Some Interactions of Information and Control in Integrated Automation Systems

Edward J. Barkmeyer
U.S. National Bureau of Standards
Bldg 220 Room A127
Gaithersburg, MD 20899

The ready availability of inexpensive and standard communications hardware in the late 1980s, coupled with sufficient standardization of intermediate-layer protocols to make reliable machine-to-machine communication through any combination of standard devices a reality, has laid the groundwork for integrated automation on a grand scale. The contribution of the Manufacturing Automation Protocols (MAP) effort [MAP87], thus far, has been to speed delivery of this powerful communications capability to the industrial community. We must realize, however, that the delivery of the capability has now outstripped our ability to use it. On the factory floor, we have no devices which are prepared to control automation on a grand scale, and even worse, we have few controllers which are capable of contributing their local automation to any kind of integrated whole. The stumbling block is the lack of common languages or even a common experience base in which to communicate about automation tasks. We are now in the midst of a flurry of intellectual activity, on both sides of the Atlantic, developing languages, architectures, control techniques, data systems, communications systems and ultimately controllers to meet this challenge. In such a burst of activity, it is to be expected that experts in individual areas will hasten to solve the problems with which they are most conversant, and that certain interdisciplinary problems will fail to get timely attention. It is the purpose of this paper to draw attention to several such issues now, in the hope that they may get due consideration as the process advances. The issues discussed here are:

- separation of data flow from control flow,
- the impact of perfect communication on real-time control,
- consistency of data with physical reality,
- the significance of standard data models,
- the identification of shared and private data.

What all of the issues have in common is that their solution requires common attention and joint consensus by the experts in communications, data systems and control.

The presentation of these issues, and the recommendations for approach to solutions, where they appear, draw on seven years of experience in integrated automation in the NBS Automated Manufacturing Research Facility (AMRF) [Nanz84], a laboratory for joint government/

NATO ASI Series, Vol. F53
Advanced Information Technologies for
Industrial Material Flow Systems
Edited by Shimon Y. Nof, Colin L. Moodie

industry/university research in flexible automation and in-process measurement supported by the Bureau of Standards and the Navy Manufacturing Technology Program.

Issue 1: Separation of Data Flow from Control Flow

Before 1984, industrial organizations who were linking control systems together were limited by the available communications technologies of the time, which were characteristically point-to-point. Thus any station which needed to communicate with more than one other station needed multiple physical links and interfaces to accomplish the communication. As a consequence, for reasons of efficiency and reliability, it became common practice, particularly for direct equipment controllers, to make a single communications interface provide the entire spectrum of external services. In particular, it became the norm to integrate the commands of a supervisory controller with the data needed to execute them, because the data had to come through the same physical channel anyway.

In the last five years, it has become inexpensive to use physical interfaces which permit a broad spectrum of logical connections through a single physical interface, e.g. IEEE 802 networks [IEEE83a,IEEE83b,IEEE84,IEEE85]. More importantly, compliments of the MAP-enforced acceptance of the Open Systems Interconnection model [ISO84], we now have software to support distinction between physical links and service "sessions" in such a way that there is no required relationship between them. Since the original motivation for integration of data and control functions has become obsolete, it is time to re-examine this integration from the pure design point-of-view.

The primary advantage of integrating control functions with the related data is that it minimizes the complexity of the subordinate process: the subordinate is not "burdened" with two logical interfaces, one for commands and the other for supporting information, like machine control programs. This allows the use of relatively simple systems for equipment control, but it has two corresponding disadvantages. First, it maximizes complexity of the supervisory system, because the supervisory system has to handle, package and communicate all of the data necessary for the functions of all of its subordinates, in addition to all of the data necessary for its own functions. Secondly, it enforces a single control architecture, in which commands and data must come together, and therefore presumably "down from the top" in a fixed hierarchy. Among other things, it makes necessary the continuing revision of control interfaces and supervisory control programs to support equipment controllers which are better designed for integration. The rigid simplicity of the interface prevents the equipment controller from operating on information units not envisioned by its supervisor and from making other choices of data repository, even local ones, not supported by the supervisory interface.

