Sharing Metainformation to Guide Cooperative Search Among Heterogeneous Reusable Agents

Susan E. Lander, *Member, IEEE*, and Victor R. Lesser, *Member, IEEE*

Abstract—A *reusable agent* is a self-contained computational system that implements some specific expertise and that can be embedded into diverse applications requiring that expertise. Systems composed of heterogeneous reusable agents are potentially highly adaptable, maintainable, and affordable, assuming that integration issues such as information sharing, coordination, and conflict management can be effectively addressed. In this article, we investigate sharing metalevel search information to improve system performance, specifically with respect to how sharing affects the quality of solutions and the runtime efficiency of a reusable-agent system. We first give a formal description of shareable metainformation in systems where agents have private knowledge and databases and where agents are specifically intended to be reusable. We then present and analyze experimental results from a mechanical design system for steam condensers that demonstrate performance improvements related to information sharing and assimilation. Finally, we discuss the practical benefits and limitations of information sharing in application systems comprising heterogeneous reusable agents. Issues of pragmatic interest include determining what types of information can realistically be shared and determining when the costs of sharing outweigh the benefits.

Index Terms—Reusable agents, information (knowledge) sharing, distributed search, multiagent systems, system performance, mechanical design, distributed artificial intelligence.

1 Introduction

The computational equivalent to a team of human specialists is the *reusable-agent system*, a multiagent system in which expert agents are dynamically selected from a library and integrated with minimal customized implementation. With reusable-agent systems, diverse information can be applied in situations that were not explicitly anticipated at agent-development time. The benefits of this type of system to an application builder are potentially large: For example, agents can be flexibly and inexpensively modified, added to a system, or deleted from a system in response to changes in specifications, resources, and technology; agents will be more reliable over many uses; and the cost of building an agent can be amortized over multiple uses. Reusable-agent systems have enormous potential in domains such as design engineering and manufacturing where business trends and global competition are forcing very rapid transitions in both technology and philosophy [1]. Complex applications that incorporate reusable agents are able to adapt quickly to changes without discarding the main body of existing work. However, to participate in an application, reusable agents must be technically capable of effective interaction. The challenge goes much deeper than the physical transfer of information between agents or even mutual semantic-level understanding of shared information. To achieve satisfactory performance in reusable-agent systems, significant attention must be paid to what information should be shared and how that information is created and used to reason about the global state of problem solving, to coordinate actions, and to resolve conflicts stemming from inconsistent or incomplete knowledge and evaluation criteria.

Local perspective is one of the most difficult problems to be overcome in any multiagent system; it is especially pronounced in reusable-agent systems because the agents are created independently and without foreknowledge of applications into which they will be embedded. Therefore, knowledge engineering during agent implementation is impossible. Each individual agent perceives the world as defined and bounded by its local knowledge rather than by the knowledge of all agents in the system. This problem could be easily addressed if all expertise were represented in the form of explicit constraints: The constraints could be collected and processed by a centralized constraint-satisfaction algorithm. However, agents are generally complex systems in which expertise is represented by a combination of declarative and procedural knowledge. When this is the case, constraint processing is not sufficient to represent the expertise of an agent. Instead, each agent must be individually invoked to solve subproblems that are relevant to its domain of expertise and the resulting solutions must be integrated into a globally consistent solution. An agent with only a local view of the search space cannot avoid producing subproblem solutions that conflict with other agents' solutions and cannot make intelligent decisions about managing conflicts that do occur. To overcome this

- *S.E. Lander is with Blackboard Technology, 401 Main St., Amherst, MA 01002. E-mail: lander@bbtech.com. Web URL: http://www.bbtech.com/lander.*
- *V. Lesser is with the Department of Computer Science, University of Massachusetts, Amherst, MA 01003. E-mail: lesser@cs.umass.edu.*

Manuscript received Dec. 14, 1995.
For information on obtaining reprints of this article, please send e-mail to: transkde@computer.org, and reference IEEECS Log Number K97011.

problem, information sharing can be used to enhance the local perspectives of agents. A better understanding of the global situation leads to more intelligent and focused search and, ultimately, to higher-quality solutions and more efficient performance.

As noted by Neches et al. [2], software reuse in any form is difficult and is impeded by the lack of tools available to foster information sharing. Concurrent investigations of languages, ontologies, and protocols for agent interaction [3], [4], [5] complement our research but focus on the development of enabling technologies for information sharing rather than on the effect of shared metainformation on problem solving. In this article, the TEAM framework [6] is used as the infrastructure technology to provide the communication and control backbone that enables effective interaction. Rather than addressing issues of enabling technology, the article will focus on how the communication of specific information among agents affects the quality of solutions and the efficiency of a reusable-agent system. Our goal is to show that reusable agents involved in distributed search can improve their joint performance by sharing metainformation with other agents, assimilating shared information from other agents, and using this information to refine their local views of the global solution space.

1.1 Shared Metainformation

Multiagent systems generally assume some form of *result-sharing* [7], i.e., the sharing of partial solutions to facilitate global coherence. In our experiments, result-sharing is an integral part of problem solving—agents are able to examine, evaluate, and extend others' proposed partial solutions. However, our focus is on the sharing of a different type of information; namely, information that describes some abstraction of an agent's solution space rather than a specific solution instance. This will be referred to as *metainformation*.[1]

To illustrate the use of metainformation, consider a generic example of meeting scheduling. (Sen and Durfee investigate the effectiveness of various forms of shared information in a meeting scheduling domain in [8]). Assume two agents, $\mathcal{A}$ and $\mathcal{B}$, are seeking a mutually acceptable meeting time. A simple result-sharing approach would be for $\mathcal{A}$ to examine its local schedule and propose a time based on its constraints: "How about Monday at 3:00?" $\mathcal{B}$ responds with "No good. How about Tuesday at 9:00?" $\mathcal{A}$ responds with "Maybe Monday at 3:30?" and so on until one of the agents suggests a mutually acceptable time. In contrast, the use of metainformation is seen in the following exchange. $\mathcal{A}$ again begins with "How about Monday at 3:00?" but now $\mathcal{B}$ responds: "No good. I'm tied up all day Monday. I've got some time Tuesday morning though, and I'm free most of Thursday." As seen in this example, the goal of sharing metainformation is not to pinpoint a specific solution, but rather to provide focusing information that will guide other agents in their search for a solution.

1. In the following text, *information* and *metainformation* are used interchangeably where it is clear from the context that this is the case.

1.2 Distributed Search

We explore the use of shared metainformation using a prototype application system, STEAM, that represents a class of cooperative distributed search systems for parametric design. Search systems are historically described in the Artificial Intelligence literature as comprising three components: a *state space* describing the current state of the search, a set of *operators* used to manipulate the state space, and a *control strategy* used for deciding what to do next, specifically, deciding what operator to apply and where to apply it [9]. When all operators reside in a single program or logical entity and have access to a central store of knowledge and databases, the search is centralized. In this article, we are concerned with the problem of distributed search as described in [10]:

> *A distributed search involves partitioning the state space and its associated operators and control regime so that multiple processing elements can simultaneously perform local searches on different parts of the state space; the (intermediate) results of the local searches are shared in some form so that the desired answer is produced in a timely manner.*

The partitioning of the state space in a reusable-agent system is induced by the a priori division of expertise of agents in the agent set. The set of operators available at an agent is also an a priori attribute of the agent. The control strategy used for a solving a particular problem should be tailored to the problem but must be chosen from the set of strategies known to the agents in the system. The selection of operators and control strategies for distributed search are addressed elsewhere in [11], [12].

1.3 The STEAM Application System

Throughout the article, we will augment the presentation of concepts with examples from a seven-agent system, STEAM, that performs parametric design of steam condensers. Fig. 1 shows the general form of a STEAM condenser. The agents in STEAM each take responsibility for either:

1) designing some component of a steam condenser; or
2) critiquing some aspect of the condenser.

The agent set in STEAM is:

```
{pump-designer, heat-exchanger-designer,
  motor-designer, platform-designer,
    vbelt-designer, shaft-designer,
      system-frequency-critic}
```

These agents were initially created for another parametric design system [13] and were subsequently modified with a more flexible external interface to make them reusable. It is important to note that these agents in particular, and reusable agents in general, come with predefined functional capabilities and constraints and with predefined declarative knowledge. In contrast to custom-built applications, it is necessary to work with what is available rather than what is desirable. The trade-off between custom-built applications and applications comprising assembled reusable agents must consider that not all information that would be useful will realistically be available in reusable-agent systems.

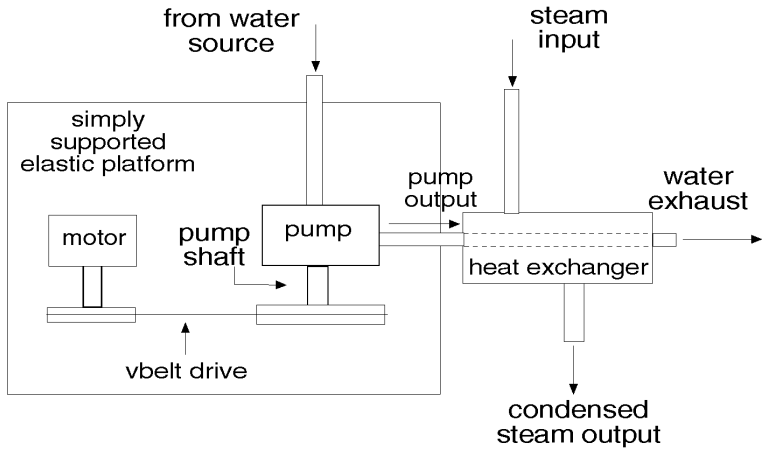


Fig. 1. A steam condenser.

