EMCS Interoperability: Today's Dilemma, Tomorrow's Reality

H. Michael Newman, Manager
Facilities Engineering Computer Section
Cornell University

"Interoperability", as the term is most often used in the context of energy management and control systems (EMCS), means the ability of EMCS computer equipment from different manufacturers to exchange information. The lack of a significant degree of interoperability is a dilemma today because it forces the users of EMCSs, who may wish to expand or consolidate their systems, to choose from among several unpalatable alternatives. They can deal with a single supplier who may yield to the temptation to charge higher prices than if he were forced to bid competitively. They can install headend computers and operator terminals for each vendor- a situation akin to being forced to buy one TV to watch ABC, another to watch CBS, and a third to watch NBC. (Never mind about PBS, ESPN, C-SPAN...) A third alternative is to avoid digital controls altogether, an option a substantial number of owners have unfortunately chosen.

Why is there a lack of interoperability? The main reason is that there is no agreed upon standard set of rules and procedures to govern intersystem communication. Such a set of rules and procedures is called a "protocol".

Note the use of the word "standard" rather than "open". There are major differences between the two concepts. An open protocol is simply one whose details are public. There are several such open protocols for EMCS equipment available today. They have been developed by vendors who, with an understandable interest in increasing their market share, have chosen to make their contents known.

A standard protocol, in contrast, is usually adopted or developed by a recognized standards-making organization. The process is generally one of achieving concensus among interested parties, all of whom have the opportunity to participate in the process to the degree they desire. The promulgating organization has the responsibility to resolve matters of interpretation of the standard and to review its contents on a regular basis. While compliance with concensus standards is voluntary on the part of manufacturers, adherence to such standards may ultimately be specified by system designers, rendering them, in effect, obligatory. They are thus a serious matter to all concerned. The standard being developed by ASHRAE's Standards Project Committee 135P is this type of voluntary, concensus standard.

The consequences of a standard protocol will be far reaching: EMCS designers will be able to select the most suitable equipment for each application; and vendors, freed from worrying about communications, will be able to concentrate on bringing to market control equipment and operator interfaces based on the latest innovations in hardware and software.

In spite of the historical lack of an EMCS communication standard, a few users have been able to achieve a limited measure of interoperability. This has been accomplished by constructing multilingual "gateways", computers which have been programmed to retrieve information from one EMCS, translate it into the format of a second system, and then transmit the reformatted data to the second system. Although its primary purpose is to provide operator interface and data storage functions, such a gateway capability is realized by our central EMCS computer at Cornell University. Because it understands the protocols of each of the different EMCS networks to which it is connected, it is able to pass data between them if an application program directs it to do so. A computer such as ours is generally referred to as a "multi-protocol host".

The Cornell EMCS

Cornell's first central monitoring and control system was installed in 1964, in conjunction with the commissioning of the university's first chilled water plant. The system was strictly electro-mechanical. In the mid-seventies our first computer arrived. It had no disk storage and was interfaced to the original control panel and to the university's power grid to provide peak demand limiting. Its program was loaded by paper tape!

Two years later, an upgraded system was installed with 5Mb of disk storage, 128K of main memory, and a communications controller that allowed us to connect six serial asynchronous devices, such as CRTs and printers, to the computer. The new equipment gave us, for the first time, access to field sensors and actuators using non-programmable digital multiplexors, albeit at only 1200 bps. As importantly, we were able to begin developing our own database and graphic display capabilities.

When, in the early eighties, the first programmable direct digital controllers (DDC) for HVAC systems appeared, we found ourselves at a fork in the road. We wanted to go with DDC but wanted to retain our interfaces with the old electro-mechanical system and the non-programmable multiplexors (and the databases and graphics we had developed). Our choices were to buy a second headend computer just to talk to the new DDC system or to obtain the DDC vendor's communication protocol and write our own

interface software. We chose the latter. When, in due course, the vendor agreed to provide the needed data, we realized that we had crossed the EMCS Rubicon: we found ourselves in the land of the multi-protocol host. To this day, our only hope of ever returning to the simple land we left behind remains the adoption of a standard protocol.

