Designing a Kanban Manufacturing System Using the Server Network Generator (SNG) CASE Tool

A. Bouchentouf-Idriss
Reuters Information Services — Boston
Wentworth Institute of Technology — Boston
USA
Telephone: 617 367 4147
Facsimile: 617 367 4108
E-mail: abdou@ipsaint

L. Zeidner
Manufacturing Engineering Department
Boston University
USA
Telephone: 617 353 3291
Facsimile: 617 353 6322
E-mail: zeidner%bumfga@buacca.bu.edu

Abstract

The absence of CASE tools for software-development of distributed cooperative systems is the major roadblock to effective use of concurrent processing. This paper presents a manufacturing engineering application, a concurrent cooperative processing model of this application, and the Server Network Generator (SNG) CASE tool that was used to design and implement it as a distributed software system in APL2, using inter-user shared variables. The application is a Japanese manufacturing control strategy called the "Kanban System."

Introduction

The recent availability of powerful distributed cooperative-processing hardware platforms presents an opportunity only if their power can be harnessed by the creation of suitable CASE tools. The development of distributed concurrent software systems requires a different sort of CASE tool than those designed for standard sequential von Neumann software development. Issues of resource distribution, asynchronous events, concurrency, data flow, bottlenecks, load leveling, and distributed versioning must be addressed. APL2's inter-user shared variables have proven to be an effective inter-communication platform upon which to develop non-distributed versions of these CASE tools [1].

Recent efforts have created a distributed APL2 shared-variable platform, described in a companion paper in these proceedings [2]. The result is an effective intercommunication platform upon which to develop distributed versions of these CASE tools.

This paper addresses a specific manufacturing engineering application, and then uses it to illustrate the software-development requirements of distributed cooperative-processing systems. The application is a manufacturing production strategy called the **Kanban system**, developed by Toyota Motor Company [3,4]. In a Just-In-Time (JIT) environment, parts are produced at the necessary time, quantity and quality to satisfy market demand, while stock inventory is kept to a minimum. The Kanban system has been a major vehicle for Toyota's drive toward Total-Quality-Control (TQC). A server network implementation of the Kanban system is presented. This illustrates the functional decomposition process that is performed graphically, in the application context, using the SNG.

The Kanban System

In the Kanban system, production is performed at work centers. Standardized carts transport all work-in-progress (WIP) between work centers. Within the factory, the raw materials, semi-finished goods, and finished products, are each considered as separate part types. A "kanban" is a card that is attached to a cart which contains a fixed number of parts, all of one particular part type. There are two types of kanbans, "conveyance" and "production."

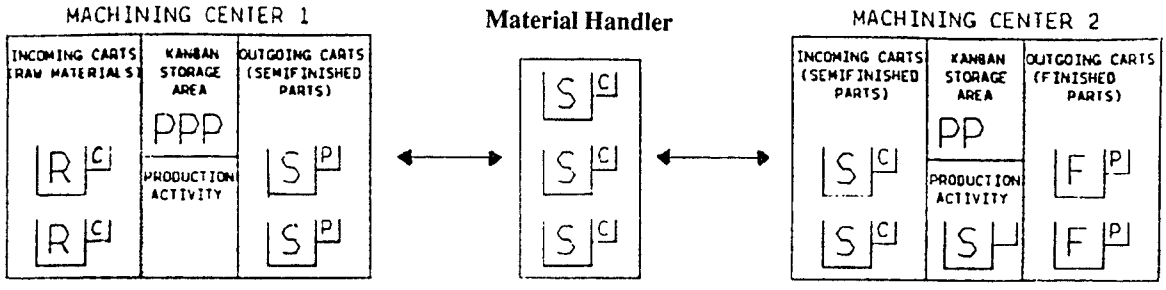

Figure 1: Manufacturing System Layout

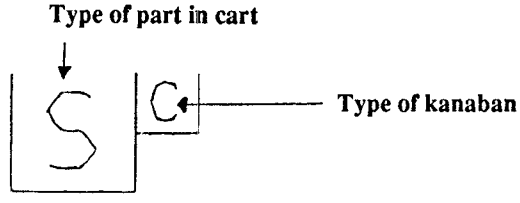

Figure 2: Schematic representation of a cart with a kanban

A factory that uses the Kanban system is depicted in figure 1. This factory is composed of:

- machining center 1, which manufactures semi-finished parts,
- assembly center 2 which assembles these semi-finished parts into finished products, and
- a material handler which maneuvers between these two centers.

