems that use a CSG or a B-rep solid modeling technique for detailed and complex design, analysis, and/or rendering of facilities. While the first group of systems does not have the representational capability of modeling three-dimensional objects non-ambiguously, the latter group guarantees the validity of the generated models by limiting their users to model *only* valid three-dimensional representations of physical objects. Both groups of modeling systems are clearly very restrictive in terms of modeling various abstractions of components throughout the facility's life-cycle. On the one hand, high-level abstractions of objects generally require much simpler representations than detailed solid model of those objects, and on other hand, a detailed representation of

an object requires a valid solid model that provides the volumetric as well as surface information of that object

Based on the above discussion, one of the major objectives of the proposed protocol is to allow any application to define and retrieve the spatial information of the components of a constructed facility at any desired level of abstraction or detail. By using the NOODLES non-manifold geometric modeling system [2], representations of different dimensionality can harmoniously coexist in the same model. Furthermore, an application can exploit any level of spatial composition of the objects, via the functional interface of the proposed protocol, regardless of the level of dimensionality used for representing those objects. One of the main advantages of this approach is that objects of lower-level dimensionality can generally be extended to higher-levels of dimensionality by producing their approximate enclosing envelopes from the available geometric attributes. For example, the line representation of a girder, often used in simple structural analysis programs, can be extruded to a two-dimensional rectangle by using the specified cross-section of the girder and direction of the extrusion. The rectangular representation of the girder can in turn be used for interference checking with distribution elements, such as pipes or cables. Furthermore, the girder stiffners can be represented as nodes on the one-dimensional, or line segments on the two-dimensional representation of the girder, respectively. Thus, depending on the level of approximation at which the spatial information is needed, various abstractions, with different dimensionalities, of a facility are composed. The next section discusses in more detail the topological relations between objects represented at different levels dimensionality.

Topological Polymorphism. As discussed earlier, spatial information of objects generally pertains to simplified models of the physical objects. These simplified models are often composed of two- or one-dimensional geometric entities which are abstractions of the three-dimensional models of the objects. In order to be able to retrieve the desired topological relations between models of different dimensionality, it is necessary to make these relations invariant with respect to the dimensionality of models involved. For example, one must be able to retrieve topological relations, such as intersection or part-of, between an HVAC piping network that is represented by either a connected set of one-dimensional lines or a set of solid objects and the structural floor systems that are modeled by either planar rectangles or three-dimensional cuboids.

Although the use of a non-manifold geometric modeling scheme is essential to address the topological polymorphism objective, it does not address this issue completely. This is due to the fact that regardless of the type and the accuracy of a modeling system, a geometric model is merely an approximation of the real physical objects, and furthermore, different applications always provide spatial information with various degrees of approximations. Therefore, for spatial reasoning about objects of different dimensionality defined with varying degrees of approximation, this work proposes to augment the existing set operators of the non-manifold geometric modeler to include a user-specified tolerance value.

To illustrate the overall idea behind the proposed augmented operators, a simple but general example is presented in Figure 3. Several cases representing the spatial relations of two floors and the floor system between them are considered in this example. The primary intent of this exercise is to show how the augmented operator could deduce the topological relationship of a floor system and its two neighboring floors by using only the spatial information available for these three objects. For simplicity, only the cross sectional views of these objects are shown here; however note that in all cases the floors are represented as two-dimensional rectangles, while the floor system is represented by either a two-dimensional rectangle or a one-dimensional line.

The first case considers the floor system to be a subset of the intersection of the two floors. Two situations may exist: first, the two floors are represented such that they overlap, and thus each includes the floor system; and second, the floors share an edge and the floor system happens to be represented by the

