

The Effects of Runtime Coordination Strategies Within Static Organizations

Edmund H. Durfee and Young-pa So

AI Laboratory, EECS Department
University of Michigan,
Ann Arbor, MI 48109, USA
durfee@umich.edu

Abstract

Much work in multiagent systems has been devoted to developing runtime mechanisms to coordinate computational agents. However, the efficacy of such mechanisms depends to a large extent on the opportunities for flexible choices of action imposed by the implicit or explicit organizational relationships among agents. This paper examines the dependencies between the design of static organizational structures and the efficacy of runtime coordination. We show that runtime coordination strategies can be viewed as transforming the configuration of an organization, with principled effects on the expected performance of the agent system. By understanding these effects in the context of organizations, we can begin to articulate the conditions under which the introduction of coordination mechanisms (and their associated costs) can be justified.*

1 Introduction

The debate over the efficacy of runtime coordination mechanisms has in many ways revisited the deliberative versus reactive debate of the planning community. At one extreme are researchers who consider runtime coordination time-consuming and unnecessary. By instituting enough structure in the system, the local decisions of agents are assured of being compatible. Social Laws [Shoham and Tennenholtz, 1992] are one example of such static structuring mechanisms. At the other extreme are researchers who strive toward endowing each agent in a complex system with the ability to dynamically maintain models of other agents and to use these models to make arbitrarily complex coordination decisions interleaved with execution (e.g. [Gmytrasiewicz *et al.*, 1991]).

Between these extremes are the many techniques for coordinating agents within the bounds of some fixed commitments upon which they can all depend (e.g. [Decker, 1995; Durfee *et al.*, 1987; Gasser *et al.*, 1989]). But these techniques have generally been developed to meet the needs of particular applications. While many have been successful, it has seldom been the case that they are found to be easily

transferred and effectively employed across many different application domains.

Why do runtime coordination mechanisms sometimes make a big difference, and sometimes have little impact? We argue that runtime coordination is situated contextually in ways that affect its impact. Simply, if an agent is placed in a world where it faces no indecision, then coordinating with others at runtime is unlikely to help it perform better. On the other hand, if it faces several choices, and making a good choice depends on knowing what others around it are doing, then runtime coordination is a major help.

The context within which an agent—especially a computational agent—works is controllable to a large extent. In particular, in multiagent systems, the designers of a system dictate many of the features of the context in which an agent will be operating. Of particular interest in this paper, for example, are the relationships between agents' roles that are defined by the authority structure and the inherent structure of the distributed tasks that they pursue. We call this role structure the **organization** within which the agents work.

We are investigating the relationship between static organizations (how roles are defined prior to execution) and runtime coordination (how agents choose to occupy and fulfill roles during execution). Our ultimate goals include being able to predict what kinds of coordination mechanisms agents should possess to be effective in a particular organization, and to provide design advice for the creation of organizations that are suited for agents with known coordination capabilities. In this paper, we describe steps toward this end. We provide a parametric characterization of a class of organizations that allows us to systematically generate and evaluate alternatives, and describe performance measures for comparison. We describe the effects that different basic coordination strategies have in different organizations under different operating conditions. With these observations, we can distill some general lessons in designing agents and organizations that they inhabit.

2 Organization Structure

We will confine our discussion to the *multi-level hierarchy* class of organization structures. Hierarchies capture many of the most common notions of organization, where author-

* This work was supported in part by NSF PYI award 91-58473.

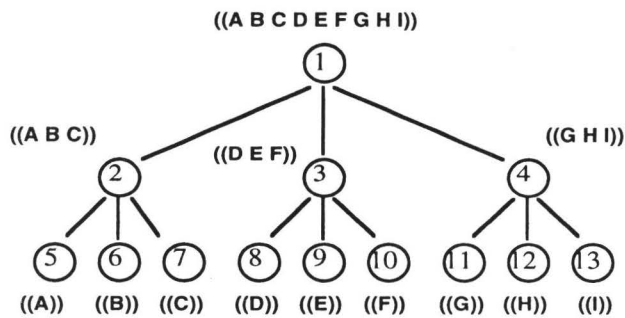


Figure 1

ity and breadth of perspective are concentrated in managers at higher levels of the organization, and tasks are decomposed and distributed down to the lowest level members. For example, this is the form of organization that emerges from the use of contracting protocols [Smith, 1980].

