

A Collaborative Parametric Design Agent

Daniel Kuokka and Brian Livezey

Lockheed Palo Alto Research Laboratories

Orgn 96-20, Bld 254F

3251 Hanover Street

Palo Alto, CA 94304

kuokka@aic.lockheed.com, livezey@aic.lockheed.com

Abstract

ParMan combines the use of agent communication protocols, constraint logic programming, and a graphical presentation interface to yield an intelligent parametric design tool supporting collaborative engineering. This provides one of the first complete, end-to-end applications of distributed knowledge-level communication among engineering tools, as envisioned by PACT (Cutkosky *et al.* 1993). In addition, it represents a significant extension of parametric design to a distributed setting. This paper describes the underlying technologies of ParMan, based on CLP(R), the Knowledge Query and Manipulation Language, and knowledge-based facilitation agents.

Introduction

Parametric design is a common and important class of design. Numerous systems and approaches have been developed to address this problem (Bouchard 1992; Frayman & Mittal 1987; Kolb 1989), but they have largely ignored a fundamental issue: the constraints typically come from multiple sources, making parametric design a collaborative task. ParMan is a distributed parameter management system that addresses this basic omission by coupling a parametric design tool, based on constraint logic programming (Jaffar & Lassez 1987; Jaffar *et al.* 1992), with an agent-based collaborative engineering infrastructure (Cutkosky *et al.* 1993; McGuire *et al.* 1993). The merged functionality is presented to the user in a highly intuitive fashion via a specialized graphical user interface. The result is an intelligent parametric design tool supporting collaborative engineering.

Ironically, just as ParMan introduces collaboration into parametric design, it proposes a much needed process for the agent-based engineering community. Work on agent infrastructures for engineering has brought previously isolated tools on-line, allowing a high degree of knowledge sharing among design tools. However, the technology for controlling such highly integrated tools has not kept pace with the integration technology. When should changes be transferred to the network of agents: as they are made, at some intermediate check

point, upon file saving, or upon version update? ParMan provides an interface for controlling when various aspects of the local design are made public, in addition to providing a means of specifying and testing parametric constraints. Thus, ParMan provides one of the first systems to support the design process by utilizing distributed yet highly integrated tools.

There is a growing body of work in collaborative engineering design systems and distributed AI such as (Birmingham *et al.* 1993; Pan & Tenenbaum 1991; Petrie 1993; Saad & Maher 1993; Sriram 1993; Weber *et al.* 1992; Werkman 1992). ParMan is notable in that it emphasizes parametric design as the single formal model for collaboration. Work on intelligent software agents, such as (Dent *et al.* 1992; Maes & Kozierok 1993) is similar in spirit, but these systems tend to emphasize autonomy and learning instead of communication among agents. In this respect, the area of computer supported cooperative work (Reeves & Shipman 1992; Stefik *et al.* 1987) is very relevant.

Throughout this paper, the term “agent” is used to refer to a semi-autonomous participant in a distributed design scenario. Agents typically are not stand-alone software systems, but consist of a tool and a human user. Agents are semi-autonomous in that they act spontaneously and dynamically according to local goals, yet they must contribute to the joint problem solving effort, so they cannot act without consideration of other agents.

The remainder of this paper is organized as follows: the next section gives a user view of ParMan, focusing on an example. Next, a system view of ParMan is presented, covering algorithms and implementation issues. Finally, we evaluate ParMan based on experiments in several domains.

User View

An engineer uses ParMan to define and analyze constraints over a set of parameters. As parameters and constraints are entered, ParMan interacts with other agents, maintaining a distributed constraint set that is presented back to the local user. Other agents are usu-

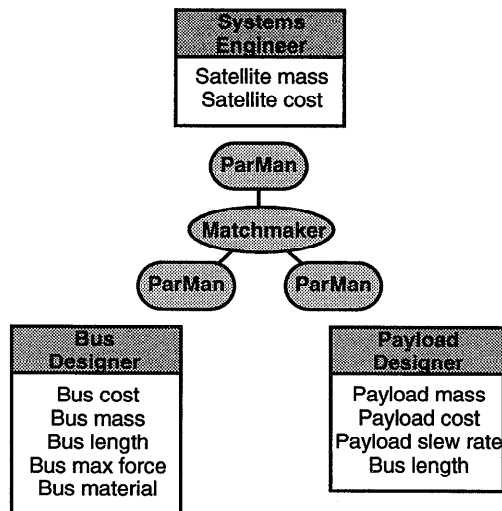


Figure 1: Each participant in the design task uses ParMan to communicate constraints on specific parameters.

ally ParMan users as well, but the underlying protocols are quite general, allowing any conformant software system to participate. Thus, ParMan may interact with a broad range of CAD tools, either autonomous or human guided, as long as they can communicate parameter constraints.

