Noname manuscript No.
(will be inserted by the editor)

Extensions to the Design Structure Matrix for the Description of Multidisciplinary Design, Analysis, and Optimization Processes

Andrew B. Lambe · **Joaquim R. R. A. Martins**

Received: date / Accepted: date

Abstract While numerous architectures exist for solving multidisciplinary design optimization (MDO) problems, there is currently no standard way of describing these architectures. In particular, a standard visual representation of the solution process would be particularly useful as a communication medium among practitioners and those new to the field. This paper presents the extended design structure matrix (XDSM), a new diagram for visualizing MDO processes. The diagram is based on extending the standard design structure matrix (DSM) to simultaneously show data dependency and process flow on a single diagram. Modifications include adding special components to define iterative processes, defining different line styles to show data and process connections independently, and adding a numbering scheme to define the order in which the components are executed. This paper describes the rules for constructing XDSMs along with many examples, including diagrams of several MDO architectures. Finally, this paper discusses potential applications of the XDSM in other areas of MDO and the future development of the diagrams.

This work was presented by the authors under the title "A Unified Description of MDO Architectures" at the 9^{th} World Congress on Structural and Multidisciplinary Optimization. This work was partially funded by a postgraduate scholarship from the Natural Sciences and Engineering Research Council of Canada.

A. B. Lambe
Institute for Aerospace Studies
University of Toronto
Toronto, Ontario, Canada
E-mail: lambe@utias.utoronto.ca

J. R. R. A. Martins
Department of Aerospace Engineering
University of Michigan
Ann Arbor, Michigan, USA
E-mail: jrram@umich.edu

Keywords Multidisciplinary Design Optimization · Visualization · Design Structure Matrix · Design Architectures · Multidisciplinary Analysis · Distributed Optimization

1 Introduction

When employing multidisciplinary design optimization (MDO) to solve a given design problem, designers are confronted with a wide range of options. For example, should the optimizer interact with the analysis software in a "black-box" fashion or should the governing equations be provided to the optimizer directly? Should a surrogate model be employed? Which disciplines or system components should be resolved in a coupled fashion? Should decomposition be employed in the optimization problem? Should the decomposition employ a penalty function scheme or a multilevel scheme? In practice, these kinds of questions are answered on a case-by-case basis after considering the problem characteristics, distribution of expertise, level of communication between designers, and available software tools. However, the answers to these questions strongly influence both the optimization problem formulation and the solution strategy employed which, in turn, strongly influence the time and resources needed to find a design solution.

We refer to the combination of the design problem formulation and the organizational framework used to solve it as an MDO *architecture*. Many architectures have been developed, including multidisciplinary feasible (MDF) (Cramer et al, 1994), individual discipline feasible (IDF) (Cramer et al, 1994), concurrent subspace optimization (CSSO) and its variants (Bloebaum et al, 1992; Sellar et al, 1996; Wujek et al, 1996), collaborative optimization (CO) and its variants (Braun and Kroo, 1997; Kroo, 1997; DeMiguel and Murray, 2000; Roth, 2008), bilevel integrated system synthesis (BLISS) and its variants (Sobieszczanski-Sobieski et al,

2000, 2003; Kodiyalam and Sobieszczanski-Sobieski, 2000), and analytical target cascading and its variants (Kim, 2001; Kim et al, 2003, 2006; Tosserams et al, 2006). Determining the conditions under which each architecture is most effective and refining their performance remains an active area of research.

Once an architecture has been selected, it must be implemented in the computational environment. As with other computational methods, incorrect implementation of the MDO architecture can prevent solution of the design problem. However, determining the correct implementation can be difficult because there is no standard way of describing the solution process. Each author uses his/her own notational set, algorithmic description, and system of diagrams in describing a new architecture. The lack of a standard representation of MDO architectures is a barrier to communicating the correct implementation and describing alternative approaches.

This work is complementary to a number of other studies in the MDO literature that seek more flexible ways of describing MDO problems and solution methods. Alexandrov and Lewis (2004a,b) proposed a linguistic approach to problem formulation called reconfigurable multidisciplinary synthesis (REMS). The REMS framework used a small set of computational components along with a directed graph representation to allow the user to quickly reformulate a specified MDO problem. Later work by Tosserams et al (2010) created the Ψ language to allow the user greater freedom in the assembly of the problem formulation, especially for decomposed problems. Another language, called χ, was created by Etman et al (2005) to specify the coordination strategy for parallel solution processes. While linguistic approaches like these are useful for automatically implementing MDO architectures in a computational environment (Martins et al, 2009), they do not provide a formal visual representation of how the architecture handles the original problem.

Our work is most similar to the work of De Wit and Van Keulen (2010), who present a unified set of mathematical notation and diagrams to describe a broad range of decomposition and coordination strategies. Unlike the linguistic approaches mentioned above, the diagrams provide a visual framework for describing MDO architectures. However, in that work, multiple diagrams are required to explain the decomposition and coordination strategies employed. We feel that a similar approach using only one diagram would be a more accessible starting point, facilitating communication about MDO architectures among experts and nonexperts alike.

In this work, we present our approach to the problem of creating a standard visual representation of the solution process of an MDO architecture. While this paper is focused on MDO architectures in particular, we have found that our diagram syntax may be more broadly applicable to a range of analysis and optimization processes. Indeed, we should expect this, given the commonality in the computational components between the processes. We also integrate mathematical notation into the diagrams and show links between the elements of the diagram and the underlying mathematics. We refer to our diagram as an extended design structure matrix, or XDSM.

This paper is organized as follows. Section 2 introduces the mathematical notation and reviews basic MDO terminology. Section 3 discusses the guiding design principles and evaluates alternative diagrams for their use in describing MDO processes. Section 4 outlines the elements and syntax of the XDSM with simple examples. Section 5 shows four different MDO architectures represented using XDSMs. Section 6 discusses potential applications of the XDSM beyond MDO architectures. Finally, Section 7 provides concluding remarks.

2 Terminology and Notation

Before discussing the diagrams themselves, we introduce the notation that is used throughout this paper. This notation was developed to compare the various problem formulations within MDO architectures and is listed in Table 1. This is not a comprehensive list; additional notation specific to particular architectures is introduced when the respective architectures are described. We also take this opportunity to clarify many of the terms we use that are specific to the field of MDO.

A *design variable* is a variable in the MDO problem that is always under the explicit control of an optimizer. Design variables may pertain to a single discipline, i.e., be *local*, or may be *shared* by multiple disciplines. We denote the vector of design variables local to discipline i by $\mathbf{x}_i$ and the shared variables by $\mathbf{x}_0$. The full vector of design variables is given by $\mathbf{x} = \left[\mathbf{x}_0^T, \mathbf{x}_1^T, \ldots, \mathbf{x}_N^T\right]^T$. The subscripts for local and shared data are also used in describing objectives and constraints.

