13th AIAA/ISSMO Multidisciplinary Analysis Optimization Conference
13 - 15 September 2010, Fort Worth, Texas

AIAA 2010-9177

VisualDOC: New Capabilities for Concurrent and Integrated Simulation and Design

Santosh Tiwari[*], Hong Dong[†], Brian C. Watson[‡], and Juan P. Leiva[§]

Vanderplaats Research & Development, Inc., Novi, MI, 48375, USA

In this paper, the redesigned VisualDOC software; a tool for design process definition, integration, execution, and automation is presented. This new version also has significant emphasis on design modules such as Optimization, Design of Experiments, Response Surfaces, and so on. The new VisualDOC is a system that allows the designer to graphically create a connected workflow of components resembling a traditional flowchart and define each component in the flowchart appropriately. This improved version presents a new approach to defining flow of execution, information transfer and associativity, and concurrent monitoring (visualization) of the execution process. It is designed to be a flexible and powerful tool that can be used to model virtually any design process. The new VisualDOC combines modeling, simulation, and visualization/presentation aspects of a design process into a single and coherent software tool. Here, a conceptual description and salient features of the new design of VisualDOC are presented.

I. Introduction

In a competitive and ever-demanding business and development environment, designers and engineers need access to increasingly more sophisticated tools to assist them in the design process. As the complexity of a design process and the knowledge (meaningful information) associated with it grows, need for more efficient and capable tools to manage and process that information also grows. More importantly, as the designers and engineers gain more experience and expertise, they move on to higher level tasks that require further critical thinking and analysis. The lower level tasks are eventually taken over by expert systems such as software tools which can execute the tasks and processes well understood, thereby freeing the engineer to focus on the next challenges.[1] This transition of tasks from a human expert to a software system continuously occurs as a design process becomes well understood and it becomes feasible to implement it in a software system. The new version of VisualDOC described in this paper is conceived as a coherent and completely integrated set of such software and algorithmic tools which can be used to assist the designers and engineers.

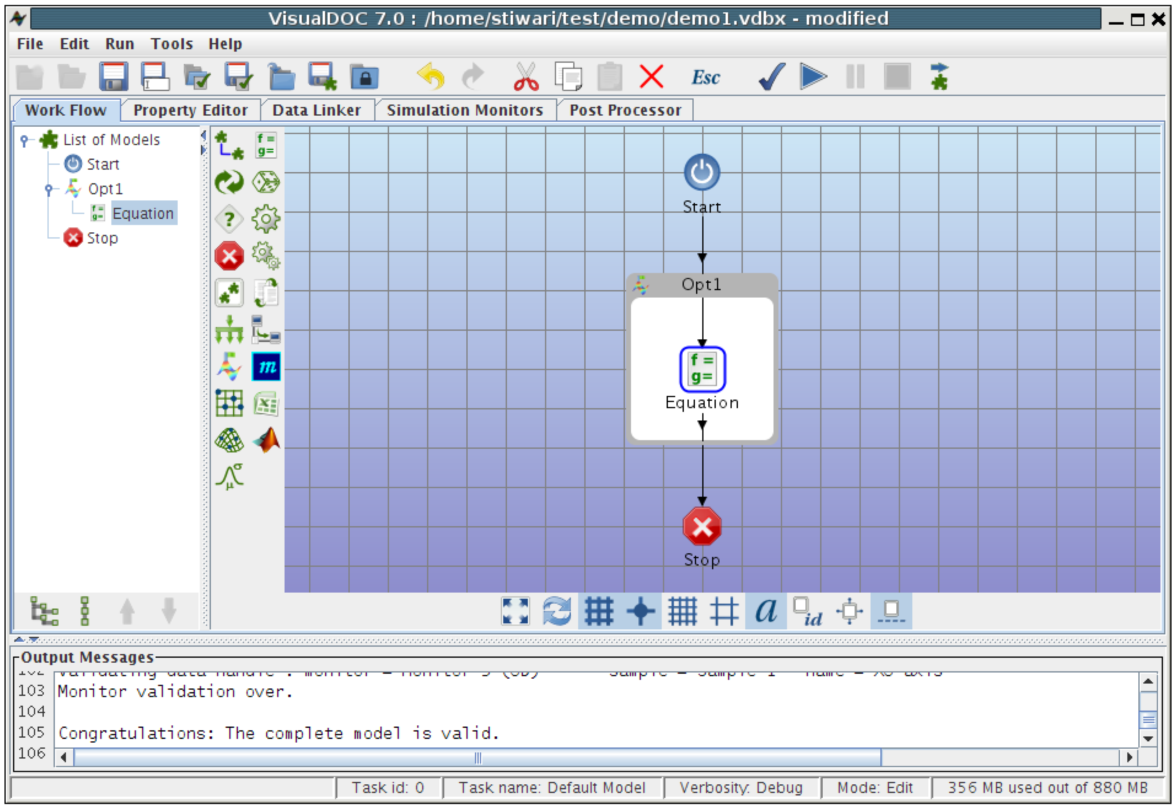
The computational tools that are often used during the execution of a design task[2,3] come from domains such as optimization, design of experiments, response surfaces, reliability and robustness, and CAE analysis modules for various types of physical models. Apart from these algorithmic tools, modules to integrate these systems together, facilitate communication between them, and store/process/visualize the associated information are also needed. The new VisualDOC is a software system that contains these tools as components and has interfaces to allow it to talk to external software that could potentially be used during a simulation/design process. In this new version, each of these modules is treated as a component and a design task is a connected workflow of such components. In this paper, the flowcharting, data definition, data transfer, simulation parameterization, and simulation monitoring aspects of the new VisualDOC system are presented. More specifically, this paper explains how the improved VisualDOC makes it easy to integrate multiple systems and sub-systems together, define the flow of execution and flow of associated data independently as well as in concert, and perform concurrent simulation, analysis, and visualization. It allows

