# Discovering Methodologies for Integrated Product Design

Cirrus Shakeri, David C. Brown, and Mohammad N. Noori

Mechanical Engineering and Computer Science Departments
Worcester Polytechnic Institute, Worcester, MA 01609, USA
E-mails: cirrus@wpi.edu, dcb@wpi.edu, mnnoori@wpi.edu
URL:http://www.wpi.edu/~cirrus, http://www.wpi.edu/~dcb, http://me.wpi.edu/Documents/People/Faculty/mnnoori.html

**Abstract**

The current methodologies for multi-disciplinary product design are based on compromising between different disciplines rather than integrating them. These methodologies do not use a systematic and holistic approach to the problem of multi-disciplinary design and thus are piecemeal rather than comprehensive. This paper presents a new approach to producing design methodologies for integration of the different disciplines in the design process. A multi-agent system has been developed that designs a 2-DOF robot arm by incorporating five proposed strategies for integration between disciplines. Design methodologies are extracted by tracking the system and generalizing the traces that are produced. The results show that the trace of the system provides invaluable information on how to improve the design process.

## Introduction

To be able to compete in today's global market, companies need continuous improvements in the quality of their products. At the same time they need to improve the performance of their design and manufacturing processes in order to reduce the cost and the time-to-market. While there are many techniques and tools for synthesizing, analyzing, simulating, and evaluating design products, there are few similar techniques and tools for design processes. This paper is primarily about the development of techniques and tools for improving the *process* of product design as opposed to optimization of the product.

One means to improve the performance of design processes is to integrate multiple disciplines that are involved in the process. To reduce the cost and the time-to-market of products, system-oriented, holistic, and integrated approaches to multi-disciplinary design are needed [NSF 96]. Integration reduces the number of failures and backtracking by facilitating information sharing and thus saving resources. Besides, integration provides collaboration between different participants that, as a result, enhances the quality of the design.

It is becoming harder, however, to develop methodologies for integration of multiple discipline in design. This is because the number of specialists is increasing, while the number of generalists, capable of doing system integration, is decreasing. At the same time, the knowledge burden on the designer keeps increasing as more materials and more options become available [NSF 96].

Recent advances in the areas of artificial intelligence, multi-agent systems, and machine learning provide theories and techniques for developing methodologies and tools for integration in multi-disciplinary design. These theories and techniques enable engineering design researchers to take advantage of the power of computers to analyze and thus improve design processes.

In this paper we are presenting an approach for synthesizing methodologies for integration of multiple disciplines based on a knowledge-based design paradigm. We have implemented the proposed approach in a multi-agent design system that simulates the design processes.

## Multi-disciplinary Product Design

Multi-disciplinary design entails participation of different disciplines in the design process. Examples of multi-disciplinary design are the design of aircraft, automobiles, robots, and buildings. Multi-disciplinary designs are very complex processes that consume a lot of time, money, expertise, information and other resources. Complexity is due to diversity of disciplines, where each possess a different point-of-view regarding the design problem. As a result, different disciplines adopt different, and usually contradictory, goals and constraints while they have to share resources such as budget, time, expertise, and information.

Multi-disciplinary product design is hard to integrate because of the following factors:

- *Departmentalization.* Different disciplines conceptualize and represent their knowledge differently from the others. Boundaries are built around disciplines with special internal languages and ontologies, with no means for communicating with the outside world. As a consequence, it becomes difficult for the participating disciplines to communicate their points-of-view, let alone collaborate with each other or resolve their conflicts.
- *Built-in Goals.* Different disciplines tend to accumulate knowledge independently. As a result, they tend to have built-in goals that are often in conflict with global goals of the design. Ignoring the conflicts between

local and global goals leaves the behavior of the system to the dynamics that is determined by the structure of the system itself [Forrester 75, p. 253].

- *Disciplinary Design in Big Chunks*. Disciplinary designs are processed in large segments that make integration very difficult because valuable information (such as decisions that may lead to conflicts) is hidden from the rest of participants. Also, considering failures and the iterative nature of design, there is a large overhead in repeating large, discipline-based segments.
- *Counter-Intuitive Behavior.* "It has become clear that complex systems are counter-intuitive, that is they give indications that suggest corrective action which will often be ineffective or even adverse in its results" [Forrester 69, p. 1]. Multi-disciplinary designs are an example of complex systems with counter-intuitive behavior. "Intuition fails to hold true when the constraints become active; it is then that the real interaction among design groups occurs" [Wujek 96, p. 370].

