Communication in Cooperative Building Design

Raghu R. Bhat
Jupp Gauchel
Skip Van Wyk

Center for Building Performance and Diagnostics
Carnegie Mellon University
Pittsburgh, PA 15213 USA

This paper addresses communication issues, which are crucial in implementations of distributed design environments. Communication needs are specified and implemented in a prototype based on a modular knowledge-based approach (Gauchel, 1992a, 1992b) for simulation of a distributed multi-user system. The results of these simulations are reported, which show communication to be scalable as the numbers of applications and the size of the design increases. Finally, the implications of the results on real distributed systems are discussed.

Keywords: building design, distributed design environments, cooperative design, communication.

1 Introduction

New approaches to computer-aided design are based on advances in artificial intelligence and object-oriented programming that allow the declaration of domain and design knowledge to support users in creating and modifying designs. Approaches such as IBDE, ICADS and ARMILLA (Fenves, 1990; Pohl, 1992; Haller, 1985) investigate the possibility of integrated design environments, with tools that act as design assistants and demonstrate important principles of future integrated CAD systems.

2 Building Design

Software environments for building design must support three basic aspects of building design projects:

- *Building design is exploratory*. At the beginning of a design project, only a few precise descriptions of important objectives, goals, and constraints are usually specified. As the project progresses, these may remain ill-defined, or will be changed or new descriptions added, making the design process non-sequential and non-monotonic (Smithers, 1992).
- *Building design is multi-disciplinary, distributed, concurrent and asynchronous.* It involves the synthesis of design knowledge from many different sources and insti-

tutions. The participants in a project vary in education, expertise, role, and geographic location, and their activities vary with the nature of the design problem, building type, design stage, and responsibilities for parts and aspects of the building.

- *Building design is a large-scale information management task.* To describe all aspects of a building design, large amounts of diverse data are necessary. Besides this, the participants in a design project are assisted by large amounts of data, which do not become part of the building description. To develop a building design, some common design descriptions are required, in order to integrate design decisions of all participants.

3 Limitations of Comprehensive, Central Building Models

One alternative, is to introduce comprehensive, central building models, which are shared by all participants. Experience of such models and their implementation (Hovestadt, 1989), composed of some thousands of instances of more than a hundred building objects – (descriptions of spaces and components of buildings), highlight two general problems of centralized models.

- *Complexity*: A knowledge-based building design can be viewed as a highly connected, hierarchical composition of interacting building objects. An obvious implementation of this is as a semantic network, with the objects being described as nodes and the edges as pre-defined connections for message passing. Since the number of connections tends towards the square of the number of nodes, the introduction of new nodes increases the complexity of the network. The larger the model becomes, creating, modifying, and deleting nodes or edges increase the possibilities of unanticipated behavior of the system and unpredictable design states.

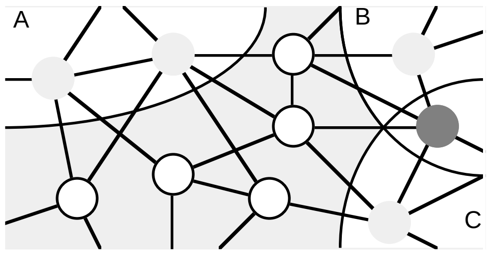

Figure 1. Interconnected and overlapping partial nets A, B and C.

- *Concurrency*: For a central model to function as a multi-user system, users would either work concurrently on the entire model, or separately on partial networks. Concurrent work on the entire network leads to unpredictable results due to the inherent connectivity. Working separately on partial networks leads to problems of interaction between connected nodes of different partial networks as well as problems with overlapping partial networks (Figure 1).

An alternative approach is to view building models as autonomous, interacting agents (Agha, 1986 and Kemper, 1990), that can be tailored to the needs of users. This approach reflects the fact that designers in a real world design environment do not communicate by comprehensive, centralized models, but by the overlapping of partial, domain and role-specific models, which are not completely predefined.

