A Modular Decision-centric Approach for Reusable Design Processes
Jitesh H. Panchal, Marco Gero Fernández, Christiaan J. J. Paredis, Janet K. Allen and Farrokh Mistree
Concurrent Engineering 2009; 17: 5
DOI: 10.1177/1063293X09102251

The online version of this article can be found at:
http://cer.sagepub.com/cgi/content/abstract/17/1/5

Published by:

SAGE
http://www.sagepublications.com

Additional services and information for Concurrent Engineering can be found at:

Email Alerts: http://cer.sagepub.com/cgi/alerts
Subscriptions: http://cer.sagepub.com/subscriptions
Reprints: http://www.sagepub.com/journalsReprints.nav
Permissions: http://www.sagepub.co.uk/journalsPermissions.nav
Citations http://cer.sagepub.com/cgi/content/refs/17/1/5
A Modular Decision-centric Approach for Reusable Design Processes

Jitesh H. Panchal, Marco Gero Fernández, Christiaan J. J. Paredis, Janet K. Allen and Farrokh Mistree*

Systems Realization Laboratory, Woodruff School of Mechanical Engineering
Georgia Institute of Technology, Atlanta, GA 31322, USA

Abstract: The reusability of design processes modeled in existing Product Lifecycle Management (PLM) and Computer Aided Engineering (CAE) frameworks has been limited to the level of flow charts or activity-based diagrams that serve as planning and organizational aids. Current simulation-based design frameworks provide limited support for reuse of design processes at a level where design processes are networks of computational operations, specifically the capabilities to reuse (a) design processes for different products, and (b) collaborative design strategies. In this article, we address these limitations by providing a modeling approach for simulation-based design processes so that they can be archived in a generic modular fashion and reused for collaborative design of different products. The proposed approach is based on four foundations: (a) modeling design processes as hierarchical systems, (b) separation of declarative and procedural information, (c) modeling design processes as decision-centric activities, and (d) modeling interactions between decision makers using game theoretic protocols. These four fundamentals of the approach are instantiated in the form of generic computational templates for products, processes, decisions, and pertinent interfaces. The approach is illustrated using a proof of concept implementation in ModelCenter. The implementation is validated by showing the reusability of design processes for two different products, a spring and a pressure vessel, in individual and collaborative design scenarios. The approach has potential for supporting reusability of broader PLM processes.

Key Words: design processes, templates, reusability, decision-centric design, modularity, collaboration.

1. Frame of Reference – Reusability of Design Processes

As engineering enterprises become increasingly concerned with meeting the dynamic requirements of a global marketplace, closer attention must necessarily be paid to reusing the knowledge associated with the mechanisms underlying product development. Perhaps the most crucial of these mechanisms is the design process. In terms of an engineering enterprise, this translates to the need for a systematic means of reusing design processes and the associated design knowledge. It has been argued that design processes play an important role in Product Lifecycle Management (PLM), which is defined as ‘a strategic approach to creating and managing a company’s product-related intellectual capital, from its initial conception to retirement’ [1]. Although much attention has been paid to addressing this issue by exploiting the reusability and scalability of products through product platform and product family design, not much attention has been paid to the reusability of an engineering enterprise’s primary intellectual capital – the design process [2].

*Author to whom correspondence should be addressed.
E-mail: farrokh.mistree@me.gatech.edu
Figures 1–3 and 6–8 appear in color online: http://cer.sagepub.com

The extent of reusability of design processes depends significantly on the level of abstraction at which it is defined. To understand the reusability of design processes, it is important to identify different levels of design processes. Design processes are categorized by Panchal and co-authors [3] into various levels (Figure 1) – business process level, inter-organizational design level, design methodology level, analysis execution level, and computing resource management level. At the highest level of abstraction (Level 5), business processes are essentially defined as information exchanges between different business units. PLM applications such as TeamCenter [4], SMARTEAM [5], and Windchill [6] capture design processes at this level as exchanges of files between different entities. At the lowest level of abstraction (Level 1), processes are composed of information processing steps as required to execute any single analysis.

At the intermediate levels (Levels 2–4) design processes consist of strategies for exploring the design space in order to achieve desired product functionality. The emphasis at these levels is on utilizing simulation-based tools to analyze the product’s behavior in conjunction with design exploration tools such as design of experiments, response surface modeling, optimization, etc. Processes at these levels are captured in terms of parameter flows by applications such as FIPER [7], ModelCenter [8], and iSIGHT [9].
These applications allow for modeling design processes in terms of various simulation codes and the required parameter flows between them. These parameter flows are generally specific to the problem at hand and the product to be designed. These tools do not support modeling design processes in a generic form that can be applied to diverse design scenarios. Hence, at a design methodology and analysis execution level, the reusability of design processes modeled using currently available tools is rather limited. In this paper, we focus on reusability at the design methodology and analysis execution level, specifically the reusability of (a) computational design processes across different products and (b) collaborative multidisciplinary design strategies.

Reusability of design processes across different products: Consider a simple example involving the design of two commonly employed mechanical components, namely, a pressure vessel and a spring. Both products are different with regard to design parameters, requirements, and design constraints. Nevertheless, a general design process that is equally applicable to both these products can still be defined at a computational level. An example of such a generic process involves constraint evaluation, analysis, goal evaluation and objective function evaluation. A common driver governs the optimization process, yet the process shown in the figure is sufficiently generic to be utilized for a parametric design of both a helical spring and a pressure vessel.

