

1990

# Integrating demand and supply management : some software issues

Sarosh Talukdar  
*Carnegie Mellon University*

Carnegie Mellon University.Engineering Design Research Center.

Follow this and additional works at: <http://repository.cmu.edu/ece>

---

This Technical Report is brought to you for free and open access by the Carnegie Institute of Technology at Research Showcase @ CMU. It has been accepted for inclusion in Department of Electrical and Computer Engineering by an authorized administrator of Research Showcase @ CMU. For more information, please contact [research-showcase@andrew.cmu.edu](mailto:research-showcase@andrew.cmu.edu).

**NOTICE WARNING CONCERNING COPYRIGHT RESTRICTIONS:**  
The copyright law of the United States (title 17, U.S. Code) governs the making of photocopies or other reproductions of copyrighted material. Any copying of this document without permission of its author may be prohibited by law.

**Integrating Demand and Supply Management --Some  
Software Issues**

by

Sarosh Talukdar

18-16-90 C§3

# **INTEGRATING DEMAND AND SUPPLY MANAGEMENT-SOME SOFTWARE ISSUES**

**Sarosh Talukdar  
Department of Electrical and Computer Engineering  
Carnegie Mellon University  
Pittsburgh,  
PA 15213, USA**

**This work has been supported by the Engineering Design Research  
Center, an NSF Engineering Research Center.**

# INTEGRATING DEMAND AND SUPPLY MANAGEMENT-SOME SOFTWARE ISSUES

Sarosh Talukdar  
Department of Electrical and Computer Engineering  
Carnegie Mellon University  
Pittsburgh,  
PA 15213, USA

## ABSTRACT

This paper lists some of the architectural and control alternatives that should be considered in seeking to integrate software for demand and supply management. The paper also describes FORS, an environment for implementing these alternatives. FORS treats both procedures and data sets as objects. A visual programming interface is provided for the manipulation of these objects. The interface shields the user from much of the drudgery of building and using systems for distributed problem solving.

## INTRODUCTION

Computer hardware grows in raw capability at an average rate of several percent per month. In contrast, energy management systems (EMS's) change at much slower rates, often taking decades for major upgrades and the incorporation of new technologies. One of the main reasons is that EMS software is not organized for expansion. Adding new programs and computers is difficult, especially if they come from different vendors.

Besides expansion, organizational factors exert large, often dominant effects on a system's computational capabilities-what it can compute and how fast-as well as the ease and effectiveness with which humans can interact with the system. Now that efforts to automate distribution systems and demand management have gained significant momentum, and ways to integrate these schemes with supply management (EMS's) must be considered, it is time to revisit the issue of how all this software ought to be organized.

Are there formal techniques for optimizing organizations of software or, for that matter, any other type of information processing agents? Unfortunately, the answer is "no." Most of the relevant knowledge has been collected from the study of human

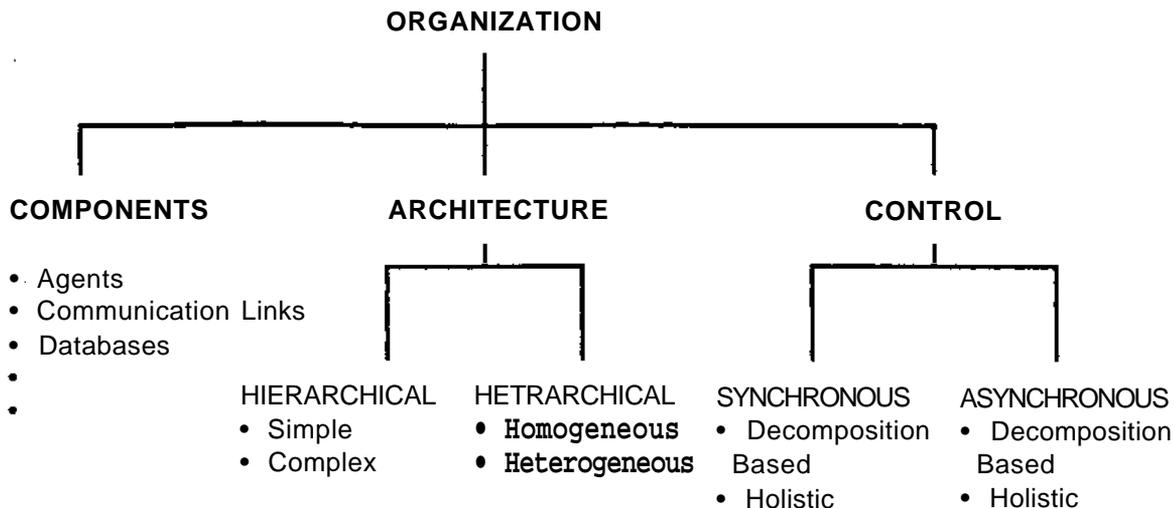
organizations. This body of knowledge tends more to guidelines and case studies than formal synthesis, optimization and analysis techniques (see, for example, [1]). Therefore, the person who wishes to design a good software organization must rely heavily on analogy, intuition and judgment. The purpose of this paper is to aid such a person in two ways. First, by listing the major categories of organizational alternatives that he or she should consider. And second, by describing a framework, called FORS, for implementing these alternatives.