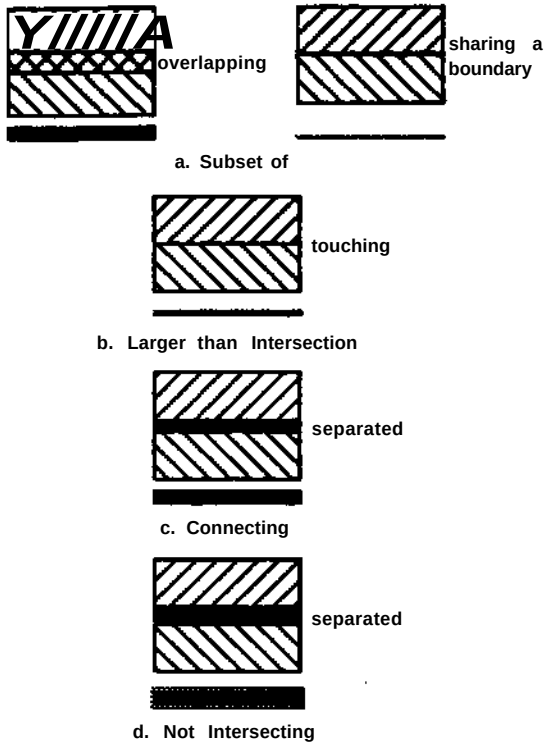

Figure 3: Representations of Two Floors and Their Joining Floor System

same edge. Computationally, this is the simplest case because one only needs to find out if the floor system is a subset of the object resulting from the intersection of the two floors. In case b, the floor system is a superset of the intersection of the two floors, and therefore a simple subset operation does not reveal the desired relationship. For this case, the user-specified tolerance value is used to construct a yirtual object. " such as a "tolerance-envelope," around the intersection of the two floors and then proceed to perform the subset operation between this object and the floor system as described in the first case. Cases c and d deal with situations when the two floors are disjoint. The third case assumes that the floor system connects the gap between the two floors, while the fourth case represents a situation where the floor system does not intersect either of the floors. In both cases, a virtual object representing the region that is "sandwiched" between the two floors must first be created and then be tested for inclusion in either the floor system object or the tolerance-envelope corresponding to the floor system object, respectively.

3.2 Non-spatial Attributes

As shown in Figure 1, each node of the graph representing the general information model of a constructed facility has several attributes associated with it These attributes vary for different types of components or abstractions used by various applications, or even by the same application during different phases of the facility's life-cycle. As described earlier in this document, the initial phase of this research has been primarily concerned with the spatial information of facilities, i.e., the components' topological relations and geometric attributes. However, the formal representation of the non-spatial attributes of facility components is an equally important aspect of the proposed information exchange protocol. Issues with regard to the information modeling techniques and the computer programs used for defining and retrieving the components* non-spatial attributes will be examined carefully later in this work, but at this time, some general aspects of these issues are worth considering.

The non-spatial attributes of a facility can be stored either with or separately from the spatial information. There are several issues associated with using either technique, most important of which is providing the most appropriate functional interface for defining and retrieving both types of information. The two types of information are distinctly different, and consequently, the operations used to define and to retrieve them also require different data structures and algorithms. As discussed earlier, the use of NOODLES greatly facilitates the management of the components' spatial attributes, however, it does not address the non-spatial attributes directly. Non-spatial attributes can be associated with the NOODLES entities, i.e., fundamental elements and point-sets, via an attribute handling mechanism that simply provides a generic link between an entity and the user-defined attributes [3]. The users, however, are entirely responsible for managing this information by implementing the necessary functions that can be automatically invoked when different events, such as "copy" or "split," are performed by the NOODLES kernel. In other words, in order to provide a relatively complete and robust facility in NOODLES for handling the non-spatial attributes of facility components, one must implement such a facility from the ground up.

Considering the time constraints of this work on the one hand and the availability of several existing information management systems, the most appropriate alternative appears to be the development of a functional interface to a hybrid system that combines the non-manifold geometric modeling of NOODLES with a general yet powerful information management system. Several types of information modeling techniques are available, among which, relational and object-oriented databases as well as the new, emerging hypermedia technology seem attractive [4]. However, the programming interfaces of such existing systems must first be carefully studied and evaluated before an appropriate system can be selected.

Using this hybrid approach, the non-spatial attributes of the NOODLES model of a facility will primarily consist of identification information, such as names and classes, for the components of that facility. This information (or parts of it) will also be used in the information management system that holds the non-spatial attributes of facility components, thus serving as a link between the two systems. Working under the assumption that components will belong to enumerated classes in a NOODLES model and each component will have a unique name in a class, the working hypothesis for now is that (using the vocabulary of relational data bases) a class name and an application name together identify the relations to be accessed for the non-spatial information, and the name identifies a unique tuple in that relation.