1.4 Globally Cooperative Systems

The STEAM system is a *globally cooperative* system, meaning that there is assumed to be a global measure of system performance that overrides any local measures. This is generally true for design problems: There is some measure of the quality of a design that is distinct from the quality of any subcomponents of that design. An important aspect of globally cooperative systems is that it is not useful for agents to attempt to maximize their local payoffs for solutions by withholding information from other agents. The overriding goal of the system is to maximize the global, rather than local, payoff for solutions. In this situation, sharing information is not restricted by selfish or adversarial motives of agents as in some multiagent domains [14], [15].

Our emphasis is on the communication and assimilation of metainformation among reusable agents, the influence of shared information on an agent's ability to focus its search efforts, and the resulting performance improvements within an application system. In Section 2, we give a formal description of shareable metainformation in reusable-agent systems. Section 3 presents experimental results from STEAM that substantiate our hypothesis that metainformation sharing can improve solution quality and problem-solving efficiency. Section 4 discusses what costs are involved in sharing metainformation and what the practical limitations of the technology are from an application-system perspective. We conclude with a summary of observed results and some speculation as to the significance of these results within the STEAM system and within the more general realm of multiagent systems.

2 Solution Spaces: A Formal Perspective

When discussing the solution spaces of agents, we distinguish between the *local* space of an agent and the *composite* space of the system. A local space is one that is private to an agent, the composite space is one that is shared by all agents.[2] In a parametric design application, the *local solution space* of an agent is defined by the parameters that are assigned values by an agent in its local solutions, i.e., its output parameters. An agent's initial view of the *composite solution space* is equivalent to its local solution space. However, this local view is unlikely to be effective in finding solutions that are mutually agreeable to all agents (solutions in the composite space). A primary goal of communication among agents, therefore, is for each agent to end up perceiving the closest approximation possible to the part of the composite solution space that contains its local output parameters. In nontrivial cases, it is unlikely that a complete and correct global view can be achieved at every agent. However, as an agent's local view approaches the global view, the agent is likely to be more effective at proposing solutions that will be mutually acceptable.

2.1 Defining a Local Solution Space

We will use examples from the STEAM system to illustrate the concepts being discussed. Fig. 2 shows a simplified version of the solution space of `pump-designer`. This figure is simplified both in the number of parameters and the specification of the parameters' domains. The set of parameters in the solution space of an agent α is $\mathcal{P}^{\alpha}$, the *parameter set* of α. The parameter set of `pump-designer`, as shown in Fig. 2, is {water-flow-rate, head, run-speed, pump-cost}.

The set of legal values for a parameter θ at agent α is its *parameter space*, $\mathcal{V}_{\theta}^{\alpha}$. To illustrate, the parameter space of `run-speed` from Fig. 2 is the set of integers {1,200, 1,800, 2,400, 3,000, 3,600}. The parameter space of an agent can be defined over various domains including integers, reals, numeric intervals of the form {(*min*, *max*), *min*, *max* $\in \mathcal{R}$}, or discrete labels such as {*model*-1, *model*-2, …, *model*-*n*}.

A solution in the solution space of α is a tuple $s_j^{\alpha} = (p_1, p_2, \ldots, p_n)$ such that $p_x \in \mathcal{V}_{\theta}\, x^{\alpha}$ and such that any constraining relationships on or between the $p_x \in s_j^{\alpha}$ are satisfied. A parameter space may be constrained by explicit constraints on solutions such as (`run-speed` ≥ 1,200) or through implicit constraints that are embedded in the functions an agent uses to search for solutions. As a trivial example of an *implicit constraint*, consider the following loop in pseudocode:

2. Spaces may also be shared by some subset of agents, but not the entire agent set [6]. These *common* spaces are outside the scope of this article, however, and we will not discuss them further here.

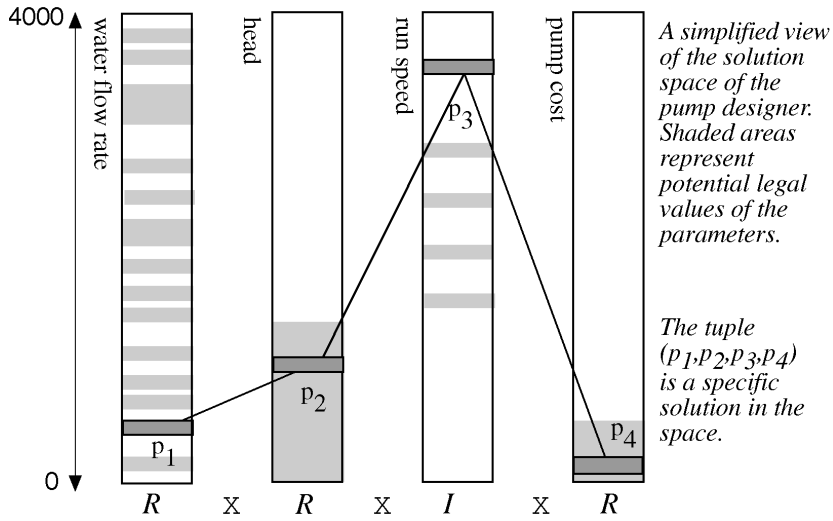


Fig. 2. The local solution space of pump-designer from the STEAM system.

```
head :== 0;
DO water-flow-rate = 0 to 500
  new-head :==
    calculate-head water-flow-rate;
  head :==
    select-best new-head head;
END DO;
```

An agent using this code implicitly constrains the parameter space of `water-flow-rate` to be the set of integers from 0 to 500, although it may not declaratively represent this anywhere. In reality, functions tend to be more complex and the implicit constraints more difficult to discern. In the above example, the value of `head` is tacitly constrained by the implicit constraint on `water-flow-rate`. However, the effect of this implicit constraint on the parameter space of head is not determinable without a deeper understanding of the constraining relationship.

The existence of implicit constraints, goals, and heuristics must be expected in the general case of expert agents. Implicit metainformation cannot normally be shared since it is an integral part of an agent's expertise and cannot be easily extricated.[3] Unshareable information strongly affects properties of the agent sets in which it is embedded. For example, in [16], Khedro and Genesereth present a distributed-search model in which agents provably converge on a globally satisfactory solution if one exists. However, the property of convergence can only be guaranteed if all constraining information can be explicitly exchanged. When implicit constraints are added, this desirable property no longer holds. When building an application, therefore, it is important to carefully examine whether all constraining information can be declaratively represented and, if so, whether centralized constraint satisfaction can be applied. In general, if centralized constraint satisfaction is appropriate, it is likely to be the most effective approach.

Explicit (declaratively represented) information can be shared and, as will be discussed in Section 3, this sharing can greatly enhance the effectiveness and coherence of the agent set. In STEAM, shareable information is limited to simple boundary constraints with the basic form (`water-flow-rate` < 800) that specify minimum or maximum values for a parameter (see Section 3.1 for a more complete explanation of shareable constraints). This limitation is not mandated by either our model of shareable information or the STEAM system, but rather was chosen due to its simplicity and accessibility. The costs of information sharing increase as agents apply more sophisticated techniques. Restricting information to boundary constraints may result in lower efficiency or lower-quality solutions than could be achieved by sharing more complex forms of information. However, the overhead costs associated with boundary constraints are low and these constraints provide a manageable, first-cut view of the composite solution space. In Section 5, we will further discuss the trade-offs inherent in supporting the sharing and assimilation of more complex forms of information; namely, the trade-offs between improved system performance due to better-informed agents and degraded system performance due to the overhead associated with sharing.

To include explicit boundary constraints in the definition of a solution space, we use the following notation: let c_j^α be a declaratively represented boundary constraint of agent α in the set of all explicit boundary constraints of α, C^α. Then, let the notation $\{c_j^\alpha : s_k^\alpha\}$ mean that c_j^α is satisfied with respect to a particular solution, s_k^α. For example if c_1^α is $(p_1 \le 10)$ and s_1^α = (9, 5, 3, 7), then c_1^α is satisfied with respect to s_1^α, $\{c_1^\alpha : s_1^\alpha\}$. When c_j^α is neutral with respect to s_k^α (it does not con-strain any parameters in s_k^α), it is considered to be satisfied.

Using this notation, the shareable solution space of agent α can be defined by specifying the parameter set of α, $\mathcal{P}^\alpha$

3. It is possible that some agents may be able to share either code or some form of abstracted explanation of implicit information. However, this requires specialized capabilities on the part of both the sending and receiving agents. Although it is possible to support these capabilities in specific situations, generalized code exchange and assimilation among heterogeneous reusable agents is not a realistic option.