4 An Approach to Distributed, Knowledge-Based Building Models

We proposed earlier (Gauchel, 1992a and 1992b) an approach, which has the following features: Highly-connected, knowledge-based building models can be composed of elementary descriptions, to fit the specific needs of the users. These models are able to communicate with one another in a distributed environment. The architecture of the elementary descriptions, their composition principle, and the communication among distributed models are based on unified principles.

The approach is based on (1) *building objects,* which are able to control their internal states and are the primitives of building models, (2) *building modules,* which describe the interaction of building with their environment, (3) *building models*, which are compositions of building modules and function as design applications, and (4) *communication* among building models.

4.1 Building Objects

Descriptions of building objects are based on object-oriented programming paradigms and are made of structural and behavioral attributes. While structural attributes describe the real-world objects, behavioral attributes, implemented in the form of functions, are design support tools. Structural and behavioral attributes may be linked to create internal actions in response to changes that are initiated outside of the object (inputs). The inputs are always directed at one structural attribute. If linked to a behavioral attribute by a "daemon," its change of state may trigger this attribute, which will change the state of another structural attribute, and so on, until no further behavioral attribute is triggered (Figure 2). Thus, building objects react to inputs and seek stable, internal conditions. Building objects are defined as concepts and instanced as necessary.

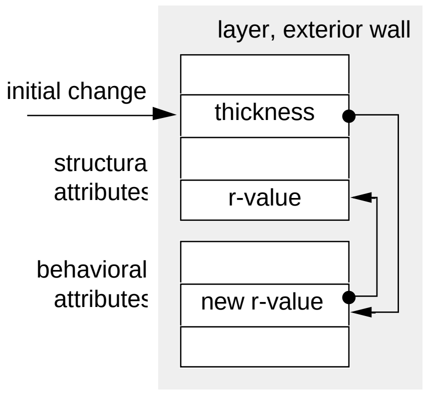

Figure 2. An example for a building object.

4.2 Building Modules

Building objects, like objects A and B in Figure 3, are sources of interaction with other building objects, and building models are compositions of these interaction patterns. If described separately as entities these patterns can be viewed as building modules (Figure 4). Since their internal structure is not dependent on the nature of the included objects and their interactions, all building modules may have a unified organization.

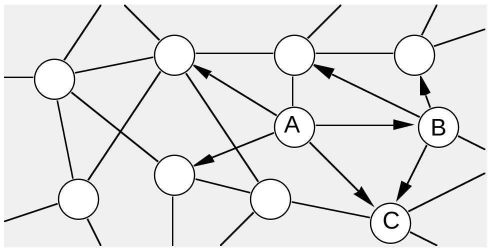

Figure 3. Building objects as sources of interactions with other building objects.

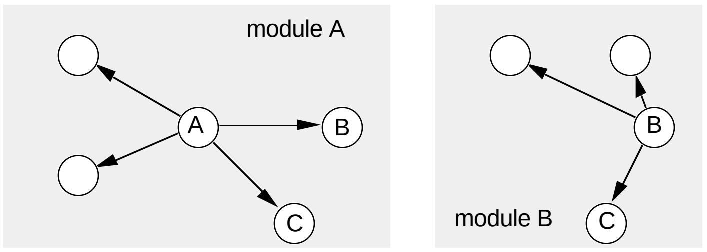

Figure 4. Building modules, redundant representations of shared objects.

While describing interaction patterns based on objects A and B as entities, it is necessary to introduce redundant representations of shared objects, and this leads to the principle of all communication among building models: they may communicate if they share objects. Since this principle is also independent of the nature of objects and their interactions, a unified communication mechanism among building models can be developed.

A building module includes a central object, one or more surrounding objects, and the interactions of the central with the surrounding objects (Figure 4). A building object is the central object of its own module and can be a surrounding object in other modules. It can receive changes initiated outside the module only as a central object. By its changes, it may communicate with the surrounding objects, allowing them to adapt themselves to its new condition with changes of their own. This communication is modeled by a set of production rules. A rule is automatically triggered by specific changes of structural attributes of the central object, and result in actions which create inputs to one or more surrounding objects, including creation, modification, and deletion of those objects. Thus, production rules are design support tools.