## ORGANIZATIONAL ALTERNATIVES

Human and software organizations for information processing have a great deal in common. For the time being, we will not distinguish between them. The main parts of an information processing organization are shown in Fig.1 and its elements defined below.

## Terminology

- **Agents and organizations.** An agent is a person or program capable of performing information processing tasks. An organization is a collection of agents that collaborate in performing a complex task. Collaboration means the exchange of raw or processed data.
- **Architecture:** the command and communication structure of an organization. The architecture determines who reports to whom, who may communicate with whom, and who has access to what resources.
- **Hierarchy:** an architecture with two or more levels in its command structure. A simple hierarchy has just two levels. Blackboards, as used in many artificial intelligence applications, are an example of a simple hierarchy in which a planner or scheduler determines which programs will have access to the blackboard and when.
- **Heterarchy:** an architecture in which all the agents have equal status, as is the case in an



**Fig. 1: An Information Processing Organization**

artificial neural net and almost the case in many insect societies.

• **Control:** the standards and policies that determine how an organization goes about doing its work. For instance, control policies determine how tasks are allocated to agents, how deadlines for the completion of tasks are set, when agents must communicate with one another, and what is to be done when things go wrong.

• **Synchronous control:** a collaboration scheme in which information exchanges occur at predetermined staging points [2]. An agent must wait at each of its staging points until all the prescribed information exchanges have been completed. Most existing software organizations use strongly synchronous control schemes. Human organizations, in contrast, gain much of their power and especially their ability to handle contingencies, from the use of asynchronous approaches.

• **Asynchronous control:** a collaboration scheme with no staging points. Agents working in parallel exchange information spontaneously or whenever they can, rather than at predetermined points [2].

• **Decomposition-based-control:** any approach that relies on breaking the overall task into loosely coupled subtasks, i.e. a divide-and-conquer approach.

• **Holistic control:** any approach in which the overall task is left essentially intact while it is worked on by a team of agents.

• **Contingency:** an unplanned event, such as a line-to-ground fault, whose occurrence could have bad effects.

In power systems, as in other forms of engineering, we use a wide variety of specialized data structures and representations, such as single line diagrams and load flow equations. Each of these structures will be called an "aspect" of a power system. More specifically, an aspect constitutes a view, model or partial description of the function, structure or behavior of a power system.

Any computational process can be thought of as tracing a path from a set of given or initial aspects, through a series of intermediate aspects, to a goal aspect. For example, the computational path of a load flow begins at aspects capturing the topology of a network, the impedances of its branches, certain given values at its load and generator buses, and an initial guess for the values of the network state variables (essentially, bus voltages and line flows). The path ends at an aspect containing converged values of the state variables, in between, it passes through aspects containing progressively better approximations to the state variables, obtained by Newton-Raphson iterations. To represent and help visualize such paths we define three new terms: aspect-spaces, operators and TAO graphs (3). [4]. An **aspect-space** is a class of aspects, for instance, the class of all possible load flow results for networks with 2000 or less buses (which happens to be the state space for these networks). An **operator** is an agent whose purpose is to transform points of one aspect-

space into those of another. (In the load flow example the main operator is a Newton-Raphson algorithm. At each iteration it maps a point from the state space into another point in the same state space.) A TAO graph is a directed and/or graph whose nodes represent aspect-spaces and whose arcs represent operators. All the computations made possible by a set of operators are represented by paths in the associated TAO graph. Cycles represent opportunities for iteration. Disconnected sub-graphs represent isolated computational processes with no means of communication.

### **Good Software Organizations**

What are the properties of a good software organization? Two are given below.

1. Flexibility, meaning the easy addition of new operators and aspect-spaces so existing functions can be readily upgraded and entirely new functions can be seamlessly integrated with existing functions. In TAO graph terms, this integration problem is equivalent to building paths to connect what would otherwise be isolated sub-graphs. A good organization would allow these paths to be easily built. For example, think of an existing package of programs written in Fortran and resident in computers built by vendor-X. Suppose the outputs of these programs are to drive a new package of expert systems written in OPS-83 and resident in computers built by vendor-Y. A good organization would make it possible to easily produce software that would overcome the hardware and language incompatibilities of the two packages.

