Modeling and Designing Computational Organizations

Young-pa So and Edmund H. Durfee
Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, MI 48109
frege@caen.engin.umich.edu, durfee@caen.engin.umich.edu

Abstract

In this paper, we address the following question: *to what extent can organization design be treated as a routine design problem, with a well-defined space of possibilities and explicit evaluation criteria?* The kinds of organizations we are particularly interested in are computational organizations which consist of computational agents that cooperate with one another on their organizational tasks. We present a model of computational organization design that uses predictive knowledge about how exactly various task-environmental and organizational factors determine the performance of the organization.

Introduction

We have been investigating ways to endow a set of agents with the capability to cooperate intelligently under various task environments. Our specific approach has been to give the agents the ability to form and change their task organization under potentially complex, uncertain, and/or dynamic task environments. The process of changing an organization by one or more members of the organization in order to improve performance has been called *organization self-design (OSD)* in the distributed artificial intelligence (DAI) literature [Corkill, 1983, Corkill and Lesser, 1983, Gasser and Ishida, 1991].

In [So and Durfee, 1993], we have presented a model of OSD which involves an organization design and evaluation component using a definite performance criterion. In this presentation, we elaborate on our model, and address the question of *to what extent can organization design be treated as a routine design problem, with a well defined space of possibilities and explicit evaluation criteria.*

Computational Organization Design

What is *computational organization design (COD)*? This seems to be the first question we must ask. It may be a simple question, but how we define it will be critical to future research in this new field. The word *computational* can be applied either to the word organization or the word design. That is, COD can either mean design of computational organizations or design of organizations by means of computers or computational modeling or possibly both. Our interest lies in both meanings. That is, we are interested in how computational methods can be applied in the design of organizations consisting of computational elements.

The question "what is an organization?" is at least as hard as that of COD. Here, also, there can be two different meanings. In the broader sense, the concept of organization is a way of looking at complex phenomena. That is, in this sense, organization is in the eye of the beholder. The elements that we see in the organization may in fact not be aware of the organization, and can be seen as independent entities. In the narrower sense, an organization is a particular entity whose elements exist solely for the purpose of that organization. The elements of the organization have no meaning outside the organizational context. For example, anarchy can be seen as a type of organization in the broader sense but not necessarily in the narrower sense - since different members of the anarchy may see the anarchy as serving different purposes, or some even may not see it as an anarchy. That is, an anarchy is a non-organization in the narrower sense of organization.

However, the distinction between these two senses become blurred when we consider dynamic and/or virtual organizations where some elements are *autonomous* and may create, modify, or destroy an organization it is a member of, or join or leave multiple organizations as they choose. In these cases, the autonomous elements which we call *agents* do seem to have existence and meaning outside of any particular organization or even outside of all organizations, yet as members of an organization they are committed to serve the organization's purpose. Alternatively, we may see an agent as a singleton organization with one element, namely itself. In our research, we start with the narrower sense of organization, and expect to incrementally broaden the definition as required.

[0]This work has been supported, in part, by NSF PYI Award IRI-9158473, and by a grant from Bellcore.