Such a design process can currently be modeled in applications such as FIPER, iSIGHT, and ModelCenter predominantly as a progression of parameter flows between successive tasks, as exemplified by the ModelCenter implementation of the design process for the pressure vessel in Figure 2. These parameter flows are inherently defined in terms of product-specific information such as design variables, parameters, response variables, etc. In the pressure vessel example, these variables are length, radius, thickness, etc. While changing numeric values required for parametric design of a single product is supported in current design processes, changing variables, underlying information flows, and tools, for a different product’s design (such as that of a spring) is not possible without remodeling the process as a whole. Designers are forced to model the design processes in both a product-specific and tool-specific manner and hence, these underlying processes cannot be reused directly from one problem to another. In other words modular reuse of design process components is not feasible. This is the first requirement for a design process modeling approach – the ability to reuse generic computational processes across different products.

Reusability of collaborative design strategies: Various computational design strategies have been developed in the multi-disciplinary design and optimization literature. Examples of such strategies include collaborative optimization [10], target cascading [11], Bi-Level Integrated System Synthesis (BLISS) [12], decentralized sequential iterative decision-making [13], etc. The focus in these strategies is on efficient exploration of design spaces via the decomposition of a complex design problem into sub-problems that are handled by different designers. Each of these strategies advocates a specific flow of information between the sub-problems. It is important to note that these strategies are

Figure 1. Applications addressing different levels of abstraction of design processes [3].
applicable to a general design problem and designers should be able to capture these strategies as reusable processes at a computational level. However as mentioned before, due to the rigid link between the processes and the product information, these strategies cannot be stored and reused as generic processes within currently available frameworks. This highlights the second requirement for a design process modeling approach – ability to reuse collaborative design strategies.

In this paper, we propose an approach to address these requirements by adopting a systems-based view of design processes and modeling design problems and solution procedures independently. The approach is discussed in detail in Section 2. The design problems are modeled from a decision-centric perspective wherein each design process can be considered to be a network of decisions; it is in terms of decisions that we model the problem-solving aspect of design. The required generic aspects of the solution procedure are captured by defining templates that can be executed, analyzed, stored, and reused, regardless of context, engineering domain, or scale of the product considered. Collaborative design processes are similarly modeled using decisions made by multiple interacting designers. The interactions between designers are modeled by specific interface protocols. An instantiation of the proposed approach in the form of computational templates is presented in Section 3 and a proof-of-concept implementation in the ModelCenter framework is presented in Section 4. The reusability of generic design processes is shown using simple design examples, that despite being rather simple in nature, nevertheless constitute a convenient means of illustrating the approach. Closing thoughts are presented in Section 5.

2. Systems-based Modeling of Decision-centric Design Processes

The proposed approach to support design process reuse at a computational level is based on four foundations: (a) Modeling design processes as hierarchical systems, (b) Separation of declarative from procedural information to enhance reusability of design processes, (c) Viewing design as a decision-centric activity, and (d) Modeling interactions between decision-makers using game theoretic protocols. These foundations are discussed next.

2.1 The Hierarchical Systems View of Design Processes

Our design process modeling strategy is based on the premise that processes are hierarchical systems that can be progressively decomposed into sub-processes, which in turn can be represented in terms of basic design process building blocks, i.e., information transformations. Hence, design processes are viewed as networks of said information transformations. These information transformations include decisions, abstraction, composition, decomposition, mapping, evaluation, and refinement, which are discussed in more detail in Ref. [14]. Each of these transformations is associated with an input product state, an output product state, and a design sub-process (which again is a network of information transformations) for its execution. These inputs, outputs, transformations, and design processes are related as shown in Figure 3 and represent core elements to this perspective. As shown in the figure, information related to formulating the transformations

![Figure 2. Design process for pressure vessel in model center.](http://cer.sagepub.com)
(declarative information) and solving them (procedural information) is clearly separated. The details of this separation are discussed in Section 2.2.

The design-related information is captured in the form of computational templates. Separate templates are developed for design processes, transformations (e.g., decisions), product information, and interfaces between transformations. The details of these four template types are provided in Section 3. Each of these templates is defined generically. The product-specific templates only contain information regarding product parameters, constraints, relationships between form and behavior, etc. A design process template is defined in terms of a network of information flows among tasks. The generic (product independent) process templates can be instantiated by populating product-specific information (such as product-specific parameters, constraints, analysis codes, etc) from the product template into the design process template. The design process templates can be executed only after their instantiation using the product information. Thus, a single generic process template can be instantiated for different products. It is the product-specific information that serves as the only differentiator among instantiated templates for designing different products; the underlying structure remains the same. Having outlined the modular systems-based perspective espoused in this research, we proceed to discuss the concept of separating declarative and procedural information in Section 2.2.