We characterize these structures in terms of the number of (primitive or supervisory) tasks assigned to members, and the branching factor of the organizational tree. For simplicity, we assume in this paper that these are constant in the structure (the organizational structure is uniformly branchy and the tasks are evenly distributed). For example, in Figure 1 is an organizational structure with a branching factor of 3 and that assumes that the "leaf" agents should perform one task (labeled alphabetically). Therefore, a middle manager's task is to supervise the performance of three tasks below it, and the top-level manager supervises three such supervisory tasks, meaning that it is indirectly overseeing 9 primitive tasks (A-I). The leaf-level agents complete their tasks, pass the results up to the middle managers, who synthesize them into a single result to pass to the top manager, who then generates from these the complete result. The solution time depends on the time required for each unit task and synthesis task (we assume the same values for these) and for the communication delays in distributing the sub-tasks and returning the results.

When uncertainty exists about the rate at which tasks will be achieved, imbalances of effort in the organization can degrade performance. A countermeasure is to give members of an organization overlapping responsibilities (tasks) such that if one is delayed in fulfilling its responsibilities (such as if it catastrophically fails), others can pick up the slack [Durfee *et al.*, 1987]. For example, to make the previous organization more robust to agent failures, we could redundantly assign supervisory and primitive tasks among the agents. Note that the top-level agent is still a single point of failure, but we can do assignments beneath it such that the organization (or any subtree of it) can be assured of tolerating a certain number of failures among those being supervised. In the example of Figure 2, the redundant task assignments ensure that a (sub)tree succeeds so long as no more than one of the branches of that (sub)tree fail. A particular failure configuration is shown. The notation ((AB)(DE)) means the agent has 2 task sets, each with 2 tasks. In some coordination schemes, the first task set might be the (possibly prioritized) primary tasks, and the second might be secondary tasks (since they are another agent's primary tasks).

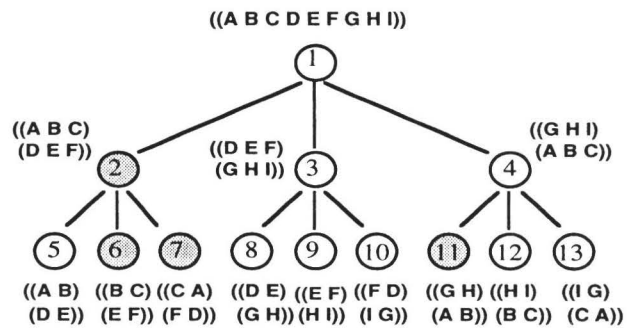


Figure 2

As has been discovered in other organizational approaches to cooperative problem solving, however, the reliability comes at the potential cost of duplicated effort [Durfee *et al.*, 1987]. So, in cases where agents do not fail, they should coordinate their choices of which of their assigned tasks to each do to avoid redundancy. And despite the built-in reliability, if too many (or the wrong combination of) agents fail, the organization will still fail.

Runtime coordination, therefore, should be employed within these organizations to overcome these problems. When agents do not fail, runtime coordination should ensure that they are working on complementary tasks. When agents do fail, runtime coordination should ensure that the remaining agents are applied most effectively. Of course, if the organization involves no redundancy, then the first reason for runtime coordination will never arise; if the organization is completely redundant (all agents have all tasks), then the second reason will never arise. Thus the effects of runtime coordination mechanisms are highly dependent on the underlying organization structure among the agents.

3 Performance Measures

To study the effects of runtime coordination mechanisms in the context of organizational structures, we need to be more precise about what features of the performance we will measure to assess the effects. To keep our discussion simple, we will consider two measures: the response time of an organization (how quickly it completes the task set) and the reliability of the organization (how probable it is that it will successfully complete the entire task set).

