DAI in Engineering Design

Michael N. Huhns
Microelectronics and Computer Technology Corporation
Enterprise Integration Division
3500 West Balcones Center Drive
Austin, TX, U.S.A. 78759-6509
huhns@mcc.com

There are three evolving facets to design: 1) the artifact to be designed, 2) the design process, and 3) the tools available to aid the design process. They are evolving because first, the design process is becoming more complex due to the need to consider the entire product life-cycle—from conception to manufacturing to sales to maintenance. Second, the artifacts of design are changing in that many products that used to be standardized are being specially designed for each customer, and more complicated artifacts, such as space stations and fusion power plants, are being attempted. Third, there are now a plethora of tools for aiding the design process, including tools for simulation, visualization, layout, test, aesthetics, compliance with standards, and manufacturability. In addition, as more and different artifacts are being designed, large amounts of data, knowledge, and experience are being accumulated that can be used to aid future design processes. The overall trend in each of the three facets of design has been towards increasing the complexity of a designer's task. This is turn has placed additional demands on the computational aids for design, with the foremost demand being for assistance with the increased complexity that has ensued.

There are several known ways to deal with complexity, including

- abstraction, whereby performing design at a more general level can focus the subsequent detailed design stages and vastly improve their efficiency;
- speed, whereby faster design techniques or faster implementations of these techniques can lead to solutions of larger design problems;

- better representation, whereby considering a design problem from the right viewpoint can lead to vast simplifications;
- past experience via learning, cases, and analogy, which can reduce the need for experimentation or trial-and-error, and focus the search for a solution;
- modularity and parallelism, whereby problems can be decomposed into independent subproblems that can be solved in parallel.

A result of this last factor is that design is often performed by teams of designers. Computational aids for design should similarly be distributed, both to mirror the way humans perform design—and thus easily fit into existing human design teams—and to take advantage of the reduced complexity engendered by modularity. Consider the following example:

> At a large chemical plant in rural Texas, dozens of expert systems are used to control the processing of petroleum-based chemicals. These expert systems are small, consisting of 10–100 rules each, and were written within a few months by the engineers at this plant. Each expert system successfully controls a single aspect of some larger process, replacing manual control with a tremendous savings in cost.
>
> Alternatively, a single large system to control all aspects of the plant could have been constructed, but it would have had roughly the complexity of DEC's XCON, making it prohibitively expensive, requiring a development time of many years, and putting it well beyond the knowledge-engineering and software-maintenance capabilities of the local engineers.

Similarly, design should be automated through the introduction of numerous small computational aids, involving a variety of reasoning and representation techniques, including knowledge bases, neural nets, and databases. However, new problems will arise involving interactions among these. Consider more of the above example:

> Typical of many chemical plants of this type, the processing is highly interconnected, with chemicals made in one part of the plant used in producing the chemicals made in another part of

> the plant. Recently, a boiler failure caused an expert system to shut down the production of a solvent that was needed in another process producing latex paint. Unfortunately, the expert system controlling the paint process did not find out about the shut down until the solvent in the input pipe to the column dried up. After the dried paint was cleaned from the column six months and $2 million later, the paint process was back on line.

The problem was that the processes in this plant were connected at the physical level, but the expert systems were not connected at the knowledge level, even though they were written in the same language, ran on the same hardware, and were connected by ethernet. They were not designed to communicate! They were unaware of the decisions being made by the other expert systems, so they were unable to take corrective action until, as in this case, it was too late.

The key problem for intelligent design tools will thus be to integrate and use all available information resources, including knowledge bases, databases, and application programs. The information resources may have been independently developed, but they must interact productively with the others.

Distributed artificial intelligence (DAI) [Huhns 87, Gasser & Huhns 89] provides some of the technology needed for this integration and interaction. DAI is concerned with how a decentralized group of intelligent computational agents should coordinate their activities to achieve their goals. When pursuing common or overlapping goals, they should act cooperatively so as to accomplish more as a group than individually; when pursuing conflicting goals, they should compete intelligently. Interconnected agents can cooperate in solving problems, share expertise, work in parallel on common problems, be developed, implemented, and maintained modularly, be fault tolerant through redundancy, represent multiple viewpoints and the knowledge of multiple human experts, and be reusable. DAI is the appropriate technology for applications where

- expertise is distributed, as in design;
- information is distributed, as in office automation;
- data are distributed, as in distributed sensing;
- rewards are distributed, as in automated markets;

- decisions are distributed, as in manufacturing control; and
- knowledge bases are developed independently but must be interconnected or reused, as in next-generation knowledge engineering.