4 Implementation of Prototypes

This section briefly presents two prototype programs that are based on the C interface of the NOODI*d's* non-manifold geometric modeling system[3J. Both programs were developed primarily to experiment wuh the conceptual ideas presented above and also to become familiar with the functional interface and the overall implementation of NOODLES. The following sections discuss each of these programs individually and present some example runs and results.

4.1 A Geometric Modeling Program for Buildings

This program creates a non-manifold geometric model of a building using the user-specified spatial information of that building. The program automatically creates the architectural floor spaces, the structural frame and core, and the zones, pipes, and mechanical room of the HVAC system. The intent of ihis program is by no means to provide a general-purpose geometric modeling program for buildings; however, it provides a simple utility for modeling the various abstractions of components and systems of facilities by using the non-manifold capabilities of NOODLES.

Figure 4 presents a sample session of using the first prototype program. The user is first prompted 10 input the overall spatial information of the building, i.e., the three orthogonal dimensions, the number of stories, and the number of bays along the two axes of the horizontal plane. Next, the user is asked to label

```
code > building

  ********************************************************************
  *       W E L C O M E   T O   T H E   G E O M E T R I C            *
  *                                                                  *
  *  M O D E L I N G   P R O G R A M   F O R   B U I L D I N G S     *
  ********************************************************************

Building Width (> 0.) => 100
Building Depth (> 0.) => 80
Building Height (> 0.) => 45
No. of Stories (> 0) => 3
No. of Bays in Width (> 0) => 5
No. of Bays in Depth (> 0) => 3
Name of the Building => student_center
Name of the Core => elevator_shaft
Name of the Architectural system => floors
Name of the HVAC zones => hvac
Name of the HVAC pipe system => plumbing
Name of the HVAC mechanical room => heater
Name of the Structural System => frame
Name of the Columns system => columns
Name of the Beams system => beams
Name of file to store this model => building.ndl
code >
```

Figure 4: A Sample Run of the Geometric Modeling Program for Buildings

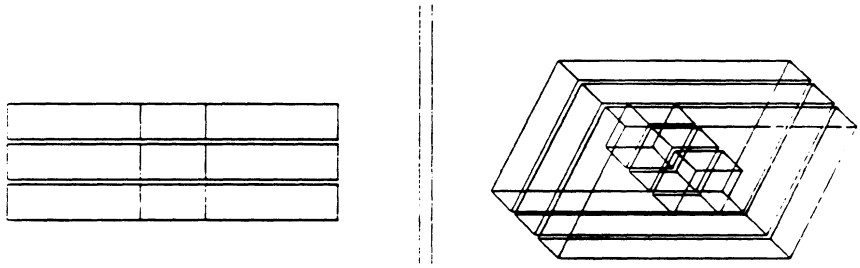

Figure 5: Model of the Architectural Spaces

various systems of the building with arbitrary names. These names can later be used in the second prototype program, to access their corresponding components. At present, this program assigns the class name of each system but the program can be easily modified to obtain this information from the user. Finally, the user is asked to input the name of an ASCII file in which the NOODLES model of the generated building is stored. This file can later be loaded into the second prototype program for various object manipulations. Figures 5 through 8 show the side and isometric views of the wire-frame representation of several systems of the building generated by the sample run presented in Figure 4.

As illustrated in these figures, the building generated by this program is composed of three main systems, i.e., architectural, structural, and mechanical. The architectural system is composed of floor spaces that exclude the core space and some space designated above each floor for placing the structural and distribution elements. The structural system is simply a rectangular grid frame with columns passing through every gird point and parallel beams connecting these columns above each floor space. The mechanical system contains two zones per floor (excluding the ground floor), where each zone corresponds to the right and left halves of a floor space. These zones are served by a distribution network that consist of several horizontal

