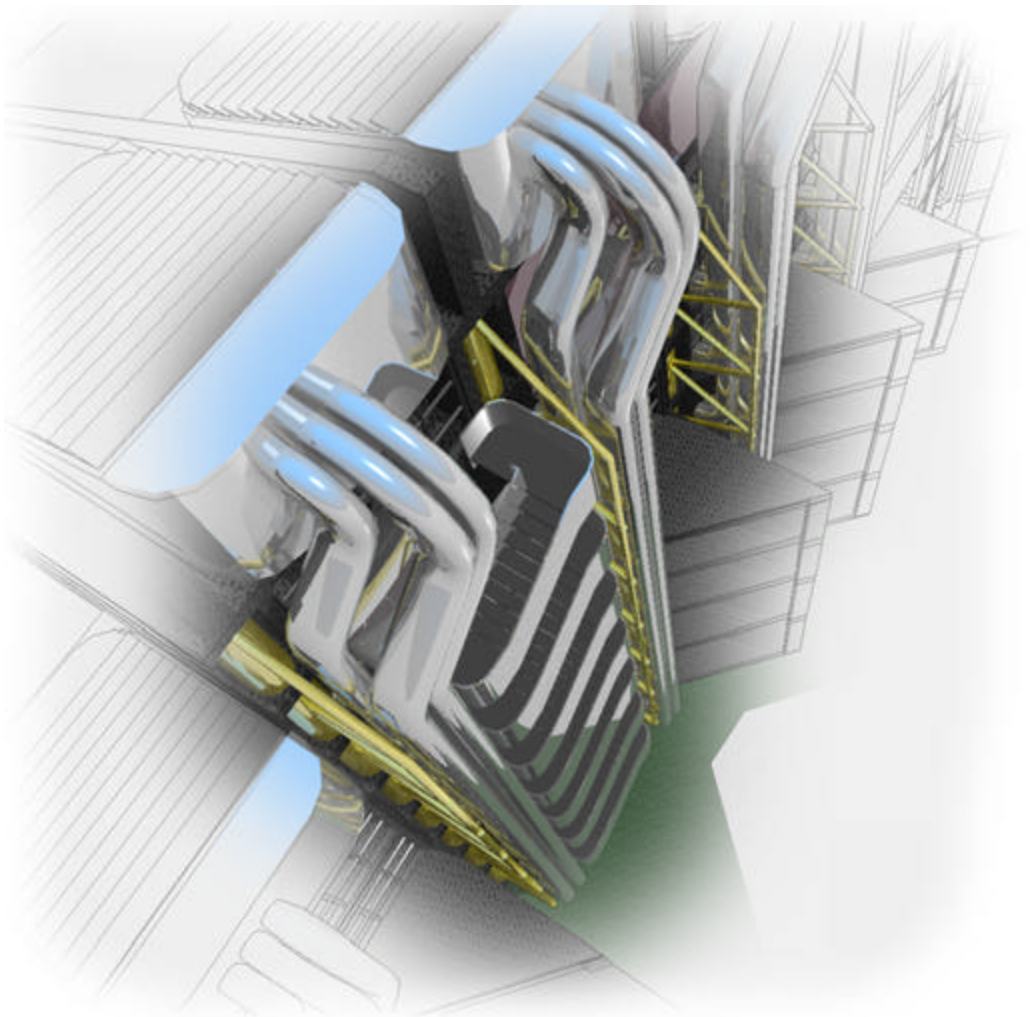


Remote Facility Management:

Possible Applications for Kit-of-parts Building Systems

A. Scott Howe, architect[1]

Summer 1996

Abstract

The central theme of our research has been the derivation of design principles and tools for the development of advanced kit-of-parts building systems. We have been involved with a variety of topics covering the entire design-to-demolition building life span, including virtual buildings (computer representations), automated construction, and facility management. In this paper we will be discussing facility management as it applies to the computer-based monitoring and control of lighting, appliances, and other architectural devices located in a building.

First, in Part I a brief survey on current off-the-shelf remote and computer-based facility management technologies is discussed. Telephone, Internet, wireless, and cable-based technologies for communicating commands to (and receiving feedback from) architectural devices will be touched upon.

In Part II an experimental Internet facility web page and server-based software engine for monitoring the current condition of a building will be described. The World Wide Web has shown great potential as a media for distributing current information in real time. Home pages for

companies and individuals have become dynamic resumes and portfolios open to the rest of the world. The experimental web page described in this paper was developed as a home page for a facility rather than an individual or company. The web page contains a Virtual Reality Modeling Language (VRML) model which is linked to the actual facility. One may 'walk-through' the VRML model and click on certain locations to download real time temperatures and other data.

Lastly, in Part III some insights into possible applications of remote facility management technologies to the development of advanced kit-of-parts building systems will be discussed. Virtual libraries of pre-engineered components with embedded device information could be browsed by designers and assembled into virtual buildings in the design stage. The virtual building could be used for analysis or walk-throughs, and linked to the actual building via the embedded device information for facility management purposes.

Introduction

In the course of researching concepts for representing kit-of-parts building systems virtually in the computer, the question arose regarding the fate of a virtual building once the real one is constructed. Having served its purpose as a design, analysis, and visualization tool for architects, engineers, contractors, and clients, should the virtual building be retired? First of all, in order to answer this question, the term 'virtual building' needs to be defined. Smith gave a concise definition (Smith 1992):

> "A finished building is the best model of what the building will be... We don't usually have the luxury of constructing the complete building as a model or prototype. If we could do this we could identify its deficiencies and our errors, and then tear it down and construct an improved version. What is needed is a Virtual Building that we could design and construct as many times as we please... A virtual building will reside in a computer environment and will look like and perform as much as possible in the same way as the actual building when constructed."[2]

A rich model produced by architects in the design stage could respond to analysis the same way the actual building would. Space planning, structural, accoustical, daylighting, thermal, and energy analysis could be conducted to provide insight into more efficient designs before the actual building is built. Performance data could be compared with optimum mathematical models to produce designs that need less artificial heating and cooling, are naturally lighted, and save energy. Early space planning computer programs such as CRAFT (Armour & Buffa), ALDEP (Seehof & Evans), DOMINO (William Mitchell), and STACKing plan (Charles Reeder) have evolved over the years into several heuristic and optimizing algorithms that have become somewhat of an industry standard.[3] Several commercial applications available today have space planning capabilities and can produce models which can be expanded into working drawings. The models are linked with a building's knowledge base and allow for some analysis. An ideal virtual building is getting easier to accomplish with the richer models made possible today.

Construction documents, the conventional products produced by architects, usually lose their usefulness after the building has been constructed. With ever richer computer models produced for representing the actual building, can their usefulness be extended after construction ? One aspect of facility management that has become common is the use of computer aided design (CAD) models produced by the architects as working drawings, for the purpose of furniture layout planning, inventory databases, and other facility management applications.

A more advanced facility management technique is using the computer to monitor and control lighting, equipment, appliances, and other devices in the building. Often refered to as 'building automation', this technique has until recently been out of reach of all but the most costly facilities. Part I of this paper will discuss automation from the remote monitoring / control point of view.

Part I: Remote Facility Management Technologies

This section will discuss remote facility management applications using four communications technologies: telephone, wireless, Internet, and cable. Briefly each technology or 'media' will be described followed by example applications. Some of the topics will be dealt with in more depth than others, allowing us to expand on ideas that are perhaps not well represented.

Telephone

The telephone system represents the most widely accessible communications network in the world. A telephone connection is generally set up between two specific points and consists of a full duplex line (where both parties can transmit and receive simultaneously). The system was originally designed to carry sounds within the range the human ear is capable of discerning, but has since been expanded to handle the sending and receiving of data as well.

A very interesting standard which developed in conjunction with the telephone system is the Dual-tone Multi-frequency code (DTMF). DTMF refers to the tones generated by a telephone during the dialing sequence. The DTMF code was originally developed by Bell Labs for the purpose of providing a robust way of broadcasting digits over the network for the purpose of announcing the dialed number to the switchboard. The old pulse technique used by rotary phones proved to be unreliable for long distances and over microwave connections. It was determined that a series of tones lying within the same frenquency range as the human ear can discern would be far more reliable. Each digit could have a single tone, but there would always be a chance that random sound on the same frequency could cause erroneous signals. That is where the dual tone system came in.

1	**2**	**3**	**A**	697 Hz
4	**5**	**6**	**B**	770 Hz
7	**8**	**9**	**C**	852 Hz
*****	**0**	**#**	**D**	941 Hz
1209 Hz	1336 Hz	1477 Hz	1633Hz	

Figure 1: DTMF code frequencies

The human voice ranges from around 2000 Hz for men, to 3000 Hz for women. Figure 1 shows the DTMF frequency grid which lies outside that which the voice usually produces, but within the range that can be discerned by the human ear. As the digits are layed out in a grid, each row and column represents a different frequency. When one of the digits are selected, ‘5’ for example, two tones, one of frequency 770 Hz from the row ‘5’ is a member of, and another of1336 Hz from the column ‘5’ is a member of, are generated.

The DTMF code is of particular interest to facility management because it is generated by common devices which occur in most every home. The DTMF signals can be encoded on one end and decoded on the other for a form of digital communication. Computers and modems decidedly do a better job at this, but the availability and portability of telephone handsets, especially cellular phones, allow them to function as a form of remote control device. On the human end, a controller device can be called using the telephone. When the connection is made digits are inputted in sequences which can communicate with the controller. The controller decodes the DTMF tones into ASCII commands and matches them to preprogrammed sets of instructions. The DTMF grid includes four extra digits which normally do not appear on the common telephone handset, called A, B, C, and D. These digits will prove useful in facility management control over wireless media (as will be explained later), but are not available using telephones.

Common pieces of hardware that can access the telephone network include the telephone (consisting of a speaker, microphone, and a DTMF keyboard), data modem, facsimile, and special dial-in controllers (discussed later).

Wireless

The transmitting and receiving of certain electromagnetic waves through space is known as wireless communication. Though generally refered to radio for communication purposes (two-way, amateur, etc.), for the purposes of this paper wireless can include television, satellite, microwave, and Amplitude Modulation / Frequency Modulation (AM / FM) radio. Where telephone requires wires and an established network infrastructure, wireless in its most basic form only requires a powered transmitter on one end and receiver on the other (with the exception of satellite which needs at least one intermediate repeater -- the satellite -- in order to qualify as such).

Above sound waves, electromagnetic waves used for wireless (as defined in this paper) range from 3 KHz Very Low Frequency (VLF) to 3000 GHz Extra High Frequency (EHF). Frequency is defined in Hertz (Hz) where1 Hz is defined as one sine wave per second, 2 Hz is two sine waves per second, etcetera. Amplitude is the height of the wave. The transmitter consists of an oscillator which first creates a wave (fc) of the desired frequency. The information to be transmitted (fs), for example a voice which is 2 KHz, is added to the oscillator-created wave in one of two ways: by altering or modulating the frequency from the pure wave form (FM) or by modulating the amplitude of the pure wave form (AM). The modulating can be done by adding (fc + fs) or subtracting (fc - fs) the information wave from the carrier wave.

The question may be asked why isn’t the voice wave transmitted directly instead of piggy-backing it onto a carrier wave? The answer is that the voice wave would be limited to the broadcasting of the frequency of the voice, which would put limits on the power and range of the broadcast. This brings up another question, which is in the case of FM wouldn’t the added voice wave change the frequency to another frequency? The answer is yes, but the continuous incrementation from the 3 KHz to 3000 GHz mentioned earlier is divided into channels defined by what is called band width. The band width of a single channel gives leeway on either side of the pure frequency of plus or minus 3 KHz to make a total band width of 6 KHz. It is inside this band or range that the carrier wave plus (or minus) the sound wave must lie. Broadcasting only one side of the band is

called Single Side Band (SSB) and using the entire 6 KHz band is called Double Side Band (DSB). Broadcast radio usually uses DSB. By standard, transmitters and receivers are manufactured to produce and receive carrier waves in the middle of the band, so there is no industry-wide overlap or slight shift of channels between manufacturers.

In most cases, the operation of a transmitter requires a license to broadcast. This includes professional and amateur (ham) radio operation. Amateur transmitters are readily available in the 1.9 MHz to 1200 MHz range. Amateur operators often use the shape of the wave itself to traverse long distances. Using the 3 MHz to 24 MHz range, the curves of the waves begin to approximate the curve of the earth. Tuning in just right will allow one to broadcast 'around the corner' so to speak. In recent years, amateur radio operators sometimes use their equipment to connect to personal computers in order to send data packets over long distances, usually powered by nothing more than a twelve volt car battery.

Waves can also be bounced off things. Using the density of the atmosphere at different altitudes and aiming just right, some waves can be bounced back to the earth for long distance communication. Some amateur operators take pride in bouncing waves off the moon. Another form of this technique is the use of repeaters. Repeaters are actually receiver / transmitters that receive a transmission, strengthen its wave, and transmit it again. Television and radio use this technique to get their broadcasts to far away homes. Everyone has seen broadcast radio towers on tops of mountains or at other places: these are repeaters. Satellites are another form of repeater. Repeaters can work singly or in multiple chains, where a signal is sent from repeater to repeater until it reaches its destination.

