

Contexts: Dynamic Identification of Common Parameters in Distributed Analysis of Complex Devices

Sanjaya Addanki, Roberto Cremonini, J. Scott Penberthy

IBM T. J. Watson Research Center

PO Box 704

Yorktown Heights, NY 10598

e-mail: jsp@ibm.com

Abstract

An important aspect of Distributed Problem Solving is that multiple intelligent agents understand each others communications. In our approach, domains (Thermodynamics, Mechanics, Fluids, etc.) are independent problem solving modules that interact by resolving constraints on parameters common to each other. Since device behaviors change through time, these parameters appear and disappear through an analysis session. Further, each domain uses domain-specific languages for problem-solving efficiency and human usability. Hence, the communication paradigm must dynamically translate parameters from one domain to another with no *a priori* knowledge of what parameters may appear and what domains may be required during a session. Fortunately, parameters in applied physics carry a lot of spatial, temporal, and structural knowledge with them. In this paper we introduce the notion of *contexts* as an encapsulated description of this knowledge. We also show how contexts can be used efficiently to provide matching of parameters across domains.

1 Introduction

Cooperation and communication between agents is an important issue in distributed problem solving ([Corkill and Lesser, 1983], [Georgeff, 1983], [Konolige, 1980], [Durfee *et al.*, 1987], [Erman *et al.*, 1980], etc.). An agent in a distributed problem solving system must determine the other agents with which it must communicate, the types of information that should be shared, and how this information can be communicated. Making these determinations depends on the fact that an agent must understand communications from other agents. Previous approaches to this can be divided into three categories (or combinations thereof): *common language*, *shared memory*, and *static multiple language* solutions.

Common language approaches (e.g. [Hewitt and Kornfeld, 1980]) assume that all agents work in the same language. In shared memory solutions, the semantics of a communication are derived from its location in the shared memory ([Erman *et al.*, 1980], [Konolige, 1980]).

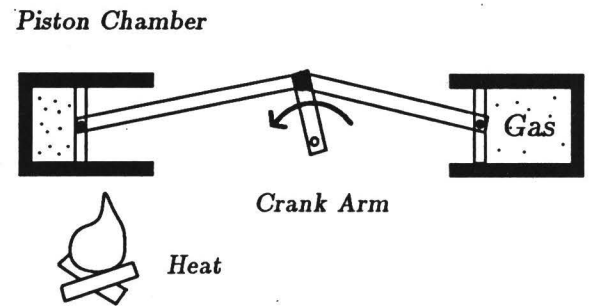


Figure 1: An Ideal Heat Engine

Static multiple language solutions assume that it is possible to dynamically translate assertions in one language to assertions in another (e.g. [Smith *et al.*, 1981]).

Unfortunately, these solutions are inadequate for systems that perform analysis in complex engineering domains ([Penberthy, 1987], [Addanki and Davis, 1985]). A behavioral description of the ideal heat engine in figure 1 requires analysis within the domains of Thermodynamics, Mechanics, and Geometry. In our approach, each domain is an active module that consists of a declarative, logical database of the knowledge in the domain, backward and forward chainers to perform inference in the domain, and domain specific heuristics that control this inference. The Thermodynamics domain gives the behavior of the gases in the piston-cylinder assemblies, Mechanics gives the kinematic and dynamic behaviors of the pistons, links, cranks, etc., and Geometry provides the structural constraints and computations.

Analysis of the heat engine requires that these different domain-specific analyses interact in order to find the final result. The interaction is a result of the different domains sharing common parameters. Each domain defines its own constraints on the common parameters and the interaction resolves these constraints giving rise to the final solution. For example, the volume of the gas is a parameter common to Thermodynamics and Geometry, the position of the piston is a parameter common to Geometry and Mechanics, etc.

