The concept of the virtual computer is one of the most significant ideas to emerge in the field of computing. Conventional models of architectural design, including state models and process models, have been based on the past on the von Neumann model of computer systems. Von Neumann systems are characterized by stored programs and data, and sequential processing on a single processor. The concept of the virtual computer enables us to break away from the von Neumann model in the representation of architectural design. Virtual computers can now be used to represent architectural design using concepts of parallel or networked systems. One of the limitations of modeling architectural design processors on the computer has been the representation of the processor as a virtual processor. Virtual computers can eliminate that bottleneck. This paper introduces the concept of representing architectural design using virtual computers. The application of the concept is an auditory design system developed by the author to briefly introduce..
Introduction

Architects are involved in the task of designing the built environment from the scale of a single room to that of a city. When architects design, they make decisions about the form and spatial arrangement of building materials and products that define physical structures and spatial environments. The physical structures and spatial environments enable a complex synthesis of visual, aural and kinesthetic experiences. The goal of many architects is to create intellectually stimulating, lively and safe environments that facilitate a wide range of positive human experiences. Seen from this viewpoint, the process of architectural design is a decision making process. The decisions are made using various methods that may be intuitive or rational.

The products of architectural design may be identified by asking the question: what do architects produce when they design? The simple answer to this question is that architects produce representations—representations of physical structures that are to be built and spatial environments that are to be created. These representations traditionally include drawings, physical scale models and written specifications. They are a mix of graphical, physical and verbal representations. The development of computer technology in the last three decades has enabled computer-based drawings and models to be included in the architect's range of representations. All these representations define a virtual world in which analogues of physical structures and spatial environments can be cognitively manipulated. Architects often dwell in this virtual world of representations.

To sum up, when architects design, they make decisions about the form and spatial arrangement of building materials and products that define physical structures and spatial environments. Architects also create various representations to depict the physical structures and spatial environments. Based on this notion, the computational modeling of architectural design presents a twofold challenge. It involves the modeling of both design decision making and the making of representations. The concept of the virtual computer can help designers meet this challenge in new ways.

Representing Architectural Design

The task of representing architectural design can be artificially split into the tasks of representing architectural design products and architectural design processes. This split is complicated by the fact that the products are both the media and the result of the processes. Visual thinking and physical thought make the separation of product and process very difficult. Visual thinking is the cognitive manipulation of graphic elements in creating representations, and physical thought is the cognitive manipulation of physical elements in creating representations. In visual thinking and physical thought, the distinction between product and process is blurred. If architectural entities are modeled as virtual computers and their synthesis is implemented using a visual programming language (discussed later in this paper), the blurring of distinction between product and process can be greatly reduced, if not eliminated. This provides an opportunity for the reversal of the graphic bias of architectural design.

Representing Architectural Design Products

Architectural design products encompass physical and conceptual entities. Physical architectural entities are individual building components (materials and products) and assemblies of building components that behave as individual components. For example, a brick is an individual component. A wall or arch made of bricks is an aggregate component whose behavior can be abstracted and modeled. Conceptual architectural entities are intangible entities like circulatory systems, ordering
Representing Architectural Design Using Virtual Computers

systems and structural systems. Commercial computer-based systems only provide a medium to represent physical architectural entities. These physical entities are represented as complex topological constructs synthesized from primitive solid geometric entities or planar surfaces. For example, entities are represented using Constructive Solid Geometry (CSG) or Boundary Representation (B-Rep). The synthesis is achieved through layering, grouping, Boolean operations (union, intersection, difference) and transformational operations (extrusions, sweeps, etc.). Conceptual entities in the mind of the designer regulate the synthesis of the physical entities. There is no explicit representation of conceptual entities. Conceptual entities can only be inferred from the organization of the physical entities. Conceptual entities are not engaged or manipulated directly in commercial systems. This significant drawback can be overcome in computer-based design systems where conceptual entities are explicitly represented along with physical entities. The component-based paradigm of Hartmann and Chen (1996) addresses the modeling of physical entities but lacks the modeling of conceptual entities.