The descriptions of building objects and building modules tend to be canonical descriptions, from which user-defined abstractions can be modeled according to special user interests or the design support tools.

4.3 Building Models

Building objects are located in a multi-dimensional data space. The dimensions may be: the xyz-dimensions of the euclidean space, time, version, design stage, user profiles, etc. All building objects have structural attributes to position them in this space.

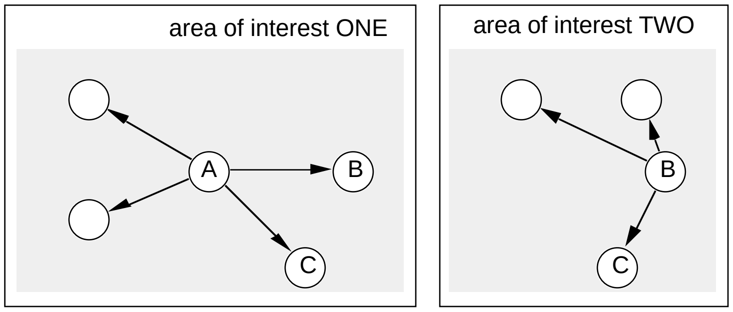

Figure 5. Modules A and B imported into different areas of interest.

The composition of a building model is based on the definitions of building modules and two further user declarations: a declaration of an area of interest, which is a partial space of the multi-dimensional data space, and a declaration of objects of interest. Instances of those modules—production rules and instances of the included objects —which are within the area of interest, are then imported and composed to form a building model.

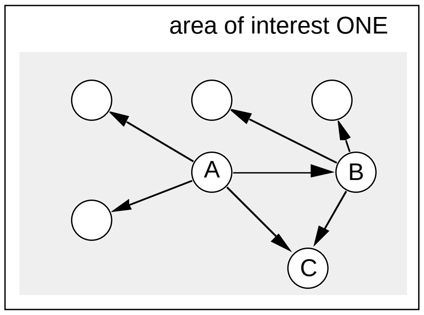

Figure 6. Modules A and B imported into one area of interest.

There are two alternatives when importing building modules: Importing them into different areas of interest would result in redundant representations of shared building objects (Figure 5), if the areas of interest overlap or even if they are identical. Importing them into the same area of interest would result in a union of instances (Figure 6), to compose building models. The complexity of the models and the amount of data can be controlled by appropriate declarations of the area and objects of interest. Working within a building model makes it impossible to control or manipulate instances of building objects, which are within other models. The more the building objects and the building modules incorporate design support tools, the more the building models have the character of design applications.

4.4 Communication among Building Models

Building models communicate by importing or exporting data, and the user is in control of this communication. In principle, the user has two options: 1) decide when to export modification to the design, and 2) what the update mode should be for changes done by other applications to objects in the user's area of interest, i.e., through setting a notify or an automatic update mode. Interactive control of data import or export is a pragmatic communication strategy, which maintains user autonomy.

5 Functional Communication Requirements

The following functional requirements in a distributed design environment were identified, which drove the prototype design and the communication issues investigated in this paper:

- Multiple users can work concurrently.
- The system offers interactive user-interfaces, combining CAD, command-line, and menu interfaces.
- The users can run any combination of applications concurrently.
- Applications can be invoked at any time.
- Users can define and dynamically change applications (area and objects of interest) and abstractions of data.
- Users can activate or inactivate design knowledge.
- An application can only access (import, edit, export) data, which is defined by its current area and objects of interest.
- All data access is under user control. Data can be accessed at any time without notifying other applications.
- An application is notified of any modification in the global data which is in its area of interest.
- A user can work autonomously, and this is accessible by other applications only after an export.
- Design conflicts are not resolved automatically but sources of possible conflicts are shown to the user for action.

