An Architecture for Concurrency Control

J.A. Barnett, D.E. Cass, and W.E. Betts

Northrop Research and Technology Center
One Research Park
Palos Verdes Peninsula, CA 90274

Extensive automation is necessary to reap the full promise of concurrent engineering because (1) many of the participating disciplines already use sophisticated automation to accomplish their functions and (2) the need for input from these disciplines continues throughout the product's life cycle even if just sporadically. The first point indicates that successful automation must address enterprise integration, while the second indicates that systems must have coherent knowledge of where functional responsibility and authority are to be found over time.

In other words, concurrent engineering must provide a ubiquitous organizational structure for the life cycle of products, not just for the first phases of the design process. Below, an architecture that accepts this challenge is discussed. The ideas emanate from our use of Justin [1], a tool to build functioning prototypes of enterprise automation.

Justin provides (1) a limited ability to simultaneously distribute design objects to multiple enterprise activities, (2) a data management facility that reduces order dependency among updates, and (3) a high-level protocol that supports object distribution, data update collection, and communication of control information. Unfortunately, these primitive mechanisms are necessary but not sufficient to provide complete automation support for concurrent engineering.

Our experience suggests an approach that we are now revising Justin to explore. The envisioned system is one where objects are maintained and controlled by a logically centralized process. Objects can be such things as the representations of products, parts and components, engineering change orders, liaison requests, etc. The controller stores representations of objects, distributes copies to proper enterprise activities, and accumulates and associates information, generated by the activities, with the objects.

Proc. AAAI-90 Workshop *Concurrent Engineering Design* (Boston, 8/1/90).

The controller must perform these tasks in concert with enterprise, customer, and government rules and procedures. National agenda, such as the concurrent engineering initiative, are changing and will continue to change these rules and procedures which already differ by object type, customer, and product class. Therefore, the controller must be able to absorb control regimes that are properly specialized.

A state-machine formalism is a good way to express these control constraints; A state is an important epic in the product's life cycle. The representation of a state should dictate which functional activities have access to object copies and what to do as a result of object modifications by these activities. Object modifications can effect specialized properties such as a CAD model associated with a part representation, however, most modifications are additions of textual critiques such as requirements statements, summaries of results computed by analysis activities, and approval signatures.[1]

State specifications provide enablement of concurrent simultaneous object processing but do not provide sufficient mechanism to control it. Consider a simple problem: simultaneously, seek the n necessary approval signatures for a document change request (DCR). If, before signing, one of the n adds a question critique to the DCR, communicate that question to the subset of the n who have not yet returned their copy of the DCR to the centralized controller. A straightforward state-machine implementation needs something in the order of 2^n states to accomplish this trivial-sounding task because there would need to be a state to represent each set of terminated activities and there are 2^n possible sets. Many other cases exist where the number of control regime states can grow exponentially.

There are simple ways to augment state machine formalisms to defeat growth problems. The easiest is to use standard programming language constructs. However, this move loses two major benefits expected from a state machine formalism: (1) application specialists, e.g., system engineers, configuration controllers, and project managers, can specify, modify, and understand control regimes and (2) the system architecture is flexible enough to provide experimentation with alternative policies.

Before proposing a solution to this problem, another must be considered

[1]Justin provides languages to specify critique regimes as well as control regimes. Both are necessary to properly express policy about who has authority to do what while the latter, in addition, is directly germane to concurrent engineering issues. The object controller interprets control regimes to determine object disposition and state changes. Interpreter state is stored with the object's representation.

in more detail. The second problem is how to determine who has authorities and responsibilities for various functions during the whole life cycle. It should be noted that in today's manufacturing environment, virtually *all* policy and procedure is expressed in terms of authorities.[2] Therefore, this problem must be addressed by any system that proposes to integrate enterprise automation whether or not it permits concurrent simultaneous object processing—concurrency just makes things worse.

An example will display the problem. Engineer E prepares a DCR and one of the necessary approval signatures is from a safety activity. The appropriate safety group is determined by the component class (electronic, hydraulic, etc.), customer class (government or commercial), E's organization (the enterprise is a matrix organization so the local political structure is involved), etc. Once the group is identified, S, its supervisor, is found to have the proper signature authority. However, S is away at a conference and she has delegated that authority to s. Unfortunately, s has a question which he adds to the DCR. The question is communicated to E, via the automatic system and he appends an answer. Next, his answer must be routed to the safety group authority, but, in the meantime, S has returned to work and reclaimed her authority from s. Who should get E's response: s or S?