But in order for agents to coordinate their activities and cooperate in solving mutual problems, it is essential that they be able not only to communicate with each other, but also to assess and maintain the integrity of the communicated information, as well as of their own knowledge. Consistency maintenance is thus crucial. However, there are many types of inconsistencies that can arise among a group of agents, such as

1. one agent could believe a datum, while another agent could disbelieve it;
2. one agent could believe a datum, while another agent could believe its negation, and these beliefs could be used to justify a datum they share;
3. two agents could believe an object to be of two incompatible types, i.e., they could use different terms for the same object;
4. two agents could believe two different objects are of the same type, i.e., they could use the same term for two different objects;
5. the agents' beliefs may be inconsistent at a semantic level, e.g., one agent could believe that an object is made out of plastic, while another believes that it is made out of steel.

What technology is needed to achieve the requisite consistency among the different information resources?

Distributed truth maintenance: There are many desirable properties for a knowledge base, such as completeness, conciseness, accuracy, and efficiency. For an agent that can reason nonmonotonically, there are additional properties used to describe the *integrity* of the knowledge base: stability, well-foundedness, and logical consistency. In addition, any algorithm that attempts to maintain well-founded stable states of a knowledge base should be *complete*, in that the algorithm should find a well-founded stable state if it exists. We desire each agent in a

multiagent environment to have a complete algorithm for maintaining the integrity of its own knowledge base.

Truth maintenance systems are a common way to achieve knowledge base integrity in a single agent system, because they deal with the frame problem, they deal with atomicity, and they lead to efficient search. Furthermore, the justification networks they create can be used for nonmonotonic reasoning, problem-solving explanations to a user, explanation-based learning, and multiagent negotiation.

However, the above definitions of properties for a single knowledge base are insufficient to characterize the multiple knowledge bases in a multiagent environment. When agents that are nonmonotonic reasoners exchange beliefs and then make inferences based on the exchanged beliefs, then concepts of *distributed* knowledge-base integrity are needed.

Nonmonotonic reasoning: Agents need to be able to maintain independent viewpoints and skepticism until they receive convincing evidence otherwise, but they should then be able to revise their beliefs consistently.

Negotiation: A few researchers have explored negotiation as a means to mediate among conflicting agents. The systems they developed involved either monotonic reasoners, or nonmonotonic, but memoryless, reasoners, i.e., reasoners that simply discard old solutions and re-solve in the face of conflicts. Negotiation is likely the correct approach, but the agents must be able to revise their plans incrementally as they interact. They must be able to communicate directly, with each other and with human agents, and they must be able to assess and maintain the integrity of both the communicated information and their own knowledge. Then they can successfully coordinate their activities and cooperate in solving mutual problems.

Semantic integration: Where the semantics of a resource are expressed (partially) in the form of data dictionary or schema information, this information must be interrelated with the semantics of the other resources through the use of class servers or global schemas, such as the Cyc knowledge base. It is essential that a common semantics be available and provided computationally.

Federated databases: Where it is necessary to retain the autonomy of individual information resources, mappings must be generated to yield interoperability.

Database management systems for design: Design DBMSs are needed that support large and long-duration transactions, relaxed transactions, large structured composite objects, versions, and aggregation.

Intentionality: Representations for agents and their actions must be developed that can express their intentions and commitments through communicative acts.

Background

Knowledge-based systems have become an important part of computing. There are estimates of over 100,000 fielded systems to date. These systems are mostly small, independent, and developed for specific applications using off-the-shelf expert system shells. These shells are most suitable for monolithic applications involving the knowledge of a single human expert. But applications in larger and more complicated domains, and attempts to use several small systems in concert when their application domains overlap argue for knowledge-based systems to be developed in a modular and distributed fashion.