6 Implementation

The translation of the high-level functional requirements of a multi-user system to an appropriate implementation was done by formal software engineering analysis called the Design Space Method (Lane, 1990). The purpose of the method is to transform high level user concepts into desired functional and structural specifications of the system. The total design space of the system is described by functional (how one wants the system to behave) and structural (how to build the system). Possible alternatives for each of these dimensions are enumerated and consistency rules which specify desirable and undesirable combinations between the alternatives of the functional dimensions, between functional and structural dimensions and between the alternatives of the structural dimensions.

This method lets high-level software design requirements drive the functional design, which then drive the structural design of the system, with the consistency rules ensuring only valid systems are generated. Finally, a scoring system was used to choose among various valid alternatives for the system. The analysis with our emphasis on minimizing communication and maximum autonomy for the users resulted in the selection of a Client-Server system (Figure 7). A more detailed description of the methodology is published elsewhere (Asada, 1992).

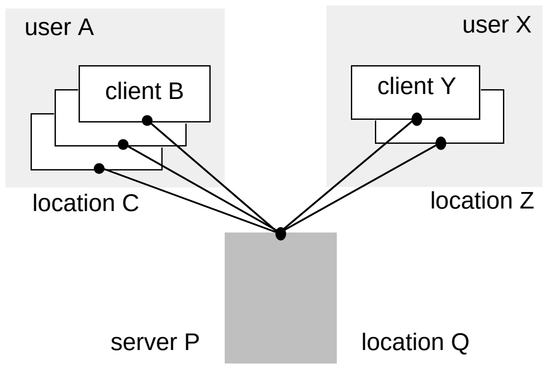

Figure 7. A Client-server System.

6.1 System Components

The Clients: A client consists of building objects, production rules, and a graphic user interface. Appropriate graphic and semantic abstractions are specified for building objects by the declaration of the area of interest. Users may activate and inactivate design tools (behavioral attributes, production rules), which may be of the following categories: manipulation tools, such as create, delete, move or shape building objects; simulation tools; consistency checking tools—for checking possible design conflicts; and automated tools. It is important to note that these tools may be triggered automatically or invoked by the user. Each client has a local data store, which maintain relationships between building objects according to the building model.

The Server: The server maintains all the permanent design data and can be thought of as a global data base, and provides these functionalities: storing data; versioning; adding, deleting or updating data; keeping track of active clients and their requirements through a filter register; notifying clients of design modifications; and storing definitions of building modules.

Client-server Communication: The data flow between the local (clients) and global database (servers) is based on importing and exporting data by the clients, according to the declarations of the area and objects of interest. Data communication events are either synchronous or asynchronous. Synchronous events are those that require a strict ordering or timely confirmation of success or failure, while asynchronous communication events are used when strict ordering requirements are relaxed or during periods of autonomous design activity. Areas and objects of interest and modes for update can be dynamically changed during a user session. Requests for import and export of data can occur at any time.

Asynchronous communication events are implemented as message passing and synchronous events as object method invocation. Though message passing and object method invocation are essentially identical in the KnowledgeCraft environment we use (CGI, 1986), we have elected to separate these conceptually into asynchronous and synchronous communication, for instrumentation and analysis purposes.

7 Communication Experiments

The purpose of the prototype and its instrumentation were to perform the following tasks:

- To implement the data communication architecture in the prototype system and instrument the system to gather data on the frequency and volume of communication in the system.
- To gather communication data from variety of operational scenarios.
- To analyze the data to determine the growth of communication as a function of the size of the design and number of active applications.

7.1 Instrumentation

The prototype was used to gather data on the frequency and volume of communications within the system. Frequency of communication is defined as the number of communication acts in a usage scenario and volume of communication is defined as number of bytes transferred during this communication. Data is gathered for each object method invocation or message sent through the system. Data on efficiency of the filtering mechanism is also recorded.