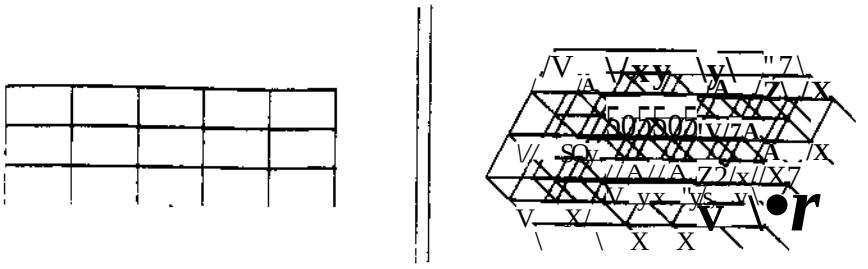

Figure 6: Model of the Structural Frame

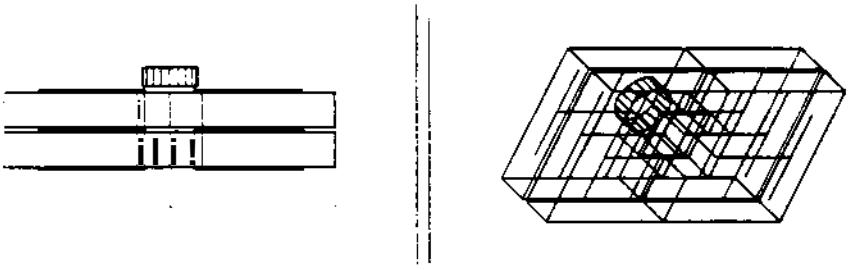

Figure 7: Model of the HVAC System

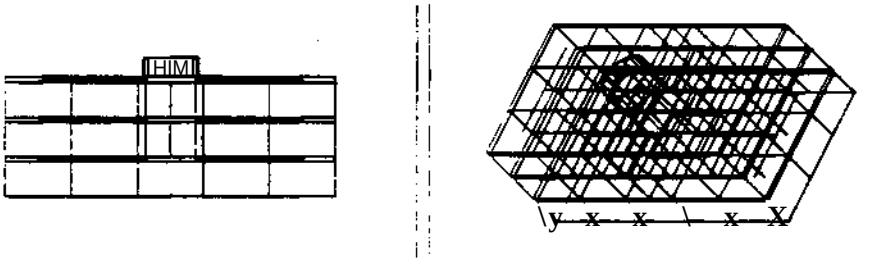

Figure 8: Model of the Complete Building

pipes and two vertical pipes that connect **the** horizontal **pipes** to **the** mechanical room located on top of the building. **The piping** system is set up such that it **does** not interfere with **the** architectural spaces, i.e., the horizontal **pipes pass** through **the** space assigned **above** each floor and **the** vertical pipes are run through the building core. As **shown** in the following section, the second prototype program can be used to check for interference **between the** piping network and the architectural system of this model.

4.2 FTOX

This section briefly describes an interactive, menu-driven spatial information management program that represents the first prototype of the proposed Functional Interface for Data exchange, FIDX. With this program one can define and retrieve the spatial information and topological relations of objects through various techniques and at any desired level of dimensionality. Furthermore, objects are defined and accessed by user-specified names and/or classes, and thus the underlying **data** organization of the NOODLES model is hidden from users. The program also provides a limited error handling facility to inform users of incorrect data input or menu selections. The following provides a brief overview of the capabilities of this program by going through the menu structure and a simple modeling exercise. This program starts with the following top-level menu:

```
code> fidx

  ************************************************************
  *              W E L C O M E T O F I D X                   *
  *                                                          *
  *        Functional  Interface  for  Data  Exchange        *
  ************************************************************

c: create objects
d: delete objects
1: list objects
g: geometry check
r: retrieve.relations
f: file handling
<ESC> : quit
 =>
```

The user can select any of these options by typing the first letter of its corresponding text. The following paragraphs describe each of the these options.