## Integration in Product Design

Based on the above factors that make integration of multiple disciplines in product design a difficult problem, we propose a solution based on the following strategies:

1. *Small Design Methods.* In order to integrate different disciplines, the big chunks of design knowledge accumulated in different disciplines should be broken into pieces. In this work the design knowledge is represented in the form of design methods. A design method is a body of orderly procedures for accomplishing various design tasks (e.g., design synthesis, design selection, and design evaluation). Therefore breaking up the design knowledge into pieces, in our approach, corresponds to breaking design methods into smaller methods. Smaller design methods means that fewer decisions are made in each method, shorter time is spent in a method, and less information is produced as a result of executing that method. Smaller design methods are simpler and consume less resources.
2. *Opportunistic Contribution.* An opportunistic problem solving strategy facilitates integration of the contributions of different disciplines in the design process. The opportunistic strategy for letting different disciplines contribute to the design is in contrast to a predetermined order of contribution that each method has to make [Kroo 90]. An opportunistic approach allows us to take advantage of the diversity of different disciplines. Every participant should get a fair chance to contribute to the goals of the design process so that all points-of-view are explored.
3. *Cooperation.* A cooperative strategy provides mechanisms by which different participants adopt the same goals. Implementation of the cooperative strategy in a multi-disciplinary design process results in favoring the common goals of the design over local goals. As a result of such strategy different disciplines spend their diverse resources in the same direction. The cooperative strategy can be extended further such that different disciplines become considerate of the other disciplines' constraints when they propose their solutions.
4. *Least Commitment*. Least commitment means deferring the decisions that constrain future choices for as long as possible [Jackson 90, p. 252]. A least commitment strategy reduces the number of conflicts, because it avoids committing to decisions that are made based on incomplete information. In the absence of a least commitment strategy, decisions may be made as soon as they can be, even if incomplete, arbitrary, or less trusted information is used. As a consequence, there is more chance for conflicts to occur in the future, because such information may turn out to be invalid.
5. *Concurrency.* "It is well known that concurrent decision making is an important and very desirable component of modern design methodology" [Badhrinath 96]. A concurrent strategy, in contrast to a sequential strategy, carries out some of the problem-solving activities in parallel to each other. Concurrent design is the main theme of the well-established Concurrent Engineering field. Concurrency in design gives freedom to all participants to contribute to the current state of the design in parallel. In a concurrent design process, design knowledge is accumulated from all design participants during the design process [Brown 93]. As a result, the design process speeds up, because the participants in the design do not have to wait in a line if they can make a contribution.

Whatever the solution to the integration problem, it needs to be represented in the form of a set of design methodologies. A *design methodology* is a scheme for organizing reasoning steps and domain knowledge to construct a solution. It provides both a conceptual framework for organizing design knowledge and a strategy for applying that knowledge [Sobolewski 96].

## Knowledge Based Design Approach

A knowledge-based model of design is adopted in order to implement the proposed strategies for integration, into the design process. A knowledge-based design paradigm applies highly specialized knowledge from expert sources to the synthesis or refinement of a design or a design process [Lander 97].

The idea is to simulate the design process by building a knowledge-based design system. The system activates design methods when they become applicable, uses small design methods, facilitates information sharing, implements control techniques for promoting collaboration, and gives more priority to design tasks that lead to fewer possible conflicts.

The system conducts the design process autonomously. By recording the steps that the system has taken during the design process, some partial methodologies are constructed using an inductive learning technique. These partially developed methodologies are then reinforced by solving more design problems. Later these methodologies will be categorized based on different sets of design requirements.

Figure 1 shows how the proposed approach works for our test domain, the design of a robot arm. There are three different disciplines (i.e., kinematics, structure, and controls) involved in the design process. Design methods in each discipline are broken up into small methods such that each one of them has its own inputs, outputs, and constraints.

A *design project* in Figure 1 is a design problem that differs from other problems in its requirements and constraints. As a result, design methods that become applicable during the design process might be different for different projects. However, there will be some similar patterns in activating design methods in different projects.

The similar patterns in activating design methods are extracted and related to the group of projects that later on will be categorized under a certain type of design projects. To reinforce and further develop the similar patterns into general methodologies, the system solves more examples by perturbing the design requirements within the given range. During this process some of the partially developed methodologies will be strengthened and converge together while some will be weakened and dropped from further development agenda. At the end there will be a finite number of design methodologies for different types of design problems.

To implement the proposed approach a knowledge-based design tool based on a multi-agent architecture is developed that simulates the design process. "Design can be modeled as a cooperative multi-agent problem solving task where different agents possess different knowledge and evaluation criteria" [Sycara 90]. The multi-agent paradigm intuitively captures the concept of deep, modular expertise that is at the heart of knowledge-based design [Lander 97].

By implementing the opportunistic strategy in the multi-agent design system, methods are dynamically selected based on the individual agents' view of the problem-solving situation and on shared information about the capabilities of agents in the system [Lander 92]. Therefore, the design methodology emerges at run time.

## Multi-agent Design System

An agent as a self-contained problem solving system capable of autonomous, reactive, pro-active, social behavior is a powerful abstraction tool for managing the complexity of software systems [Wooldridge and Jennings 95] [Wooldridge 97]. A multi-agent system is "a system composed of multiple interacting agents, where each agent is a coarse-grained computational system in its own right" [Wooldridge and Jennings 98].