One of the disadvantages to wireless communication is that, contrary to closed-linked telephone, transmitters broadcast by 'disturbing the ether' much the same way that a thrown rock creates waves in a pond. The waves go in every direction and anyone with a receiver can pick them up. Wireless communication is not private. This problem can be solved in a number of ways. One way is to scramble the message and decode it on the receiving end (which is illegal in some cases). Another way is to use extremely low power to reduce the range of a broadcast (assuming that the receiver is close by). A third way is by the use of extremely powerful directed broadcasts such as microwave.

Being able to receive any wave being broadcast can pose another problem, which is receiving unwanted transmissions. To partially solve this, some receiver transmitters are equipped with, or can be equipped with a filter called a Continuous Tone Coded Sub-audible Squelch (CTCSS) card. CTCSS card produces a continuous tone which is *not* within the range a human ear can discern. When information is broadcast, the tone is added to the wave. The receiver is set to the same frequency, so it will only put transmissions containing that tone on the speaker. There are 38 fixed frequencies reserved for the CTCSS tone, so there is still a possibility that another transmitter may be using the same CTCSS tone. Another filter which can be used in conjuction with the CTCSS takes advantage of the DTMF capabilities and is called Dual Tone Signal Squelch (DTSS). This method sends a series of DTMF tones at the beginning of the transmission to notify the receiver (which is set to receive transmissions containing the same tones) of an incoming transmission. Using these two filters in parallel, it is almost impossible to receive unwanted transmissions.

The use of DTMF tones brings us back to the topic at hand, which is facility management. For facility management purposes, wireless can provide an extremely powerful medium for remote device monitoring and control. Using an extremely low energy level, commands can be sent to devices within the building equipped with receivers. Broadcasting FM signals over the wiring system of the building provides a way of communicating with any device which is plugged into the power grid. Wireless connection of computer equipment such as keyboards, ethernet hubs, and mouses are increasing in popularity. These devices send out a wave at an extremely low energy

level so the range is only limited to across the room in most cases and will not send unwanted signals out of the building. Hardware for device control and monitoring for the purpose of 'building automation' are available off-the-shelf. These will be discussed later.

The transmitters and receivers used in wireless communication often have built-in tools that lend themselves toward remote control. Two-way and amateur radios contain a DTMF keyboard, much similar to a telephone. Contrary to telephones, radios have the extra A, B, C, and D digits as well. Sending DTMF tones over radio waves, devices can be controlled with the keyboard simply by decoding the tones into ASCII commands to be sent to a controller. With the right hardware, DTMF tones can be decoded right from the speaker of the radio. Decoders, such as those manufactured by Genave are available on the market from around $300 or so. Using DTMF tones 'on the air' always has risks attached, since the possibility exists that they can be intercepted and duplicated by outsiders. Transmitting DTMF tones in sequence over several frequencies may reduce the risk.

Internet

One of the most exciting technologies of today is the Internet. The potential for decentralized hard core computer applications is literally without limit. Whatever can be connected and controlled by the computer can be controlled remotely via the Internet. In 1995, we developed an Internet-based kit-of-parts library of VRML components[4]. The components themselves were located on a server in Tokyo, Japan, and the building on a server in Michigan. The building model had no components in it, only path names and instance transformation parameters. When the model was viewed over the Internet, each of the parts are collected all the way from Japan in real time and loaded into the local machine at their proper location[5]. In other research, we successfully controlled a robot from Denmark and Japan to both construct and disassemble a Michigan-based model kit-of-parts building. The commands were made using a laptop computer and local telephone line, with a local telephone call to a neighborhood Internet service provider in each of the respective countries[6]. Use of the Internet has tremendous potential for facility management applications as well. Part II of this paper describes an experimental web page developed for a facility, which allows WWW visitors to 'walk-through' a VRML model of the facility and download current temperature data and other information.

The Internet started in 1969 as a way to link geographically separated U.S. military computer installations. It was essentially a connection between local networks consisting of computers manufactured by different companies. In order to allow them to communicate with each other, a neutral interface called Internet Protocol (IP) was developed. Over the years the Internet has gobbled up local network after local network until it has become the global infrastructure of recent years.

Today the Internet is a network of millions of computers located all over the world. Each of the computers have a unique identification number called an IP address, which consists of four clusters of digits separated by periods (for example, Kajima Corporation's gateway server IP address is 126.4.21.90). From left to right, each of the number clusters represent ever tighter domain designations until the last one, which represents the number of the machine itself. Each of the individual networks of which the Internet consists has a 'gateway' server which allows the network to link to other networks. These series of dedicated servers located in various locations throughout the world form the backbone of the Internet. The individual network is called a Wide Area Network (WAN), and represents a certain domain referenced by one of the number clusters in the IP address. The WAN consists of Local Area Networks (LAN) which are the next smallest domain numbers, followed by the number of the computer itself.

The Internet Protocol allows for another more intuitive address in parallel with the IP address. The secondary address has all of the same information as the IP address, but is represented by names

instead of numbers (Kajima Corporation's gateway address in this intuitive form is 'kajima.co.jp'). In this way each computer has an address unique from any other in the world. When combined with path name conventions used within the computer, *each individual file* located on the computer also has its own address unique from any other document located anywhere in the world. E-mail is facilitated this way, where a specific directory is created on a computer which is named the unique name of the user. For example, A. Scott Howe's unique name at Kajima is 'ash'. A directory named 'ash' is created on the Information Systems division server 'ipc', to make the e-mail address out to be 'ash@ipc.kajima.co.jp'.

In the same path name convention, serial ports also have a unique address. Devices connected to the serial ports are then made accessible directly to the Internet using the unique address, or indirectly using data dump files. The entire unique address of a document, file, or device is called its Uniform Resource Locator (URL). This is very significant to remote facility management applications.

Now that we have described the unique address system of each and every computer, device, and file on the Internet, the next item that needs to be discussed is the development of the World Wide Web. The World Wide Web (WWW) is not a network, but is an interlinking of all the individual files located on the computers of the Internet. The WWW consists of another protocol which is ASCII text-based and completely platform independent. This means that every computer ever made which recognizes the ASCII standard can read the data which is passed between each other via the Internet Protocol, because it is all text-based. Data is actually transfered using File Transfer Protocol (FTP) which is another standard adopted generally by the computer industry. The WWW is built upon a special text-based language called HyperText Markup Language (HTML) which allows the the varous documents and files all over the Internet to be linked to each other.

Special browser applications installed on the local computer are designed to view these various documents and transparently facilitate FTP functions for sending and receiving copies of the documents. The linking is accomplished by inserting special tags into the document which spell out the entire URL of another document or file. The called document or file is copied transparently via FTP and displayed on the screen. The special tags that allow this to happen are invisible when viewed with the browser application. In this way the World Wide Web consists of many interlinked 2D documents.

The way the browser application calls for documents and files by their URL is called 'pulling' the data, since it is all initiated by the program running on the local computer. Another important way of passing documents is by 'pushing'. Pushing is facilitated by invoking another program located on another computer and causing that computer to do some work. These remote programs are called 'cgi' programs. The cgi's can do anything from generate new documents spontaneously based on time or other outside parameters, to operating robots and specialty equipment. Like the files, documents, computers, and devices, cgi's also have their own URL.

Another recent technology on the Internet and World Wide Web is Virtual Reality Modeling Language (VRML). Where HTML is a text-based language for 2D documents, VRML is a text-based language for 3D models. Equiped with the right browser, one can walk through, spin, and select parts of VRML models. Just as the HTML creates clickable text that leads to other documents, VRML creates clickable objects which bring other objects. In addition, the clickable VRML objects can call HTML documents, invoke cgi's, and otherwise do most everything HTML is capable of in three dimensions[7].

Having such an advanced common protocol system in place, linked to the millions of other computers on the Internet, possibilities for advanced facility management applications become limitless. Any kind of device that can generate some kind of signal or data can be given a URL. Temperature sensors, HVAC systems, audio, video, and many other devices can be connected

to the computer via data loggers or other specialty hardware. In this way, the actual devices themselves are accessible by clicking on text or VRML representations. Navigating through the building and clicking on a model lamp can activate the lamp or turn it off. Clicking on a VRML model television can turn the real television on and actually display the moving video image on the screen of the VRML representation (this could be done using a cgi program which controls both the television and the operation of a connected frame grabber).

Just as individuals and companies have home pages on the Internet, buildings and other artifacts can also have home pages which tell about themselves or their current state. It is conceivable that the day will come where all of our automobiles have their own home pages as well, with special logon screens and live video of the view the driver sees. Want to see where your friend is? Look up his car's home page, watch the live video, and find out he is driving down Main Street.

Cable

The potential for using cable television in remote facility management applications is also quite large. Currently, most cable television systems are installed with only a single cable with one-way transmission from the cable company to the home. Some systems, however, have two cables and are thus capable of both transmission and reception.

There are some recent inventions that take advantage of single cable setups, which hold potential for use with remote facility management. Several devices available this year combine single-cable television and telephone lines. The single cable allows reception of signals and the telephone allows transmission of signals. The devices connect to the Internet for browsing and manipulation of the standard HTML documents just as a computer would, without the computer[8]. Another device allows web pages to be piggy-backed onto standard television signals for "more information". When a certain topic is discussed on television, the web page reference would provide sources for further information for those who are interested[9]. While these ideas in themselves would not necessarily become tools for facility management, the remote monitoring and control possible on the Internet would be applicable here as well, without the overhead of the computer.

Upcoming technologies using double cable systems, however, are much more exciting. New cable modems that can be hooked up to the cable system will carry tremendous amounts of data for the transmission of smooth video on the Internet. Using cable modems, the possibility of having a server at each household suddenly becomes a viable option (much more so than standard computer modem connections would allow). A server at each household means that every home could function as a small network and run local cgi's for transmitting video and operating various devices in the home.

Remote Facility Management Applications

In the context of this paper, applications of remote facility management technology or 'automation' will be confined to the monitoring and control of architectural devices such as HVAC systems, lighting, and temperature sensors, as well as devices and appliances located within the facility. The entire point for using automation technologies is to be able to control devices remotely or automatically according to some predetermined plan. There are various reasons for having such control, which include convenience, providing for security, and saving energy. To facilitate the control, there must be a way to communicate the intentions and to actuate them on the device. Both of these require special hardware at both ends which can range in sophistication depending on the work to be done. The controlling of various devices is known as Direct Digital Control (DDC) technology.

On the controlling end, a simple adaptation of a television hand-held remote control unit coupled with an infrared receiver can be an inexpensive way of communicating commands. More complex

methods of control include the use of computers that automatically communicate commands based on external factors including time of day, state of the building, temperature, performance specifications, etcetera. On the actuating end, simple devices which open and close power flow, increment values, generate currents, and generally perform some form of work are necessary. These devices are called actuators. Actuators take either binary or analog commands. Since the signals being received are usually from a computer or controller, the signals tend to be fairly low voltage (up to 5V). These signals are not sufficient to power the architectural device itself, but can be used to flip relays that switch power on and off. This form of control is a digital signal, but using software functions that control many of these devices, sophisticated boolean AND's and OR's can be implemented. Analog devices actually use the electric signals to power devices, such as valves, dampers, and motors. Another form of control which is actually digital, but which can simulate analog characteristics is called 'pulse width modulation'. Using this technique, timed binary pulses can be continuously derived which correspond to an integer count that the actuator interprets as a range of values. This latter technique is rather inexpensive, but not as precise, and can be used to brighten or dim lights and control thermostats.

In all control situations there must be some sort of feedback in order to verify whether the control command was realized or not, or to discover the current state of the device. In the case of remote control where the persons communicating commands can actually verify through their own senses that the work has been done, no additional hardware need be added to the system. But in the case of computer control or remote control from some other location, additional features must be added to the system to facilitate the feedback. These additional devices are called sensors. Sensors used in the building industry can include devices which measure temperature, pressure, velocity & flow, humidity, air quality, motion, and light levels. Sensors can range in cost, sensitivity, and ruggedness depending on the application. Like the actuators, sensors are either digital or analog. An inexpensive analog temperature sensor is the thermocouple, where two different type metals twisted together in a wire produce a current when heated. The current must be interpreted with a voltmeter and translated into digital data for the computer to use. Digital sensors use similar innate properties in different materials to detect differences and produce digital signals to the computer.

Having introduced actuators and sensors, the newest generation of 'smart' actuators and sensors can be described. In addition to the functions mentioned above, these smart devices have microprocessors and means of communicating sophisticated digital signals, for the purpose of interacting with higher-level computers and controllers. In the case of actuators, simple digital signals would be received from the computer or controller. The microprocessor interprets the signals and causes the actuator to do its work. Microprocessors built into sensors interpret the input immediately and send off binary signals to the computer or controller. The advantage of having smart actuators and sensors is that the devices can become more standardized in a 'plug and play' form. The media of communication between the device and the computer can be anything that allows the digital signals to be carried. This technique can include the power wiring itself, RF wireless, infrared, and all sorts of 'hard wired' media such as twisted pair, fiberoptics, and cable[10].

The use of smart devices opens up other doors. Instead of central control, the microprossesor present in each of the devices allows communication among the devices as well. A sensor device can send a message directly to an actuator device for some desired effect. For example, smart motion detectors can send a message to a smart switch to turn on a light in a dark room when someone enters. Controllers and computers are not eliminated, but become tools for human interface, programming of advanced sequences, storing of event records, gateways for remote control, and computer aided analysis.