Figure 2 shows the schematic representation of a cart. The cart holds a fixed quantity of a particular part type (eg. semi-finished parts), and either a conveyance or production kanban.

Figures 3a-f illustrate the kanban system in operation. Initially (fig. 3a), machining center 1 contains one cart of raw materials, and two carts of semi-finished parts. Assembly center 2 contains two carts of semi-finished parts. There is one production kanban in assembly center 2's kanban storage area. This production kanban triggers assembly center 2 to begin production.

STEP 1: Assembly center 2 uses one cart of semi-finished parts for production. It uses the process information listed on the production kanban, in its kanban area, to assemble the parts. Before processing the first semi-finished part, the operator detaches the conveyance kanban from the cart containing the semi-finished parts, and stores it in the kanban storage area (fig. 3b).

STEP 2: The material handler visits center 2, and finds a conveyance kanban in the kanban storage area. The material handler picks up that kanban which contains all the information describing which part type to convey, where it is located, and to which work center it will go. The material handler heads back to center 1 which produces semi-finished parts (fig. 3c).

STEP 3: At center 1, the material handler takes one cart full of semi-finished parts. This cart has a production kanban attached to it. The material handler removes the production kanban from the cart and places it in center 1's kanban storage area. The material handler then puts the conveyance kanban in the cart, and heads back to center 2 (fig 3d).