A *discipline analysis* is a simulation that models the behavior of one aspect of a multidisciplinary system. Conducting a discipline analysis consists of solving a system of equations and returning a set of *response variables*. The response variables may or may not be controlled by the optimization, depending on the formulation. We denote the vector of response variables computed within discipline i by $\mathbf{y}_i$. Analogously to the design variables, the full vector of response variables is denoted $\mathbf{y}$. In a multidisciplinary system, most disciplines are required to exchange their response variables with other disciplines to model the interactions of the whole system. Often, the number of variables exchanged is much smaller than the total number of response variables computed in a particular discipline. Those variables that are exchanged are referred to as *coupling variables*.

Table 1 Mathematical notation for MDO problem data

Symbol	Definition
$\mathbf{x}$	Vector of design variables
$\mathbf{y}^t$	Vector of coupling variable targets (inputs to a discipline analysis)
$\mathbf{y}$	Vector of coupling variable responses (outputs from a discipline analysis)
f	Objective function
$\mathbf{c}$	Vector of design constraints
$\mathbf{c}^c$	Vector of consistency constraints
N	Number of disciplines
$()_0$	Functions or variables that are shared by more than one discipline
$()_i$	Functions or variables that apply only to discipline i
$()^*$	Functions or variables at their optimal value
$\tilde{()}$	Approximation of a given function or vector of functions

In many formulations, copies of the coupling variables must be made to allow discipline analyses or optimizations to run independently and in parallel. These copies are known as *target* variables and are denoted by a superscript t. For example, the copy of the response variables produced by discipline i is denoted $\mathbf{y}_i^t$. These variables are used as the input to disciplines other than discipline i that are coupled to discipline i through $\mathbf{y}_i$. To preserve consistency between the coupling variable inputs and outputs at the optimal solution, we define a set of *consistency constraints*, $\mathbf{c}_i^c = \mathbf{y}_i^t - \mathbf{y}_i$, that we add to the problem formulation. (These constraints are sometimes referred to as *compatibility constraints*. However, we use the term *consistency* to avoid confusion with the compatibility constraints present in structural analysis.)

Each of these elements of the MDO problem is handled differently in different architectures. Several of the problem formulations are discussed in Section 5 along with their corresponding XDSMs.

3 Design Principles

We now return to the primary objective of this paper: developing a standard visual representation of an MDO solution process. In particular, we want to capture the basic algorithm and all the data connections in a single diagram. We based the graphic design on the following guiding principles:

1. *Simplicity.* MDO processes can be complex because of the number of different analysis codes needed to model a coupled system and the volume and varied types of data that these codes exchange. In spite of this complexity, the diagram syntax should have as few rules and symbols as possible to make the description of the process clear. We expect the average user to understand the syntax well enough that they can create their own diagrams after an hour of study.
2. *Clarity.* The depiction of the data exchange and algorithm should have an unambiguous meaning to the viewer. We are particularly concerned with the reproducibility of the depicted processes and avoiding errors in communication. This is especially important in explaining MDO processes to nonspecialists and those new to the field.
3. *Information Density.* Because MDO processes involve many computational elements exchanging data within an overall algorithm, we wish to depict both the algorithm and the data exchanges simultaneously. Incorporating data and process into a single diagram gives us an immediate impression of the whole software architecture without having to compare and cross-check multiple diagrams.
4. *Integration of Mathematics.* Because MDO processes incorporate many mathematical representations in the form of systems of equations, optimization problem statements, inequalities defining convergence conditions, etc., the diagram should be able to incorporate the notation where possible to enhance understanding. This should be done so long as it does not interfere with the other principles outlined above. If the mathematics cannot be incorporated into the diagram directly, the locations of the relevant expressions should be obvious to the viewer. Using mathematics directly in the diagram also helps to compact the data into a small space without loss of information (Tufte, 1983).

Using these principles, we now review some commonly used diagrams in the MDO and complex system literature.

The first diagram we review is the standard flowchart. Flowcharts depict algorithms by representing simple operations (selection, initialization, input and output, etc.) by different block shapes connected by lines. Flowcharts can also be scaled to show large, complex algorithms. While these diagrams are simple to understand and have been used successfully for many years, they do not track data movement among the computer program components. Because of the nature of the MDO computing environment, with many computational elements exchanging data at key points in the algorithm, some representation of the data transfer is also necessary. Separate data flow diagrams may be employed

but, as mentioned above, we prefer to use a single diagram. Adding data transfer lines to a flowchart is another possibility. However, the number of lines required in complex architectures could turn the diagram into "spaghetti" and make it hard to interpret.

A common visualization technique used in the MDO literature is some type of block diagram (Braun et al, 1996; Sobieszczanski-Sobieski et al, 2000; Perez et al, 2004). Typically, these diagrams use blocks to represent computational components and labeled arrows to show data flow. These diagrams suffer from the opposite problem of the flowchart: they show the data exchange well but do not clearly define the process. This is especially true when multiple loops are present in the algorithm or when certain components are reused within a loop. Another consideration is that there is no standard way of constructing the block diagram, so each diagram must be interpreted individually.

Another option for visualization is the unified modeling language (UML) (Booch et al, 2005). UML was designed to model and visualize software systems, especially object-oriented software, under a unified standard. (An example of UML class diagrams is given by Martins et al (2009).) Because of the close connections between object-oriented software and MDO (Alexandrov and Lewis, 2004a,b), we may expect that the same visualization tools can be used in both cases. In contrast to the other methods that we have described, UML is standardized, and this adds to its appeal for widespread communication. However, describing an MDO process using UML would require multiple diagrams to show both the process and data flows, a situation that we want to avoid. We also feel that UML is difficult to learn, and this is an additional barrier to communication about MDO.

Finally, a diagram that is common in systems engineering is the N^2 diagram (Lano, 1977) or design structure matrix (DSM) (Steward, 1981). An example DSM for an aircraft design problem is shown in Fig. 1. While several different types of DSM exist (Browning, 2001), the basic diagram is unchanged. The components of a system are placed on the main diagonal of a matrix with the inputs to each component placed in the same column and the outputs from each component placed in the same row. External inputs and outputs, if they are considered, are placed on the outside edges of the diagram. The key advantage of DSM is a strict diagram structure. Whereas traditional block diagrams and flowcharts allow the system components and interconnections to be placed arbitrarily, the DSM structure effectively "untangles" the collection of components and gives the reader a complete view of the coupling structure within the system. The DSM in Fig. 1 further enhances this view by using larger dots to denote stronger coupling between the disciplines. Other notation may be used to show more coupling information, e.g., see Yassine and Braha (2003).