Since device behaviors change through time, parameters are both problem- and time-specific. As behaviors change, old parameters disappear and new parameters

are created. This dynamic nature of common parameters makes it difficult to use either a common language, a shared memory semantics, or a language translator. Even though domains may use the same logical language, they use different extensional variable names since parameters are both problem- and time-specific. As a result, the domains would have to perform a multi-phase commit to agree with one another on all global name assignments, clearly a tedious approach. Shared memory approaches require all common locations (parameters, in our case) to be predefined, a difficult problem when parameters come and go dynamically. Lastly, matching common parameters requires translations to take place over the extensions of the different languages. The dynamic nature of the extensions, when combined with the problem of multiple extensional names, makes a *a priori* specification of the translation tables difficult.

Fortunately, parameters in applied physics carry a lot of spatial, temporal, and structural knowledge with them. This knowledge can be expressed in a common sublanguage. Although this sublanguage is dynamic, it is possible to carry out translation over its extensions. In this paper we introduce the notion of *contexts* as the encapsulated description of a parameter in this common sublanguage. We show how contexts can be used efficiently to provide matching of parameters across domains. We begin by defining the structure of contexts. We then show how contexts can be matched in order to determine common parameters across domains. By means of an example we demonstrate the utility of contexts. Finally, we discuss our work in relation to other communication paradigms in distributed problem solving.

2 What's in a Context?

A context is defined for each parameter and consists of four parts: (1) related spatial information, (2) the parameter's dimension (e.g. *length*, $\frac{\text{mass} \cdot \text{length}}{\text{time}}$), (3) temporal information, and (4) units of measure. This section describes each of these parts in detail.

Every physical quantity represented by a parameter¹ is associated with an underlying geometric or structural entity. It is impossible to define a quantity without referring to this structural entity. For example, a "force," by its very definition, requires the existence of an agent and a target [Martin, 1982]. The agent is either a physical object or a field and the target is another object. Hence, these objects (or fields) are the structural entities that must be mentioned when defining a force.

Properties in Thermodynamic systems, as another example, are either *intensive* or *extensive* [Moran and Shapiro, 1988]. Intensive properties are functionally dependent upon position, a geometrical entity. Examples of these properties include pressures, temperatures and densities. Extensive properties are functionally dependent upon the structural boundaries that define the system; such boundaries comprise a structural entity. For example, the volume of a gas (an extensive property) is

the volume of its container. The structural "entity" associated with the gas volume would be the surfaces that define the container's inner boundaries.

In addition to structure, the *dimension* of parameters is important. A set of dimensions that suffices for applied physics includes {*Mass*, *Time*, *Length*, *Temperature*, *Electrical Current*}. All physical quantities in applied physics can be measured in terms of these dimensions (i.e. {[*M*], [*T*], [*L*], [*K*], [*I*]}). Parameters in one domain can be compared with parameters in another domain *if and only if* they have the same dimensions. Obviously, temperatures (dimension [*K*]) must be compared with temperatures, forces (dimension $\frac{[M] \cdot [L]}{[T^2]}$) with forces, etc.

Dimension and structure can obviously disambiguate between parameters that refer to different objects. Sometimes, however, we need to disambiguate between two or more parameters with the *same* dimension that refer to the *same* object. The *length* and *height* of a rectangular object, for example, share both the same object and the same dimension. In this case we depend upon an unambiguous structural description to discriminate between them; just identifying the "object" of both parameters is insufficient. In general, the structural description for a 3d-segment (volume) of an object is specified by the 2d-boundaries (surfaces) that surround it. Similarly, a 2d-patch (surface) of an object is specified by its 1d-boundaries (segments). Finally, any 1d-segment (edge, arbitrary line segment) of an object is specified by its 0d-boundaries (endpoints). These points are described in an object-specific coordinate system.² By using these descriptions we can differentiate parameters, such as *length* and *height*, that refer to different pieces of the same object.

We also are interested in analyzing devices whose behavior changes through time, devices whose parameter values may vary through time. One must therefore ensure that when domains compare the values for parameters they compare the values at the same point (or interval) in time. Thus, time must be included in the description of a physical quantity to disambiguate temporally separate measurements.