Modelling Architectural Design Processes

Many models of computer-aided architectural design processes have emerged. In a recent article, Mitchell discusses three paradigms of computer-aided design (Mitchell 1995). The paradigms include design as a problem-solving activity, design as a knowledge-based activity and design as a social activity. In that article, Mitchell advocates the modeling of design as a social activity as “the only way to go.” He calls for a “society of design” making reference to Minsky’s classic work The Society of Mind (Minsky 1988). The three paradigms of Mitchell reflect the transition from a serial modeling of design processes to a parallel or networked modeling of design processes.

From a serial to a parallel or networked modeling of architectural design processes

Models that have been proposed so far for computer-aided architectural design processes have shown this transition from serial models to parallel or networked models. These models have included problem solving (Newell & Simon 1976), constraint-based decision making (Gross 1987), puzzle making (Arabesque 1997), bi-directional search (Carrara, Kally and Novembrini 1994) and the distributive cooperative model (Poll and Myers 1994) to name a few. The key diagrams explaining these processes reflect the serial or parallel characteristics of the processes.

With the concept of virtual computers, other models can now be explored. Some parallel models as described by Williams (Williams 1996) that can be used are: (1) Array Processing: In array processing, a number of identical processors or virtual computers act under the control of a common unit. All the processors in a processor array execute the same process at the same time. Each processor has its own data. Some of the processors can be masked from executing a process allowing for flexibility. (2) Pipeline Processing: In pipeline processing, a process that would have been performed as a series of steps on one processor, is passed through a ‘pipeline’ of processors, each executing one step of the process. This model will work efficiently only if each step of the process utilizes the results of the previous step of the process and its results are subsequently needed by the next process. Each processor can execute a different process unlike in array processing. Information flows along one direction in a pipelined process involving multiple processors. A multidimensional pipeline is sometimes referred to as a systolic array. (3) Shared Memory Processing: In shared memory processing, a number of processors have access to a common, shared memory. Processes using shared memory are controlled using “fork” and “join” operations. Synchronization is the key factor in processes using
shared memory, because a process should not corrupt the results of another process. Techniques to prevent this include “semaphores” and “monitors,” and (Message Passing: In message passing, data is explicitly passed from processor to processor. All other data is private. Messages are passed along “channels” linking processes. Message passing can be asynchronous or synchronous. These models can be implemented using the concept of virtual computers. The case study presented at the end of the paper shows briefly how this is possible.

**virtual computers**

In object-oriented computing, entities are modeled as encapsulations of data, and operations that can be performed on that data. Encapsulation is a computer abstraction. A collection of data and operations normally performed on the data are closely related, so they are treated as a single entity (rather than separate) for purposes of abstraction. Each encapsulation can be thought of as a virtual computer that is mapped onto a physical computer (see Figure 1) with its own private memory (its data) and instruction set (its operations). The reference to objects as virtual computers was made by Alan Kay (Kay 1977). He envisaged a host computer being broken down into thousands of virtual computers, each having the capabilities of the whole, and exhibiting a certain behavior when sent a message which is a part of its instruction set. He called these virtual computers “activities.” According to him, object-oriented systems should be nothing but dynamically communicating “activities.” As such they form an interesting model with which to simulate architectural design. Mitchell’s recent call (Mitchell 1996) for a “society of design” with a “collection of agents of different kinds interacting over a network” echoes the ideas of Alan Kay. In another interesting perspective, encapsulations have been likened to integrated circuits rather than virtual computers by Levlette & Cox (1985).

Before the development of object-oriented computing, the main computer abstractions being used were data structures and procedures in keeping with the von Neumann model. The concept of the virtual computer helps us to move away from the von Neumann model. This is because many virtual computers can be mapped onto a single physical computer or many different physical computers enabling the modeling of parallel or networked systems.