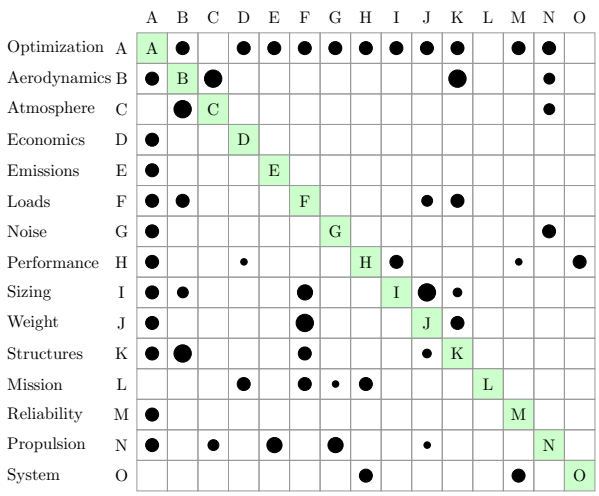

Fig. 1 Example DSM for aircraft design problem.

The principal disadvantage of the DSM lies in the definition of the order of execution of the components. While some DSMs show a sequence by ordering the components along the diagonal of the matrix (Browning, 2001), we require more flexibility in the ordering because of the selection, iteration, and parallel computing structures present in MDO architectures. These structures can be considered a high-level algorithm driving the architecture. Furthermore, we would like to maintain the property of the standard DSM that a reordering of the components does not change the underlying system. Therefore, we have chosen to add additional graphical features to the DSM to show process flows at the same time as data flows. These features are described in detail in the next section.

4 XDSM Syntax Description

As explained above, the DSM structure offers a convenient and compact way to represent the connections between system components. When the DSM is applied to the specific case of a computational MDO environment, the components of the system are computational elements (disciplinary analyses, optimizers, surrogate models, etc.), and the connections consist of the data passed between them. However, for processes, such as the algorithms of MDO architectures, we require the diagram to show the order in which the components are executed without ordering the components. Furthermore, we need a way to show iterative procedures on the diagram. This section describes the extensions necessary to create an XDSM from a DSM.

(All XDSMs shown in this work were created in LaTeX using the TikZ package. To simplify the diagram creation, we have developed template files containing all necessary block shapes and line styles. These templates and some ex-

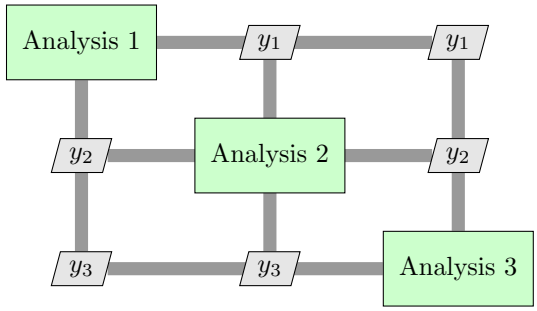

Fig. 2 Generic, three-discipline, fully coupled system.

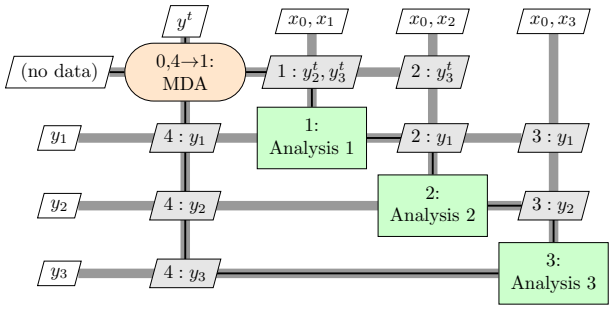

Fig. 3 Gauss–Seidel MDA procedure.

ample XDSM source code are freely available for download at `http://mdolab.engin.umich.edu/content/xdsm-overview`. Comments and suggestions are welcome.)

We start with the task of performing a multidisciplinary analysis (MDA) on a coupled system with three disciplines. An MDA is an iterative process that computes a set of coupling variables that satisfies the consistency constraints of a coupled system. Figure 2 shows the original DSM for this system. In this diagram, we have used different block shapes to distinguish between the computational analyses and the data connections. The shapes were chosen for their similarity with flowchart syntax: rectangles for generic processes, parallelograms for data input and output. The data dependency of the components is further emphasized by the use of thick gray lines to connect the nodes.

In a Gauss–Seidel MDA process, each analysis is executed in sequence using a set of design variables (chosen outside the MDA) and the most recent information on the states of the other disciplines. The analyses continue until a consistent set of state variables is returned by the individual disciplines. These states are then returned to the user or some external process. Figure 3 shows the XDSM for the Gauss–Seidel MDA.

We now address the new elements that we have introduced to create an XDSM. To control the iteration we have introduced a special component, labeled "MDA," whose function is to distribute the state target information appropriately and to check the convergence of the MDA process. We refer to the components controlling an iterative procedure as *drivers*. We use a separate shape (a rounded rectangle, in this case) to distinguish drivers from other components.

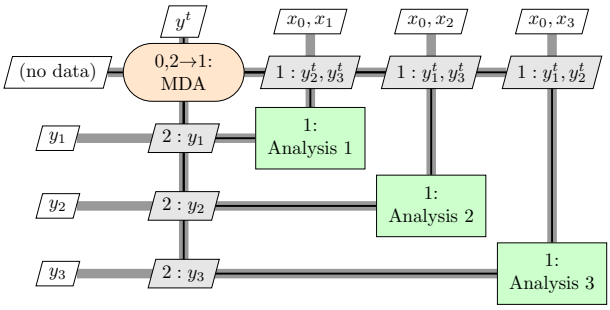

Fig. 4 Jacobi MDA procedure with parallel execution of components.

The order in which the components are executed is defined by numbering the components, starting at zero. The components are executed in numerical order unless a loop is present. A loop is denoted by $m \to n$, where $n < m$. This notation means that before proceeding to $m+1$, the sequence returns to component n until the condition defined by the driver is satisfied. The data blocks are also numbered to clarify the step at which the data are passed to the component. This is useful in more complicated diagrams where the same component takes input from different sources at different steps. (The BLISS-2000 architecture depicted in Fig. 12 is a good example of this.) Finally, to give an added visual cue to the process flow, we use a distinct line style to connect consecutive components. In our XDSMs, a thick gray line represents data connections, while a thin black line represents process connections. These line styles were chosen so that they could coexist on the diagram, i.e., so that one does not overwrite the other.

The same MDA can be solved using other types of iterations, such as Jacobi iterations. Figure 4 shows a Jacobi iteration, where all the analysis components are executed using the state information from the previous Jacobi iteration. To show that each of these components may be executed in parallel, we use the same number for each component. Note that no data are exchanged between these components during the parallel execution. If it is not possible to evaluate the discipline analyses in parallel, we can also use the XDSM to capture sequential execution. A Jacobi iteration with sequentially executed components is depicted in Fig. 5.