3.1 Response Time

The response time of an organization is the time it takes for the agents to decompose and distribute the tasks to subordinates (who then decompose and distribute, and so on), for the agents doing the unit tasks to do them all, and for the results to be passed up and synthesized. We will omit the details of the calculation of the response time in this paper; we use the model described by [So and Durfee, 1997] where we are additionally assuming that the communication delay is equivalent to the unit task execution time. Thus, we characterize the response time of the organization when it is successful as $T_s(\gamma, N, k, m, o, f, s)$, where:

- γ is the ratio of unit task execution time to communication delay (in this paper, $\gamma=1$)

- N is the size of the overall task set
- k is the organization's branching factor
- m is the number of unit tasks per leaf role
- o is the overlap, representing how many subtrees can fail while still being assured of success (note that $o < k$)
- f is the probability of failure for each agent (we assume agents fail independently)
- s is the coordination strategy outlined below

Of course, given the possibility of agent failures, the performance will depend on which agents do and do not fail. We refer to a particular arrangement of working (and non-working) agents as a **configuration**. The expected response time for successful configurations is thus the weighted (by probability that a configuration will occur given f) average of the response times of successful configurations. The response time of a successful configuration is itself generally an expected value, since a particular configuration will behave differently depending on the order in which redundantly assigned tasks are pursued.

As an example, for the organization in Figure 2, one configuration is where agents 2, 6, 7, and 11 have failed. Despite these failures, eventually all of the unit tasks will be done by agents that can supply results back to the top (by agents 8, 9, 10, 12, and 13). But note that the task (set) ordering represented in the figure means that task A will be done last of all (by agent 13). A reordering by 13 could decrease the delay for A (which only 13 can accomplish); in fact, agents 3 and 4 could reorder their subtask sets such that agent 4 would prioritize ABC over GHI, and this would yield better performance. Given random orderings, therefore, we will have a distribution of response times for this configuration, as shown in Figure 3.

To get the distribution of response times for the organization as a whole, we combine the distributions of the different configurations, weighted by their probabilities. This leads to another distribution, from which we can compute best-case, worst-case, and expected-case response times for the organization, *when it is successful*, given f (Figure 4).

3.2 Reliability

Of course, if very unlikely configurations are the only ones that ever succeed, the organization is not a very good one no matter what its expected response time is conditioned on success. We can compute the probability of success by simply summing up the probabilities for the successful configurations. The probability of not succeeding (failing) is one minus the success probability, and for larger values of f

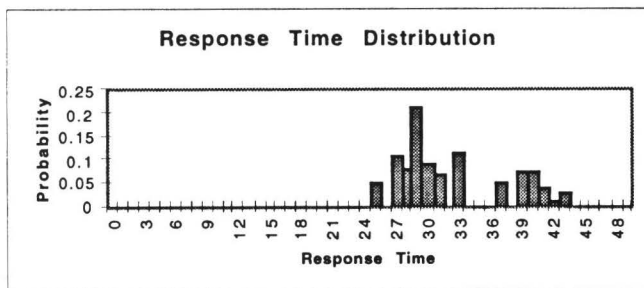


Figure 3

the failure probability can be substantial! We can thus evaluate organizations on likelihood of success independently of response time.

3.3 Overall Performance

Ultimately, to compare organizations with each other, we need a performance measure that combines the relevant features. In our case, we need to combine reliability and response time. Our way of doing this makes several assumptions that allow us to cast unreliability as delay in the achievement of goals. Simply put, our performance measure assumes that when an organization fails, it tries again and again until it succeeds. Thus, the organization times out when the worst-case response time across all successful configurations has elapsed. It then restarts the process, assuming that a new random configuration of the organization will be manifested, which might allow success. If the worst-case response time goes by again, it tries again, and so on until it finally succeeds.

Graphically, this means that the organization's response time distribution (Figure 4) is repeated over and over again, each time with a lower amplitude, such that the areas under all of the distributions sum to one (the area of the first distribution is the probability of success, of the second is the probability of failure times probability of success, of the third is the probability of failing twice before success, and so on). This is shown in Figure 5. We truncate this pattern when the remaining area is negligible, and we can compute the expected response time from this overall distribution.