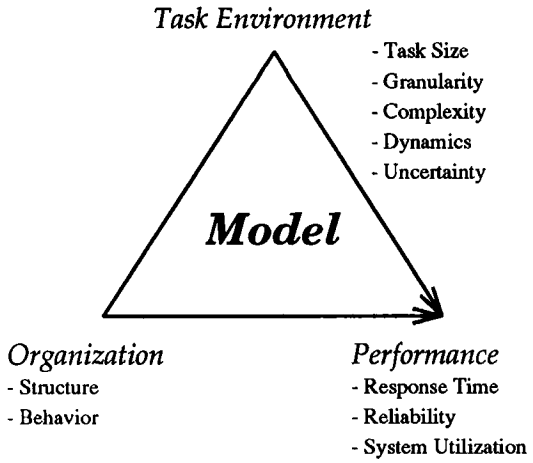

Figure 1: Model of Organizational Performance

Our model of COD is directly related to our model of organization and the relevant ontologies surrounding it. In order to design an organization, we must first know what is to be designed, that is, know what are the components and features of the organization we can select and combine. These we call *organizational factors*. Secondly, we need to know what is a good design or a bad design, that is, we need some criteria for evaluation. These we call the *performance metrics*. Thirdly, we need to know how an organization works in order to link the structure of elements and their features to the evaluation criteria. This we call the *organizational performance model*. If we see an organization as a closed system, these three things may suffice. However, many interesting organizations are open systems, and interact with non-organizational external elements and processes. These we call *task-environmental factors*. Moreover, in many cases, these task-environmental factors affect the performance of the organization. Thus we require the knowledge of task-environmental factors in addition to the organizational factors for organization design. Figure 1 is a diagram showing our model of organizational performance. The arrows indicate the dependency relationships.

The organizational and task-environmental factors define an *organizational design space (ODS)*. A designed organization is a point in ODS. The performance measure is a function over ODS onto the performance metric space. Then, we can define the COD problem as a search through ODS for an organization with acceptable performance. In other words, our design process model (DPM) for COD is a generate-and-test or search process model. We note that our model requires the designer to have predictive knowledge concerning exactly how the various factors determine the performance of the organization. This knowledge is then embodied in the performance evaluation function to be used in the design process. This allows automation of computational organization design.

In the following sections, we elaborate on the components of our model. It consists of three components: the organization model, the task-environment model, and the performance model. In its general form, it is still informal and verbal at this stage of research, and thus not amendable to precise computational implementations. However, for specific cases, as we show later in this paper (see also [So and Durfee, 1993]), it can be realized computationally. Our goal is to formalize them more generally so that we can embody them as *organizational knowledge* into autonomous agents capable of intelligent cooperation, enabling them to use it in the process of organizational self-design (OSD).

Organization Model

We start with the narrower sense of the word 'organization'. Thus, we initially focus our attention to work organizations. That is, organizations that are designed for some definite work to be done. More specifically, we mainly deal with computational organizations where the type of work done by the organization is the computation (or execution) of a complex function which is functionally decomposable into subfunctions.

More concretely, we think that a specification of a work organization should at least include the following components [So and Durfee, 1993]:

1. The set of tasks and subtasks to be done.
2. The set of agents participating in the organization.
3. An assignment of the tasks and subtasks to the participating agents.
4. A work flow structure which dictates how the tasks and subtasks are to be distributed among agents and how the result and partial results are to be synthesized.
5. Optionally, a set of resources aside from the agents and a set of constraints on the usage of those resources may apply to agents.

The model we have for agents in the organization is simple. They are capable of transferring and routing the (sub)tasks and (sub)results, and capable of executing a set of primitive functions.

Task Environment Model

By a task environment, we mean task and environmental characteristics that affect the performance of the organization. For example, type, size, rate of change, and structure of the task and the world are common important characteristics affecting the performance of many organizations.

In our research, we seek to be able to come up with a more comprehensive model of tasks and environments so that we can explicitly represent and reason about different kinds of task characteristics, and also incorporate task and environmental uncertainties, complexities, and dynamics into the task environment model.

We believe that many terms such as task complexity, task dynamics, task uncertainty, environmental complexity, environmental uncertainty, and environmental dynamics must be precisely defined.

Task Model

As mentioned previously, we start with the simplest kind of tasks, namely computational tasks which can be represented as a transformation of input domain elements into output domain elements. We give a first shot at attempting to define some terminologies more precisely for functional task organizations:

- **Task Complexity:** The space and time complexity defined for functions in Computer Science.
- **Task Dynamics:** The distribution of multiple tasks of different characteristics over time. Alternatively, it may mean how the attributes of a given task change over time.
- **Task Uncertainty:** The number of different possible interpretations of the task given the specification of the task.

Environment Model

The world we consider consists of objects, agents, events, and actions. Agents are considered as a special class of objects. Actions are considered as a special class of events. All objects and events are located in some space and time. Since agents are also located in space, in order for an agent to perform a task which requires some input object, there is an access time involved usually proportional to the distance between the agent and the object. Events are changes to attribute-values of the objects in the world as well as the creation and destruction of objects/agents. Events can be divided into two classes: *natural* and *artificial*. Natural events are caused by nature, that is by the nature of the objects in the world. Artificial events are caused by agents and usually called actions.

