ABSTRACT
This paper presents a functional modeling application, FunctionCAD, based on integrated functional and process modeling within the Function Design Framework. FunctionCAD provides for mixed, hierarchical models of environment, process and function with relationships represented via flows of material, energy and signal. This paper discusses the technical details of the FunctionCAD software including its two major components: (1) the library, libFCAD, for handling the internal representation of the model and (2) the GUI for user-based model manipulation and visualization. The application of FunctionCAD within a computational, function-based conceptual design process is discussed along with the plugin manager and interface that have been developed as a part of the FunctionCAD software to allow interconnectivity with existing conceptual design tools. And finally, a detailed description of model creation within the FunctionCAD environment is provided to illustrate software operability.

1 INTRODUCTION
A functional model may be considered a framework for design activities. Within this framework, customer or societal needs are first identified. Then each need is used to identify necessary functionality; functions are aggregated, and a single model is generated representing all of the transformations expected by the customer base. This process is often considered the translation of “customer speak” into “engineering speak,” and from this point forward, function may be used to seed concept generation where solution principles are paired to functionality and are evaluated for their suitability, behavior analysis is performed to analyze product dynamics, and function-based failure analysis investigates risk potential. As solutions are chosen, the functional model may be updated to reflect necessary supporting functions, and further concept generation may be performed for the newly identified functionality. The continuation of this iterative process through a product design cycle insures that customer needs are inherently designed into a product, and concludes with a pairing of need to function to component to behavior. Once this pairing is established, it may be uploaded to a design repository for archival and inspiration of future design activities.

The design process, however, rarely goes this smoothly, and while a number of tools have been developed to assist designers along the way, the cohesiveness of the tools remains to be fully realized. To that end, this paper presents research on the development of an extensible software application, FunctionCAD, which utilizes the Function Design Framework [1] for the generation of hierarchical models of function, process and environment. FunctionCAD is based on a modular architecture and utilizes a plugin interface to increase versatility and allow for future expandability. Its application is designed to be a part of a computational conceptual design framework utilizing tools developed to assist design automation. A survey of the automated design tools anticipated to initially integrate with FunctionCAD is provided in this paper as well as an overview of the Function Design Framework. The technical details of the FunctionCAD software are then discussed followed by an operation overview.
2 BACKGROUND
In the late 1980s, Hundal and Byrne [2] first recognized the need for the computerization of systematic design and developed function-based conceptual design software for the development of functional models. Functionality is drawn from a function database and three flow types are supported: energy, material and signal. This is later updated by Hundal and Langholtz [3] with the addition of a solution database such that solution principles can be selected and paired with model functions.

In a similar vein, Bracewell and Sharpe [4] generate function-means tree structures within the Schemebuilder environment and apply bond graphs for design reasoning and to assist with conceptualization and embodiment of interdisciplinary design problems. To address flexibility issues with computational representation, Gorti et al. [5] present an object-oriented approach to generate hierarchical models for the product and design process following top-down (function oriented), bottom-up (product oriented), step-wise or constraint-driven approaches. To assist a designer with concept selection and behavior analysis, Deng et al. [6] present a computerized environment to support a function-based conceptual design of mechanical products. Within their framework, functional decompositions are generated, behavior scenarios are detailed, and physics-based attributes are mapped into the conceptual design from a physical phenomena library.

A number of approaches toward the computerization of conceptual design activities have been developed in systems engineering and are based on Unified Modeling Language (UML) or Systems Modeling Language (SysML). Recognizing the need to create a framework for conceptual design from function-to-form, Roy et al. [7] develop an object-oriented environment using UML and C++/Java which focuses on mapping artifacts and their behavior to functionality. Their work recognizes the importance of functional iteration and specifies that as artifacts and behavior are mapped into a functional model, functionality is added, and during this process, function becomes more closely related to attributes and artifacts. FMDesign, presented in [8], utilizes an UML-based information model to integrate function-means trees, stakeholder trees, requirement trees, product concept trees, implementation trees as well as behavior models developed in Modelica. Product models developed in FMDesign are stored in a design library database, and through the generic object inheritance mechanism presented in [9], design artifact information stored in the library database can be passed on to new design artifacts creating a repository for artifact reuse. In [10], a similar Model-Driven Development (MDD) approach, is presented which uses SysML as a formal modeling language to capture multi-disciplinary design knowledge, and an algorithm-based approach is utilized to automate system-level analysis tasks.