The use of smart actuators and sensors requires smart organization. The building DDC industry has taken technology developed in the computer industry and put it to use networking these

smart devices in such a way that each and every one could be controlled separately without being hard-wired. A 'fieldbus' standard has been developed which gives each device its own identity for precision use in control situations. The term 'fieldbus' refers to a series of nodes connected along a single line. One signal goes out to all the devices, but only the summoned device answers. Just as computers are given unique addresses and individual files have unique pathnames on the Internet, standards have been developed which give each smart device a unique address in the system. More sophisticated standards even have unique addresses that are worldwide for the very purpose of interfacing with the Internet itself.

Some advanced research on the use of smart devices has been conducted by various organizations around the world. In the United States the National Association of Home Builders conducted research on a centralized hard-wired automation system called Smart House[11]. In the Netherlands, the Dutch House of the Future built in the Autotron Theme Park in Rosmalen has become a research project for Dutch businesses, institutions, and universities. In Japan Tokyo University has been working in partnership with top electronics industry representatives on the ongoing TRON project[12], which is another hard-wired example[13]. Another example is the research conducted by the Electronics Industries Association (EIA) which gave birth to the Consumer Electronics Bus (CEBus) standard mentioned below. The CEBus work is genuine remote smart objects which can plug into existing wiring systems as easily as use hard-wired installation[14]. In Norway, Gjovik College has teamed up with industry to construct a scale model town which is entirely controlled by LonWorks technology (described below). The town includes controls for everything from control of traffic, street lighting, domestic heating, access and various functions in automobiles[15].

Automation Standards

The building industry has been plagued with a lack of common interface and standard. In recent years several distinct standards have evolved into workable solutions. Older applications of the technology have facilitated limited control, but because of the inexpensive hardware are still very much a viable solution for remote control automation today. Other newer applications have more sophisticated control built into their smart devices but are slightly more expensive. The one characteristic of all the standards that will be discussed in this paper is that they are all 'plug and play' design, which can be easily expanded as the need arises without special wiring or redesign of the system.

Before discussing actual commercial applications of the technology, a word on some overall standards is in order. One standard which governs overall networking is the ISO's Open Systems Interconnection Basic Reference Model (OSI). This standard consists of seven layers of independent data communication addressing different aspects of protocol. These layers are: 1) Application Layer, where the function or purpose for communication is established, 2) Presentation Layer, where the proper language or protocol for the intended destination is established, 3) Session Layer, where the priority of the communication is established, 4) Transport Layer, where the verification strategy is established (to assure the proper delivery of the communication), 5) Network Layer, where the destination address is established, 6) Data Link Layer, where the method of conveyance is established, and 7) Physical Layer, where actual data and media (cable, twisted pair, etcetera) are established. These layers provide a basis for the design of EIA-232 (RS-232) connections and protocols for sending packets of information between devices.

In the building industry, the American Society of Heating, Refrigeration and Air-conditioning Engineers (ASHRAE) have established a standard protocol for the purpose of enabling various smart devices, computers, and controllers to interoperate. The standard, known as ASHRAE Standard 135-1995, was coined Building Automation and Control NETwork (BACnet)[16]. BACnet attempts to address the seven layers from a building industry perspective. The Application Layer

describes a collection of network-visible objects which define the function or purpose of the device to the outside without being concerned about the actual implementation of the technology. The Presentation Layer consists of 'services' which provide commands for accessing and manipulating the objects. The other layers consist encoding concepts, networking protocols which actually can include the communication of any smart device to any other smart device anywhere on the Internet, and the media by which the information is carried.

Commercially Available Applications

There are several systems and standards used for remote facility management or 'building automation' applications that have developed over the last few years. The standards include X-10, CEBus, LonWorks, and a few others. In order of increasing sophistication, three of these standards are described below:

X-10 standard: X-10 is essentially an exceptionally low cost remote control system. The standard is an older one that predates the BACnet standard, but is quite popular due to its low cost. The system relies on the building's electrical wiring system and has no need of special wiring or installation. The system is entirely plug-in in nature, where switch modules plug into the nearest electrical outlet, and control units plug into any other convenient oulet in the system. Lamps, televisions, heaters, pumps, and other devices plug into the switch modules. Each of the switch modules can be set with their own unique address, up to a total of 256 different addresses. The controller can be manually operated or automatically set to send out certain signals to specific switch modules to either turn them on, off, or dim. When the switch module receives the signal, it closes or opens the electricity flow to the device allowing it to turn on or off[17].

X-10 switcher modules can be used to control any device that takes a current. Some common uses are for lamps and household appliances, but may also include electrically operated solonoid valves for sprinkler systems and bathroom plumbing, HVAC thermostat control, and security systems. The controller can also be connected to a personal computer for advanced programming. Timed events and graphic point and click control can be facilitated. An infrared (IR) interface unit is also available that can cause X-10 signals to trigger IR commands.

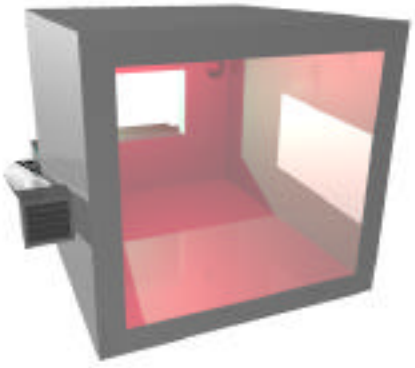
X-10's disadvantages include the fact that only on, off, and dim control is possible. There is no built-in feedback capacity to automatically determine whether the device was actually turned on or off. The X-10 standard includes a product that partially remedies this deficiency, called HomeBase. HomeBase works in conjunction with a PC and performs some of the functions of a data logger, with up to 16 digital and 8 analog inputs, and 8 relay outputs. Various temperature sensors, moisture sensors, etcetera can be connected to the I/O interfaces for feedback. X-10 switch modules start at around $10, with the beginning cost of a simple controller only around $15. Computer control interfaces begin at around $30 or thereabouts.

CEBus standard: CEBus is an open architecture set of protocols for making devices communicate through power line wires, low voltage twisted pairs, coax, infrared, RF, and fiber optics. Contrary to the simple on, off, and dim of the X-10 units, the CEBus standard allows devices to communicate via data packets that vary in length depending on how much data is included. The minimum packet size is 64 bits, but could go up to hundreds of bits in length. In addition to on, off, and dim, other defined controls include play, rewind, fast forward, pause, skip, and temperature up or down one degree. The CEBus standard uses addresses which are manufactured into the hardware in the factory, which have a possible four billion combinations[18].

LonWorks standard: The LonWorks standard also allows devices to communicate in a network environment through power line wires, low voltage twisted pairs, coax, infrared, RF, and fiber optics. LonWorks standard also allows the devices to communicate with data packets varying in length from 64 bits to hundreds of bits long. There are 32 predefined commands which include on, off, dim, play, rewind, fast forward, pause, skip, and temperature up or down one degree. LonWorks standard allows for more than 32,000 devices in a network, with addresses preset in the factory or alterable at time of installation[19].

The LonWorks standard uses the Neuron chip manufactured by Motorola and Toshiba. The system fully incorporates ASHRAE BACnet standard and is endorsed by them. With LonWorks, all devices communicate via a common language, 'LonTalk', regardless of the manufacturer.

Comparing the three standards, X-10 is decidedly the least expensive, but does not lend itself toward large facilities. Residences, small businesses, churches, and small commercial establishments could take advantage of the affordability and incorporate X-10 technology into their remote facility management systems. X-10 modules might be vulnerable to power surges and other signals penetrating in through the electrical system from the outside (even though filters are available). On the other hand CEBus and LonWorks are of a more robust design for protection against power surges, and are built for commercial use in networks with greater numbers of devices. All in all the LonWorks standard seems to have the best backing and may prove to outlive or become incorporated in all the others. There are integrated computer applications that combine the control of X-10, CEBus, and LonWorks together into one package. CyberHouse by Savoy Automation[20] interfaces all three standards and costs around $200 for the software.



Part II: Facility Management Experiment: A Facility Web Page

In the course of remote facility management studies, it was determined that an experimental research project should be conducted which demonstrates an example application of the technology. Among the five main media technologies of telephone, Internet, wireless, and cable, it was decided to develop an experimental software application that monitors and displays facility information over the Internet. Conceivably, a building or facility would have its own web page from which building managers and other visitors can observe its current state from any location in the world. Part II of this paper describes an experimental web page set up for a certain facility at the University of Michigan. An appendix to this report explains how the software can be used to establish web pages for other buildings.

FMonitor Overview

The experimental software consists of various independent modules which run over a networked computer environment. The most visible part of the system is the home page, which is a panel made up of three separate frames.

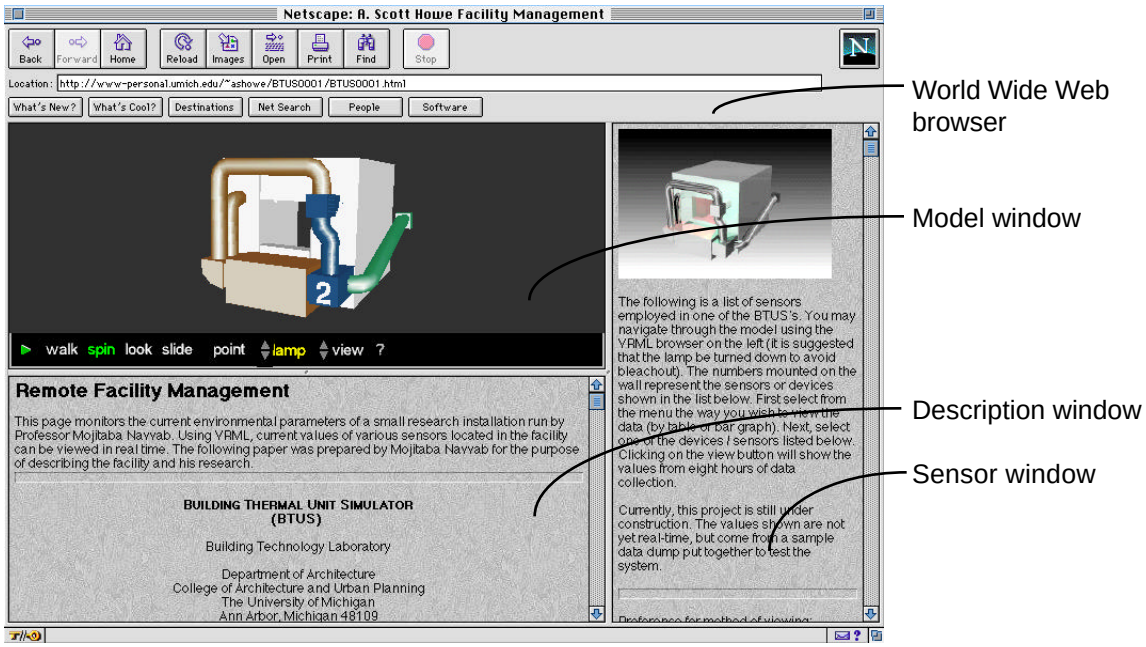
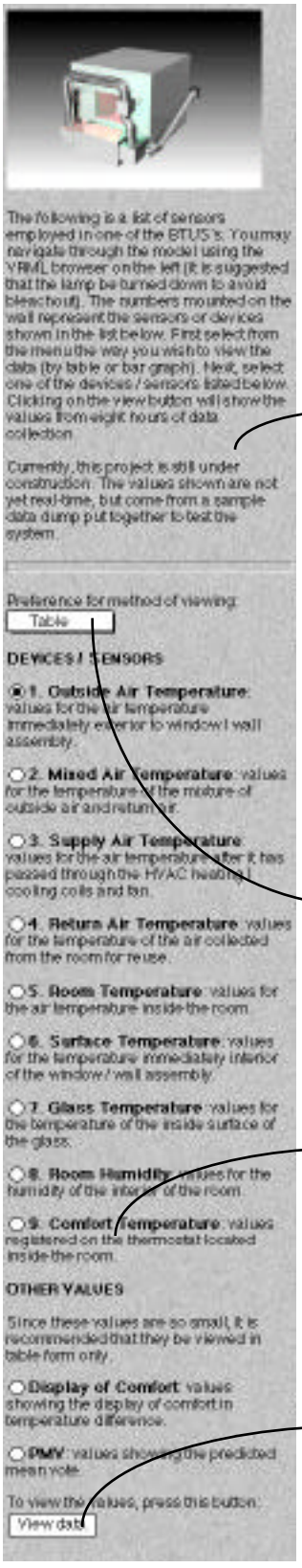


Figure 2: FMonitor Web Page Control Panel



The upper left frame is the model window. The model window displays a Virtual Reality Modeling Language (VRML) model of the building, which building managers and other visitors can use to walk-through or fly-through the building. The VRML model will have numbers showing the location of sensors. It is also possible to configure the model in such a way that clicking on the numbers will display data from the sensors.

The lower left frame is the description window. The description window is a plain HTML document, and can contain anything needed to describe the building. The description window is scrollable, so documents can be any length.