*R&D Engineer, Vanderplaats Research & Development, Inc., 41700 Gardenbrook, Novi, MI 48375, Member AIAA
†R&D Engineer, Vanderplaats Research & Development, Inc., 41700 Gardenbrook, Novi, MI 48375
‡Director of Technology Development, Vanderplaats Research & Development, Inc., 41700 Gardenbrook, Novi, MI 48375
§President and COO, Vanderplaats Research & Development, Inc., 41700 Gardenbrook, Novi, MI 48375, Senior Member AIAA

the designer to navigate the design process (step-by-step if desired), modify certain aspects of the workflow as the simulation proceeds, and perform concurrent decision making. It is interesting to note that this new version makes it easy to execute various MDO methodologies[4] such as CSSO, BLISS, MDF, IDF, etc. It also makes it easy to execute multi-level simulation processes such as ATC[5] and EMDO.[6] Also highlighted in this paper are a few design elements of VisualDOC which enforce and encourage established design practices. In the sections that follow, a conceptual sketch of design process definition, execution, and monitoring using the VisualDOC software is presented.

This paper does not intend to provide a comprehensive list of features and capabilities of the software nor it demonstrates its application to a specific problem. Rather this paper primarily focuses on its conceptual design and related framework. This paper is organized as follows. In section 2, the VisualDOC Workflow is described. Section 3 contains a description of the Component Editor of the new software. The data communication (information transfer) using the Data Linker is described in section 4. Section 4 also includes a description of process parameterization and automation capabilities of this new version. In section 5, the concurrent process monitoring capabilities are discussed. Finally, section 6 concludes this paper.

II. VisualDOC Workflow



```
Monitor validation over.

Congratulations: The complete model is valid.
```

Task id: 0 | Task name: Default Model | Verbosity: Debug | Mode: Edit | 356 MB used out of 880 MB

Figure 1. The Main VisualDOC Window

When launched, the main window of VisualDOC appears with the *Workflow Editor* (Fig. 1) as the selected tab. The first step in the design process definition using this new version is the creation of the workflow. The designer (user) drags the desired components from the toolbars and drops them on the canvas. The flowchart is then completed by connecting different components. The connection between different components represents the flow of execution. The resultant flowchart is referred to as the VisualDOC model. The visual appearance of this model is designed to resemble a typical flowchart so that it is intuitive to create and easy to manipulate. The new version of the software allows for creation of sub-flows (up to any level) and provides components to virtually program any kind of logic including conditional statement blocks, loops, and parallel sub-flows as part of the flowchart. A snapshot of a typical VisualDOC flowchart is shown in Fig. 2. There are two distinct advantages of this visual representation.