ParMan is illustrated via an example from the Lockheed FSAT (Frugal Satellite) program, an effort to design a simple, inexpensive, reusable satellite. The satellite consists of a simple tubular bus with a payload. Imagine an Integrated Product Design (IPD) team working on the conceptual design of the satellite. The team consists of several participants: a systems engineer, a bus designer, and a payload designer. Each participant has specific interests, shown in Figure 1 (the Matchmaker, described in the Agent Interface section, is used by ParMan to route messages).

ParMan aids in the conceptual design of the satellite as follows. First, the engineer uses the Project Selector to choose from a list of all projects that are of interest to members of the design team (see Figure 2). The list is populated dynamically by the distributed set of ParMan users, consistency being maintained by the exchange of KQML messages such as *advertise*, *subscribe*, *tell*, and *ask* (see the Agent Interface section). Thus, all users are kept apprised of all projects and can collaborate on any or all of them. In the example at hand, each team member selects the FSAT project.

Once a project is selected, each team member uses the Parameter Graph Editor to define a set of parameters of specific interest (also in Figure 2). The graphical structure allows the user to define components and associate parameters with those components. When a

parameter or component is placed in the graph, ParMan asserts its existence to the other agents (users may also define private parameters). Upon receipt of a parameter assertion, ParMan adds it to the Parameter Graph Editor, using a different color from that used for parameters specified by the local user. This allows the user to identify forgotten or inconsistently-named parameters.

From the Parameter Graph Editor, each user selects a subset of the parameters to be displayed in the Parameter Table, a worksheet of parameters relevant to his aspect of the design (see Figure 3). Notice that the hierarchical namespace implicit in the Parameter Graph Editor is carried over to the Parameter Table.

The user can now begin defining constraints over the parameters in the Parameter Table. These constraints are expressed in ParMan's Constraint Language, which includes standard infix arithmetic constraints and an extensible set of predicates. Constraints are entered via the Constraint Editor (also shown in Figure 3), which is brought up by selecting a Parameter Table cell. Equality constraints (e.g., *bus.length = 3m*) appear as entries directly in the Parameter Table; otherwise, "***" appears. As constraints are entered, ParMan looks for inconsistencies among the constraints and indicates them by displaying the associated cells in red.

Constraints entered by the user are reflected in the center column, which represents the user's local constraints. There are also right and left columns (the left constraint column is collapsed in the figure), which represent constraints shared with other agents (those "to the right" and "to the left," as described below). The arrow columns between constraint columns control when and how constraints are *propagated* between columns. The arrow columns on the far right and left similarly control when and how constraints are *advertised*, *subscribed*, *asked*, and *told* to other agents.

At any point, the user may choose to share his constraints with other agents. He can also ask to be kept informed about any constraints that other agents place on selected parameters. This is a two-step process: first the user turns on advertisement and subscription via the arrows in the right-hand column, which expresses ParMan's commitment to answer questions and receive updates about parameter constraints. Next, the user controls which constraints are actually shared by *propagating* (via an interior column of arrows) the constraints to the right-hand column. If advertisement is turned on, those constraints present in the right-hand value column are sent to the network of agents.

If subscription is turned on, constraints arrive from other agents and are added to the set of constraints in the right-hand column, and the consistency of the expanded set of constraints is tested. If there is an inconsistency among this distributed set of constraints, ParMan displays the cell for each parameter involved in the inconsistency in yellow (recall that if a local conflict