In this work we have used the notion of an agent as an abstraction tool for conceptualizing, designing, and implementing the knowledge-based design approach that was proposed in the previous section.

### System Architecture

The overall architecture of the developed multi-agent design system is shown in Figure 2. There are three different layers in the system: *Data*, *Control*, and *Flow.*

The data layer contains the design requirements and design constraints defined by the user at the beginning of

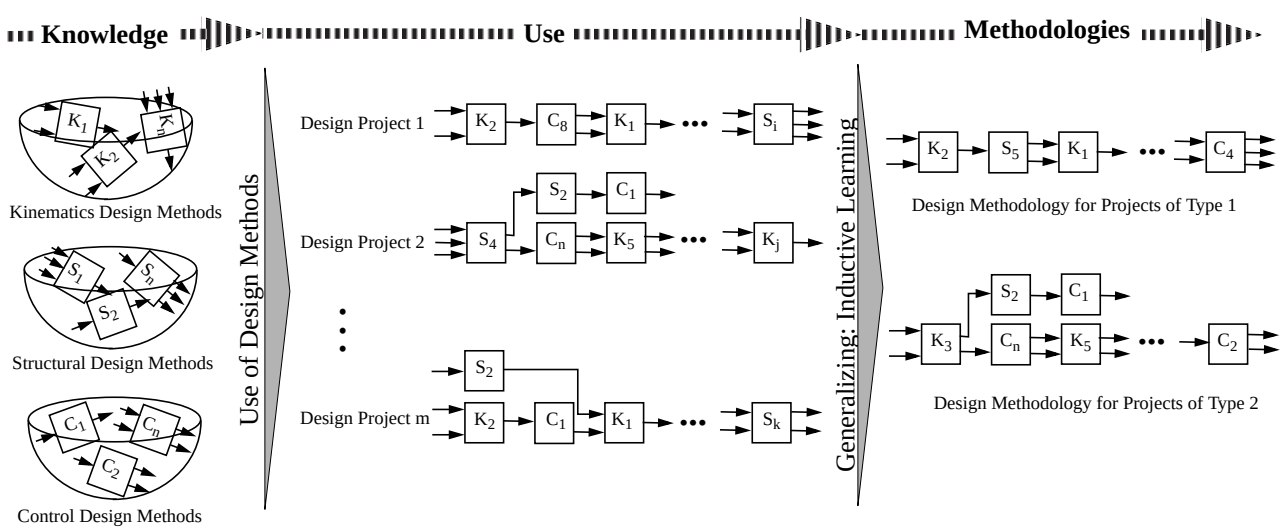


**FIGURE 1. Methodology Extraction from Traces**

each design project. The data layer also contains the state of the design process at any moment and the description of the product as it evolves during the process. Database agents update data and answer the queries of the other agents. A coordinator agent manages the consistency of the data between different database agents and synchronizes the updates and queries.

The control layer contains the design knowledge as well as the knowledge for how to use the design knowledge. In Figure 2 each Designer_m_n agent is responsible for carrying a specific design method in discipline *m* (*k* for kinematics, *s* for structural, and *c* for control design of a robot arm).

The rest of the agents in the control layer are responsible for coordination and carry out generic design tasks such as evaluation of the partial designs. They discover and provide the dependency between designers, and provide an agenda for various design tasks such as backtracking.

The flow layer of the system contains a mechanism for communication among agents based on sending and receiving messages. This mechanism consists of a registry and a message passing protocol. Each message has its own thread for processing that not only provides concurrency between agents but also it allows each agent to handle multiple messages simultaneously.

## Structure of an Agent

An agent is composed of some generic components for accomplishing common tasks (e.g., communication) and some specialized components for achieving its specific goals. The following are generic components of each agent:

1. *Message composer*: composes a message that may be sent to one or more agents. Message composer receives the name of the receiver agent(s), a performative, and the message content.
2. *Message sender*: sends the composed messages to other agents,
3. *Message receiver*: receives the messages from other agents,
4. *Message processor*: processes received messages,
5. *Observable*: sends notifications about internal events to other interested components of the same agent,
6. *Logger*: records various internal events of an agent in different log files. The logger is also responsible for cleaning up when the agent is no longer needed.

## Posing Design Goals

Coordinator agents are responsible for posing abstract goals, decomposing them into sub-goals, and following up the other agents to achieve those sub-goals. Coordinator agents decide what the other agents should accomplish in order to eliminate any need for negotiation between different agents in the system. There are three coordinator agents in the system shown in Figure 2: Coordinator, DesignersCoordinator, and DatabaseCoordinator.