4 Effects of Runtime Coordination

Given this distribution over performance, we can now be more precise about what runtime coordination is intended to do. Put simply, runtime coordination should move this distribution toward faster response times. There are two obvious ways that the distribution can be "moved" like this. One would be to increase the areas of the "humps" at faster response times, which means increasing the probability of success for any given execution of the organization. The other is to shift the expected value of the initial distribution to a faster time. That is, making the response time when successful faster will shift each of the repeated "humps" to

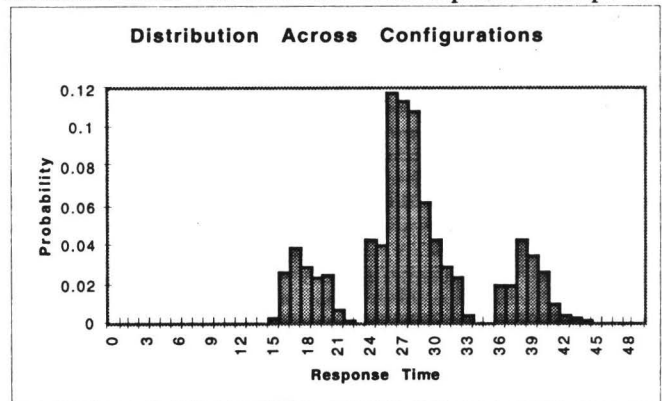


Figure 4



Figure 5

have faster expected response times, which will lead to an overall faster expected response time.

Thus, we are able to say something about the *potential* effects of runtime coordination without having to be specific about the mechanisms for realizing the coordination. Instead, we can characterize runtime coordination mechanisms in terms of their effects on the performance distributions, rather than by implementation details of the mechanisms. More substantially, as we shall see, we can talk about whether runtime coordination mechanisms that affect the distributions in particular ways would be expected to benefit particular organizational structures.

5 Coordination Strategies

In this paper, we will consider two main coordination strategies, corresponding to the transformations on response time distributions previously mentioned. Again, while the detailed design and implementation of these strategies are certainly of concern (as we discuss later), our goal here is to ascertain whether the runtime coordination through either (or both) of these strategies is potentially worthwhile however the strategies are implemented.

5.1 Role Reallocation

A **role** corresponds to a position in the tree structure of the organization. Agents occupy roles. If an agent fails, then the role that the agent plays in the organization is not filled. As we saw with our example in Figure 2, loss of roles affects performance. In fact, for particular configurations, the lost roles can render the organization ineffective.

Role reallocation is a runtime strategy whereby agents can reassign roles among themselves when they discover that certain critical roles are unfilled. Note that we assume that it is still the case that an agent can only occupy one role. But an agent can abandon a less important role to take on the responsibilities of a fallen comrade. For example, in Figure 2, agent 5 serves no useful purpose since the synthesis pathway for its results is blocked by agent 2's failure. 5 could therefore take on the role of failed agent 11, which (given a fortunate ordering of when tasks are done) can lead to faster organization response times than had it not done so.

More importantly, role reallocation (RR) can overcome agent failures that cause an organization to be unsuccessful. This strategy in principle allows agents at runtime to transform the current configuration (with some agents working and some failed) into any other configuration with the same number of working agents. Thus, they can transform the configuration into the best configuration with that many failed agents. Of course, this implies that the agents not only are able to recognize the current configuration, but can identify and agree on a better configuration. Examples of

reconfiguration mechanisms exist in the multiagent systems literature (e.g. [Smith, 1980; So and Durfee, 1992]). Our investigation asks what kinds of organizations and environments could exploit agents that can do this.

5.2 Local Task Reordering

The second coordination strategy we look at here focuses on the local decisions agents make about which of their tasks to do next. This problem appears in many distributed applications, ranging from job-shop scheduling to distributed operating systems, with many approaches to supporting this kind of runtime coordination having been identified (e.g. [Decker, 1995; Liu and Sycara, 1996]). In general, task interdependencies and uncertainties about execution time complicate the ordering/scheduling of tasks in distributed systems.

In the context of our analysis, local task reordering (LTR) is a coordination strategy that lets agents take actions that complement those of other agents. In the simple scenarios we have outline here, this amounts to doing tasks that others will not have already done first, although our results should apply to more complex task relationships as well. Viewed as a transformation on the organization, LTR transforms a particular configuration into an instance of the same configuration but with the best possible orderings over the tasks of the agents. That is, if agents know which of them have failed, and thus which of the redundantly assigned tasks are now mandatory, they could reorder their tasks more optimally (as illustrated in section 3.1). This shifts the distribution of expected response times toward the faster end.