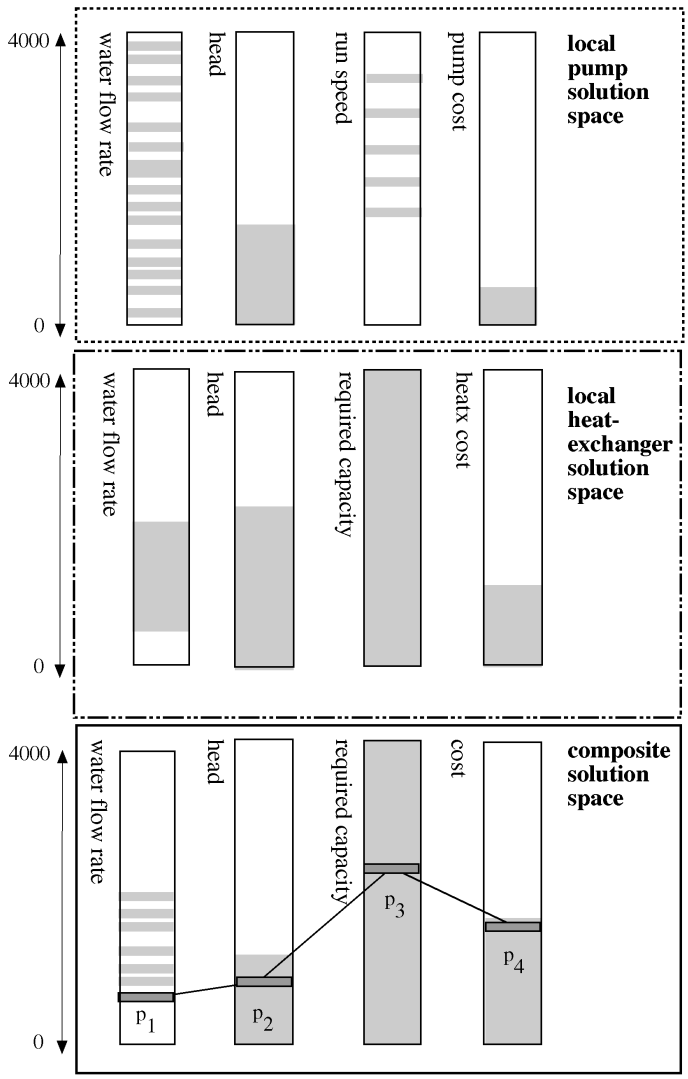
and the set of explicit constraints over those parameters, C^{α}. This shareable solution space is an approximation of the actual local solution space since it does not represent any implicit solution requirements that are embedded in the agent. We formally describe the shareable local solution space of agent α as follows: $\delta^{\alpha} = \{(p_1, p_2, \ldots, p_n) \mid (p_x \in \mathcal{V}_{\theta}^{\alpha}), (\forall c_j \in C^{\alpha}, \{c_j : (p_1, p_2, \ldots, p_n)\})\}$. In nontrivial cases, δ^{α} will be a superset of the valid solutions of agent α since it does not take implicit constraints into account.

2.2 Defining the Composite Solution Space

Given a set of agents, $\mathcal{A}$, and a problem that they are cooperating to solve, the desired composite solution must derive its parameter values from the local solution spaces of the agents. However, the parameter set of the composite solution space, $\mathcal{P}^{\alpha}$, is not necessarily the union of the parameter sets in the local solution spaces, as can be seen in Fig. 3.

In Fig. 3, the solution space of agent *p* (the pump agent) contains the parameters *water-flow-rate*, *head*, *run-speed*, and *pump-cost*. The solution space of agent *h* (the heat-exchanger agent) contains the parameters *water-flow-rate*, *head*, *required-capacity*, and *heatx-cost*. We find in Fig. 3 that the parameters *water-flow-rate* and *head* are common to both agents while *run-speed*, *pump-cost*, *required-capacity*, and *heatx-cost* represent parameters unique to individual agents. The composite solution space shown in Fig. 3 contains the shared parameters *water-flow-rate* and *head*, the parameter *required-capacity* from agent *h*, and also a unique parameter, *cost*. *Cost* is not local to either agent *p* or agent *h*, but represents a transformation on local parameters of those agents, i.e., the sum of *pump-cost* and *heatx-cost*. To summarize, each parameter in the composite solution space is local to either agent *p* or agent *h*, is local to both agent *p* and agent *h*, or is a unique parameter whose value can be derived from parameters local to agent *p* and / or agent *h*.

The *run-speed* output parameter of agent *p* does not appear in the composite solution. In this application system, *run-speed* can be characterized as a throw-away parameter: one that does not appear in a composite solution and is not used as an input parameter by any other agent. Throwaways are a common occurrence in reusable-agent systems since the agents are constructed without any foreknowledge of what output information will be re-



A simplified view of the relationship of the solution spaces of the pump agent and heat-exchanger agent to the composite solution space.

Fig. 3. Constructing a composite solution from the local solutions of agents.

quired for any specific application system. Therefore, there is no guarantee that the set of output parameters implemented in the agent will be exactly what is needed for any particular application.

In Fig. 3, notice that the constrained set of values (the shaded areas) of the shared parameters, *water-flow-rate* and *head*, are not identical for the two agents. If we are looking only at constraint-satisfaction problems (problems in which all constraints must be satisfied or no solution can be found), the constrained composite parameter space of a shared parameter is the intersection of the constrained local parameter spaces. For example, the shared composite parameter space of *water-flow-rate* from Fig. 3 is the intersection of the local *water-flow-rate* parameter spaces of the two agents *p* and *h*. We denote the composite *water-flow-rate* parameter space as w^C. If w^C is empty, no solution exists that will be mutually acceptable to agents *p* and *h*.

As discussed earlier, an agent's perception of w^C will not be identical to the actual composite parameter space in the general case. Formally, let w_α^C be an agent's perception of the composite parameter space, w^C. After all shareable information has been communicated and assimilated, agent α perceives some superset of w^C as defined by the explicit constraints of other agents and its own explicit and implicit constraints. If w_α^C is not empty, there are two possibilities:

1) a mutually acceptable composite solution, $s_x^C = (p_1, p_2, \ldots, p_n)$, exists with $p_w \in w^C$; or
2) no composite solution exists because there are implicit constraints at other agents that exclude values in w^C, but that do not exclude values in w_α^C.

Therefore, because of the possibility that implicit constraints exist, it is impossible to tell by looking at w_α^C whether or not a mutually acceptable solution exists. An example of the discrepancy between an agent's local solution space and other agents' perceptions of that space based on transmitted constraints is shown in the next section in Fig. 5.

Intersection of the fully constrained local solution spaces of agents (spaces constrained by both implicit and explicit constraints) defines the composite solution space in a constraint-satisfaction problem. However, in a *constraint-optimization* problem, not all constraints must be satisfied in a solution. Instead an attempt is made to satisfy constraints to the fullest extent possible. Constraints may have differing amounts of flexibility: Some may be *hard*, meaning that they must be satisfied in any legal solution, while others may be *soft*, meaning that they can be relaxed if necessary. Soft constraints can have different degrees of flexibility: Some can be "softer" than others. In these types of problems, composite solutions must lie within the intersection of the local parameter spaces under the set of hard constraints, but not necessarily under all soft constraints. The order in which constraints are relaxed can strongly affect system performance and solution quality [17], [18]. A discussion of these issues is beyond the scope of this article, however, in [6], Lander presents the algorithms used by STEAM agents to determine which constraints to relax in conflict situations and in situations where problem-solving progress has stalled.

In this section, we have defined the local, composite, and locally perceived composite solution spaces of a system of heterogeneous reusable agents. In complex application systems, the composite solution space is an amalgam of local spaces, each of which may be constrained in ways that cannot be articulated outside of the local context. Information sharing is viewed as a mechanism for reducing the difference between agents' local perceptions of the composite solution space and the actual space. The hypothesis is that as agents begin to understand the "big picture" they become more effective at generating high-quality solutions quickly. In the next section, we investigate this hypothesis through experimental observation of how information-sharing affects solution quality and processing time in the STEAM system.

3 Empirical Analysis of Information Assimilation

In this section, we empirically demonstrate the effectiveness of sharing potentially useful information among agents during distributed search. The experiments reported below were run in the STEAM system with the seven active agents listed in Section 1.3. There were two categories of experimental trials: *assimilation trials* and *nonassimilation trials*. In the assimilation trials, three agents instantiated the capabilities required for information assimilation: the *pump designer*, *motor designer*, and *heat-exchanger designer* agents. The other agents did not attempt to assimilate information; the reasons for limiting assimilation to three agents will be explained in Section 4.1. In the nonassimilation trials, the assimilation capabilities were not active at any agent.

The agents in STEAM, as in any multiagent system, must have an underlying infrastructure that supports agent interaction. STEAM was built on top of the commercial blackboard product GBB. Blackboard systems are an effective core technology in integrated multiagent environments because they provide highly efficient data sharing and sophisticated control capabilities.

The shared language in STEAM that is used to represent data on the global blackboards is a customized language that was created specifically for describing steam condensers. Shared languages in the general case can be thought of as ontologies that represent the terms and concepts required for interaction about the global theme of the integrated environment. In STEAM, local languages may include terms not represented in the shared language and, likewise, may not include all terms that are represented in the shared language. For internal processing, each agent uses a local dialect that is syntactically modified from the shared language. For example, *pump-designer* uses the term pump:run-head for internal processing of the run-head parameter and removes the prefix, pump:, for shared information. Boundary constraints have a format that is known to and understood by all agents to the degree that the terms embedded in the constraints are understood. An agent that sends out a constraint translates it into the shared language before transmission and the receiving agents translate it into their local idioms. If an agent does not have a local representation for some term in a boundary constraint, it ignores it.

Control of processing in STEAM is fairly complex, representing the need to develop and extend multiple alternative solutions, examine and resolve conflicts that occur in each solution, and evaluate solutions under both global and local evaluation criteria. The control strategy used is described in detail in [12]; it is an externally specified strategy that does not require sophisticated control reasoning mechanisms within the agents.

As described above, in order for an agent to use information received from an external source to guide its local processing (i.e., *learn* about other agents' requirements for solutions), the agent must be able to receive constraining information sent from other agents, translate that information into a locally usable form, and store the translated information into a local knowledge base so that it can be easily retrieved and applied. We call this process *information assimilation*. Notice, however, that with reusable agents, the usefulness of shareable information cannot be determined at agent-development time since it is dependent on capabilities and interests of other agents that may eventually be integrated into a joint agent set. Therefore, not all shared metainformation will be assimilated by all agents.