The Coordinator agent has the most abstract goal in the design process, that is to achieve a design that satisfies the design requirements and constraints. Figure 3 shows how Coordinator conducts the design process in a loop until it finds either a satisfactory design or it fails to find a design that satisfies the requirements and constraints.

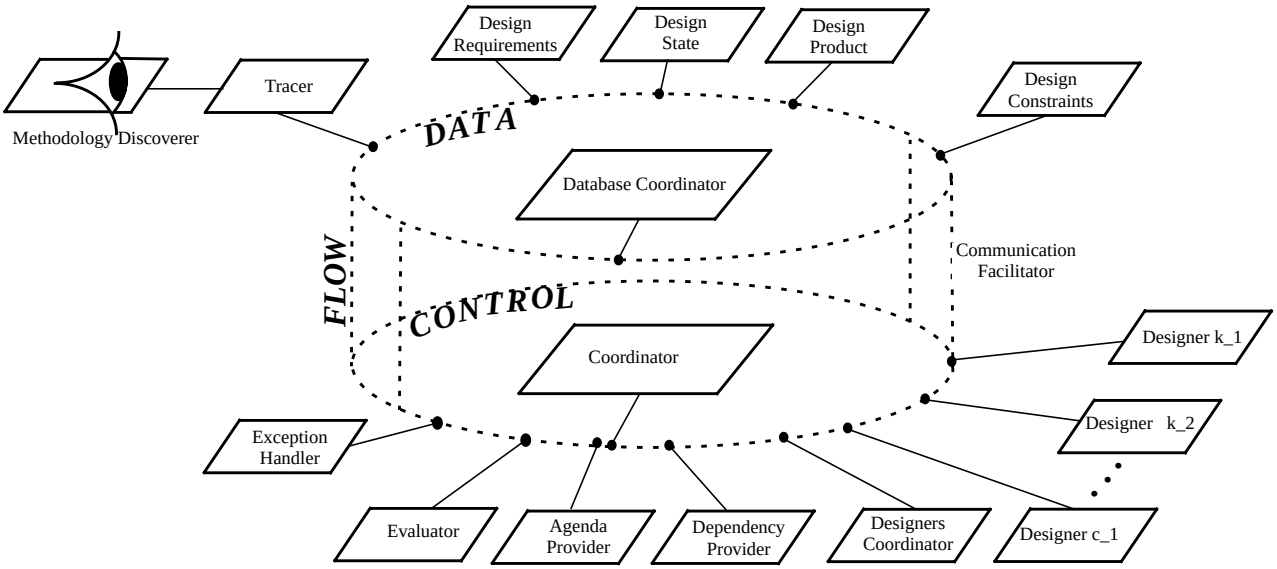


**FIGURE 2. The Architecture of the Multi-agent Design System**

## Designer Agents

Each designer agent may have multiple approaches for carrying out its design method. The available space does not allow us to describe each design approach in detail. As an example we briefly describe the design approaches of Designer_k_1, the designer that decides about the location of the base of the robot.

Designer_k_1 has four different approaches. The first approach sets the base of the robot halfway outside of the longer side of the rectangle that circumscribes the workspace. This approach generates wide sweep angles for the robot in order to cover the workspace. The second approach is similar to the first approach except that it sets the base halfway outside of the shorter side of the workspace. This approach produces longer link lengths but smaller sweep angles for covering the workspace. The third approach finds the location of the base of the robot so that the sum of the link lengths is minimized. And finally, the fourth approach sets the base in a location that minimizes the area of the robot's accessible region.

Ten designer agents participate in generating the partial designs. Designer_k_1 to Designer_k_4 are responsible for generating kinematic parameters. Designer_s_1 to Designer_s_5 generate structural specifications of the robot arm. Finally, Designer_c_1 produces the control parameters of the robot.

## Combining Design Approaches

The set of approaches in each designer agent are prioritized based on their desirability. Desirability of an approach is decided by the experts in the domain and is based on the cost of the approach in the design process as well as its effect on the cost and quality of the product. Designer agents participate in the design process as soon as all of their input parameters become available.

Inputs to a designer agent become available either by the user as design requirements or by other designers as their outputs. Designer agents use their first approach to generate a design unless there is a failure (i.e., constraint violation). When a failure occurs, designers re-design

