FORS: An Integration Framework for Design

by

Sarosh Talukdar

18-17-90 C.3

FORS: AN INTEGRATION FRAMEWORK FOR DESIGN

Sarosh Talukdar, Lily Hou and Pedro Sergio de Souza

Engineering Design Research Center
Carnegie Mellon University
Pittsburgh, PA 15213

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

FORS: AN INTEGRATION FRAMEWORK FOR DESIGN

Sarosh Talukdar, Lily Hou and Pedro Sergio de Souza

Engineering Design Research Center
Carnegie Mellon University
Pittsburgh, PA 15213

ABSTRACT

This paper lists the different architectural and control alternatives that the builders of design-tool-organizations should consider. The paper also describes FORS, an environment for implementing these alternatives. FORS treats both tools and sets of data as objects. A visual programming interface is provided for the manipulation of these objects. The interface shields the user from the low level details of distributed problem solving.

INTRODUCTION

The design of complex artifacts, such as micro-electronic chips, cars and high-rise-buildings, is increasingly dependent on suites of computer-based tools. These suites increase productivity by amounts that depend, in large part, on their internal organizations (that is, on how their tools are interconnected and controlled). Organizational factors grow in importance with increase in the size of the suite and affect critical productivity issues such as the effort required to expand and reconfigure suites of tools, the effort required to learn and use the tools, and whether the tools will interact in beneficial or destructive ways.

The importance of organizational factors raises the question: are there formal techniques for optimizing them? Unfortunately, the answer is "no." Organizational theory has developed largely through empirical studies of human organizations. Though fairly mature and supported by a considerable body of literature, it 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 tool-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.

ALTERNATIVES

Engineering design is a special case of information processing or decision making. The main parts of an information processing organizations are shown in Fig.1. Some useful terminology is 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.
- **Hetrarchy:** an architecture in which all the agents have equal status, as is the case of **a** neural net.
- **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.
- **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.

Engineering design is distinguished from other forms of information processing by the use of specialized data sets we call **aspects.** Each of these aspects constitutes a view, model or partial description of the function, structure or behavior of the artifact being designed. Examples of aspects are sets of specifications, circuit diagrams, blueprints and test results.

The process of design can be thought of as calculating desired or goal aspects from given aspects. For example, in the design of **a** house the goals are often to obtain construction drawings, a construction schedule and a set of cost estimates, given the customer's specifications.

In designing complex artifacts, the computational paths that lead from the given aspects to the goal aspects are often long and pass through many intermediate aspects. To help visualize these paths we define three new terms: aspect-spaces, operators and TAO graphs. An **aspect-space** is a class of aspects, for instance, all resistive circuits with 20 or less nodes. An **operator** is an agent whose purpose is to transform elements of one aspect-space into those of another by processes of synthesis, optimization or analysis. (For a more extensive taxonomy of operators see [3].) A **TAO graph** is a directed and/or graph whose nodes represent aspect-spaces, whose arcs represent operators, and whose cycles represent opportunities for the iterative refinement of aspects [3], [4].

The TAO graph for a design organization with several departments contains a sub-graph for each department. The absence of connections among such sub-graphs indicates that the departments have no means of direct collaboration, and therefore, are likely to produce conflicting results. Processes for establishing collaborative arrangements among departments, such as manufacturing, maintenance and marketing, are beginning to be called "concurrent design" or "simultaneous engineering" [3]. In TAO graphs, such arrangements are represented by arcs connecting the sub-graphs for the different departments.

Good Design Organizations

Three desirable properties for design organizations are:

1. Easy addition of new operators and aspect-spaces so the organization can grow and change painlessly;
2. Close coordination among sub-divisions of the organization so the principles of concurrent design can be practiced. This means placing operators and aspect-spaces to link the computational paths used by the sub-divisions.
3. Effective contingency handling. (By a contingency we mean an unexpected disturbance that causes the existing plans for satisfactorily completing a task to go awry.) Many techniques for contingency handling have been described in the literature (see, for example, [1] and [5]). In essence, they involve strengthening the lateral connections among computational paths, making provisions for spontaneous exchanges of information (asynchronous control), and providing alternate computational paths to take when failures in the originally selected path occur.

Remarks

What architectures and control strategies should be employed to obtain good tool-organizations? As pointed out earlier, there are no definitive answers. Human organizations have been extremely effective in their use of complex hierarchies [1], [5]. However, we suspect that getting the most out of a complex hierarchy requires more intelligence than existing tools can provide. Therefore, we favor hetrarchical architectures. With regard to control, we feel that asynchronous and holistic approaches have great potential and need to be explored. Very complex problems must, of course, be decomposed before they can be tackled. But problems of medium complexity are often best tackled by team based approaches that leave the problem intact [2].

FORS

FORS is a visual programming interface for DPSK, a facility for distributed problem solving. Together, FORS and DPSK provide the means to assemble architectures and implement control strategies for tool organizations. Since it is not clear that any single type of architecture or control strategy will always be best, FORS has been designed to allow for all the alternatives listed in the previous section. In addition, facilities have been provided to shield users from details that are of no interest to them. For instance, designers using organizations built with FORS need know nothing about how tools are actually invoked or in what computers they reside. Instead, tools and aspects are represented by icons that the designer can interconnect on the screen to form arbitrary computational paths.

There is a symmetry between design tools and aspects that has been maintained in FORS. Both tools and aspects are treated as objects of equal importance. This is in contrast to other organization building frameworks which tend to be either tool-centered or data-centered.

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 some activity in Agent A finishes before Agent B is allowed to begin, one would insert primitives into Agent A at the appropriate point to assert an 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 to handle such exceptions.

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 also allows 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 designer to manipulate them easily.

Status

FORS is an experimental package that is still under development though it has been used to build a number of design organizations for research projects. These include ASE, a system for the design of automobile parts that is notable for its use of autonomous programs called critics [8], [9]; CQR, a system for handling contingencies in electric power systems [10]; and IBDE, a system for designing high-rise buildings [12].

SUMMARY

Traditional tool-organizations rely heavily on simple hierarchical architectures and synchronous, decomposition-based control schemes. This paper has pointed out that complex hierarchies and hetrarchies may have advantages as architectures, and asynchronous, holistic approaches may have advantages as control schemes. The paper has also described a framework, called FORS, for rapidly implementing arbitrary architectures and control schemes.

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., Fenves, 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" 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.f "FORS: Flexible Organizations,", Masters Project Report, Department of Electrical and Computer Engineering, Carnegie Mellon University, November 1988.
[8] Sapossnek, M_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., 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