Toward the end of the nineties, commercial software packages also begin to assist with requirement mapping and function identification. Software flowcharting packages such as VISIO [11], OmniGraffe [12] or Concept Draw [13] include templates for generating static block or flow diagrams as well as function and process models based on standards such as IDEF0 [14] and UML [15]. Also, commercial systems engineering packages geared at Product Life Cycle Management (PLM) such as Agile PLM from Oracle [16], ENOVIA from Dassault Systemes [17], Teamcenter from Siemens [18], and TEAMs from Qualtech Systems [19] are developed to link conceptual design with detailed design. These systems engineering packages boast a number of features such as allowing for requirement, stakeholder and specification linking as well as function-tree modeling and integration with CAE and CAD activities. Some systems engineering packages even interface with flowcharting applications; for example, Teamcenter’s live interface with VISIO for visualization of design element interconnectivity [20]. These applications have been developed from a systems engineering view point and provide excellent tools for traceability and change propagation; however, their integration with creativity-based design engineering applications is nonexistent. These systems engineering tools are more for design representation than design inspiration.

In the vein of engineering design, function tends to be used as part of the abstraction process to enhance design creativity. A functional model, employed during conceptual design, is based on what a product must do, instead of the means of how a design will be done. The application of function provides many benefits to the conceptual design process such as: explicit identification of customer needs, comprehensive understanding of the design problem, enhanced creativity through abstraction, innovative and creative concept generation with a focus on what will be done, and structured organization that can be applied to both the design problem and the design team [21, 22].

In an effort to standardize the use of functional representations in engineering design, Szykman et al. [23] propose a standard for the computer-based representation of function. Function and flow taxonomies are represented independently as separate schemas and model elements include artifacts, flows and functions. Szykman et al. [23] discuss the importance of standardized taxonomies for function and flow built into the structure of the modeling application, and in [24], Extensible Markup Language or XML is proposed for the representation of both function and flow schemas. The taxonomy proposed by Szykman et al. [23] is based on the function and flow organizational approach outlined in Pahl and Beitz [25], and its roots can be traced back to value analysis where Miles [26] and Rodenacker [27] first used verb-noun pairs and input-output transformations to describe product functionality. The taxonomy proposed by Szykman et al. [23] has subsequently been reconciled with the independently proposed taxonomy of Stone and Wood [28] to form the Functional Basis [29].

3 PRIOR WORK

3.1 Function Design Framework
The Function Design Framework (FDF), first proposed in [1], integrates functional modeling based on the Functional Basis [29] with process modeling [30, 31] to provide a unified modeling architecture where function provides depth to system modeling and process provides breadth. A single architecture of process and functional modeling allows for the creation of models at user defined levels of fidelity which can be created across multiple operational modes of a system. FDF utilizes the following terminology as descriptors of its hierarchical models:

- **Functional Modeling**: The overall approach to modeling what a product must do in terms of elementary operations such that the product may achieve an overall goal or purpose [32].
- **Process Modeling**: The overall approach to modeling a series of customer-driven, product-based actions related through input and output flows, the product being designed, and time.
- **System**: The combination of artifacts and actions which together from a complete, operational product. In the realm of product design, components are combined to produce a complete functioning product to meet identified customer needs. The system combines the product with its use and its usage environment.
• **Flow**: A material, energy, or signal, which interacts with the product; flows are expressed as nouns [32].

• **Boundary**: An outer limit defining the space where all actions and related energies, materials and signals must originate and exist.

• **Function**: A description of an operation, expressed as the active verb, performed by an artifact, as a part of a larger product, to transform an input flow to a desired output flow [32].