5.3 Combined Strategies

The two strategies outlined above are not mutually exclusive. At runtime, agents might be able to identify the configuration of empty roles (due to agent failures), and could both reassign roles among themselves and, within those roles, find the best local task reorderings. Both of these factors would be brought to bear on the coordination decisions: the agents should transform their configuration into the configuration (with the same number of failed agents) that has the best expected performance for the best ordering (compared to the RR strategy alone which transforms the configuration to the one with the best expected performance given random orderings).

5.4 Other Strategies

While the strategies outlined above cover the predominant runtime coordination mechanisms in the literature, they are not the only possible ones. Even better runtime coordination might occur, for example, if agents can redefine roles (swap tasks) or adopt multiple roles. However, as strategies like this begin to emerge, the notion of a "static" organizational structure begins to blur, since redefining roles, or dynamically creating new roles from merged pieces, amounts to runtime organizational redesign [Ishida *et al.*, 1992]. Organizational redesign by agents is an important runtime strategy, but is beyond the scope of this paper.

6 Empirical Observations

By defining in a precise way a class of organizations, a set of performance measures, and the transformations on an organization caused by runtime coordination mechanisms, we can begin to understand the multiagent system design problem in integrated terms. To motivate this discussion, though, let us first look at some experimental results to illustrate some trends we see in studying the effects of runtime coordination in the context of organizational structures.

To keep things simple, the experiments we outline here are based on two organizations. One is a simple variation of Figures 1 and 2, where instead the organization was given $N=36$ tasks, and each "leaf" agent did 4 tasks ($m=4$). Thus, there were still 13 roles ($r=13$), with a branching factor $k=3$. The other organization has a branching factor $k=2$ and each "leaf" does $m=9$ of the 36 tasks so there are $r=7$ roles. In all cases, we assume $\gamma=1$.

In these experiments, we vary four parameters: the organization (as captured by the number of roles r), the organization's role overlap (o), the agent failure probability (f), and the coordination strategy (s) of the agents. The role overlap ranges from 0 to 2, the failure probability ranges from 0 up to 0.7, and the coordination strategy is one of four values: RR, LTR, BOTH, or NONE. (When there is no overlap, ALL are the same.) To gather the data, we ran each experiment that involved randomization at least 100 times. The results are summarized in Table 1. We have also experimented with many other organization structures, and the trends we emphasize in this paper appear to generalize.

6.1 Organization Structure

To begin with, the designer of an organization within the limited class we are considering can choose an organization size based on the branching factor and task sizes at the leaves. Not unexpectedly, an organization of 13 agents can solve the problem faster than an organization of 7 agents, so long as all agents are available. Our experiments show,

Table 1

	r	o	$f=0$	$f=.1$	$f=.2$	$f=.3$	$f=.4$	$f=.5$
ALL	13	0	14	176	4391	10^6		
NONE	13	1	20.2	26.2	44.5	120	534	3950
LTR	13	1	14	21.6	40.9	117	532	3948
RR	13	1	20.2	20.5	21.3	23.7	32.3	74.7
BOTH	13	1	14	15.6	17.1	20.3	29.6	72.5
NONE	13	2	25.3	33.7	45.5	65.2	104	198
LTR	13	2	15	24.4	37.6	58.6	98.6	194
RR	13	2	25.3	26.0	26.7	27.7	29.4	32.2
BOTH	13	2	15	16.2	17.9	19.9	22.6	26.6
ALL	9	0	17	55.8	323	2317	10^5	10^6
NONE	9	1	28.6	37.7	52.7	82.9	156	369
LTR	9	1	17	29.1	46.2	77.9	152	366
RR	9	1	28.6	30.1	32.2	35.4	41.3	54.4
BOTH	9	1	17	21.2	25.0	29.5	36.6	50.9

however, that a smaller organization is inherently more robust, because the chances of any agent failing are smaller (Figure 6, $o=0$ cases). A simple change to the organization structure of introducing redundancy (o greater than 0) can compensate at lower failure rates, and larger values of o postpones organizational failures to higher values of f , but at a cost of being less efficient at lower f values (Figure 6). This figure shows that at high f , a 13-role organization with $o=2$ is best (this can still succeed even losing 10 of 13 agents), but is inferior to the same organization with $o=1$ at low values of f . The 7-role organization with $o=1$ is better than the 13-role organization with $o=1$ at high f , needing only 3 of 7 working agents compared to 7 of 13.