As time has passed, two more digital control systems have been added and our central computer has been upgraded to a pair of 32-bit supermicro DEC MicroVAX II's. We now have 50 communication ports, over 1000 MB of disk storage, and the capability to support 16 simultaneous users. Our present system is shown in Fig. 1.

What have we learned? Adding a new system in a multi-vendor environment involves much more than merely overcoming communication problems. In addition to developing communication software it has been necessary to provide configuration support, database support, and operator-machine interface support. Let's look at each of these areas briefly before returning to communications.

Configuration Support

Of our four systems, one is entirely passive and non-programmable, one is configured by writing programs in a vendor-specific language and two are programmed using computer-aided techniques which involve filling in entry screens or "templates". In the case of the vendor-specific language, we obtained a version of the source-language-to-binary translator written in IBM PC Pascal which we were able to convert to the Pascal used on the VAX (where the translator runs several orders of magnitude faster than on a PC)!

The two systems configured by filling in the blanks were each treated differently. In one case we received source code for the configuration programs, again written to run on a PC only this time in C. These had to be modified to understand the keyboard and screen mapping of our terminals. The second vendor provided us with a complete library of routines, happily already written in VAX FORTRAN, which were simple to install and which allow us to exercise all communication and configuration functions. It is interesting to note that this latter vendor is the supplier of the industrial grade process control system we use in our central heating and chilled water plants. The process control industry has always seemed to be a step ahead of the HVAC control industry and our experience with this vendor reaffirmed this perception. They have understood the desirability of computer-to-computer interfacing for years- and have made it easy!

Database Support

One of the numerous benefits of supporting multiple systems from a single headend is that a database can be set up that allows appication programs to access field points without having to know the details of how the point is connected, i.e., the details of each vendor's addressing scheme. This desire for "device independence" led to the definition of a database containing entries for each sensor, actuator, and calculated point which contains the vendor-specific hardware and software details. The key to this indexed file is the point name in the form of "FACILITY.SYSTEM.POINT". This convention has allowed the creation of commands like "SHOW" and "READ" which permit accessing related data with simple command language input. For example, the command "READ /ENTRY ...CW...FLOW" allows the EMCS operator to read the value of all points whose database entries match the given entry pattern, in this case chilled water flows in any facility or system. Because the application program databases refer to the field points by means of the database names rather than hardware addresses, changes in field connections or other point parameters require modification of only a single database entry.

Operator-Machine Interface Support

This type of support involves adding capabilities to the graphics and tabular report composer programs that understand how to do input and output to the new system. We have tried to minimize this problem by using the named database entry approach mentioned above wherever possible, thus requiring no special I/O routines to be added at this level. But each system has its own peculiarities which in some instances can be used to advantage. In the case of the controllers configured by means of the vendor-specific language, we added an extension to the translator that automatically generates a name-to-address cross reference file for each controller when the source file is translated. The names referred to here are the one to six character names used in the control program. In the case of the template-configured systems, there are no source programs and hence no symbolic names. In these systems all points are referred to numerically thus ruling out this rather elegant approach to keeping all databases consistent in the event of field hardware or software changes.

Communication Support

Finally we come to communication support. It should be fairly clear that two things are required in order to build a multi-protocol system: a detailed knowledge of each protocol specification and enough programming expertise to turn the specifications into code. While the types of support mentioned above are important to the mission of the

EMCS as a user tool, and thus represent a reasonable investment of programmer resources, implementing multiple protocols- while essential- is a comparatively unproductive endeavor.

The first thing that has to be done is to get the protocol specification from the vendor. Since most vendors have rarely had to provide this information, they tend to be reluctant to do so. Once obtained, the documentation can be incomplete and difficult to understand, having never been intended to be seen outside of the vendor's engineering department. Moreover, it is not enough just to know message formats. One has to know how each message is to be acted upon in each circumstance. This is the essence of a "protocol". Acquiring this knowledge involves not only a bruisingly steep learning curve, it also generally requires finding a source of information deep within the bowels of the vendor's engineering staff who actually knows what is going on. Such individuals usually don't talk to customers, though I have found them to be most cordial once the corporate bureaucracy has been pushed aside. A comparison of the communication aspects of our four systems is shown in Fig. 2.