Because the situation in Fig. 4 of many similar components being executed in parallel occurs frequently in MDO, we have adopted a special convention to make the diagram more compact. The convention is that a reference to component i implies that the diagram structure is repeated for all disciplines. As an example, Fig. 6 is identical to Fig. 4,

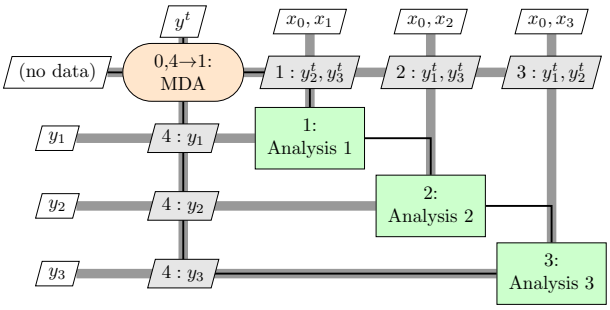

Fig. 5 Jacobi MDA procedure with sequential execution of components.

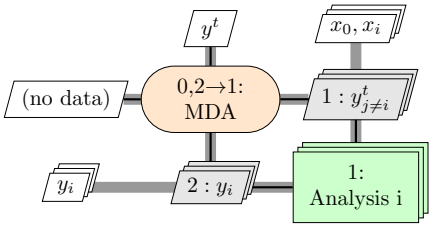

Fig. 6 Jacobi MDA procedure with parallel execution of components using our convention for parallel diagram structure. Process shown here is identical to that shown in Fig. 4.

except that it applies this convention. The presence of the Analysis i block in Fig. 6 implies that all the disciplinary analyses are evaluated in parallel before the algorithm continues. This convention is further enhanced by showing stacks of component and data blocks.

Using our syntax, we can also describe optimization processes. Figure 7 shows the solution of a standard nonlinear programming problem. An initial guess of the design variables, $\mathbf{x}^{(0)}$, is passed to the optimizer. In each iteration, the objective and constraints are evaluated, (in parallel, in this case,) and an updated set of design variables is computed. The iterations continue until the optimization driver converges, returning the optimal solution $\mathbf{x}^*$. Note that Fig. 7 is independent of the type of optimizer used, i.e., it is independent of the specific calculations that take place within the optimization driver. If the optimizer also required the gradients of the objective and constraint functions, these could be placed in the diagram as additional components with the appropriate connections to the optimization driver. Figure 8 shows an optimization algorithm with a separate component that computes all the gradient information required by the optimizer.

To conclude this section, we make a few remarks on the relationship between the XDSMs we have constructed and the underlying mathematics of the MDA and optimization processes. The key point is that, in all of our diagrams, each component corresponds to a set of mathematical ex-

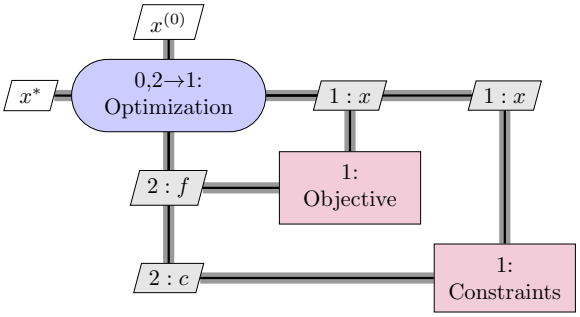

Fig. 7 Simple optimization algorithm.

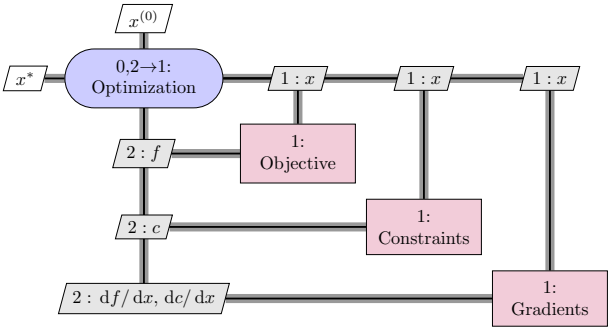

Fig. 8 Optimization algorithm where optimizer requires gradient information. Gradients are calculated by separate component.

pressions, and entry into each component corresponds to the evaluation of these expressions. For example, evaluating the "Constraints" component in Fig. 7 or 8 is the same as computing the values of all the constraint functions in the nonlinear programming problem. The "MDA" and "Optimization" drivers evaluate expressions that compute the next point to test in the iteration. The stopping criterion for the iteration is also checked by the driver. We emphasize that this interpretation extends to all components in all XDSMs, closely linking the mathematics of the MDO problems to the XDSM description of the solution process.

5 MDO Architecture Examples

We now have all the tools needed to provide a complete description of an MDO architecture. We describe only a few architectures in detail here. However, we note that we have been able to describe a total of fifteen general architectures using the XDSM, all of which will be presented in detail in a forthcoming survey paper.

In the multidisciplinary feasible (MDF) architecture (Cramer et al, 1994), all the disciplines are coupled together, and an MDA is performed to compute the full set of state variables for a given choice of design variables. MDF is effectively an

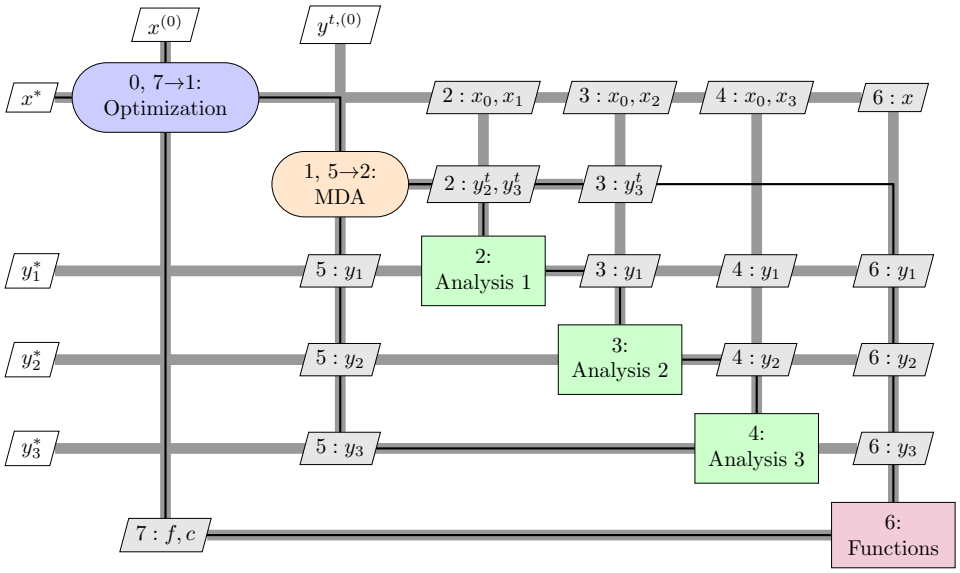

Fig. 9 Multidisciplinary feasible (MDF) architecture.

extension of a single-discipline optimization to the multiple-discipline case, where the MDA replaces a single-discipline analysis. The optimization problem formed in this architecture is given by

$$\begin{aligned} \text{min.} \quad & f_0\left(\mathbf{x}, \mathbf{y}\left(\mathbf{x}\right)\right) \\ \text{w.r.t.} \quad & \mathbf{x} \\ \text{s.t.} \quad & \mathbf{c}_0\left(\mathbf{x}, \mathbf{y}\left(\mathbf{x}\right)\right) \geq \mathbf{0} \\ & \mathbf{c}_i\left(\mathbf{x}_0, \mathbf{x}_i, \mathbf{y}_i\left(\mathbf{x}_0, \mathbf{x}_i, \mathbf{y}_{j\neq i}\right)\right) \geq \mathbf{0} \quad i = 1, \ldots, N. \end{aligned} \tag{1}$$

Note the functional notation. Objective f_0 is a function of both the design variables $\mathbf{x}$ and state variables $\mathbf{y}$. Because of the MDA process, the state variables are themselves functions of $\mathbf{x}$.