Choosing to separate data from control is an application of the design principle of *separation of concerns*, and realizes the primary benefit of that principle - flexibility, the ability to make independent optimal choices for the subsystems, in this case control services and data services. It is accomplished by reducing the data present in the commands to the keys to the major information units and the parameters of the control function itself, and by providing a separate data service function and protocol in the subordinate controllers [Albu81].

Now the choice of control architectures becomes arbitrary; hierarchical, cooperative, reactive and hybrid architectures are all possible, because the method of accessing data in the controller is independent of the method of determining the functions which use it. Similarly, the choice of data services may be made optimally. The control system can take advantage of whatever local data caches and databases it may have, and can obtain or modify data from whatever sites in the integrated facility its data service may access. The external control functions still direct the use of the data, and may provide direction for early acquisition or planned usage, but the supervisor (if there is one) does not have to handle the data itself, or in many cases, even be aware of all the data which is accessed. Where the communications are still point-to-point, the physical flow of the data will be the same, whether data and control are integrated or logically distinguished. But here the engineering cost of providing the separate data service path will be more than justified by the simplification and increased longevity of supervisory control programs. And when the communications are upgraded, into a broadband facility, for example, the equipment controllers can obtain workpiece descriptions and machine control programs directly from engineering or inventory systems, and the need for copying such data down the hierarchy vanishes, *without change to the control programs*. And even when the equipment controller is upgraded to use and provide more information in its operations, the supervisory controller may still survive unchanged. The flexibility achieved by separating data from control is a mandatory step to future experimentation in integrated systems; and the resulting opportunities for improved control schemes and elimination of "middlemen" in the movement of engineering data from producer to consumer, can result in significant improvements in facility performance.

Issue 2: The impact of perfect communication on real-time control

One of the fundamental concepts in computer-based automation is that of interprocess communication, i.e. communication between separate programs engaged in different aspects of the same larger function. Forgetting all paradigms in human communication, we intuitively expect communication among computer processes to be *perfect*: we demand that every utterance produced by a sending process be heard by the designated receiver. In the automation of systems divorced from dynamic physical environments, especially those who have infrequent, weighty communications, this requirement is appropriate and can be met without unacceptable overhead. But in the automation of systems tightly coupled to rapidly changing physical environments, the consequences of this requirement are dramatic. In such an environment, a control process cannot always afford to wait for the intended recipient to get and acknowledge the messages. But, using the perfect communication model, the control process has no choice: it must at least wait for acknowledgement, and occasionally wait for recoveries of failed transmissions; otherwise information may be lost. The conventional solution to the sender wait problem, borrowed from environments unburdened by physical coupling, is *message queuing*. Message queuing, supported by some class of multiprogramming for communications, appears to allow the sender process to drop a message in a queue and go about its business, with the assurance that the message will ultimately arrive at the intended destination. Unfortunately, message queues were invented solely to insulate the sender process from the vagaries in communications timing, always assuming that the recipient process can, on average, read its mail faster than the sender can generate it. When this assumption is untrue, or untrue for a long enough period of time, messages will gradually back up, first in the receiver's inbound queue and then when it is full, in the sender's outbound queue, until at some inopportune time, the sender control process will come by with its regular message, and find that there is no place to drop it. Then, queuing notwithstanding, it will be faced with the inevitable dilemma: *wait* until there is room in the queue, or *throw the message away*. Formulated concisely, *perfect communication requires the sender to be prepared to wait*. If the sender *refuses* to wait, then some of its communications must be *imperfect*: some messages must be thrown away.