Lastly, the type (scalar or vector) and units of measure for each parameter are added to contexts. This information is used to perform mathematical conversions of the values passed between domains. Type information ensures that values are handled in an *appropriate* way. For example, to compare two scalar values, one simply checks their value in the same system of measurement. Comparing vectors, on the other hand, requires three steps. First, the vectors must be translated to a common frame of reference. Second, the vectors must be converted to a common representation (e.g. polar vs. cartesian). Third, the vector coordinates must be converted to a common system of measurement. Units of measure are needed to support the different sys-

¹From now on we will use the terms *quantity* and *parameter* interchangeably.

²As seen later in this section, these coordinate systems may differ. We provide a mechanism for converting between these coordinate systems using standard analytic geometry formulae.

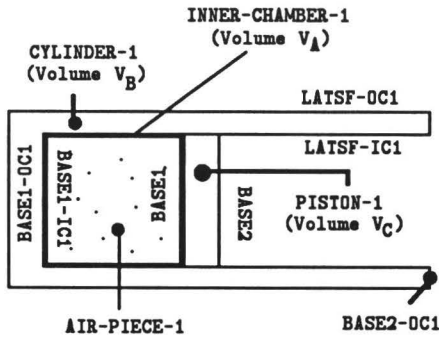


Figure 2: Structure of the Piston Cylinder Assembly

tems of measurement adopted by each domain of applied physics (*i.e.* SI, CGS or EES)[Martin, 1982].

These four types of information (space, dimension, type, and units) unambiguously define parameters of physical systems. As a result, if two contexts match they describe the same parameter. The task of identifying common parameters between domains is therefore reduced to the task of finding contexts from one domain that match contexts of another domain.

3 Using Contexts

Now that contexts have been described in general, this section will illustrate how contexts are used to effectively determine common parameters across domains. In particular, this section shows how the domains of Thermodynamics and Geometry use contexts to determine common volume parameters in the ideal heat-engine of figure 1. Each of these domains first generates its own set of contexts and then matches them against contexts from the other domain.

3.1 Generating Contexts

To begin, suppose that heat is applied to the first piston chamber from time t_1 to time t_2 . Thermodynamics models the gas behavior as a polytropic process, deducing that its volume must increase during time interval $[t_1, t_2]$. The following propositions are used by Thermodynamics to represent these changes in volume (*e.g.* from $0.1m^3$ to $0.2m^3$):

(gas-volume air-piece-1 v1 T1) $v1=0.1m^3, T1=t_1$
 (gas-volume air-piece-1 v2 T2) $v2=0.2m^3, T2=t_2$

Contexts for these propositions must now be created. The structural context for a gas volume is the set of surfaces that define a region of space which exactly contains the gas. By examining the heat engine, Thermodynamics finds that the gas is contained by the surfaces of the inner chamber (*i.e.* the surfaces base1-ic1, base1, and latsf-ic1). This structural information is contained in the global structure specification for the piston-cylinder assembly (see figure 2).

The dimension and temporal information for each parameter must also be included. Thermodynamics knows that the dimension for volumes is $[L^3]$. The temporal extent for each volume parameter is contained in the corresponding proposition [Penberthy, 1988]. The resulting contexts for the gas-volume propositions can be seen in figure 3.

(gas-volume air-piece-1 v1 T1)	
Structure:	base1-ic1, base1, latsf-ic1
Dimension:	$[L^3]$
Unit, Type:	m^3 , scalar
Time:	t_1
Value:	0.1

(gas-volume air-piece-1 v2 T2)	
Structure:	base1-ic1, base1, latsf-ic1
Dimension:	$[L^3]$
Unit, Type:	m^3 , scalar
Time:	t_2
Value:	0.2

Figure 3: Volume Contexts in the Thermodynamics Domain

The Geometry domain also computes volumes, but it uses a different language and a different set of rules. The Geometry domain determines three distinct volumes for each chamber in the heat-engine example (see figure 2). Volume V_A is the one bounded by the inner piston surface and the inner surfaces of the cylinder housing. This volume depends on the position of the piston relative to the cylinder housing. Volume V_B is the one for the material needed to build the cylinder. Volume V_C is the volume for the material needed to build the piston.