Figure 9 shows the XDSM for MDF, using the Gauss–Seidel MDA shown in Fig. 3. The sequence of operations can be inferred from Fig. 9 following the numbering rules defined previously. The operations corresponding to each number in the diagram are as follows:

0. Pass initial data to Optimization and MDA drivers.
1. Initiate MDA.
2. Evaluate Analysis 1.
3. Evaluate Analysis 2.
4. Evaluate Analysis 3.
5. Check MDA convergence. If it has not converged, return to 2; otherwise, continue.
6. Compute objective and constraint function values.
7. Compute new design point. If optimization has not converged, return to 1; otherwise, return optimal solution.

Note that we have not chosen the convergence criteria of the MDA or the optimization, but we have specified where these convergence criteria are applied. These criteria can be chosen by the user and assigned to the corresponding architecture loop. There is also flexibility in the choice of optimizer and multidisciplinary analysis method; we need not use the Gauss–Seidel strategy shown in Fig. 9. If, for example, a Jacobi or Newton-like strategy is available, it may be substituted into the architecture. For the purpose of this work, we have tried to keep the description of both problem and solution strategy as generic as possible. Specific implementation details are left to the user, but all can be described with an XDSM without difficulty.

In the individual discipline feasible (IDF) architecture (Cramer et al, 1994), individual disciplines are not coupled together in the analysis of the system. Instead, coupling variable targets are used to share information between disciplines, and consistency constraints are used in the optimization problem to ensure that the coupling variable inputs match the disciplinary outputs at the optimal solution. The resulting optimization problem is given by

$$\begin{aligned} \text{min.} \quad & f_0\left(\mathbf{x}, \mathbf{y}\left(\mathbf{x}, \mathbf{y}^t\right)\right) \\ \text{w.r.t.} \quad & \mathbf{x}, \mathbf{y}^t \\ \text{s.t.} \quad & \mathbf{c}_0\left(\mathbf{x}, \mathbf{y}\left(\mathbf{x}, \mathbf{y}^t\right)\right) \geq \mathbf{0} \\ & \mathbf{c}_i\left(\mathbf{x}_0, \mathbf{x}_i, \mathbf{y}_i\left(\mathbf{x}_0, \mathbf{x}_i, \mathbf{y}^t_{j\neq i}\right)\right) \geq \mathbf{0} \quad i = 1, \ldots, N \\ & \mathbf{c}_i^c = \mathbf{y}_i^t - \mathbf{y}_i\left(\mathbf{x}_0, \mathbf{x}_i, \mathbf{y}^t_{j\neq i}\right) = \mathbf{0} \quad i = 1, \ldots, N. \end{aligned} \tag{2}$$

Note that the coupling variable outputs, $\mathbf{y}$, are treated as functions of design variables and coupling variable inputs, $\mathbf{y}^t$, because each iteration of the method involves the evaluation of each discipline analysis. Therefore, coupling vari-

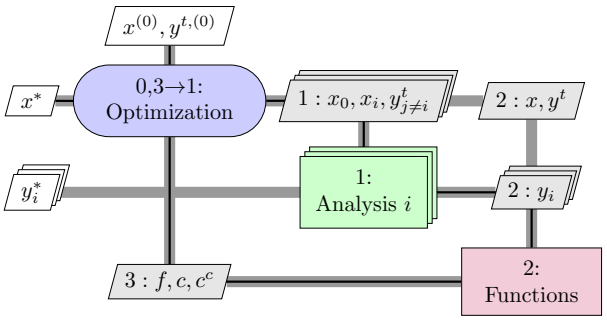

Fig. 10 Individual discipline feasible (IDF) architecture.

able outputs may take only values that satisfy the governing equations of each discipline.

Figure 10 shows the XDSM for IDF. Note the depiction of the parallel analyses using our convention for repeated diagram structure. If parallel processing was not available, the disciplinary analyses could instead be evaluated independently in sequence. The sequence of operations defined by Fig. 10 is the following:

0. Pass initial data to Optimization driver.
1. Evaluate all Analysis components in parallel.
2. Compute objective and constraint function values.
3. Compute new design point. If optimization has not converged, return to 1; otherwise, return optimal solution.

As with MDF, we have presented a generic version of IDF, and the specific implementation details are left to the user.

The collaborative optimization (CO) architecture (Braun and Kroo, 1997) is quite different from the previous architectures in that a decomposition and coordination scheme is introduced to allow each discipline to operate with greater autonomy. As a result, an optimization subproblem is defined for each discipline in addition to the top-level system optimization subproblem. The system subproblem is given by

$$\begin{aligned} \text{min.} \quad & f_0\left(\mathbf{x}_0, \hat{\mathbf{x}}_{1\ldots N}, \mathbf{y}^t\right) \\ \text{w.r.t.} \quad & \mathbf{x}_0, \hat{\mathbf{x}}_{1\ldots N}, \mathbf{y}^t \\ \text{s.t.} \quad & \mathbf{c}_0\left(\mathbf{x}_0, \hat{\mathbf{x}}_{1\ldots N}, \mathbf{y}^t\right) \geq \mathbf{0} \\ & J_i^* = ||\hat{\mathbf{x}}_{0i} - \mathbf{x}_0||_2^2 + ||\hat{\mathbf{x}}_i - \mathbf{x}_i||_2^2 + \\ & ||\mathbf{y}_i^t - \mathbf{y}_i\left(\hat{\mathbf{x}}_{0i}, \mathbf{x}_i, \mathbf{y}^t_{j\neq i}\right)||_2^2 = 0 \qquad i = 1, \ldots, N, \end{aligned} \tag{3}$$

whereas the discipline i problem is given by

$$\begin{aligned} \text{min.} \quad & J_i\left(\hat{\mathbf{x}}_{0i}, \mathbf{x}_i, \mathbf{y}_i\left(\hat{\mathbf{x}}_{0i}, \mathbf{x}_i, \mathbf{y}^t_{j\neq i}\right)\right) \\ \text{w.r.t.} \quad & \hat{\mathbf{x}}_{0i}, \mathbf{x}_i \\ \text{s.t.} \quad & \mathbf{c}_i\left(\hat{\mathbf{x}}_{0i}, \mathbf{x}_i, \mathbf{y}_i\left(\hat{\mathbf{x}}_{0i}, \mathbf{x}_i, \mathbf{y}^t_{j\neq i}\right)\right) \geq \mathbf{0}. \end{aligned} \tag{4}$$