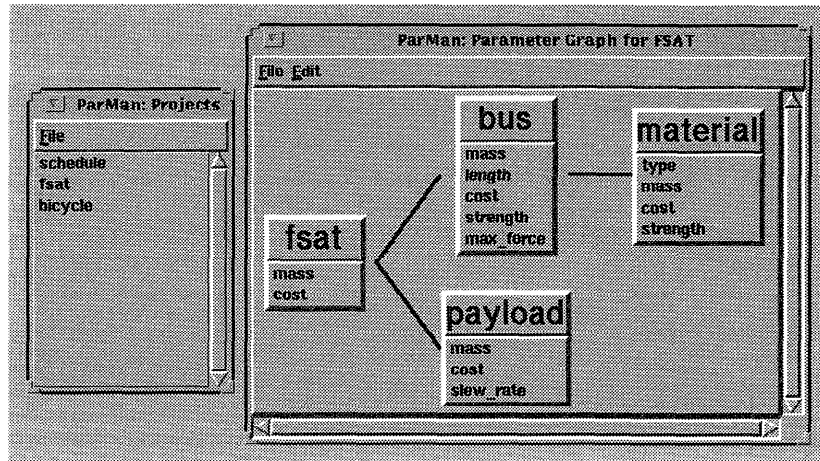


Figure 2: The Project Selector and Parameter Graph Editor permit users to collaboratively define projects and parameter spaces.

is detected, the cells turn red, since local conflicts are considered more serious). The entire set of constraints, those defined locally and those defined by other agents, can be observed by expanding the “right constraints” region in the Constraint Editor (see Figure 3).

ParMan provides several tools to aid conflict resolution. The Constraint Grapher, shown in Figure 4, presents a graphical display of the parameters and constraints, with links connecting related parameters and constraints. Constraint nodes are presented in green if the constraint is satisfied. Red is used to indicate constraints that are in direct conflict. Yellow is used to indicate constraints that participate in a conflict, but cannot be directly implicated. In Figure 4, the shaded nodes would appear in red.

The Constraint Grapher is valuable in isolating problems and finding solutions, even across multiple users. In our example, the bus designer notices that there is a problem centered around the strength of the bus (the red shading draws his attention to those constraints). He might be tempted to try different materials to overcome the strength limitation, but the *bus.material.type* node is not highlighted, indicating that choice of material has no bearing on the problem at hand. (The constraint checker determined that all other materials have problems as well: composite construction is too expensive and alloy construction is too heavy.)

Noticing that a constraint on *bus.length*, supplied by the payload designer, does participate in the problem, the bus designer calls the payload designer to see if the constraint can be loosened. This is acceptable to the payload designer, so she loosens the constraint on the bus length, which is automatically communicated between the ParMan agents. The bus designer’s ParMan checks the new constraint set, determines that there is

no longer a conflict, and removes all the highlighting in the Parameter Table. In this manner, ParMan not only provides automated constraint analysis, it also serves as an excellent visualization and collaboration tool.

Thus far in our example, parameters have only been propagated to and from the “right.” ParMan also includes a symmetrical “left” side, which is typically used for communication with a CAD tool. In our example, the user could have a rigid-body dynamics tool that, given the payload mass and slew rate, computes the force exerted on the mounting. In this case, the user would advertise mass and slew rate to the left, and subscribe to force from the left. (Since our example does not include such a dynamics tool, we approximate force via the simple formula shown).

ParMan includes several other facilities not illustrated in this example: a constraint solver, which attempts to find a closed form solution to the set of constraints; a Constraint Tester, which allows the user to drag individual constraints into a test region to isolate problems; a clique finder, which separates the constraints into independent sets; and a units converter, which is used implicitly in the example above, and can also be called explicitly.

From a more global perspective, there are several modes of use spanning two dimensions: use with or without a tool, and with or without an external agent network. As illustrated above, ParMan can be used without a tool if the user has constraints for, or manually computes all parameters. The addition of a tool simply automates aspects of this process. ParMan is useful in the absence of other agents if the user simply wants to trade-off local parameters under local constraints. The addition of other agents makes the parametric design problem distributed and collaborative.