Problems like the above are sure to occur frequently because the typical employee in the USA is away from his work place more than 10% of the time—total of vacation, holidays, illness, personnel business, and off-site business appointments. Concurrent engineering will make it a central issue because the amount of communications will increase as will the reliance on rapid input from other disciplines. Further, products such as complex weapons systems have life cycle times that exceed two decades. People come and go and enterprises reorganize as these years pass but design changes and improvements happen throughout the entire period. For all of these reasons, it is clear that authority and activity bindings do not remain constant.

State explosion and authority determination problems make implementation of automated concurrent engineering tools difficult. These problems are encountered in the real world already and they are the problems that new automation must solve if there is to be any genuine progress. Since many of the issues raised here do not arise in idealized or abstract enterprise models, they have received less attention than is appropriate.

[2]Task responsibility is represented as an authority to officially submit documentation for a requirements sign off and even these requirements are ultimately boiled down to a specification that the requirement is met if and only if one who has the authority to do so signs that the requirement is met.

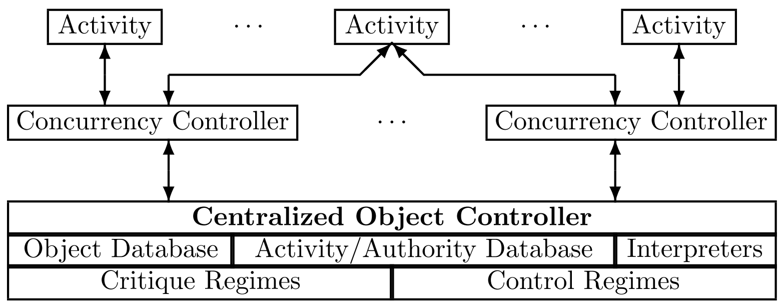

Figure 1: An architecture for concurrent engineering.

Figure 1 is an overview of an architecture to deal with these problems. The centralized controller maintains the object database and a database of authority-to-activity mappings. It also maintains the control and critique regimes both for the use of its interpreters and for dissemination to other parts of the system. The control regime interpreters use the object database to store their processing states. When interpreted control regimes make state transitions, parameters that grossly characterize a set of activities and their interactions are used to initialize a separate concurrency controller. This controller coordinates activities as they perform their functions in regards to a single object.

Concurrency controllers communicate, to the central interpreters, events that must be logged or can cause state transitions in addition to routing data among the activities involved. We are currently designing a specialized language for the concurrency controllers. CSL [2] is an example of a base language that may be suitable for this application. The goal is to make concurrency control perspicuous rather than just efficient.

The activities and authorities database help make the system function coherently. The activity specifications passed to the concurrency controllers are actually handles to *activity expressions* that can be evaluated, on request by the centralized data management system. Evaluation of an activity expression produces a destination address and an indication of the procedure to be followed when an object copy is transmitted there. If the destination is a gateway to an activity's automated subnetwork, this is just business as usual. Interesting cases occur when an activity interface is not electronically connected to its individuals because of security or economics.

The activity and authority database can be modified in many ways. The main variation is to formally copy the system on authority transmittal documents. If this were always done in a timely manner, there would be no problem. However, the system will often find out, for the first time, that a particular authority has been transfered (and where) when the old authority holder is addressed. The system must learn from these encounters too. Many times, an authority will be transfered to another individual in the same activity but not always. Thus, authority transfers can effect the protocol at the activity interface in addition to the destination used.

The reason that the authority database must be centralized is that the information it learns may be needed to properly interpret other objects as well as other states of the same object. For example, an authority associated with a part might be used on its subcomponents also. The object database and the interpreters are centralized because a state transition of one object can effect others. For both databases the requirement is for logical centralization which is another way of saying that things must be kept reasonably consistent. In many cases, an actual implementation can take liberties. One class of liberties is institutionalized by the idea of concurrency controllers. They are possible precisely because they take advantage of opportunities where pinpoint synchronization among objects is unnecessary.

References

[1] Barnett, J.A., An architecture for integrating enterprise automation, Northrop Research and Technology Center, Palos Verdes, CA, 1990.

[2] Barnett, J.A., Module linkage and communication in large systems, in D.R. Reddy (ed.), *Speech Recognition: Invited Papers of the IEEE Symposium*, pp. 500–520, Academic Press, NY 1975.