Assimilated information is used by the assimilating agent to guide its search for local solutions. We have developed mechanisms that extend or replace the traditional retrieval capability of an agent to extract relevant constraining information from its knowledge base. These mechanisms were developed specifically to enable reusable agents to handle potentially conflicting information that has been received from external sources since there is no guarantee that shared external information will be consistent with internal information. The goal of the retrieval process is to find the most restrictive, but nonconflicting, set of known constraints on solutions for the current problem using both local and assimilated information. This set of constraints defines the closest possible approximation of the composite solution space in which globally acceptable solutions must lie. However, it requires that intelligent conflict-resolution capabilities be applied to select which constraints to relax when conflicts do occur. Although we do not address specific techniques for conflict management in this article, it is an important and encompassing problem. Work describing computational conflict-management techniques includes [19], [20], [12], [21], [8], [22], [15].

In these experiments, agents transmitted boundary constraints directly in response to conflict situations rather than transmitting information that might potentially be useful. In other words, a boundary constraint was shared only when it conflicted with a proposed solution. The relative benefits and costs of reactive versus proactive information sharing are not explored here. The conflict that triggered the information exchange was handled by the agent's conflict-resolution capabilities while the exchanged information was assimilated by other agents and used to avoid similar future conflicts.

The system was run on each of 100 different solvable problem specifications, once with active assimilation capabilities and once without. The problem specifications were generated by randomly choosing a feasible value for each of the steam condenser attributes, {*required capacity, maximum platform deflection, platform side length*}, for each specification. The complete set of input problem specifications and observed data from the experiments are tabulated in [6].

3.1 Information Shared in the STEAM System

The information shared in these experiments was limited to simple boundary constraints of the form {`constraint-form, flexibility`}. A `constraint-form` is a single clause with the syntax ($x < n$), ($x <= n$), ($x > n$), or ($x >= n$), as previously discussed in Section 2.1. x is a shared numeric parameter and n is some numeric value. These constraints define an n-dimensional "box" in the solution space, where each dimension represents a parameter in the parameter set of the agent α, $\mathcal{P}^{\alpha}$. Ignoring for the moment the `flexibility` attribute, Fig. 4 shows the boundary constraints derived by two agents to represent their local solution spaces. The accuracy of the representation is dependent upon characteristics of the space. For example, when the solution space is sparse, as in Fig. 4a, the shareable view defined by boundary constraints will not be highly accurate, as shown in Fig. 4c.

Continuing the example in Fig. 4, Fig. 5 shows the perceived situation from each agent's perspective after the boundary constraints have been transmitted, along with the actual composite space.

Agent A has a much more accurate perspective than Agent B because, as noted earlier, Agent B's boundary constraints more accurately represent its solution space. Therefore, it is important to note that when boundary constraints are used as the primary mechanism for information sharing, it is possible to either predict (through agent analysis) or learn which agents have will the best global perspective. This is the basis of work that has been done by Lander [6] on making effective assignments of roles to agents within an agent set. It also provides the foundation for ongoing work in building reusable-agent sets that can automatically adjust role assignments for effective problem solving without requiring knowledge-based analysis of agent characteristics [23].

Returning to our discussion of how boundary constraints represent shareable perspectives of agents' solution spaces, notice that in some situations an agent may choose its boundary constraints such that some local solutions are excluded from the shared information. For example, if there are spurious outlying points, it may be advantageous to the entire system to exclude those points. By excluding them, the area that needs to be searched can be made much smaller, although this may result in missing valid or even optimal solutions. This idea is similar to that of relaxing the admissibility condition in A^* search whereby h^* is allowed to overestimate the distance to the goal, resulting in less search but a possibly nonoptimal solution. If an agent intentionally misstates its boundaries, whether its motives are benevolent or malicious, issues of deception arise. Deception in multiagent environments is an important and ongoing area of research [24], [25], [26] that must be addressed in the larger scope of multiagent systems in general. However, because we are looking at globally cooperative systems (in which the interests of individual agents are preempted by global measures of quality and performance), there is considerably less motivation for agents to misrepresent their expertise or results.

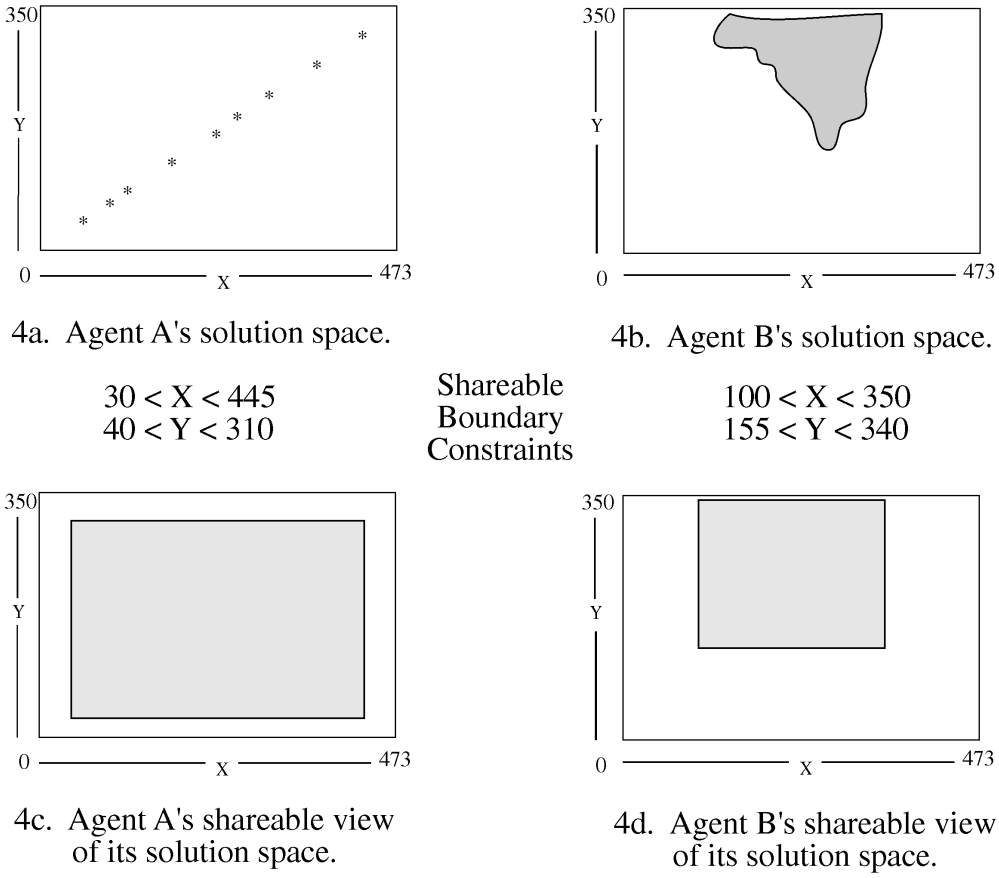


Fig. 4. Shareable views of agent solution spaces built using boundary constraints.

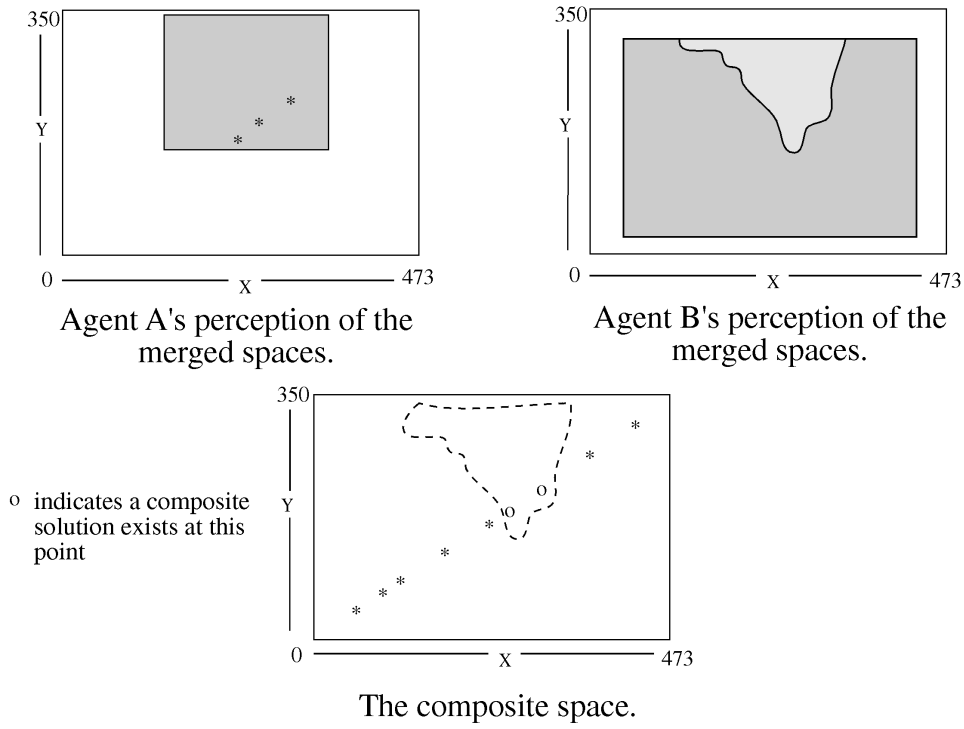


Fig. 5. Perceived solution spaces contrasted with the actual space.

In addition to the `constraint-form` clause of a constraint, the boundary constraints used by the STEAM agents include a `flexibility` attribute, loosely based on the notion of *utility* suggested by Fox [27]; `flexibility` is represented as an integer from 0 to 4. A flexibility of 0 specifies a hard (nonrelaxable) constraint that must be met by any feasible solution. A flexibility of 4 implies that solutions satisfying the constraint are of high quality and meet the most restrictive requirements of the agent. The use of a flexibility attribute on boundary constraints defines a *set* of n-dimensional boxes in the solution space, each representing an equivalence class of solutions of a given quality. For example, the box circumscribed by an agent's boundary constraints at flexibility 4 contains solutions that satisfy the most demanding local constraints of the agent. In contrast, the box circumscribed by an agent's boundary constraints at flexibility 0 contains solutions that are feasible but not necessarily desirable. For example, `pump-designer` specifies a soft constraint, {(*water-flow-rate* < 175), 4}, meaning that all pumps rated as *excellent* will have a water-flow-rate less than 175. It also specifies a hard constraint, {(*water-flow-rate* < 415), 0}, that indicates any value ≥ 415 will result in an infeasible solution.