The right-hand frame is the sensor window (shown left). The sensor window is also a scrollable HTML document, but contains a form which calls the main engine of the page, 'fmonitor.cgi'. The sensor window contains a list of the sensors, each of which have radio buttons which can be selected. In order to view the data, one would select one of the sensors on the list, choose either 'table' or 'graph' format, and click on the 'view data' button. The sensor window would then be replaced by either a table or a graph, depending on which was selected. This data is provided by 'fmonitor.cgi' which is called by the following HTML anchor (the URL shown would be the location on the Internet where 'fmonitor.cgi' is compiled):

```
<FORM METHOD="POST"
ACTION="http://spring.eecs.umich.edu/ashowe/fmonitor.cgi">
```

'Table' or 'graph' selection menu. The HTML coding for the selection menu is as follows:

```
<SELECT NAME="view_option">
<OPTION SELECTED> Table
<OPTION> Graph
</SELECT>
```

Sensor list. Each sensor has two names that distinguish it from other sensors: 1) the 'NAME' is the building name which consists of exactly eight characters, and 2) the 'VALUE' is the sensor point name which consists of exactly six characters. Typical HTML coding for a sensor item is as follows:

```
<INPUT TYPE="radio" NAME="BTUS0001"
VALUE="D27036"><B>9. Comfort Temperature</B>
```

'View data' button. The HTML coding for the button is as follows:

```
<INPUT TYPE="submit" VALUE="View data"></FORM>
```

Figure 3: Sensor Window

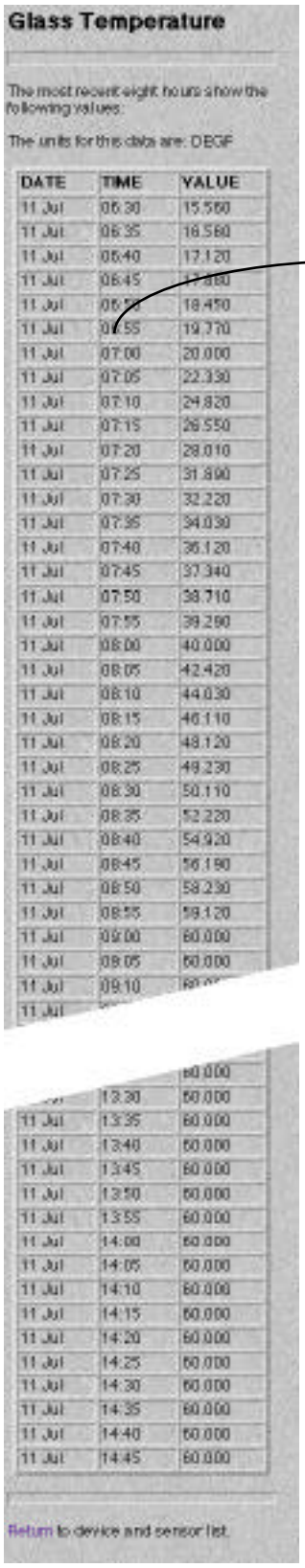


Figure 4: Table View

Clicking on the 'view data' button will display either a table or a bar graph. FMonitor will open a file containing a data dump of the last eight hours (100 samples), extract the requested data, and display it in sequence.

Table view. The table view shows the units of the data value to be displayed, and simply lists the date, time, and value for each of the 100 samples.

Graph view. The graph view shows the units of the data value to be displayed, and lists the date, time, nearest positive integer to the value, and a graphical bar reflecting the value.

Each of the views are scrollable so the 100 samples can be viewed on the web page.

Though the program currently only has the capability to display the data in these two forms, future versions of FMonitor will have added capabilities. Anyone viewing the web page will be able to select several sensors at once and view the data on a multiple line graph for comparison. For the line graph, FMonitor will not display all of the data, but will only show a profile of the activity over a certain period of time. The multiple line graph could then be captured on a screen capture and outputted by printer.

Using a web page for the user interface, it is possible to access the same data on any platform using common browser tools. The rest of this section will be devoted to a detailed look at the program, and will show how to set up a home page for any building.

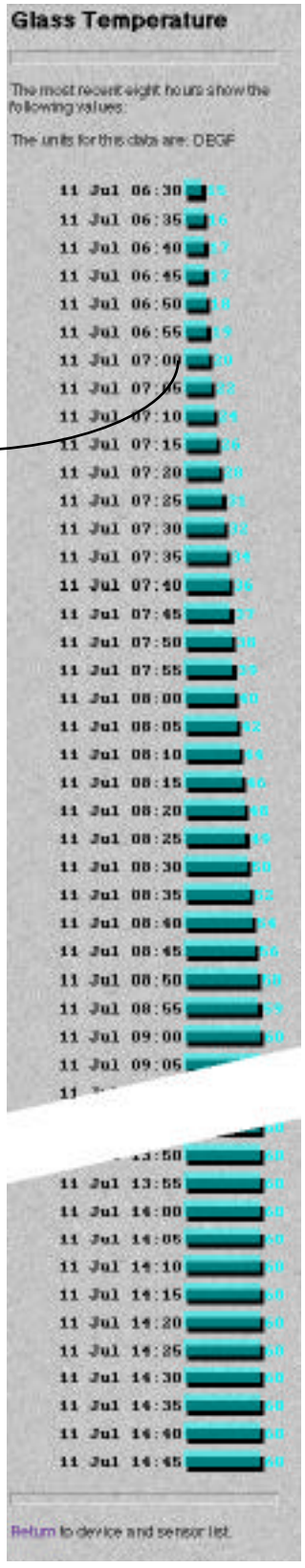


Figure 5: Graph View

The Experiment

The experiment was conducted in partial response to the hypothesis, “Can the Internet be useful as a tool for remote facility management?” The scope of the experimental research was confined to the monitoring of a facility, but the possibilities of adding controls in future experiments will be discussed in part III of this paper.

The experimental software was used to establish a web page for a small research facility maintained by Mojitaba Navvab at the University of Michigan School of Architecture + Urban Planning. The facility, called Building Thermal Unit Simulator (BTUS) is a pair of small 8’ x 8’ isulated chambers which have one removable wall which faces south and are exposed to the outside environment. The purpose of the chambers is to provide a controlled environment for experimenting with various types of windows and wall assemblies. The windows and wall assemblies would be attached to the south facing removable sides of the chambers and compared with each other for thermal insulative qualities. The chambers have their own independent HVAC systems in order to simulate the function of actual buildings.

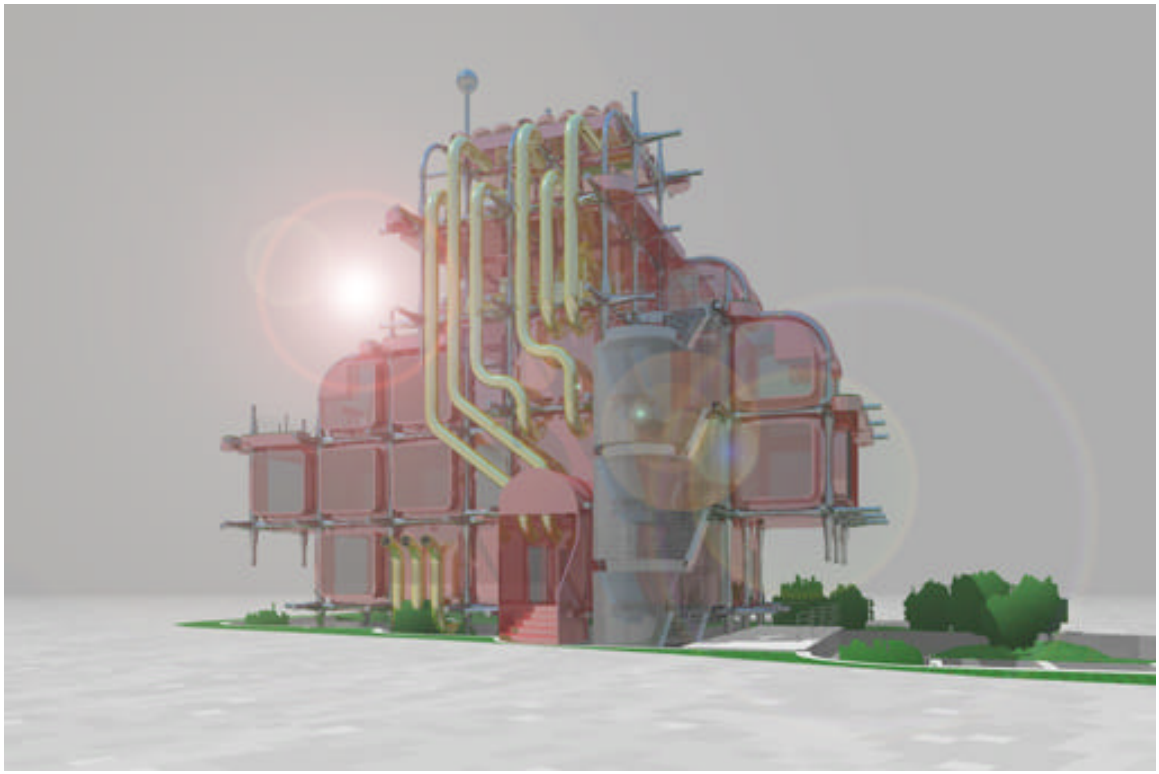
Each of the BTUS facilities are connected to 30 sensors which produce a measurable current. The sensors include temperature, humidity, air flow velocity, and water flow sensors, as well as binary (on / off) sensors to tell whether fans and pumps are running, and pressure sensors to sense how far open or closed dampers are. In this experimental web page, only eleven of the sensors are to be monitored:

1. Outside air temperature
2. Mixed air temperature
3. Supply air temperature
4. Return air temperature
5. Room temperature
6. Surface temperature
7. Glass temperature
8. Room humidity
9. Comfort temperature
10. Display of comfort
11. Predicted mean vote

The sensors provide raw currents to a data logger, which is a hardware device which translates the currents into ANSI text data and dumps it into a text file on a computer. As of this writing, the dumping of the data is done manually from the data logger to the computer. The scope of this experiment only covers the monitoring of sensors as picked up by the data dumps already provided and does not cover the automated dumping of the data itself. For the purpose of this experiment (which is still ongoing) a sample data dump from July 1996 was obtained. It is assumed that automated data dumping will be implemented for the BTUS facility soon.

Conclusion

The experiment was felt to be a success in that the data was observable in a selective manner from common Internet browsers. Any user located anywhere in the world could selectively view the data with a Java enhanced World Wide Web browser, and collect the data for whatever purposes. The experiment shows that data can be collected from a data dump, selectively manipulated, and observed via various means from a remote location. It is conceivable that independent software engines can be developed which collect data (similar to ‘fmonitor.cgi’) and rechannel the selected reformatted data to other software engines located elsewhere on the Internet for analysis or automated control. Part III of this paper addresses some of the possibilities a decentralized network-based system naturally presents.



Part III: Toward the Development of Kit-of-parts Building Systems

Object Oriented Programming (OOP) provides a method of programming which constructs applications out of independent building blocks of code called objects. The interfaces between the objects are preserved as abstract standards that all connections must strictly adhere to. What goes on inside the objects and how they fulfil their function is hidden. When the application is assembled, the various objects can come from different programmers, with code implemented in various styles, as long as the interfaces are strictly observed.

In the world of computers, object oriented programming has proved itself to be an efficient way to quickly build and maintain a programming application. Since the various objects are independent, they can easily be modified and replaced later with more efficient modules which do the same job, without changing the rest of the application. Since the interfaces are strictly observed, the object becomes a plug-in plug-out component in the overall application.

This same ‘object oriented’ mentality seems to be popping up in other industries as well. As a matter of fact, the computer industry is actually late in the game. The idea of having standard interfaces to actual objects has been around for a long time. A simple bolt for instance, follows a standard which will allow it to be used over and over again, no matter if the final construction is a building, a car, a plane, a ship, or each of them in sequence. The standard interface defines the dimension of the threads, but the type of material or color of metal can be anything and it would still fit.

The field of architecture has also experienced an ongoing ‘object oriented’ revolution. The standardization of building material sizes is one example of this. The development of prefabricated and modular systems is a higher order ‘object oriented’ approach. The manufacture of large modular components that can be snapped or plugged together has proved to be an efficient way of building that allows quick assembly and superior maintenance.

Some of the early pioneers of modular construction include Buckminster Fuller. In 1937 Fuller patented the Dymaxion bathroom which was a premanufactured module that included toilet, sink, and bath in one package[21]. Fuller's work partially influenced Archigram[22] and the Metabolist movement whose work was exemplified in Expo '70 in Osaka, Japan. Kisho Kurokawa's Takara Beautilion consisted of premanufactured space frame trusses that could be bolted together, with plug-in stainless steel capsules[23]. Fuller, Archigram, and Metabolism influenced the Pompidou Center of 1977 by Richard Rogers and Renzo Piano, which consisted of premanufactured structural components and mechanical systems[24]. In 1986 Richard Rogers and Norman Foster took plug-in architecture to new heights with the two projects, respectively, Lloyd's of London and Hong Kong Bank Headquarters[25].

In all of these cases the architects attempted to express the building components as a factory manufactured kit-of-parts with a rigorous set of interfaces that could be assembled or plugged into the building. In this section, we would like to conceptually bring the pure concepts of object oriented programming and plug-in architecture together in a meld of the 'spiritual' software and 'physical' hardware. The 'Virtual Building I' project introduced here is conceptual but fully manufacturable. It is hoped that the system can serve as an example to illustrate the possibilities advocated in this paper.