As can be deduced from the figures, our systems share a number of common characteristics: each is accessed through a serial, asynchronous RS232 communication port; each requires some type of interfacing "gateway", although the functions of the gateway vary from simple modulation-demodulation to extensive protocol conversion; and each network is accessed, as far as the VAX is concerned, by means of a master/slave protocol (even though a peer-to-peer protocol may be in effect between the field panels).

The main differences between the systems, in terms of the functionality of the device driver subroutines we had to write, stem from the differences in "intelligence" of the gateways. In some cases, for example, the gateways are responsible for error detection and recovery, in other cases this is the responsibility of the headend. In some cases the gateways alone need to know the protocol of the field panels because they perform message structure conversion; in other cases the gateways simply pass without modification messages which have been fully structured at the headend.

Benefits of Interoperability

There are two principal benefits associated with being able to access all of our systems from a common point. The first is the ability to pass data from any field panel to any other field panel. We use this capability, for example, to collect chilled water system differential pressures which are then fed to the panel controlling our variable speed

pumping. The delta P transmitters, which are scattered all over campus, are simply connected to the nearest field panel, irregardless of manufacturer. In this case, the VAX basically acts like an intelligent star controller with each limb a different vendor network.

The second benefit is the ability to develop common, integrated operator interface capabilities. Our building and campus-wide graphics and tabular reports combine data from all systems based on the logical relationships between the data, not on the basis of which vendor's system a point happens to be attached to.

The net effect is that the EMCS operators are able to concentrate on managing building systems rather than worrying about differences in vendor field panels or communication networks. This is, after all, what EMCSs are supposed to be all about!

Conclusions

When I was asked to write a paper for the Second Open Protocols Symposium one of the sponsors suggested that it be called "Innovative Interconnection of Major Direct Digital Control Panels", a title which has appeared in the announcements of the symposium. While I must admit I was flattered that someone would view our efforts as "innovation", the fact is that what we have done has been done out of necessity, out of a simple desire to maintain some semblance of control over our own destiny. To me, a multi-protocol host is a manifestation of the fact that the EMCS industry has not yet fully matured. The focus needs to be on how to achieve genuine interoperability, as reflected by the revised title of this paper, not on how to cobble together today's widely disparate systems! Though gateways and multi-protocol hosts do enable us, even now, to integrate systems from multiple vendors to some limited degree, this is certainly not a sound, long-term approach.

The key to achieving true interoperability is the development of a powerful, comprehensive, and extensible protocol that allows both vendor and user to get on with the business of applying EMCS technology for the conservation of our energy resources and the betterment of our society. This goal is not only attainable but, based on the work of ASHRAE SPC 135P, attainable in the not too distant future!