Create objects. Users can create a new object by selecting this menu. First the user is prompted for ihc name and the class of the object

```
Name of the object [<ESC> : quit] => rooml
Class of this object => rooms
```

After inputing the name and the class of the object, the user can define the spatial information of iJiis object by several methods presented in the following menu:

```
>» Input Data for Object Name <rooml> of Class <rooms> by :

g : geometry
s : subparts
b : boolean of existing objects
r : retry name
 => g
```

The user can explicitly specify the geometry of the object by selecting the first option. Upon selecting this option, the following menu is presented:

```
Type:
b: block
r: rectangle
1: line
 => b
```

For the block and the rectangle selections, the user specifies first the dimensions of the object and then the model space coordinates of a base-point (presently the bottom, lower left corner of the object) to position the object in three-dimensional space. The object can also be translated and rotated with respect to the local coordinate system whose origin is at the base-point and is aligned with the model space axes. For the line option, however, the user simply inputs the model space coordinates of the two end-points. Continuing with the above example, the block option is selected and the following information is input:

```
Width (> 0.) => 10
Depth (> 0.) => 7
Height (> 0.) => 5
Base Point (Bottom-Lower-Left Corner):
X => 0.0
Y => 0.0
Z => 0.0
Rotate? (y or n) => n
Translate? (y or n) => n
>»Defined Object <rooml> Geometrically.
```

After an object is completely defined, FIDX returns to the top-level menu. Objects can also be created as composition of their sub-part objects. In this case, the user can define the sub-parts of an object by the same methods provided in the "create objects" menu. Furthermore, an object can be the result of a boolean operation on two existing objects. The user is simply prompted for the names and classes of two existing objects and then presented with the following menu:

```
Boolean Operation
u : union
i : intersect
.d : difference
 =>
```

For this example, five more objects arc created. The first object is a wall that spatially coincides with the front face of the room created above and consists of the second and third objects as its sub-parts. The fourth object is a pipe that intersects the room, and the fifth object is that segment of this pipe that is within the room. The following presents the user interactions for creating these objects:

```
Name of the object [<ESC> : quit] => walll
Class of this object => walls
>» Input Data for Object Name <walll> of Class <walls> by :

g : geometry
s : subparts
b : boolean of existing objects
r : retry name
 => s
Name of the object [<ESC> : quit] => tilel
Class of this object => tiles
>>> Input Data for Object Name <tilel> of Class <tiles> by :

g : geometry
s : subparts
b : boolean of existing objects
```

```
r : retry name
 => g
Type:
b: block
r: rectangle
l: line
 => r
Width (> 0.) => 2
Depth (> 0.) => 2
Base Point (Bottom-Lower-Left Corner):
X => 0
Y => 0
Z => 0
Rotate? (y or n) => y
About X Axis (thru base point) => 90
About Y Axis (thru base point) => 0
About Z Axis (thru base point) => 0
Translate? (y or n) => n
>>>Defined Object <tile1> Geometrically.
>>> Sub-part of Object <wall1> :
Name of the object [<ESC> : quit] => tile2
Class of this object => tiles
>>> Input Data for Object Name <tile2> of Class <tiles> by :

g : geometry
s : subparts
b : boolean of existing objects
r : retry name
 => g
Type:
b: block
r: rectangle
l: line
 => r
Width (> 0.) => 3
Depth (> 0.) => 2
Base Point (Bottom-Lower-Left Corner):
X => 7
Y => 0
Z => 0
Rotate? (y or n) => y
About X Axis (thru base point) => 90
About Y Axis (thru base point) => 0
About Z Axis (thru base point) => 0
Translate? (y or n) => n
>>>Defined Object <tile2> Geometrically.
>>> Sub-part of Object <wall1> :
Name of the object [<ESC> : quit] => ^[
>>> Defined All Sub-parts of Object Name = <wall1> of Class = <walls>

c: create objects
d: delete objects
l: list objects
g: geometry check
r: retrieve relations
f: file handling
<ESC> : quit
 => c
Name of the object [<ESC> : quit] => pipe1
```

```
Class of this object => pipes
>» Input Data for Object Name <pipel> of Class <pipes> by :

g : geometry
s : subparts
b : boolean of existing objects
r : retry name
 => 9
Type:
b: block
r: rectangle
1: line
 => 1
End-1 X => 1
End-1 Y => -3
End-1 Z => 1
End-2 X => 1
End-2 Y => 10
End-2 Z => 1
»>Defined .Object <pipel> Geometrically.

c: create objects
d: delete objects
1: list, objects
g: geometry check
r: retrieve relations
f: file handling
<ESC> : quit
 => c
Name of the object [<ESC> : quit] => pipel_of_rooml
Class of this object => pipes
>>> Input Data for Object Name <pipel_of_rooml> of Class <pipes> by :

g : geometry
s : subparts
b : boolean of existing objects
r : retry name
 => b
Name of the First Object => pipel
Class of the First Object => pipes
Name of the Second Object => rooml
Class of the Second Object => rooms
Boolean Operation
u : union
i : intersect
d : difference

>>>Defined Object <pipel_of_rooml> by Boolean Operation.
```