ParMan may, in fact, be used in several modes

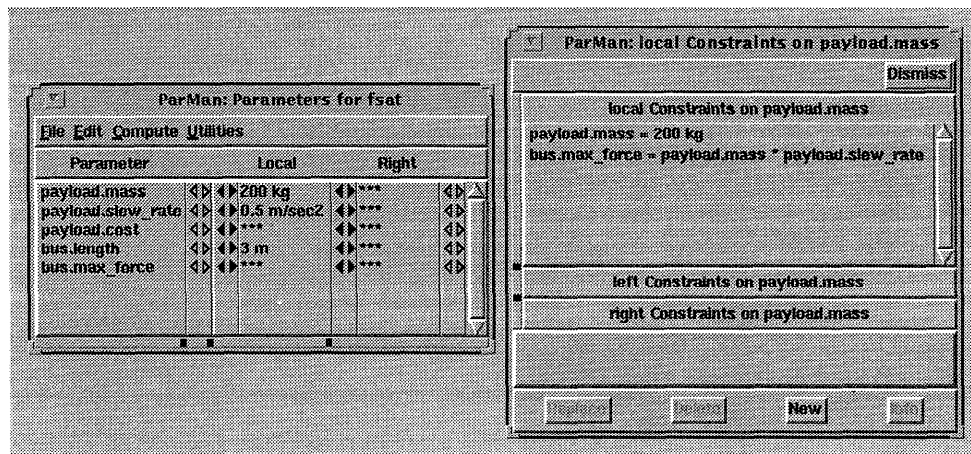


Figure 3: Parameters and simple constraints are displayed on the Parameter Table, more complicated constraints for each parameter are entered and viewed via the Constraint Editor

throughout the life of a single project. In the early stages of a project, a single user might experiment locally with various parameter settings. As he becomes confident about some of his choices, he may advertise his parameters and begin to collaborate and negotiate with other designers with ParMan's support. As the design evolves, initial rough estimates may be refined by incorporating design tools into the process.

System View

ParMan is implemented in Tcl/Tk, utilizing CLP(R) for constraint computation, and the SHADE agent infrastructure for communications. Each of these is described below.

Agent Interface

The collaborative aspect of ParMan is built on research underway in the area of agent-based, knowledge-level engineering communications (Cutkosky *et al.* 1993; McGuire *et al.* 1993). Much of this work has focused on four basic problems: developing an adequate knowledge representation to serve as an interlingua; developing a vocabulary, or ontology, that defines the terms used in communication; developing an agent speech act protocol; and developing a set of facilitation agents that improve communication among end-user agents.

ParMan is designed to work with one set of solutions to these problems, corresponding to the SHADE architecture (Kuokka *et al.* 1993). Specifically, ParMan uses KIF (Knowledge Interchange Format (Genesereth & Fikes 1992)) as its interlingua, although other representations such as Step/Express, ISO 10303, are being explored. KQML (Knowledge Query and Manipulation Language (Finin *et al.* 1992)) is used as the speech act language, which carries the embedded KIF sentence. Furthermore, ParMan assumes the existence

of a Matchmaker (Kuokka *et al.* 1993), a facilitator that matches and routes advertisements and subscriptions among the set of cooperating agents.

Even though ParMan assumes a very basic ontology for communicating parameter existence and constraints, it does not depend on the existence of an ontology for the engineering domain. Instead, the Project Selector and Parameter Graph Editor allow a distributed set of users to define a vocabulary of projects, components, and parameters interactively. The Constraint Language allows the users to define relationships among the components and parameters. This capability illustrates an incomplete but practical solution to the semantic unification problem (Petrie 1992), and is critical to successful knowledge-level communication among agents. Previous solutions have been centered around the definition of common data models (e.g., STEP) and formal shared ontologies (Gruber 1993). Both of these approaches require a high degree of a priori agreement among the participants, even before the specifics of the problem are known. By contrast, ParMan's Parameter Graph Editor provides a very dynamic, albeit human-aided, approach.

Looking more closely at the messages exchanged among ParMan agents in the example, when the user advertises his willingness to supply constraints on a particular parameter (e.g., *bus.mass*), the following messages are sent:

```
(advertise :content
  (stream-about :language kif :content
    (mass (bus fsat))))
```

```
(advertise :content
  (subscribe :content
    (stream-about :language kif :content
      (mass (bus fsat))))))
```