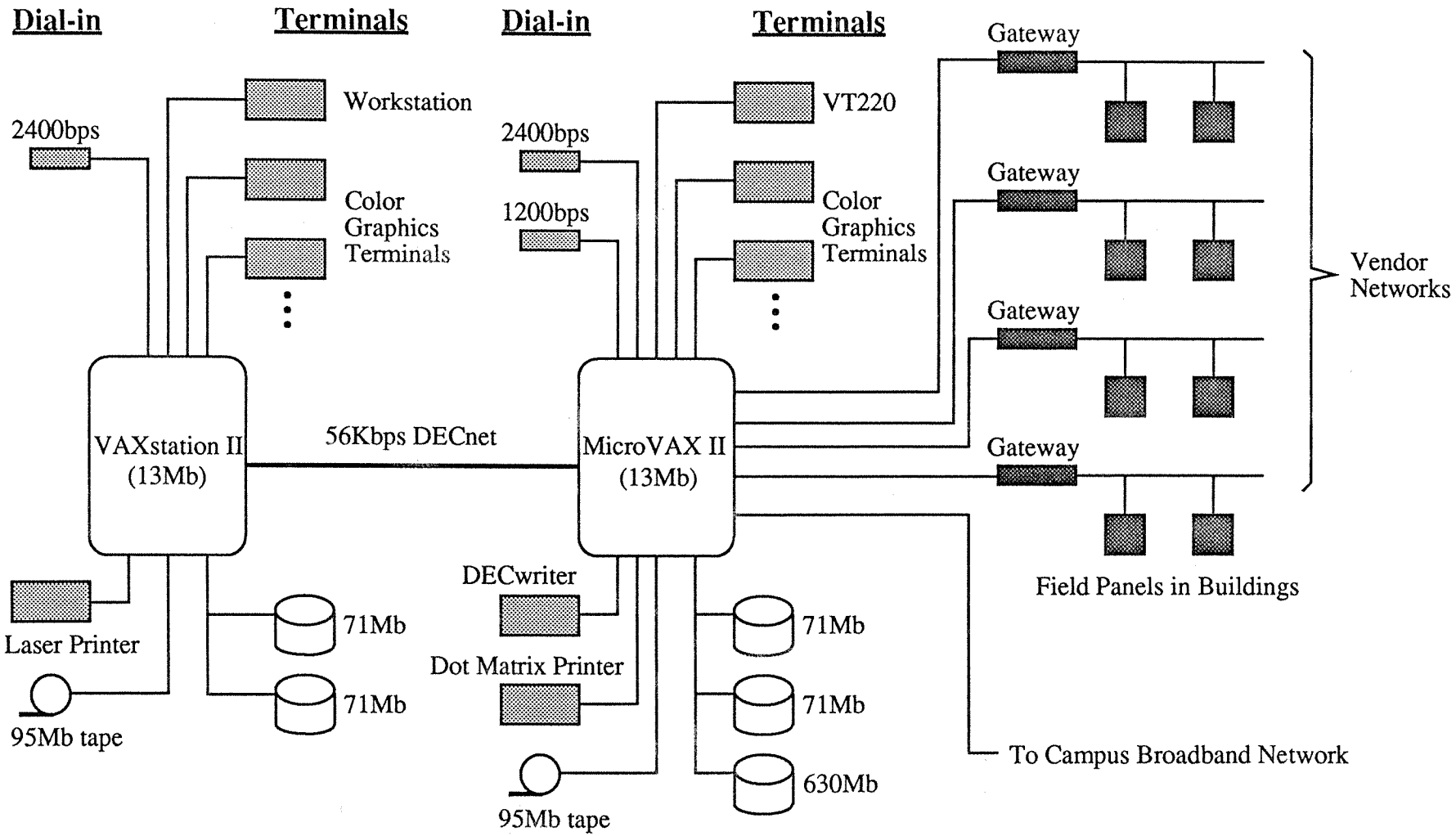

Fig. 1 - Overview of the Cornell University EMCS

Physical	Network Access	Gateway Functions	Driver Functions
RS232 at 9600bps to gateway then 1200bps on 4-wire Bell 203 multipoint circuit	Master/Slave with Request/Response	Converts 4 or 5 byte message to 25-bit asynchronous packet with 9 error checking bits	Passes parameters to gateway for generation of BI, BO, AI, and AO commands
RS232 at 9600bps to 2-wire modem operating on split frequencies in the 10-40 kHz range	Master/Slave with Poll/Poll-response and Request/Response	Signal conversion only; passes messages without regard to content	Does name-to-I/O port translation; performs handshaking to keep remotes "on-line"; performs checksum and 1-bit flip-flop sequencing
RS232 at 19200bps to gateway then 500kbps on redundant twin-axial cables (RG-22-BU) using Manchester encoding	Master/Slave with Request/Response to gateway then peer-to-peer tokenless ring	Performs protocol conversion and maintains a point table containing change-of-value data	Calls vendor-supplied subroutines; gateway supports 30 message types
RS232 at 9600bps to gateway then 9600bps 2-wire RS485 circuit	Master/Slave with Poll/Poll-response and Request/Response to gateway then peer-to-peer token ring	Handles encapsulation of messages to vendor bus; buffers bus messages for headend	Performs checksum calculations; builds gateway message frames containing previously encoded bus messages

Fig. 2 - Comparison of the interconnection of Cornell's four EMCS networks