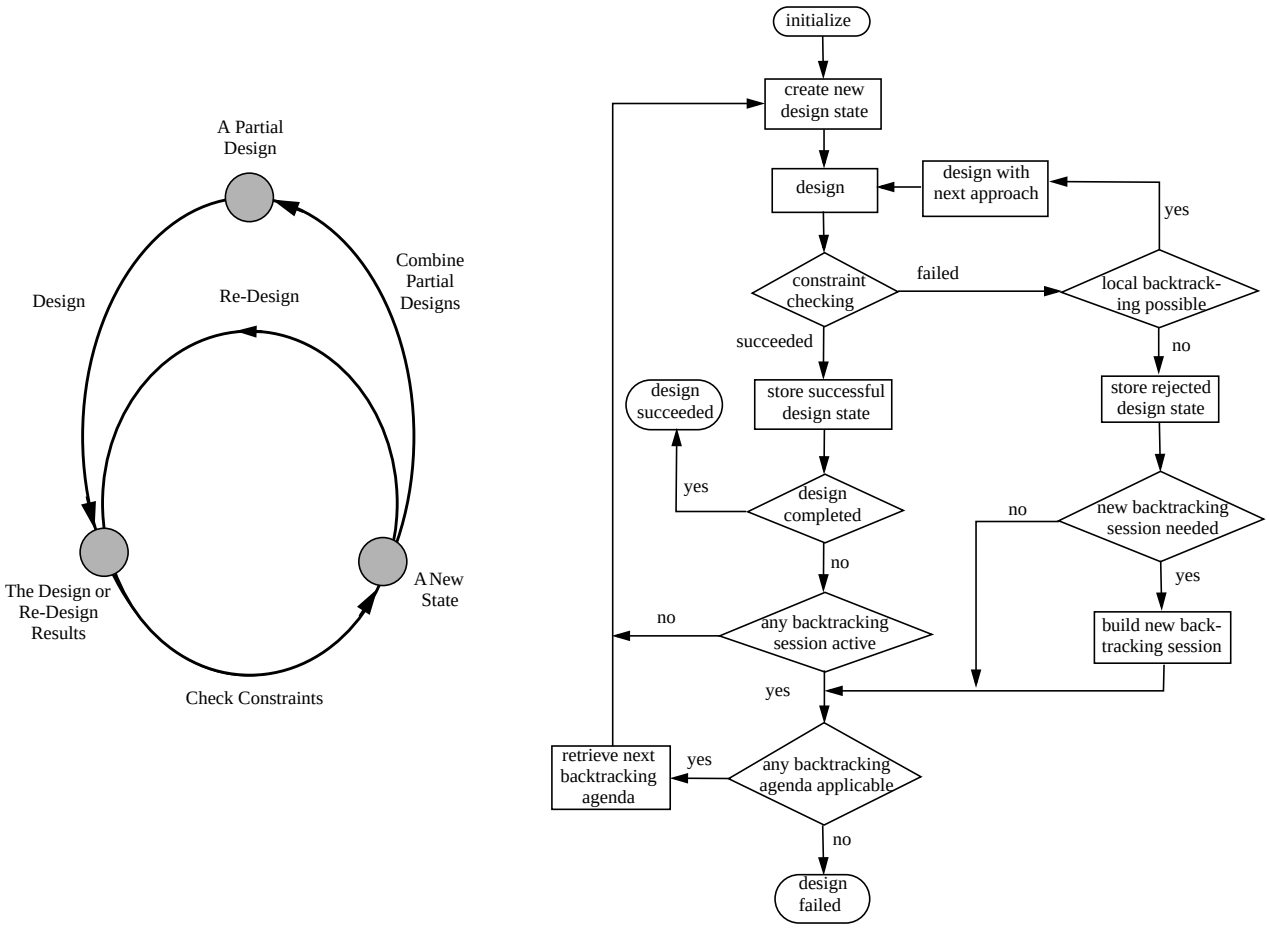


**FIGURE 3. The Flow of Design Process**

based on a backtracking agenda that is dictated by DesignersCoordinator agent. The DesignersCoordinator agent prepares and enforces the re-designing agenda so that all possible combinations of design approaches are considered. In either case (no failure or failure), design approaches are combined together in a sequence that starts a path from the designers at the root of the dependency tree to those in the leaves.

The number of possible paths is the product of the number of design approaches in all designers. Different paths are explored using a depth-first search algorithm. The system fails to produce a design if there is not any path (i.e., no combination of design approaches) that satisfies all the design constraints.

## Design Constraints

Design constraints define the criteria for acceptance or rejection of the partial designs that are generated by designer agents. Different types of constraints that are applicable to numeric or symbolic values can be defined in the system. A constraint is violated if its parameter's value is not a member of a pre-defined set. Design constraints may have been extracted from the design domain in order to satisfy physical constraints or to impose boundaries on some features of the product (e.g., cost, weight, etc.) that control the goodness of the design product.