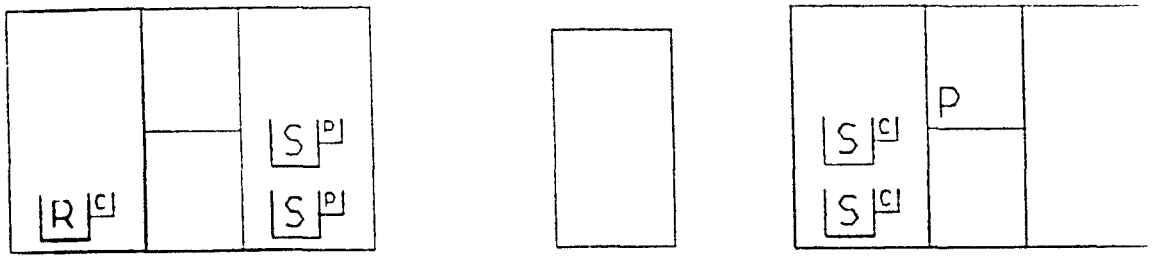

Figure 3a: The initial state of the manufacturing system

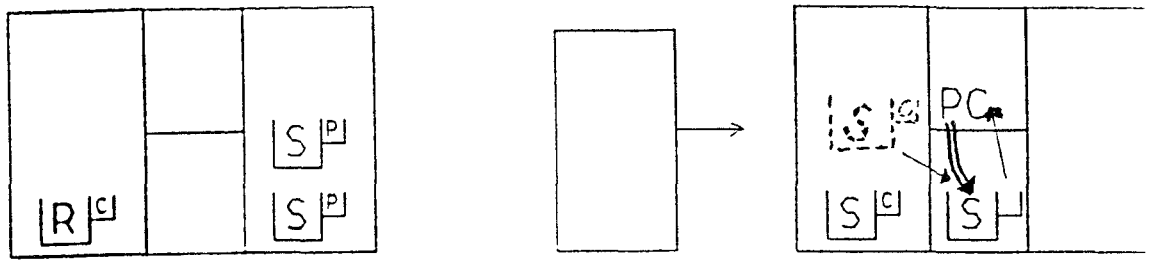

Figure 3b: The manufacturing system after STEP 1

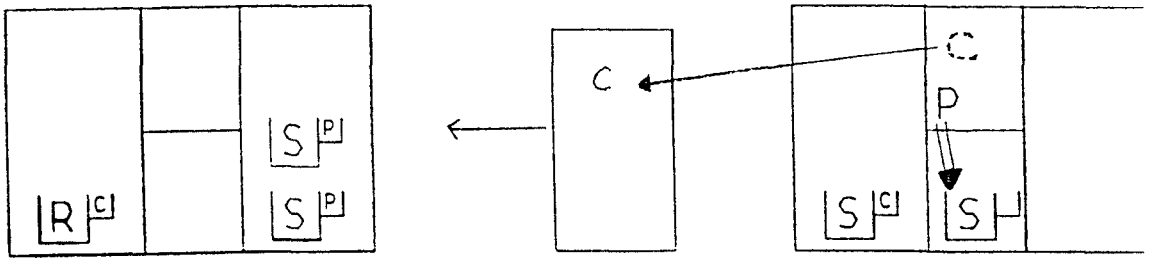

Figure 3c: The manufacturing system after STEP 2

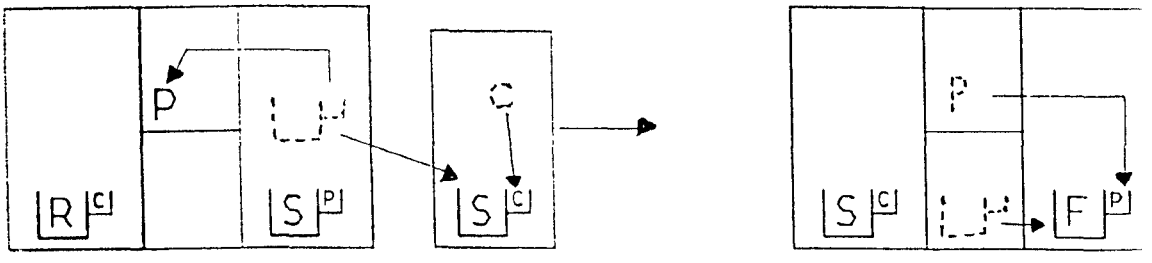

Figure 3d: The manufacturing system after STEP 3

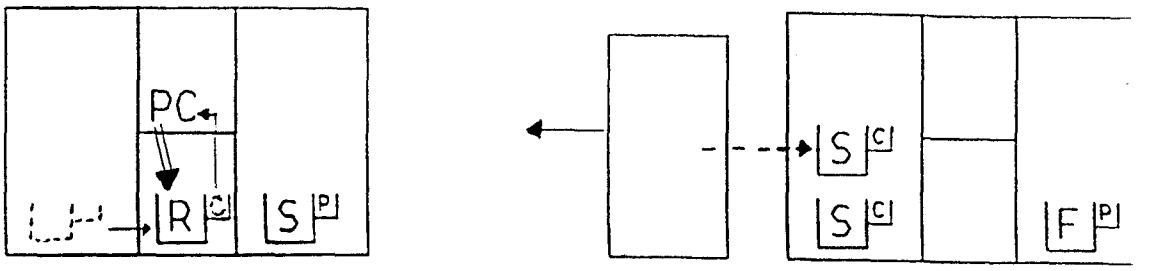

Figure 3e: The manufacturing system during STEP 4

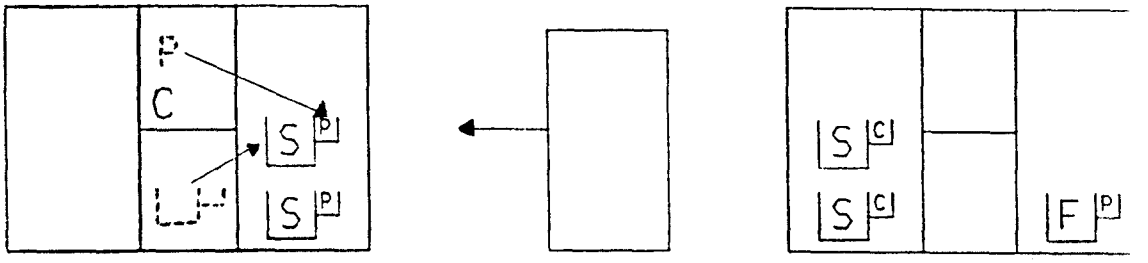

Figure 3f: The manufacturing system after STEP 4

STEP 4: Center 1 checks its kanban storage area, and finds one production kanban. The operator uses the process information listed on this kanban, and starts production using a cart of raw materials. The operator places the conveyance kanban, which is attached to the raw materials cart, in the kanban storage area. The operator starts production, and when a full cart of semi-finished parts has been processed, he attaches the production kanban to it, and moves it to the stock area (fig. 3e-f).

The Kanban system links the throughput of center 1 to the production needs of center 2. This process, when duplicated across a number of machining center, effectively creates a production chaining mechanism; the Kanban system provides a very powerful tool for controlling the maximum amount of WIP on the manufacturing floor. No production occurs without a production kanban, and no stock quantity leaves the stock area without a conveyance kanban. Further, the existence of 2 conveyance kanbans and 3 production kanbans of some part-type means that a maximum of 5 carts of such part-type may exist in inventory. The number of kanbans of each type is varied to control the inventory level.

The Server Network Generator (SNG) CASE Tool Highlights:

The SNG is a CASE tool for graphically decomposing a large problem into a network of inter-communicating functional components that can operate concurrently. These components are implemented as software processes [1,2]. The cooperative network of these inter-communicating software processes is distributed across a computing network.

This approach to CASE separates the overall software-development problem into its two elements of complexity: the top-down decomposition into functional components and the bottom-up software development of each component. The decomposition is expressed graphically using the SNG, while the bottom-up development of each component's software is a more traditional application for CASE, and is performed in APL2.

Each functional component is called a "server" and occupies a virtual machine, containing its own application software and application-independent communication software. Functional decomposition of the application system is not constrained by the SNG; the software designer is free to graphically specify virtually any decomposition, simply by sketching its block diagram in the SNG. The bottom-up design of each functional component constitutes a modular approach to system design. The modularity of virtual machines assures information- and assumption-hiding. Only information that is explicitly shared via APL2 inter-user shared variables is known outside of each server.