These three volumes are represented by the Geometry domain with the following propositions:

(volume-of inner-chamber-1 V_A I1) $V_A = 0.1m^3$
 (volume-of cylinder-1 V_B I1) $V_B = 0.09m^3$
 (volume-of piston-1 V_C I1) $V_C = 0.03m^3$
 I1= $[t_1, t_2]$

The Geometry domain must convert these propositions into contexts before trying to communicate with other domains. Since every volume is defined by a unique set of bounding surfaces, these surfaces make up the structural context for a volume. As before, this information is written explicitly in the problem description (see figure 2) and is easily retrieved. The Geometry domain knows that all volumes have dimension $[L^3]$, and it knows the temporal extent of each parameter by examining the corresponding propositions. This information is collected for the three volumes and then formed into separate contexts as seen in figure 4.

(volume-of inner-chamber-1 V_A I1)	
Structure:	base1-ic1, base1, latsf-ic1
Dimension:	$[L^3]$
Unit, Type:	m^3 , scalar
Time:	$[t_1, t_2]$
Value:	0.1

(volume-of cylinder-1 V_B I1)	
Structure:	base1-oc1, base1-ic1, latsf-ic1, latsf-oc1, base2-oc1
Dimension:	$[L^3]$
Unit, Type:	m^3 , scalar
Time:	$[t_1, t_2]$
Value:	0.09

(volume-of piston-1 V_C I1)	
Structure:	base1, base2, latsf-p1
Dimension:	$[L^3]$
Unit, Type:	m^3 , scalar
Time:	$[t_1, t_2]$
Value:	0.03

Figure 4: Volume Contexts in the Geometry Domain

At this point each domain has calculated values for its volume parameters and has generated the corresponding contexts. The domains must now compare their contexts, look-

ing for contexts whose structure, dimension, and time values match.

3.2 Matching Contexts

The algorithm for matching contexts consists of three phases: "canonicalize," "hash," and "temporal match." In the *canonicalize* phase each domain ensures that the structure and dimension portions of each context are in canonical form.³ Since the structural elements of a device are uniquely named and common to all domains, an alphabetized string is used for the canonical representation of a structural set. For example, the set of structural elements from Geometry's context for V_A , {base1-ic1, base1, latsf-ic1}, is represented by the canonical form "base1 base1-ic1 latsf-ic1". The canonical form for a given dimension is an expression in a tree form, identical to that used in [Martin and Fateman, 1971]. Since the current implementation uses *Macsyma* for calculations, each domain already uses this representation for dimension equations; no extra conversion work is needed.

Assuming that the Geometry domain transmits its canonicalized contexts to the Thermodynamics domain and *vice versa*, the *hash* phase of the matching algorithm begins in both domains. In this phase a hash-sort of contexts is used. The key for each context is an ordered pair, consisting of the context's dimension followed by its structure. Since all the elements of these lists are in canonical form, the buckets of the hash table contain contexts whose dimension and structure values match. In our example three buckets (represented by the following hash keys) contain contexts:

1. ($[L^3]$, "base1 base1-ic1 latsf-ic1")
2. ($[L^3]$, "base1 base2 latsf-p1")
3. ($[L^3]$, "base1-ic1 base1-oc1 base2-oc1 latsf-ic1 latsf-oc1")

The *temporal match* phase of the algorithm finds all context pairs in the hash buckets whose time values overlap. Currently we use the standard $O(N^2)$ search, where N is the number of contexts within a given bucket, to match the temporal components of local and remote contexts. The temporal comparisons are done by referring to a global partial order of timepoints, as described in [Penberthy, 1988]. In our example, only bucket 1 (as defined above) contains matching contexts. Two pairs are found in this bucket, identifying Geometry's volume V_A and Thermodynamic's volumes V_1 and V_2 as common parameters. Geometry and Thermodynamics can now exchange values for these volumes, possibly using the unit conversion techniques described earlier.