In designing control systems which are tightly coupled to real-world equipment, and which also have to communicate with others, this rule must be taken into account. Controllers must be designed either to tolerate unpredictable waits at the message enqueuing point, or to deliver all relevant information at the next messaging opportunity. Waiting is not normally a problem in issuing commands or supplying needed data to a subordinate process, but it can be a serious problem in reporting status *from* a real-time control or sensory process. If the status reports are designed to be incremental, i.e. separate messages for each interesting event, then the controller cannot deliver all relevant information at one opportunity, and it must be prepared to experience processing delays caused by communication waits. Optimistic engineers may believe that the solution is to connect such a controller only to supervisory systems which will always be able to consume the status reports of all subordinate controllers with time to spare, or conversely, to greatly limit the messaging volume permitted of subordinate controllers. Such a solution is intrinsically inflexible and closed-ended, and a controller so designed is obsolescent the day the prototype is built.

The alternative approach is the one we have always used in interfacing such systems to human operators: systems are designed to produce a *total* status report on each "cycle". In the human case, the presentation method is an array of gauges, displays, lights and alarms, but regardless of how busy he might otherwise be, when the operator examines the status, he gets the complete picture. This presentation method is chosen, because the communication is known to be *imperfect* - the operator can miss any number of changes in the state of the panel, but he can't miss anything important, because when he looks at the panel he gets the up-to-the-minute information on all aspects. This is the paradigm which should be used for designing status reports from real-time controllers. Each status report message should contain all of the information describing the subordinate controller which may be useful to the supervisor. The subordinate does not have to distinguish and report changes per se; the supervisor detects changes, in those information units which it considers important, between the status reports it actually sees [Barb82].

This approach has three side-affects. First, just as the human operator of a complex machine must be trained to recognize events from, and find relevant information in, the status display, supervisory control processes must be programmed to recognize events needing attention and find interesting information in the status report. This is not easy, but the second effect makes it worth the investment: there is now no need for any relationship between the rate of production of status reports by the subordinate controller and the rate of consumption of those reports by the supervisory controller, other than that required for responsiveness to changes in the physical environment, i.e. that required for the task at hand. Each controller processes at whatever rate it can, and the supervisory process can get as much data as it can absorb. The interface is open-ended and simultaneously upward and downward compatible. The third effect is that there is no need for deep queuing of messages from the subordinate; in fact, deep queuing only serves to slow down the communication of current information. "Double-buffering" is ideal: for the sender, one buffer is the report being transmitted and the other is the next report to go; for the receiver, one buffer is the last report received and the other is the next arriving. The control process rewrites the "next to go" buffer on each controller cycle or event, and whenever a communications opportunity arises, the communications process copies the "next to go" into the transmission buffer. Any further queuing simply introduces out-of-date status reports into the path, delaying the arrival of current information. The definition of "communications opportunity" is a protocol issue: it can be a request from the supervisory process, or the combination of local change, acknowledgement of the previous status report and available time on the communications channel. The former protocol is incorporated in the current draft of the Manufacturing Messaging Service [EIA87], while the latter protocol has been in use in the AMRF since 1983

[Mitc84]. The MMS method is simpler, but it is slightly less efficient and it makes exception-driven control more difficult. Either protocol permits the coupling of supervisory and subordinate systems with arbitrary reporting and consumption rates, and with arbitrary communications equipment.

Issue 3: Consistency of data with physical reality

In business automation systems, a process's only source of knowledge about the outside world is in the databases available to it. When there is more than one such database, or when information in two elements of a single database is interrelated, there is the potential for problems in consistency. The universal rule is: *When a process can see two views of the world, it is vital that the overlap between those views be consistent.*

Because the only views available to a process in a business automation environment are of the available databases, this rule has been interpreted to require management and/or software services which maintain consistency between multiple databases. This interpretation is still necessary, but no longer sufficient, in the case of materials flow systems. In automated materials flow systems, much of the active data mirrors physical reality in a much more intimate way than in business systems automation. In such systems, it is entirely possible, and often the case, that a process has both access to the databases describing the physical realities of the workplace and sensory access to other measures of the same physical reality as well. Take, for example, the workstation which can detect the arrival of an automatic guided vehicle (AGV), or the robot whose clearance depends on the current location of a transport element or the position of a window. In such cases, whenever databases are used to determine or support the physical circumstances, we must provide software services which maintain consistency between the databases and the real world. That is, *the database must be kept consistent with the physical reality*.