![Diagram of computer object relationship](image.png)

**Figure 1. The mapping of encapsulations (virtual computers) onto a physical computer.**

**modeling architectural entities as virtual computers**

Architectural entities can be classified as physical or conceptual. For example, a column can be classified as a physical architectural entity. A column can be modeled as a virtual computer (see Figure 2). The data of the column comprises its topology, its dimensions, its loading conditions, its end conditions, its dimensional constraints and its material specification. The operations of the column include a method to size itself based on its loading conditions, end conditions and constraints. The operations also include methods for the column to formulate itself in different structural systems. Columns modeled as virtual computers can interact with structural systems (also modeled as virtual com-
computers) to be sized according to loads on the structural system. Columns can also interact with beams to define structural systems like a simple frame structure. Columns can also maintain an internal mechanism that administers constraints when the column is executing methods to size itself.

The grid can be classified as a conceptual architectural entity. A grid can be modeled as a virtual computer. Figure 3. The data of a Cartesian grid are the grid values along the three coordinate axes. The operations of the grid include formatting other entities with the grid values along the three coordinate axes. Grids in two dimensions and grids with alternating grid values, as in a tartan grid, can also be modeled in this way. Different grid patterns with different rhythms, i.e., different sets of alternating values for the grid cells, can also be modeled. Grids are essentially place holders for other architectural entities. Grids can also interact with other grids to form complex field objects. The interaction of grids can actually produce or instantiate field objects. Grids can also produce representations of themselves using graphic elements.

A circulatory system can be modeled as a virtual computer using graph theory. Figure 4. The data of the graph include its nodes and its edges. The node can represent a space and the edges can represent links between spaces. Methods that operate on the graph’s data include finding the centrality of a node (the Konig number), finding the shape index of the graph, finding the beta index of the graph and optimizing the graph for minimum circulation distances. Duals of graphs (Broadbent 1973) or Teague networks (Mitchell 1977) can be used to derive spatial enclosure patterns that reflect circulation patterns represented by the graphs. Figure 5. Ordering systems can also be represented as virtual computers using graph theory. The data in an ordering system consist basically of connectivity information. The data represent adjacencies of spaces.

Physical structures can be modeled using the methods of void modeling (Yessien 1987). Void modeling uses the same concepts as solid modeling but uses voids as elements rather than solids.

A computable model for architectural design using virtual computers

The concept of virtual computers provides a computational basis for the creation of new types
of computer-based design systems in architecture. One system is based on modeling architectural design as synthesizing interaction (Mahalingam 1999a). The synthesizing interaction model has fundamentally different implications for the design of computer-based design systems in architecture. Architectural entities become agents that can interact, and are not passive elements that are arranged or composed. Architectural designs are generated by the synthesizing interaction of physical and conceptual entities that are modeled as virtual computers. (see Figure 6) It is more common for architectural designs to result from a dynamic synthesizing interaction of physical and conceptual entities than it is from an explicit problem solving process.

The interaction model is based on the idea that architectural entities can be represented as virtual computers that interact to create designs by performing computations. It is this interaction of virtual computers that provides new computable models for design systems in architecture. This interaction model of architectural design can be implemented in design systems using a three stage process. The implementation involves: (1) the modeling of physical and conceptual architectural entities as virtual computers, (2) structuring their interaction and (3) managing the synthesized design using techniques like the one developed by Nijssen and Halpin. (Nijssen and Halpin 1989)

This model can be compared to the one advocated by Mitchell (Mitchell 1999) which he characterizes thus: "There are multiple agents with their own (not necessarily consistent, comparable or compatible) knowledge-bases <data> and problem solving capabilities <operations>. They proceed by exchanging proposals, arguments, and counter-proposals and counter-arguments, and seek to form a consensus. They import knowledge into a common pool, they construct some common intellectual ground and sometimes change each other's minds." Mitchell goes on to specify the requirements for this model thus: "At the most basic level, the agents concerned must have their own, local computational resources and must be linked in an efficient network. Next, it seems to me, they must have some form of concurrent, joint access to a digital version of the model - the proposal that is "on the table" - so that they can point and refer to it, analyze it, modify it, and so on."

It is when Mitchell defines what the different agents are that his model differs significantly. Mitchell's set of agents include (information) interpretation and translation agents, reporting agents, research agents, problem-solving agents and contract negotiation agents. These agents assist the designer in performing design tasks by...
managing information and solving problems. They set outside the actual creation of the design artifact even though they assist and inform the process. In the author's model, the design artifact is the resolution of the synthesizing interaction of conceptual and physical entities. It is an open "society of entities" that can interact and re-solve in many ways generating different state models of designs. The resolution of the design can be a simple act of stopping when a desired state is reached, or be based on performance analyses to see if the design meets various preset goals. The state models can reside in a single "database" or exist in a distributed state as part of the different entities that are re-solved continuously based on design operations.