Other examples of context pairs exist in the heat engine example.⁴ The domains of Kinematics and Geometry find the position of the piston and crank-arm assemblies as common parameters. The Kinematics domain discovers that the Solid Dynamics domain shares an interest in the acceleration of all mechanical linkages. Solid Dynamics and Thermodynamics exchange values for the forces exerted by the expanding gases. All of these common parameters are identified by matching their structural context, their dimensions, and their time values. In addition, the use of units and type information is especially useful when Kinematics uses polar coordinates to represent accelerations and Dynamics uses Cartesian coordinates for the same parameters (e.g. acceleration of the crank arm).

³We canonicalize both the set of structural elements and the dimension formula to provide for efficient matching.

⁴Due to a lack of space, the actual matching processes for these contexts are not described in detail.

4 Implementation

The language used by each domain is built from a general first-order temporal language as described in [Penberthy, 1988]. This general language also supports an interface to *Macsyma* ([Martin and Fateman, 1971], [Symbolics, Inc., 1988]) for performing the symbolic mathematical calculations required by each domain.

Contexts are implemented as frame-like structures. Each domain translates its language descriptions into contexts by using pre-defined functions. As demonstrated in the example, these functions perform two tasks. First, they perform syntactic extraction from the individual language descriptions for the time, dimension, and value sections of a context. Second, they unify simple clauses in the domain knowledge bases to complete the structural section of a context. Since the structural specification is global to all domains, these unifications are reduced to the matching of conjunctions against a database of facts. No inference is required.

A framework for distributed problem solving has been implemented to support the passing of contexts between domains. This framework consists of several LISP machines, each with its own implementation of a particular domain of applied physics. As of this writing, four domains have been implemented and tested: Thermodynamics of Ideal Gases, Planar Solid Dynamics, Planar Kinematics, and Planar Geometry. These domains communicate with one another using a network protocol specifically designed for exchanging contexts. This protocol supports multiple machines (e.g. domains) connected together on a LAN.

5 Discussion

Many researchers have recognized the importance of interaction between cooperating agents in a distributed problem solving system (e.g. [Corkill and Lesser, 1983], [Georgeff, 1983], [Konolige, 1980], [Durfee *et al.*, 1987], [Erman *et al.*, 1980]). However, many of the proposed styles of interactions are typically *static* in nature. By static, we mean that the different problem solvers in the system know *a priori* the classes of data that must be exchanged and to whom or where the data must be sent.

The Hearsay II architecture proposed in [Erman *et al.*, 1980] is typical of many distributed problem solving systems [Durfee *et al.*, 1987], and is an excellent example of static interaction. It consists of multiple knowledge sources, a blackboard divided into separate sections, and meta-level demons. Each section of the blackboard contains the data relative to a specific task performed by the architecture and is statically defined. Knowledge sources interact with one another by reading and writing data to these sections. However, since the blackboard sections are static, the types of interaction between knowledge sources are static.

Suppose that one wanted to build a system for analyzing mechanical devices by using a similar architecture. In this approach each knowledge source would represent a given domain of applied physics. Since these knowledge sources would necessarily exchange values for common parameters, the blackboard should contain a unique location for each parameter. Unfortunately, the task of specifying blackboard locations for parameters is quite difficult. This difficulty arises from the fact that the set of parameters used to analyze a device often changes during the analysis.