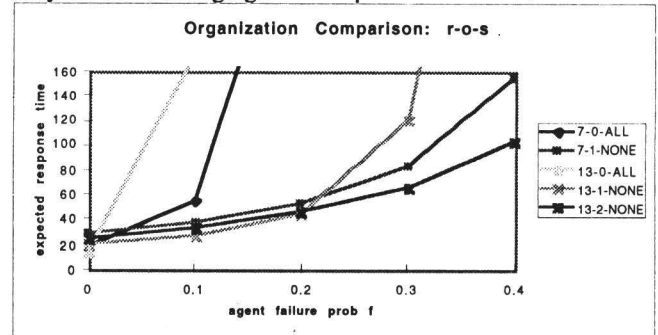


Figure 6

6.2 Runtime Coordination

Introducing redundancy illustrates a trend that we have recognized in all our other experiments, and that has intuitive appeal. Under low failure rates, what drags down response time is poor task selection, leading to duplication of effort on some tasks and delays in attending to others. Low failure rates mean that the redundant assignment seldom is needed, so the inefficiencies are incurred without the benefits unless an LTR strategy is employed. For $r=13$ and $o=2$ (Figure 7), LTR can cut response time by about 40%. As failure rates increase, two things happen. First, there is less redundant activity because fewer agents with redundant tasks are active. Second, the dominant influence on response time is the repeated retrying of the problem. Thus, LTR has decreasing impact, while RR's effect becomes more pronounced (Figure 7) since, even for configurations for which the number of failed agents is high, there is often a very

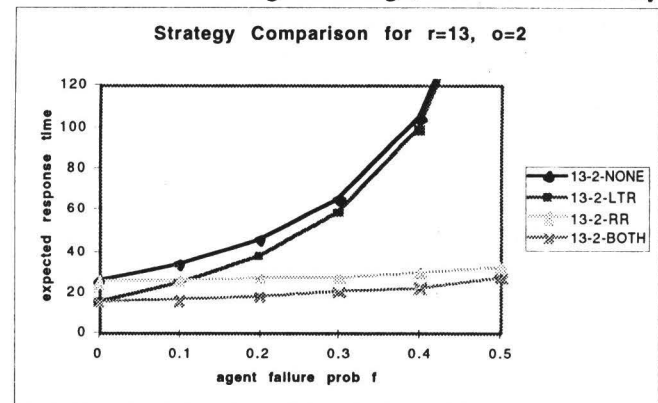


Figure 7

small number of configurations with the same number of failed agents where the organization can succeed. While the chances of randomly being in such configurations are small, the RR strategy allows these to be found and exploited.

6.3 Runtime Coordination and Role Overlap

Another important observation is that the coordination strategies are less effective as the role overlap decreases. That is, when the role overlap approaches 0 (no redundant task assignments), there is less opportunity for duplication of effort. The distribution for a single configuration (Figure 3) approaches a single spike, and the number of successful configurations (Figure 4) approaches the single "no failure" configuration, so LTR is less useful. In our data, LTR reduces runtime by about 40% with $o=2$ (for $r=13$), but only reduces runtime by about 30% with $o=1$, and of course has no effect at all for $o=0$. And, with smaller overlap, tolerance of *any* failure is reduced, so RR has no alternative configuration to move to. Thus, the efficacy of the coordination strategies not only varies with environmental factors like failure rate, but also the efficacy varies because strategies can only coordinate within the boundaries specified by the organizational structure. Structures that give less opportunity for agents to adapt to runtime conditions are less affected by runtime coordination mechanisms.

6.4 Runtime Coordination Costs

Of course, all of the preceding considered the potential advantages of runtime coordination without considering its costs. When there are no potential advantages, we can avoid introducing runtime coordination mechanisms that would not help even if they were overhead free.

More generally, when we accept that runtime coordination requires time (and other resources) that is taken away from performing tasks, then we need to trade off the costs and benefits of the runtime mechanisms. A thorough exploration of the costs is one of our future tasks. Here, we give a brief flavor of the kinds of results we are working toward.