Nature's Kit-of-parts

As a design concept, the 'Virtual Building I' project was based on ideas found in nature, which we will call Nature's Kit-of-parts. One of the most basic of all building blocks in the system is the atom (Figure 6). An atom can be thought of as a very simple micro-electronic device which runs on electricity and light energy. They receive and transmit similar to miniature wireless transceivers, communicating to each other by electromagnetic waves of disturbed media and orbiting and jostling each other in a sea of self-generated ether. They consist of two parts: things that act and things that are acted upon. The nucleus of the atom provides mass and substance which can be used to build things. Electrons are automated around the nucleus in an orbit or cloud and provide the little family with a means of intercourse with other nuclei. This very simple system, the most basic being hydrogen, is the kit-of-parts from which the entire universe as we know it is constructed. The universe is fundamentally digital in its very nature. Each electron, proton, neutron, etcetera can be thought of as being programmed to react a certain way when bonded with others of its number. In this way the same atom responds to temperature differently when by itself or coupled with other atoms in molecules.

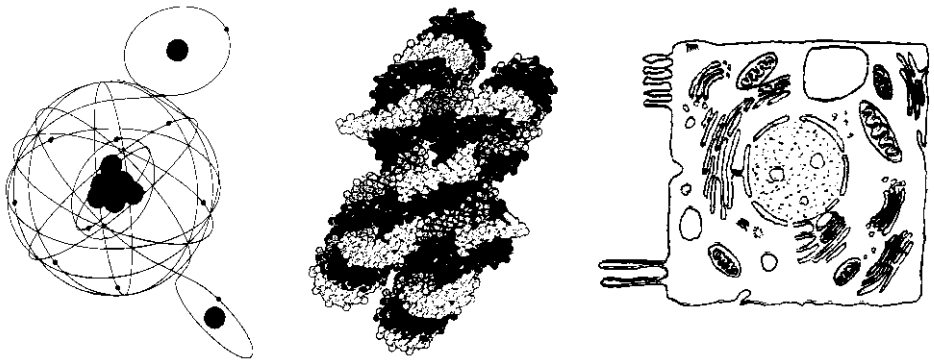


Figure 6: Hydrogen atom, DNA, and cell

A key to life in all its complexity is DNA which is an assembly of molecules. DNA is a virtual existence of a being, a plan for how the actual being will be shaped and constructed. The DNA contains coding which represents all the attributes and information required to describe the being

and its functions. In possession of DNA, each cell "assembly" knows its function and also that of all the other cells in the entire being. There are bounds set in which each cell is free to operate according to the instructions of the DNA. The actual shape of the cell is determined by the virtual shape held within the DNA.

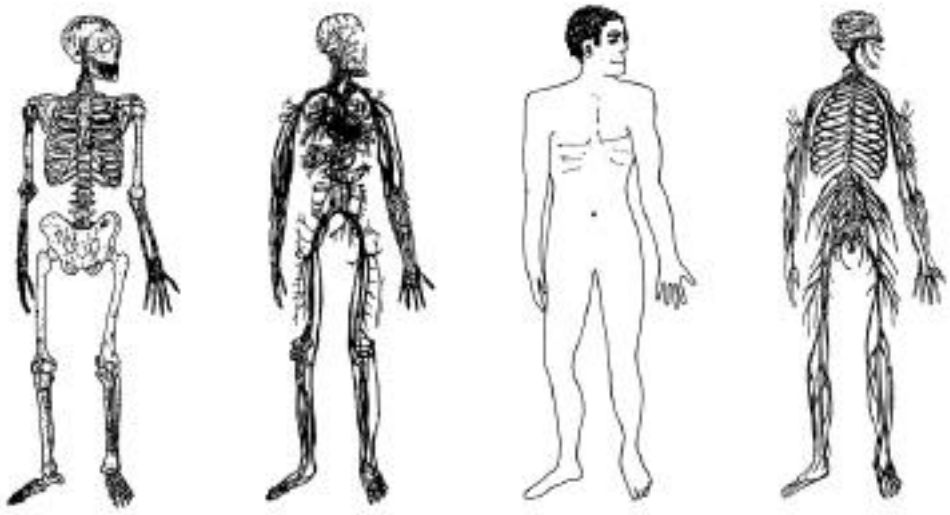


Figure 7: Body systems

In this very simple way, parts make up assemblies which make up entire systems, such as the body's structural system, mechanical system, exterior envelope system, and power and information systems, among others (Figure 7). The systems all work together in perfect integration for the well being of all. External influences from the environment, such as rain falling or vibration on the floor of a moving train are sensed by certain of the body's systems. Information is passed to the central processing unit (the brain) where data is assessed and reactions determined. Then automated machinery (muscles, etcetera) carry out the reactions according to plan, via information received on the neuro network.

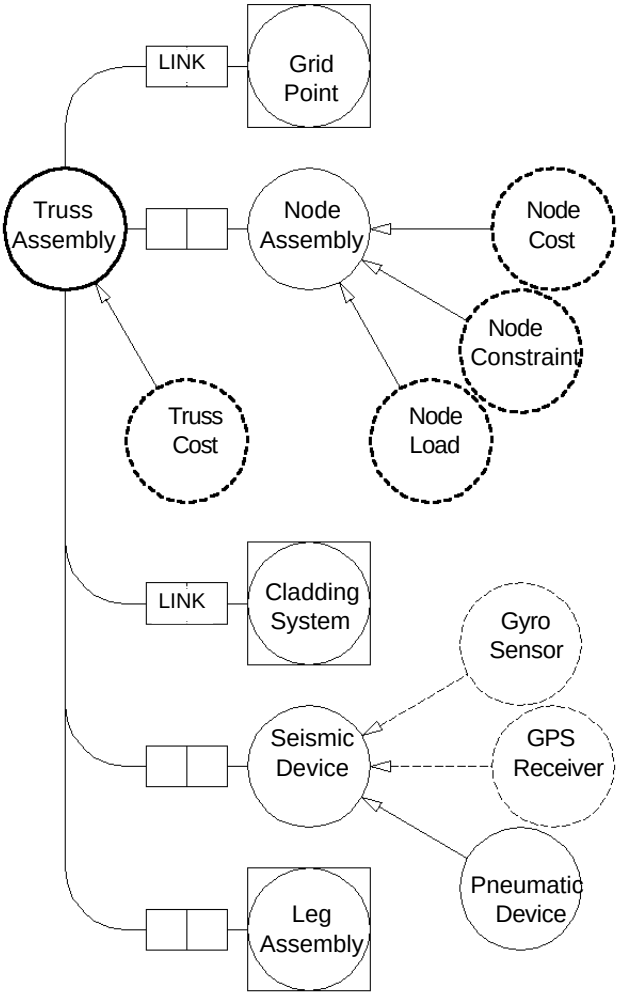
Virtual Kit-of-parts

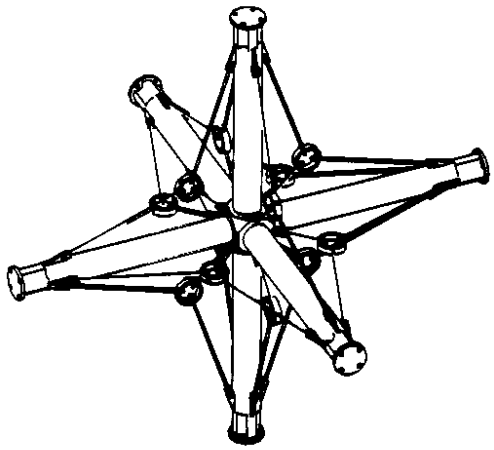
The most basic elements in buildings, like atoms in nature, are its separate parts. One such part, a simple bolt, may have many, many attributes. Not only its shape is important, but the type and strength of the material, its cost, the manufacturer, the knowledge of how it fits together with other parts (such as a nut for example), its overall place in the whole system, etcetera are all attributes of a bolt which allow it to be used as a part of the building.

Parts, assemblies, systems and buildings can be represented by an "architectural DNA" much similar to that which appears in nature. Various types of circles symbolizing parts, assemblies, systems and attributes can be connected with linking symbols defining the roles of relationship. Every individual member and its attributes can be defined this way and keyed back and forth to other diagrams. This DNA (which includes geometrical data within its attributes) can be stored in a computer and viewed via output devices which convey the condition of the building much similar to the way we observe actual buildings. The entire set of "architectural DNA" of a particular building can be refered to as a virtual building.

A collection of virtual parts with their associated attributes becomes an assembly, much similar to cells in the body. Many of these virtual cells assembled together become an organ or system such as the building's skeleton (structural system), circulation system (mechanical), skin system

(external envelope), and neuro network (power & information system) among others. The virtual organs all work together in good integration for the well being of the building. Influences from a virtual environment, such as simulated wind, temperature or seismic activity can be produced inside the computer. The "architectural DNA" of a virtual building can be analyzed before the building is actually built, to determine whether the design is adequate or not. Virtual environmental information could be picked up by virtual sensors and passed to the master virtual building program where data is assessed and reactions determined. Then virtual machinery (louvers, ventilation systems, etc.) carry out the reactions according to plan, and their virtual performance is assessed.





A simple notation of 'architectural DNA' for this nodal truss could be as shown on the left.

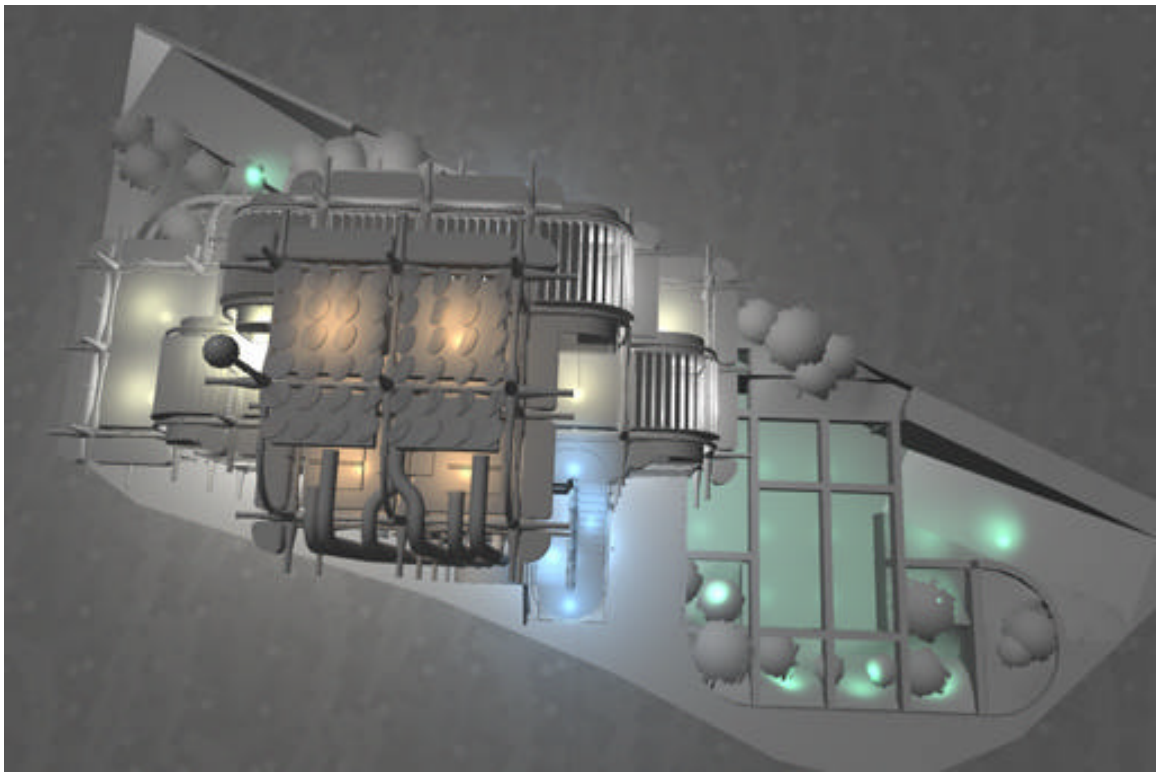
'Architectural DNA' would consist of C++ objects at every level which return their own VRML geometry and other attributes.

Figure 8: Architectural DNA

In the virtual building the 'architectural DNA' consists of C++ objects which represent actual components of the building assemblies. Depending on the information received, the objects return their own geometries in the form of VRML models, or other attributes necessary for analysis and design[26].

The knowledge base of the entire building project would be held in the "architectural DNA". There is now much research being conducted to attempt to create virtual buildings, or systems which can support the modeling and data bases required for virtual buildings. Most of the research involves fitting a computer model to a modern buiding of today with all its millions of parts. The lack

of success of many of these projects is due to the fact that computers are just not yet powerful enough to handle such a large task. Among other problems, it is extremely difficult to link state-of-the-art computer technology with building technology developed, in some cases, hundreds of years ago.



Real Kit-of-parts

Kit-of-parts architecture involves organizing a building into assemblies of standard easy-to-manufacture parts. The construction of the building is generally carried out on the assembly level as opposed to the part level. The architect defines a parts library describing every major assembly in the building. The library could be similar to an interactive LEGO set, which has many standard parts already available but is open for additions by creative users trying to meet new needs. The assemblies are conceived in a systematic way, perhaps based on a certain increment or size. Standard connections between the assemblies are defined, allowing greater freedom in the form itself: anything goes as long as the connection rules are observed. As long as rules of connection and increment are established, the number of possible shapes and appearance the parts could take is literally limitless. Rules of connection and increment may include requirements for materials, structural strength, force reaction paths, center of gravity, insulative qualities, transparency, etcetera.