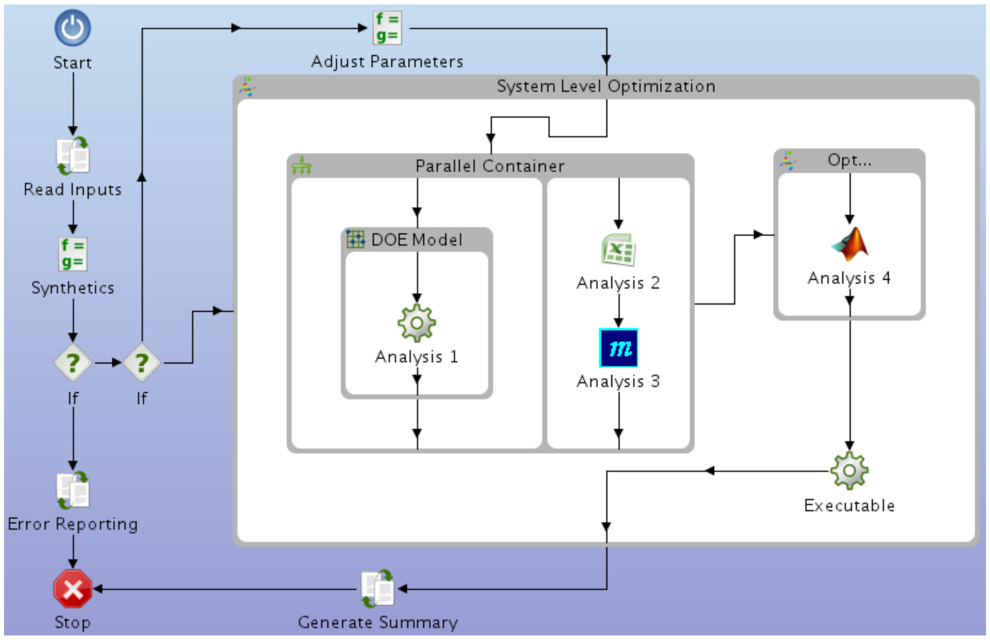


Figure 2. A VisualDOC Flowchart

1. The flowchart is designed to be as similar as possible to what the designer (user) would typically draw if the entire process was conceptualized on a piece of paper. It thus makes it natural and easier for the designer to communicate his/her knowledge of the design process to the software.

2. It assists the user to correctly envision the flow of execution. It does not allow creation of invalid workflows. Any process that is defined by the user is checked for the semantic validity of connections. It also enforces the user to correctly define various sub-systems (if multiple sub-flows are used) and their relation.

As shown in the flowchart in Fig. 2, the user can model virtually any kind of design process. Each component (block) in the flowchart performs a specific function. Any component that has a flow-logic of its own (e.g. loops, optimization, parallel container, etc.) contains a sub-flow inside. In this new version, external components (that are not part of VisualDOC) can also contain sub-flows if desired. The user can drag-and-drop, move around, and cut-copy-paste blocks (components) at will to modify the flowchart. The flowchart defines the flow of execution. The execution always begins at the *Start* block (there is always a single *Start* block in the flowchart which cannot have any incoming connection) and ends at one of the multiple *Stop* blocks. The execution is assumed to be over when a *Stop* block is encountered. If there is conditional or parallel branching, then a branch of execution is assumed to be over when it encounters a *Stop* block. The flow of execution is always sequential unless it is inside a parallel container; in such a case, all the sub-flows execute simultaneously. Every component in the flowchart can have any number of inputs, but a single output except an *If* block which has two outputs (only one of which will actually be executed depending on the condition). The following are some of the highlights of the VisualDOC workflow:

- The components (blocks) themselves do not contain the logic of the flow. They merely perform a task. The only exception to this rule is components that can contain a sub-flow; in such a case, they perform their task and then transfer the control to the contained sub-flow for execution. When the sub-flow is done executing, the control is transferred back to the containing model. Here, the flow of execution is defined by the connections that the user creates (primarily via drag-and-drop). The user can modify the flow of execution by rearranging the connections and relocating the components.

- The new version fully supports and facilitates reusability and modularity. If identical functionality is required at different places in the sub-flow, only one component needs to be created and configured. It can then be copy-pasted/referenced at other places. Also, if a slight change in functionality is required, then the user can copy-paste the component and then edit the copy as desired.

- The redesigned software also supports full undo/redo capabilities throughout. Every operation can be undone/redone. The number of undo/redo steps is user configurable. It also fully supports time-stamps where the user can save a snapshot of the model (referred to as a task), and then continue editing the model. At any time, the user can switch back and forth between different tasks as desired.

- The VisualDOC workflow can also be interpreted as a directed cyclic graph with a single root node. It is possible to create cyclic workflows and a cycle may span multiple levels (sub-flows). Furthermore, if conditional blocks (*If* modules) are present in a cycle, a simulation may never complete. It automatically checks for all possible cycles in a workflow during the model validation phase and highlights all the cycles (if conditional blocks are present, it checks all possible branches recursively to make sure that infinite loops do not occur). It also checks for disconnected models and incomplete connections. It does not allow the user to create any invalid connection at any stage of the modeling process.

- Since a VisualDOC workflow is a collection of sub-flows, the user can execute only part of a sub-flow if desired (helpful during the model validation and testing phase). The user can also execute a single block in the workflow. Each component (block) has a pre-condition and a post-condition. Before a block can be executed, its pre-condition must be satisfied, and if it has successfully executed, the post-condition will be satisfied. Thus, to execute a sub-flow, it is necessary to satisfy the pre-condition for all possible entry points of that sub-flow. An example of a pre-condition is: the inputs to the entry point must be defined and have a value before that component is executed.

- A powerful model validation and testing feature of the new software is the debugging interface which is similar to a debugger built into IDEs for programming languages. The user can stop, pause, continue, and advance a single step of the design process manually by mouse clicks to step-through the model execution. The user can also add break points in the model which when encountered during the execution prompts the user for further action. The model validation and stepping through is accompanied by visual feedback that shows the current block being executed and state of the VisualDOC model. The debugging process is completely mouse-driven and requires minimal keyboard input. The break points can be added/removed when the debugging is in progress.