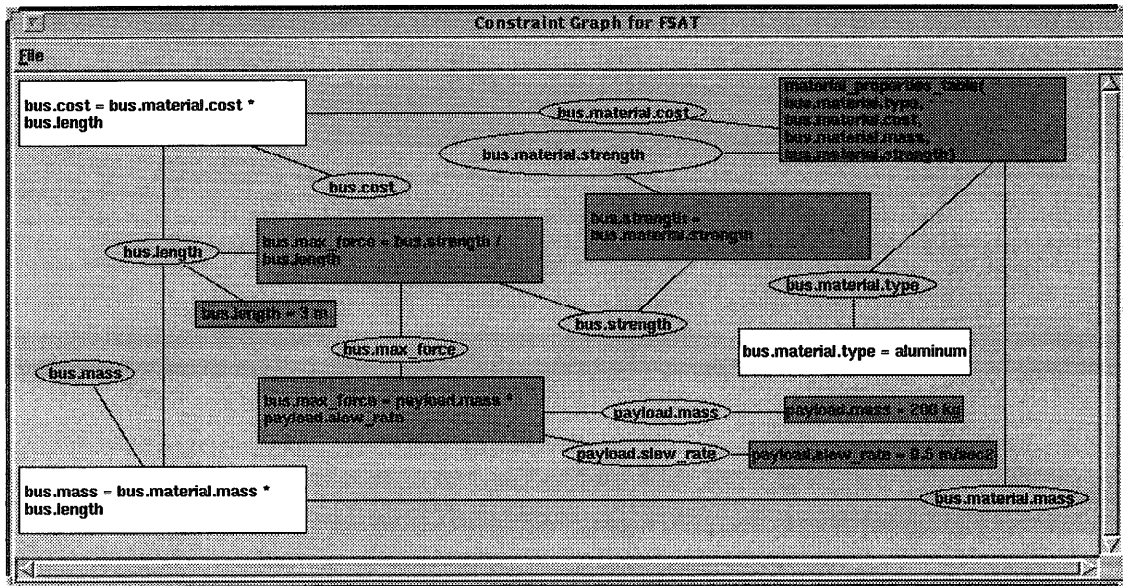


Figure 4: The Constraint Grapher aids in the visualization of relationships among parameters and constraints.

Conversely, when the user wants to hear about constraints on a particular parameter, ParMan seeks to *recruit* all agents that might assert constraints on that parameter. This is achieved via the following messages:

```
(recruit-all :content
  (stream-about :language kif :content
    (mass (bus fsat))))

(recruit-all :content
  (subscribe :content
    (stream-about :language kif :content
      (mass (bus fsat)))))
```

In each case, the first message (with top-level content *stream-about*) indicates a query/response modality. *Stream-about* is essentially a one-time query with the stipulation that replies are in the form of a stream. The second message (with top-level content *subscribe*) indicates a continuous update modality.

These messages depend on a Matchmaker, which matches subscribes to *stream-about*s and recruits to *advertises*. Upon finding a match, the Matchmaker sends the appropriate subscription or query on to the advertising agent. By depending on the Matchmaker, ParMan need not worry about sending the right message to the right place, it need only send it's capabilities, interests, and assertions to the Matchmaker, which performs the appropriate message routing.

Once a connection is made between an information producer and information supplier, specific constraints are then sent via messages of the form:

```
(tell :content
  (< (mass (bus fsat)) (* 100 kg)))
```

Constraints are withdrawn via messages of the form:

```
(untell :content
  (< (mass (bus fsat)) (* 100 kg)))
```

ParMan depends heavily upon the agent communications infrastructure being developed by projects such as SHADE. However, in return, it provides one of the first complete agent-based engineering tools to make extensive use of the infrastructure, providing valuable insights about useful protocols and necessary extensions.

Constraint Solver

The constraint checking of ParMan uses the CLP(R) language (Jaffar *et al.* 1992), which is an instance of the Constraint Logic Programming scheme defined by Jaffar and Lassez (Jaffar & Lassez 1987). ParMan applies the basic CLP(R) engine in several different ways to implement the clique finding, conflict detection, and constraint simplification functions. Each of these is described below. Since engineering domains typically intermingle a number of different unit systems, all constraints are converted to SI units before being passed to any of the constraint solving functions. This ensures that values are related accurately.

The clique finder may be used by the user to simplify the parameter space, and it is used by the conflict detection algorithm as a preprocessor. A clique is a group of all parameters, such that each parameter is linked to each other parameter in the group, directly or transitively, by one or more constraints. Cliques are detected

by simply traversing the graph formed by taking the parameters as nodes and the constraints as links.

Conflict detection is invoked whenever a constraint is added, deleted, or modified. ParMan first forms two sets of *potentially affected* constraints. A constraint is said to be potentially affected by a modified constraint if it contains parameters that are in the same clique as one or more of the parameters that occur in the modified constraint. The first set of potentially affected constraints consists solely of local constraints. The second set is the union of the first set and the potentially affected right and left constraints.