Implementing the interaction model of architectural design

The interaction model of architectural design can be implemented in many ways. One of them is through the use of a visual language. According to Shu (Shu 1988), the use of visual languages is a paradigm for expressing system computations that offers the possibility of directly manipulating computational objects or virtual computers.

A visual program or a visual sentence is written in a visual language by a spatial arrange-ment of icons that represent virtual computers. The spatial arrangements can be literal or metaphoric representations of the systems to be designed. Traditional programs are written as a sequence of instructions. The constructive operation in putting these programs together is concatenation. In a visual language, because a two-dimensional space is used, the constructive operations involve horizontal linking, vertical linking and spatial overlaps. (see Figure 7). The visual interaction is used to develop the syntax of the program. Because the syntax used is visual, problems that can be solved by visual thinking or visual operations can be easily modeled in a visual language. When architects work in section or in plan, they are, in effect, solving problems in a two-dimensional visual language.

One can expand this concept and imagine a three-dimensional visual language with additional spatial construction operations. (see Figure 8): Architects, in essence, work with such visual languages by spatially arranging building materials and products. Using visual programming or a visual language as the basis for a computer-based design system in architecture utilizes the mapping of analogous processes. Just as there is a syntax for programming, so there is a syntax for building. Using graphic icons to represent architectural virtual computers and syn-
The designing of architectural designs using visual operations is a natural way for architects to explicate and explore the syntax of building. The level of sophistication of a visual programming language can be extended by realizing that “icons” can be graphic elements such as lines or primitive solid geometric entities. Thus a computational process can be implemented by drawing or three-dimensional modeling. In fact, computational processes occur when you draw or create a three-dimensional model in a conventional CAD system. However, these processes seldom involve more than the creation of a data structure. What is being suggested here are computational processes that go beyond the creation of data structures. The same rigor used in writing computer programs can be applied to the design decisions made by architects. Architects can program a building, as they often do in another sense.

The desktop metaphor used in the Macintosh™ operating system interface can provide a visual basis for structuring this interaction. Icons representing architectural virtual computers can be presented on the screen. The designer can then click on one of the icons, drag it over to another icon and click on it to set an interaction in motion. For example, an icon can represent the spatial enclosure of an auditorium. There can be another icon that represents a grid. When the designer clicks on the grid icon and then drags it and clicks on the auditorium icon, the spatial enclosure is formatted according to the grid values. Further, a spatial enclosure (modeled as a virtual computer) can interact with a structural system (also modeled as a virtual computer) to define the dimensions of the structural system. The structural system can then compute the dimensions of its individual members. Each of these virtual computers should have methods defined for interaction with all other relevant virtual computers. This will define a language of interaction for each virtual computer. Synthesizing an architectural design by the interaction of virtual computers uses a connectionist model and generates designs by using what Bakhtin (1981) calls dialogic mediation.

Another model for structuring the interaction of virtual computers is the use of Petri nets. Reinsig (1992) discusses Petri nets in detail in his book on the subject. Petri nets were introduced in the 1970s as channel-agency nets. (See Figure 3.) The channels were the passive components, and the agencies were the active components. The state and behavior of virtual computers can be mapped onto channels and agencies, respectively. Petri nets were introduced to overcome the drawbacks of flow charts that were being used to model computational tasks. Petri nets are used in the initial stages of system design to model hardware, communication protocols, parallel programs and distributed databases, all of which involve complex interactions. Petri nets are effective not only to model computer systems but any organizational system. Architectural design can be conceived of as organization, hence it can be represented by Petri nets. Petri net modeling enables the checking of the formal correctness of the system being modeled. It also enables the derivation of precise mapping rules that can be used to generate algorithms from the formal specification of the system. Petri nets are strict bipartite graphs with the under-
lying mathematical model and semantics. The use of Petri nets ensures that a mathematical model can be established for the system being modeled. This makes the system amenable to computation. There are different kinds of Petri nets. These include condition-event nets, place-transition nets, individual-token nets and channel-agency nets. These nets are used to model different aspects of systems. It is possible to switch the model of a system from a channel-agency net to the other kinds of nets. These different kinds of Petri nets and their relationships are described in detail by Reisig (1992). The study of Petri nets is becoming increasingly important and there are annual international conferences on the applications and theory of Petri nets. As such Petri nets are a promising model with which to structure the synthesizing interaction of virtual computers for architectural design.