The alternative is to deprive some controllers of one of the two views, thereby avoiding the potential inconsistency. This is sometimes the best solution, applying a reliable engineering rule-of-thumb: "Always use the best data available." Unfortunately, it is not always obvious what the "best data" is. The usual interpretation of this rule for the real-time controller is: "Don't use the database for information you detect and measure directly." And in our experience, this interpretation is frequently unsatisfactory. Consider the case in which a workstation controller detects the arrival of an AGV and the unload of the tray/palette it carries. The only part of the physical reality that the controller can detect locally is the unload. The true nature of the load, that which is on the tray/palette, its configuration, contents, identification, is contained in carefully controlled databases which are guaranteed to be consistent by a global data system implementing consistency controls. If the controller uses the tray arrival as an indicator, it has no way to associate that arrival reliably with the contents data. In other words, the best data available, the local detection of the tray arrival, is inadequate when the controller needs to know more than that it arrived! On the other hand, the database update is not something that the controller can readily detect; so in this example, we are left with the consistency problem.

There are three common methods of maintaining consistency among databases, so we will consider them as alternatives for maintaining consistency with physical reality. They are: time-stamping, two-phase commitment and scheduled release.

One of the most flexible techniques for accommodating the cooperation of separate processes acting on the same data units is the time-stamp. The idea is that the control process simultaneously issues the database update and the command for the physical change; and the database update is time-stamped. Then any retrieval of the data prior to the change will get the data preceding the change and any later retrieval will get the data matching the real-world after the change. Using the AGV unload example, if the workstation detects the arrival and starts the contents retrieval operation after the transport system starts the update operation, even if the update has not yet completed, the data system must recognize the chronological dependence and stall the retrieval until it produces the correct (i.e. consistent) answer. But, while the time-stamp is a reliable method of enforcing proper subtask serialization *within* a data system or control system, it is a very unreliable mechanism for serialization of operations *at the interface*. If the data system assigns the time-stamps to the operations when they arrive at the data system interface, so that the time-stamps are based on a common clock, it is almost impossible to avoid "race" conditions between the two client processes. The network services, the operating system services, and even the sampling algorithm used by the data service itself, are all capable of making two messages with clearly staggered start times by-the-clock appear to arrive out of order at some point in the message management process. If, on the other hand, each controller assigns the time-stamp to its own request, then global clock synchronization is necessary to guarantee that the time-stamps are meaningful. But global clock synchronization is subject to tolerances, as a result of the medium of distribution and the variation in response time in the recipient systems, and as soon as more than a few systems are involved, these tolerances may exceed those of the operations we are trying to synchronize, The bottom line is that transaction synchronization by time-stamping is difficult, at best, and some other means of consistency control must be found.

The two-phase commitment technique appears at first glance much simpler and more reliable.

Using this technique, a control process simultaneously requests the data system to *commit* to the update and the subordinate or device to *commit* to the physical change. The *commitment request* essentially says: "Here's what I want you to do; don't do it yet, but tell me if you can and when you are ready." Each recipient responds "yes" or "no" and the data service locks out any further transactions until it gets the second "phase" of the request. When both recipients respond "yes", the controlling process then sends the second phase *execute* to both to effect the changes. If either responds "no", the controlling process then sends a second phase *abort* to both, and deals with the error recovery. In practice, many data systems cannot determine a priori whether to commit to an update or not, so they save the affected records, attempt to execute the update, report the results and lock out further transactions until they get the second phase of the request. If the second phase is *execute*, they then simply release the locks. But if the second phase is *abort*, they must undo or *rollback* the update they have done, before releasing the locks. The application of this method to consistency with physical reality has similar characteristics. In our example, the transport controller could conceivably direct a commitment request to the global data manager to update the tray/palette location at the same time it directs a commitment request to the AGV to unload. If they both respond "yes", the transport controller issues the *execute* to both and the workstation controller only sees the arrival after the database is updated. If the AGV can truly commit to the unload before attempting it, this method will successfully prevent consistency problems in the workstation views. But if the AGV has to *try* the unload in order to commit, the purpose of the commitment protocol is defeated outright, because the workstation, with its arrival detector, may see the physical change and initiate the contents retrieval while there is as yet no confirmation that the data system has seen the update transaction. So we must revise the approach as follows: The control process first requests the data system to commit to the update. When the data system responds "yes", then the control process commands the physical change. And when the physical change completes, the control process issues the second phase *execute* to the data system. This is a workable technique, which avoids the previous difficulty. But it does require the global data service to support two-phase commit protocol for the *user process*, not just internally, and, since commitments are subject to timeout, it requires the commitment holding time to be long enough to complete the physical change. These are somewhat strong requirements, which may affect overall data service performance adversely.

