- Regulate functions require an analog control signal to adjust flow quantity.

6. Use the function *sense* to detect or measure a flow (*material, signal, energy*).
   - Sense functions require an input of the flow of interest to output a status signal representing data collected.
7. Use the function *indicate* to provide system status to the user.
   - Indicate functions end flow paths; thus status flows exiting an indicate function block leave the system and do not connect to other function blocks.
   - Indicate functions do not receive a carrier flow.
8. Use the function *process* to execute a series of operations to extract conditional information on a flow *signal*.
   - Either control or status flows can enter process functions; however, respective entering flows must also exit.
   - To change a status function to a system-usable control signal, a convert function must be implemented.
9. Use the function *convert* to perform the conscience act of changing a signal flows type.
   - A status flow input to the convert function should output as a system-usable control signal, or a control flow input to the convert function should output as a status flow.
10. Use the function *import* to bring a flow *control signal* from outside of the system boundary to the inside of the system boundary.
    - Flow arrows should be drawn into an input function block to represent flow into the system.
11. Use the function *export* to send a flow *signal* outside of the system boundary.
    - Flow arrows should be drawn leaving the export function to represent control flowing from the system.
12. Use the function *transfer* to move a flow *signal* through a system.
    - Either control or status flows can enter transfer functions; however, respective entering flows must also exit.

4.2 Syntax for signal flows

A grammar has been enumerated for both energy and material flows [12, 13]; however, signals, while having inherent Functional Basis guidelines, have no established usage grammar. The previously presented signal flow morphology rules are used to build syntax for signal flows. The syntax rules manifest themselves as functional modeling templates that can be inserted into a functional model, aiding the manual or automatic assembly of functional models, thus increasing the accuracy of product and design representation. Each syntax rule is explained and visually represented in Figure 2. Taken together, the sets of morphology and syntax rules constitute the signal grammar for the functional modeling language.

Actuator: An actuator is a discrete control device used to turn on or off another flow. In conceptual design, if it is known that a flow will be toggled, an actuator should be implemented. To functionally build an actuator, the function term *actuate* must be used in conjunction with rules 3 and 4. A control signal, its carrier, and the flow to be toggled should be imported with *import* function-flow blocks. Then, following rule 12, a *transfer control* function-flow block is applied to route the control signal to an *actuate flow* function-flow block. Optionally the actuator can output its status. To indicate status, rule 9 is followed to convert the control signal to a status signal and rule 7 is followed to indicate the status through a *indicate status* function-flow block.

Regulator: A regulator is an analog control device to adjust a flow in variable manner. When it is known in conceptual design that a flow is to be adjusted in a variable manner, a regulator should be implemented. To functionally build a regulator, the term *regulate* must be used in conjunction with rules 3 and 5. To implement a regulator, a control signal, its carrier, and the flow to be adjusted should be imported with an *import* function-flow block. The control signal and its carrier are routed to the *regulate flow* function-flow block via a *transfer control* function-flow block following rule 12. If the regulator indicates its status, rule 9 is followed to convert the control signal to a status with a *convert control to status* function-flow block, rule 7 is followed to indicate the final status.

Sensor: A sensor is a device used to detect or measure a flow and then output a signal representing collected information. Sensors would be used during conceptual design if a designer realizes that a design must ascertain information about itself or its surroundings. To functionally build