The boundary constraints described above are only one form of information that can potentially be shared. They will not be adequate or appropriate in all domains. However, one result of our work has been the recognition that they are adequate to significantly improve processing in our domain. There is a high degree of complexity inherent in building heterogeneous agents that can understand each other well enough to positively affect mutual work. As the type of information to be shared increases in complexity, the design and implementation of agents needs to be more tightly coordinated to support effective generation and assimilation of shared information. Even the simple form of sharing shown here requires some uniform mechanisms across agents for representing, assimilating, and applying constraints. A primary principle in the development of reusable agents is that the degree of coordination required among agent implementers should be kept as small as possible. These constraints provide a minimal basis for interaction that can be shown to positively influence the overall search activity of the agent set.

There are two measures of system performance in the STEAM system: solution quality and runtime. We expected to see that extra costs associated with sharing information would be balanced, in the majority of cases, by improvements in performance. In the following sections, we first present the results from the information-sharing experiments on solution quality and runtime. We then discuss the underlying mechanisms in the STEAM system that produce the results: what is it about the agents that is affected by having external information available?

3.2 Solution Quality

We compared the results of running the system when agents assimilated constraining information and when they did not. In STEAM, solution quality is determined by the monetary cost of each solution: the minimum-cost acceptable design is considered the most highly rated. The results of the 100 experimental trials are graphically summarized in Fig. 6. In this figure, the results are sorted into ascending order based on cost in the assimilation trial.

For the 100 problem specifications tested, the mean cost in the assimilation trials was $8,504.77; in the nonassimilation trials, it was $9,020.43. The mean cost improvement with assimilation operators enabled was 5.72%, meaning that the monetary cost of the most highly rated solution in an assimilation trial was 5.72% lower on average than that in the associated nonassimilation trial under the identical problem specification.

We had hypothesized that enabling assimilation would lower the cost of a design (thereby improving solution quality) and the experimental results appeared to support this hypothesis. To statistically confirm this result, we applied a paired difference t-test. In this type of test, the results from two matched trials are compared—in our case nonassimilation trials are compared to assimilation trials performed under the same problem specification. For each paired trial, the difference between the resulting design costs is calculated. Then the mean of the differences is com-

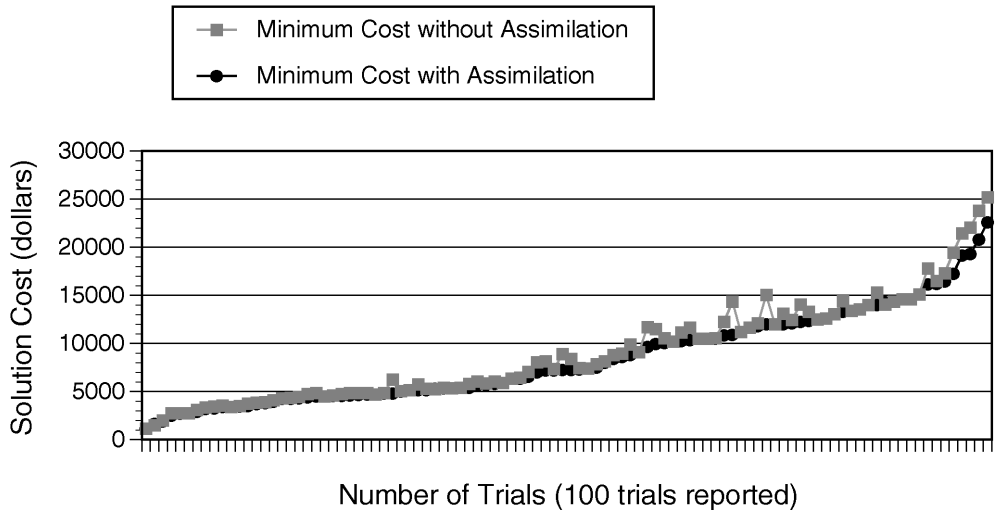


Fig. 6. Solution quality results in assimilation experiments.

puted over the entire set of trials. The null hypothesis in this case is $H_o : \mu_D = 0$ (the population mean of the differences is 0), meaning that the results of the two types of trials are not significantly different. The alternative hypothesis is $H_a : \mu_D > 0$ (the population mean of the nonassimilation trial results minus the assimilation trial results is greater than 0), meaning that the cost of designs in the nonassimilation trials are higher than those in the assimilation trials. Applying the paired t-test results in a t-score of 6.455, which allows us to reject the null hypothesis with a confidence of more than 99%. We can thus say with a high level of confidence that when STEAM agents apply assimilation capabilities, the average quality of solutions improves.

An inherent characteristic of the STEAM domain is that good solutions are easy to find under many problem specifications (the solution space is dense). We believe that there is a significant *floor effect* in the domain, meaning that minimum-cost designs are easy enough to find even in the nonassimilation trials that it is difficult to dramatically improve solution quality. However, the ability to consistently lower design costs approximately 5.72% by sharing simple boundary constraints is compelling evidence that information sharing and assimilation is an important technique for improving solution quality in multiagent systems. Furthermore, if it is the case that a floor effect is influencing our results, larger improvements could be expected in some domains.

The 5.72% figure given above for the improvement in solution quality may understate the significance of the improvement if the assimilation trials are approaching optimality. For example, assume that our average figures of $8,504.77 and $9,020.43 represent the results of a matched pair trial. If the optimal solution in this trial was actually $8,300, the assimilation run would be suboptimal by $204.77 while the nonassimilation run would be suboptimal by $720.43. With respect to the optimal solution, then, the assimilation run would show approximately a 72% improvement over the nonassimilation run in terms of error. Although this is a hypothetical situation since we do not know the average optimal solution for these trials, we believe it represents a reasonable estimate.

3.3 Runtime

Runtime is directly measured in these experiments as the elapsed real time from the invocation of the system until termination of the system.[4] The average runtime with assimilation is 121.98 seconds, without assimilation the average runtime is 132.67 seconds. The assimilation runtimes are, on average, 8.06% lower than the nonassimilation runtimes. However, direct comparison of the runtimes of assimilation and nonassimilation trials is somewhat misleading. There is a bias in the system that indirectly favors the nonassimilation trials; both the bias and its remedy are discussed next.

STEAM is a *satisficing* system [28]: There is no way to determine if an optimal solution has been achieved, and it is difficult to decide when to stop looking for a better solution. In any satisficing system, some policy must be provided that defines under what conditions the system will terminate processing. The termination policy used in the STEAM system is that when three acceptable solutions are found, the system enters a termination phase. The rationale behind creating at least three alternative solutions is that alternative solutions represent various design trade-offs, and the user should have the opportunity to decide which of the trade-offs is best for her needs. Furthermore, instead of halting immediately with the first three completed acceptable solutions, STEAM finishes all remaining acceptable partial solutions (as long as they remain acceptable). This policy is appropriate for the STEAM domain since solution quality and user participation are higher priorities than runtime.

Due to the focused search that occurs when information is shared, there are likely to be more acceptable solutions produced per run in the assimilation trials than in the nonassimilation trials. We found that this is indeed the case: More of the solution paths that are started in the assimilation trials turn out to result in acceptable solutions. Therefore, one result of the termination policy used in STEAM is a bias in which direct runtime measures favor nonassimilation trials because, in those trials, many of the potential solutions are poor enough to prune.

To overcome this bias and make runtime comparisons more meaningful, we divided the runtime of each trial by the number of solutions completed during that trial, resulting in a *runtime-per-solution* measure. The results obtained using this method are graphed in Fig. 7.

The *runtime-per-solution* observations in the assimilation and nonassimilation trials were averaged over the 100 experiment sets for comparison. The average runtime per solution in the assimilation trials was 11.58 seconds and in the nonassimilation trials it was 19.50 seconds. The average percent improvement achieved by the assimilation trials over the nonassimilation trials in runtime per solution was 40.62% (also a statistically significant result). In addition to highlighting the improvement in system efficiency in the assimilation trials, the bias discussed above indicates that the system could be tuned for faster runtime at the expense of solution quality by changing the termination policy to halt without finishing partial acceptable solutions.

3.4 Understanding the Effect of Shared Metainformation on System Performance

We stated above, based on the t-test analysis of solution quality, that when STEAM agents apply assimilation capabilities, the average quality of solutions improves. However, knowing that quality improves is not equivalent to understanding why it improves.

As described earlier, the goal of metainformation sharing is to improve agents' local perceptions of the composite solution space in order to make local search more productive. The more accurate the view of the composite solution space, the less time is wasted in producing solutions that are locally, but not globally, acceptable. Therefore, we expected to see runtime measurements improve in the assimilation trials because agents would waste less time in

4. These experiments were run on a TI Explorer II. Incremental garbage collection was turned off during the runs. However, the recorded time includes time spent on process and memory management tasks. Therefore, recorded times varied slightly across identical runs.