2. Effective contingency handling, meaning that a good organization will recover from the contingencies that plague computational processes such as errors, missing data, convergence failures, and getting stuck in a local optimum.

### **Remarks**

What architectures and control strategies should be employed to obtain good software organizations? As pointed out earlier, there are no definitive answers.

Human organizations are by far, the most capable organizations known and therefore, models after which software organizations might be patterned. From an architectural point of view, human organizations are notable for their use of complex hierarchies and

powerful facilities for lateral relations (collaborations among agents at the same level [1], [5]). From the viewpoint of control, human organizations are notable for their use of distributed problem solving for tasks of all levels of complexity, and some of the most productive human organizations [13] rely heavily on asynchronous, holistic or team-based approaches for tasks of medium to low complexity.

I suspect that complex hierarchies can be effective only if they are staffed with agents far more intelligent than we can make programs today. This leaves simple hierarchies and hierarchies as options for software organizations. Both are good options but the latter has been under exploited even though it has great potential. Computational processes at the level captured by a TAO graph are hierarchical. It should be easier to implement such processes in organizations or subdivisions of organizations that are also hierarchical.

Aside from complex hierarchies, all the architectural and control features from which human organizations draw their strength would seem useful to organizations of less intelligent agents, and therefore, should be included in the techniques used by the designers of software organizations.

Turning now to the two properties of a good organization listed earlier, note that at present, flexibility in software organizations is limited by incompatibilities among the programming languages, operating systems and computers that are in use. Therefore, the first step in increasing flexibility is to develop environments that can overcome these incompatibilities. A prototype of one such environment is described in the next section.

Many techniques for contingency handling have been described in the literature (see, for example, [1], [2], [5], [13]). Four recommendations constitute their essence:

- provide alternate computational paths so that an obstacle along one path does not bring all progress to a halt;
- strengthen the lateral relations among agents so they can better collaborate;
- make provisions for spontaneous and opportunistic collaborations (asynchronous control) so that "mid-course corrections" can

be easily made;

- use team-based approaches to pool knowledge and capabilities as well as to reduce the chances of any one agent becoming indispensable.

## **FORS**

FORS is an aid for building software organizations. It provides the means for assembling arbitrary architectures and implementing arbitrary control strategies. The rationale is that no single type of architecture or control strategy can suit all situations, and therefore, it is best to allow for the variety of alternatives listed in the previous section. In addition, FORS has the following features:

- facilities to easily interconnect programs written in different languages and resident in different types of computers.
- object oriented approaches to both operators (tools) and aspects (data). There is a symmetry between these entities in many, if not most, engineering problems that is maintained in FORS. In contrast, other organization building aids tend to be biased either in favor of tools or data.
- a visual programming interface that is almost self explanatory, and shields users from details in which they have no interest. In particular, users can set up and run serial or parallel processes without having to bother with how programs are actually invoked or where they reside.

FORS is built on top of an older set of aids called DPSK. In the following material we will first describe DPSK and then FORS.

## **DPSK (Distributed Problem Solving Kernel) [5], [6]**

DPSK provides the organization builder with a small set of primitives. These primitives have been designed to be inserted in the instructions of an expandable set of languages. Presently, the set includes C, Lisp, Fortran and OPS5. With the primitives, organization builders can readily synthesize all the alternatives from the preceding section and thereby, assemble arbitrary organizations composed of agents written in a variety of dissimilar languages, and distributed over a network of computers. Theoretically, the numbers of programs and computers can be arbitrarily large.

DPSK itself is written in C for networks of computers running Unix. Internally, DPSK

employs a shared memory that is distributed over the participating computers.

## **Primitives**

DPSK contains 12 primitives that can be divided into four categories -commands, synchronizers, signals, and transactions. The command primitives are used to activate and control programs. An agent can "run," "suspend," "resume," or "kill" other agents in any of the processors in the network. This also allows any number of program clones to be created and run in parallel.

The synchronization primitives are used to create and check for the occurrence of "events" and thereby, implement synchronous control strategies. For instance, to ensure that an activity X in agent A finishes before agent B is allowed to begin, one would insert primitives in agent A to assert the event X, and in the beginning of B, to wait for the assertion of X.

The signal primitives are used to signal the occurrence of a contingency or to interrupt the execution of preselected groups of processes and cause them to execute portions of their code designated for exception or contingency handling.

Transaction primitives are used to structure and access the shared memory.

## **FORS (Flexible Organizations)**

FORS is an object-oriented framework for integrating tools and data. FORS comprises two major entities: data-objects and tool-objects. In addition, FORS addresses the issue of control and has an icon-based interface suitable for both novice and expert designers.

### **Data-Objects**

Each data object can store one or more aspects. FORS allows an expandable library of data objects. Data objects have facilities to handle functions such as translation from one format to another, editing, browsing, and error detection and correction.