In our 'Virtual Building I' conceptual design project, we have established a basic cell unit which can be plugged together to assemble a building. The basic unit is a nodal truss similar to that used by Kurokawa on the 'Takara Beautilion' at Expo '70 (Ross 1978). Nodal trusses are multi-directional in that beams and columns are engineered to be interchangeable. The truss is manufactured in one piece with six legs situated orthogonally from each other such that when joined with other trusses, half legs become full columns or beams depending on how the truss is oriented. Joining the trusses in the middle of spans places plug-in or bolted connections where the shear forces are weakest and moment is strongest. All other parts connect to the trusses in a lower order of heirarchy. The blocks of space surrounding the truss itself are also considered to be parts of

digital spatial volume which also have attributes (similar to an atom which generates its own space around itself).

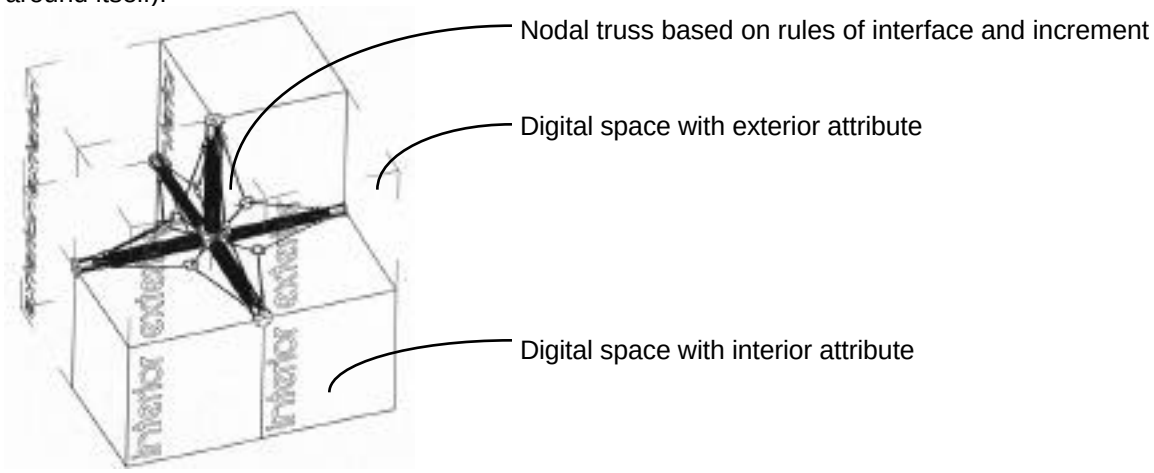


Figure 9: Digital spatial volume

The structural joint of the 'Virtual Building I' has bolted connections. Incorporated in the joint are two plug-in poles for AC power wiring which take the form of a concentric double ring. In the center of the rings is a plug-in plumbing connection with O-ring gaskets. The double ring power poles will allow the truss to be oriented in any way while preserving the proper pole connection. Cladding components and other assemblies would plug into the truss, which would have fixed receptacles halfway down each leg. Since all the components are sized according to the interface rules based on the truss-size increment, no more than one component would span between any two ninety degree legs.

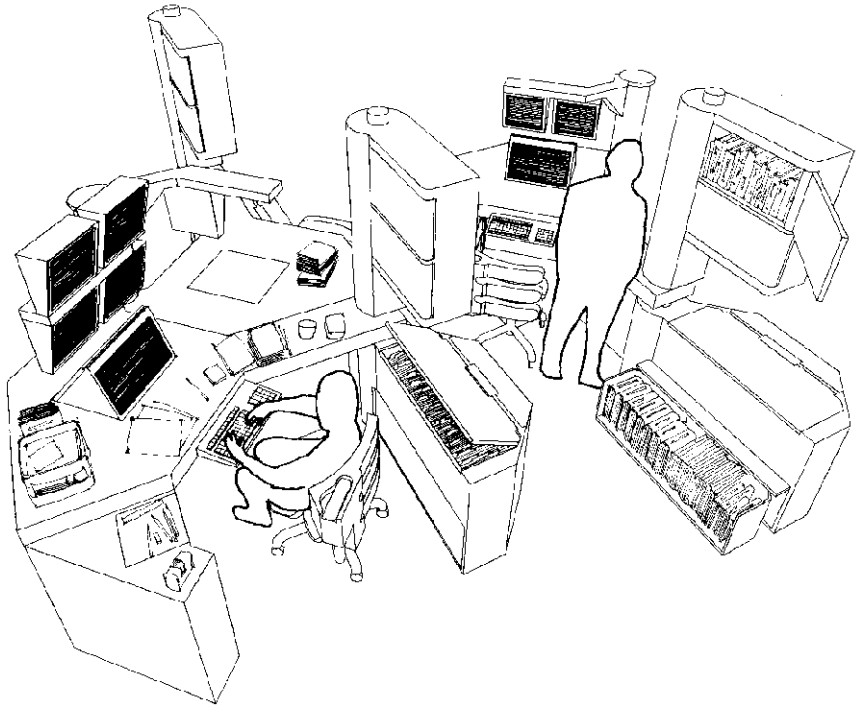
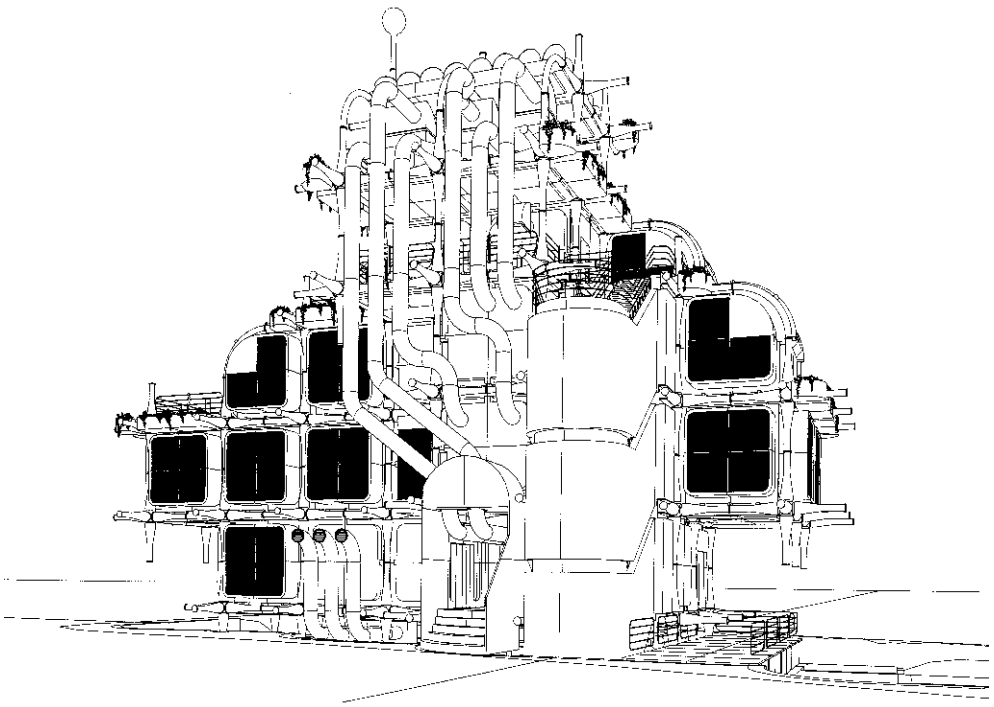


Figure 10: Plug-in office furniture

In a fully computer integrated building, a single central computer would function as a component of the building, and would hold the virtual building representation. Components could each contain a small microchip or simple CPU based on the CEBus or LonWorks standard. When floor assemblies with their pressure sensors, or wall assemblies with their temperature sensors, etcetera are connected to the truss, they are plugged into the AC power wiring of that truss. Each of these parts have microchips which would have a unique address. Though independent, each device would supply information about itself to the central computer. Through the central computer each actuator and data from sensors can be accessed. Sensors would be exposed to the interior and exterior digital blocks of space to monitor temperature, humidity, and other factors which could provide more information for adjusting the microclimate. Plug-in ducting systems and package HVAC units would be designed around zones based on the digital blocks of space.



In the Future

In a more advanced system, the microchip of each part would hold the "architectural DNA" of itself as well as be intelligent enough to store information on its orientation in relation to adjacent parts. Gyro sensors, global positioning system (GPS), lasers, etcetera could be utilized in the truss for locating it in the overall building. Robots or construction machines (properly designed and based on the rules of connection and increment) could be considered as temporary or permanent parts of the building and could assemble the building automatically. When a part is brought from the factory, the positioning system locates it in the virtual construction site of the computer and checks its intended location with the virtual building already completed. Assembly by robots is performed along predetermined parts paths to the final location. As the trusses are connected together, they immediately link with the next CPU in the network and to the host workstation and begin performing their intelligent functions. The building's data base which exists in the form of its "architectural DNA" could be accessed by the user at any time. Unlimited sized data bases of entire virtual cities could be spread out over the city itself, held within the simple chips of each individual part. Navigating through the city, a device giving view to virtual reality could link into the network by wireless and read all the geometrical DNA of parts within a specified range. As the navigator moves forward, new DNA is read in the front, and old data is dumped in the back

eliminating the need to contain all the database in one piece of hardware. The state of things far away could be observed virtually. Travelers could move through different cities, navigating in the same way as they would in their own town.

Conclusion

Using kit-of-parts concepts as a foundation for complete computer integration, the possibility of producing an actual / virtual building system becomes not only plausible but practical as well. Instead of dealing with dinosaurs of the building and construction industry with their millions of little pieces, a more technologically advanced parts library and "architectural DNA" could be a step toward full integration. All building components and assemblies may become more and more electronic in nature, incorporating X-10, CEBus or LonWorks chips. Frames, wall panels, flooring, etcetera may all be considered electronic devices in themselves, with sensors and actuators factory built into them.

Where the Internet and information superhighway boom has layed a foundation for tremendous advances in the development of vast cyberspace infrastructures, the uitlization of economical monitoring and control hardware devices such as those manufactured under the X-10, CEBus, and LonWorks standards could link it all back to the real world with never before imagined possibilities!

Appendix A: FMonitor Setup

Required Hardware

It is necessary that a system be in place which collects raw data from sensors and dumps the data in text form to any computer connected to the Internet with an IP address (**computer A**). The format of the data will be discussed later. A server which can hold the 'fmonitor.cgi' must also be connected to the Internet with an IP address, and can be the same computer as that which holds the data dumps (**computer B**). It is preferable that the server be a UNIX machine: this paper will not cover other machines. A third computer equipped with an http server is needed to hold the web page. This computer can also be the same as the one which holds the data dumps (**computer C**).

User Skill Level & Access Priveleges

To set up a home page that utilizes FMonitor, one need only have enough knowledge of HTML to be able to edit text in existing documents. Also, a little knowledge of UNIX is preferable in order to do simple tasks like create a directory, etcetera. It is also necessary to have certain permissions from the system administrator of the various computers involved: 1) assuming the data is already safely being dumped, the data dump files on computer A must be accessible to the public with read-only permissions. 2) The http server must be configured such that 'fmonitor.cgi' is given permission to function as a cgi program, called on from the outside. In addition, a '/tmp' directory with universal read / write privileges must be set up on the same computer B. 3) Computer C must be set up properly as a World Wide Web server, with all the necessary access privileges granted to the account holder.

FMonitor Component Overview

FMonitor consists of nine software components.Each of the components are listed below with short descriptions. Hard copy examples of each of the the files are included in the appendix of this report. In the following list, "edited by user" is underlined to show which files need to be altered for custom use.