• **Configuration**: A specific discrete instance of the overall functionality of the product occurring as a part of an event. Collectively many configurations define an event of the product.

• **Event**: A set of configurations of a product, which may relate to the environments where the product is used, changes to the operability of a product, specific applications of a product, or sequencing of operations during the usage of a product.

• **Process**: The sum of defined events that occur with respect to the product as a whole and aim to meet a particular goal. Processes are tied together via the product, material, energy and signal flows.

• **Environment**: The sum of the system’s surroundings, product operations, configurations and customer interactions.

To generate models within FDF, a methodology consisting of four high-level steps is presented in [1]. A summary of these steps is regenerated below:

1. Identify the boundaries of the environment in which the system is being designed to operate. A single environmental boundary encompasses all of processes in which the system operates and defines external boundary flows. Step one should be performed following the identification of the customers’ needs.

2. Identify the process boundaries, modeled as events, which define the operational aspects of the system. All of the flows required by the process originate from within the environment. At the environmental and process level, the system being designed exists as a flow between events within the process.

3. Identify the physical boundaries of the system itself. Physical boundaries define the black box functional model of the system being designed, and while the system has been explicitly represented as a flow in both the environment and the process (defined in Steps 1 and 2), it should not be represented as such in the context of the system’s functional model.

4. Decompose the process into event and configuration models and the system into functional models until the desired level of fidelity has been reached. The final combinations of decomposed models generated within FDF may be process-specific, function-specific or mixed depending on where designers decide to focus attention.

The generation of these multi-level, function-based models in a static modeling application can be difficult and time consuming, thus development of FDF considered the necessity of computational implementation and the future integration with existing conceptual design tools. Its architecture was developed to simplify this computational implementation. Both process and functional models, at varying levels of fidelity, have the same hierarchical structure and contain the same basic objects (functions and flows), and as a result can use the same parent data object in a computational implementation. Additionally, the models are set up to allow “linking” from a single function in one model to a more detailed higher-fidelity multi-function model along with mapping between the environment, functional models and process models. Also, the framework has no predefined limit on number of levels of fidelity allowing this linking process to continue as far as necessary to capture the desired detail of the system being modeled. To capture these multi-level models, FunctionCAD is designed to model systems that vary in complexity from a single function (black box model) to hundreds of functions existing as a part of multiple product processes in multiple operational environments.

### 3.2 Related Design Tools

These models, generated following FDF within FunctionCAD must be represented computationally to allow the integration with a number of priorly developed design tools based on functional modeling with Functional Basis. Tools whose integration is currently anticipated includes:

• **Knowledge-based Morphological Analysis**: Function-flow pairs identified from customer needs may be paired to solution principles stored in a design repository. A morphological chart is returned containing a user defined number of components known to solve each of the function-flow pairs queried [33, 34].

• **Automated Concept Generation**: MEMIC or Morphological Evaluation Machine and Interaction Conceptualizer requires the user to input a functional model in matrix form describing adjacency between function-flow pairs. The input undergoes a series of matrix multiplications mapping solution to functionality and filtering out component-to-component connections that are not possible based on repository data [35, 36].

• **Configuration Flow Graphs**: Grammar rules developed from repository data convert function structures into a graph of connected components. The graph, referred to as a Configuration Flow Graph or CFG, is based on nodes and arcs to represent function and flow. The openness of the graph-based representation allows for automated concept generation, which is not constrained by singular function-to-component mappings [37].

• **Three-Dimensional Concept Visualization**: Lightweight, three-dimensional VRML (Virtual Reality Modeling Language) models are utilized to represent physical geometry of components paired to the functional model during function-based concept generation. Each element in the three-dimensional model can be manipulated (scaled, shifted and transformed) to generate a full concept visualization [38].

• **Function and Risk Analysis**: Potential failures and associated risk can be identified during conceptual design based on functionality required in the final design. The Function Failure Design Method (FFDM) identifies potential failure modes during conceptual design by starting with a function structure and mapping failures to functions from historical and expert elicited failure databases [39-41]. The current implementation of FFDM is based on the Design Repository [42] and allows a designer to query the repository for known failure modes related to function and flow pairs. Potential risk associated with functionality may also be determined by utilizing Risk in Early Design (RED) where the potential for the failure of a function is calculated and plotted on a Fever chart [43].