Once the two constraint sets have been formed, each is passed separately to the constraint solver. Given a constraint set, CLP(R) successively adds the constraints to a set of collected constraints, at each step determining whether the set of constraints has a solution. If there is no solution, the system backtracks, trying alternative solutions to some of the previous constraints. If no solution can be found in the first set of constraints, the corresponding parameters are marked as locally inconsistent (and appear in red in the Parameter Table). If the first set is found to be consistent, the second set is tested by an identical method. If no solution can be found, the corresponding parameters are marked as globally inconsistent (and appear in yellow in the Parameter Table).

When an inconsistent set of constraints is displayed in the Constraint Grapher or in the Constraint Tester, a slight variant of the above procedure is used to determine the status of each constraint. A single set of constraints is formed. Initially, all constraints are labeled as *exonerated*. An exoneration of a constraint is lifted if removal of that constraint from any inconsistent subset leaves the subset consistent. Next, all constraints whose exoneration has been lifted are labeled as *indicted*. An indictment is lifted if removal of that constraint from one of the inconsistent subsets leaves that subset inconsistent. All constraints that remain unlabeled are now labeled as *accused*. Exonerated constraints are displayed in green, accused constraints in yellow, and indicted constraints in red.

Finally, constraint simplification proceeds in a manner similar to conflict detection. However, if the constraint set is found to be consistent, the CLP(R) **dump** predicate is used to produce the simplified set of constraints.

Evaluation and Conclusions

ParMan has been used on example problems in several domains: the FSAT domain, a bicycle design domain, and a meeting scheduling domain. In all cases, the users were geographically distributed (in separate offices).

In each test, it is striking how the apparent complexity of the global problem was significantly reduced when distributed among multiple people. Distribution allows each user to focus on and ensure satisfaction of

those constraints representing a single perspective.

Once each user assumed his role, ParMan proved to be a very natural system for encoding and solving distributed constraint satisfaction problems. Each user's constraints were easily entered, and their propagation to other users as external constraints was simple and understandable. Flagging of conflicts by color coding also proved to convey the essential information without complexity. Finally, the facility to find a solution proved quite beneficial. In fact, in early tests without this facility, the users expressed concern that, even though the system indicated that there were no conflicts, the solution was not readily apparent.

Another problem was encountered with an earlier prototype which allowed only numerical parameters and algebraic constraints. This proved to be extremely limiting, as every domain had significant symbolic parameters (e.g., material type) and extrinsic constraints (e.g., tables of material cost). As a result, ParMan's constraint language was extended to permit symbolic constants and user-defined predicates.

In many domains, ParMan's clique finder divides constraints into subsets of manageable computational complexity. However, in domains where these subsets become large, performance of constraint testing and satisfaction may become a problem. The practical limitations of ParMan, and techniques to improve constraint satisfaction efficiency are under investigation.

The underlying agent communication infrastructure used by ParMan has been invaluable to the overall system. KQML provides the language by which agents coordinate the exchange of knowledge, and the Matchmaker allows ParMan to route messages by content rather than by name of a responsible agent.

Even though ParMan, in its current form, is quite promising, there are still several areas for improvement. The user interface does not yet allow direct editing of the constraint graph, and the interface to extrinsic constraints is still under development. Finally, significant user testing by engineers is needed to further gauge the applicability of the basic parametric design paradigm.

ParMan represents an integrated, cross-disciplinary contribution to three fields: artificial intelligence, engineering design, and human-computer interaction. From an AI perspective, it demonstrates and extends the use of CLP(R). Also, it represents one of the most significant applications of emerging agent communication techniques. In fact, the development of ParMan has resulted in several extensions to and clarifications of KQML and matchmaking. From an engineering perspective, it represents a highly interactive, intelligent concurrent engineering paradigm, with an emphasis on practicality as a key requirement. And from a human computer interaction perspective, ParMan represents a general tool for knowledge-level computer supported collaboration, with emphasis on an intuitive interface to a complex task.

Acknowledgments

The authors would like to acknowledge the contributions of Larry Harada for his implementation of the Matchmaker and feedback on ParMan interface, and Bill Mark for his insights and support of this work.