**case study**

Some of the concepts presented in this paper were used by the author in the implementation of design systems for proscenium-type auditoriums. In papers by the author (1992a, 1992b, 1996), these design systems were presented. The design systems generate preliminary designs of fan-shaped and rectangular proscenium type auditoriums based on acoustical, functional and programmatic parameters. The auditorium is modeled as a virtual computer. The various acoustical, functional and programmatic parameters are its data. Procedures that compute the spatial parameters of the auditorium are its operations. The systems were developed using the Visual Works™ object-oriented programming environment. Visual Works™ is a programming environment based on Smalltalk, a completely object-oriented programming language. Smalltalk was developed by researchers at the Xerox Palo Alto Research Center in the late 70s and early 80s.

The design systems include many entities that are modeled as virtual computers. They work in conjunction using parallel models. A list of the entities is as follows: Auditorium; Auditorium Plane View; Auditorium Frame View; Planes; Light Source; View Point; Point Vector; Translation Matrix; DXF Exporter; EASE Exporter.

The main entity to be modeled as a virtual computer is the auditorium. Subsidiary entities are planes, light sources, view points, point vectors and translation matrices. Related entities are the views of the auditorium (wire-frame and shaded-plane) and the export file generators (DXF and EASE). These entities (virtual computers) interact to create the designs for the auditoriums.

How is their interaction parallel? The parallel model of message passing is built into the programming environment. The “channels” of communication are the various messages defined as part of each entity which is a virtual computer. The application works entirely by message passing from virtual computer to virtual computer. Take, for instance, the procedure (see Figure 24) that generates the shaded plane image of the auditorium. The image of each plane with its screen coordinates and color is computed by the plane (modeled as a virtual computer). This is a parallel process involving array processing because there is no mechanism available to place a breakpoint in the image generation of plane 1/2, for instance. This breakpoint has to be included in the procedure defined as part of the plane, whereby it will
occur in all the planes. The application also reflects shared memory processing. The independent variables of the auditorium are available to all the other entities (virtual computers). Synchronization of the variables is maintained in the auditorium entity. A diagram showing the relationships among the different variables of the auditorium is shown in Figure 11. If the auditorium is a virtual computer, then this is like an IC (integrated circuit) in that computer. Different paths are traced from the inputs to the spatial image of the auditorium. Because the input of the various parameters (independent variables) can happen in any order, the process of generating the spatial design is a pipelined process. Changing the order of inputting variables does not affect the result generated. A view of the computer screen when running the software is shown in Figure 12. The software runs on a virtual machine which makes it platform independent. It can run on Windows, Macintosh or UNIX operating systems.

**Conclusion**

The main tasks of the designers and implementers of computer-based design systems using virtual computers are the definition of architectural virtual computers and the structuring of their interaction. The modular approach of using virtual computers allows the designers and implementers of computer-based design systems to concentrate on each virtual computer and its characteristics. The behavior of architectural entities and their interaction with each other in different contexts has to be modeled. This is a very complex task initially but has its rewards later. The structuring of the interaction of architectural virtual computers involves the formulation of design strategies. Established design strategies can be organized into systems which can be reused. These strategies will clearly reflect how architects synthesize designs. Formulation of the design strategies will elucidate architectural design processes. This will lead to the development of new kinds of design systems in architecture. There are many benefits in using the concept of the virtual computer in the development of computer-based design systems in architecture. Some of the benefits that can be realized were discussed in an earlier paper by the author in the context of object-oriented systems (1992).

**References**


Representing Architectural Design
Using Virtual Computers


61