1. ***'fmonitor.cgi':*** the software engine which runs the entire FMonitor system. Fmonitor.cgi is written in C and should be compilable on a variety of platforms. It has been tested on Macintosh and Linux machines. 'fmonitor.cgi' must be located in computer B. The URL address of 'fmonitor.cgi' must appear accurately in the 'sensor.html' document. 'fmonitor.cgi' can be compiled from 'fmonitor.c' coding.[27]

2. ***'XXXXXXXX_settings':*** a text file that is edited by the user. There is a separate 'XXXXXXXX_settings' file for each building, where "XXXXXXXX" is replaced by the name of the building. The name of each building must be exactly eight characters long and must coincide exactly with the 'NAME' parameter on the sensor HTML window. The settings file also contains a list of data sampling points, each consisting of exactly six characters and coinciding exactly with the 'VALUE' parameter on the sensor HTML window. All of the sensors listed in 'sensor.html' *must* be listed in the settings file. The home page URL is also listed in the settings file, as are the data dump file name and background image file name. Each settings file must be present in the same directory as the 'fmonitor.cgi' program in computer B.[28]

3. ***Data dump:*** the text file which is automatically dumped from raw data supplied by the sensors. 100 samplings of each data point present in the settings file must all be present in the data dump. The data dump file must be located in computer A.[29]

4. ***'XXXXXXXX.html':*** an HTML document defining the overall web page of the building. There is a separate 'XXXXXXXX.html' for each building, where 'XXXXXXXX' is replaced by the name of the building in the manner mentioned in (2) above. 'XXXXXXXX.html' must be located in a directory named '/XXXXXXXX' under the home page directory in computer C. 'XXXXXXXX.html' defines the three frames which contain the model, description, and sensor HTML windows. This document is edited by the user only to reflect the full URL address of the VRML model. The document should be copied and renamed for each building.[30]

5. ***VRML model:*** the model of the building in VRML format. The full URL address should be present in the 'XXXXXXXX.html' document. This model can be located on any computer in the world so long as it is connected with the Internet with a permanent IP address.[31]

6. ***'description.html':*** the HTML document which describes the building. This document can be anything and is edited by the user. 'description.html' must be located in the directory named '/XXXXXXXX' with the rest of the files pertaining to that building in computer C.[32]

7. ***'sensor.html':*** the HTML document which contains the selectable list of sensors, and form for evoking 'fmonitor.cgi'. This document is edited by the user, and contains each of the sensors listed in the 'XXXXXXXX_settings' file. Each sensor must list the 'NAME' name of the building and the 'VALUE' name of the data point. 'sensor.html' must be located in the directory named '/XXXXXXXX' with the rest of the files pertaining to that building in computer C.[33]

8. ***'chart.java':*** a Java applet that generates a bar graph from the data 'fmonitor.cgi' has extracted. 'chart.java' is a text file containing the Java coding, and must be located inside a directory named '/java' under the home page directory in computer C. 'chart.java' should not be altered by the user.[34]

9. ***Background image:*** the image used for the background on the HTML web pages. This file must be located in a directory named '/images' under the home page directory in computer C and should be either GIF or jpeg format. The image name should be typed into the settings file. If this image does not exist, the program will still run, but the default background color setting in the browser will take affect instead. One should not leave the settings file blank if there is no image.[35]

Setup of FMonitor

The setup sequence of FMonitor should be done in a certain order. Remember that computer A, B, and C can be the same computer, but for the purposes of this report it will be assumed that there are three separate computers. At the beginning of each step, the responsible person for executing the step is listed. The steps for setting up the program and establishing a web page for the first building are as follows:

1) User: arrange to have raw sensor data downloaded automatically into a text file in computer A. This text file is the data dump. The user should know the sensor locations, purpose, and data point names.

2) User: arrange to have a personal (or department) home page set up on an appropriate server computer C.

3) System Administrator: dedicate a directory in computer B that has permissions to run a cgi application.

4) System Administrator: configure the http server on computer B to reflect the status of the cgi directory. Computer B should have a C compiler.

5) User: load 'fmonitor.c' code into the directory in computer B.

6) User: compile 'fmonitor.cgi' by typing in the following at the UNIX prompt, carefully inputting spaces as shown:

```
cc -o fmonitor.cgi fmonitor.c
```

7) User: edit the 'XXXXXXXX_settings' file to reflect the home page URL, data dump file name, background image name, and list of data point names and descriptions for the subject building.

8) User: load 'XXXXXXXX_settings' file into the same directory as 'fmonitor.cgi' is located on computer B.

9) User: rename the 'XXXXXXXX_settings' file to the building name, being careful to assure that it is exactly eight characters long.

10) User: add three directories under the home page directory on computer C named '/java', '/XXXXXXXX', and '/images', where "XXXXXXXX" is renamed the same eight character building name used to rename the settings file.

11) User: arrange to have a VRML model of the building prepared (architectural students can be hired to do this) and loaded up into any computer on the Internet.

12) User: load 'XXXXXXXX.html' into the '/XXXXXXXX' directory on computer C, renaming it as the name of the building. Make sure the VRML model's URL address is correctly typed in.

13) User: load 'chart.java' into the '/java' directory on computer C.

14) User: prepare a background image (if desired) in the '/images' directory on computer C.

15) User: prepare or edit 'description.html' to reflect the building information as desired. Load 'description.html' into the '/XXXXXXXX' directory on computer C.

16) User: edit 'sensor.html' to reflect the proper sensor titles, descriptions, names ('VALUE'), and building ('NAME') on the selectable list. Make sure 'fmonitor.cgi' URL address is correctly typed in. Load 'sensor.html' into the '/XXXXXXXX' directory on computer C.

The procedure has many steps, but if done properly the program should work. Remember that FMonitor is at this writing not robust at all, and will not be able to handle typos in the HTML documents or settings file. All spellings should be perfect and consistent with the same case.

Adding Another Building Web Page

To add a second or third (or more) building to the first building, the required steps are a lot simpler. Follow these steps for each building:

1) User: copy the 'XXXXXXXX_settings' file, using the new eight character building name in place of "XXXXXXXX". Edit the file to reflect the new data dump file name and new data points. This file should be in the same directory as 'fmonitor.cgi' and the first settings file.

2) User: add another directory under the home page directory on computer C named the new eight character '/XXXXXXXX' building name.

3) User: arrange to have a VRML model of the new building prepared and loaded up into any computer on the Internet.

4) User: copy the first 'XXXXXXXX.html' into the new '/XXXXXXXX' directory on computer C, renaming it as the name of the new building. Make sure the VRML model's URL address is correctly typed in.

5) User: copy the first 'description.html' into the new '/XXXXXXXX' directory on computer C. Prepare or edit the new 'description.html' to reflect the new building information as desired.

6) User: copy the first 'sensor.html' into the new '/XXXXXXXX' directory on computer C. Edit the new 'sensor.html' to reflect the proper sensor titles, descriptions, names ('VALUE'), and building ('NAME') on the selectable list.

This should be sufficient to establish web pages for any number of buildings.

Appendix B: FMonitor Software Components

fmonitor.c

The following is the C coding for ‘fmonitor.cgi’. The C coding is first loaded onto computer B and then compiled on that machine using a native C compiler.

```
#include <ctype.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#define MAX_ENTRIES 1000
#define FOUND             1
#define NOT_FOUND     0
#define MAX_VALUES    100
#define LF 10
#define CR 13
#define WORK_FILE "/tmp/device_output"
void html_header (char *home_page, char *back_image);
void html_footer ();
void data_retrieve (char *device_name, char *url);
void html_table ();
void html_graph (int col, char *home_page);

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/// PROGRAM:    A. Scott Howe's DEVICE monitor program                          ///
/// PURPOSE:    This program creates an HTML file containing device description  ///
///                                 and current values collected from the device.   ///
/// LANGUAGE:   C                                                               ///
/// DATE:           1996.9.16                                                   ///
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
main(int argc, char *argv[])
{  int                  i, j, col, status;
   char                 value_name[256], data_url[256], home_page[256];
   char                 display_type[256], device_name[256], back_image[256];
   char                 word[500], building_name[256], settings[256], c;
   FILE                 *file1;

   for (i = 0; i < 12; i++)
   {            c = fgetc (stdin);
   }
   for (i = 0; i < 5; i++)
   {            display_type[i] = fgetc (stdin);
   }
   display_type[6] = '\0';
   c = fgetc (stdin);
   for (i = 0; i < 8; i++)
   {            building_name[i] = fgetc (stdin);
   }
   building_name[9] = '\0';
   c = fgetc (stdin);
   for (i = 0; i < 6; i++)
```

```
  {         device_name[i] = fgetc (stdin);
  }
  device_name[7] = '\0';
  *strcpy(settings, building_name);
  *strcat(settings, "_settings");
  status = NOT_FOUND;
  file1 = fopen (settings, "r");
  fscanf (file1, "%s %s %s", word, word, home_page);
  fscanf (file1, "%s %s %s", word, word, back_image);
  fscanf (file1, "%s %s %s", word, word, data_url);
  fscanf (file1, "%s %s", word, word);
  for (i=0; status != FOUND || strcmp(word, "end") == 0; i++)
  {         fscanf (file1, "%s", word);
            do {c = fgetc (file1);} while (isspace(c) != 0);
            value_name[0] = c;
            for (j=1; value_name[j] != '\n'; j++)
            {         c = fgetc (file1);
                      if (c == '\n')
                      {         value_name[j] = '\0';
                                break;
                      }
                      else
                      {         value_name[j] = c;
                      }
            }
            if (strcmp(word, device_name) == 0)
            {         col = i;
                      status = FOUND;
            }
  }
  fclose (file1);
  html_header (home_page, back_image);
  data_retrieve (device_name, data_url);
  printf ("<H2>%s</H2>\n", value_name);
  printf ("<HR SIZE=10>\n<P>\n");
  printf ("The most recent eight hours show the following values:\n<P>\n");
  if (strcmp(display_type, "Table") == 0)
  {         html_table ();
  }
  else
  {         html_graph (col, home_page);
  }
  printf ("<HR SIZE=10>\n<P>\n");
  printf ("<A HREF=\"%s/%s/sensor.html\">Return</A>", home_page, building_name);
  printf (" to device and sensor list.\n<P>\n");
  html_footer ();
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/// FUNCTION:     A. Scott Howe's HTML HEADER MAKER function                                                     ///
/// PURPOSE:      This function makes a header for an html file.                                                 ///
/// LANGUAGE:     C                                                                                              ///
/// DATE:         1996.7.19                                                                                      ///
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void html_header (char *home_page, char *back_image)
{  printf ("Content-type: text/html\n\n");
   printf ("<HTML>\n");
   printf ("<body background=\"%s/%s\">\n", home_page, back_image);
   printf ("<BODY>\n");
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/// FUNCTION:     A. Scott Howe's HTML FOOTER MAKER function                                                     ///
/// PURPOSE:      This function makes a footer for an html file.                                                 ///
/// LANGUAGE:     C                                                                                              ///
/// DATE:         1996.7.19                                                                                      ///
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void html_footer ()
{  printf ("</BODY>\n");
   printf ("</HTML>\n\n");
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/// FUNCTION:     A. Scott Howe's DATA RETRIEVAL function                                                        ///
/// PURPOSE:      This function retrieves data from text file which has been                                     ///
///               generated by others from sensor output. The data is loaded                                     ///
///               into an array and then dumped into a temporary work file.                                      ///
/// LANGUAGE:     C                                                                                              ///
/// DATE:         1996.8.19                                                                                      ///
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void data_retrieve (char *device_name, char *url)
{  int       date[MAX_VALUES], hours[MAX_VALUES], minutes[MAX_VALUES];
   int       status, column, i, j, k;
   float     value[MAX_VALUES];
   char      word[1000], arch_device[256], c, month[MAX_VALUES][4];
   char      unit[24];
   FILE      *file2, *file1;

   file2 = fopen (url, "r");
   status = NOT_FOUND;
   while (status != FOUND)
   {         fscanf (file2, "%s", word);
             while (strcmp(word, "Name") != 0)
             {         fscanf (file2, "%s", word);
             }
             column = 0;
```

```
            fscanf (file2, "%s", word);
            do
            {       fscanf (file2, "%s", arch_device);
                    column++;
                    if (strcmp(arch_device, device_name) == 0)
                    {       do {c = fgetc (file2);} while (c != '\n');
                            fscanf (file2, "%s", word);
                            fscanf (file2, "%s", word);
                            for (k = 0; k < column; k++)
                            {       fscanf (file2, "%s", unit);
                            }
                            do {c = fgetc (file2);} while (c != '\n');
                            do {c = fgetc (file2);} while (c != '\n');
                            do {c = fgetc (file2);} while (c != '\n');
                            do {c = fgetc (file2);} while (c != '\n');
                            for (i = 0; i < MAX_VALUES; i++)
                            {       fscanf (file2, "%d-%s %d:%d:", &date[i], month[i],
                                            &hours[i], &minutes[i]);
                                    for (j = 0; j < column; j++)
                                    {       fscanf (file2, "%f", &value[i]);
                                    }
                                    do {c = fgetc (file2);} while (c != '\n');
                            }
                            status = FOUND;
                            break;
                    }
            } while (strcmp(arch_device, "Units") != 0);
    }
    fclose (file2);
    file1 = fopen (WORK_FILE, "w");
    fprintf (file1, "%s\n", unit);
    for (i = 0; i < MAX_VALUES; i++)
    {       fprintf (file1, "%02d %s %02d %02d %7.3f\n",
                    date[i], month[i], hours[i], minutes[i], value[i]);
    }
    fclose (file1);
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/// FUNCTION:     A. Scott Howe's HTML TABLE function                                                            ///
/// PURPOSE:      This function takes values and loads them into HTML table                                      ///
///               format.                                                                                        ///
/// LANGUAGE:     C                                                                                              ///
/// DATE:         1996.9.16                                                                                      ///
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void html_table ()
{ int       i, date[MAX_VALUES], hours[MAX_VALUES], minutes[MAX_VALUES];
  float     value[MAX_VALUES];
  char      month[MAX_VALUES][4];
  char      unit[24];
  FILE      *file1;
```

```
  file1 = fopen (WORK_FILE, "r");
  fscanf (file1, "%s", unit);
  printf ("The units for this data are: %s\n<P>\n", unit);
  printf ("<CENTER>\n<TABLE BORDER=2 WIDTH=200>\n\n");
  printf ("<TR><TD COLSPAN=2 WIDTH=50>\n<B>DATE</B></TD>\n");
  printf ("<TD COLSPAN=2 WIDTH=50>\n<B>TIME</B></TD>\n");
  printf ("<TD COLSPAN=2>\n<B>VALUE</B></TD></TR>\n\n");
  for (i = 0; i < MAX_VALUES; i++)
  {       fscanf (file1, "%d %s %d %d %f", &date[i], month[i], &hours[i],
                  &minutes[i], &value[i]);
          printf ("<TR><TD COLSPAN=2 WIDTH=50>\n%02d %s</TD>\n", date[i],
                  month[i]);
          printf ("<TD COLSPAN=2 WIDTH=50>\n%02d:%02d</TD>\n", hours[i],
                  minutes[i]);
          printf ("<TD COLSPAN=2>\n%7.3f</TD></TR>\n\n", value[i]);
  }
  fclose (file1);
  printf ("</TABLE>\n</CENTER>\n<P>\n");
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/// FUNCTION:    A. Scott Howe's HTML GRAPH function                                                          ///
/// PURPOSE:     This function takes values and loads them into HTML graph                                    ///
///              format, derived with a Java applet.                                                          ///
/// LANGUAGE:    C                                                                                            ///
/// DATE:        1996.9.16                                                                                    ///
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void html_graph (int col, char *home_page)
{ int       i, scale, height=22, poop;
  int       date[MAX_VALUES], hours[MAX_VALUES], minutes[MAX_VALUES];
  int       value[MAX_VALUES];
  char      month[MAX_VALUES][4], color[24];
  char      unit[24];
  FILE      *file1;

  file1 = fopen (WORK_FILE, "r");
  fscanf (file1, "%s", unit);
  scale = 1;
  printf ("The units for this data are: %s\n<P>\n", unit);
  printf ("<applet code=\"Chart.class\" ");
  printf ("CODEBASE=\"%s/java\" ", home_page);
  printf ("WIDTH=250 HEIGHT=%d>\n", (MAX_VALUES*height)+5);
  printf ("<param name=orientation value=\"horizontal\">\n");
  printf ("<param name=scale value=\"%d\">\n", scale);
  printf ("<param name=columns value=\"%d\">\n", MAX_VALUES);
  printf ("<param name=title value=\" \">\n");
  if (col == 0){*strcpy(color, "yellow");}
  else if (col == 1){*strcpy(color, "orange");}
  else if (col == 2){*strcpy(color, "red");}
  else if (col == 3){*strcpy(color, "pink");}
  else if (col == 4){*strcpy(color, "magenta");}
  else if (col == 5){*strcpy(color, "blue");}
```

```
  else if (col == 6){*strcpy(color, "cyan");}
  else if (col == 7){*strcpy(color, "green");}
  else if (col == 8){*strcpy(color, "darkGray");}
  else if (col == 9){*strcpy(color, "gray");}
  else if (col == 10){*strcpy(color, "white");}
  else {*strcpy(color, "cyan");}
  for (i = 0; i < MAX_VALUES; i++)
  {       fscanf (file1, "%d %s %d %d %d.%d", &date[i], month[i], &hours[i],
                  &minutes[i], &value[i], &poop);
          printf ("<param name=c%d_color value=\"%s\">\n", i+1, color);
          printf ("<param name=c%d_label value=\"%02d %s %02d:%02d\">\n", i+1,
                  date[i], month[i], hours[i], minutes[i]);
          printf ("<param name=c%d_style value=\"striped\">\n", i+1);
          printf ("<param name=c%d value=\"%d\">\n", i+1, value[i]);
  }
  fclose (file1);
  printf ("</APPLET>\n<P>\n");
}
```