As in [Shoham, 1993], we adopt the view that what is called an agent is a particular way of seeing an object such that we ascribe agenthood to objects. However, the precise meaning of agenthood, then must be defined such as having memory and computational and communicational capabilities. This may well vary from one domain to another. We note that some agents modeled may not be members of the organization.

When the environment of the organization consists of other organizations, the other organizations must be modeled as a special kind of object as well. The internal model of organization of interest usually will not be useful for characterizing external organizations since the internal model consists of organizational factors that are relevant to the performance of the organization of interest, whereas when modeling external organizations, we are not generally interested in aspects of the external organization relevant to the performance of the *external* organizations but those relevant to the organization of interest. Moreover, the internal factors that determine the performance of an organization may be different across different organizations.

As we have done for our task model, here we give a first pass at defining some terms more precisely for our environment or world consisting of objects, agents, events, and actions:

- **Environmental Complexity:** The number and kinds of objects and events and their relations.
- **Environmental Dynamics:** The model of change of object attribute values across time. This may include birth and death rate of objects of certain class.
- **Environmental Uncertainty:** The uncertainty in object attribute values and relations among them, and the uncertainty of object/relation existence. Alternatively, it may be defined as the number of different possible interpretations of the world. It can also be characterized as the inverse of environmental information available.

Performance Model

The following are the potential performance metrics we initially consider.

1. *Response Time* is the total time taken to accomplish a task. It is also called the turnaround time.
2. *Throughput* is the number of tasks accomplished per unit time. Without a definition of a unit task this measure is ill-defined.
3. *System Utilization* is the fraction of the total system capacity being used at any given time. For a given resource, it is the fraction of time the resource is busy.
4. *Communication Cost* is the cost of transmitting a number of bits across the channel. If time is used as the cost, it may include the connection time plus the time to transmit a number of messages across the channel. Alternatively, the number of bits or message packets transmitted across the communication channel may be used as a measure for communication cost.
5. *Reliability* refers to the probability that the system or a component under consideration does not experience any failures in a given time interval. It is typically used to describe systems that cannot be repaired (as in space-based computers), or where the operation of the system is so critical that no downtime for repair can be tolerated. When a system is composed of multiple subsystems and/or components, the reliability of each component can be used to evaluate the reliability of the total system. By using redundant components, the system reliability can be improved. [1]

[1] The concept of reliability for distributed systems, however, is a little tricky since they may involve redundant

6. *Availability* refers to the probability that the system is operational according to its specification at a given point in time. Availability can be used as some measure of "goodness" for those systems that can be repaired and which can be out of service for short periods of time during repair. [2]

7. *Solution Quality* refers to some objective measure of the quality of task results defined for the particular task domain.

Multiagent Computational Task

Here we define a class of tasks we call *multiagent computational tasks*. The motivation for such a definition is to provide a model of task and its environment which is general enough to apply to many different concrete domains but specific enough to get practical knowledge by using the model to get analytical results and mapping them to many different concrete domains.

The task is for a set of agents located in space to compute a complex function given the initial set of data objects which are also scattered around in space. The function to be computed by the agents are assumed to be decomposable into smaller subfunctions until at the bottom-most level, primitive functions are computed using data objects. However, we assume that there may be many different decompositions of the same function, possibly using different sets of subfunctions or applying them in different orders.

Since there is some distance between an object and an agent, it takes time for an agent to access an object. Objects are assumed to be unique. That is, an object may only be located in one place at one time. However, objects are assumed to be transportable among agents.

The capability of an agent is the set of functions it possesses. It is assumed that, given a function to compute, the set of agents assigned the task are jointly capable of computing the whole function. More specifically, the set of functions required for the task is a subset of the union of the set of functions possessed by the agents. We allow agents to have overlapping capabilities.

Given the above setting, we can formulate organization design problems of different complexities depending on how much is assumed to be given and how much is considered as unknowns to be instantiated. The following are the components that compose a computational organization design problem for the above mentioned multiagent computational task environment:

- The distribution of objects in space.
- The distribution of agents in space.
- The distribution of capabilities among agents.
- The decomposition structure of the task.
- The allocation of tasks and subtasks to agents.