### 2.2 Separation of Declarative from Procedural Information

As discussed in Section 1, the primary reason for the lack of support for design process reuse in Computer Aided Engineering (CAE) and PLM frameworks is that current tools such as FIPER, ModelCenter, and iSIGHT capture process information in a manner that is tightly integrated with the information, specific to the product at hand. Hence, it is not possible to reuse different process definitions in product design. Currently available information models and design support tools force designers to think in terms of the underlying procedure for solving a particular problem rather than conceptualizing and declaring the problem itself. Hence, in developing the computational templates, we separate the representation of problem formulation related (declarative) information and process execution specific (procedural) information. The information associated with design transformations and the product states is declarative information because it refers to what is done by the designer via that transformation. Declarative information thus captures all the pieces of information/knowledge and the associated relationships among them that represent the transformation to be carried out. Hence, the templates associated with (a) design transformations, and (b) product information are referred to as declarative templates. The mechanics of how that information transformation is carried out constitutes the procedural information; transformation
The idea of separating declarative from procedural information is analogous to understanding the behavior of a system that is represented by a set of linear equations. The first step for understanding the system behavior is formulating (declaring) all the equations that correspond to the information/knowledge available to designers. In our approach, these equations are analogous to the problem formulation and are modeled using declarative templates. Once the equations have been formulated, the next step is to select a process to be used for solving those equations simultaneously. Various algorithms (that correspond to the processes for solving the equations) such as Cramer’s rule, Gaussian elimination, LU decomposition, the Jacobi method, etc are available for solving such a set of linear equations. These algorithms for solving systems of equations are analogous to design processes and are captured using procedural templates. One of the advantages inherent in separating declarative and procedural information is that this scheme forces designers to focus on design problem formulation before considering the details of the solution. This is important because without appropriately formulating the design problem, the designers are likely to incur penalties associated with inefficient iteration and costly redesign. A further advantage is that the reusability of design processes for solving different kinds of design problems is enhanced. The separation of declarative and procedural information is based on the assumption that models for design transformations are available. The design transformations can be developed from various different views of design. In this paper, we advocate adopting a decision-centric view of design as a basis to model key design transformations. The details of this view are discussed in Section 2.3.

2.3 Decision-centric View of Design Processes

The decision-centric view of design addresses the limitations of the model-centric and tool-centric views discussed in Section 1. From a decision-centric perspective, the emphasis is on modeling design processes as networks of decisions. According to many researchers such as Hazelrigg [15], Muster and Mistree [16], and Thurston [17] the fundamental premise of decision-based design is that engineering design is primarily a decision-making process. Specific advantages of adopting a decision-centric perspective include the ease with which both model-centric and tool-centric views are generated. Furthermore, domain independent representation of design processes becomes feasible. Hazelrigg describes decision-based design as omni-disciplinary, “the seed that glues together the heretofore disparate engineering disciplines as well as economics, marketing, business, operations research, probability theory, optimization and others” [15]. Herrmann and Schmidt [18] describe a complete product development organization as a network of decision-makers who use and create information to develop a product. Although principles of decision-based design have been accepted in design theory research, they have not been implemented in design frameworks. Current tools do not capture information related to designers’ decisions; the decision-related information is captured in the form of meta-data (if at all). In this article, we use decision-centric design to model the building blocks of design processes in the form of decision problems.

The underlying need for reusability of information related to design processes necessitates representing information in a domain neutral form that supports designers in providing and structuring required information content in a reusable fashion. This in turn calls for a domain independent means of capturing design information. In order to facilitate designer interactions, required for effective collaboration from a decision-based perspective, expression of information related to design decisions in a standardized format is also required. Such a standardized form for representing information is provided by the Decision Support Problem (DSP) technique proposed by Mistree and co-authors [19–22], specifically the compromise Decision Support Problem (cDSP) [23]. In the DSP technique, support for human judgment in designing is offered through the formulation and solution of DSPs, which provide a means for modeling decisions encountered in design. The cDSP formulation consists of four key steps – (a) given, (b) find, (c) satisfy, and (d) minimize. In the ‘given’ step the information available to designers for decision-making, which includes the available simulation models that generate information about the system’s behavior and a designers’ preferences, is captured. In the ‘find’ step of the cDSP, information about the design variables that designers can control in order to satisfy the design objectives is captured. The information about bounds on design variables, any problem constraints, and the design goals is captured in the ‘satisfy’ step of the cDSP. The overall objective function, framed in terms of deviation from target, to be minimized is captured in the final step.
Since the keywords and descriptors are domain independent, they represent a common structure (or a conceptual schema) for DSPs from any domain. This is one of the most important characteristics of the DSP technique that enables reuse of design information across domains. In summary, decision-based design is chosen as a framework here because of its (a) domain independence (decisions are common across different engineering domains), (b) design phase independence (the structure of decisions remain the same during any phase of the design process), and (c) ability to be used for modeling processes at other levels defined in Figure 1 in terms of design decisions. Having discussed the constructs for modeling individual design decisions, we present the game theoretic protocols for modeling interactions between decision-makers in Section 2.4.

2.4 Game Theoretic Protocols for Collaborative Design

In this article, we have chosen to embody both cooperative and noncooperative game theoretic protocols in the pursuit of conflicting objectives, subject to a common set of constraints. Game theory is defined as a theory of competition stated in terms of gains and losses among opposing players or a mathematical method of decision-making in which a competitive situation is analyzed to determine the optimal course of action for an interested party [24]. Von Neumann and Morgenstern are credited with introducing the subject of game theory in their book *The Theory of Games and Economic Behavior* [25] in which they also provide their treatment on the subject of decision theory in terms of utility.