References

- Birmingham, W.; Darr, T.; Durfee, E.; Ward, A.; and Wellman, M. 1993. Supporting mechatronic design via a distributed network of intelligent agents. In *Proceedings of the AAAI Workshop on AI in Collaborative Design*.
- Bouchard, E. 1992. Concepts for a future aircraft design environment. In *Proceedings of the Aerospace Design Conference*.
- Cutkosky, M.; Englemore, R.; Fikes, R.; Gruber, T.; Genesereth, M.; Mark, W.; Tenenbaum, J.; and Weber, J. 1993. PACT: An experiment in integrating concurrent engineering systems. *IEEE Computer* 26(1).
- Dent, L.; Boticario, J.; McDermott, J.; Mitchell, T.; and Zabowski, D. 1992. A personal learning apprentice. In *Proceedings of the National Conference on Artificial Intelligence*. AAAI Press.
- Finin, T.; Weber, J.; Wiederhold, G.; Genesereth, M.; Fritzon, R.; McGuire, J.; McKay, D.; Shapiro, S.; Pelavin, R.; and Beck, C. 1992. Specification of the KQML agent communication language. Official document of the DARPA Knowledge Sharing Initiative's External Interfaces Working Group. Technical Report 92-04, Enterprise Integration Technologies, Inc.
- Frayman, F., and Mittal, S. 1987. Cossack: A constraints-based expert system for configuration tasks. In *Knowledge Based Expert Systems in Engineering: Planning and Design*. D. Sriram and R. Adey (eds.).
- Genesereth, M., and Fikes, R. 1992. Knowledge interchange format, version 3.0 reference manual. Technical Report Logic-92-1, Computer Science Department, Stanford University.
- Gruber, T. 1993. A translation approach to portable ontology specifications. *Knowledge Acquisition* 5(2).
- Jaffar, J., and Lassez, J. 1987. Constraint logic programming. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages*. Association for Computing Machinery.
- Jaffar, J.; Michaylov, S.; Stuckey, P.; and Yap, R. 1992. The CLP(R) language and system. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 14(3).
- Kolb, M. 1989. Investigation of constraint-based component modeling for knowledge representation in computer aided conceptual design. Technical report, Massachusetts Institute of Technology, Dept of Aeronautics and Astronautics.
- Kuokka, D.; McGuire, J.; Weber, J.; Tenenbaum, J.; Gruber, T.; and Olsen, G. 1993. SHADE: Knowledge-based technology for the re-engineering problem; annual report. Technical report, Lockheed Artificial Intelligence Center.
- Maes, P., and Kozierok, R. 1993. Learning interface agents. In *Proceedings of the National Conference on Artificial Intelligence*. AAAI Press.
- McGuire, J.; Kuokka, D.; Weber, J.; Tenenbaum, J.; Gruber, T.; and Olsen, G. 1993. SHADE: Technology for knowledge-based collaborative engineering. *Concurrent Engineering: Research and Applications* (1).
- Pan, J., and Tenenbaum, J. 1991. Toward an intelligent agent framework for enterprise integration. In *Proceedings of the National Conference on Artificial Intelligence*. AAAI Press.
- Petrie, C. 1992. Introduction. In *Enterprise Integration Modeling*. MIT Press.
- Petrie, C. 1993. The Redux' server. In *Proceedings of the Intl. Conf. on Intelligent and Cooperative Information Systems*.
- Reeves, B., and Shipman, F. 1992. Supporting communication between designers with artifact-centered evolving information spaces. In *CSCW*.
- Saad, M., and Maher, M. 1993. A computational model for synchronous collaborative design. In *Proceedings of the AAAI Workshop on AI in Collaborative Design*.
- Sriram, D. 1993. Computer supported collaborative engineering. In *Proceedings of the AAAI Workshop on AI in Collaborative Design*.
- Stefik, M.; Foster, G.; Bobrow, D.; Kahn, K.; Lanning, S.; and Suchman, L. 1987. Beyond the chalk board: Computer support for collaboration and problem solving in meetings. *Communications of the ACM* 30(1).
- Weber, J.; Livezey, B.; McGuire, J.; and Pelavin, R. 1992. Spreadsheet-like design through knowledge-based tool integration. *International Journal of Expert Systems: Research and Applications* 5(1).
- Werkman, K. 1992. Multiple agent cooperative design evaluation using negotiation. In *Artificial Intelligence in Design*.