Analysis requires multiple sets of parameters since a given set often depends upon the structural configuration of a device. Since a device may undergo changes in configuration, the corresponding parameter set might change. For example, the gas volumes within the heat engine of figure 1 depend on

the respective pistons being inside their cylinder housings. If a piston were to leave its housing, the parameters describing the corresponding gas volume would be invalid. The piston might also collide with other parts of the heat engine, requiring new parameters to describe the new forces and accelerations caused by collision. If a problem solving system were to analyze such a behavior, it would start with a set of parameters that included gas volumes, switch to one without gas volumes, then switch to a set describing the interaction of falling pistons and other phenomena.⁵

This continual change in parameters would pose serious difficulties for a blackboard system that depends upon statically declared interaction. The blackboard would necessarily undergo a dynamic reconfiguration for each new parameter set. Further, this reconfiguration would have to be known to all knowledge sources. While this could be implemented, such overhead would be computationally expensive. Static declarations of interaction, therefore, are not feasible for distributed problem solving systems that analyze mechanical devices.

Dynamic interaction, where the individual problem solvers are free to change communication patterns, is needed. A few such reconfigurable systems (*e.g.* [Hewitt and Kornfeld, 1980], [Smith and Davis, 1981], [Findler and Gap, 1987]) have been proposed. Hewitt *et al.* describe the concept of actors and message passing, and how this metaphor can be used in problem solving. Hewitt proposes a fine-grain parallel architecture of actors that can change dynamically. He also defines the semantics for messages passed between these actors. One might think of contexts as the semantics for messages passed between domains.

In [Smith and Davis, 1981] and [Smith and Davis, 1983] Smith *et al.* recognize the utility of dynamic interaction between multiple nodes (problem solvers). In particular, they identify *result sharing* as a form of cooperation in which individual nodes assist one other by sharing partial results, based on somewhat different perspectives of the overall problem. Result sharing permits the dynamic exchange of common results but, as the authors point out, implementing this requires a solution to several communication problems. These problems demonstrate that a node must (1) select particular results to be transmitted, (2) choose the nodes that will receive these results, and (3) upon receipt of a result, decide whether or not to accept it. The concept of a context, as introduced in this paper, is the key to solving these communication problems in the domain of applied physics.

Similarly, Riesbeck in [Riesbeck, 1979] identifies a solution (known as *conceptual dependency* (CD) structures) for communication problems in a distributed system for story understanding. CD structures represent conceptual meanings and, like contexts, are exchanged between problem solvers. Whereas CD structures are designed for communications between entities in a natural-language understanding system, contexts are designed solely for communications in systems that analyze engineering systems.

6 Conclusions and Future Work

In this paper we described *contexts*, a representation that permits the dynamic identification of common parameters between distributed problem solvers in a system that analyzes mechanical devices. A context represents the structural elements, temporal extent, dimensions, and units associated with each physical parameter. The four concepts described in

contexts are inherent in all domains of applied physics. Each domain therefore can use a language particular to its area of expertise, without being restricted by a global, common language for *all* concepts.

Contexts provide the ability to determine, for each problem, the parameters that one domain must exchange with another. However, this is only the first step in distributed analysis of engineering systems. In the future we will explore the utility of contexts and related structures in situations arising from dynamic interaction. For example, one must determine what actions to take when conflicting parameter values are detected. This would require a mechanism to determine which domain is in error, *i.e.*, a mechanism for comparing justifications across domains. This problem is further complicated by the use of different languages in different problem solvers. In addition to the resolution of conflicts, one must also provide devices for the exchange of partial solutions, for queries between domains, etc. Contexts provide a framework for these and related tasks, but the details have not been formalized. Nevertheless, all of this work relies on contexts. Without contexts, domains could not identify common information easily. Without the exchange of common information, complex devices could not be analyzed.

