An MDO advisory system supported by knowledge-based technologies

Maurice F. M. Hoogreef[*] and Gianfranco La Rocca[†]
Delft University of Technology, Delft, The Netherlands

Multidisciplinary Design Optimization (MDO) can aid designers to improve already mature design solutions, as well as to explore innovative, complex engineering products. It is a methodology, complete with mathematical formulations, that assists in the optimization of a complex product, whilst considering and exploiting discipline interactions. Although the first MDO applications have been developed decades ago, this discipline is not yet fully exploited within industry. One of the main reasons is the lack of understanding due to the inherent complexity of the discipline itself and the lack of awareness of existing MDO technologies, software and implementation strategies. This paper introduces a potential measure to lower the accessibility level of MDO: an MDO advisory system supported by knowledge-based technologies. This MDO advisory system will enable the user to first specify an MDO problem and it will return a ranked list of suitable MDO architectures, based on the characteristics of the specified problem, to the user. Additionally, the advisory system will support the user during the implementation of the suggested optimization approach by providing (links to) specific documentation and, most of all, take care of some of the software intensive operations required to integrate the selected architecture in a commercial MDO framework. This paper provides an overall discussion of the envisioned advisory system and focuses on the knowledge-based technologies, and the implications of their implementation. These technologies make up the backbone of the envisioned advisory system, including a domain-specific ontology for MDO and a reasoning engine to provide the required reasoning capabilities for advice. Preliminary results include an ontology to enable the use of monolithic and distributed MDO architectures/problems and an extension of the reasoning functionalities of an open-source reasoner. Finally, a combination of a rule engine and query mechanism is proposed to support the use of rules on top of the ontology.

I. Introduction

Although MDO has been around for several decades, the implementation of MDO within industry still is limited, today. In 2002, Belie[1] highlighted the non-technical barriers to MDO in the aerospace industry. The first barrier is the inherent complexity of the MDO discipline, not just in terms of its mathematical formulation, but also in terms of the required organization, people, data and data management. Other practical barriers addressed by Belie[1] concern the working culture of engineers and the confidence in using MDO and trusting its results. In 2006 a European-U.S. Multidisciplinary Optimization Colloquium was held in Göttingen, Germany, attended by nearly seventy professionals from academia, industry and government. The event was summarized in a paper by Agte et al.,[2] which highlights the MDO shortcomings identified during the event and the potential directions for future research that were identified by the MDO experts. Agte et al.[2] identified that the acceptance of MDO in industry is hampered by barriers that are present at technical, organizational, cultural and educational level. Agte et al.[2] specifically mention the need for engineers who understand the concepts and methods of MDO.

One way of overcoming the educational barrier in industry is, of course, education in MDO. However, re-education of a significant number of engineers that have to apply MDO can be difficult to enforce. Another solution is to provide assistance to these engineers in the application of MDO. Assistance can, for example, be provided by handbooks, or, more effectively, by some sort of dedicated MDO advisory systems.

[*]PhD Candidate, Faculty of Aerospace Engineering, Kluyverweg 1, 2629 HS Delft, The Netherlands, AIAA student member
[†]Assistant-Professor, Faculty of Aerospace Engineering, Kluyverweg 1, 2629 HS Delft, The Netherlands

Nowadays, most of the applications of optimization is supported by dedicated software tools, also referred to as MDO frameworks. These MDO frameworks are software integration systems where a user can compose an optimization problem, by coupling different tools in a certain way and specifying the discipline interactions, as well as the optimization algorithm and its options. The framework acts as a manager between the disciplines, the optimization problem and the mathematical computations performed by the algorithm to optimize the solution to a problem. However, it leaves the burden of defining the most convenient MDO implementation to the expert user and does not provide any advisory functionality. Ideally, an advisory system could be integrated in such software tools.

In their discussion of MDO framework requirements, Kodiyalam and Sobieszczanski-Sobieski[3] were the first to present the need for an optimization advisor:

> *"...and most importantly, an optimization advisor that can appropriately recommend the optimization algorithm or a combination of algorithms (hybrid optimization plan) to be used for solution of the user problem."*[3]

In addition, they state that the MDO framework should:

> *"Provide support to easy description and set up of MDO problems using formal, decomposition based MDO methods such as Global Sensitivity Equations (GSE) based optimization, CO [Collaborative Optimization], and BLISS [Bilevel Integrated System Synthesis]."*[3]

Indeed, advisory functionalities are rarely present in modern day MDO frameworks and when present, the advice is limited to suggestions on the type of algorithm (e.g. gradient-based or evolutionary) that should be used.[3–7] However, the algorithm does not define the way disciplines are coupled in an optimization problem, which variables are exchanged between disciplines or which constraints are applied to a discipline. This type of information is defined by the MDO architecture. Currently, no advisory system exists that assists with the selection and implementation of MDO architectures.

Many MDO architectures exist in literature, which are well summarized by Martins and Lambe.[8] Also, there are many examples in literature that discuss the way to formulate a given problem, such that a specific MDO architecture can be used.[9–19] However, in industry, there is often not the flexibility to change the definition of a given problem. Hence, an architecture suited to the problem at hand should be selected and not vice versa. In such cases, advice on the suitability of architectures based on the definition of the problem at hand can be utmost beneficial. Moreover, an advisory system for architectures could guide the user through the implementation of a selected architecture, based on its formal definition. The integration and set-up of the architecture inside an optimization workflow can be a cumbersome process that requires a lot of software intensive operations and manual programming.

In addition to aiding in the selection of an appropriate algorithm, some examples in literature can be found that determine when to switch between algorithms. For example, in an article by Arora and Baenziger from 1986,[20] a method using heuristics is described to determine when to switch when a combination of algorithms is selected to solve an optimization problem. Carchrae and Beck[21] describe a form of machine learning to determine, according to the behavior of an optimization problem, when to switch to another algorithm. Balachandran and Gero[22] present a method to select an algorithm suitable for the optimization problem. The selection criteria in this application are represented by *"if-then-rules"*, as is illustrated by an example:[22]

If	all the variables are of continuous type
and	all the constraints are linear
and	the objective function is linear
then	conclude that the model is linear programming
and	execute linear programming algorithm

Lee and Kim[23] present a knowledge-based expert system as a pre- and post-processor for engineering optimization. The paper describes more advanced requirements and features of an expert system than what has actually been implemented. The implemented pre-processor uses past experience from a knowledge base to select suitable input data, such as the bounds on design variables or settings for the implemented genetic algorithm. The post-processor aids in the selection of a single, final design point out of a set of Pareto optimal solutions based on metrics derived from experience that has been stored in a knowledge base. The limitations are also illustrated by the authors: *"The knowledge on controlling the optimization process as a step of reasoning is a typical architecture that needs to be investigated."*[23]

Based on the need from industry to select an MDO architecture for a given problem formulation and the lack of understanding of the available MDO architectures, we propose an innovative MDO advisory system. To our knowledge, there is no system that is able to assist the user in the selection of a suitable MDO architecture for the problem at hand, in the appropriate definition of the optimization problem and in the integration of the MDO architecture in an executable workflow. The proposed MDO advisory system is presented in Section II.

When considering the need for an advisory system for optimization algorithms and MDO architectures, it is important to note that Wolpert and Macready developed the so-called No Free Lunch Theorem for optimization.[25,26] This theorem states that there is no single way of solving an optimization problem that performs best on all optimization problems. An optimization algorithm that performs better than average for a certain class of problems, will perform worse than average on another class of problems. Indeed, there are classes of problems that are intrinsically harder to solve with certain algorithms.[27] Hence, an optimization advisory system that can determine the most suitable algorithm and MDO architectures, for a problem at hand, can provide a significant advantage to the user, both in terms of optimization time and optimization results.

II. Overview of the MDO advisory system

The intrinsic complexity of MDO problems and the lack of understanding of MDO yield that the MDO discipline is not yet fully exploited. We propose an advisory system to support non-MDO-experts, both in industry and in academia, in the application of MDO, thereby lowering the accessibility level of MDO. The proposed MDO advisory system should have three main functionalities:

1. Advise.

   It should allow users, i.e. engineers who are not necessarily experts in MDO, to describe the problem at hand, with additional selection criteria (e.g. speed, accuracy, number and type of design variables, number and types of constraints, inputs and outputs of the disciplines, required feasibility at every iteration, involved software tools) and receive a ranked list of suitable MDO architectures in return (e.g. 1. IDF; 2. MDF; 3. SAND). Selection will be done by reasoning on the knowledge contained in the knowledge base, based on the problem description and the selection criteria.

2. Formalize.

   The advisor should interactively assist the user with the formalization of the selected architecture. This can be done by means of a template that is derived from the formal definition of the MDO architectures in the knowledge base.

3. Integrate.

   The advisory system will translate the formalized MDO architecture into an executable workflow, that can be automatically implemented inside an MDO framework.

In order to provide the user with the necessary information related to a certain MDO architecture, the advisor will also provide (links to) scientific documentation. Moreover, based on the actual execution and selections from the user, the knowledge base can also be automatically enriched through machine learning. The MDO advisory system is graphically illustrated in Fig. 1.