The next step after creating the workflow is configuring each component (block) which is discussed in the next section. The user can always switch back and forth between flowcharting, component editing, and linking interfaces.

III. VisualDOC Component Editor

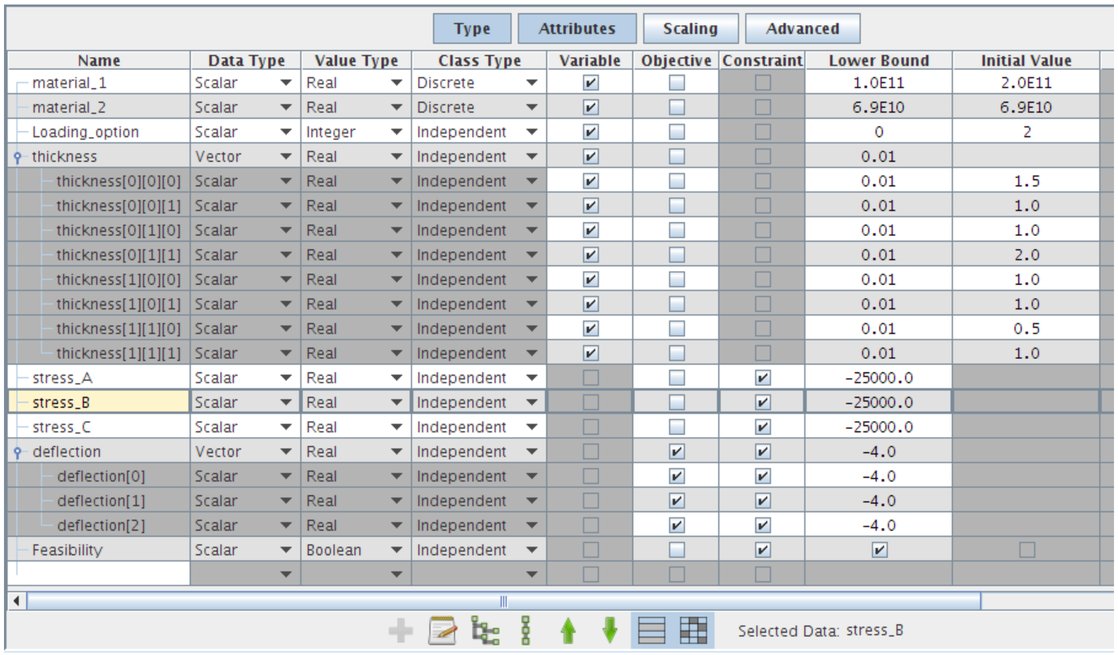
The editing of a component is a two-step process (which can be performed in any order) in the redesigned software. The first step involves defining simulation data (inputs and outputs) for a component. A component may also have additional data which may not be input/output (e.g. local/dependent variables). The user interface for defining the simulation data is the same for every component. Any data/parameter whose value must be known before a component is executed is an input to that component. Any data/parameter whose value is modified as a result of execution of that component is an output from that component. No data is allowed to be the input and output at the same time. If a component needs to read/write values to the same data, data related parameters can be created (discussed later). The following three data types are supported by the new version of the software:

1. *Scalar*: A scalar data is a field with a single value. The value type may be real, integer, boolean, or string. This is the simplest supported data type.

2. *Vector*: A vector data type is an N-dimensional matrix ($N \geq 1$). The size of each dimension is configurable. All the elements in the vector themselves are scalars with the same value type. The user can independently edit the value of each element or can use the vector to simultaneously effect the change on all the elements.

3. *Structure*: A structure is a collection of data. Thus, a structured data type can contain other structured data types. The primary purpose of this data type is to group similar data together. Each data field in a structure is independently configurable. The creation of structured data type is automatically done by VisualDOC, and hence this option is not explicitly made available to the user.

A data may also have additional attributes such as independent, dependent, constant, or discrete. A snapshot of the data editor is shown in Fig. 3. The data shown in Fig. 3 corresponds to a single component (in this case, the optimization component).