## Implementation

A computer program called RD (Robot Designer) based on the multi-agent paradigm has been implemented for parametric design of a two degrees of freedom (2-DOF) planar robot arm. The program has been implemented in Java and runs on an Intel-based version of the Solaris operating system. The run time depends on how deep the

```
DESIGN REQUIREMENTS
{   Parameter: operational_plane, Value: horizontal,
        Owner: Agent: DesignRequirements, ID: 2
    Parameter: workspace, Value:
        [ 1.0, 1.1, 1.0939, 1.05, 1.05, 1.0939, 1.3, 1.4,  ],
        [ 0.15, 0.15, 0.1061, 0.15, 0.3, 0.5561, 0.6, 0.6,  ],
        Owner: Agent: DesignRequirements, ID: 2
    Parameter: workload, Value: 1.0,
        Owner: Agent: DesignRequirements, ID: 2
    Parameter: settling_time, Value: 3.0,
        Owner: Agent: DesignRequirements, ID: 2
    Parameter: maximum_overshoot, Value: 50.0,
        Owner: Agent: DesignRequirements, ID: 2
}
DESIGN CONSTRAINTS
{   constraint a < link1_length < b of type numeric_7:
        0.01 < link1_length (0.10036738222533641) < 0.105
    constraint a < structural_safety_factor < b of type numeric_7:
        1.0 < structural_safety_factor (1.3) < 1.6
    constraint a < link1_cross_section_dimension < b of type numeric_7:
        0.0010 < link1_cross_section_dimension (0.0044426216545476955) < 0.1
    constraint a < link2_cross_section_dimension < b of type numeric_7:
        0.0010 < link2_cross_section_dimension (0.0038776329914356573) < 0.1
    constraint a < theta2_min < b of type numeric_7:
        -0.0010 < theta2_min (0.0) < 0.1
    constraint a < proportional_gain1 < b of type numeric_7:
        0.1 < proportional_gain1 (0.14988805368956284) < 10.0
    constraint a < derivative_gain1 < b of type numeric_7:
        0.1 < derivative_gain1 (0.18224504291419413) < 2.0
    constraint a < proportional_gain2 < b of type numeric_7:
        0.01 < proportional_gain2 (0.06628060700867518) < 10.0
    constraint a < derivative_gain2 < b of type numeric_7:
        1.0E-4 < derivative_gain2 (0.0805888913181342) < 0.0010
}
DESIGNER TRACES
{   Designer_k_1: depth: 0, # of approaches: 2,
        current approach: 1 (minimize_link_lengths_summation),
        # of design cases: 40,
        # of suppliers: 1 (DesignRequirements, ) ,
        # of consumers: 2 (Designer_k_2, Designer_k_3, )
    Designer_k_2: depth: 1, # of approaches: 3,
        current approach: 2 (link_lengths_ratio_2.0),
        # of design cases: 231,
        # of suppliers: 2 (DesignRequirements, Designer_k_1, ) ,
        # of consumers: 5 (Designer_k_3, Designer_s_1,
                        Designer_k_4, Designer_s_5, Designer_c_1, )
    Designer_k_3: depth: 2, # of approaches: 1,
        current approach: 0 (default),
        # of design cases: 673,
        # of suppliers: 3 (DesignRequirements,Designer_k_1,Designer_k_2, )
        # of consumers: 1 (Designer_k_4, )
    Designer_k_4: depth: 3, # of approaches: 1,
        current approach: 0 (default),
        # of design cases: 672,
        # of suppliers: 2 (Designer_k_2, Designer_k_3, ) ,
        # of consumers: 0
    Designer_s_1: depth: 2, # of approaches: 6,
        current approach: 5 (thickness_dimention_ratio_.2),
        # of design cases: 673,
        # of suppliers: 5 (Designer_k_2, Designer_s_4,
                        Designer_s_2, DesignRequirements, Designer_s_3, ) ,
        # of consumers: 2 (Designer_s_5, Designer_c_1, )
    Designer_s_2: depth: 0, # of approaches: 15,
        current approach: 7 (steel_stainless_AISI_302_cold_rolled),
        # of design cases: 40,
        # of suppliers: 0,
        # of consumers: 3 (Designer_s_1, Designer_s_5, Designer_c_1, )
    Designer_s_3: depth: 0, # of approaches: 5,
        current approach: 2 (safety_factor_1.3),
        # of design cases: 40, # of suppliers: 0,
        # of consumers: 1 (Designer_s_1, )
    Designer_s_4: depth: 0, # of approaches: 2,
        current approach: 0 (hollow_round),
        # of design cases: 40,
        # of suppliers: 0, # of consumers: 3 (Designer_s_1, Designer_s_5,
                        Designer_c_1, )
    Designer_s_5: depth: 3, # of approaches: 1,
        current approach: 0 (default),
        # of design cases: 672,
        # of suppliers: 5 (Designer_k_2, Designer_s_4, Designer_s_2,
                        DesignRequirements, Designer_s_1, ) ,
        # of consumers: 0
    Designer_c_1: depth: 3, # of approaches: 1,
        current approach: 0 (default),
        # of design cases: 672,
        # of suppliers: 5 (Designer_k_2, Designer_s_4, Designer_s_2,
                        DesignRequirements, Designer_s_1, ) ,
        # of consumers: 0
}
```

**FIGURE 4. Trace of the System**

solution is in the design space and varies from a few minutes to several hours.