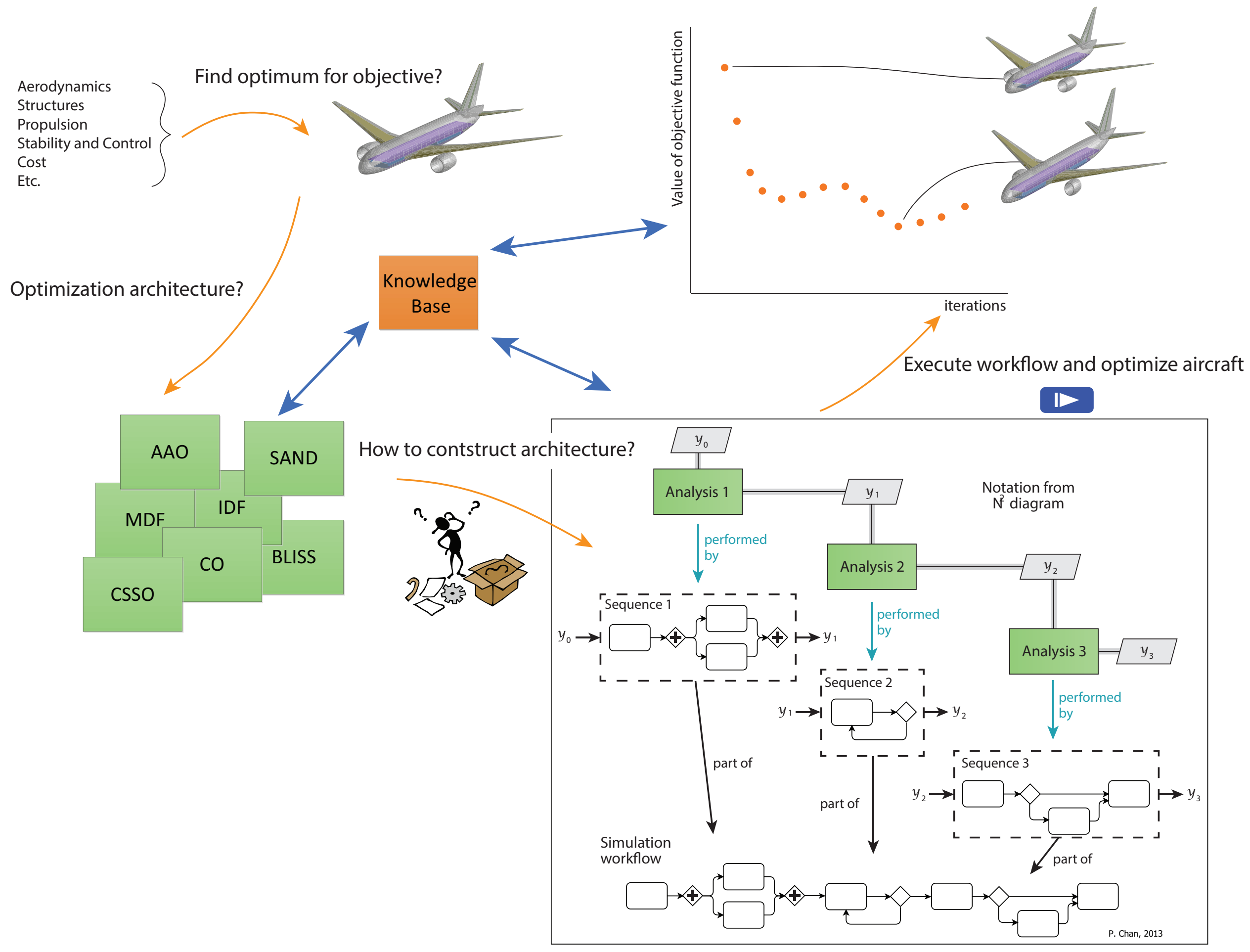


Figure 1. Graphical representation of MDO advisory system.
Top left illustrates an MDO problem. The central knowledge base (KB) (structured by ontologies) helps with making the decision about what architecture/algorithm to use. The advisory system can help to construct architectures based on the knowledge in the KB. The architecture that is constructed is then automatically formalized in a workflow in an MDO framework. This workflow should be executable by the click of a button. Results in terms of performance of the MDO advisor and execution of the workflow are communicated back to the knowledge base.

For a machine to apply expert knowledge, a knowledge base is required, to organize, capture, formalize and store the disperse knowledge on MDO and MDO architectures. This knowledge base should also contain rules on the applicability of certain architectures and optimization algorithms to problems of a specific nature. Besides classical textbooks on optimization, such as those by Vanderplaats,[28] Papalambros and Wilde[29] or Keane and Nair,[30] several scientific papers are available to extract the necessary knowledge for the knowledge of the advisory system, such as the paper by Martins and Lambe[8] on the classification of MDO architectures, the papers by Perez, Liu and Behdinan,[31] Tedford and Martins,[32] Yi, Shin and Park[33] and Kodiyalam and Sobieszczanski-Sobieski[3] on benchmarking MDO architectures.

The main components of a knowledge-based expert system are a domain-specific ontology and a reasoning engine, which form the backbone of the proposed MDO advisory system. The knowledge base of the MDO advisory system must be structured in such a way that a machine can reason on the stored knowledge and new knowledge can be easily added. Therefore, these two components will need to be developed before the actual MDO advisory system can be implemented and tested. Structuring of the knowledge base can be done through a dedicated ontology. The initial development of such a domain-specific ontology will be discussed in Section III. The reasoning engine is required to make choices according to the inputs provided by the user for the problem at hand and the knowledge stored in the knowledge base. Details on the implications of using ontologies and the reasoning engine will be discussed in Section IV.

III. Domain-specific ontology for MDO

An ontology is a formal representation of domain knowledge, based on a set of concepts. Ontologies provide a formal, shared vocabulary, that can be used to model types of objects or concepts, their properties and the relationships between them. Using an ontology, knowledge within a certain domain (e.g. diseases and medication, or aircraft parts) can be modeled in a human-readable format that is also suited for machine reasoning. This way, ontologies can structure the knowledge in a knowledge base and the definition of the concepts and their relations can be used to reason on data that is contained in the knowledge base. For example, an ontology that describes types of aircraft can structure a knowledge base of aircraft types. The aircraft (e.g. a Boeing 737 and an Airbus A320 as conventional, single-aisle aircraft) that are added to this knowledge base form the data. These aircraft are instantiations (Boeing 737 and Airbus A320) of the classes (conventional and single-aisle) defined in the ontology. These instantiations are called individuals. Individuals are related via properties to instances of other classes. These properties are defined as the relations between these classes in the ontology. E.g. a conventional aircraft (class) has a number of engines (property) of the type jet-engine (class) or of the type propeller engine (class).

Formal MDO or optimization ontologies do not exist. However, the conceptualization of the MDO domain by Sobieszczanski-Sobieski,[34] the categories derived by Giesing and Barthelemy,[35] the specification language for MDO by Tosserams et al.[36] or the classification by Martins and Lambe[8] can provide the basis for an MDO ontology. The latter, in particular, forms an excellent basis for modeling MDO architectures. The extensive survey of MDO architectures by Martins and Lambe[8] lists several common architectures, all formalized using a unified description. It also includes a classification based on the problem formulation and decomposition strategy used, and discusses the strengths and weaknesses of the architectures. The focus is primarily on methods that solve MDO problems with a single objective function and continuous design variables. An optimization ontology was previously developed in the iProd project,[37] with contributions from the authors of this paper. The iProd optimization ontology can be adapted and used for the knowledge base of the optimization advisory system proposed here.

For the proposed MDO advisory system, the Web-Ontology Language version 2.0, known as OWL 2.0,[38] is used to model the ontology. This W3C standard[38] can be used to represent knowledge with machine understandable semantics. In OWL 2.0, the properties that relate classes are called object properties. Other properties are: datatype properties, which represent, for instance, integer values or boolean values, and annotation properties, which can, for instance, be used to comment on classes or to describe units.

An OWL ontology was chosen to define the semantic structure of the knowledge base, because it allows modelling logic that could not be expressed using simple XML or other conventional database schemas. This logic allows for reasoning, which will be discussed in Section IV. Moreover, ontologies are flexible and the possibility to use reasoners removes the need for complex queries to retrieve knowledge from the knowledge base.

The preliminary version of this MDO ontology is based on the four monolithic MDO architectures described by Martins and Lambe,[8] namely: "All at once" (AAO), "Simultaneous Analysis and Design" (SAND), "Individual Disciplinary Feasible" (IDF) and "Multidisciplinary Feasible" (MDF), and three distributed MDO architectures, namely: "Collaborative Optimization" (CO), "Bi-Level Integrated Systems Synthesis" (BLISS) and "Concurrent Subspace Optimization" (CSSO). To model these architectures, several classes for the problem components are modelled and relations between these classes (object properties) are added to create the necessary restrictions. An overview of the classes for monolithic architectures in the ontology is presented in Fig. 2.