7.2 Usage Scenarios

Usage scenarios result in communication scenarios. Each communication scenario consists of a set of communication transactions. Each transaction consists of communication acts and these acts are either synchronous or asynchronous. The following usage scenarios were used to gather communication data:

- *Start-up and Initial Import Scenario*: In this scenario, the user starts an application, the application registers itself, its area of interest and abstraction with the server, a graphic user interface is created for the application, the application then requests an import of objects in its area from the server, the server filters the data using the registered information and sends it to the application. This scenario was run under the following conditions: 1) fixing the number of applications and varying the size of the design, 2) varying the number of applications with area of interest of each covering the entire design, while keeping the size of the design constant, and 3) varying the number of applications and their areas of interest, while keeping the size of the design constant.
- *Edit Scenario*: In this scenario, the user does local editing and then exports the data. The application sends the changed instances to the server and the server updates its database. The server then processes the changed objects through the filters of the other active applications and sends a message to each application interested in this data. This scenario was run under the following conditions: 1) varying the number of applications, making a fixed number of edits in one application, with all applications having the same area of interest, 2) varying the number of applications, making a fixed number of changes in an application, with all applications having varying areas of interest, and 3) fixed number of applications, varying number of edits in a single application, with all applications having a common area of interest. In all the cases, applications had different abstractions for their objects of interest.
- *Exit-scenario*: In this scenario, the user quits the application which de-registers itself by sending a message to the server. The server updates its registers and the application then exits. This scenario was performed by: 1) varying the number of

applications while keeping fixed the size of design, and 2) fixed number of applications, while varying the sizes of the design.

7.3 Data and Analysis

Data: The prototype was tested with seven clients which were run in various combinations. Each client was composed by a maximum of five and a minimum of one building module(s) (Figure 8).

modules / clients	site	building	roof	room	wall	surface	opening	edge
client 1		○	○	○	○			
client 2					○		○	
client 3				○				
client 4			○	○			○	○
client 5	○	○						
client 6				○	○		○	
client 7					○		○	

Figure 8. Clients composed of building modules.

The scenarios described above were run with building designs ranging in size from 21 to 10164 instances of building objects. “Designs” are some combination of objects and larger designs are composed from smaller ones by arrangement in a matrix, and were generated automatically by a program, and the above scenarios executed by scripts. Each of the scenarios was run by varying the area of interest and the number of clients, with designs of varying sizes.

Analysis: All data is stored in log files which are analyzed by tools external to the prototype. The data is analyzed by a analysis program which disassembles the communication data into a tabular spreadsheet format. Number and size of both synchronous and asynchronous communication acts are totaled for each user scenario. The client-server architecture, and the design of the system lets us predict certain results which were then corroborated by the empirical results.

8 Results

The declaration of the clients area of interest can have a significant impact on the volume of communication among clients. If the area of interest is relatively small compared to the total design, the server will filter out a majority of the data which might have been sent to the clients. The converse of this is that when the area of interest is large, the volume of communication increases correspondingly.

Another aspect which determines communication frequency and volume is the amount of overlap between different clients' areas of interest. Any changes made by one application to the global data may have to be sent as updates to the other interested clients after appropriate filtering. Thus, in Figure 8, Client 6 and Client 7 share opening, surface and wall building objects, so that commitments made by one client may have to be sent to the other, if the appropriate update mode is set by the other client. The abstractions specified by each client similarly affect the volume of communications. A client interested in an abstraction of an object, sees only a small attribute set of the object and will receive a smaller volume of communication from the server.

The experiments analyzed synchronous and asynchronous communication by varying the number of clients, varying number of building objects and varying areas of interest. Graphs were plotted between frequency of messages and number of clients, volume of messages and number of clients, volume of communications, and number of building objects.

8.1 Synchronous Communication Results

This is used by clients in registration of area of interest, registration of abstractions, registration of import modes and deregistration. It is also used in generation of a unique identifier for new instances of building objects. Application start-up and exit scenarios generate data for synchronous communication. Some of the results are intuitively obvious and all were confirmed by empirical results.

Growth in Number of Clients: Messages can be analyzed according to the frequency and size of each message.