Name	Data Type	Value Type	Class Type	Variable	Objective	Constraint	Lower Bound	Initial Value
material_1	Scalar	Real	Discrete	✓	☐	☐	1.0E11	2.0E11
material_2	Scalar	Real	Discrete	✓	☐	☐	6.9E10	6.9E10
Loading_option	Scalar	Integer	Independent	✓	☐	☐	0	2
thickness	Vector	Real	Independent	✓	☐	☐	0.01	
thickness[0][0][0]	Scalar	Real	Independent	✓	☐	☐	0.01	1.5
thickness[0][0][1]	Scalar	Real	Independent	✓	☐	☐	0.01	1.0
thickness[0][1][0]	Scalar	Real	Independent	✓	☐	☐	0.01	1.0
thickness[0][1][1]	Scalar	Real	Independent	✓	☐	☐	0.01	2.0
thickness[1][0][0]	Scalar	Real	Independent	✓	☐	☐	0.01	1.0
thickness[1][0][1]	Scalar	Real	Independent	✓	☐	☐	0.01	1.0
thickness[1][1][0]	Scalar	Real	Independent	✓	☐	☐	0.01	0.5
thickness[1][1][1]	Scalar	Real	Independent	✓	☐	☐	0.01	1.0
stress_A	Scalar	Real	Independent	☐	☐	✓	-25000.0	
stress_B	Scalar	Real	Independent	☐	☐	✓	-25000.0	
stress_C	Scalar	Real	Independent	☐	☐	✓	-25000.0	
deflection	Vector	Real	Independent	☐	✓	✓	-4.0	
deflection[0]	Scalar	Real	Independent	☐	✓	✓	-4.0	
deflection[1]	Scalar	Real	Independent	☐	✓	✓	-4.0	
deflection[2]	Scalar	Real	Independent	☐	✓	✓	-4.0	
Feasibility	Scalar	Boolean	Independent	☐	☐	✓	✓	☐
				☐	☐	☐		

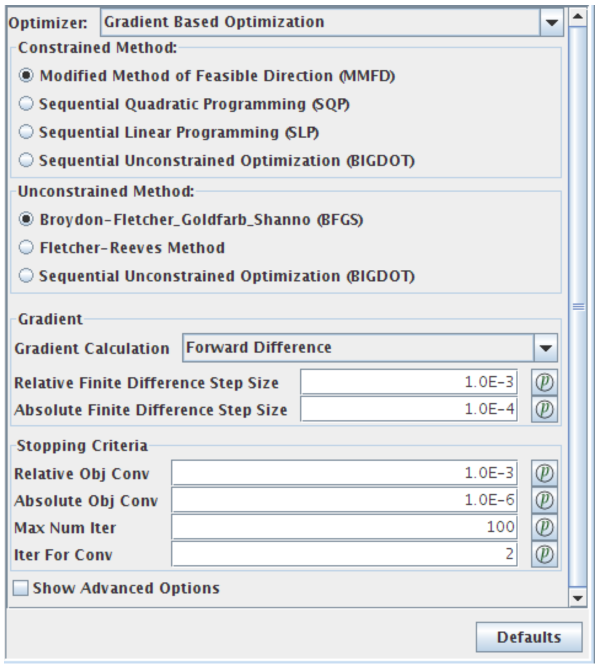
Selected Data: stress_B

Figure 3. The VisualDOC Data Editor

From the Fig. 3, it is obvious that the simulation data has additional attributes (e.g. lower bound, initial value, upper bound, etc.) The additional attributes are component specific and are shown as columns in the data editor. A novel aspect of this new version is that additional attributes (properties) can be assigned to a data after it is created. Every data has some common attributes such as *name*, *value*, *input/output*, etc. The number and types of attributes that can be associated to a data generally increase as a component becomes more specialized. For example, a generic data may only have attributes such as *name*, *value*, *input/output*, etc., but the same data when used in an optimization component may have additional attributes such as whether it is a *variable/objective/constraint* and its *lower/upper* bounds if any. Again, that same data when used in an analysis component may only be identified by *inputs* and *outputs* (responses). Having a flexible attribute system allows downstream developers to assign additional meaning to a data as and when needed. It is a facility primarily intended for the downstream developers and API users of VisualDOC who wish to perform customization and/or add extra functionality. Furthermore, the new data editor allows for automatic creation and configuration of additional attributes based on the value of other attributes (e.g. in an optimization component, if a data is set as a variable, it automatically becomes an output from that component and lower/upper bound attributes are added). The second step in configuring the component is the component specific property editor. In this editor, the component specific details are defined. The property editor for the optimization component is shown in Fig. 4.

A. Parameters

Another novel concept introduced with the new VisualDOC is the notion of *parameters*. Parameters are data associated with the components which are not explicitly defined (and shown) in the data editor and are not treated as inputs or outputs. Their meaning (interpretation) and attributes are context dependent. The user does not explicitly create or edit any parameters, rather they are provided by the component as



Optimizer: Gradient Based Optimization

Constrained Method:
- ● Modified Method of Feasible Direction (MMFD)
- ○ Sequential Quadratic Programming (SQP)
- ○ Sequential Linear Programming (SLP)
- ○ Sequential Unconstrained Optimization (BIGDOT)

Unconstrained Method:
- ● Broydon-Fletcher_Goldfarb_Shanno (BFGS)
- ○ Fletcher-Reeves Method
- ○ Sequential Unconstrained Optimization (BIGDOT)

Gradient
- Gradient Calculation: Forward Difference
- Relative Finite Difference Step Size: 1.0E-3
- Absolute Finite Difference Step Size: 1.0E-4

Stopping Criteria
- Relative Obj Conv: 1.0E-3
- Absolute Obj Conv: 1.0E-6
- Max Num Iter: 100
- Iter For Conv: 2

☐ Show Advanced Options

Figure 4. The Optimization Component Property Editor

and when needed. The parameters can be used to fully automate any simulation inside VisualDOC. The primary purpose of the parameters is to transfer information (communicate values) that is not contained in the user-defined data. As such, the parameters are only visible in the linking interface (discussed in the next section). There are three types of parameters available in this improved version of the software.