This leaves us with the scheduled release mechanism. In conventional systems, this is a management technique, rather than a software technique, in which database B is not permitted to be used until it has been reconciled with database A. Fortunately, this management technique translates directly to the operation-gating control technique: some coordinating process must officially "release" the information describing the physical reality, so that the control process which detects the change in the real-world must also detect the information release before it can start the dependent operation. That is, some control process has to be responsible for making the data and the physical reality consistent *at some point in time* and that process must post the command or indication that marks that point in time. In our example, the transport controller, or some higher-level coordinator, must post an indicator that the arrival and unload of the AGV has been made consistent with the global databases, so that the workstation may retrieve whatever contents information is appropriate. And the workstation controller must follow the rules, so that its use of the tray arrival indicator, and the corresponding access to data, occurs *after* the consistency has been established. The actual algorithm assumes the existence of some agreed-upon consistency indicator for this particular physical situation. The coordinating control process first clears the consistency indicator, then orders both the database update and the physical change, and finally, when both are complete, sets the consistency indicator. The affected process gates the dependent operations with the logical *and* of the change-detect and the consistency indicator. This mechanism follows well-known control techniques, and works without extraordinary changes to existing systems. It is recommended as a method of resolving potential conflicts between the databases and the physical reality in the design of controllers in materials flow systems.

Issue 4: Significance of standard data models

In its work on standard database definition languages [ISO82], the ISO working group formulated what is now called the Helsinki Principle: "Any meaningful exchange of utterances depends upon the prior existence of an accepted set of semantic and syntactic rules. The recipients of the utterances must use *only* these rules to interpret the received utterances, if the received meaning is to be the same as that which was meant by the utterer."

The first sentence of this principle is intuitively obvious, but only recently has the computer applications community begun to understand the ramifications of the second sentence. Until very recently, automation systems tended to operate in closed communities - a particular organization or a single vendor's equipment - so that the "accepted set of ... rules" contained many unwritten elements implicitly used by members of that community. What we have found as we try to design for integration across vendors and organizational elements is that these implicit rules vary substantially from community to community even for the same application. The Initial Graphics Exchange Specification (IGES) is a case in point. The intention of IGES 1.0 [IGES80] was to facilitate the transport of designs from one computer-aided-design (CAD) system to another. But the carefully crafted data exchange rules, in the light of the Helsinki principle, permitted exactly the exchange of the drawing: the only interpretation of the information provided by the rules of IGES was what the picture should look like, not, except superficially, the geometry or features of the object(s) being depicted. Several IGES implementations attempted to extend the drawing information to deliver geometry, but their experiences proved the pessimistic assessment to be correct. More recent versions of the standard [IGES88], therefore, have been extended to express geometric information specifically.