The object classes in RD can be categorized into two types: agent and non-agent classes. Agent classes are those that inherit from a superclass, naturally called Agent, that contains the generic components of an agent. While for each specific agent class there is only one instance, the system might create as many non-agent class instances as necessary. Some of the non-agent classes are: Message, DesignParameter, BacktrackingSession, Event, Constraints, DesignCase and many more.

## Traces of the System

RD produces many output files including detailed logs of sent, received, processed and ignored messages, as well as reports about specific tasks in each agent. It also produces a trace file that for each design project contains design requirements and constraints as entered by the user followed by the status of each designer agent at the time of tracing.

Figure 4 shows a partial trace of the system for an example project. The trace shows that the last constraint (on derivative_gain2) is not yet satisfied. The system stops when it either finds a set of parameters that satisfies all the constraints or exhausts all the possible combination of approaches.

In the DESIGNER TRACES block of Figure 4 the first row shows the name, the depth of the designer in a dependency tree, and the total number of design approaches. The second row shows what design method was used by the designer at the time of recording the trace. The number of design cases that the designer agent has done so far is given in the third row. Design cases differ in the design approach that is used or in the values of the input parameters. Finally, the fourth and the fifth rows show what designers provided the inputs and what designers used the outputs of this designer.

## Discussion

The trace files that are produced by RD can be used to generate a tree that reveals the dependency between designer agents and thus between design decisions. Figure 5 shows such a dependency tree that was generated based on the trace file of Figure 4.

Distribution of disciplinary designers throughout the process is evident from Figure 5 that is the sign of integration among disciplines. The integration of multiple disciplines has been facilitated by the flow of information

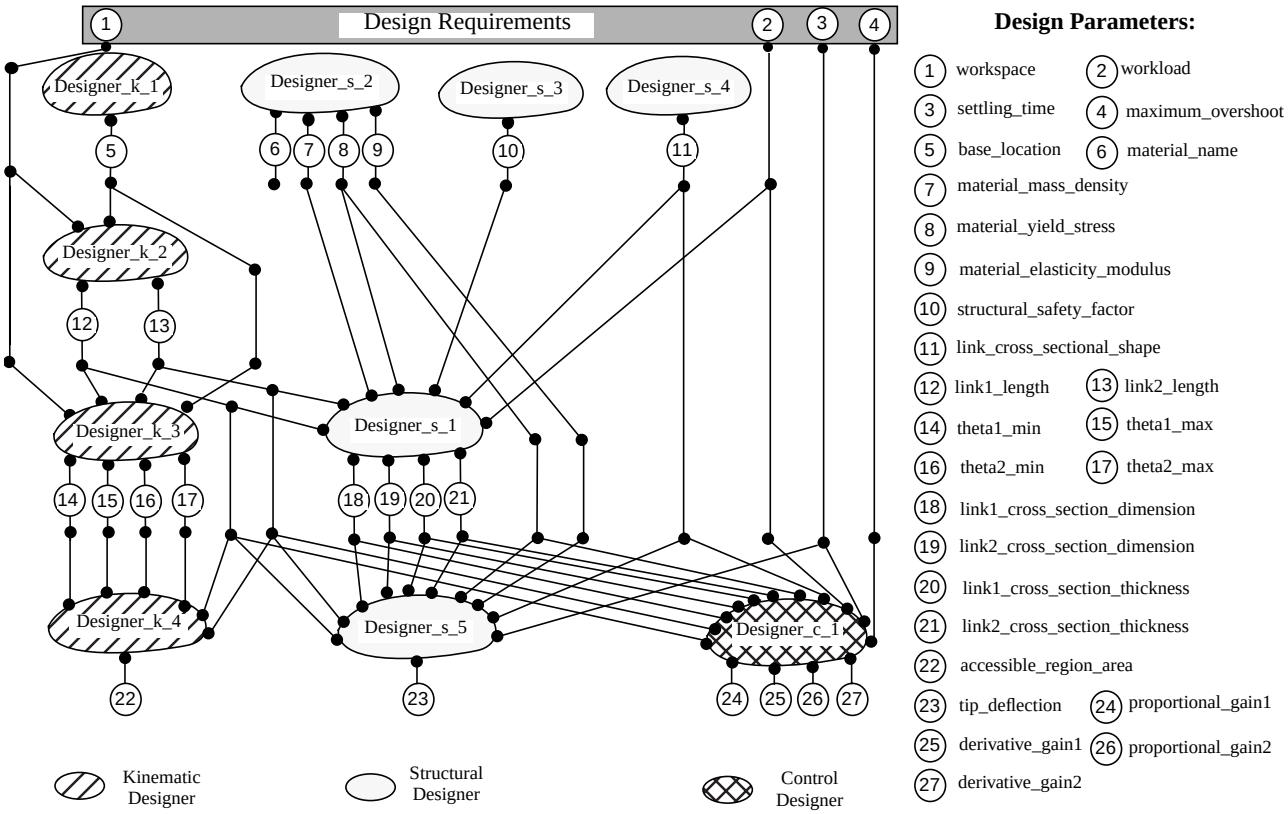


**FIGURE 5. Dependency between Designers**

between small designer agents and by letting them contribute to the design as soon as they have enough information.