The communication requirements between two or more servers are defined simply by graphically linking the servers in the SNG block diagram, and choosing protocols. The designer uses the SNG to choose the communication protocols, and consequently is freed from the responsibility of managing them at the implementation level. The SNG offers a productivity tool with which to specify these communication requirements. Communication paths can be grouped, in the application context, into "communication macros." Each path within the macro has its own protocol specification. A macro can be used between any two servers, simply by graphically assigning the macro to the link connecting the servers on the SNG block diagram. Communication macros are link attributes, and each link can have any number of attributes.

This macro structure makes design change extremely productive. To change a communication relationship between two types of servers, the macro definition is altered; every instance of this macro will adjust to the change automatically. When adding another copy of a server to an existing network, the designer can simply point to any server in the network and duplicate all or some of that server's communication connections for the new server. Communication macros greatly improve the productivity of the software designer by enabling him to attend to modeling the information requirements of his application, without the burden of considering the implementation of these requirements.

The SNG can be used to design and implement an application system that controls real hardware devices, such as those which comprise a manufacturing system. First, the SNG is used to model the system, develop and validate the design. Then, the individual servers that model components are selectively replaced by the real devices. A liaison server, which is able to interface with both the model server and with the physical device, is interposed between the model server and the rest of the network. During development, the liaison server interfaces only with the model server. During operation of the system, the liaison server interfaces only with the physical device. The rest of the network observes the liaison server's outward behavior, which is not dependent upon whether it is interfacing with the model server or with the real device.

Server Network Implementation of the Kanban System

The Kanban system was used to model a Flexible Manufacturing System (FMS). The system is illustrated in figure 4. It is composed of two assembly lines, an FMS machining cell, a fleet of automated guided vehicles (AGV's), and a raw-materials receiving area. The SNG was used as a CASE tool to identify the information flow and communication protocols necessary to implement this real-world manufacturing system. This section presents the Kanban model of the FMS.