The "retry name" selection simply gives the user the option of getting back to the name input pmmpt for either exiting the object creation process or reentering the name of an existing object.

Delete objects. This selection provides a facility for removing the NOODLES point-set[1] corresponding to an object from the model. When the user selects this option, he/she is prompted to specify the name and class of the object to be deleted. If an object with that specification exists, FIDX prompts the user to confirm

[1] **A point-set is a collection of fundamental elements, i.e., vertices, edges, faces, and regions.**

his/her decision before deleting that object **Once a** confirmation is given, **the** specified object is deleted from **the model.**

Note that presently this option removes *only* those fundamental elements of a point-set that are solely used by **the specified object** and *not* those elements of this point-set that are also used in other point-sets as the result of a merge operation.

List objects. This selection allows the user to check the names and classes of objects in a model. The user can either get a list of names corresponding to a particular class or get the names and classes of all objects. The list of objects of class "tiles" in the above example are obtained by:

```
Class of Desired Objects (<ESC> : all classes] => tiles
Object name <tile2> of class <tiles>
Object name <tilel> of class <tiles>
```

Check geometry. In order to check the geometry of a NOODLES point-set that corresponds to a particular object, this option can be used. For example, the geometry of the point-set corresponding to the "pipel" object is revealed as:

```
Name of the Object [<ESC> : quit] => pipel
Node at X=l.000000 Y=-3.000000 Z=l.000000
Node at X=l.000000 Y=10.000000 Z=l.000000
Edge of Length = 3.000000
Node at X=l.000000 Y=0.000000 Z=l.000000
Node at X=l.000000 Y=7.000000 Z=l.000000 '
Edge of Length = 3.000000
Edge of Length = 7.000000
Total of 4 Nodes; 3 Edges; 0 Faces; 0 Regions
```

Note that the "pipe 1" object consists of three colinear edges that make up the original line segment speci fled by the user. This is due to the fact that the original edge representing the "pipel" object was merged with the "rooml" object, and as a result, two extra nodes were created at the intersection points of these two objects.

Retrieve relations. With this option, users can retrieve a variety of topological relations between any objects, or combinations of any objects, in a model. Presently, these relations are limited to "subsct-oP and "intersects-with," however, other operations, such as "adjacent-to" or "composed-of," can be provided in the future, more complete versions of FIDX.

By selecting this option, the user is first prompted to input the name(s) of one (or two) object(s) and then is asked to optionally specify a particular class of objects to prune the resulting objects of the query operation. If only one object is specified, FIDX displays two lists of objects: those objects which are spatially a subset-of the specified object; and objects that intersect with the specified object However, if two objects are input, the user is further prompted to select the type boolean operation to be performed on those two objects. FIDX subsequently returns all the subset and intersecting objects in the point-set resulting from the boolean operation. Note that if a taiget class is given, FIDX returns only those objects that belong to the specified target class.