• **Mathematical Simulation**: A function structure is utilized as a framework to divide a simulation task, whereby the complete behavior (system dynamics) is modeled by isolating functional elements and identifying the flow variables passing through the boundaries of each functional element. Equations relating to flow variables and functional elements are identified and utilized to mathematically model the complete system behavior [44].
The integration of these automated and assistive design tools within FunctionCAD will make possible the application of function as a framework to a more iterative conceptual design process. Through an integrated output with FunctionCAD, these design tools will be capable of directly querying a database to provide immediate feedback on potential solution principles, components, known failure modes, and potential risk associated with functionality. A designer can in turn utilize this data to update the functional model adjusting for potential problems and required sub-functions related to specific component usage. The first step to generating this iterative function-based design framework is the development of an extensible, purpose specific modeling tool—FunctionCAD.

4 FunctionCAD Technical Details

FunctionCAD is a modular, open source application designed to create functional models based on FDF for product development. FunctionCAD was written with C++ and uses Qt4 by Nokia (previously Trolltech) for a cross-platform Graphical User Interface (GUI). Files created from FunctionCAD are based on the open outline format OPML (Outline Processor Markup Language) to provide compatibility with existing conceptual design tools. It is officially supported on MacOSX and Windows, and the source code is available and known to work on Linux. FunctionCAD builds are available at http://www.DesignEngineeringLab.org/FunctionCAD.

FunctionCAD is divided into two major components, libFCAD for handling the internal representation of the model and FunctionCAD for providing the GUI for model manipulation. (To avoid confusion with FunctionCAD the application, FunctionCAD the GUI will simply be referred to as the GUI through the remainder of this paper.) These two major components of FunctionCAD are illustrated in Fig. 1. At the same hierarchical level as the two primary constituents of FunctionCAD are plugins. Plugins, however, are user developed and user managed to provide extensibility, thus are not considered a part of FunctionCAD’s primary components.

4.1 libFCAD Technical Details

The shared library, libFCAD, is developed to manage each model developed within FunctionCAD as well as to support the saving and loading of FunctionCAD files and the importation and management of plugins. libFCAD provides the common framework for FunctionCAD and other FunctionCAD-based programs for use when creating and modifying models. The library is licensed under the Lesser General Public License (LGPL), making it open source, but still allowing for plugins or tools based on it to be released commercially. Each of the following sub-sections describe the major systems which comprise libFCAD.

4.1.1 libFCAD::Model

The most important system in libFCAD is libFCAD::Model. libFCAD::Model manages the entire structure of each user model by allowing user creation and editing of models, management of model styles, and utilization of plugins. Models in libFCAD are represented by two major types of elements termed nodes and arcs—taken from graph theory. There are three types of nodes, taken from FDF, functions, processes and environments, which are displayed as blocks and represent the boundaries of each of the model elements. Nodes can contain other nodes creating parent-child relationships between nodes to represent the hierarchical design required by FDF; for instance, a system modeled functionally is a part of a larger environment. Figure 2 represents this hierarchical relationship with the function block named Transfer Material containing other function blocks named Import Material, Channel Material and Guide Material. All of these function blocks are also a part of the larger node—Environment—providing three hierarchical layers. Arcs connect each of the blocks representing flows in the model. An arc can connect on either side (inside and outside) of the different boundaries of functions, processes and environments to allow connections between model elements at the same hierarchical level as well as connections between parent and children. To define and constrain arcs and nodes, both have a set of properties defining how a specific object looks and behaves. Property control, displayed by the style system, is discussed more in the following section libFCAD::Property.

FIGURE 1: DATA FLOW CHART OF THE FUNCTIONCAD SOFTWARE.