The Kanban system is classified as a "pull" system because market demand triggers production and literally pulls materials through the factory as if they were tied together with strings [5]. It is thus reasonable to begin a discussion of this model from the output end.

The **market demand emulator** (in this section, server names are indicated in **bold** print) signals the **assembly lines** to start producing finished products. The **assembly lines** assemble semi-finished parts that are produced by the **FMS machining centers**. The **assembly lines** begin their assembly, and thus generate conveyance kanbans in the process. The AGV's go to the **assembly lines**, pick up these conveyance kanbans, and then travel to the **FMS machining centers** to get new carts of semi-finished products.

This process generates production kanbans in the **FMS machining centers**, which, in turn, trigger the **FMS machining centers** to begin production

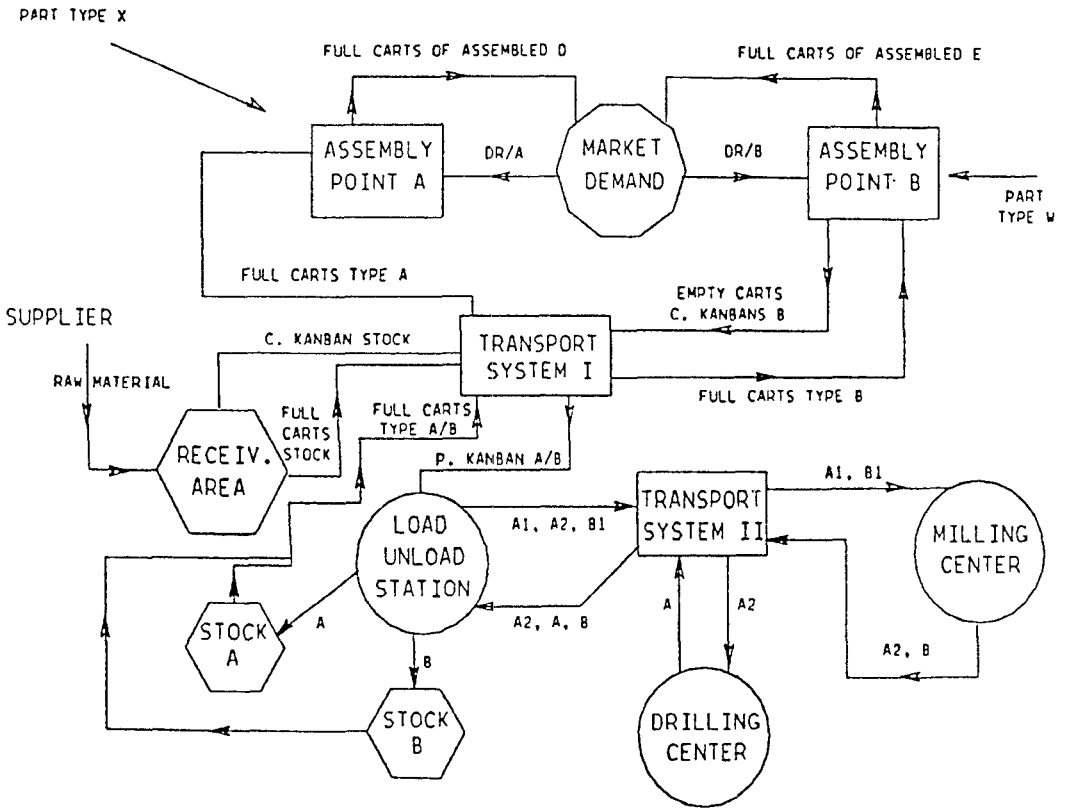

Figure 4: The Kanban System server network structure, as graphically programmed in the SNG

of more semi-finished parts. This production of semi-finished parts can be thought of as replenishing those removed by the AGV, or as replenishing those used in assembly, or ultimately, as replenishing those demanded or "pulled" by the market.

As the **FMS machining centers** produce semi-finished parts, they consume raw material stock, and generate conveyance kanbans. The AGV's pick up these kanbans which cause them to visit the **raw material receiving area** to get more stock. This process generates production kanbans which are transmitted to the **vendor** as a signal to supply more stock.