### **Tool-Objects**

FORS allows for an expandable library of tool objects. Tools may be written in a number of languages. Currently, the list includes C, FORTRAN, LISP and OPS5. Each tool object contains a template to describe the principal

characteristics of the tool, for example, specifications for input and output and which machine it resides on. Tools may be run in parallel regardless of where they are located.

### **Interface [7], [11]**

FORS provides a multi-window graphical user interface, where the tool and data objects are represented by icons. Each window and icon in turn, have various menus and methods attached to them. This type of a visual interface hides the lower level systems details concerning the tools and data, allowing the user to manipulate them easily.

### **Status**

FORS is very much an experimental package that undergoes frequent revisions. Nevertheless, it is being used to build software organizations for a number of research projects including CQR, a system for handling contingencies in electric power systems [10]; ASE, a system for the design of automobile parts [8], [9]; and IBDE, a system for designing high-rise buildings [12].

### **SUMMARY**

The software organizations used in power systems and other engineering domains tend to be inflexible and poor at handling contingencies. I suspect that these weaknesses result from the use of simple hierarchical architectures and strongly synchronous, decomposition-based control schemes, to the exclusion of all other options.

Any computational process is, at its core, hetrarchical and can be represented by a TAO graph. When a software organization replicates this TAO graph in part of its architecture, then there is a straightforward and natural mapping between the process and the organization. If the organization can easily reconfigure this TAO graph, then its ability to accommodate different computational processes, that is, its flexibility, is greatly increased.

Asynchronous and team based approaches to control have advantages over strongly synchronous and decomposition based approaches. In particular, asynchronous schemes allow for more opportunistic decision making and are potentially more robust (better at contingency handling). Team based schemes

are also more robust, especially for tasks of medium to low complexity.

A package called FORS has been developed to aid the builders of distributed software organizations in implementing their architectural and control choices. FORS can accommodate both the traditional and the newer options, but is especially well suited to implementing the hetrarchical architectures captured in TAO graphs. Changes can be made fairly easily and a visual programming interface eliminates much of the drudgery of working with distributed computer systems and programs written in different languages.

### **REFERENCES**

- [1] Shafritz, J. M., Ott, J. S., (editors), *Classics of Organization Theory*, Dorsey Press, 1987.
- [2] Talukdar, S. N., Pyo, S. S., Mehrotra, R., "Designing Algorithms and Assignments for Distributed Processing," EPRI Report EL-3317, Nov. 1983.
- [3] Talukdar, S. N., Westerberg, A. W., "A View of Next Generation Tools for Design," presented at 1988 Spring National Meeting, AIChE, New Orleans, LA, March 6-10, 1988.
- [4] Talukdar, S. N., Fennes, S., "Towards a Framework for Concurrent Design," Proceedings of the ASME Winter Annual Meeting, San Francisco, CA, December 11-14, 1989.
- [5] Talukdar, S. N., Cardozo, E., "Building Large-Scale Software Organizations<sup>1\*</sup> in Expert Systems for Engineering Design, edited by M. D. Rychener, Academic Press, 1988.
- [6] Cardozo, E., "A Kernel for Distributed Problem Solving," Ph.D. Thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, January 1987.
- [7] Papanikolopoulos, N., "FORS: Flexible Organizations," Masters Project Report, Department of Electrical and Computer Engineering, Carnegie Mellon University, November 1988.
- [8] Sapossnek, M., Talukdar, S., Elfes, A., Sedas, S., Eisenberger, E., Hou, L., "Design Critics in the Computer-Aided Simultaneous Engineering (CASE) Project," presented at the ASME Winter Annual Meeting, Symposium on Concurrent Product and Process Design, San Francisco, CA, Dec. 1989.
- [9] Talukdar S. N., Sapossnek, M., Hou, L., Woodbury, R., Sedas, S., Saigal, S., Jaeger, J., "Autonomous Critics," Proceedings of the

Second National Symposium on Concurrent Engineering, West Virginia University, Morgantown, WV, February 7-9, 1990.

[10]. Stoa, P., Talukdar, S. N., Christie, R., Hou, L., Papanikolopoulos, N., "Environments for Security Assessment and Enhancement," Second Symposium on Expert Systems Applications to Power Systems, Seattle, WA, July 1989.

[11]. Vidovic, N., Siewiorek, D., and Newbery, F., "A Graph Based Environment," Technical Report CMU-CAD-87, 1987.

[12] Fenves S. J., Hendrickson C, Maher M. I., Flemming U., Schmitt g., "An Integrated Software Environment for Building Design and Construction," Computer Aided Design 22(1):27-36, 1990

[13] Dertouzos, M. L, Lester, R. K., Solow, R. M. "Made in America," MIT Press, 1989.