FIGURE 2: EXAMPLE HIERARCHICAL MODEL GENERATED IN FUNCTIONCAD.
To further specify the properties of arcs and nodes, a style system is controlled by libFCAD::Model. Colors and appearance properties are all set by the style system. Both global and local styles can be adjusted to allow users to modify and create different styles for the same element type (such as two functions each having different colors or nonstandard line-types for arrows) twice contained in a model.

Saving and loading of files is also controlled by libFCAD::Model. FunctionCAD’s save format follows the ZIP file format for data compression, and files are given the “.fcad” file extension. The use of the ZIP file format provides expandability. The ZIP file contains several different files allowing for model information to be distributed across multiple files yet bundled to provide easy transfer between computers, and as other files or formats are required for concept generation tools, the additional information can be included within the ZIP file.

There are currently three data files are stored within the ZIP file: one generated as a text file and two others following the Extensible Markup Language (XML) OPML format. Following the OPML and text file standards allows common tools and libraries to access the information directly. The current file specification for each of the three files include: (1) arc_matrix.txt, (2) function_outline.opml, and (3) properties.opml. The primary save file, containing all of the object data for a model, is properties.opml. Properties.opml stores all the data for all objects in a given model required to load a model into the editor. Function_outline.opml provides a simplified version of properties.opml eliminating everything except for the basic structure of the model. The file more closely resembles a function tree, and provides a format that could more easily be parsed into a systems engineering application designed for requirements mapping. The final file, arc_matrix.txt, contains an augmented adjacency matrix in a standard Comma-Separated Values (CSV) format. The augmented adjacent matrix contains function-flow adjacency information as well as additional flow information to capture carrier flows [46] and parent-child relationships.

4.1.2 libFCAD::Property

The libFCAD::Property manages the property system used to define all model element attributes. Element properties allow separation between the functional information and the attribute information specific to FunctionCAD’s visualization. Examples of attributes include: element position, color and type. Each element’s attributes can be viewed and changed via the attribute editor window shown in Fig. 3. Attributes for each element are saved as a part of the properties.opml and can easily be parsed out to allow other tools to only use the attributes they care about while safely ignoring the rest. Changes to model element properties are passed to the style system in libFCAD::MODEL for output on a functional model in the GUI.

4.1.3 libFCAD::Event

To manage user-based events, the libFCAD::Event system is implemented. The libFCAD::Event system monitors for all user inputs such as adding a node, clicking an arc or moving these elements. When an input is detected by either the GUI or libFCAD, libFCAD::Event places the event in an event manager and notifies either the GUI or libFCAD::Plugin that an action is requested.

4.1.4 libFCAD::Plugin

The libFCAD::Plugin defines the interface for plugins to use libFCAD. In FunctionCAD, all plugins are shared libraries (DLLs on Windows, SOs on Linux, DYLIBs on MacOSX) which can be dynamically loaded into models. This on-the-fly plugin interface allows the capabilities of FunctionCAD to be extended during model development.

A plugin contains a C++ object which can interact with the main library (libFCAD) through a plugin interface. The plugin interface allows the plugin to access the current model, the user interface, and most of the libFCAD’s internals. A user interface has been built for user management of plugins, and when enabled, plugins can setup an interface for the user, add properties to the model, modify model configurations, add additional model elements, etc. Once the plugin is enabled, user generated events are passed to their respective plugin via libFCAD::Event.

A major goal with the plugin system is to have flexibility such that future unanticipated tools can be interfaced with the software package. Currently, plugins have been written to log all events passing through libFCAD::Event, to manually edit attributes, to enable the use of the Functional Basis lexicon, and to export functional models in other model formats. A number of plugins are currently in the planning stages including a plugin to interact with the Design Repository [42] for archival of model information and a plugin to generate assembly models detailing component-based supporting functions.