The **market demand emulator** generates requests to both **assembly lines** for finished products. The order quantities for each part represent product-specific demand rates. These are treated as exogenous decision variables, and are conveyed to the system as a function of time. The **market demand emulator** is in constant communication with both **assembly lines** and has a significant impact on the total production. The **market demand emulator** is equipped with an

algorithm that controls manufacturing production, a cost function to penalize the system for generating production surplus. It also computes backlog demand, policy cost, and production cost. The policy cost is computed by taking into consideration the number of kanbans used.

The **assembly lines** are composed of one stock area, one machine, and one operator. The incoming semi-finished parts arrive in carts. The operator, before processing a cart, detaches its conveyance kanban and stores it in the kanban area. The operator starts production, and continues until exactly one full cart of finished product has been assembled. The same procedure is repeated as long as there are production kanbans, and full carts of semi-finished parts to be processed. If these carts are not available, the **assembly line** stops until more stock is available. Assembly will also stop if the **market demand emulator** orders it to stop.

The AGV's service both **assembly lines**, the **FMS machining centers** and the **raw materials receiving area**. The Kanban system informs each AGV of the number of full carts to bring in during its next trip. Further, beside carrying semi-finished products, raw materials, and kanbans, the AGV's are also equipped with algorithms to handle the part-routing sequences among the **FMS machining centers**. To make semi-finished parts, raw materials stock must go through a number of machining sequences at different centers, and consequently must be transported to these centers in the proper sequences.

The **FMS machining centers** produce semi-finished parts for both **assembly lines**. The **FMS machining centers** include **milling** and **drilling** centers, and a **load/unload** area where raw material stock is mounted on pallets or fixtures for processing, or semi-finished parts are unloaded at the end of their routing cycle. The **FMS machining centers** produce a mixture of part types according to the needs of the **assembly lines**. This need is specified by the production kanbans; whenever the **FMS machining centers'** stock of semi-finished parts is depleted (or "pulled" by the AGV), the machining center is prompted to produce more semi-finished parts. Raw materials are fed into the **FMS machining centers** by the AGV system. The quantity of raw materials required is identified by the Kanban system. This demand is conveyed to the **vendor** by conveyance kanbans which are generated at the **FMS machining centers** as they deplete their stock of raw materials.

Because of the scarcity of manufacturing resources such as raw materials, pallets and fixtures, machines, and AGV's, decision-making control mechanisms are provided to allocate these resources according to algorithms selected by the user. Each machining center is provided with its own scheduling scheme, as well. Machines select which part to process next according to current needs, and consequently may have variations in their set-up times. It is important to note that, for parts moving through the **FMS machining centers**, the AGV's must keep track of all the part routing sequences. Some parts are routed to the **drilling center**, others to the **milling center**, semi-finished parts to the **unload area**, and raw material stock to the **load area**.

Implementation

A Flexible Manufacturing System (FMS) with kanbans was modeled using the SNG CASE tool. The implementation server network consists of a set of 19 concurrent cooperative processes, intercommunicating along 255 APL2 inter-user shared variables. Each server, an APL2 workspace within a VM/370 virtual machine, contains a) application-independent server communication software, b) application-dependent automatically code-generated server network communication drive data, and c) application-dependent response functions. APL2 inter-user shared variables are currently constrained to connect only virtual machines within a single host. All of the virtual machines for this server network were operating within an IBM 4341/12. A related paper within these proceedings [2], describes a recent successful effort that has extended inter-user shared variables to connect virtual machines on any number of VM/370 hosts via a new APL2 auxiliary processor that employs the PCCF feature of PVM [6] as an underlying intercommunication vehicle. This platform enables this interconnectivity to be extended between VM/370 virtual machines and processes operating within personal computers too.

Each FMS component was modeled as a server, thus making it possible to build the entire software system as an integrated composite of

modular components. The functional decomposition into servers is a powerful vehicle for identifying system requirements, and component interaction of a real-world system. The designer only concentrates on his application, i.e. the identification of information needs of a manufacturing system under real conditions. The software designer does not have to manage the communication protocols, because the SNG handles this task automatically for him.