Lewis and Mistree [26] model the strategic relationships among designers sharing a common design space using game theoretic principles and identify Pareto Cooperation, Nash Non-Cooperation, and Stackelberg Leader/Follower as the three game theoretic protocols most representative of the interactions required for decentralized design.

(a) **Pareto Cooperation** is employed to represent centralized decision-making, where all required information is available to every collaborating designer. A Pareto optimal solution is achieved when no single designer can improve his or her performance without negatively affecting that of another. In this scenario, designers have full access to the information about each other’s decision-making process including their cDSPs, and associated engineering tools. The Pareto cooperation scenario is solved by combining all designers’ cDSPs.

(b) **Nash Non-Cooperation** refers to decentralized decision processes where designers have to make decisions in isolation due to organizational barriers, time schedules, and geographical constraints. Its mathematical models are suitable for formulating decisions in collaborative design [27]. In Nash Non-Cooperative protocols, decision-makers formulate Rational Reaction Sets (RRS) or Best Reply Correspondences (BRC). A RRS is a mapping (either a mathematical or a fitted function) that relates the values of design variables under a designer's control to values of design variables controlled by other stakeholders. The Nash Noncooperative solution to the coupled, decentralized decision-making problem is the point of intersection of the RRSs pertaining to the different designers. The resulting Nash equilibrium to the design problem has the characteristic that no designer can unilaterally improve his/her objective function [28].

(c) **Stackelberg Leader/Follower** protocols are implemented to model sequential decision-making processes where the 'leader' makes his or her decision, based on the assumption that the 'follower' will behave rationally. The follower then makes his or her decision within the constraints emanating from the leader’s choice. In this scenario, the leader constructs a RRS by predicting the follower’s reactions and makes decisions by using this RRS into the leader's cDSP. This is an effective way to solve the collaboration problems in the case where there is a dominant design objective that must be satisfied.

Having discussed the game-theoretic protocols as means for modeling interactions between design decisions, we now present the templates for modeling design information.

3. Templates for Modeling Design Information

In this section, we present the templates for (a) Product information (Section 3.1), (b) Decision problem information (Section 3.2), (c) Process information (Section 3.3), and (d) Interface information (Section 3.4). The templates for products and decision problems are declarative templates, whereas the process and interface templates are procedural templates.

3.1 Templates for Product Information

The information specific to the product being designed includes aspects related to the product’s form, function, and behavior and the relationships between them. Since this information is treated in a process independent manner, it can be used for populating any design process template. The product
The template is derived from the Core Product Model (CPM) proposed by Fenves [29]. The key aspect of this information model is an entity. A product is composed of many such entities. Each entity, in turn, is composed of various sub-entities and is defined by a number of attributes. Each attribute can be either of type form or type behavior. Form attributes combine together to represent the form of an entity, whereas behavioral entities taken together represent the behavior of that entity.

Entities are abstractions of reality. These can be abstract concepts such as elements in Finite Element Models, boundary conditions, etc. Hence, the product hierarchy does not necessarily correspond to the part/subpart (assembly) hierarchy. The hierarchy represents a design perspective that the designer is interested in. The abstract nature of the entities allows us to view attributes as special types of entities. Since the product information is defined by the designers based on their respective perspectives of design, different designers may model the same product in an entirely different manner. Relations can be of two types – entity-inherent relationships and external relationships. Entity inherent relationships are the relationships between the sub-entities of the entity under consideration. External relationships are the relationships between the sub-entities of the entity under consideration. The behavioral model is a special type of entity-inherent relationship. The relationships may be given by simple mathematical equations or complex software codes. For example, the relationship between form and behavior is often captured by analysis codes.

3.2 Templates for Decision Information

The templates for design decisions are based on the cDSP construct presented in Section 2.3. The key components of decision templates include information about design variables, responses, parameters, constraints, goals, preferences, and objectives. An information model for Decision Support Problems is shown in Figure 4. The topmost entity is the Decision Problem. This decision problem contains all the declarative information related to a Decision Support Problem. The decision problem consists of four important elements – design space, response space, problem constraints, and preferences. Design space is defined by all the design variables that can be controlled by designers. Response space is defined by all the parameters that constitute the behavior space; these parameters have targets based on the mapping between customer requirements and engineering specifications associated with them. Both the design variables and response variables are special types of attributes, described in the schema for the product model in Section 3.1. It is important to note that the relationship between design variables and response variables is not defined in the problem description, but is defined in the product-specific information model. This separation of information is important for reusability as discussed in Section 1.

The constraint component of the problem definition only captures the constraints that are due to the manner in which the problem is defined. Product-specific constraints are captured as relationships in the product model. The preference part of the information model captures how much a designer values different outcomes in a manner that can be mathematically evaluated. These preferences can be captured in different mathematical forms – Archimedean, pre-emptive, or using utility functions. In the Archimedean formulation, different goals are assigned weights and the overall objective function value is evaluated by taking the weighted sum of individual goals. In the pre-emptive formulation, different levels of objective functions are defined and satisfied in order from highest to lowest.

![Figure 4. Information model for decision problems.](http://cer.sagepub.com)
Using the utility-based preference representation, preference values can be defined to vary with the value of each goal. In addition to these entities, there is a driver that captures the information about the optimization algorithm used for executing the decision problem.