The following user-interactions illustrate several retrieve operations for the above example.

```
Name of First Object => rooml
Name of Second Object [<ESC> : if not applicable] => *[
Class of Resulting Objects [<ESC> : for all objects] => A[
> » » Objects that are a subset of the result :
>>> 1. <pipel_of_rooml>
>>> 2. <tile2>
```

```
>» 3. <walll>
>» 4. <tilel>
»> 5. <rooml>
»»> Objects that intersect the result :
>>> 1. <pipel>

c: create objects
d: delete objects
1: list objects
g: geometry check
r: retrieve relations
f: file handling
<ESC> : quit
 => r
Name of First Object => walll
Name of Second Object [<ESC> : if not applicable] => *[
Class of Resulting Objects [<ESC> : for all objects] => "[
>»» Objects that are a subset of the result :
>>> 1. <tile2>
>» 2. <walll>
»> 3. <tilel>
>>>>> Objects that intersect the result :
>» 1. <pipel_of__rooml>
>>> 2. <pipel>
>>> 3. <rooml>

c: create objects
d: delete objects
1: list objects
g: geometry check
r: retrieve relations
f: file handling
<ESC> : quit
 => r
Name of First Object => rooml
Name of Second Object [<ESC> : if not applicable] => pipel
Class of Resulting Objects [<ESC> : for all objects] => pipes
Operation
u : union
i : intersect
d : difference
 => i
>>>>> Objects that are a subset of the result :
>>> 1. <pipel_of__rooml>
> » » Objects that intersect the result :
>» 1. <pipel>

c: create objects
d: delete objects
1: list objects
g: geometry check "
r: retrieve relations
f: file handling
<ESC> : quit
 => r
Name of First Object => pipel
Name of Second Object [<ESC> : if not applicable] => tile2
Class of Resulting Objects [<ESC> : for all objects] => A[
Operation
```

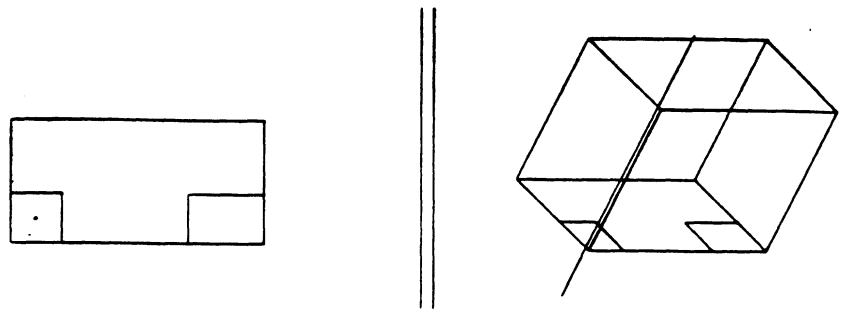

Figure 9: Side and Isometric Views of the Example Model

```
u : union
i : intersect
d : difference
 => i
>>> No object was matched.
```

File handling. This selection basically provides the operations for manipulating the disk files created or used by FIDX. Users can store the information about a model onto a disk file and later read that information back into FIDX. Models generated in other programs can also be read into FIDX; however, presently FIDX stores and restores only the "name" and "class" attributes along with the NOODLES point-sets. Users can also create a file containing the wire-frame information of objects and then process this file to create a POSTSCRIPT file for viewing or printing purposes. The following user-interactions illustrate the procedure for generating a display file and saving the model onto the disk.

```
l: load model from file
s: save model in file
g: generate display file for objects
 => g
Object name [<ESC> : quit] => room1
Object name [<ESC> : quit] => wall1
Object name [<ESC> : quit] => pipe1
Object name [<ESC> : quit] => ^[
Name of the Display File (with extension) => test.out
Stats: 32 edges in this pset

c: create objects
d: delete objects
l: list objects
g: geometry check
r: retrieve relations
f: file handling
<ESC> : quit
 => f

l: load model from file
s: save model in file
g: generate display file for objects
 => s
Name of the Model File (with extension) => test.ndl
```

Figure 9 presents the front and isometric views of the model generated in the above example. This picture corresponds to a POSTSCRIPT file generated from the display file of the above example.