- *Frequency*: Registration of area of interest and abstractions for a client will take place once each time a client is invoked. Registration of update mode can occur at least once, but can occur multiple times at user request. Each client sends a single de-registration message on shutdown. As expected, the experimental results show that for start-up as well as shutdown of the client the frequency of synchronous communication is a linear function of the number of active clients in the system.
- *Size:* Each synchronous communication has a nearly constant size for each application with the variation due to differing abstraction needs of each client. The abstraction list is a subset of the attributes list of each class of objects of interest and is relatively small, even for objects with large attribute lists. We found a linear trend when we analyzed the size of communication in bytes as the number of clients grew. On application exit, a constant size message is sent to the server for shutdown. Thus, given a linear frequency and a nearly linear size of each message, volume of synchronous communication grows approximately linear with size.

Growth in Number of Building Objects: None of the synchronous communications events refer to the number of objects in the design hence it is independent of the number of building objects in the design.

8.2 Asynchronous Communication Results

The prototype uses this type of communication in requesting an import of data, replying to a import request, exporting data, and for updating clients who have an interest in data which is changed. The size of each of these communication can be potentially very large and the frequency of such communication has to be controlled.

Growth with Number of Applications: The design of the system uses one message for each import request, reply to an import request, and for export message. There can be as many update messages after an export as there are clients interested in the changed objects.

- *Frequency:* Clients on start-up generate exactly two asynchronous messages: an import request for data and the corresponding message from the server honoring the request. The prototype sends one message for each import, reply to an import request or an export request. On an export, there is one message to the server and potentially one update message can be sent to all other clients who may be interested in the changed data. Thus each export request in an Edit scenario can potentially generate as many messages as there are clients. Thus the frequency of asynchronous messages will grow linearly with the number of clients in the system.
- *Size:* Import requests are of constant size, and thus the volume of import requests can at most grow linearly with the number of clients. Replies to import requests, exports, and the potential updates may vary in size and are examined separately. Contents of a reply message to an import request are all the building objects within that area of interest, appropriately abstracted. This has no relation to the number of clients currently running, but depends only on the area of interest and abstraction needs of the requesting client. The size of the asynchronous message on export of data depends entirely on the local changes made in client and is not dependent or affected by the number of clients running. The update message contains a list of changes committed to the global design which must be updated in the local workspace of interested clients. The size of each update message then depends, as before, on the area of interest and abstractions of the clients. In the worst case where all clients share the same area of interest and abstractions and update messages are sent to each of the clients, the total volume of communication will be a function of the total number of clients. Thus the total volume of communication can, at the most, vary linearly with the number of clients.

Growth with Number of Building Objects:

- *Frequency*: The import request is a single message which results in a single reply containing all the objects which are requested. Similarly, frequency of export depends only on the activities of the user in the local workspace and update messages are potentially sent to all the other clients as we have seen before. Thus, frequency of asynchronous messages in both cases do not depend on the number of objects in the system.
- *Size*: The import request has a fixed size and will not vary with the number of objects in the system. The size of the reply to the import request will depend on the number of building objects in the system. In the worst case, where the area of interest is the entire design, the size of the reply will be a linear function of the number of objects in the system. In a particular case, the size of the reply will depend linearly on the number of objects within the area of interest of the client. The volume of the export and consequent update messages are dependent on the work performed in the local workspace and in the worst case we can assume that every object is edited and exported and then sent as updates to all other clients. Thus, in the worst case the total size of the communication will be a function of the product of the number of clients and the number of objects. In general, we can expect the number of clients to be very small compared to the number of objects in the design.

8.3 Filters

The prototype uses filtering functions to reduce the communication load on the system. Basically, three filters were used: client, object and attribute filters. The client filter determines which clients should receive update messages, object filter screens which objects fall within an area of interest of the client, and the attribute filter determines which abstractions of the object are relevant to a client.