Historically, interchange standards and shared databases have been developed by getting the practitioners into a room to hammer out the list of information units which they all had in common and then devise a data organization or presentation scheme for those units. This technique is fraught with exactly the problems which motivated the formulation of the Principle. The fact that we agree on the syntax of the data, its name, position and formation rules, does not necessarily mean that we agree on the meaning of the data, exactly what it describes and how it is intended to be used. Without a more complete specification of the semantics, the shared data is often inadequate to the purpose, as in the IGES case, and the information units which are shared are often misinterpreted or misused. In order to get real meaning out of the data, we must also have formulated, and agreed on, a model of the world the data describes. We now understand that this actually involves two different kinds of model [Brod84]: the static associations between the data and the real-world physical and conceptual objects it describes, called the *information model*, and the rules for the use and modification of the data, which are derived from the dynamic characteristics of the objects themselves, called the *functional model*. The significance of these models to data interchange for manufacturing and materials flow was recognized early in the Air Force Integrated Computer Aided Manufacturing (ICAM) Project and gave rise to the IDEF formal modeling project [ICAM84]. IDEF produced a specification for a formal functional modeling approach (IDEF0) and an information modeling language (IDEF1) [ICAM81]. The more recent Product Data Exchange Specification (PDES) project in the U.S. [PDES88], the related ISO Standard for the exchange of product model data (STEP) [Kall88] and the Computer Integrated Manufacture Open Systems Archictecture (CIMOSA) [ISO87] project in the European Economic Community have wholeheartedly accepted the notion that useful data sharing is not possible without formal semantic models of the context the data describes. Within their respective spectra of efforts, each of these projects has a panoply of information models for manufactured objects, materials and product characteristics, and for manufacturing and assembly processes. Each also has a commitment to detailed functional models of the various phases of product life cycle. The object of all of these recent efforts is to standardize the interchange of information in many aspects of product design, manufacture, delivery and support. And following the Helsinki Principle, the standardization of information and functional models in these areas is seen as a necessary means to that end.

But, while all of these efforts have concentrated on the problem of data exchange, the specification of functional and information models for the related processes has significant impact on the design of control systems as well. Consider that a functional model must describe all the significant changes that the objects described by the data can undergo, within some process context. If a standard functional model for some process context is adopted, then clear boundaries have been defined for the functions of a controller operating within that context, at least with respect to the perceived effects of that controller on the outside world. The controller can do some or all of those functions, but it cannot do other functions without participating in other contexts, and it is not permitted to have nonstandard side-effects on the modelled context(s) in which it participates. It is likely that the standardization of a functional model for a fairly sweeping context, like transport, will in fact become a hard boundary on controllers operating in that context, that is, such a controller must not participate in any other context. What will inevitably arise out of standardization of functional models, therefore, is a kind of standard modularization of various materials flow processes, into which nearly "plug-compatible" controllers can be fitted. The externally distinguishing features of controllers will be reduced to the degree of "completeness" in their implementations of the functional model, although there may be numerous internally distinguishing characteristics - actual technique, speed, power consumption, footprint, hardiness, etc. - which are incidental to the external perception of the process performed.

Lest we take these last implications as pie-in-the-sky, let me paraphrase the second sentence of the Principle: "Unless we can agree on what can happen to the objects described, we cannot meaningfully exchange the data describing them!" Integrated automation requires controllers to exchange data on a rather large scale. The Principle tells us, and experience has verified, that non-trivial exchange of data requires prior common functional models. It follows that successful steps toward integrated automation will inevitably lead to standardization of controller functionality in many aspects of materials flow. And the current strong desire for integrated systems suggests that such standardization will occur rather sooner than later. It is important to assure that control architectures are studied now, and that control system designers participate in the standardization efforts, with a view to imposing sufficient flexibility in the functional models to support the desirable control architectures. Otherwise we may find that our haste to standardize interchanges results in the proscription of potentially valuable system architectures.

Issue 5: Shared Information and Private Information

In attempting to modify existing "islands of automation" into integrated systems, the first realization that some of the data from SystemA must be used and perhaps modified by SystemB begets *information resource automation*, in which a common model, a common vocabulary and a shared data dictionary are derived, and each system is modified to import and export the relevant elements of the common databases for its own use. This "refitting-for-integration" produces a loose coupling between subsystems rather than a robust integration, but it is incrementally practicable. It also produces a distinction between "private" information, that data which is of use to only one subsystem, and "public" information, that data which shared among different subsystems. Experience in the AMRF indicates that such divisions tend to be arbitrary. The public information units are identified by committee, based on some consensus perception of current information needs and those of the near future, and the result is not a complete model of the common context, but rather a kind of greatest-common-factor of the existing individual models.