As mentioned earlier, the model generated with the first prototype program can be loaded into FIDX and subsequently be manipulated. The following presents some query operations on the example model of the building created in the previous section.

```
c: create objects
d: delete objects
1: list objects
g: geometry check
r: retrieve relations
f: file handling
<ESC> : quit
 => f

1: load model from file
s: save model in file
g: generate display file for objects
 => 1
Name of the Model File (with extension) => building.ndl

c: create objects
d: delete objects
1: list objects
g: geometry check
r: retrieve relations
f: file handling
<ESC> : quit
 => 1
Class of Desired Objects [<ESC> : all classes) => A[
Object name <frame> of class <struc_system>
Object name <columns> of class <column_system>
Object name <beams> of class <beam_system>
Object name <heater> of class <hvac_mech_room>
Object name <plumbing> of class <hvac_pipes>
Object name <hvac> of class <hvac_zones>
Object name <floors> of class <arch_system>
Object name <elevator_shaft> of class <core>
Object name <student_center> of class <building>

c: create objects
*d: delete objects
1: list objects
g: geometry check
r: retrieve relations
f: file handling
<ESC> : quit
 => r
Name of First Object => floors
Name of Second Object [<ESC> : if not applicable] => plumbing
Class of Resulting Objects (<ESC> : for all objects] => *[
Operation
u : union
i : intersect
d : difference
 -> i
>>> No object was matched.

c: create objects
d: delete objects
1: list objects
g: geometry check
r: retrieve relations
f: file handling
<ESC> : quit
```

```
=> r
Nan* of Firsfc Oitffcct -> frame
Name of Second Object [<ESC> : if not applicable] => ^[
Class of Resulting Objects [<ESC> : for all objects] => ^[
> » » Objects that are a subset of the result :
»> 1. <frame>
>>> 2, <columns>
>>> 3. <beams>
> » » Objects that intersect the result :
>>> 1. <heater>
>>> 2. <hvac>
>>> 3. <floors>
>>> 4. <elevator_shaft>
>>> 5. <student_center>

c: create objects
d: delete objects
1: list objects
g: geometry check
r: retrieve relations
f: file handling
<ESC> : quit
 => A[

 ** T H A N K   Y O U   F O R   U S I N G   F I D X  **
```

5 Conclusions and Future Extensions

This report provides a brief overview of the initial phase of the proposed research for the design and development of an information exchange protocol for constructed facilities. This part of the research has primarily focused on the issues with regard to the definition and the retrieval of spatial information, i.e., the geometric attributes and the topological relations of various abstractions of facility components. A general graph-based model has been proposed for dealing with the organization and the attributes of facility components. Based on this model, a more specific model is suggested for the spatial decomposition of constructed facilities. Based on several objectives identified in this work, such as compositional and topological polymorphism with respect to the dimensionality of the geometric representations of facility components, two prototype programs have been developed by using the C interface of the NOODLES non-manifold geometric modeling system [3]. Descriptions of these two programs along with some sample runs are presented.

This part of this research has provided, more focused insights into the problem of exchanging spatial information used at various levels of abstractions by different applications of a constructed facility. However, the ideas presented here and their implementations are by no means complete. During the remaining pan of this work several issues with regard to the spatial as well as non-spatial attributes of facilities will be closely investigated. These issues include (but are not limited to): implementation of the augmented set operators using the proposed tolerance envelope mechanism; development of additional techniques for defining and retrieving the spatial information of facility components; representation of the non-spatial attributes in an information management system linked with the non-manifold geometric modeler, and, formalization of the functional interface to the proposed hybrid system for defining and retrieving the facility information into an information exchange protocol.

Acknowledgements **This research has been supported by the Engineering Design Research Center, an NSF Engineering Research Center at Carnegie Mellon University. Dr. Levent Gursoz and Atul**

Sudhalkar generously provided excellent support for NOODLES and spent many hours answering questions and discussing various issues related to this research. Adriana Kurfess also provided the most up-to-date documentation, fresh off the printer.

References

[1] M. S. K. Zamanian. An information exchange language for constructed facilites. Ph.D. research proposal submitted to the Department of Civil Engineering at Camegie Mellon University, January 1990.

***[2]* E. L. Gursoz, Y. Choi, and F. B. Prinz. Vertex-based represenation of non-manifold boundaries. In M. J. Wozny, J. Turner, and K. Preiss, editors, *Second Workshop on Geometric Modelling,* New York, September 1988. IFIP TC 5/WG 5.2.**

[3] A. Kurfess and E. L. Gursoz. *A User's Guide To* NOODLES^ *Geometric Modeling System (version 6).* Engineering Design Research Center, Carnegie Mellon University, Pittsbuigh, PA, 1990.

[4] K. Parsaye, M. Chignell, S. Khoshafian, and H. Wong. *Intelligent Databases.* John Wiley and Sons, Inc., 1989.