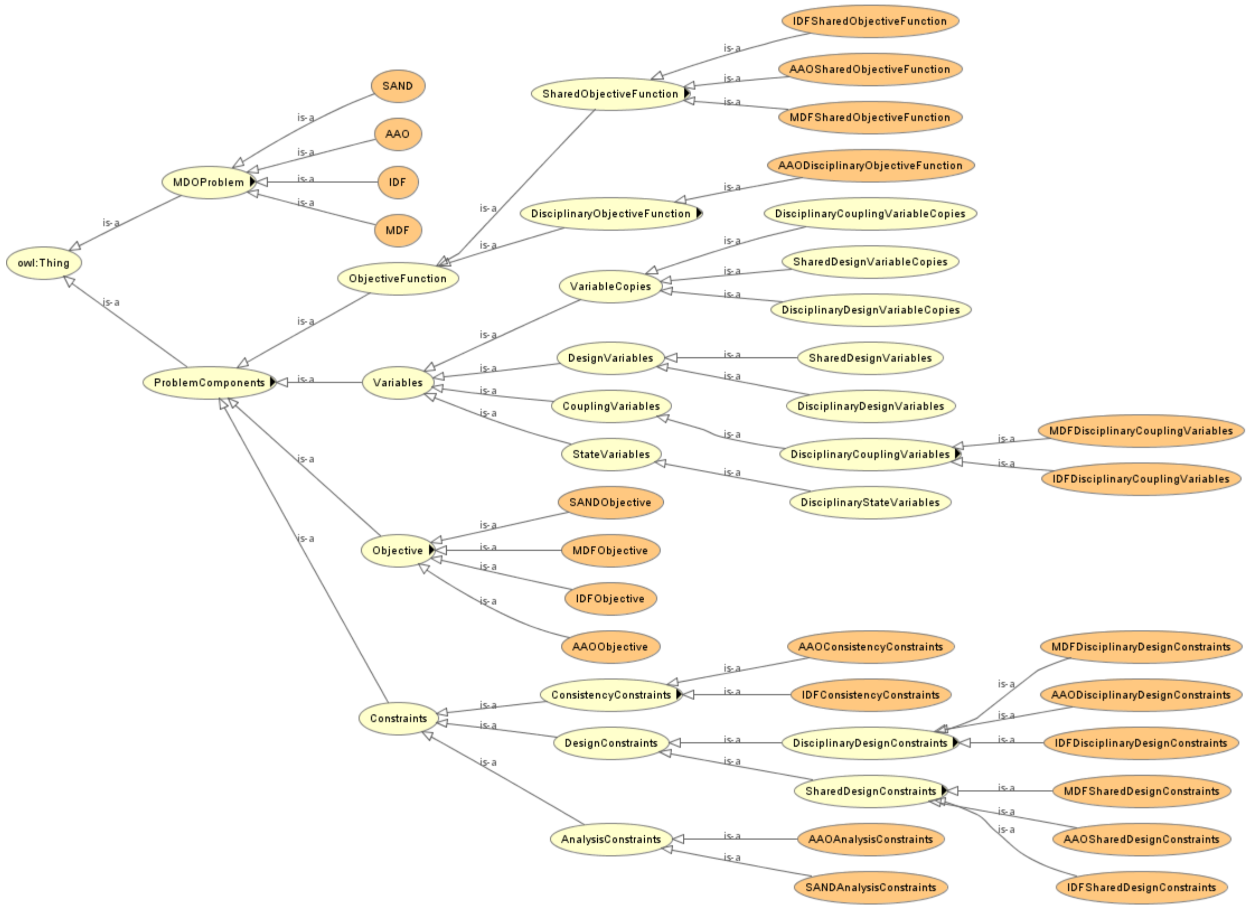

owl:Thing
is-a
MDOProblem
SAND
AAO
IDF
MDF
ProblemComponents
ObjectiveFunction
SharedObjectiveFunction
IDFSharedObjectiveFunction
AAOSharedObjectiveFunction
MDFSharedObjectiveFunction
DisciplinaryObjectiveFunction
AAODisciplinaryObjectiveFunction
Variables
VariableCopies
DisciplinaryCouplingVariableCopies
SharedDesignVariableCopies
DisciplinaryDesignVariableCopies
DesignVariables
SharedDesignVariables
DisciplinaryDesignVariables
CouplingVariables
DisciplinaryCouplingVariables
MDFDisciplinaryCouplingVariables
IDFDisciplinaryCouplingVariables
StateVariables
DisciplinaryStateVariables
Objective
SANDObjective
MDFObjective
IDFObjective
AAOObjective
Constraints
ConsistencyConstraints
AAOConsistencyConstraints
IDFConsistencyConstraints
DesignConstraints
DisciplinaryDesignConstraints
MDFDisciplinaryDesignConstraints
AAODisciplinaryDesignConstraints
IDFDisciplinaryDesignConstraints
SharedDesignConstraints
MDFSharedDesignConstraints
AAOSharedDesignConstraints
IDFSharedDesignConstraints
AnalysisConstraints
AAOAnalysisConstraints
SANDAnalysisConstraints

Figure 2. Draft MDO ontology for monolithic architectures.

In Fig. 2, all classes that are necessary to completely model the four monolithic architectures are shown. It can be observed that not all classes are detailed to subclasses representative for a specific architecture. The reason is that this is only required in case a more specific description is required. In the case the general description of a class is sufficient to represent the concept within a certain architecture, then this more specific description is not necessary and the concept is shared between different architectures. For example, MDFDisciplinaryCouplingVariables class represents the disciplinary coupling variables in MDF. The coupling variables of one discipline are a function of the design variables and coupling variables of other disciplines, whereas in the AAO and SAND architecture the disciplinary coupling variables are "simple variables" and not a function of any other variable. Hence, a more specific class is required for MDF, whereas the existing class is sufficient for AAO and SAND. Similarly, for IDF, disciplinary coupling variables are a function of design variables and variable copies (also called target-variables), thus another subclass is required to model the IDF disciplinary coupling variables. The four monolithic architectures included in the current ontology are also modelled as separate classes. The object properties that are connected to the individual architectures make up the actual definition of the architecture. To illustrate this, the definition of the MDF architecture according to Martins and Lambe[8] is used as an example.
The optimization problem is defined by Martins and Lambe[8] as:

$$\begin{aligned} \text{minimize} \quad & f_0\left(\mathbf{x}, \mathbf{y}\left(\mathbf{x}, \mathbf{y}\right)\right) \\ \text{with respect to} \quad & \mathbf{x} \\ \text{subject to} \quad & \mathbf{c}_0\left(\mathbf{x}, \mathbf{y}\left(\mathbf{x}, \mathbf{y}\right)\right) \geq 0 \\ & \boldsymbol{c}_i\left(\mathbf{x}_0, \mathbf{x}_i, \mathbf{y}_i\left(\mathbf{x}_0, \mathbf{x}_i, \mathbf{y}_{j\neq i}\right)\right) \geq 0 \text{ for } i = 1, ..., N \end{aligned} \tag{1}$$

where:	$\mathbf{x}_0$	shared design variables
	$\mathbf{x}_i$	disciplinary design variables
	$\mathbf{y}_i$	disciplinary coupling variables
	$\mathbf{c}_0$	shared design constraints
	$\mathbf{c}_i$	disciplinary design constraints

The MDF architecture from Martins and Lambe[8] is presented in Fig. 3 for three disciplines and a Gauss-Seidel multidisciplinary analysis.

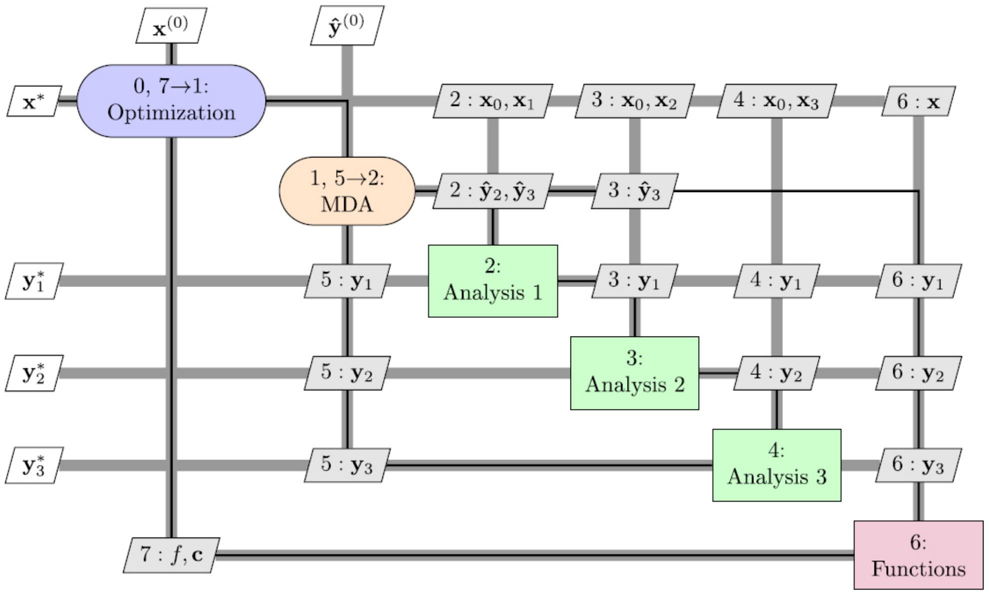


Figure 3. Representation of the MDF architecture with a Gauss-Seidel multidisciplinary analysis by Martins and Lambe.[8]

The MDF architecture is formalized in the MDO ontology according to the following translation:

- `optimizeWithRespectToVariable` - The problem should be optimized with respect to these variables.
- `isFunctionOfVariable` - Something is a function of this variable.
- `hasOptimizationObjective` - The problem has this objective statement, which can be a combination of shared and disciplinary objective functions.
- `hasSharedObjectiveFunction` - The objective statement contains this function definition of the shared objective (only 1 allowed).
- `hasDisciplinaryObjectiveFunction` (Not applicable for the MDF architecture) - The objective statement contains these function definitions of the disciplinary objectives.
- `hasInequalityConstraint` - The problem has these inequality constraints.
- `hasEqualityConstraint` (Not applicable for the MDF architecture) - The problem has these equality constraints.

A visualization of the ontology classes is presented in Fig. 4 as a graph-diagram, where the MDF architecture is presented with its object properties. This figure provides a more detailed view of the MDF class from Fig. 2, where only a class - subclass representation was presented and no relations between classes were shown.

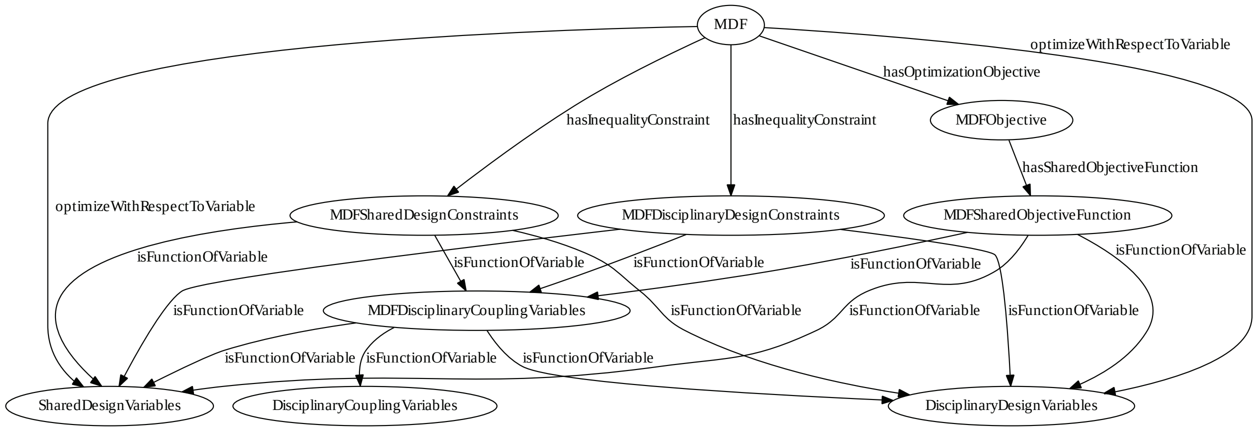


Figure 4. Description of the MDF architecture in the MDO ontology.