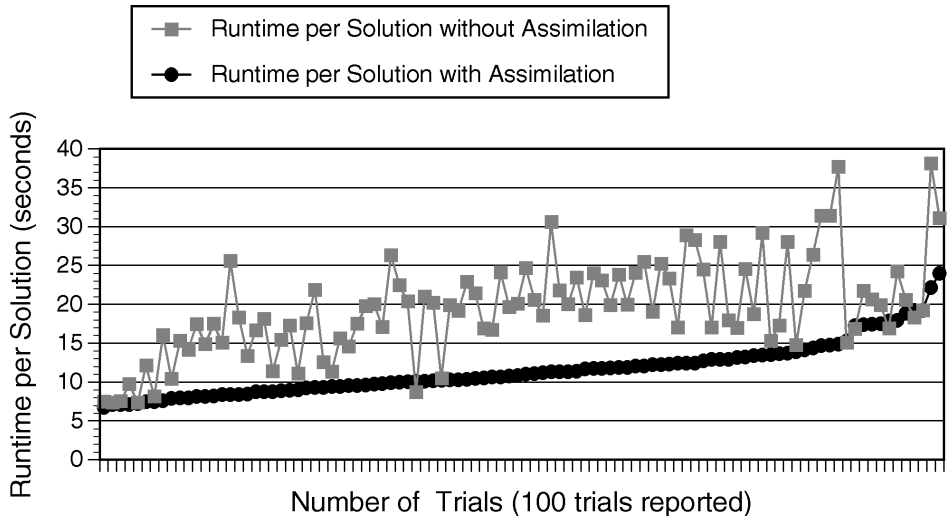


Fig. 7. Runtime-per-solution results in assimilation experiments.

unproductive tasks. This expectation was fulfilled in the experiments performed.

Though it seems clear that system runtime would be affected by information sharing, it is not as easy to see why solutions would improve in quality as well. To understand the relationship between solution quality and information assimilation in the STEAM system, it is necessary to understand the relationship between constraint relaxation and solution quality. In this section, we describe the mechanisms of constraint relaxation that are affected by information sharing and that, in turn, affect the quality of solutions produced.

In any nontrivial agent environment, there will be conflicts among the boundary constraints of different agents. These conflicts are often *soft conflicts*, meaning conflicts that occur over soft constraints. In order to find any solution in the composite solution space, these soft conflicts must be resolved by relaxing one or more of the conflicting constraints. In the STEAM domain, there is an approximate mapping between the degree of local constraint satisfaction and the quality of a solution. In general, the more relaxation has occurred in the environment, the lower the quality of solutions produced. This is an approximate mapping because the global evaluation of a design in a globally cooperative system is not necessarily a direct function of local agent evaluations. For that matter, a local agent evaluation is not necessarily a direct function of local constraint satisfaction. However, it is usually the case that there is a relationship between constraint satisfaction and solution quality and, in the STEAM domain, this is a reasonable assumption.

There are three primary types of constraint relaxation used in STEAM [6]:

- *Unilateral* relaxation occurs at an agent in direct response to a problem specification—the agent will relax local soft constraints in order to meet requirements imposed by the specification. Unilateral relaxation is not dependent on information sharing and will not be further discussed.
- *Responsive* relaxation occurs when there are explicit conflicts between an agent's constraints and some other agent's constraints. In responsive relaxation, an agent that has received conflicting constraints from another agent determines which of its own constraints to relax or which of the received constraints to ignore based on some conflict-management criteria. In this case, relaxation is used to remove a conflict and it is specifically enabled by information sharing.
- *Automatic* relaxation occurs in response to a lack of problem-solving progress. Because not all relevant information is shareable, it is sometimes the case that problem solving stalls over implicit conflicts without any agent being able to pinpoint the cause. One way to handle this situation is to set up the system so that one or more agents must select and relax a local constraint after some amount of time has gone by. Decisions about which agent should relax which constraint are not discussed here—this is a complex problem and in general these decisions are highly domain-dependent and agent-set specific. However, the basic rationale for automatic relaxation is that unless it is possible to directly attribute a lack of problem-solving progress to a particular constraint, there must be a mechanism for selecting and relaxing arbitrary constraints until the obstacle is removed. This basic idea has been part of the DAI literature for some time. For example, in the Hearsay-II speech understanding system developed in the late 1970s, a group of hypotheses for words were generated and rated, and the most highly rated hypotheses were passed to the word-sequence level of processing that would then try to build multiword sequences. If problem solving stagnated at that level, rating thresholds were relaxed at the word level in order to provide more breadth in the word-sequence search space [29].

Given this brief introduction to constraint relaxation, we now return to our discussion of how information sharing and assimilation affects global solution quality in a globally cooperative system. Without information sharing, the default form of constraint relaxation is automatic relaxation:

basically a blind search for the source of a problem. In the course of "fixing" the problem, other nonproblematic constraints are also often relaxed. This leads to lower standards for solutions and, ultimately, lower quality solutions. In contrast, responsive relaxation supports the agents in making globally beneficial decisions about which constraints to relax. By providing an agent with specific knowledge about the source of the problem, information sharing enables the agent to make an intelligent decision about the most appropriate solution to the problem. Knowledge that can be brought to bear in deciding which constraint to relax includes power relationships between agents and the flexibility of the constraints involved in the conflict.

Although automatic constraint relaxation occurs both with and without information sharing and assimilation, its effects are mitigated when sharing occurs. A general consequence of automatic relaxation policies is that the local quality of proposed solutions degrades over time as requirements become less stringent. Because of this, the timing of solution generation is important—it is important that each agent quickly focus its local search as narrowly as possible. Because information assimilation and sharing accomplishes precisely that, the quality of solutions improves simply because good solutions are proposed before any arbitrary relaxation occurs.

3.5 Summary

The results presented in this section demonstrate that information sharing can positively affect both solution quality and runtime in a heterogeneous reusable-agent system. However, there are costs associated with information sharing and, in fact, the richer the sharing environment, the higher the costs are likely to become. In the following section, we discuss the costs involved in information sharing with an eye toward determining how sophisticated it is practical to become.

4 Information-Sharing Costs

Sharing information has five primary costs (where cost is measured in time):

1) *Generation*: the cost of generating shareable information at the sending agent specifically intended to be transmitted to other agents to guide their local searches;
2) *Determination*: the cost of determining what information to share at a given point in problem solving;
3) *Transmission*: the actual costs associated with the physical transmission of messages among agents;
4) *Translation*: the cost of translating shared information either directly from one agent's language to another's or from the local language to a shareable format at the sending agent and then from the shareable format into a local language at the receiving agent;
5) *Local Management*: the cost of managing shared information at the receiving agent; determining the applicability of received information (sorting, filtering, detecting conflicts and locally resolving those conflicts) and managing the greater volume of information that results from accumulating received information (storage and retrieval costs).

In the timing analysis described below, we do not include transmission costs (those associated with the physical transfer of information from one agent to another). This is consistent with the current implementation of STEAM in which all agents reside on the same machine and run in the same process, making transmission costs virtually nil. This exclusion of transmission costs should not be extended to the general case however.

The list above summarizes the types of costs incurred as a direct result of information sharing. We next give a more complete description of those costs and present observed data from their measurements in the STEAM system. We then move on to discuss the relevance of our observations in the more general context of reusable-agent systems. We conclude with some thoughts on how agent reusability affects information sharing, discussing both the positive and negative issues that arise.

Generation: The costs listed as *generation* costs should represent only the time an agent spends generating shareable information that would not otherwise be declaratively represented. This can potentially entail a great deal of 'self-analysis'. Types of information that can be used to focus other agents' searches include:

1) constraints that are completely independent of the specific problem being addressed (*independent* constraints);
2) constraints that are dependent only on the problem specification without regard to any particular solution (*problem-dependent* constraints); and
3) constraints that are dependent on existing instantiated parameters for a particular solution (*solution-dependent* constraints).

These different categories are explained in more detail in [6]; the effect of each category in the timing studies reported are described briefly below.

In the timing studies reported here, we investigate the use of problem-dependent constraints. In other words, we measure the time it takes to construct boundary constraints that are dependent on a particular problem specification. For example, a problem specification in STEAM includes a fixed value for the required capacity of the desired steam condenser. Required capacity is an input parameter to the *heat-exchanger-designer* that affects possible values of other local parameters. Therefore, each time a new problem specification is provided to the system, the heat-exchanger designer must recompute the boundary constraints on any parameters affected by the assigned value of required capacity.

Costs associated with generating independent constraints are not considered to be part of the normal cost of developing a solution because these constraints can be generated in a one-shot preprocessing procedure.

Solution-dependent constraints are not used in STEAM although, in some domains, it is possible to exploit these constraints. If the agents in an application system (or some subset of the agents) have a limited number of local alternative solutions, it may be effective to develop guiding information that relates specifically to a single solution. For example, say an agent, Z, in a globally cooperative system has only two alternative local solutions, either *proposal 1* or *proposal 2*. Any global solution will therefore incorporate

one of those two proposals. In this situation, it might well be worthwhile to specify solution-dependent constraints such as {if x = *proposal1.x*, then $y > 10$}. Even though deriving this constraint might entail considerable runtime analysis, it will be applicable 50% of the time. However, assume instead that Z has 100,000 alternative solutions. In this case, the above constraint is only applicable in 0.001% of the potential solutions, and it is unlikely that the time spent in generating the constraint will be recovered by its effect on focusing the composite search. In their work on multistage negotiation, Conry et al. have developed a formalism that generates solution-dependent constraints for a set of subplans through iterative agent interactions. These constraints focus the system and enable it to determine that no solution exists when no nonconflicting constraint set can be found [30]. However, solution-dependent constraint generation and manipulation techniques are not useful in the STEAM domain because of the size of the solution spaces at each of the agents.

Determination: An agent must decide what information to transmit. In STEAM, agents transmit information directly in response to conflict situations rather than transmitting information that is anticipated to be potentially useful. Therefore, only constraints that are in direct conflict with an existing solution are transmitted. The costs of retrieving potentially conflicting constraints and checking each constraint to see if it conflicts with the existing solution are reported in this measure.