The information associated with the decision problems for the design of a helical spring and a pressure vessel is shown in Figure 5. Notice the similarity in structure between the design problems. This similarity is due to the standardization of decision templates, independent of product templates. Furthermore, in this formulation, nothing is coupled to a process for execution.

### 3.3 Templates for Process Information

In the context of simulation-based design, the activities in a design process are computational tasks. Each activity has a set of inputs and outputs. The manner in which outputs of one activity are modified to serve as inputs to another activity is referred to as an interface between the two activities. Interfaces are used to perform functions such as the formatting and parsing of data. The process model also captures information regarding the sequence in which activities are performed. The information model used for capturing process information is discussed by Panchal and co-authors [14].

We recognize that the information models presented in this paper are relatively simple and defined at a high level of abstraction. However, the focus in this article is on the overall approach. More comprehensive information models can be substituted to enrich the semantics of the information models presented.

### 3.4 Templates for Interface Information

Interface templates serve as domain-independent communications protocols for regulating the way in which experts (operating in different functional domains) share information for effective collaboration. An interface in a design process separates or partitions multiple dependent or interdependent designers and their respective design activities. Interface templates serve as means for connecting decision templates to one another in a computer interpretable manner. The nature of the collaboration between designers determines the form of the interface. Appropriate interface templates are developed based upon the underlying informational dependencies between the decision-makers. Consequently, the interactions between decision-makers can be easily adapted by changing the interface template, while reusing the same problem formulations. Since the decision templates and their instantiation remains the same, the designers still control the formulation of their own decisions. The required level of modularity is maintained via the development of domain-independent interface templates that are distinct from the decision templates being linked. The information flow between the design decisions is computationally modeled and executed by an
interface template that captures the chosen game theoretic protocols, such as the iterative noncooperative techniques and other noncooperative as well as cooperative instantiations discussed in Section 2.4.

Noncooperative games represent a type of interface, where there is no communication between the stakeholders during the decision-making process. They are employed to model the solution of strongly coupled decisions, characterized by interdependent information flows, and are characteristic of decentralized design processes where designers are required to tackle design sub-problems in isolation, due to organizational barriers, time schedules, and geographical constraints. The non-cooperative protocol is represented as an interface template between decision templates (Figure 6).

Decision-makers generate a strategy or Best Reply Correspondence (BRC), which represents how they would respond given a range of possible responses from another stakeholder. Response Surface Methodology is used to construct the BRCs. An intersection is sought between these designers’ respective BRCs, and a solution can be selected from the resulting intersection.

Cooperative games, on the other hand, are ideal scenarios in which either stakeholders, or players, have full access to the information about each other’s decision-making process. They are employed to represent centralized decision-making, where all required information is available to every collaborating designer. The interface template for a cooperative game theoretic protocol essentially combines information about design variables, constraints, parameters, goals, preferences and objective functions into a single cDSP without altering the form of the templates being joined.

In addition to the decision formulation, the analysis models are combined accordingly. This combined cDSP is solved to result in a Pareto optimal solution. Note that the interface template is procedural in nature. The approach presented in Section 2 and the templates discussed in this section are generally applicable to any CAE and PLM design framework. We illustrate the fundamental ideas behind this approach using a proof-of-concept implementation in ModelCenter. Even though the implementation is simple, it provides insight into the advantages of the key principles embodied in this approach. The implementation is discussed in Section 4.

4. Implementation of the Proposed Approach in the ModelCenter Framework

The approach presented in Section 2 is implemented for two scenarios: (a) single decision and (b) multiple decisions. The first scenario is based on the assumption that a single designer formulates the design problem as a compromise decision and executes a process for making this decision. In the second scenario, multiple designers are involved in making design decisions about a product. Each designer formulates his/her decision using the cDSP construct; the required interaction between these decisions is modeled using game theoretic protocols as discussed in Section 2.4. The first scenario is used to illustrate the approach for reusability of design processes for different products (Section 1), whereas the second example is used to demonstrate the reusability of templates in collaborative design scenarios.
4.1 Single Decision Scenario – Reusability of Design Processes for Different Products

In order to increase reusability of information, we separate the information pertaining to design problem (decision), product, and generic design process in ModelCenter [8], developed by Phoenix Integration Inc. These three components are captured separately as templates. The declarative templates for problem- and product-specific information are modeled in XML. The procedural templates for the elements of generic design processes are modeled using JavaBeans and the design process is defined via generic information flows between these elements. The design problem is defined in terms of the cDSP construct. The elements of cDSP and the design processes defined as information flows between these elements remain the same for the design of either a spring or a pressure vessel as shown in Figure 7. Furthermore, required information flows between these elements are generic and independent of the product being designed. Examples of information flows between the analysis module and constraints evaluation include the problem name, an array of input names (i.e., design variables), and an array of input values. The flows remain the same regardless of the product for which the process is used. Here, the information flows relevant for the solution of a cDSP, remain invariant, regardless of whether the product being designed is a pressure vessel or a spring.