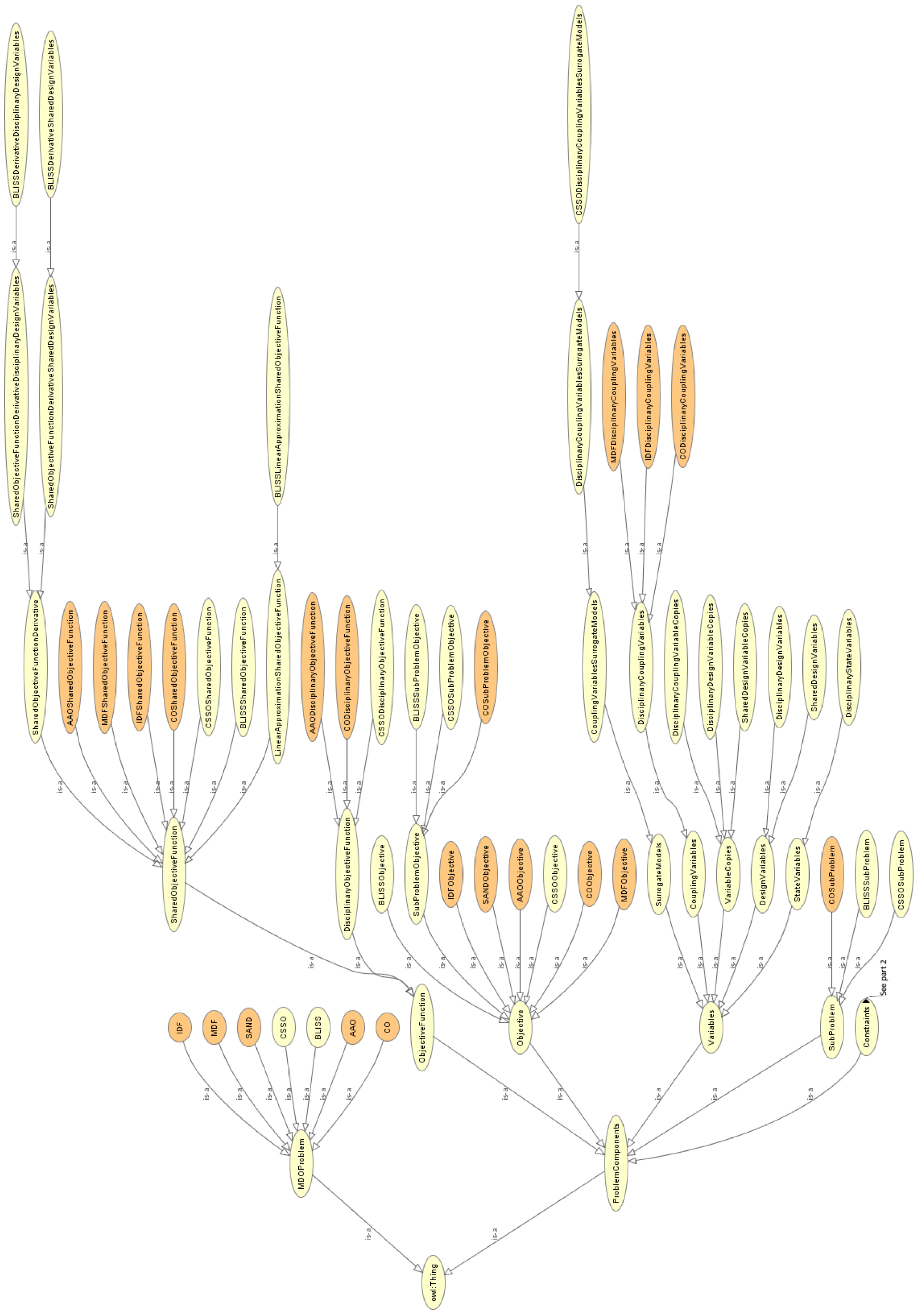
Distributed architectures can be defined in a similar manner as was demonstrated for monolithic architectures, using MDF. However, the ontology that was used for single level architectures is not completely suited to model distributed architectures. Therefore, some additional classes and relations have to be added. The resulting ontology that can model both the monolithic and the three distributed architectures (CO, BLISS and CSSO) is shown in Fig. 5 and Fig. 6. These pictures illustrates the additional classes that had to be added to the ontology for monolithic architectures to cover CO, BLISS and CSSO. These classes (`SubProblem` and `SubProlemObjective` for all multilevel and some specific classes for the different multilevel problem statements) are necessary to account for the fact that the optimization problem is now decomposed into several levels. The relationship between the subproblem optimizations and the system level optimization, and the exchanged information between these levels, is added through additional object properties. This is illustrated below for the definition of the CO architecture according to Martins and Lambe,[8] similar to the method followed for MDF.
The optimization problem for CO is defined by Martins and Lambe,[8] for the system level subproblem, as:

$$\begin{aligned} \text{minimize } & f_0\left(\mathbf{x}_0, \hat{\mathbf{x}}_1, ..., \hat{\mathbf{x}}_N, \hat{\mathbf{y}}\right) \\ \text{with respect to } & \mathbf{x}_0, \hat{\mathbf{x}}_1, ..., \hat{\mathbf{x}}_N, \hat{\mathbf{y}} \\ \text{subject to } & \mathbf{c}_0\left(\mathbf{x}_0, \hat{\mathbf{x}}_1, ..., \hat{\mathbf{x}}_N, \hat{\mathbf{y}}\right) \geq 0 \\ & J_i^* = ||\hat{\mathbf{x}}_{0i} - \mathbf{x}_0||_2^2 + ||\hat{\mathbf{x}}_i - \mathbf{x}_i||_2^2 + \left|\left|\hat{\mathbf{y}}_i - \mathbf{y}_i\left(\hat{\mathbf{x}}_{0i}, \mathbf{x}_i, \hat{\mathbf{y}}_{j\neq i}\right)\right|\right|_2^2 = 0 \text{ for } i = 1, ..., N \end{aligned} \tag{2}$$

And for the discipline subproblems as:

$$\begin{aligned} \text{minimize } & J_i\left(\hat{\mathbf{x}}_{0i}, \mathbf{x}_i, \mathbf{y}_i\left(\hat{\mathbf{x}}_{0i}, \mathbf{x}_i, \hat{\mathbf{y}}_{j\neq i}\right)\right) \\ \text{with respect to } & \hat{\mathbf{x}}_{0i}, \mathbf{x}_i \\ \text{subject to } & \mathbf{c}_i\left(\hat{\mathbf{x}}_{0i}, \mathbf{x}_i, \mathbf{y}_i\left(\hat{\mathbf{x}}_{0i}, \mathbf{x}_i, \hat{\mathbf{y}}_{j\neq i}\right)\right) \geq 0 \end{aligned} \tag{3}$$

where:	$\mathbf{x}_0$	shared design variables
	$\hat{\mathbf{x}}_{0i}$	shared design variable copies passed to and manipulated by discipline i
	$\mathbf{x}_i$	disciplinary design variables
	$\hat{\mathbf{x}}_i$	disciplinary design variable copies passed to the system subproblem
	$\mathbf{y}_i$	disciplinary coupling variables
	$\hat{\mathbf{y}}_i$	disciplinary coupling variable copies
	$\mathbf{c}_0$	shared design constraints
	$\mathbf{c}_i$	disciplinary design constraints
	J_i^*	interdisciplinary compatibility constraint



owl:Thing
MDOProblem
ProblemComponents
IDF
MDF
SAND
CSSO
BLISS
AAO
CO
ObjectiveFunction
Objective
Variables
SubProblem
Constraints
See part 2
SharedObjectiveFunction
DisciplinaryObjectiveFunction
BLISSObjective
SubProblemObjective
IDFObjective
SANDObjective
AAOObjective
CSSOObjective
COObjective
MDFObjective
SurrogateModels
CouplingVariables
VariableCopies
DesignVariables
StateVariables
COSubProblem
BLISSSubProblem
CSSOSubProblem
SharedObjectiveFunctionDerivative
AAOSharedObjectiveFunction
MDFSharedObjectiveFunction
IDFSharedObjectiveFunction
COSharedObjectiveFunction
CSSOSharedObjectiveFunction
BLISSSharedObjectiveFunction
LinearApproximationSharedObjectiveFunction
AAODisciplinaryObjectiveFunction
CODisciplinaryObjectiveFunction
CSSODisciplinaryObjectiveFunction
BLISSSubProblemObjective
CSSOSubProblemObjective
COSubProblemObjective
CouplingVariablesSurrogateModels
DisciplinaryCouplingVariables
DisciplinaryCouplingVariableCopies
DisciplinaryDesignVariableCopies
SharedDesignVariableCopies
DisciplinaryDesignVariables
SharedDesignVariables
DisciplinaryStateVariables
SharedObjectiveFunctionDerivativeDisciplinaryDesignVariables
SharedObjectiveFunctionDerivativeSharedDesignVariables
BLISSDerivativeDisciplinaryDesignVariables
BLISSDerivativeSharedDesignVariables
BLISSLinearApproximationSharedObjectiveFunction
DisciplinaryCouplingVariablesSurrogateModels
CSSODisciplinaryCouplingVariablesSurrogateModels
MDFDisciplinaryCouplingVariables
IDFDisciplinaryCouplingVariables
CODisciplinaryCouplingVariables
is-a

Figure 5. Draft MDO ontology for monolithic and distributed architectures - part 1.