Early attempts to develop systems of cooperating agents, involved agents with independent knowledge bases. The independence was achieved by restricting agent interactions to modifications of a global data structure—a blackboard—and by minimizing overlap in the agents' knowledge. Later systems allowed richer agent interactions and overlapping knowledge, but the agents were required to have consistent knowledge and to reason monotonically. This led to representational problems, because different experts in the same domain often have different perspectives and *conflicting* knowledge, making it difficult to construct a coherent problem-solving system for that domain.

MCC is addressing the above problems through the development of *RAD* [Arni *et al.* 90]. *RAD* enables a set of knowledge-based systems, constructed independently, to act as a set of cooperating agents, working together to solve a problem. Developers of distributed reasoning systems can exploit a

divide-and-conquer approach to development; they can build smaller, more manageable knowledge-based agents. These smaller agents might represent alternative points of view on a problem; there is no longer a need for global consistency across an entire large system. These smaller agents can also be *reused* in different combinations for solving additional problems as they arise. They can be physically distributed in the world, just as the problems that they address are. The intelligence needed for such problems can be embedded throughout a computer network and made available where appropriate.

The *RAD* agents operate within this network asynchronously and, in general, autonomously. *RAD* permits the collection of agents to be dynamic, allowing agents to come and go. The agents can be either reactive or ingenuous, i.e., they can either respond to questions and commands from another agent or initiate dialogs with another agent. *RAD* also allows other types of agents, including OPS5 expert systems and human agents, to interact with *RAD* agents.

RAD incorporates a distributed TMS [Huhns & Bridgeland 91] that allows each agent to rely on the results of another's reasoning without having to keep track of the details of that reasoning. However, there is no requirement for two agents to agree completely. The DTMS enforces local consistency within each agent, while enabling negotiation about inconsistencies among agents. When two or more agents disagree about belief in a datum *and when this disagreement is encountered during problem solving*, then negotiation among the agents will ensue to resolve the disagreement. The negotiation procedure involves an exchange of justifications among the agents. The negotiation is necessary to ensure that the global solutions to the problems posed to the agents are coherent.

The DTMS is agnostic about what data should be shared among agents. The research of [Courand 90] and [Galliers 90] has produced principles governing the incorporation of data from other agents. In [Courand 90], agents share goals and plans in order to achieve the mutual beliefs necessary to take cooperative action, but only when the resultant belief system will be more coherent. In [Galliers 90], the agents are skeptical, rather than cooperative, and prefer to adopt beliefs that reinforce existing beliefs without revising any. The Rational Distributed Reason Maintenance System [Doyle & Wellman 90] similarly suggests a basis for deciding rationally which beliefs and plans to revise.

References

[Arni *et al.* 90] Natraj Arni, et al., "Overview of RAD: A Hybrid and Distributed Reasoning Tool," MCC Technical Report No. ACT-RA-098-90, Microelectronics and Computer Technology Corporation, Austin, TX, March 1990.

[Huhns & Bridgeland 91] Michael N. Huhns and David M. Bridgeland, "Multiagent Truth Maintenance," *IEEE Transactions on Systems, Man, and Cybernetics*, December 1991.

[Courand 90] G. J. Courand, "Cooperation Via Consensus Formation," *Proceedings of the 10th International Workshop on Distributed Artificial Intelligence*, Bandera, TX, Chapter 10, MCC Technical Report No. ACT-AI-355-90, October 1990.

[Doyle & Wellman 90] J. Doyle and M. P. Wellman, "Rational Distributed Reason Maintenance for Planning and Replanning of Large-Scale Activities (Preliminary Report)," *Proceedings DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, San Mateo, CA: Morgan Kaufmann, November 1990, pp. 28–36.

[Galliers 90] J. R. Galliers, "Cooperative interaction as strategic belief revision," *Proceedings of the International Working Conference on Cooperating Knowledge-Based Systems*, Keele, England, October 1990, pp. 148–163.

[Gasser & Huhns 89] Les Gasser and Michael N. Huhns, eds., *Distributed Artificial Intelligence, Volume II*, Pitman Publishing, London, 1989.

[Huhns 87] Michael N. Huhns, ed., *Distributed Artificial Intelligence*, Pitman Publishing, London, 1987.