1. *Data related parameters*: This type of parameter is associated with the data that the user creates. For example in an optimization component, corresponding to every data (say variable x), there exist parameters such as $x_{initial}$ (initial value - starting point for optimization), x_{best} (best value found so far), x_{lower} (lower bound), x_{upper} (upper bound), etc. It is possible (and often desired) that the user may want to read/write these values from other components. It should also be noted that each of these attributes (parameters) are shown as a column in the data editor. Thus, in this new version of the software, the user can create a parameter for any cell (if the data and attribute corresponding to that cell allow it) in the data table, and can use the created parameter for linking. Similar to data, the parameters can also be scalar, vector, or structure. The parameters and data can read from and write to each other. By default, all data related parameters are disabled, and can be enabled by right clicking on the cell and choosing the "*Enable parameter*" option.

2. *Simulation parameters*: This type of parameter is pre-configured and created by the component. They are not associated with the data but are produced as a result of the component execution. For example, in the DOE (design of experiments) component, the coefficients of the approximate model are output as a result. The DOE component automatically creates parameters associated with the coefficients and makes them available for linking. Generally, a large number of simulation parameters are automatically created by each design component. By default, the simulation parameters are hidden, and they need to be made visible in the linking interface.

3. *Tuning parameters*: This type of parameter is associated with the fields that are typically shown in the component specific property editor. For example, for the optimization component (Fig. 4), the tuning parameters could be step sizes, iterations, convergence criteria, etc. Each component creates a parameter associated with every tunable attribute and makes it available for linking. By default, all the tuning parameters are disabled. These can be enabled by clicking on the button to the right of the tunable attribute (the button is marked with symbol p in Fig. 4).

The linkable parameters provide a very powerful and flexible framework for communicating information between components. For example, if performing an optimization inside a for loop the user wants to vary the random seed for each optimization run, the loop counter could be connected to a synthetic block that generates a pseudo-random value using the loop counter as input. The output value from the synthetic block can then be linked to the random seed tuning parameter of the optimizer. Similarly, if the name of some input/output file depends on some counter or other variables, the name of the file can be generated inside a synthetic block as if it was a string variable and can be linked to the file descriptor of the requisite components. Such a facility allows for extensive scripting and automation of the simulation process in VisualDOC. After the simulation data, parameters, and properties for each component are defined, the next step is to link the data and parameters between different components.

IV. VisualDOC Data Linker

The data linking interface in the new design allows the user to link data and parameters between any two distinct components which belong to the same sub-flow. Only the data having the same value type can be linked. The data and parameter linking (flow of information) is independent of the execution control. Data linking implies transfer of values from one component to another. The value of the data and parameters are updated after a component finishes executing. The same data (information) can be referred to by different names (identifier) in different components. Data linking between two sub-flows which are inside the same parallel container is not allowed due to synchronization issues. This is due to the fact that if component A depends on the output from component B, then A and B cannot be executed simultaneously (A will have to execute after the execution of B has finished). In such a case, the user should break the parallel container into sequential (dependent) and parallel (independent) sub-flows; i.e. the portions of the sub-flow that do not depend on each other for data can be executed simultaneously. A snapshot of the data linking interface is shown in Fig. 5.