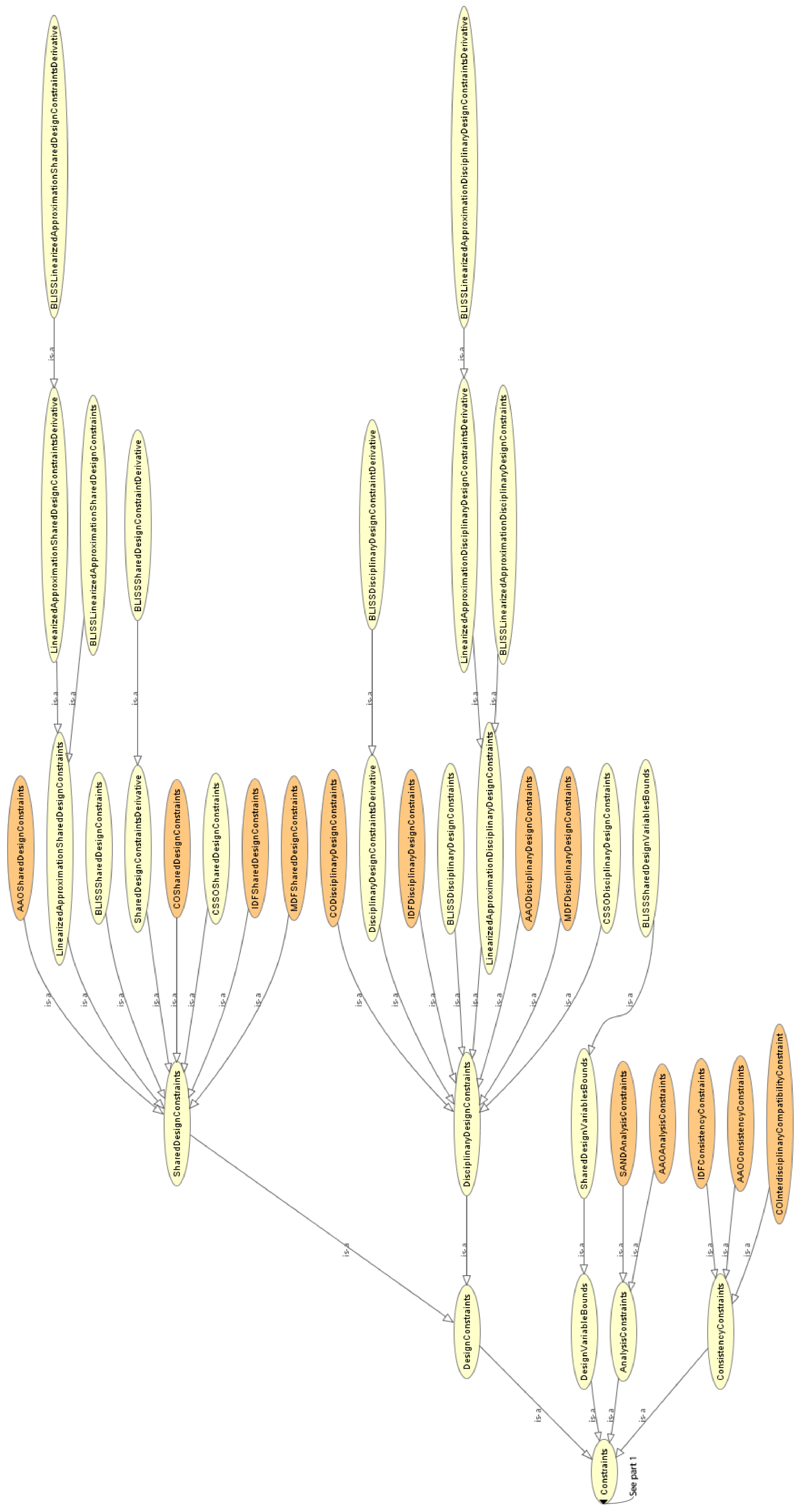

BLISSLinearizedApproximationSharedDesignConstraintsDerivative
LinearizedApproximationSharedDesignConstraintsDerivative
BLISSLinearizedApproximationSharedDesignConstraints
BLISSSharedDesignConstraintDerivative
BLISSDisciplinaryDesignConstraintDerivative
BLISSLinearizedApproximationDisciplinaryDesignConstraintsDerivative
LinearizedApproximationDisciplinaryDesignConstraintsDerivative
BLISSLinearizedApproximationDisciplinaryDesignConstraints
AAOSharedDesignConstraints
LinearizedApproximationSharedDesignConstraints
BLISSSharedDesignConstraints
SharedDesignConstraintsDerivative
COSharedDesignConstraints
CSSOSharedDesignConstraints
IDFSharedDesignConstraints
MDFSharedDesignConstraints
CODisciplinaryDesignConstraints
DisciplinaryDesignConstraintsDerivative
IDFDisciplinaryDesignConstraints
BLISSDisciplinaryDesignConstraints
LinearizedApproximationDisciplinaryDesignConstraints
AAODisciplinaryDesignConstraints
MDFDisciplinaryDesignConstraints
CSSODisciplinaryDesignConstraints
BLISSSharedDesignVariablesBounds
SharedDesignConstraints
DisciplinaryDesignConstraints
SharedDesignVariablesBounds
SANDAnalysisConstraints
AAOAnalysisConstraints
IDFConsistencyConstraints
AAOConsistencyConstraints
COInterdisciplinaryCompatibilityConstraint
DesignConstraints
DesignVariableBounds
AnalysisConstraints
ConsistencyConstraints
Constraints
See part 1
is-a

Figure 6. Draft MDO ontology for monolithic and distributed architectures - part 2.

The CO architecture according to the representation by Martins and Lambe is presented in Fig. 7.

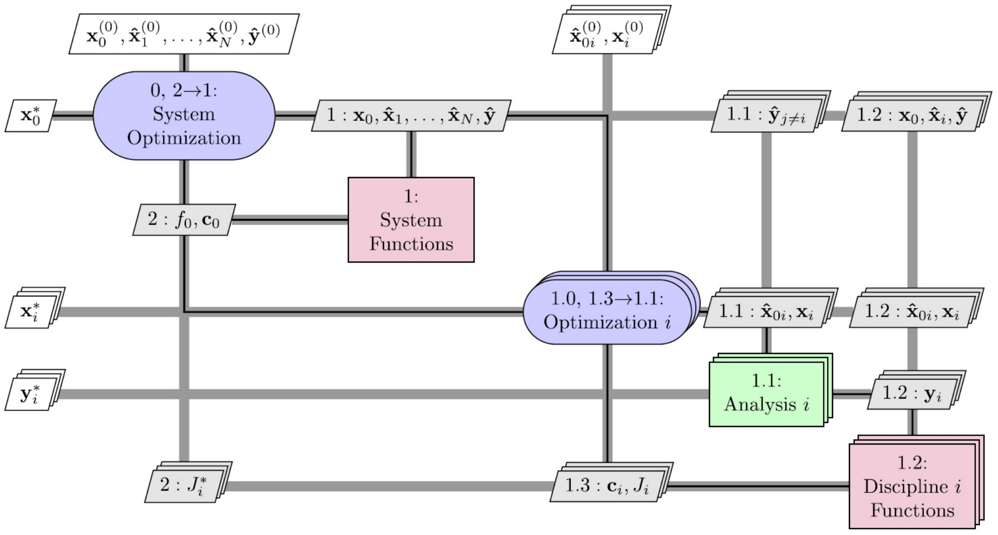

$\mathbf{x}_0^{(0)}, \hat{\mathbf{x}}_1^{(0)}, \ldots, \hat{\mathbf{x}}_N^{(0)}, \hat{\mathbf{y}}^{(0)}$
$\hat{\mathbf{x}}_{0i}^{(0)}, \mathbf{x}_i^{(0)}$
$\mathbf{x}_0^*$
0, 2→1: System Optimization
$1 : \mathbf{x}_0, \hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_N, \hat{\mathbf{y}}$
$1.1 : \hat{\mathbf{y}}_{j \neq i}$
$1.2 : \mathbf{x}_0, \hat{\mathbf{x}}_i, \hat{\mathbf{y}}$
$2 : f_0, \mathbf{c}_0$
1: System Functions
$\mathbf{x}_i^*$
1.0, 1.3→1.1: Optimization i
$1.1 : \hat{\mathbf{x}}_{0i}, \mathbf{x}_i$
$1.2 : \hat{\mathbf{x}}_{0i}, \mathbf{x}_i$
$\mathbf{y}_i^*$
1.1: Analysis i
$1.2 : \mathbf{y}_i$
$2 : J_i^*$
$1.3 : \mathbf{c}_i, J_i$
1.2: Discipline i Functions

Figure 7. Representation of the CO architecture by Martins and Lambe.[8]

The CO architecture is formalized in the MDO ontology according to the following translation:

- `optimizeWithRespectToVariable` - The problem should be optimized with respect to these variables.
- `isFunctionOfVariable` - Something is a function of this variable.
- `hasOptimizationObjective` - The problem has this objective statement, which can be a combination of shared and disciplinary objective functions.
- `hasSharedObjectiveFunction` - The objective statement contains this function definition of the shared objective (only 1 allowed).
- `hasDisciplinaryObjectiveFunction` - The (subproblem) objective statement contains these function definitions of the disciplinary objectives.
- `representsDisciplinarySubpropblemObjective` - Relates the CO interdisciplinary compatibility constraint to the objectives of the disciplinary subproblems.
- `hasInequalityConstraint` - The problem has these inequality constraints.
- `hasEqualityConstraint` - The problem has these equality constraints.
- `hasSubProblem` - The problem has this/these disciplinary subproblem(s).
- `hasSubProblemObjective` - The subproblem has this objective statement.