The following are three computational organization design problems of increasing complexity:

1. Given a task, its decomposition structure, an initial distribution of objects and agents in space, and the capabilities of agents, find the best allocation of subtasks to the agents.
2. Given a task, an initial distribution of objects and agents in space, and the capabilities of agents, find the best decomposition of the task and the best allocation of subtasks to the agents.
3. Given a task, and an initial distribution of objects and agents in space, find the best decomposition of the task, the best distribution of capabilities among the agents, and the best allocation of subtasks to the agents.

Multiagent Addition Task using Tree Hierarchy

In our previous work, we have solved a simple version of the computational organization design problem for a multiagent addition task using hierarchical tree structures. Given the task of adding N numbers, the problem was to find the best decomposition of the addition task for tree-like hierarchical organizations. Initially, it was assumed that all the input objects were located at one source agent, and they were to be distributed down the hierarchy, and partial sum results passed bottom-up. It was assumed that all agents had the same capability, namely the capability to add numbers. It was also assumed that processing speed of every agent in the hierarchy were identical, communication delay between any two agents was assumed to be constant, and subtasks were distributed to subordinates in a sequential manner. The problem was to find the best decomposition of the task into subtasks.

The task-environmental factors modeled were the size of the task and the *task environment granularity*, defined as the ratio between unit task execution time and the unit message transmission time.

The organizational factor modeled was the structure of the hierarchy in terms of number of levels and number of branches at each level.

The performance metric used was the response time or the time to complete the entire task.

We have shown that the relative speed of the processors compared to the speed of communication links (i.e. task environment granularity) is an important factor determining the best organizational structure. In particular, we have shown that, for large task sizes binary trees are better than the single level trees, but for certain ranges of task sizes and certain task environment granularities, one-level trees outperform binary trees.

components for improving reliability, and therefore failure of some part of the overall system may degrade the performance of the system along other dimensions (e.g. response time) rather than making the entire system fail.

[2] For example, an organization capable of OSD may be out of service while undergoing OSD.

Organization	4-level binary tree	2-level 4-ary tree	one-level 16-ary tree
$\gamma > 1$	$8\delta + 10\tau$	$4\delta + 10\tau$	$2\delta + 18\tau$
$0 < \gamma \leq 1$	$12\delta + 6\tau$	$10\delta + 4\tau$	$17\delta + 3\tau$

Table 1: Performance of Tree Organizations for $N = 32$ Addition Task.

Our previous work only dealt with binary trees and single level k-ary tree organizations. We have made further progress into generalizing our work into arbitrary k-ary tree organizations and for arbitrary trees in general. The performance function for a k-ary tree could be expressed in algebraic terms. However, for arbitrary trees with possibly varying branching factors at each node, we had to come up with a recursive performance evaluation function. Due to space limitation, we could not present the full recursive algorithm to compute the total execution time of an arbitrary tree. But we give the performance function for using k-ary tree to add N numbers under task environment granularity γ with leaf nodes assigned the task of adding m numbers. We denote the total time to accomplish such task by $T(N, k, \gamma, m)$.

Definition 1 *Let $\gamma = \frac{\tau}{\delta}$ where τ is the unit task execution time, and δ is the unit message transmission time. We call γ the* task environment granularity.

In the addition task, τ is the time to add one number, and δ is the time to send any message from one agent to another assuming that every message is of same size.

$$\begin{aligned} T(N,k,\gamma,m) &= \chi_{\gamma\leq 1}(\gamma)\cdot\{(k+1)l\delta + (l+m)\tau\} \\ &+ \chi_{\gamma>1}(\gamma)\cdot\{2l\delta + (kl+m)\tau\} \end{aligned}$$

where

$$l = log_k(\frac{N}{m})$$

$$\chi_{\gamma\leq 1}(\gamma) = \begin{cases} 1, & if \gamma \leq 1 \\ 0, & otherwise \end{cases}$$

$$\chi_{\gamma>1}(\gamma) = \begin{cases} 1, & if \gamma > 1 \\ 0, & otherwise \end{cases}$$

For k-ary trees, we have found that there exists a branching factor k which is larger than 2 but smaller than the branching factor for a single level tree such that the k-ary tree outperforms both the binary tree and the single level tree.