References

- [Addanki and Davis, 1985] S. Addanki and E. S. Davis. A Representation for Complex Physical Domains. In *IJCAI-85*, Los Angeles, California, 1985.
- [Corkill and Lesser, 1983] D. D. Corkill and V. R. Lesser. The Use of Meta-Level Control for Coordination in a Distributed Problem Solving Network. In *IJCAI-83*, Karlsruhe, West Germany, 1983.
- [Durfee *et al.*, 1987] E. H. Durfee, V.R. Lesser, and D.D. Corkill. Cooperation Through Communication in a Distributed Problem Solving Network. In M.N. Huhns, (ed), *Distributed Artificial Intelligence*, Morgan Kaufmann Publishers, London, Great Britain, Inc., 1987.
- [Erman *et al.*, 1980] L.D. Erman, F. Hayes-Roth, V. Lesser, and R. Reddy. The HEARSAY-II Speech Understanding System. In *Computing Surveys*, vol. 12, no. 2, 1980.
- [Findler and Gap, 1987] N.V. Findler and J. Gao. Dynamic hierarchical control for distributed problem solving. In *Data and Knowledge Engineering*, vol. 2, 1987.
- [Genesereth and Nilsson, 1988] M.R. Genesereth and N.J. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann Publishers, Inc., Palo Alto, California, 1988.
- [Georgeff, 1983] M. Georgeff. Communication and Interaction in Multi-agent Planning. In *AAAI-83*, Washington, D.C., 1983.
- [Hayes-Roth, 1985] B. Hayes-Roth. A Blackboard Architecture for Control. In *Artificial Intelligence*, vol. 26, 1985.
- [Hewitt and Kornfeld, 1980] C. Hewitt and B. Kornfeld. Message Passing Semantics. In *SIGART Newsletter*, October 1980.
- [Konolige, 1980] K. Konolige. *A First-order Formalization of Knowledge and Action for a Multi-agent Planning System*. Tech Note 232, SRI International, Menlo Park, California, 1980.
- [Martin, 1982] G.H. Martin. *Kinematics and Dynamics of Machines*. McGraw-Hill Book Company, New York, New York, 1982.

⁵ Although this is an unusual example, any mechanical device that involves changes in configuration (*e.g.* typewriters, cam shafts, locks) would require multiple sets of parameters.

- [Martin and Fateman, 1971] W.A. Martin and R.J. Fateman. The MACSYMA System. In *Proceedings of the Second ACM Symposium on Symbolic Algebraic Manipulation*, Los Angeles, California, March, 1971.
- [Moran and Shapiro, 1988] M.J. Moran and H.N. Shapiro. In *Fundamentals of Engineering Thermodynamics*. John Wiley & Sons, Inc., New York, 1988.
- [Penberthy, 1988] J.S. Penberthy. Temporal Unification and the Temporal Partial Order In *Proceedings of the Fifth IEEE Conference on Artificial Intelligence Applications*, San Diego, CA, 1988.
- [Penberthy, 1987] J.S. Penberthy. *Incremental Analysis and the Graph of Models*. MIT S.M. Thesis, Cambridge, Massachusetts, 1987.
- [Riesbeck, 1979] C.K. Riesbeck. Representations to Aid Distributed Understanding in a Multiprogram System. In N. Findler, (ed), *Associative Networks: Representation and Use of Knowledge by Computers*, Academic Press, New York, 1979.
- [Smith *et al.* , 1981] J.M. Smith. *et.al.*. Multibase — Integrating Heterogeneous Distributed Database Systems. In *Proceedings of AFIPS NCC*, 1981.
- [Smith and Davis, 1981] R.G. Smith and R. Davis. Frameworks for Cooperation in Distributed Problem Solving. In *IEEE Transactions on Systems, Man, and Cybernetics*, vol. smc-11, no. 1, January 1981.
- [Smith and Davis, 1983] R.G. Smith and R. Davis. Negotiation as a Metaphor for Distributed Problem Solving. *Artificial Intelligence*, vol. 20, 1983.
- [Symbolics, Inc., 1988] Symbolics, Inc. *MACSYMA Reference Manual*. ver. 13, doc. no. SMI0500030.013, 1988.