A visualization of the ontology classes is presented in Fig. 8 as a graph-diagram, where the CO architecture is presented with its object properties. This figure provides a more detailed view of the CO class from Fig. 5 and Fig. 6, where only a class - subclass representation was presented and no relations between classes were shown.

The ontological representations of BLISS and CSSO are less trivial. Care has to be taken to make sure that the discipline subproblems are properly linked to the system subproblem in the definition of the classes and object properties relating the classes to each other. The relation `hasSubProblem` connects the disciplinary subproblems to the system level subproblem, but does not represent exchanged information. This already shows that the ontological representation of the monolithic architectures is much more uniform than the ones of multilevel architectures, as every multilevel architecture requires additional extensions to the ontology. The choice of using ontologies to structure the knowledge base implies that certain additions must be made to the chosen, open-source, reasoner.

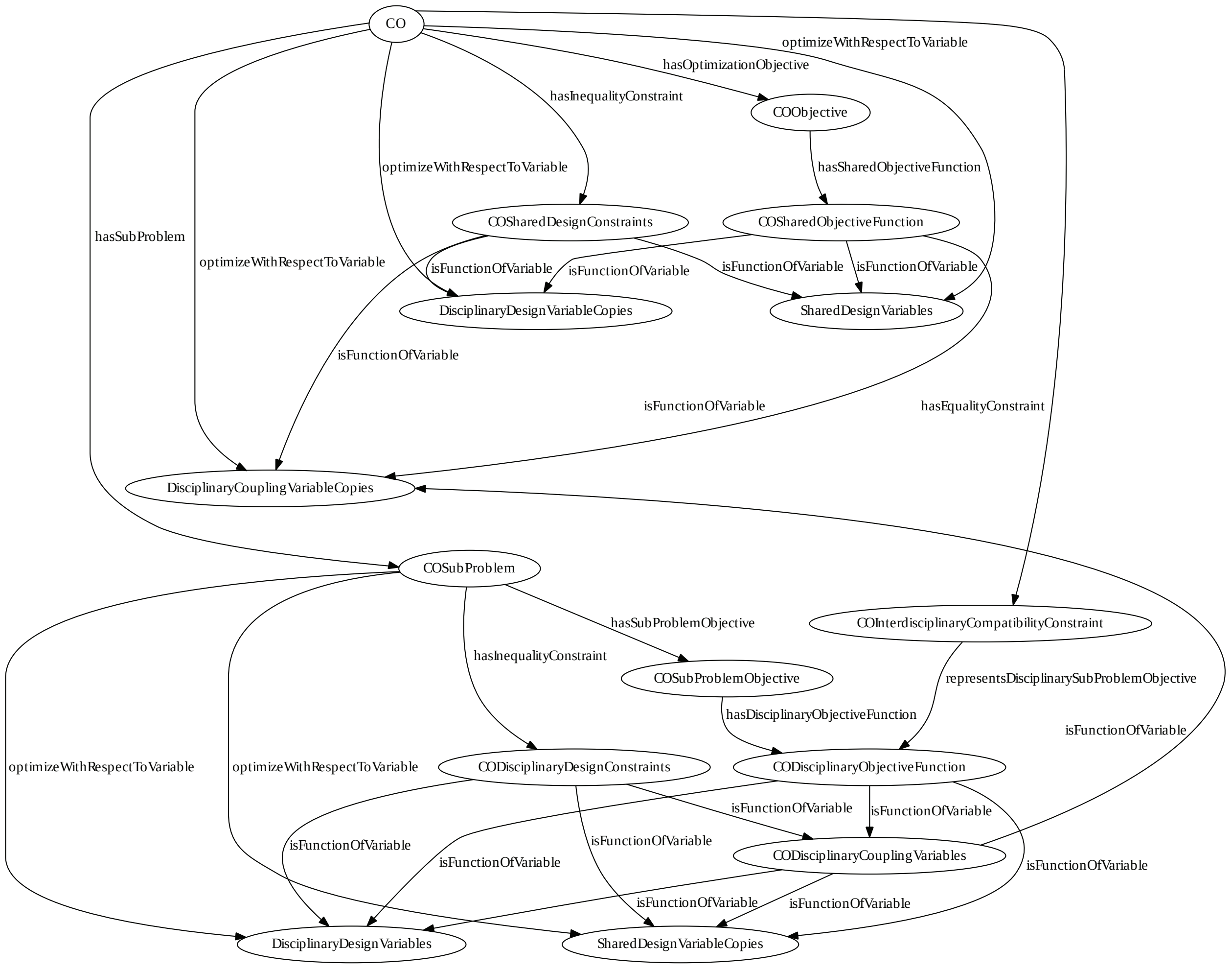

CO
optimizeWithRespectToVariable
hasOptimizationObjective
hasInequalityConstraint
COObjective
hasSharedObjectiveFunction
optimizeWithRespectToVariable
hasSubProblem
COSharedDesignConstraints
COSharedObjectiveFunction
optimizeWithRespectToVariable
isFunctionOfVariable
isFunctionOfVariable
isFunctionOfVariable
isFunctionOfVariable
isFunctionOfVariable
DisciplinaryDesignVariableCopies
SharedDesignVariables
isFunctionOfVariable
isFunctionOfVariable
hasEqualityConstraint
DisciplinaryCouplingVariableCopies
COSubProblem
hasSubProblemObjective
COInterdisciplinaryCompatibilityConstraint
hasInequalityConstraint
COSubProblemObjective
representsDisciplinarySubProblemObjective
hasDisciplinaryObjectiveFunction
isFunctionOfVariable
optimizeWithRespectToVariable
optimizeWithRespectToVariable
CODisciplinaryDesignConstraints
CODisciplinaryObjectiveFunction
isFunctionOfVariable
isFunctionOfVariable
isFunctionOfVariable
isFunctionOfVariable
isFunctionOfVariable
CODisciplinaryCouplingVariables
isFunctionOfVariable
isFunctionOfVariable
isFunctionOfVariable
DisciplinaryDesignVariables
SharedDesignVariableCopies

Figure 8. Description of the CO architecture in the MDO ontology.

IV. Reasoning Engine

The ontology can be used to structure the knowledge base and when individuals (instantiations of the classes) are added to this knowledge base, the knowledge base is populated. Because of the relations between classes that are defined in the ontology, it is possible to derive extra information about the individual. This can be done using a semantic reasoner, or reasoning engine. A reasoner is a software program that infers so-called logical consequences.

The object properties can be used to infer to which class a certain instance (individual) belongs. For example, a constraint, that has certain properties, can be inferred to belong to a certain, more specific, category of constraints (e.g. consistency constraints). Thus the reasoner could deduce what type of constraint is defined, without the user of the framework knowing which type he defined or having to define this explicitly.

The formalization of the domain knowledge, inside the ontology, and the reasoning engine allow the MDO advisor to behave as a domain expert. It could advise the user to use a type of architecture, that matches the problem description. By extending the reasoner's capabilities, it could also infer to what extent a certain problem definition inside the MDO framework matches the architecture formalizations contained in the knowledge base. Therefore, a ranked list of the most suitable architectures could be returned, in combination with suggestions for what to change to match a certain architecture.

Using reasoning, one can for example infer that an instance of one class is also an instance of another more specific class, if it satisfies the conditions of the latter class. These conditions were defined using the object and datatype properties in the ontology. For reasoning, these conditions must be made "necessary and sufficient" in the OWL ontology to allow for the inference of logical consequences. This defines that something that matches these conditions is equivalent to the more specific class. For example, the class "old aircraft," a subclass of aircraft, can be defined as being equivalent to an aircraft that is at least 20 years old. The class "old aircraft" has the "necessary and sufficient conditions" that something must be an aircraft and at least 20 years old, to be an old aircraft.

Next to the "necessary and sufficient conditions," "necessary conditions" exist as well. However, "necessary conditions" mean that an instance of a class must have the properties that are defined for this class. Thus an old aircraft is an aircraft older than 20 years. For example, when the conditions for the old aircraft are "necessary conditions," an aircraft that is 30 years old, cannot be inferred to be an old aircraft. On the contrary, when the conditions are "necessary and sufficient," it could be an old aircraft, because it at least satisfies the condition for equivalence (necessary and sufficient). This is illustrated below:

> Consider the definition of an old aircraft as an aircraft that is at least 20 years or older and has red paint: (where $\equiv$ and $\wedge$ stand for "is equivalent to" and "and", respectively)
>
> `old aircraft` $\equiv$ `aircraft` $\wedge$ `hasAge` $\geq$ `20 years` $\wedge$ `hasColor ''red''`
>
> Now consider an aircraft ($aircraft_1$) that is 30 years old and has the color "red":
>
> $aircraft_1$ `= aircraft` $\wedge$ `hasAge = 30 years` $\wedge$ `hasColor ''red''`
>
> A human being would typically say that $aircraft_1$ is an old aircraft, since it is older than twenty years and has red paint. A reasoner would act differently depending on whether the condition `aircraft` $\wedge$ `hasAge` $\geq$ `20 years` is a "necessary condition" or a "necessary and sufficient condition":
>
> necessary : $aircraft_1$ `= aircraft` $\wedge$ `hasAge = 30 years` $\wedge$ `hasColor ''red''`
>
> necessary and sufficient : $aircraft_1$ `= old aircraft`

According to these conditions, a reasoner would deduce that an instance whose properties exactly match the "necessary and sufficient conditions" of a certain class, is a member of this class. However, this is not as simple for OWL ontologies and reasoners. OWL ontologies and OWL reasoners use the Open World Assumption (OWA).[39] This means that something that cannot be deduced from the knowledge in the knowledge base is unknown, whereas the Closed World Assumption (CWA) implies that anything that is unknown is false (not true). This difference is illustrated below, for the case of the old aircraft:

> Consider the old aircraft, with the **necessary and sufficient** condition of being an aircraft that is at least 20 years or older and does not have the color red:
>
> `old aircraft` $\equiv$ `aircraft` $\wedge$ `hasAge` $\geq$ `20 years` $\wedge$ ***not*** hasColor "red"
>
> Now consider an aircraft ($aircraft_2$) that is 30 years old:
>
> $aircraft_2$ `= aircraft` $\wedge$ `hasAge = 30 years`
>
> A human being would typically say that $aircraft_2$ is an old aircraft, since it is older than twenty years, but we do not know whether it is red or not. There is a difference between the OWA and CWA for the answer of a reasoner, though:
>
> CWA : $aircraft_2$ `= old aircraft`
>
> OWA : $aircraft_2$ `= ''I do not know whether this is an old aircraft''`

In the OWA, the reasoner does not know whether this is an old aircraft, since this aircraft could have the color "red". We did not specify explicitly that it is not red, so the reasoner cannot assume that it is not red. For the CWA, the reasoner would assume that it is not red. (Everything that is unknown, is false.) Therefore, because of the fact that we cannot deduce that there is no additional information about this aircraft, we do not know whether it is an old aircraft in the case of the OWA. Next to this, there is another consequence of the OWA, as is illustrated below:

> Consider a large aircraft, with the **necessary and sufficient** condition of being an aircraft that has 4 engines:
>
> `large aircraft` $\equiv$ `aircraft` $\wedge$ `hasEngines = 4 engines`
>
> Now consider an aircraft ($aircraft_3$) that has four engines:
>
> $aircraft_3$ `= aircraft` $\wedge$ `hasEngine` $engine_1$ $\wedge$ `hasEngine` $engine_2$ $\wedge$ `hasEngine` $engine_3$ $\wedge$ `hasEngine` $engine_4$
>
> A human being would typically say that this is a large aircraft, since it has four engines. Again, there is a difference between the OWA and CWA for the answer of a reasoner:
>
> CWA : $aircraft_3$ `= large aircraft`
>
> OWA : $aircraft_3$ `= ‘‘I do not know whether this is a large aircraft’’`

The OWL reasoner does not know whether the engines are different. For example, $engine_1$ could be the same individual as $engine_2$. We need to specify explicitly that these engines are different. Therefore, all individuals must be explicitly made unique. Luckily, this can be done in OWL using the “unique name assumption” which makes all individuals different.

There is, however, another problem for the reasoner with the OWA in the above example, even if we use the “unique name assumption”. We did not specify explicitly that $aircraft_3$ has no other engines, and it needs to have exactly 4 engines to be a large aircraft. There might be other engines of which the reasoner does not know that they belong, or do not belong to this aircraft. The restriction that was imposed on the large aircraft is a “cardinality restriction”, in this case the “exact cardinality”. Changing it to a “minimum cardinality restriction” would solve this issue, since any aircraft that has at least 4 unique engines is a large aircraft:

> The large aircraft, with the **necessary and sufficient** condition of being an aircraft that has at least 4 engines:
>
> `large aircraft` $\equiv$ `aircraft` $\wedge$ `hasEngines` $\geq$ `4 engines`
>
> OWA : $aircraft_3$ `= large aircraft`

Reasoners, such as the Hermit Reasoner 1.3.8[40] that was chosen in this research, are capable of deducing the logical consequences of these necessary and sufficient conditions when individuals are unique. However, certain cases may exist when we only want to deduce logical consequences for instantiations that exactly match the necessary and sufficient conditions of a given class:

> `old aircraft` $\equiv$ `aircraft` $\wedge$ `hasAge` $\geq$ `20 years` $\wedge$ `hasColor ‘‘red’’`
>
> Now consider an aircraft ($aircraft_4$) that is 30 years old and has red and white colors:
>
> $aircraft_4$ `= aircraft` $\wedge$ `hasAge = 30 years` $\wedge$ `hasColor ‘‘red’’` $\wedge$ `hasColor ‘‘white’’`
>
> A reasoner using the OWA in OWL would say that this is an old aircraft, and would return $aircraft_4$ if we asked it for all `old aircraft`.
>
> OWL reasoner : $aircraft_4$ `= old aircraft`

Additionally, if we define a white aircraft in the same ontology:
`white aircraft` $\equiv$ `aircraft` $\wedge$ `hasColor ‘‘white’’`

A reasoner using the OWA in OWL would say that $aircraft_4$ is a white aircraft.

$$\text{OWL reasoner} : aircraft_4 = \texttt{white aircraft}$$

This becomes problematic when we only want aircraft that have only red paint and are older than 20 years. To have this exact reasoning, everything that is not true about an individual must be expressed. However, this can become prohibitive when a lot of individuals and classes exist. Alternatively, queries could be used to search for information matching a pattern inside the knowledge base. Yet, the order of execution for these queries is fixed and therefore it is difficult to have query patterns that are interdependent on each other's results.

The exact reasoning is required for the MDO advisory system to determine whether a problem that has been defined completely matches with, for example, the formal definition of MDF in the MDO ontology. Without having this exact reasoning, also problems that have slightly more characteristics than MDF could be reasoned to be MDF. In the advisory system it is important to know what these "other" characteristics are, to either give a different advice to the user, or question the user on whether this characteristic could be dropped from the problem definition. Additionally, it is important in the formalization and integration steps, because the MDO advisor should assist in the implementation of the problem as it was defined by the user.

Therefore, the reasoner used in this research is extended with functionalities to determine whether an individual exactly matches the described "necessary and sufficient conditions". We assume that things that are not defined for an individual are not true, "closing the world" of the data. That is, we assume that the aircraft that is 30 year old and has red paint does not have any other paint color. The Hermit Reasoner 1.3.8[40] was chosen, because it performed well in initial tests for inferencing logical consequences in an open world and because it is open-source and can therefore easily be extended.

Because of the OWA, the reasoner is also not able to determine whether certain cardinality restrictions occurring in the properties of classes are met, as was illustrated earlier. Therefore, the reasoner is also extended with the ability to check whether cardinality restrictions are met. Here, we also assume that things that are not defined for an individual are not true when we check for minimum, exact and maximum cardinalities. I.e. we assume that $aircraft_3$ has no other engines.

Through these modifications, the reasoner is able to determine what type of architecture is being defined and the advisory system can be told to assist in the construction of this architecture, based on the formal definition in the knowledge base. However, the advisory functionality must also be supported when an architecture is not yet defined and the user seeks advice on how to structure his problem, based on higher level characteristics.

A. Rule Based Functionalities

The ranked list of suitable architectures for a specific problem can also be based on other aspects that are defined with the problem, such as the types of variables, the need for feasibility when the optimization is stopped prematurely, the availability of gradient information from external software or time constraints. One way of supporting the advisory functionality, based on higher level characteristics, is by including rule based advice. The rule based advice is implemented through the use of the Semantic Web Rule Language (SWRL), which can be modeled inside an OWL ontology. Thus, SWRL can use OWL class expressions. SWRL can be used to model rules that cannot (or not easily) be modeled through restrictions in the ontology, for example to assert something when a condition is met. An example of such a SWRL rule is shown below, to derive that if there is an aircraft a and this aircraft has passengers y and z and these passengers are different, then this means that these passengers are fellow travelers on the same aircraft. (Note that the relation `differentFrom` is included in OWL and supported by SWRL)

```
Aircraft(?a) ∧ hasPassenger(?a, ?y) ∧ hasPassenger(?a, ?z) ∧ differentFrom(?y, ?z)
→ fellowTraveller(?y, ?z)
```

However, SWRL, being modelled in OWL ontologies, is restricted by the same open world assumption. Hence, rules that try to enumerate individuals are not always possible (similarly to the examples for OWL) and negation as failure (used to derive ***not p***, i.e. that ***p*** is assumed not to hold, from failure to derive ***p***) is not supported. Only when something is explicitly stated as not being an instance of a certain class, it can be asserted that this individual is not an instance of that class. To overcome these limitations, and switch to a closed world for the data, the SWRL rules are used in combination with the SPARQL Protocol and RDF Query Language (SPARQL). SPARQL is a semantic query language for graph databases, that can retrieve and manipulate data stored in the Resource Description Framework (RDF) format. This is the same format as is used to store OWL ontologies. The world of the data is thus restricted to the information that is contained in the data (knowledge) base and SPARQL can be used to query these data.

The reason behind combining both technologies is that SWRL rules can be used in combination with the restriction in the OWL ontology to derive (assert) new facts using the semantic reasoner. SPARQL queries can be used to handle conflicts between these rules, by retrieving potentially conflicting information from the knowledge base and then defining what to do when certain conflicting information has been retrieved (launching another SPARQL query to modify/delete data in the knowledge base). The reason of not using SPARQL queries for the first step lies in the fact that the execution of SWRL rules can be explained by the reasoner and that the derived information is less sensitive to the order in which the rules are fired. In the case of SPARQL queries, there is a fixed order in which the queries are executed.

V. Conclusions and Future Work

The MDO advisory system presented in this paper has three functionalities: to advise on the most suitable MDO architecture for the problem at hand, to formalize the architecture for the problem in a proper implementation and to integrate the formalized problem definition in an MDO framework. To provide these functionalities, we make use of knowledge-based technologies; mainly a dedicated MDO ontology and a reasoning engine. Because of the implementation of the domain-specific ontology for MDO, for monolithic and distributed architectures, in OWL 2.0,[38] modifications to the chosen reasoning engine were required. These modifications are necessary to recognize user defined problems, match them to formalized architectures and provide advice based on the characteristics of the user defined problem. It is also important that the reasoning engine is able to determine which parts of a user defined problem match with the formalized architectures and which parts do not. This could not be done with the unmodified reasoning engine. The differences between user defined and formalized architectures are important as they may either trigger a different advice to the user, or question the user on whether these characteristics could be dropped from the problem definition. Moreover, the advisory system should integrate the complete problem that the user defined and not just the part that was matched to a formalized architecture.