Notice that in some domains, agents might be more proactive than reactive with respect to when information is transmitted. For example, an agent might broadcast its constraints without waiting for a conflict to occur, thereby facilitating conflict avoidance rather than conflict resolution. If all constraints are transmitted proactively at some set time (e.g., as soon as problem solving begins), determination costs would be insignificant; otherwise, some time must be spent in deciding what to send.

Translation: In the general case of heterogeneous reusable-agent systems, local information can be represented at an agent in any form that is appropriate for that agent but some mechanism must be provided to ensure that agents are able to understand each other. When translation is necessary, the cost can vary greatly depending on exactly what is entailed. Some agents may share a language and have no translation costs, others may translate using simple syntactic procedures, and others may require complex semantic translation. In STEAM, the local representation of an agent's information is unrestricted, but in order for information to be shared, it must be translated into a globally specified language. All agents use the same simple syntactic procedures for translation between local and global formats. Translation costs, therefore, do exist but are relatively small.

Local Management: Conflict between local and assimilated information is one factor that potentially undermines the benefits of information sharing: What happens when an agent receives information that contradicts something it already knows? With independently developed and heterogeneous agents, it must be assumed that conflict will occur. In the STEAM system, the costs of managing conflicts between inconsistent local and external information are categorized as local management. Other local management costs include costs that accrue from the greater volume of information that must be stored and retrieved due to assimilated information.

4.1 Observed Information-Sharing Costs in STEAM

In these experiments, the costs attributed to sharing information are broken down into four categories:

1) generation (for problem-dependent constraints);
2) determination;
3) translation; and
4) local management, as described earlier.

The observed costs for each of these categories over the 100 problem specifications are summarized in Fig. 8 (the complete set of data is tabulated in [6]).

The approximate average time spent in constraint generation per problem is 7.3 seconds, in determination of which constraints to transmit is 0.4 seconds, in translation is 0.5 seconds, and in local management is 4.2 seconds, for an average total time for information assimilation of approximately 12.4 seconds per run (out of 121.98 seconds average runtime). The average total percentage of time spent in information sharing in these trials is 10.17%. These figures are highly domain-dependent and each of the different areas could be more or less expensive in other situations. For example, if more elaborate constraints were being generated or if a more sophisticated analysis of the local search space was performed, the constraint generation time would be higher and consume a larger proportion of the processing time. Likewise, if translation were more difficult and involved some semantic interpretation as well as strict syntactic replacement, it would take more time. The most important point to bring away from these experiments is that these costs will be incurred in any domain though the figures will vary. Determining the degree of information sharing to support in a particular application requires that the agent implementer understand where the costs lie and whether the potential gains outweigh them.

In the assimilation experiments described above, only three of the seven agents instantiated information-assimilation capabilities. The primary reason for this is that implementing these capabilities is very difficult. For each agent, the implementation is unique and requires a thorough understanding of the information requirements and search mechanisms of that agent. This suggests that it must be done by the agent implementer at the time the agent is built. The agent implementer cannot be responsible for determining what local information will be relevant in a particular application system since the agent may be embedded in different systems. However, the agent implementer must determine what local information will be shareable. Furthermore, the agent developer must anticipate the types of information that may become available to the agent during problem solving and build into the agent the capabilities required to effectively apply that information.

We demonstrate the difficulty inherent in implementing effective information assimilation through an example.

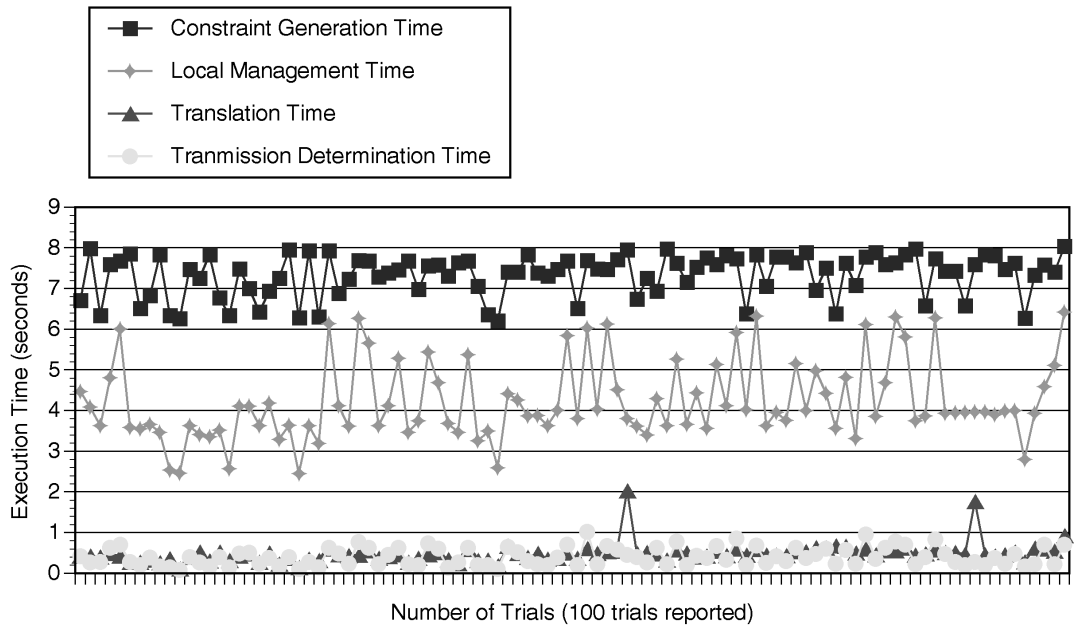


Fig. 8. Information-sharing costs in STEAM.

Say that the `pump designer` receives a constraint from the `heat-exchanger designer` that restricts a parameter called *run-head* of pumps proposed by the pump agent. This constraint is not directly applicable during the search for candidate pumps because the value of *run-head* is computed after the specific pump is chosen: it is an output parameter rather than an input parameter. However, once a candidate pump has been generated, the *run-head* for that pump can be computed and the constraint can be applied as a filtering mechanism to eliminate nonviable candidates. If `pump designer` does apply the filtering constraint, it will still have to iteratively generate and test candidate pumps locally, but will eliminate infeasible ones before other agents are asked to respond to them. Therefore, by appropriately applying assimilated information, it can reduce the workload of other agents.

The point here is that it is not only necessary to understand the language of received information, it is also necessary that the agent know how to apply it. Applying the information appropriately can be subtle—it may have to be applied differently than the agent's own local information, for example, as a postsearch filter as described above. This implies that an agent must anticipate the kinds of information it may receive and have internal procedures available to effectively use that information.

5 Conclusions

In this article, our objective was to clarify the costs and benefits that are attributable to information sharing in systems of heterogeneous reusable agents. The experiments in Section 3 demonstrated that sharing and assimilating meta-information about the composite solution space positively affects both solution quality and runtime. When external information is assimilated by an agent, that agent is able to focus its search efforts in areas of its local solution space that are more likely to be contained in the composite solution space as well. By focusing its search in areas that are likely to be mutually acceptable, the agent's work is more productive and will tend to improve both solution quality and system performance. However, there are implementation and performance costs associated with information sharing and, in some situations, these costs may outweigh the benefits.

We classified the costs of information sharing as involving: the generation of information to share; the translation of information into and out of a shared language; the determination of what information to communicate at any given time; the transmission of information (not included in our experiments); and local management (storage, retrieval, and use of potentially conflicting assimilated information). We observed these costs within the STEAM system and found them to total approximately 10.17% of the overall runtime. In this domain, the time spent in sharing information is more than balanced by the productivity enhancement that comes from focusing on mutually acceptable areas of the composite solution space: We recorded mean improvements in solution quality of 5.72% and in runtime-per-solution of 40.62%.

Our experience with information sharing suggests some conflicting perspectives on its achievement in systems of heterogeneous reusable agents. On the one hand, our experiments showed that information sharing can be potent tool for improving system performance, both in terms of solution quality and runtime. On the other hand, we found sharing difficult to realize effectively because it utilizes both global understanding of the needs of a complete system and an in-depth understanding of characteristics of individual agents. Either the application developer (who integrates a set of reusable agents into a comprehensive system) or the agent implementers (who build individual agents) must be responsible for achieving information

sharing in a multiagent application. The application developer cannot be expected to have a deep enough understanding of individual agents to install mechanisms such as the postsearch filter described above. Likewise, the agent implementers don't know what information will be needed or available in the system and must therefore make decisions about what mechanisms to install in individual agents in an ad hoc manner.

Can reusable-agent search systems be built without giving agents the ability to exchange metainformation? The answer depends on what is required. Multiagent search without metainformation reduces to blind search or possibly search guided by local heuristics in the composite search space. In small, simple application systems, this may be enough. However, reusable agents that cannot coherently focus their search are unlikely to be effective in complex application systems. Future research in reusable-agent systems should examine questions of balancing the information-sharing capabilities of agents with the benefits of sharing various types of information. It may be that some general guidelines will emerge that can be applied by agent implementers to decide what capabilities are likely to be most beneficial and cost-effective in an agent.

In conclusion, we have shown that information sharing and assimilation can enhance system performance in the STEAM system. Although there is no basis on which to generalize any specific figures outside of STEAM, the STEAM domain is typical of a class of small-scale globally cooperative design domains and our results indicate that information sharing and assimilation can improve performance in this class of systems. Furthermore, the categories of information-sharing costs hold across all domains. Both the empirical evidence demonstrated here and intuitive arguments for the benefits of focused search suggest that information sharing and assimilation will be effective in more complex domains.

Although sharing metainformation is potentially beneficial, it is not particularly easy to achieve. Most of the work must be done at agent-implementation time when nothing is known about the application system(s) into which the agent will be embedded. The costs of making agents that are highly proficient in sharing and using assimilated information may outweigh the benefits that accrue from applying those capabilities. Future work may clarify the boundaries of benefit versus hindrance based on types of information and the capabilities required by agents to use those various types. However, it is clear that information sharing and assimilation should be considered a potential source of performance enhancement when designing distributed-search systems comprising heterogeneous reusable agents.