The notations $\hat{\mathbf{x}}_{0i}$ and $\hat{\mathbf{x}}_i$ are used to denote design variable copies introduced to separate the system and discipline subproblems as much as possible. The variables $\hat{\mathbf{x}}_{0i}$ are copies of the $\mathbf{x}_0$ design variables employed in the discipline i subproblem, and the $\hat{\mathbf{x}}_i$ variables are copies of the $\mathbf{x}_i$ design variables employed in the system problem. In practice, only the variables that appear directly in nonlocal objective and constraint function evaluations—i.e., not through a discipline analysis—need to be copied.

The J_i functions defined in CO preserve consistency between the variable copies contained in each subproblem at optimality. Therefore, the goal of the discipline subproblems is to minimize inconsistency, whereas the system subproblem minimizes a system objective while maintaining consistency among the optimized discipline subproblems.

Figure 11 shows the XDSM for CO. The sequence of operations shown is the following:

0. Pass initial data to both system and discipline optimizers.
1. Compute system subproblem objectives and constraints, and do the following in parallel:
   1.0. Initiate disciplinary subproblem optimizations.
   1.1. Evaluate all disciplinary analyses.
   1.2. Compute disciplinary subproblem objectives and constraints.
   1.3. Compute new disciplinary subproblem design points. If subproblem optimization has not converged, return to 1.1; otherwise, return J_i function values to system subproblem.
2. Compute new system subproblem design point. If system subproblem optimization has not converged, return to 1; otherwise, return optimal solution.

Note that each discipline subproblem must be solved once to complete a single iteration of the system subproblem. This is a standard bilevel optimization approach. In Fig. 11, nested numbering is used to display the parallel operations of solving the disciplinary optimization problems and evaluating the system subproblem objective and constraints. This organization is allowed because the system functions do not require information from the discipline subproblems. If desired, these operations could also be done in sequence. Finally, we emphasize that, even though the disciplinary subproblems are solved in parallel using the same sequence of operations, those operations need not be synchronized between subproblems. In other words, each disciplinary optimization operates independently of the others and it is only when all the disciplinary subproblems are solved that the algorithm proceeds.

The final architecture that we describe is the BLISS-2000 variant (Sobieszczanski-Sobieski et al, 2003) of bilevel integrated system synthesis (BLISS). Like CO, BLISS-2000 decomposes the optimization problem into system and discipline subproblems. Unlike CO, BLISS-2000 uses surrogate

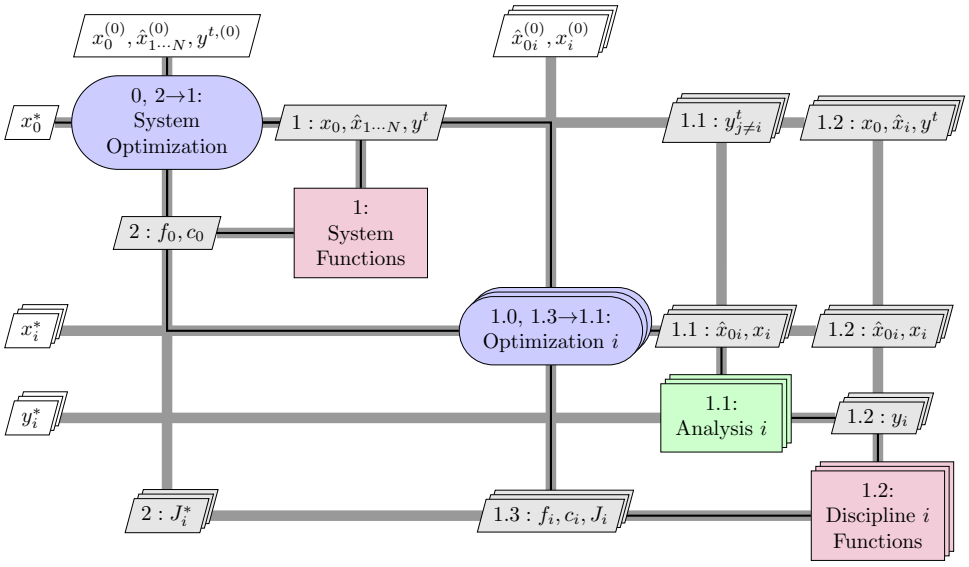

Fig. 11 Collaborative optimization (CO) architecture.

models to capture the influence of the shared design variables on the optimality of the disciplinary subproblems. The system subproblem for BLISS-2000 is given by

$$\begin{aligned} \text{min.} \quad & f_0\left(\mathbf{x}_0, \tilde{\mathbf{y}}\left(\mathbf{x}, \mathbf{y}^t\right)\right) \\ \text{w.r.t.} \quad & \mathbf{x}_0, \mathbf{y}^t \\ \text{s.t.} \quad & \mathbf{c}_0\left(\mathbf{x}, \tilde{\mathbf{y}}\left(\mathbf{x}, \mathbf{y}^t\right)\right) \geq \mathbf{0} \\ & \mathbf{c}^c = \mathbf{y}_i^t - \tilde{\mathbf{y}}_i\left(\mathbf{x}_0, \mathbf{x}_i, \mathbf{y}_{j\neq i}^t\right) = \mathbf{0} \quad i = 1, \ldots, N, \end{aligned} \tag{5}$$

where $\tilde{\mathbf{y}}_i$ represents the state variable estimates computed by the disciplinary surrogate models. The discipline subproblem for discipline i is given by

$$\begin{aligned} \text{min.} \quad & f_i = \mathbf{w}_i^T \mathbf{y}_i\left(\mathbf{x}_0, \mathbf{x}_i, \mathbf{y}_{j\neq i}^t\right) \\ \text{w.r.t.} \quad & \mathbf{x}_i \\ \text{s.t.} \quad & \mathbf{c}_i\left(\mathbf{x}_0, \mathbf{x}_i, \mathbf{y}_i\left(\mathbf{x}_0, \mathbf{x}_i, \mathbf{y}_{j\neq i}^t\right)\right) \geq \mathbf{0}, \end{aligned} \tag{6}$$

where each $\mathbf{w_i}$ is a vector of user-defined weighting coefficients. The BLISS-2000 architecture introduces the added complexity of forming and updating a surrogate model of each discipline's optimal output. These models must be included in the XDSM as additional components, and the timing of the updates must be noted in the algorithm.