The MDO ontology can be used to store a problem definition, according to a formal representation that can be reused in the actual implementation of the optimization problem and can easily be translated to other systems. The incorporation of selection criteria such as: speed, accuracy, feasibility at all iterations, types of variables, availability of gradients, etc. in the MDO ontology allows for selection rules to be triggered, using the SWRL and SPARQL functionalities.

Testing will still need to be performed to verify the behavior of the backbone functionalities in an integrated set-up on actual MDO problems. Additionally, a coupling to a simulation workflow manager (MDO framework) will be made, to support the interactive construction of MDO workflows and to solve the actual optimization problems. This will provide a means to test the usefulness of the advisory system's functionalities. Moreover, links to relative documentation will be added to the knowledge base. A feedback coupling between the MDO framework and the advisor/knowledge base can provide the possibility to enrich the knowledge base with information obtained during the optimization/simulation process and improve the advisory system by means of machine learning. In addition to this, the system can be expanded to incorporate the possibility to have user-defined architectures and let these be stored in the knowledge base.

Acknowledgements

This research is partly funded by the ITEA2 call 8 project IDEaLiSM of the European Union, Grant No. 13040.

References

[1]Belie, G., "Non-Technical Barriers to Multidisciplinary Optimization in the Aerospace Industry," *Proceedings of the 9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, American Institute of Aeronautics and Astronautics, September 2002.

[2]Agte, J., de Weck, O., Sobieszczanski-Sobieski, J., Arendsen, P., Morris, A., and Spieck, M., "MDO: assessment and direction for advancement – an opinion of one international group," *Structural and Multidisciplinary Optimization*, Vol. 40, No. 1-6, 2010, pp. 17–33.

[3]Kodiyalam, S. and Sobieszczanski-Sobieski, J., "Multidisciplinary design optimization - some formal methods, framework requirements, and application to vehicle design," *International Journal of Vehicle Design*, Vol. 25, No. 1/2 (Special Issue), 2001, pp. 3–22.

[4]Long, T., "The Optimization Assistant - Helping Engineers Explore Designs Through Collaboration," *Proceedings of the 4th International Conference on Intelligent User Interfaces*, IUI '99, ACM, New York, NY, USA, 1999, p. 200.

[5]Anonymous, *iSight User Guide version 8*, Engineous Software, Inc., 2003.

[6]Anonymous, *iSight Reference Guide version 8*, Engineous Software, Inc., 2003.

[7]Anonymous, *Phoenix Integration*, "Optimization Tool Benchmark," [Online]; `http://www.phoenix-int.com/software/benchmark_report/index.php`; visited on 22 September 2014.

[8]Martins, J. R. R. A. and Lambe, A. B., "Multidisciplinary Design Optimization: A Survey of Architectures," *AIAA Journal*, Vol. 51, No. 9, July 2013, pp. 2049–2075.

[9]Cramer, E., Dennis Jr., J., Frank, P., Lewis, R., and Shubin, G., "Problem Formulation for Multidisciplinary Optimization," *SIAM Journal on Optimization*, Vol. 4, No. 4, November 1994, pp. 754–776.

[10]Alexandrov, N. and Lewis, R., "Algorithmic perspectives on problem formulations in MDO," *Proceedings of the 8th Symposium on Multidisciplinary Analysis and Optimization*, American Institute of Aeronautics and Astronautics, September 2000.

[11]Alexandrov, N. and Lewis, R., "Analytical and computational properties of distributed approaches to MDO," *Proceedings of the 8th Symposium on Multidisciplinary Analysis and Optimization*, American Institute of Aeronautics and Astronautics, September 2000.

[12]Alexandrov, N. M. and Lewis, R. M., "Analytical and Computational Aspects of Collaborative Optimization for Multidisciplinary Design," *AIAA Journal*, Vol. 40, No. 2, February 2002, pp. 301–309.

[13]Alexandrov, N. and Lewis, R., "Reconfigurability in MDO Problem Synthesis, Part 1," *10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, American Institute of Aeronautics and Astronautics, August 2004.

[14]Alexandrov, N. and Lewis, R., "Reconfigurability in MDO Problem Synthesis, Part 2," *10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, American Institute of Aeronautics and Astronautics, August 2004.

[15]Dennis Jr., J. E., Arroyo, S. F., Cramer, E. J., and Frank, P. D., "Problem formulations for systems of systems," *Proceedings of the 2005 IEEE International Conference on Systems, Man and Cybernetics*, Vol. 1, October 2005, pp. 64–71.

[16]Brown, N. F. and Olds, J. R., "Evaluation of Multidisciplinary Optimization Techniques Applied to a Reusable Launch Vehicle," *Journal of Spacecraft and Rockets*, Vol. 43, No. 6, November 2006, pp. 1289–1300.

[17]Roth, B. and Kroo, I., "Enhanced Collaborative Optimization: Application to an Analytic Test Problem and Aircraft Design," *Proceedings of the 12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, American Institute of Aeronautics and Astronautics, September 2008.

[18]de Wit, A. and van Keulen, F., "Overview of Methods for Multi-Level and/or Multi-Disciplinary Optimization," *Proceedings of the 51st AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, American Institute of Aeronautics and Astronautics, April 2010.

[19]Lu, Z. and Martins, J., "Graph Partitioning-Based Coordination Methods for Large-Scale Multidisciplinary Design Optimization Problems," *Proceedings of the 12th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference and 14th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference.*, American Institute of Aeronautics and Astronautics, September 2012.

[20]Arora, J. S. and Baenziger, G., "Uses of artificial intelligence in design optimization," *Computer Methods in Applied Mechanics and Engineering*, Vol. 54, No. 3, 1986, pp. 303–323.

[21]Carchrae, T. and Beck, J. C., "Applying Machine Learning to Low-Knowledge Control of Optimization Algorithms," *Computational Intelligence*, Vol. 21, No. 4, 2005, pp. 372–387.

[22]Balachandran, M. and Gero, J. S., "Use of Knowledge in Selection and Control of Optimization Algorithms," *Engineering Optimization*, Vol. 12, No. 2, 1987, pp. 163–173.

[23]Lee, D. and Kim, S.-Y., "A knowledge-based expert system as a pre-post processor in engineering optimization," *Expert Systems with Applications*, Vol. 11, No. 1, 1996, pp. 79–87.

[24]Chan, P. K. M., *A New Methodology for the Development of Simulation Workflows: Moving Beyond MOKA*, Master's thesis, Delft University of Technology - Faculty of Aerospace Engineering, 2013.

[25]Wolpert, D. and Macready, W., "No Free Lunch Theorems for Search," SFI-TR 05-010, Santa Fe Institute, 1399 Hyde Park Rd, Santa Fe, NM, 87501, USA, February 1995.

[26]Wolpert, D. and Macready, W., "No free lunch theorems for optimization," *Evolutionary Computation, IEEE Transactions on Evolutionary Computation*, Vol. 1, No. 1, Apr 1997, pp. 67–82.

[27]Macready, W. G. and Wolpert, D. H., "What Makes an Optimization Problem Hard?" *Complexity*, Vol. 1, No. 5, May/June 1996, pp. 40–46.

[28]Vanderplaats, G., *Multidiscipline Design Optimization: Textbook*, Vanderplaats Research & Development, Incorporated, 2007.

[29]Papalambros, P. and Wilde, D., *Principles of Optimal Design: Modeling and Computation*, Cambridge University Press, 2000.

[30]Keane, A. and Nair, P., *Computational Approaches for Aerospace Design: The Pursuit of Excellence*, Wiley, 2005.

[31]Perez, R., Liu, H., and Behdinan, K., "Evaluation of Multidisciplinary Optimization Approaches for Aircraft Conceptual Design," *Proceedings of the 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, American Institute of Aeronautics and Astronautics, August 2004.

[32]Tedford, N. P. and Martins, J. R. R. A., "Benchmarking multidisciplinary design optimization algorithms," *Optimization and Engineering*, Vol. 11, No. 1, 2010, pp. 159–183.

[33]Yi, S. I., Shin, J. K., and Park, G. J., "Comparison of MDO methods with mathematical examples," *Structural and Multidisciplinary Optimization*, Vol. 35, No. 5, 2008, pp. 391–402.

[34]Sobieszczanski-Sobieski, J., "Multidisciplinary Design Optimization: an Emerging New Engineering Discipline," NASA Technical Memorandum 107761, National Aeronautics and Space Administration, May 1993.

[35]Giesing, J. and Barthelemy, J.-F., "A summary of industry MDO applications and needs," *Proceedings of the 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, American Institute of Aeronautics and Astronautics, September 1998.

[36]Tosserams, S., Hofkamp, A., Etman, L. F. P., and Rooda, J. E., "A specification language for problem partitioning in decomposition-based design optimization," *Structural and Multidisciplinary Optimization*, Vol. 42, No. 5, 2010, pp. 707–723.

[37]iProd Project, 7th Framework Programme of the European Community, G. n. ., "Deliverable 3.6 - Domain Independent iProd Ontology," 2013.

[38]Anonymous, *The World Wide Web Consortium (W3C)*, "OWL 2 Web Ontology Language," [Online]; `http://www.w3.org/TR/owl2-overview/`; visited on 3 November 2014.

[39]Drummond, N. and Shearer, R., "The Open World Assumption," [Online]; `http://www.cs.man.ac.uk/~drummond/presentations/OWA.pdf`; visited on 11 November 2014, 2006.

[40]Anonymous, *Information Systems Group, University of Oxford*, "HermiT OWL Reasoner," [Online]; `http://www.hermit-reasoner.com/index.html`; visited on 3 November 2014.