Each of the process elements defined in JavaBeans interacts with the product information templates discussed in Section 3.1 and decision templates discussed in Section 3.2. Notice that the product and decision templates presented in Section 3 can be used for reasonably complex design scenarios. However, in order to maintain the simplicity of the implementation, the templates are embodied as a set of XML schemas. XML is used in this paper because it offers a convenient means of capturing information and is supported by ModelCenter. The JavaBeans associated with process elements, discussed earlier in this section, parse required information from appropriate XML files corresponding to product and decision information and subsequently make this information available for processing in ModelCenter. Hence, the actual variable names, constraints, their values etc. are all extracted from the product and decision templates.

For the simple example problem of designing both a pressure vessel and a helical spring through the use of a common design process, the problem information is stored in four XML files: the problem definition $XML$, the constraints $XML$, the goals and preferences $XML$, and the analysis code $XML$. These templates correspond to the declarative (problem-specific) information. The XML files associated with these templates are provided in Figure 8. For brevity, the XML file representation associated with the problem definition template is not shown in this paper. The sum total of these XML files constitutes the decision template. In the current implementation, the analysis codes for simulating the behavior of both the spring and the pressure vessel are written in Visual Basic, although any
Note that the structure of the XML files is independent of the problem (spring or pressure vessel) being addressed. The generic XML files are instantiated using the product-specific information (see Figure 8 for XML files corresponding to the pressure vessel). As shown in Figure 7, the information from XML files is extracted by the generic design process elements (developed as JavaBeans) to result in an executable design process. In order to change the product from a pressure vessel to a spring, the designers only need to change the XML files that declare the specific variables, constraints, goals, analysis, etc. In fact, it is via XML file replacement that the generic processes can be instantiated readily for different products. The process defined in ModelCenter remains the same because it parses the XML files to extract problem-specific information. This is a primary advantage of the proposed approach. The proposed architecture separates the design process definition from the problem-specific information, and hence ensures reusability of the design processes being modeled for different products, satisfying the first requirement discussed Section 1.

### 4.2 Execution of Multiple Decisions – Reusability of Collaboration Strategies using Game Theory

It is assumed here for demonstration purposes that two designers (e.g., Designers A and B) are collaborating in the design of a pressure vessel. In this effort, Designer A’s goal is to minimize the overall weight of the vessel, while Designer B is in pursuit of the maximum attainable volume. Both designers are subjected to the same set of constraints, but are in control of a different set of design variables. Specifically, Designer A has control over radius $R$ and length $L$, while Designer B controls thickness $T$. The game theoretic interface templates discussed in Section 3.4 are used as domain-independent communications protocols for regulating the way in which experts (operating in different functional domains) share information for effective collaboration. The individual

---

**Figure 8.** XML files for elements of a design decision modeled using cDSP.

<table>
<thead>
<tr>
<th>Constraints</th>
<th>Goals and Preferences</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;Constraints&gt;</code></td>
<td><code>&lt;GoalsAndPreferences&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Constraint&gt;</code></td>
<td><code>&lt;Goal&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Name&gt;Length Constraint&lt;/Name&gt;</code></td>
<td><code>&lt;Name&gt;Volume&lt;/Name&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Definition&gt;Strength-(Pressure*Radius)/Thick&lt;/Definition&gt;</code></td>
<td><code>&lt;Weight&gt;0.5&lt;/Weight&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Constraint&gt;</code></td>
<td><code>&lt;Target&gt;500000&lt;/Target&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Name&gt;Radius Constraint&lt;/Name&gt;</code></td>
<td><code>&lt;Monotonicity&gt;Increasing&lt;/Monotonicity&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Definition&gt;Radius-5*Thick&lt;/Definition&gt;</code></td>
<td><code>&lt;Goal&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Constraint&gt;</code></td>
<td><code>&lt;Name&gt;Weight&lt;/Name&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Name&gt;ThickConstraint&lt;/Name&gt;</code></td>
<td><code>&lt;Weight&gt;0.5&lt;/Weight&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Definition&gt;40-Radius-Thick&lt;/Definition&gt;</code></td>
<td><code>&lt;Target&gt;300&lt;/Target&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Constraint&gt;</code></td>
<td><code>&lt;Monotonicity&gt;Decreasing&lt;/Monotonicity&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Name&gt;Stress Constraint&lt;/Name&gt;</code></td>
<td><code>&lt;Goal&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Definition&gt;150-Length-2*Radius-2*Thick&lt;/Definition&gt;</code></td>
<td><code>&lt;GoalsAndPreferences&gt;</code></td>
</tr>
</tbody>
</table>