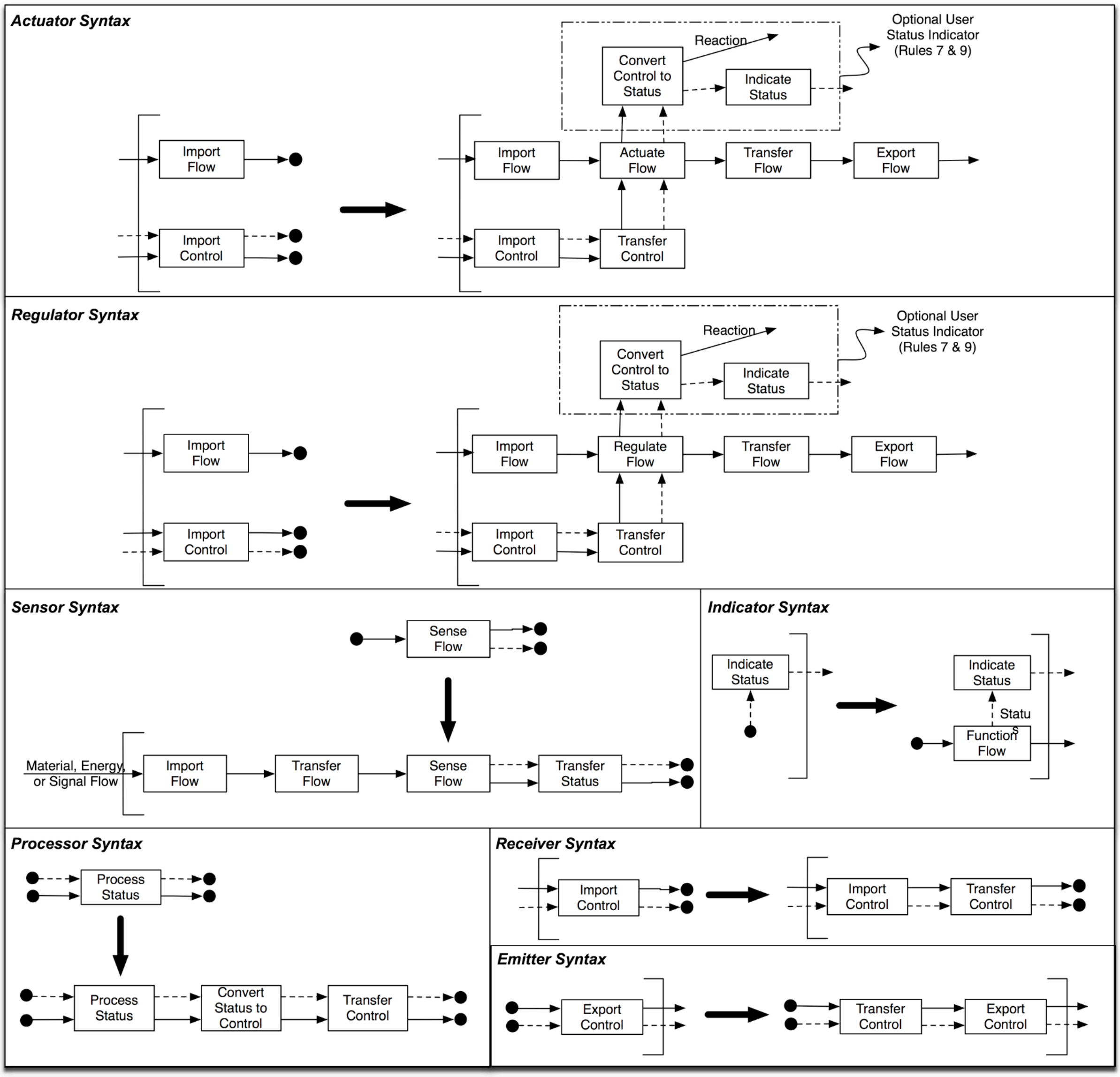

Figure 2. Seven signal-based syntax rules (actuator, regulator, sensor, processor, indicator, receiver, and emitter) have been extracted from the Functional Basis. Black nodes represent connection points, square brackets represent system boundary, and black arrows represent required function/flow insertion.

a sensor, the term *sense* must be used in conjunction with rules 3 and 6. To implement a sensor, transfer a flow (material, energy, or signal) to a *sense flow* function-flow block. The *sense flow* function-flow block outputs primary status flow and its respective carrier flow, which can then be transferred with a *transfer status* function-flow block by rule 12. The final destination of the status flow and carrier can be the system, the user, or both.

The *sense* function can be further detailed at the tertiary level by the reconciled Functional Basis with its tertiary functions *detect* and *measure*. The general sensor functional model can be modified to reflect the increased detail of tertiary terms by using one of the two tertiary terms in place of the *sense* function.

Processor: A processor is any device that analyzes a status signal obtained from a sensor that has ascertained information, either internal or external to the system. Following signal analysis, the processor sends control information to system elements. A processor might be used during conceptual design if a designer knows that the product will need to analyze the state on a series of conditions and make decisions based on the analysis. To functionally build a processor, the term *process* must be used in conjunction with rules 3 and 8. To use a processor, run a primary status flow and a carrier flow into a *process status* function-flow box. Following rule 9, to get a system usable control signal, connect the *process status* function-flow box to a *convert status to control* function-flow box with another primary status flow and carrier flow pair. Then connect the *convert status to control* to a *transfer control* function-flow box with a primary control signal and carrier flow. By the application of rule 12, the control and its carrier flow is routed via a *transfer control* function-flow box to the controlled system elements.

Indicator: An indicator is any device with the goal of providing vital system information to the user. A designer might use an indicator during conceptual design when it is known that some form of feedback is required from the system. To functionally build an indicator, the term *indicate* must be used in conjunction with rule 7. To implement an indicator, run a status flow from the function-flow block from which system information should be obtained. An indicator is the exception to the primary/carrier rule (rule 3) since it can be as simple as the operation of the system or complex as a series of components providing full diagnostics on system behaviors; in either case, however, an indicator is not required to send the signal outside of the system boundary. An *indicate status* function-flow block ends a flow path (rule 7), thus the status flow that exits an *indicate status* should not enter another function block.