A challenge in comparing coordination mechanisms is that the quality (or lack thereof) of implementation can obscure design comparisons. We sidestep this by introducing parametric models of coordination mechanisms, based on the parameters we have already described, such that we can assess performance under different assumptions about mechanism efficiency. For example, we might consider the cost of LTR at any given time to be dependent on the size of the space of possible collective orderings, which can be approximated by: $(m(o+1))^a$, where a is the number of active agents in the organization. Note as more agents fail, the cost of LTR decreases dramatically (along with effective role overlap). The cost of RR might simply be based on periodic communication between all of the active agents. One simplistic model of coordination cost, therefore, is that it adds to the runtime a factor proportional to the actual runtime (assuming longer runs need more coordination):

$$\Delta RT = [\alpha * (m(o+1))^a + \beta * a^2] * RT$$

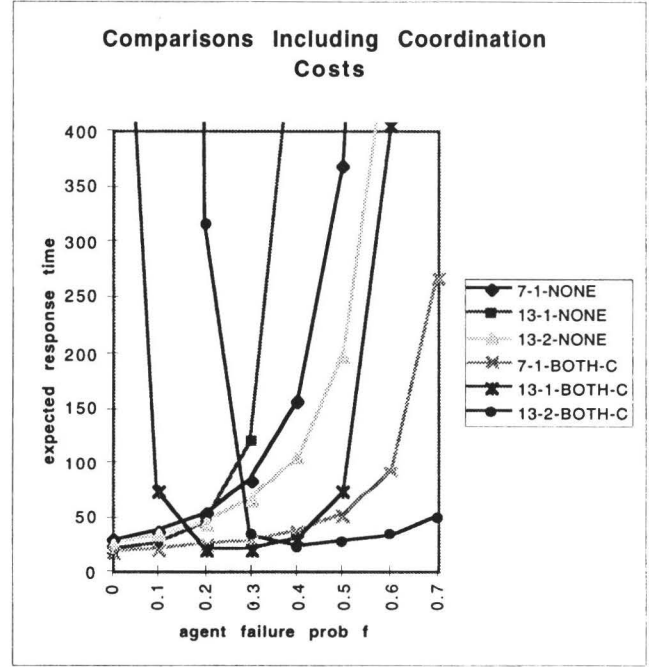


Figure 8

where α and β are constants capturing in a gross form the quality of the coordination algorithm design and implementation. α and β are typically small relative to task execution time, and for purposes of illustration we will assume assessing a single collective ordering is very fast ($\alpha=10^{-10}$) and selecting the right configuration is somewhat fast ($\beta=10^{-4}$) compared to the application tasks. For the 7 and 13 role organizations, approximating a as $(1-f)r$, the comparison between using both or neither of the coordination strategies is shown in Figure 8. This illustrates that, for the given parameters, coordination overhead in the smaller (7-role) organization is cost-effective throughout. In the larger 13-role organization, the combinatoric costs swamp the benefits of runtime coordination until higher values of f , where the benefits are more substantial and the costs have decreased in this simple model because fewer "live" agents remain to coordinate. Note that for the 13-role case there is an interval for which less overlap is better than more, because more overlap increases coordination costs as well.

We should again emphasize that this example is simplistic and is intended to be only suggestive, rather than prescriptive. The combinatorics we have outlined for larger organizations have been addressed numerous ways in more practical coordination mechanisms, including reducing the frequency of coordination, coordinating agents in smaller local groups, and searching less of the collective task-ordering space (such as by using hill-climbing) [Durfee and Lesser, 1991]. These relaxations can lead to suboptimal coordination; using the model we have presented here, our research can work toward prescriptions about what relaxations make sense given the costs and the potential benefits for runtime coordination within an organization.

7 Designing Agents and Organizations

In this paper, we have taken initial steps toward characterizing how statically-defined organizational roles and relationships affect the potential utility of making runtime coordination decisions about what roles agents should fill and which of the tasks associated with a role each agent should perform next. From the agent-design perspective, our work emphasizes that building agents that are better at coordinating only makes sense in organizational structures that provide agents with sufficient latitude to exploit these capabilities. Agile organizations that are built to be robust under changing environmental conditions (such as when organization members fail to deliver) provide more opportunities for coordination mechanisms. Organizations that have evolved for stable conditions, however, might benefit little from better coordination mechanisms. In human terms, putting coordination technology on an assembly line would not be expected to help much, given our studies. Improving agent coordination capabilities should be considered in the context of the organization, whether agents are computational or human.