**Analysis**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;Input&gt;&lt;Name&gt;Radius&lt;/Name&gt;&lt;Type&gt;double&lt;/Type&gt;&lt;Unit&gt;m&lt;/Unit&gt;&lt;Value&gt;1.0&lt;/Value&gt;&lt;/Input&gt;</code></td>
<td><code>&lt;Output&gt;&lt;Name&gt;Volume&lt;/Name&gt;&lt;Type&gt;double&lt;/Type&gt;&lt;Unit&gt;m3&lt;/Unit&gt;&lt;Value&gt;765054.72&lt;/Value&gt;&lt;/Output&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Input&gt;&lt;Name&gt;Length&lt;/Name&gt;&lt;Type&gt;double&lt;/Type&gt;&lt;Unit&gt;m&lt;/Unit&gt;&lt;Value&gt;2.0&lt;/Value&gt;&lt;/Input&gt;</code></td>
<td><code>&lt;Output&gt;&lt;Name&gt;Weight&lt;/Name&gt;&lt;Type&gt;double&lt;/Type&gt;&lt;Unit&gt;kg&lt;/Unit&gt;&lt;Value&gt;6845.18796333332&lt;/Value&gt;&lt;/Output&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Input&gt;&lt;Name&gt;Density&lt;/Name&gt;&lt;Type&gt;double&lt;/Type&gt;&lt;Unit&gt;kg/m3&lt;/Unit&gt;&lt;Value&gt;4.0&lt;/Value&gt;&lt;/Input&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;Input&gt;&lt;Name&gt;Strength&lt;/Name&gt;&lt;Type&gt;double&lt;/Type&gt;&lt;Unit&gt;kN/m2&lt;/Unit&gt;&lt;Value&gt;3.9&lt;/Value&gt;&lt;/Input&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;Input&gt;&lt;Name&gt;Pressure&lt;/Name&gt;&lt;Type&gt;double&lt;/Type&gt;&lt;Unit&gt;N/m2&lt;/Unit&gt;&lt;Value&gt;100&lt;/Value&gt;&lt;/Input&gt;</code></td>
<td></td>
</tr>
</tbody>
</table>

**Execute:** E:\PressureVesselAnalysisCommandLine.exe

---

other model wrapped as a ModelCenter component could be employed instead.

Note that the structure of the XML files is independent of the problem (spring or pressure vessel) being addressed. The generic XML files are instantiated using the product-specific information (see Figure 8 for XML files corresponding to the pressure vessel). As shown in Figure 7, the information from XML files is extracted by the generic design process elements (developed as JavaBeans) to result in an executable design process. In order to change the product from a pressure vessel to a spring, the designers only need to change the XML files that declare the specific variables, constraints, goals, analysis, etc. In fact, it is via XML file replacement that the generic processes can be instantiated readily for different products. The process defined in ModelCenter remains the same because it parses the XML files to extract problem-specific information. This is a primary advantage of the proposed approach. The proposed architecture separates the design process definition from the problem-specific information, and hence ensures reusability of the design processes being modeled for different products, satisfying the first requirement discussed Section 1.
decisions are modeled using the compromise DSP templates as discussed in Section 4.1 for single decision scenarios. Since the process defined in ModelCenter is generic, the process definition is directly reused for defining the individual design processes of designers A and B. Only the XML files associated with the problem definitions of the two designers are replaced. The information flow between decisions is generically modeled using the interface templates. The procedural interface templates between designers are embodied as JavaBeans in ModelCenter.

Since the interface templates are also generic and can be instantiated for any product, they can be replaced while keeping the same formulation of design decisions. Hence, the designers can explore the impact of different interfaces on the final design. This also allows designers to change control of design variables. This flexibility is a significant advantage from the standpoint of reconfiguring design processes and addresses the second requirement presented in Section 1, namely the ability to reuse collaborative design strategies.

The communication or interaction protocols currently instantiated within the interface template are those of cooperative and noncooperative behavior. As evidenced in Table 1, the nature of the interaction protocol used to structure the collaboration among the interacting decision-makers has a significant effect on the outcome attained. The best outcome (i.e., objective function, $Z = 0.4958$) is obtained for full cooperation, while there is a significant spread in objective values obtained for noncooperative behavior, ranging from $Z = 0.4976$ to $Z = 0.9077$. In fact, the fully cooperative outcome among the two interacting decision-makers, each in pursuit of their own objectives, matches that obtained for a single decision-maker. In the case of noncooperative behavior it is clear that the results obtained are directly affected by decision-maker precedence and design variable control. While in this particular case, the average effect of precedence and control on the objective value obtained is almost the same (i.e., $Z = 0.2446$ and $Z = 0.2396$, respectively), the comparative significance of these effects is problem-specific and depends directly on the specifics of the particular problem at hand – constraints, design variable sensitivities, etc.

As illustrated at the hand of the pressure vessel design problem, explored for two collaborating decision-makers with regard to (1) use of interaction protocol, (2) order of precedence, and (3) control over design variables in this section, the design process can have a significant effect on the outcome obtained. This is especially true in light of continuously evolving information content. It is our objective to provide a consistent means of interfacing collaborating decision-makers, whose design sub-problems are modeled in a consistent fashion. Using the modeling approach presented in this paper, it becomes possible to explore different design process scenarios and effectively leverage an engineering enterprise’s intellectual capital [30].

5. Conclusions

In this article, we address two aspects of lack of reusability of the current CAE and PLM frameworks: (a) reusability of computational design processes across different products, and (b) reusability of collaborative design strategies. To mitigate these limitations, we present a modular, decision-centric, template-based approach for modeling design information. The fundamental differences between the proposed approach and other commonly available approaches are that in the proposed approach: (a) design processes are modeled as hierarchical systems; (b) declarative and procedural models are effectively separated in the form of templates; (c) design processes are viewed as decision-making processes; and (d) interactions between designers are modeled using game theoretic protocols.
These four fundamentals of the approach are instantiated in the form of generic computational templates for products, processes, decisions and interfaces. The proposed templates are computer interpretable, allowing for the execution, and reuse/reconfiguration of design processes and any of their components. A modular, generic formulation of the process required for the solution of design decisions defined as compromise DSPs is conceived and presented. Consequently, it is possible to effectively model design processes and sub-processes, regardless of functional domain or complexity. The approach has been formalized and the resulting approach implemented in ModelCenter, as shown in Section 4. The instantiation is validated for two examples (i.e., pressure vessel and spring design) in order to show the reusability of the generic process (Section 4.1). This addresses the first limitation of the existing frameworks. Finally, to address the second limitation, game theoretic collaborative design strategies are implemented as interaction templates and shown via the integration of interacting decision-makers sharing responsibility for the design of the pressure vessel (Section 4.2).