At least three important characteristics of the design process can be extracted from the dependency tree of Figure 5: 1) ordering of design decisions, 2) concurrency among some of the decisions, and 3) distribution of disciplinary decisions throughout the design process.

Every trace that the system produces at the end of the design process is an instance of a design methodology. Due to the depth-first search algorithm the trace of the system implicitly contains the failures that had led to backtracking. As a result, the generalized traces have the advantage of avoiding the failures and starting from a point that has a much better chance for fast convergence.

The trace of the system reveals the most dependent and the most independent designers, indicating which decisions are more costly to change. For instance, by looking at the results in Figure 5 we realize that Designer_c_1 is the most dependent designer and thus most vulnerable to changes. Designer_s_2 to Designer_s_4, on the other hand, are the most independent designers, because in the absence of any user's requirements they use their domain knowledge to generate a design. For instance, Designer_s_2 chooses a material from the set of options that it has. It selects another material if a re-design is needed.

By studying the amount of time and the information (i.e., sent and received messages) that a designer agent needs in order to make a design decision the bottlenecks and costly tasks can be found. This information is stored in each designer agent's log files. For instance, sometimes Designer_k_1 takes considerably longer periods of time to make its design decision (i.e., the location of the base of the robot). Investigating Designer_k_1 reveals that in those cases it uses one of its iterative approaches, hence taking longer time. As another example, Figure 5 shows that Designer_k_2 (that decides about the length of the links of the robot) acts as a bottleneck, because no other designer can participate in the process concurrently.

## Conclusion

Simulation of design processes based on a multi-agent paradigm is a new area of research that has a high potential for practical as well as theoretical impact on the design of products. The results show that invaluable information about the design process can be discovered by simulating the design process. This information can be used to improve design processes as it is done by human.

Enriching the design methodologies so that they contain more knowledge on how to conduct the design process is an extension to the current work. Specializing the methodologies in other aspects such as Design For X (e.g., manufacturability, assembly, etc.) is another area to which the proposed approach can be applied.

## References


Badhrinath, K. and Jagannatha Rao, J. R. 1996. Modeling for Concurrent Design Using Game Theory Formulations. *Concurrent Engineering: Research and Applications*, 4(4):389-399.

Brown, D. C. and Douglas, R. 1993. Concurrent Accumulation of knowledge: A View of CE. *The Handbook of Concurrent Design and Manufacturing*, (Eds.) Parsaei, H. R., and Sullivan, W. G.: Chapman & Hall: 402-412.

Forrester, J. W. 1969. *Urban Dynamics*: MIT Press.

Jackson, P. 1990. *Introduction to Expert Systems*: Addison-Wesley.

Kroo, I., and Takai, M. 1990. Aircraft Design Optimization Using a Quasi-Procedural Method and Expert System. *Multidisciplinary Design and Optimization Symposium*, Nov. 1990.

Lander, S. E., and Lesser, V. R. 1992. Customizing Distributed Search Among Agents with Heterogeneous Knowledge. In *Proceedings of 5th International Symposium on AI Applications in Manufacturing and Robotics*, Cancun, Mex., Dec. 1992.

Lander, S. E. 1997. Issues in Multi-agent Design Systems. *IEEE Expert: Intelligent Systems and their Applications* 12(2): 18-26.

NSF 1996. Research Opportunities in Engineering Design. *NSF Strategic Planning Workshop Final Report*, April 1996.

Sobolewski, M. 1996. Multiagent Knowledge-Based Environment for Concurrent Engineering Applications. *Concurrent Engineering: Research and Applications* 4(1): 89-97.

Sycara, K. 1990. Cooperative Negotiation in Concurrent Engineering Design. *Cooperative Engineering Design*: Springer Verlag.

Wooldridge, M., and Jennings, N. R. 1995. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115-152.

Wooldridge, M. 1997. Agent-based software engineering. *IEEE Transactions on Software Engineering*, 144(1): 26-37.

Wooldridge, M., and Jennings, N. R. 1998. Pitfalls of Agent-Oriented Development. In *Proceedings of the Second International Conference on Autonomous Agents*, 385-390. Minneapolis/St. Paul, MN. USA, May 9-13, 1998.

Wujek, B. A., Renaud, J. E., Batill, S. M., and Brockman, J. B. 1996. Concurrent Subspace Optimization Using Design Variable Sharing in a Distributed Computing Environment. *Concurrent Engineering: Research and Applications* 4(4): 361-377.