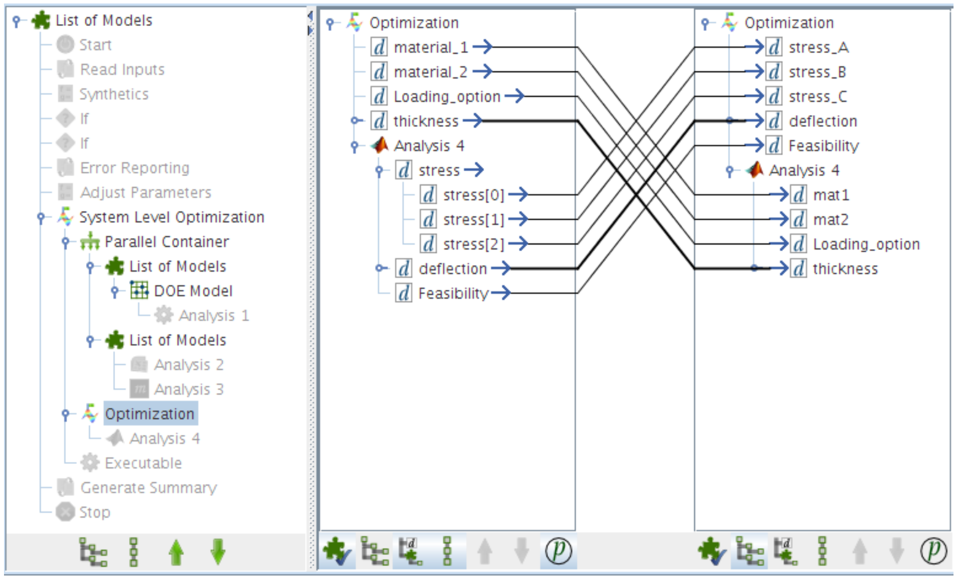
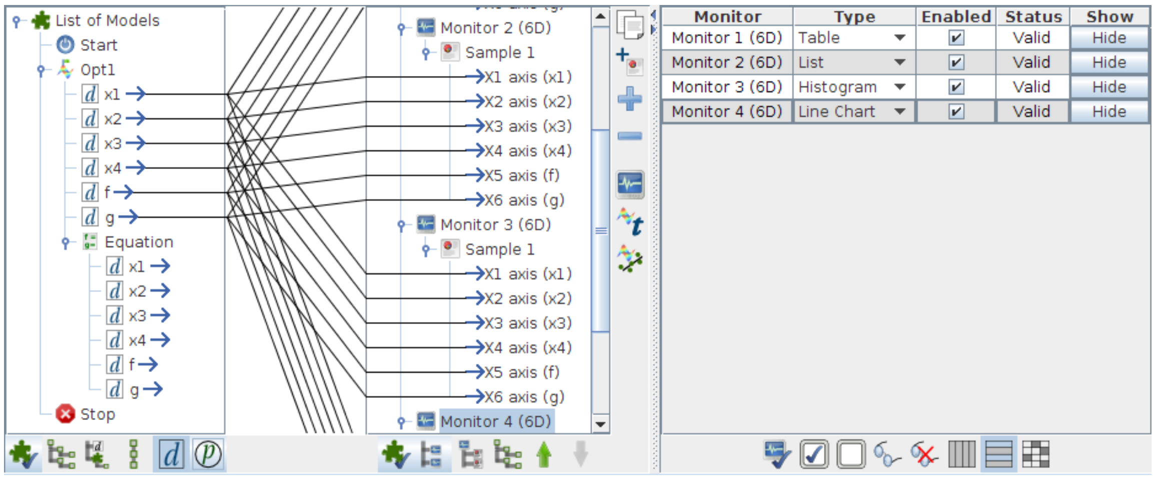


Figure 5. The VisualDOC Data Linker

In Fig. 5, the inputs/outputs from the *Optimization* component are linked to the *Analysis 4* component. Three types of links are supported. i) scalar to scalar links: The values are simply copied from one scalar data/parameter to another, ii) scalar to vector links: The value of the scalar data/parameter is copied to all elements in the vector, and iii) vector to vector links: The value of each element in the source data/parameter is copied to the corresponding element in the destination data/parameter. The vector to vector links are shown with bold lines. Two vectors can only be linked directly if they have the same number of dimensions and identical size. A data/parameter can have multiple outgoing links, but only one incoming link. The link creation is accomplished via drag-and-drop. To create a link, the user drags the source data and drops it on the destination data. Continuous visual feedback is provided throughout the drag-and-drop operation. Since linking is accomplished via drag-and-drop, it is a single step operation. Whenever the linked data/parameter change in a way such that the existing links become invalid, they are highlighted in red, and the user must either remove those links, or edit the data appropriately.

V. Simulation Monitors

It is often desirable to concurrently monitor the progress of a simulation. In VisualDOC, the simulation (execution) of the model is completely independent of the monitoring process. The simulation monitors (e.g. visualization plots, data tables, and real-time animations) are additional modules that the user can add before, during, or after a simulation to monitor desired simulation parameters. The connections between different components define the transition (transfer of execution) from one component to another. Also, it is after the execution of a component that the simulation data is updated. Hence, the simulation monitors are notified whenever a component finishes executing. With this version of software, the user can add any number of simulation monitors and independently configure each of them. A snapshot of the interface to create and configure the monitors is shown in Fig. 6.



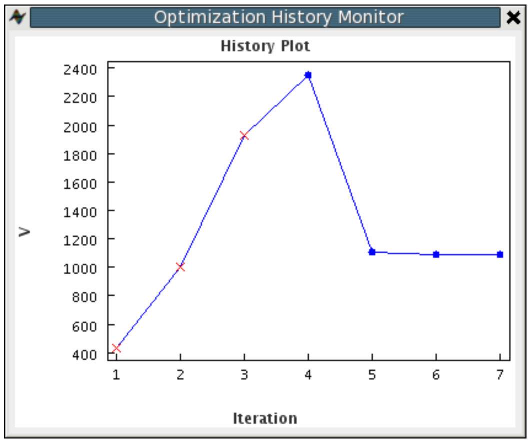
Monitor	Type	Enabled	Status	Show
Monitor 1 (6D)	Table	✓	Valid	Hide
Monitor 2 (6D)	List	✓	Valid	Hide
Monitor 3 (6D)	Histogram	✓	Valid	Hide
Monitor 4 (6D)	Line Chart	✓	Valid	Hide