The principal advantage of this modeling technique is the enhanced reusability of information and knowledge achieved via the separation of information pertaining to problem formulation, process, product, and interfaces between decision-makers. In this article, the proposed approach is only demonstrated for computational design processes in the later stages of design. While it is possible to formulate templates, at least structurally, even in the early stages of design, the information, and knowledge gained by exercising the resulting models becomes more concrete as the design matures. The advantage of relying on completely modular templates is the provision of a consistent means of capturing and exploiting knowledge that reflects evolving information content throughout the design process. We believe that the proposed approach is a stepping-stone towards top-down design of design processes and any of their components. The ability to rely upon a repository of design process building blocks will greatly facilitate the original, adaptive, derivative, and variant design of products and serve as a springboard for the effective evolution of product portfolios by engineering enterprise.

Acknowledgments

We acknowledge the support from National Science Foundation grants DMI-0085136 and DMI-0100123, as well as, Air Force Office of Scientific Research grant F49620-03-1-0348. During this research, Marco Gero Fernández was sponsored by a National Science Foundation IGERT Fellowship through the TII:GER Program (NSF IGERT-0221600) at the Georgia Tech College of Management and a President’s Fellowship from the Georgia Institute of Technology.

References


Christiaan J. J. Paredis

Dr Paredis is an Associate Professor in the G.W. Woodruff School of Mechanical Engineering at Georgia Tech. He received his MS degree in Mechanical Engineering from the Catholic University of Leuven (Belgium) in 1988, and his MS and PhD in Electrical and Computer Engineering from Carnegie Mellon University in 1990 and 1996, respectively. From 1996 to 2002, he was a Research Scientist at the Institute for Complex Engineered Systems at Carnegie Mellon University. Dr. Paredis has a broad, multidisciplinary background. In his research, he combines aspects of information technology, simulation, and systems theory to support the design of mechatronic systems, focusing in particular on decision-making under uncertainty in conceptual design. In these areas, he has published more than 80 refereed journal articles and conference papers. Dr. Paredis received the 2007 CETL/BP Junior Faculty Teaching Excellence Award and the 2007 SAE Ralph R. Teetor Educational Award. In 2007–2008, he was the Chair of the ASME Computers and Information in Engineering (CIE) Division.

Jitesh H. Panchal

Dr Jitesh H. Panchal is an Assistant Professor at the School of Mechanical and Materials Engineering at Washington State University. He received his Bachelor of Technology from Indian Institute of Technology at Guwahati, and MS and PhD in Mechanical Engineering from Georgia Institute of Technology. His research interests are in the field of Engineering Systems Design. Specifically, his current research focus is on simulation-based distributed design methodologies and technologies for multiscale systems. Dr. Panchal is a member of American Society of Mechanical Engineers (ASME) and American Institute of Aeronautics and Astronautics (AIAA).
Marco Gero Fernández

Marco Gero Fernández received his PhD in Mechanical Engineering from the George W. Woodruff School of Mechanical Engineering at the Georgia Institute of Technology in 2005. As a graduate student, he was a National Science Foundation Research Fellow, a National Science Foundation Integrative Graduate Education and Research Traineeship Program Fellow, and a Georgia Tech Presidential Fellow. He received a BSE from Duke University in 1999, an MS from the Georgia Institute of Technology in 2002, and an MBA from the College of Management at the Georgia Institute of Technology in 2006. He was a member of the Systems Realization Laboratory from 1999 to 2006 and currently works in Business Transformation Consulting for the Consulting Services Practice of Capgemini Consulting.

Farrokh Mistree

Professor Farrokh Mistree’s design experience spans mechanical, aeronautical, structural, and industrial engineering. His teaching experience spans courses in engineering design, naval architecture, solid mechanics, operations research, and computer science. His current research focus is on learning how to manage design freedom in multiscale design (from molecular to reduced order models) to facilitate the integrated design of materials, product, and design process chains. And what does Farrokh Mistree want to do in the future? He wants to have fun—fun in defining the emerging science-based discipline of design. Fun in providing an opportunity for highly motivated and talented people to learn how to achieve their dreams.

Janet K. Allen

Professor Allen’s research is in the area of systems design of complex, multiscale products especially the design of integrated products and materials. She has a long-standing interest in robust design and understanding the effects of uncertainty in simulation-based design. Dr Allen is a professor in the George W. Woodruff School of Mechanical Engineering at Georgia Tech and a Fellow of ASME. Dr Allen received her PhD from the University of California, Berkeley and her SB from the Massachusetts Institute of Technology, she has authored more than 200 technical publications.