The XDSM for BLISS-2000 is given in Fig. 12. The sequence of operations shown in the XDSM is as follows:

0. Pass initial data to both system and discipline optimizers.
1. Initiate discipline subproblem optimizations.
2. Evaluate disciplinary analyses.
3. Compute disciplinary objective and constraint function values.
4. Compute updated solutions to disciplinary subproblems. If subproblems have not converged, return to 2; otherwise, return optimal solutions.
5. Update metamodels of optimized disciplinary subproblems with new data.
6. Initiate system subproblem optimization.
7. Interrogate metamodels with current values of system variables.
8. Compute system objective and constraint function values.
9. Compute updated solution to system subproblem. If subproblem has not converged, return to 7; otherwise, continue.
10. Check convergence of architecture based on change in system design variables. If architecture has not converged, return to 1; otherwise, return optimal solution.

In this diagram, the "Metamodel *i*" components serve the dual function of both updating and evaluating the metamodels. We determine which function is active at a given step based on the data passed to the component. In the updating steps, the results of the disciplinary optimizations are passed to the metamodel components in addition to the system variable values, whereas only the system variable values are passed in the evaluation steps. This setup is a concrete example of different data being passed to a component at different points in the algorithm and motivates our labeling of the data nodes with the times at which they are passed to the components.

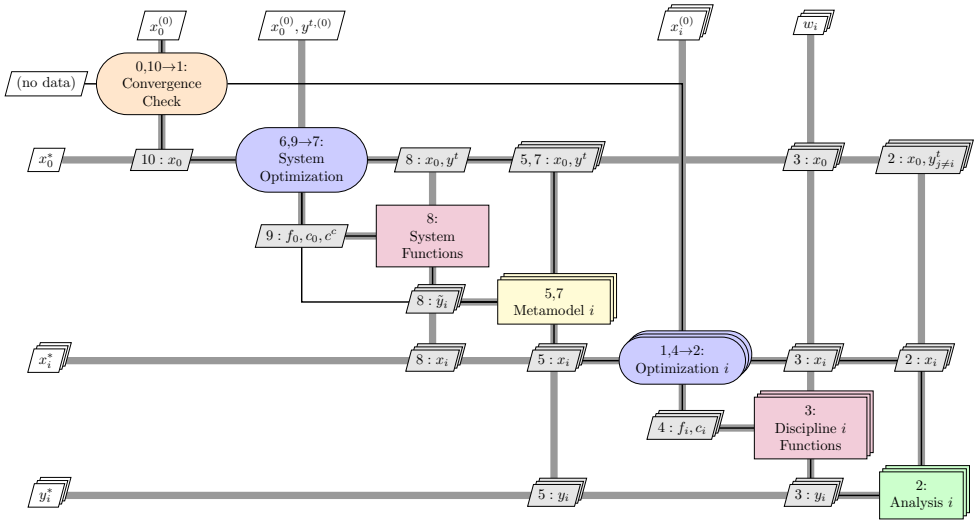

Fig. 12 BLISS-2000 architecture.

6 Applications and Future Development

Our primary motivation for the development of the XDSM was to develop a graphic representation of complex MDO processes in a single, intuitive diagram. These diagrams, together with the mathematical descriptions of the analyses and optimization problems, may then be used as a communication medium for explaining new architectures and drawing comparisons with existing ones. As demonstrated in Section 4, however, this type of diagram can also be used to describe optimization, analysis, and related processes more generally. While we have not explored this idea in more detail, we believe that there are more cases in which XDSMs could be used as visualization tools.

An important application of the XDSM is in MDO software frameworks such as iSight, ModelCenter, pyMDO (Martins et al, 2009), and OpenMDAO (Gray et al, 2010). These are tools developed to assist engineers in integrating multiple analysis codes to run various analysis and optimization processes. Because the XDSM is designed to show such processes, it could form the basis of a graphical user interface to such software. The ability to generate the XDSM for a given process could also facilitate debugging of the process by comparison with reference diagrams. However, more work is needed to expand the syntax to be able to show all of the necessary functionality.

One property of the DSM that has not been studied in detail in our work is the ability to nest diagrams. In a standard DSM, the components themselves may be systems of smaller components, and these may be systems of even smaller components, etc., forming a multilevel hierarchy. This is the basis of the "system of systems" (Sobieszczanski-Sobieski, 2008) mode of thinking. While we have not specifically accounted for this nesting in the XDSM, it is not difficult to imagine the integration of lower-level XDSMs. For example, many discipline analyses are themselves based on iterative procedures that can be modeled using an XDSM. These XDSMs can then be integrated into multidisciplinary analyses or optimizations using higher-level XDSMs. The MDO architectures containing multiple levels could also be displayed as multilevel XDSMs. However, we deliberately chose not to show them this way to demonstrate that we could capture the complete architecture in a single diagram.

Another aspect of our work that deserves more study is the grouping of elements, such as the function evaluations in the MDO architectures. We chose to display these operations as single components to focus on the wider architecture. However, breaking these components down to a more fundamental level could yield deeper insights into the calculations. For example, analysis of these calculations at the most granular level could lead to reordering the function evaluations to reduce the computational cost or developing an adjoint operator for a complex disciplinary analysis module. This type of application uses the underlying mathematical nature of the XDSM components.

Finally, by basing a visualization tool for MDO on a DSM, we introduce the possibility of using existing DSM analysis methods (Eppinger et al, 1994; Yassine and Braha, 2003) to study and improve the MDO architectures themselves. DSM partitioning methods, which reorder the tasks within a DSM to shorten the design process (see Gebala and Eppinger (1991) and Austin et al (2000)), are of particular interest. However, these partitioning methods would have to be adapted to handle the iterative procedures already present in MDO architectures in a way that does not prevent the convergence of the architecture. In the proper context, we believe that the XDSM could be used for many more interesting applications.

7 Conclusion

In this paper, we have developed a graphical representation of MDO processes that we call the extended design structure matrix, or XDSM. This diagram allows for the simultaneous display of data dependency and process flow between computational components. We have shown how these XDSMs can be used to display several well-known architectures as well as simple multidisciplinary analysis and optimization processes. Subsequent work will show many more examples of XDSMs applied to MDO architectures.

These diagrams show strong potential not only as a visual means of representing MDO processes, but also as a graphical interface to MDO software and an analysis tool for the processes themselves. Many questions remain as to where and how to apply these diagrams most effectively. While we developed these diagrams strictly for describing MDO architectures, we also discussed other potential applications related to algorithm analysis to improve computational performance. We believe that there are numerous other areas in which XDSMs may be applied, and the future work includes finding new applications.