Figure 6. The VisualDOC Monitor Editing Interface

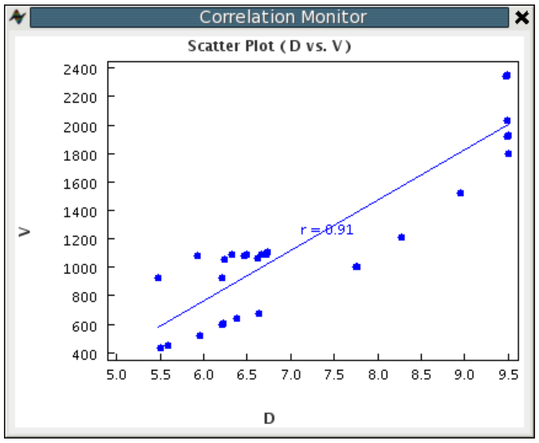
The following are some of the salient features of the monitors.

- The simulation monitors themselves are not part of the workflow, but they are observers that fetch and display the desired information as per the user requirements. Thus, the monitors do not interfere in the execution of the VisualDOC model.
- The monitors can be used both during (concurrent monitoring) and after a simulation (post-processing). Some monitors, e.g. summary reports can only be invoked after a component has finished executing.
- The monitors can be added/edited/removed while the simulation is under progress. Furthermore, the user has the choice to enable/disable a specific monitor at any time during the simulation.
- The monitors in this new version of the software are true observers in that there is absolutely no communication between the model execution engine and the monitoring process. Furthermore, the models do not know anything about the monitors (the model code does not rely on the monitors). However, the monitors know about the models (they rely on the model code to determine what data to fetch from the database) and communicate directly with the database engine to fetch the relevant data for display.

The new version of VisualDOC also provides pre-configured monitors for design components (e.g. variable/objective/constraint history monitor for optimization). A snapshot of two pre-configured monitors is shown in Fig. 7. The monitors are automatically disabled when the execution is performed in batch mode or run in the background. Furthermore, when the simulation is run in non-interactive mode, the monitors



(a) Optimization History Monitor



(b) Correlation Monitor

Figure 7. A snapshot of VisualDOC monitors

default to displaying the data as formatted text output on the console (and/or the user-specified log file). The monitors are configured (fed data) via drag-and-drop. The data/parameter to be monitored is simply dragged and dropped on the appropriate data handles of the monitors. Another important feature of the new version is that the main window itself acts as a monitor when a simulation is running. The flow-chart highlights the currently running model, the data editor shows the current value of the data in the components and it updates itself as soon as new values are available, the message area reports all the simulation related information (if debug mode is enabled), etc. When a simulation is running, the model cannot be edited (all the editing functionality is disabled), but all the display/visualization related functionality is still available. As soon as the simulation is over (because of completion, user action, or some error), the main window automatically returns to the edit mode. It is also possible to launch the simulation (model execution) in batch mode as a separate process; in such a case, the user can continue to edit the current model, whilst a different snapshot (task) is running in the background.

VI. Conclusion

In this paper, we have presented the conceptual sketch of the basic skeleton of the new VisualDOC model definition and execution paradigm. We have discussed five basic elements (flowcharting, data creation and editing, parameter usage, data linking, and process monitoring) of the software. This paper describes the fundamental design principles on which this improved version is based. The snapshots included in this paper are from an early version of the new software (currently under development and testing). The primary emphasis behind the new design of VisualDOC is to make it an easy, intuitive, flexible, and powerful tool for design process definition and execution. It also contains a comprehensive collection of tools for process monitoring and real-time visualization.

References

[1]Chandrasekaran, B., "Generic Tasks in Knowledge-based Reasoning - High-level Building Blocks for Expert System Design," *IEEE Expert*, Vol. 1, 1986, pp. 23–30.

[2]Ullmann, D., *The Mechanical Design Process*, McGraw-Hill, St. Louis, MO, 1992.

[3]Pahl, G. and Beitz, W., *Engineering Design - A Systematic Approach*, Springer-Verlag London, 2nd ed., 2003.

[4]Perez, R. E., Liu, H. H. T., and Behdinan, K., "Evaluation of Multidisciplinary Optimization Approaches for Aircraft Conceptual Design," *10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, No. AIAA 2004-4537.

[5]Michelena, N., Park, H., and Papalambros, P. Y., "Convergence Properties of Analytical Target Cascading," *AIAA Journal*, Vol. 41, No. 5, 2003, pp. 897–905.

[6]Vanderplaats, G. N., "EMDO - An Engineering Approach to Multidisciplinary Design Optimization," *11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, No. AIAA 2006-7110.