XXXXXXXX_settings

The settings file is as follows. While editing the settings file, keep in mind that the various objects must remain in place as they are shown here: ‘home page: something’ on first line, ‘background image: something’ on second line, ‘data dump: something’ on third line, ‘data points:’ on fifth line, ‘XXXXXX description description’ from the sixth line, and ‘end’ on the last line. ‘Something’ in all cases means a single name without spaces. ‘Description description’ may have spaces. ‘XXXXXX’ represents the six character data point name. If the background image does not exist, put a dummy name in that place.

```
home page: http://www-personal.umich.edu/~ashowe
background image: images/speckle.jpg
data dump: DATA_DUMP

data points:
D27004 Outside Air Temperature
D27005 Mixed Air Temperature
D27008 Supply Air Temperature
D27009 Return Air Temperature
D27012 Room Temperature
D27013 Surface Temperature
D27016 Glass Temperature
D27017 Room Humidity
D27036 Comfort Temperature
D27037 Display of Comfort
D27044 Predicted Mean Vote
end
```

Data Dump

The data dump is automatically downloaded into computer A. The file is a text file and can be full of all kinds of information, as long as ‘Name’ is followed by ‘:’, followed by any number of data point names. These names are ‘VALUE’ in the ‘sensor.html’ document, and are the first six characters in the data point name list in the settings file. In the data dump file, ‘fmonitor.cgi’ looks for ‘Name’ and ‘:’ and begins counting columns from there, until it finds the ‘VALUE’ that ‘sensor.html’ passed to it. From there, ‘fmonitor.cgi’ counts the columns after ‘Units’ and retrieves the units for the data point, passes over the ‘----------’, retrieves the date and time, and counts over the correct number of columns to retrieve the value. If the point name does not match up with ‘VALUE’ on the first list of data points, ‘fmonitor.cgi’ will continue to search lower in the document for the proper value. As stated earlier, as of this writing FMonitor is not robust enough to handle typos. If the point name is not found it will continue to loop forever until manually cancelled. Therefore, IT IS IMPERATIVE THAT ALL SENSOR DATA POINTS LISTED IN THE ‘sensor.html’ BE PRESENT IN THE DATA DUMP. The sample data dump file has been abridged for insertion in this report, but the actual file must have exactly 100 samples of each point.

1>Which trend (<CR> for 1st or exit)?

2>Next point name? D27037

Etcetera

Trend Multi-point Report interval: 5 minute(s) 14:47 11-Jul-1996

Name : D27004 D27005 D27008 D27036 D27037

Units : DEGF DEGF DEGF DEGC DEGC

11-Jul 06:30: -1.250 15.230 65.450 20.000 1.170

11-Jul 06:35: -1.150 16.980 67.890 20.000 2.430

11-Jul 06:40: -1.040 17.120 69.450 20.000 2.250

11-Jul 06:45: 0.060 17.340 70.340 20.000 2.930

11-Jul 06:50: 0.110 18.450 74.980 20.000 1.340

********** Continues for exactly 100 samples **********

This list can be followed by other text and any number of additional sets of data points similar to the first set (i.e. begin with ‘Name’ and ‘:’ on the first line, ‘Units’ and ‘:’ on the next line, ‘-------’ on the next line, and date / time / value columns after that.

XXXXXXXX.html

The following is the HTML coding for the 'XXXXXXXX.html' file. The only editing required by the user would be to insure that the VRML model's URL address is typed in correctly, and that the 'TITLE' is appropriate for the building. In this example, the VRML model is named 'device.wrl'

```
<HTML>

<HEAD>
<TITLE>BTUS facility</TITLE>
</HEAD>

<FRAMESET COLS="*, 256">

<FRAMESET ROWS="50%, 50%">
<FRAME SRC="http://www-personal.engin.umich.edu/~ashowe/vrml/device.wrl"
NAME="Main" SCROLLING="none">
<FRAME SRC="description.html" NAME="View" MARGINWIDTH="5" MARGINHEIGHT="5"
SCROLLING="auto">
</FRAMESET>

<FRAME SRC="sensor.html" NAME="List" MARGINWIDTH="5" MARGINHEIGHT="5"
SCROLLING="auto">

</FRAMESET>

</HTML>
```

description.html

The following is the HTML coding for the 'description.html' file. The contents of this file is entirely up to the user to edit and produce, but the name of the file should remain the same. The 'body background' should show the same image as that which is to be listed in the settings file. If a background image is not desired, the line can be deleted.

```
<HTML>
<body background="http://www-personal.umich.edu/~ashowe/images/speckle.jpg">
<BODY>

<H2>BTUS Facility Management</H2>

<HR>
This page blah blah blah (insert an explanation, description, paper, or desired text here)
<HR>

<P>
</BODY>
</HTML>
```

sensor.html

The following is the HTML coding for the ‘sensor.html’ file. As with the ‘description.html’ file, the ‘body background’ should reflect the same background image name listed in the settings file, if desired. This sample file shows only one data point. The user is responsible for adding other sensor data points and assuring that the exactly eight character ‘NAME’ name of the building and the exactly six character ‘VALUE’ name of the data point are typed in correctly for each point. In addition, the user must assure that ‘fmonitor.cgi’ URL address is correctly typed in.

```
<HTML>
<body background="http://www-personal.umich.edu/~ashowe/images/speckle.jpg">
<BODY>

<P>
The following is a list of sensors blah blah blah (insert an explanation or desired text here)
<P>

<HR><P>

<FORM METHOD="POST"
ACTION="http://spring.eecs.umich.edu/ashowe/fmonitor.cgi">

Preference for method of viewing:

<SELECT NAME="view_option">
<OPTION SELECTED> Table
<OPTION> Graph
</SELECT>
<P>

<B>
D<FONT SIZE=-1>EVICES </FONT>
/ S<FONT SIZE=-1>ENSORS</FONT>
</B><P>

<INPUT TYPE="radio" NAME="BTUS0001" VALUE="D27004" CHECKED><B>1.
Outside Air Temperature</B>: values for the air temperature immediately exterior to
window / wall assembly.
<P>

To view the values, press this button: <INPUT TYPE="submit" VALUE="View data">
<P>
</FORM>
</BODY>
</HTML>
```

NOTES & REFERENCES

[1] A. Scott Howe is an architect with Kajima Corporation of Tokyo, Japan and is currently in the Rackham Doctoral Program in Architecture at The University of Michigan.
Home Page URL:
http://www-personal.umich.edu/~ashowe
E-mail:
ash@ipc.kajima.co.jp

[2] Edward F. Smith, "Virtual Buildings: Knowledge Based CAD Models for Design, Analysis, Evaluation and Construction", *Computer Solutions*, Summer 1992, pp30-32.

[3] Jeffrey M. Hamer, *Facility Management Systems*, Van Nostrand Reinhold Company, New York, 1988, pp79-84.

[4] http://www-personal.umich.edu/~ashowe/netkit.html

[5] A. Scott Howe, "Internet-based Architectural Visualization", presented at the *ACSA European Conference*, Copenhagen, Denmark 27 May 1996.

[6] A. Scott Howe, "Designing for Automated Construction", May 1996 (unpublished paper).

[7] Mark Pesce, *VRML: Browsing & Building Cyberspace*, New Riders Publishing, Indianapolis, Indiana, 1995.

[8] Mike Snider, "Now you can tune your TV to Internet", *USA Today*, 19 September 1996, p4D.

[9] Brian C. Fenton, "Rabbit ears meet the mouse", *Popular Mechanics*, October 1996, pp86-89.

[10] H. Michael Newman, *Direct Digital Control of Building Systems: Theory and Practice*, John Wiley & Sons, Inc., New York, New York, 1994.

[11] V. Elaine Gilmore, "U.S., Japan, Europe: The World's Smartest Houses", *Popular Science*, September 1990, pp56-65, 102.

[12] http://tron.is.s.u-tokyo.ac.jp/TRON/overview.html, TRON project, Sakamura Laboratory, University of Tokyo.

[13] Ken Sakamura and Richard Sprague, "The TRON Project", *Byte*, Vol14 No4, April 1989, pp292-301.

[14] Denny Radford (Intellon Corp.), "Spread-spectrum data leap through ac power wiring", *IEEE Spectrum*, Vol33 No11, November 1996, pp48-53.

[15] http://www.hig.no/avdeling/ea/lonwork

[16] http://www.ashrae.org

[17] http://www.hometeam.com

[18] http://www.cebus.org

[19] http://lonworks.echelon.com

[20] http://www.savoysoft.com

[21] Robert W. Marks, *The Dymaxion World of Buckminster Fuller*, Reinhold Publishing Corporation, New York, New York, 1960, pp88-91.

[22] Charles Jencks, *Architecture 2000: Predictions and Methods*, Praeger Publishers, New York, New York, 1971, pp94-95.

[23] Michael Franklin Ross, AIA, *Beyond Metabolism: The New Japanese Architecture*, McGraw-Hill Book Company, New York, New York, 1978, pp36-38.

[24] Toshio Nakamura, "Richard Rogers 1978-1988", *A + U Architecture and Urbanism*, December 1988 Extra Edition, pp48-61.

[25] Colin Davies, *High Tech Architecture*, Thames and Hudson, London, Great Britain, 1988, pp42-55, pp68-85.

[26] A. Scott Howe, "A Genesis System" in: *Special Research Report of the Sensitivity Engineering Product Development Research Group: A Sensitively Designed City*, (Tokyo: Japanese Ministry of International Trade and Industry 1994) pp.73-92. Discusses the use of kit-of-parts building systems and automated construction. Report in Japanese.

[27] http://spring.eecs.umich.edu/ashowe/fmonitor.cgi

[28] http://spring.eecs.umich.edu/ashowe/BTUS0001_settings

[29] http://spring.eecs.umich.edu/ashowe/DATA_DUMP

[30] http://www-personal.umich.edu/~ashowe/BTUS0001/BTUS0001.html

[31] http://www-personal.engin.umich.edu/~ashowe/vrml/device.wrl

[32] http://www-personal.umich.edu/~ashowe/BTUS0001/description.html

[33] http://www-personal.umich.edu/~ashowe/BTUS0001/sensor.html

[34] Sami Shaio, "chart.java" code, Sun Microsystems, Inc. March 1995. 'chart.java' for this research project is located at: http://www-personal.umich.edu/~ashowe/java/chart.java

[35] http://www-personal.umich.edu/~ashowe/images/speckle.jpg