Note that server networks are an approach to software-development for distributed cooperative processing systems, and not merely a simulation tool. There are many simulation tools that enable an engineer to model the behavior of an application and simulate it under various conditions. Server networks enable an engineer to model, simulate, develop, validate, and operate the application system. This is particularly significant in the manufacturing engineering domain, because the constraints are so volatile that it is necessary to return to modeling and simulation many times, long after the system is first used to control production. Server networks allow for incremental modification to the system, without any need to start from initial conditions. Simulations can be used to examine alternative opportunities, using data gathered from production operation. For example, an opportunity to take on additional work can be simulated using last month's production data, to determine how good a choice that would have been if made last month. Or an opportunity to purchase an additional machine can be simulated to determine how it would have helped or hurt last year's production.

The Kanban system required initialization, and testing under a variety of circumstances. Server networks include "utility servers" which are application-independent and provide control of the server network, and a means of evaluating its behavior. Examples of utility servers are the console, logger, status display, and clock servers.

Conclusions

Current CASE technology has not been focused on the design of concurrent cooperative processes. Server networks provide an attractive approach to developing such systems. They benefit from the following advantages of APL2 inter-user shared variables:

- symmetry: as opposed to an asymmetric client/server relationship,
- robustness: the designer can focus on his design problem, rather than on checking to guarantee that each message sent actually arrived correctly,
- dynamic creation: no declarations or groundwork are necessary ahead of time, so that as the need arises for a shared variable it can be created spontaneously, or destroyed later if necessary,
- communication speed: efforts have been made to assure the fastest possible speed of communication between virtual machines, through the use of a discontiguous shared segment (DCSS); however, this is also the factor that limits the APL2 inter-user shared variable implementation to one host,
- access control and status information: these tools enable the software developer to build his own communication constructs upon the shared-variable platform, and
- data-driven creation: because their creation is based upon a function call (quadsvo) using the shared variable name and partner identifier as data arguments, it was possible to build application-independent software to establish and manage these shared variable connections.

The SNG has been ported to a distributed cooperative processing hardware platform consisting of a set of IBM 7437 VM/SP Technical Workstation processors interconnected via token-ring LAN [7]. This presents dramatic possibilities for creating concurrent applications that can benefit from the substantial computing power that has recently become available in distributed cooperative processing hardware. Its success depends upon the continued development of new high-productivity CASE tools to harness this computing power. The SNG currently addresses the top-down software-development task with powerful graphical programming and automatic code generation techniques. The bottom-up software-development task currently relies upon the power and productivity of APL2 to address the complexity of each server's functionality. Current research efforts aimed at enhancing the SNG CASE tool are focused on providing better CASE tool support of this bottom-up task. The integration of our Expert-System Generator (ESG) into the SNG will address this need in a variety of ways.

References

1. L.E. Zeidner, "Server Networks: Software Integrated Tools for CIM," Proceedings of the International Conference on Computer Integrated Manufacturing, RPI, NY (1988).

2. L.E. Zeidner, "The Server Network Generator (SNG): A CASE Tool for Distributed Cooperative Processing," Proceedings of APL'91 (these proceedings), ACM (1991).

3. Y. Sugimori, K. Kusunoki, F. Cho, and S. Uchikawa, "Toyota Production System and Kanban System: Materialization of Just-in-Time and Respect-For-Human System," International Journal of Production Research, 15(6), pp 553-564, (1977).

4. Y. Monden, Toyota Production System, IIE (1983).

5. Y. Seo and P.J. Egbelu, "Configuration and Operation of a Pull-Type Flexible Manufacturing System," Manufacturing Review, 4(1), pp 44-50, ASME (1991).

6. IBM, Virtual Machine/Pass-Through Facility, Managing and Using, Release 4, SC24-5474-00 (1988).

7. IBM, 7437 VM/SP Technical Workstation: User's Guide and Reference, SA23-0351-00 (1988).