Currently, the Functional Basis is the most complex plugin that can be enabled in FunctionCAD. By default, FunctionCAD only knows the basic rules for how to generate nodes and how to connect nodes via arcs. (This in fact is actually performed by the “default” plugin, which is hidden and cannot be disabled by the standard user.) When generating a functional model, however, it is not useful if the elements cannot not be labeled as function or flow, so to allow naming,
the Functional Basis plugin is developed. When the plugin is enabled, two custom types are created for the property system to store the function and flow information: (1) FunctionalBasis::Function for functions and (2) FunctionalBasis::Flow for flows. Nodes and arcs are then each provided with an extra attribute to store their name: (1) functionalbasis.function for nodes and (2) functionalbasis.flow for flows. Callback functions and editors are setup so that the captioning and editing of an element can be controlled by the plugin.

4.1.5 libFCAD::Gui

The libFCAD::Gui provides the interface for plugins to display graphics. With this library, a plugin can create windows, text boxes, buttons, and other user elements for the plugin. The library is implemented by the GUI itself (such as FunctionCAD) allowing plugins to work within any program based on libFCAD, thus it is not required for the plugin to know anything about the host program (whether it is using Qt4, GTK, a simple console application, or any variety of other available toolkits). Host programs can be based on the same back-end library (libFCAD), and can manipulate models in ways not anticipated with the initial GUI (such as a simplified GUI for teaching functional modeling). This flexibility allows plugins to work with models in a consistent way across multiple applications.

4.2 FunctionCAD (Qt4 GUI)

The GUI is the interface that users directly see, and currently, it is the primary user of libFCAD. A screenshot of the default GUI is shown as Fig. 4. Qt4 is used to provide a cross-platform GUI library that allows for builds that can be run on multiple platforms including Windows and MacOSX. The GUI creates Qt4 widgets using the QGraphicsView subsystem, and implements all of libFCAD::Gui for plugins to use. It provides easy editing and viewing of models, along with direct access to plugins through a plugin manager. When plugins are available (placed in FunctionCAD’s plugin directory), they are automatically detected to allow easily selection for use in the model. The following section provides a walk-through of FunctionCAD's GUI as well as general model creation.

5 MODEL CREATION IN FUNCTIONCAD

Upon starting FunctionCAD, the user is presented with an empty model and a default tool pallet. The empty model is generic, specifying only an empty Environment block. The default tool pallet has just six elements for model construction: (1) environment block, (2) process block and (3) function block—corresponding to the three boundary types defined in FDF—and three flow types of (4) material (represented with a bold arrow), (5) energy (single width arrow) and (6) signal (dashed arrow). The screenshot provided in Fig. 3 shows the initial configuration and environment model provided to a user upon opening FunctionCAD. The configuration of FunctionCAD is flexible, and the floating window can be docked on any side of the GUI window as shown in Fig. 5.

![Figure 4: Default Modeling Space in FunctionCAD.](image)

To generate a model within FunctionCAD, the user selects the block type desired (environment, process or function) and drags the block into the work space. Any block type can be the child or, conversely, the parent of another block type; the only exception to this rule is that the highest level block is constrained to be an environment. To visually represent this model hierarchy, blocks are simply placed within their desired parent. Since this visual representation, can however, become cluttered when a number of children are present, the option to visually “close” and “open” blocks is provided. When a block is “open,” its name is displayed top-center and its children are visible; however, when a block is “closed” its name is displayed at block center, and its functional decomposition is hidden. Double clicking a block will toggle the block between its “open” or “closed” states. Figure 6 shows examples of process blocks in the “open” (left-most process block) and “closed” (right-most process block) states.

![Figure 6: FunctionCAD Model with Process Blocks in the “Open” (Left Most Process Block) and “Closed” (Right-Most Process Block) States.](image)
To connect functions, processes and environments, flows are first dragged from the default pallet into an empty portion of the parent block containing the model elements which are to be connected. Once the flow is placed, its tip and tail are dragged separately to the blocks which should be connected. Flows can connect all blocks within a single level of a parent and can connect children blocks with their immediate parent block. Figure 7 adds flows to the model shown in Fig. 6.

![Figure 7: FunctionCAD model with material and energy flows added where flows originate from the environment, move into the processes, and are utilized at the functional level.]