The recommended design-for-integration rule for data sharing is that all data handled by any subsystem is public, except that which is meaningless outside of the subsystem and that which is treated *by the subsystem itself* as duplicate, uninterpreted or unreliable. This rule works comfortably with the principle of modeling the objects before you define the interchanges, because anything which is externally comprehensible about the objects and processes with which the subsystem deals must be part of the information model, and therefore part of the public information base. Properly applied, the rule allows the manufacturer to protect certain private information units which are intimately involved with the method by which a controller carries out a generic function, while requiring it to make available information units which may be important to outside systems working on coordination, planning and control.

Not necessarily are these information units completely distinct. For example, it may be important to report the position or envelope of moving elements of the controlled apparatus so as to avoid collisions with other moving objects in the area. However, the public position data must be phrased in some standardized workspace coordinates and dimensions, which are globally meaningful, rather than reporting local joint or turntable positions which are meaningless outside the controller. Similarly, it is important to report such things as coolant levels and available power in terms of the ability of the subsystem to execute its nominal processes, rather than in absolute measures, which mean nothing to any system which doesn't know the consumption profiles of the device. What is hidden in this recommendation, therefore, is the further requirement that controllers must, perhaps continuously, *render certain private information units into meaningful public information.*

Part of the problem with the consensus agreements is that they are often colored by a second distinction: Access to public information must use *global data systems*, while access to private information can use the local data system, with which the authors of the local control programs are comfortable. The more public information a controller uses, the more involved it becomes with, and more importantly the more *dependent* it becomes on, the global data systems, which are *out of its control* (and out of the control of the engineers who built it). Fundamentally, no one wants his system to be dependent on elements out of his control, so left to his own devices, he designs and negotiates for the absolute minimum retrieval and insertion of public data by his controller. There is, in effect, a built-in characteristic of human engineers to design *against* integration! The imposition of standards by a higher authority is a path to solution, but standards reflecting a strong design-for-integration approach will only result from a standards development effort dominated by users and system integrators rather than a consensus of implementors.

Moreover, the adherence even to limited public data utilization standards is often implemented by a technique known as the "shadow database". The controller maintains a section of the private local databases which is logically identical to the segments of the global databases it is required to share. The internal workings of the controller always use only the local databases, and at certain key points in time, the public data is copied from the global databases into the local (shadow) databases, and at other key points, the updated data is copied in the other direction. From the engineer’s point of view, this encapsulates the global data dependence in the copy-in/copy-out functions, so that the controller can proudly run with "local data" whenever "necessary". This form of "design for dis-integration" has one serious drawback: it moves the responsibility for maintenance of database consistency out of the global data services and into the distributed control processes. Unsurprisingly, the various control processes are rarely up to the task. In our experience, the single most common cause of integration test failures has been inconsistency of the local databases. And this is simply because too many decisions in the management of the shared data have been left to the individual control systems. The recommended design principle here is that *consistency of shared data is a vital part of integrated control* - the rules for maintenance of public data consistency, whether global systems are used directly or shadow databases are constructed, must be embodied in the control standards and protocols.

Summary:

This paper identifies five issues for interdisciplinary attention in the automation of materials flow. My contentions are:

- That one must logically separate data flow paths from control paths, so that data systems architectures and control systems architectures can be chosen independently for optimal performance in a given facility.

- That the conventional perfect communication model may cause delays in real-time control systems and that either the control systems must be prepared for the delays or an imperfect communication model, necessitating total status reporting, must be adopted.

- That consistency between databases and the physical world can be an acute problem in real-time control environments, and that consistency indicators and gating, among other choices, may represent the most practicable solution to such problems.

- That data sharing means standard information and process models, and standard models will mean standards for controller functionality.

- That the identification of shared versus private data cannot be left to the consensus of the controller builders, nor can the management of shared data within the controllers, but both must be part of a concerted design-for-integration approach.