Acknowledgements The authors would like to thank Justin Gray and Kenneth Moore of the OpenMDAO development team at NASA Glenn Research Center for their feedback on the diagram structure and notations, as well for their help in verifying the XDSMs for the various MDO architectures. Their suggestions substantially enhanced the quality of the final diagrams. Additional suggestions were provided by Evin Cramer, William Crossley, John Dannenhofer, Raphael Haftka, Robert Haimes, Michael Kokkolaras, and Sean Torrez. Finally, we thank one of the anonymous referees, whose comments helped us properly place our work in relation to some of the MDO literature cited herein.

References

Alexandrov NM, Lewis RM (2004a) Reconfigurability in MDO problem synthesis, part 1. In: Proceedings of the 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference

Alexandrov NM, Lewis RM (2004b) Reconfigurability in MDO problem synthesis, part 2. In: Proceedings of the 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference

Austin S, Baldwin A, Li B, Waskett P (2000) Analytical design planning technique (ADePT): A dependency structure matrix tool to schedule the building design process. Construction Management and Economics 18(2):173–182, DOI 10.1080/014461900370807

Bloebaum CL, Hajela P, Sobieszczanski-Sobieski J (1992) Non-hierarchic system decomposition in structural optimization. Engineering Optimization 19(3):171–186, DOI 10.1080/03052159208941227

Booch G, Rumbaugh J, Jacobson I (2005) Unified Modeling Language User Guide, 2nd edn. Addison-Wesley Professional

Braun RD, Kroo IM (1997) Development and application of the collaborative optimization architecture in a multidisciplinary design environment. In: Alexandrov N, Hussaini MY (eds) Multidisciplinary Design Optimization: State-of-the-Art, SIAM, pp 98–116

Braun RD, Gage P, Kroo IM, Sobieski IP (1996) Implementation and performance issues in collaborative optimization. In: Proceedings of the 6^{th} AIAA, NASA, and ISSMO Symposium on Multidisciplinary Analysis and Optimization, AIAA Paper 1996-4017

Browning TR (2001) Applying the design structure matrix to decomposition and integration problems: A review and new directions. IEEE Transactions on Engineering Management 48:292–306, DOI 10.1109/17.946528

Cramer EJ, Dennis JE, Frank PD, Lewis RM, Shubin GR (1994) Problem formulation for multidisciplinary optimization. SIAM Journal on Optimization 4(4):754–776, DOI 10.1137/0804044

De Wit AJ, Van Keulen F (2010) Overview of methods for multilevel and/or multidisciplinary optimization. In: Proceedings of the 51^{st} AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, Orlando, Florida

DeMiguel AV, Murray W (2000) An analysis of collaborative optimization methods. In: Proceedings of the 8th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Long Beach, CA, aIAA 2000-4720

Eppinger SD, Whitney DE, Smith RP, Gebala DA (1994) A model-based method for organizing tasks in product development. Research in Engineering Design 6:1–13, DOI 10.1007/BF01588087

Etman LFP, Kokkolaras M, Hoftkamp AT, Papalambros PY, Rooda JE (2005) Coordination specification in distributed optimal design of multilevel systems using the χ language. Structural and Multidisciplinary Optimization 29:198–212, DOI 10.1007/s00158-004-0467-z

Gebala DA, Eppinger SD (1991) Methods for analyzing design procedures. In: ASME Conference on Design Theory and Methodology, Miami, FL, vol 31, pp 227–233

Gray J, Moore KT, Naylor BA (2010) OpenMDAO: An open-source framework for multidisciplinary analysis and optimization. In: Proceedings of the 13^{th} AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, Fort Worth, TX

Kim HM (2001) Target cascading in optimal system design. PhD thesis, University of Michigan

Kim HM, Michelena NF, Papalambros PY, Jian T (2003) Target cascading in optimal system design. Journal of Mechanical Design 125(3):474–480, DOI 10.1115/1.

1582501

Kim HM, Chen W, Wiecek MM (2006) Lagrangian coordination for enhancing the convergence of analytical target cascading. AIAA Journal 44(10):2197–2207, DOI 10.2514/1.15326

Kodiyalam S, Sobieszczanski-Sobieski J (2000) Bilevel integrated system synthesis with response surfaces. AIAA Journal 38(8):1479–1485, DOI 10.2514/2.1126

Kroo IM (1997) MDO for large-scale design. In: Alexandrov N, Hussaini MY (eds) Multidisciplinary Design Optimization: State-of-the-Art, SIAM, pp 22–44

Lano RJ (1977) The N^2 chart. Tech. rep., TRW

Martins JRRA, Marriage C, Tedford N (2009) pyMDO: An object-oriented framework for multidisciplinary design optimization. ACM Transactions on Mathematical Software 36(4):2–25, DOI 10.1145/1555386.1555389

Perez RE, Liu HHT, Behdinan K (2004) Evaluation of multidisciplinary optimization approaches for aircraft conceptual design. In: Proceedings of the 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, Albany, NY, aIAA 2004–4537

Roth B (2008) Aircraft family design using enhanced collaborative optimization. PhD thesis, Stanford University

Sellar RS, Batill SM, Renaud JE (1996) Response surface based, concurrent subspace optimization for multidisciplinary system design. In: Proceedings of the 34^{th} AIAA Aerospace Sciences and Meeting Exhibit

Sobieszczanski-Sobieski J (2008) Integrated system-of-systems synthesis. AIAA Journal 46(5):1072–1080, DOI 10.2514/1.27953

Sobieszczanski-Sobieski J, Agte JS, Sandusky RR (2000) Bi-level integrated system synthesis. AIAA Journal 38(1):164–172, DOI 10.2514/2.937

Sobieszczanski-Sobieski J, Altus TD, Phillips M, Sandusky RR (2003) Bi-level integrated system synthesis for concurrent and distributed processing. AIAA Journal 41(10):1996–2003, DOI 10.2514/2.1889

Steward DV (1981) The design structure matrix: A method for managing the design of complex systems. IEEE Transactions on Engineering Management 28:71–74

Tosserams S, Etman LFP, Papalambros PY, Rooda JE (2006) An augmented Lagrangian relaxation for analytical target cascading using the alternating direction method of multipliers. Structural and Multidisciplinary Optimization 31(3):176–189, DOI 10.1007/s00158-005-0579-0

Tosserams S, Hoftkamp AT, Etman LFP, Rooda JE (2010) A specification language for problem partitioning in decomposition-based design optimization. Structural and Multidisciplinary Optimization 42:707–723, DOI 10.1007/s00158-010-0512-z

Tufte ER (1983) The Visual Display of Quantitative Information. Graphics Press, Cheshire, Connecticut

Wujek BA, Renaud JE, Batill SM, Brockman JB (1996) Concurrent subspace optimization using design variable sharing in a distributed computing environment. Concurrent Engineering 4:361–377, DOI 10.1177/1063293X9600400405

Yassine A, Braha D (2003) Complex concurrent engineering and the design structure matrix method. Concurrent Engineering 11(3):165–176, DOI 10.1177/106329303034503