ACKNOWLEDGMENTS

We thank M.V. Nagendra Prasad for his valuable suggestions and for his considerable expertise in helping to organize and edit this material. We also thank Daniel Corkill for his insightful suggestions and comments.

This work was supported in part by Advanced Research Projects Agency contract N00014-92-J-1698, and National Science Foundation contract CDA 8922572. The content of the information does not necessarily reflect the position or the policy of the government, and no official endorsement should be inferred.

REFERENCES

[1] S.E. Lander, S.M. Staley, and D.D. Corkill, "Designing Integrated Engineering Environments: Blackboard-Based Integration of Design and Analysis Tools," *Concurrent Eng. Research and Applications*, special issue on the application of multiagent systems to concurrent engineering, vol. 4, no. 1, pp. 59-72, Mar. 1996.

[2] R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senator, and W.R. Swartout, "Enabling Technology for Knowledge Sharing," *AI Magazine*, vol. 12, no. 3, pp. 36–56, Fall 1991.

[3] T. Finin, R. Fritzson, D. McKay, and R. McEntire, "KQML as an Agent Communication Language," *Proc. Third Int'l Conf. Information and Knowledge Management*, Nov. 1994.

[4] T.R. Gruber and G.R. Olsen, "An Ontology for Engineering Mathematics," J. Doyle, P. Torasso, and E. Sandewall, eds., *Proc. Fourgh Int'l Conf. Principles Knowledge Representation and Reasoning*, Gustav Stresemann Institut, Bonn, Germany, Morgan Kaufmann, 1994.

[5] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, 1991.

[6] S.E. Lander, *Distributed Search and Conflict Management Among Reusable Heterogeneous Agents*, PhD thesis, Univ. of Massachusetts, Feb. 1994.

[7] R.G. Smith and R. Davis, "Frameworks for Cooperation in Distributed Problem Solving," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 11, no. 1, pp. 61–70, Jan. 1981 (also published in *Readings in Distributed Artificial Intelligence*, A.H. Bond and L. Gasser, eds., pp. 61–70, Morgan Kaufmann, 1988).

[8] S. Sen and E.H. Durfee, "A Formal Study of Distributed Meeting Scheduling: Preliminary Results," *Proc. Conf. Organizational Computing Systems*, Atlanta, pp. 55–68, Nov. 1991.

[9] A. Barr and E.A. Feigenbaum, *The Handbook of Artificial Intelligence*, vol. 1, William Kaufmann, 1981.

[10] V.R. Lesser, "An Overview of DAI: Viewing Distributed AI as Distributed Search," *J. Japanese Soc. Artificial Intelligence*, vol. 5, no. 4, pp. 392–400, July 1990.

[11] S.E. Lander and V.R. Lesser, "Customizing Distributed Search Among Agents with Heterogeneous Knowledge," *Proc. First Int'l Conf. Information and Knowledge Management*, Baltimore, pp. 335–344, Nov. 1992.

[12] S.E. Lander and V.R. Lesser, "Understanding the Role of Negotiation in Distributed Search Among Heterogeneous Agents," *Proc. Int'l Joint Conf. Artificial Intelligence*, Chambery, France, Aug./Sept. 1993.

[13] K.L. Meunier, "Iterative Respecification: A Computational Model for Automating Parametric Mechanical System Design," master's thesis, Univ. of Massachusetts, Amherst, Feb. 1988.

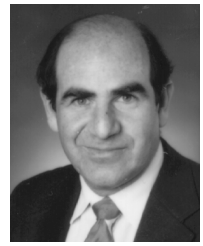
[14] K. Sycara, "Multi-Agent Compromise via Negotiation," *Distributed Artificial Intelligence*, vol. 2, L. Gasser and M. Huhns, eds., Los Altos, Calif.: Morgan Kaufmann, Sept. 1989.

[15] G. Zlotkin and J.S. Rosenschein, "Cooperation and Conflict Resolution via Negotiation Among Autonomous Agents in Non-Cooperative Domains," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 21, no. 6, pp. 1,317–1,324, Nov./Dec. 1991.

[16] T. Khedro and M.R. Genesereth, "Progressive Negotiation for Resolving Conflicts Among Distributed Heterogeneous Cooperating Agents," *Proc. 12th Nat'l Conf. Artificial Intelligence*, Seattle, July/Aug. 1994.

[17] M.S. Fox, B. Allen, and G. Strohm, "Job-Shop Scheduling: An Investigation in Constraint-Directed Reasoning," *Proc. Nat'l Conf. Artificial Intelligence*, Pittsburgh, pp. 155–158, Aug. 1982.
[18] Y. Nishibe, K. Kuwabara, and T. Ishida, "Effects of Heuristics in Distributed Constraint Satisfaction: Towards Satisficing Algorithms," *Proc. Workshop Distributed Artificial Intelligence*, Michigan, pp. 285–302, Feb. 1992.
[19] M.R. Adler, A.B. Davis, R. Weihmayer, and R.W. Worrest, "Conflict-Resolution Strategies for Non-Hierarchical Distributed Agents," *Distributed Artificial Intelligence*, vol. 2, L. Gasser and M.N. Huhns, eds., Research Notes in Artificial Intelligence, pp. 139–161. Los Altos, Calif.: Morgan Kaufmann, Sept. 1989.
[20] M. Klein, "Supporting Conflict Resolution in Cooperative Design Systems," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 21, no. 6, pp. 1,379–1,390, Nov./Dec. 1991.
[21] A. Sathi and M.S. Fox, "Constraint-Directed Negotiation of Resource Reallocations," *Distributed Artificial Intelligence*, vol. *2*, L. Gasser and M. Huhns, eds., chapter 8, pp. 163–193. Los Altos, Calif.: Morgan Kaufmann,, 1989.
[22] K. Sycara, "Resolving Goal Conflicts via Negotiation," *Proc. Seventh Nat'l Conf. Artificial Intelligence*, St. Paul, Minn., pp. 245–250, Aug. 1988.
[23] M.V. Nagendra Prasad, V.R. Lesser, and S.E. Lander, "Learning Experiments in a Heterogeneous Multi-Agent System," *Proc. Workshop Adaptation and Learning in Multiagent Systems, IJCAI-95*, Montreal, Aug. 1995 (also available as CS Technical Report 95-35, Univ. of Massachusetts, Amherst).
[24] E. Ephrati and J.S. Rosenschein, "The Clarke Tax as a Consensus Mechanism Among Automated Agents," *Proc. Ninth Nat'l Conf. Artificial Intelligence*, Anaheim, Calif., pp. 173–184, July 1991.
[25] P.J. Gmytrasiewicz and E.H. Durfee, "Toward a Theory of Honesty and Trust Among Communicating Autonomous Agents," *Group Decision and Negotiation*, vol. 2, no. 3, pp. 237–258, 1993.
[26] G. Zlotkin and J.S. Rosenschein, "A Domain Theory for Task Oriented Negotiation," *Proc. 13th Int'l Joint Conf. Artificial Intelligence*, Chambery, France, pp. 416–422, Aug. 1993.
[27] M.S. Fox, *Constraint-Directed Search: A Case Study of Job Shop Scheduling*, Research Notes in Artificial Intelligence. London: Pitman Publishing, 1987.
[28] H.A. Simon, *The Sciences of the Artificial*, MIT Press, 1969.
[29] L.D. Erman, F. Hayes-Roth, V.R. Lesser, and D.R. Reddy, "The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty," *Computing Surveys*, vol. 12, no. 2, pp. 213–253, June 1980.
[30] S.E. Conry, K. Kuwabara, V.R. Lesser, and R.A. Meyer, "Multistage Negotiation for Distributed Satisfaction," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 21, no. 6, pp. 1,462–1,477, Nov./Dec. 1991.

Susan E. Lander received the BS, MS, and PhD degrees in computer science from the University of Massachusetts, Amherst. After receiving her PhD in February 1994, she continued her research at the university as a postdoctural research associate until September 1994. Since that time, she has been a staff scientist with Blackboard Technology Group Inc., Amherst, Massachusetts. Dr. Lander's academic and industrial affiliations have led to a strong interest in technology transfer. Her research interests include distributed AI, multiagent architectures, control and conflict management in cooperative multiagent systems, agent-based legacy software reuse, and concurrent engineering. She served as guest coeditor for the March 1996 special issue of *Concurrent Engineering: Research and Applications* on the application of multiagent systems to concurrent engineering. She is a member of the IEEE.



Victor R. Lesser received his BA in mathematics from Cornell University in 1966; and the MS and PhD degrees in computer science from Stanford University in 1969 and 1972, respectively. He then worked as a research computer scientist at Carnegie Mellon University until 1977. He has been a professor in the Computer Science Department of the University of Massachusetts, Amherst, since 1977 and, in 1992, also became director of the Distributed Problem Solving Laboratory. His major research focus is on the control and organization of complex AI systems. Professor Lesser is a founding fellow of the American Association of Artificial Intelligence and is considered a leading researcher in the areas of blackboard systems, distributed AI/multiagent systems, and real-time AI. He has also made contributions in the areas of interpretation, diagnostics, plan recognition, and intelligent user interfaces, and recently has been working on techniques for more closely integrating signal processing technology with signal understanding for use in application domains involving acoustics and radar. He served recently as general chair of ICMAS-95, the first International Conference on Multiagent Systems, and is on the Editorial Boards of *IEEE Expert*, *Group Decision and Negotiation*, the *International Journal on Intelligent and Cooperative Information Systems*, and the *Real-Time Systems Journal*. He is a member of the IEEE.