For example, for addition task of size 32 (i.e. $N = 32$) and $m = 2$, the 2-level 4-ary tree organization outperforms both 4-level binary tree and the one-level 16-ary tree organization *for any task environment granularity* γ. This is interesting since this means that the 4-ary organization is better than the binary and the one-level organization no matter how the processing speeds of the nodes (i.e. τ) and/or the communication delays between the nodes (i.e. δ) change (albeit the assumption that τs and δs are uniform). Table 1 shows the results derived from applying the above equation to the case where $N = 32$, $m = 2$, and $k = 2, 4, 16$. It can be verified that $\forall\gamma$,

$$\begin{aligned} T(32,4,\gamma,2) &\leq T(32,2,\gamma,2), \\ T(32,4,\gamma,2) &\leq T(32,16,\gamma,2). \end{aligned}$$

Organization Optimization using Genetic Algorithm

Since the entire ODS for an arbitrary tree organization for even a simple task like addition is large (i.e. growing exponentially to the size of the task), we have tried using a genetic algorithm (GA) to search through the ODS. That is, by generating a population of random trees and defining valid genetic operators, and applying the genetic algorithm to breed better performing tree organizations, we hoped to find the optimal tree organization for a given task size and a given task environment granularity. We have used a somewhat similar approach to Koza's Genetic Programming [Koza, 1992]. Like Koza's LISP program, our organizations are tree structured, and therefore internal nodes of the tree can be used as crossover points and mutation points. Our tentative results show that, although such a method for searching through the ODS gives results close to the optimal solution, it is very hard to converge on the optimal solution. It seems that unless there are a variety of trees with many different branching factors at the upper levels of the trees in the initial population, GA will more likely miss an optimal solution because there are many more crossover points near the bottom of the tree than in the upper levels of the tree.

Conclusion and Open Problems

We believe that an interesting class of computational organization design problems can be solved by means of the design process model described in this paper. That is, by characterizing computational organization design process as a search through a definite organization design space with definite performance metrics.

In order for such a method to be more widely applicable, we need more precise concepts and terminologies for characterizing the ontologies relevant to organization performance. We have shown our starting point, where we attempt to define concepts which organization scientists have been thinking as important to organizational performance such as complexity, dynamics,

and uncertainty of the task and the environment in computationally precise terms. However, much more research is needed for enriching and refining the various parts of the model.

One open question we would like to address before ending is the following. How is organization design different from organization adaptation? More precisely are we to see organization design as an organization adaptation process or see organization adaptation process as a behavioral feature of an organization to be considered in the organization design process?

The problem of designing ways to redesign the organization in various conditions may well be called the *meta-organization design problem.* How is it related to the base-level organization design problem? If any agent is changing its behavior and relation to others, is it engaged in local organization redesign or just working under the legitimate guidelines for changing ones behavior specified by the meta-organization?

References

Daniel D. Corkill and Victor R. Lesser. The use of meta-level control for coordination in a distributed problem solving network. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 748–756, Karlsruhe, Federal Republic of Germany, August 1983. (Also appeared in *Computer Architectures for Artificial Intelligence Applications*, Benjamin W. Wah and G.-J. Li, editors, IEEE Computer Society Press, pages 507–515, 1986).

Daniel David Corkill. *A Framework for Organizational Self-Design in Distributed Problem Solving Networks.* PhD thesis, University of Massachusetts, February 1983. (Also published as Technical Report 82-33, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, December 1982.).

Les Gasser and Toru Ishida. A dynamic organizational architecture for adaptive problem solving. In *Proceedings of the National Conference on Artificial Intelligence*, pages 185–190, July 1991.

John R. Koza. *Genetic Programming.* MIT Press, 1992.

Yoav Shoham. Agent-oriented programming. *Artificial Intelligence*, 60:51–92, 1993.

Young-pa So and Edmund H. Durfee. An organizational self-design model for organizational change. July 1993. (Presented at the AAAI-93 AI and Theories of Groups and Organizations Workshop, Washington, D.C., July 11, 1993.).