Empirical results show that filters can be used at a variety of levels of granularity to reduce data communication within a system, without affecting the functionality of the system. However, it should be noted that while filters can drastically reduce the communication volume and frequency, they can increase the computational load on the server; thus, the server must be capable of processing large amounts of data in a timely manner.

9 Conclusions

We implemented the prototype as a Client-Server system and simulated the communication in a multi-user system using clients which are composed of modular, knowledge-based modules. The prototype implementation showed that communication frequency and volume are scalable as the number of clients and the size of the design grows. This was achieved by a judicious use of filters and by restricting communication to the client level. In moving to a real multi-user system, further issues such as inter user communication, communication between sessions, as well as problems of networks and distributed operation have to be considered. We feel the scalability of communication demonstrated here is an encouraging step in the design of such large-scale systems.

Acknowledgments

We acknowledge the contributions of the armilla/A4 research project: Professor Fritz Haller and researchers of the Institut für Bauproduktion, Universität Karlsruhe, Germany and Professor Volker Hartkopf and researchers of the Center for Building Performance and Diagnostics, Carnegie Mellon University, Pittsburgh, PA, USA.

The communication issues discussed here are based on a prototype, developed by the team of Toru Asada, Nadine Bounds, Paul Duerig and Roy Swonger under the direction of Robert Firth and Len Bass of the Software Engineering Institute at Carnegie Mellon University. Financial support was provided by the RETEX project (BMFT, Germany and the Company of ROM, Germany) and The Advanced Building Systems Integration Consortium (ABSIC), Carnegie Mellon University.

References

Agha, G.,1986: *ACTORS: A Model of Concurrent Computation in Distributed Systems*. Cambridge: The MIT Press.

Asada, T., Swonger R. F., Bounds, N., and Duerig, P., 1992. *The Quantified Design Space: A tool for the Quantitative Analysis of Design*. Technical Report CMU-CS-92-213, Carnegie Mellon University, Pittsburgh, PA.

CGI, 1986. *KnowledgeCraft*, Pittsburgh: Carnegie Group Inc.

Fenves, S. Flemming U., Hendrickson, C., Maher, M. L., and Schmitt, G., 1990. “Integrated Software Environment for Building Design and Construction,” *Computer Aided Design*, Vol. 22, No. 1.

Gauchel, J., Van Wyk, S., Bhat, R., and Hovestadt, L., 1992a. “Building Modeling Based on Concepts of Autonomy,” in J. Gero (ed.), *Artificial Intelligence in Design ´92,* Kluwer Academic.

Gauchel, J., Hovestadt, L., Van Wyk, S., Bhat, R., 1992b: “Modular Building Models,” in *Proceedings*, Design Decision Support Systems in Architecture & Urban Planning, Eindhoven, Netherlands.

Haller, F., 1985. *ARMILLA - ein installationsmodell*. Universität Karlsruhe, Germany.

Hovestadt, L., 1989. *Ein 'intelligentes CAD-System' für die Planung und Verwaltung von Leitungsnetzen in Hochinstallierten Gebäuden*. VDI-Berichte 775, Germany.

Kemper, A., Lockemann, P.C., Moerkotte, G., Walter, D.D., and Lang, S.M., 1990. “Autonomy over Ubiquity: Coping with the Complexity of a Distributed World,” in *Proceedings, International Conference on Entity-Relationship Approach, La*usanne, Switzerland.

Lane, T G.,1990: *Studying Software Architecture through Design Spaces and Rules*. Technical Report CMU-CS-90-175, Carnegie Mellon University, Pittsburgh, PA.

Pohl, J., et al., 1992. “A Computer-Based Design Environment—Implemented and Planned Extensions of The ICADS Model,” California Polytechnic State University, San Luis Obispo.

Smithers, T., 1992. “Design as Exploration: Puzzle Making and Puzzle Solving,” position statement at *AID ´92*, Pittsburgh, PA, June 1992.

Swonger, R., Asada, T., Bounds, N., and Duerig P., 1992. *Prototype Analysis Report for the ARMILLA Prototype Evaluation and Development Project*. Unpublished Report, MSE, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.