Although this paper provides approaches to some of these problems, it does not mean to solve them so much as to draw attention to the need for interdisciplinary attention to problems such as these in the automation of materials flow.

References:

[Albu81] Albus, J.S., Barbera, A.J., Nagel, R.N., "Theory and Practice of Hierarchical Control", Proceedings of the 23rd International Conference of the IEEE Computer Society, September, 1981.

[Barb82] Barbera, A.J., Fitzgerald, M.L., Albus, J.S., "Concepts for a Real-Time Sensory-Interactive Control System Architecture", Proceedings of the 14th Southeastern Symposium on System Theory, April, 1982.

[Brod84] Brodie, M., Mylopoulos, J., Schmidt, J.W., editors, "On Conceptual Modeling", Springer-Verlag, New York, 1984.

[EIA87] Electronic Industries Association, Standardization Project 1393A: "Manufacturing Messaging Standard Service Specification and Protocol", draft 7, August 1987, unpublished.

[ICAM81] "ICAM Architecture Part 2, Volume 5: Information Modeling Manual (IDEF1)", AFWAL TR-81-4023, Air Force Materials Laboratory, Wright Aeronautical Laboratories, USAF Systems Command, Wright-Patterson Air Force Base, OH, June, 1981.

[ICAM84] "ICAM Conceptual Design for Computer Integrated Manufacturing Framework Document", Air Force Materials Laboratory, Wright Aeronautical Laboratories, USAF Systems Command, Wright-Patterson Air Force Base, OH, 1984.

[IEEE83a] IEEE Standards Project 802.2: "Local Area Networks - Logical Link Control", Institute of Electrical and Electronics Engineers, New York, NY, September, 1983.

[IEEE83b] IEEE Standards Project 802.3: "Local Area Networks - Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications", Institute of Electrical and Electronics Engineers, New York, NY, September, 1983.

[IEEE84] IEEE Standards Project 802.4: "Local Area Networks - Token-Passing Bus Access

Method and Physical Layer Specification", Institute of Electrical and Electronics Engineers, New York, NY, September, 1984.

[IEEE85] IEEE Standards Project 802.5: "Local Area Networks - Token-Passing Ring Access Method and Physical Layer Specification", Institute of Electrical and Electronics Engineers, New York, NY, September, 1985.

[IGES80] Nagel, R., Braithwaite, W., Kennicott, P., "International Graphics Exchange Specification (IGES) - Version 1.0", NBSIR 80-1978R, National Bureau of Standards, Washington, DC, January, 1980.

[IGES88] Smith, B., Rinaudot, G., Wright, T., "International Graphics Exchange Specification (IGES) - Version 4.0", National Bureau of Standards, Gaithersburg, MD, in publication.

[ISO82] "Concepts and Terminology for the Conceptual Schema and the Information Base", ISO-TR-6007, International Organization for Standardization, Geneva, 1982.

[ISO84] International Standard ISO 7498, "Information Processing Systems - Open Systems Interconnection - Basic Reference Model", International Organization for Standardization, Geneva, 1984.

[ISO87] "Reference Model for a Computer-Integrated-Manufacture Open System Architecture", ISO/TC184/SC4/WG1/N95, March, 1987, unpublished.

[Kall88] Kallel, M., "Standards for Data Exchange in an Integrated Environment: A Methodological Approach", Master's Thesis, Sloan School of Management, Massachusetts Institute of Technology, January, 1988.

[MAP87] "Manufacturing Automation Protocol Specification, Version 3.0", Society of Manufacturing Engineers, Detroit, MI, July, 1987.

[Mitc84] Mitchell, M. and Barkmeyer, E., "Data Distribution in the NBS AMRF", Proceedings of the IPAD II Conference, Denver, CO, April, 1984.

[Nanz84] Nanzetta, P., "Update: NBS Research Facility Addresses Problems In Set-ups for Small Batch Manufacturing," Industrial Engineering, pp 68-73, June 1984.

[PDES88] Proceedings of the SME Conference on Product/Process Definition Data, R. A. Carringer, Ed., Dallas, 1988.