From the organization-design perspective, this tells us that we might well design organizations differently depending on the coordination abilities of the participants in the organization. Better responsiveness and reliability can both be attained with appropriate coordination mechanisms. But agents with such mechanisms should be organized differently for different environments, as we saw in Figure 8, where different organizational structures were better over different failure probability ranges. And, of course, if all we have are simpler agents, then we should tailor the organization to *not* depend on runtime coordination!

Most of our observations have been based on identifying *potential* for improvement. By looking at how runtime coordination might affect performance in the "best" case, we can validate beforehand whether there is sufficient opportunity to make investment in implementing coordination mechanisms for an application, and what class of coordination mechanisms (such as reconfiguration, or reordering, or both) hold the most promise. But we have also acknowledged that coordination comes at a cost, so that even when there are potential benefits from coordination, it might well be the case that the costs of the runtime mechanisms outweigh their benefits. To advance the design of multi-agent systems from an "art" into an engineering discipline, therefore, will require a careful articulation of the various design parameters for organizations and agents, and their impact on the performance measures [Carley and Prietula, 1994]. In this paper, we have taken steps in this direction so that we can begin to make recommendations for the restricted set of organizations outlined here, organizations for multiagent systems that have already been shown to be useful in multiagent applications including distributed sensor interpretation [Durfee and Lesser, 1991] and distributed network monitoring [So and Durfee, 1992].

References

- [Carley and Prietula, 1994] Carley, K. M., and Prietula, M. J., (Eds.), *Computational Organization Theory*, Hillsdale, NJ, Lawrence Earlbaum Associates.
- [Decker, 1995] Decker, K., *Environment Centered Analysis and Design of Coordination Mechanisms*, PhD dissertation, University of Massachusetts.
- [Durfee *et al.*, 1987] Durfee, E., Lesser, V., and Corkill, D., "Coherent cooperation among communicating problem solvers." *IEEE Trans on Computers*, C36(11):1275-1291.
- [Durfee and Lesser, 1991] Durfee, E., and Lesser, V., "Partial Global Planning: A coordination framework for distributed hypothesis formation." *IEEE Trans on Systems, Man, and Cybernetics*, SMC21(5):1167-1183.
- [Gasser *et al.*, 1989] Gasser, L., Rouquette, N. F., Hill, R. W., and Lieb, J., "Representing and Using Organizational Knowledge in Distributed AI Systems." In Gasser and Huhns (eds.) *Distributed Artificial Intelligence II*, San Mateo, CA, Morgan Kaufmann Publishers.
- [Gmytrasiewicz *et al.*, 1991] Gmytrasiewicz, P., Durfee, E., and Wehe, D., A decision-theoretic approach to coordinating multiagent interactions." In *IJCAI-91*, pp 62-68.
- [Ishida *et al.*, 1992] Ishida, T., Gasser, L., and Yokoo, M., Organization self-design of distributed production systems, *IEEE Trans on Knowl and Data Eng DKE4*(2):123-134.
- [Liu and Sycara, 1996] Liu, J. and Sycara, K., Multiagent coordination in tightly coupled task scheduling, *International Conf on Multi-Agent Systems*, pp. 181-188.
- [Shoham and Tennenholtz, 1992] Shoham, Y. and Tennenholtz, M., On the synthesis of useful social laws for artificial agent societies, *AAAI-92*, pages 276-281.
- [Smith, 1980] Smith, R. The contract-net protocol, *IEEE Trans on Computers* C29(12):1104-1113.
- [So and Durfee, 1997] So, Y. and Durfee, E., Designing span-of-control for computational agent organizations, *Journal of Computational and Mathematical Organization Theory* 2(3):219-246.
- [So and Durfee, 1992] So, Y. and Durfee, E., A distributed problem-solving infrastructure for computer network management." *Int Journal of Intelligent and Cooperative Info Sys*, 1(2):363-392.