To increase the versatility of FunctionCAD, a plugin interface is utilized. To load a plugin, a user would select Plugins > Manage from the menu bar to bring up the plugin manager. The plugin manager, shown in Fig. 8, displays all of the plugins, which have been added to FunctionCAD’s plugin directory. Enabling a plugin is performed by selecting the enabled checkbox. Selecting the default checkbox will add the plugin to FunctionCAD’s default startup configuration.

![Figure 8: FunctionCAD plugin manager interface.]

In Fig. 8, the Functional Basis plugin has been enabled for the modeling lexicon. Enabling the Functional Basis plugin allows each function and flow to be named, and loads a dictionary for quick access to Functional Basis terminology. To name an element, a user right clicks on any element in the model. An editor window appears allowing the user to either type the name in the type field or to select the desired term from the pull down menu. A screenshot of this entry box is provided in Fig. 9.

![Figure 9: Functional Basis editor window for the naming of model elements.]

To use the Functional Basis dictionary, a user would select Plugins > Functional Basis > Dictionary from the menu bar. A dialog box appears, which allows hyperlink-based navigation of function and flow terms. Clicking on a term, in either the function or flow hierarchy, displays its definition at the top of the dialog box. Figure 10 provides a screenshot of this dictionary dialog box for the Functional Basis. The term, Guide, has been selected, shown in bold without an underline, and its definition, along with a usage example and correspondents, is shown at the top of the dialog box above the hierarchy.

![Figure 10: Dictionary dialog box for the Functional Basis lexicon.]

Each of the function elements in the model, built through the previous figures is now updated with the Functional Basis lexicon. The resultant functional model, shown as a screenshot in Fig. 11, is spread between two process blocks and is a part of the larger environment. In the first block, two materials are coupled, and in the following process, they are separated.

![Figure 11: Example of model with Flow-CAD - FunctionCAD.]
To save a model, the user would either use the File > Save or the File > Save As depending on whether or not he or she wants to change the model name. The save window interface follows the standards of the operating system—either MacOSX or Windows. And finally, once a model is completed, it may be exported from the FunctionCAD software as an image file. There are two export options available. The first, available by going to the File > Export menu item, exports the entire model; the second, available by going to the File > ExportVisible menu item, exports only the portion of the functional model currently visible in the working area of the screen. The supported file formats include: PNG, TIFF and JPEG.

6 DISCUSSION

Once a digital representation of a functional model is generated in FunctionCAD, the data can be ported into other applications for concept generation [33-37] and visualization [38], failure [39-41] and risk analysis [43], mathematical simulation [44] and design archival [42] or the model can be used as is. This integration, shown in Fig. 12, places FunctionCAD at the center of the conceptual design process, where FunctionCAD ties together each of these design tools such that they can be utilized seamlessly through the conceptual design process.

A functional model developed during the conceptual design process is, however, just a single piece of a larger design process. This process, shown in Fig. 13, begins with the designer first understanding a product opportunity. Then looking to fill the opportunity by considering options and conceptualizing a new design. This conceptual design is then detailed and finalized before its implementation. Each piece in this design process builds upon subsequent steps and cannot proceed without the next, as demonstrated with the interconnected blocks in Fig. 13.
with an accessible plugin interface for management of plugin-based extensions for both existent and nonexistent conceptual design tools. This represents the first step at the creation of the discussed unified approach where all of the conceptual design tools are integrated into a single software package.

The future work for the FunctionCAD software includes the implementation of an upgraded plugin interface to support multiple coding languages, development of plugins to realize the FunctionCAD-based conceptual design process, and better GUI support for scalable functional models. As these features are developed, FunctionCAD will be tested during student design projects to quantify its effectiveness at facilitating a more iterative design process, and if a more iterative design process allows for quicker convergence on a good design. Also, tests will focus on if the quality of a functional model improves when a designer no longer has to think about the specific structural elements of a functional model but instead only is required to think about the model content and the conceptual design questions posed.

ACKNOWLEDGEMENTS

Support for this research has been provided by General Motors Corporation. Any opinions, findings and conclusions or recommendations presented in this paper are those of the authors and do not necessarily reflect the views of General Motors Corporation.

REFERENCES