As with the *sense* function, the *indicate* function can be further detailed at the tertiary level by the reconciled Functional Basis with its tertiary functions *track* and *display*. The general indicator functional model can be modified to reflect the increased detail of tertiary terms by using one of the two tertiary terms in place of the *indicate* function.

Receiver: A receiver is used to bring a control signal into the system. To functionally build a receiver, the term *import* must be use in conjunction with rules 3, 10 and 12. To implement a receiver, a new control signal flow and its respective carrier must be imported into the system and thus cross the system boundary. The primary and carrier flows are then routed into the overall system by tying the *import control* function-flow block to a *transfer control* block.

Emitter: An emitter is used to send a control signal from the system. To functionally build an emitter, the term *export* must be used in conjunction with rules 3, 11 and 12. To implement an emitter, run a control signal flow and its carrier flow through a *transfer control* function-flow block. Then tie the control signal and its carrier to an *export control* function-flow block. From the *export control* function-flow block, draw an exiting control signal flow and its carrier to represent them leaving the system boundary.

5 APPLICATION

As an application of the signal usage grammar defining the use of signals and their associated functions, consider an automatic garage door opener. The automatic garage door opener has been modeled following the functional modeling procedure outlined in *Development of a Functional Basis for Design* by Stone, *et al.* [10]. The functional modeling procedure, provided in abbreviated form below, is a three-step method outlining the application of functional modeling techniques to modern product design. The procedure, while intended for the design of a product, still provides a useful set of guidelines for product dissection and reverse engineering [10].

1. Generate black box model
2. Create function chains for each flow
3. Aggregate function chains into a functional model

The first step to developing a functional model is to develop a black box model. At the black box level, the functionality of an automatic garage door opener is to *Open Door*. Inputs include a garage door, human, obstacle, human energy, electrical energy, and wired and wireless on/off control signals.

The second step to generating a sub-functional model is to generate function chains for each input flow. Each sub-function chain should consider the changes and operations that occur to each flow. Considering only the signal input flows, function chains are generated for the obstacle and garage door detection and the wireless and wired on/off control signals. The two function chains developed to detect obstacles and the garage door are solid detectors. The solid detectors, shown in Figure 3, are built by applying the sensor and a processor syntax rules. The sensor rule is required to represent the detection of a solid, and outputs a status indicating the system state into a processor. The

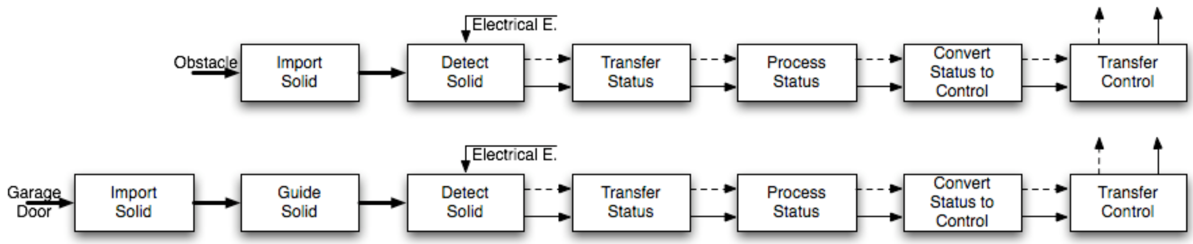

Figure 3. Obstacle and garage door detection function chains

processor determines what to output as a control based on the system status. The wireless and wired on/off switch function chains, shown in Figure 4, are developed following the actuator rule. Both function chains require a control signal carried by the human energy. The control signal is transferred to an *actuate electrical energy* block to turn on or off the garage door opener.

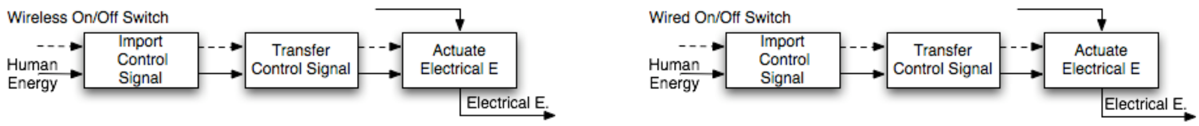

Figure 4. Wireless and wired on/off switch function chains

The third step is to aggregate each of the function chains into an overall functional model of the system, which is provided in Figure 5. The obstacle detector is a safety feature of the garage door, thus its control signal routes into the master *actuate electrical energy* function-flow block controlling the overall operation of the garage door. The garage door detection chain is used to determine when the garage door is up or down and again routes into the master *actuate electrical energy* function-flow block. The wired on/off switch is a master control, and thus routes into the *actuate electrical energy*. Finally, the wireless on/off switch is different: its control signal routes into an *actuate electrical energy* that is part of a separate transmitter that sends a control signal out to the garage door providing a remote actuation feature.

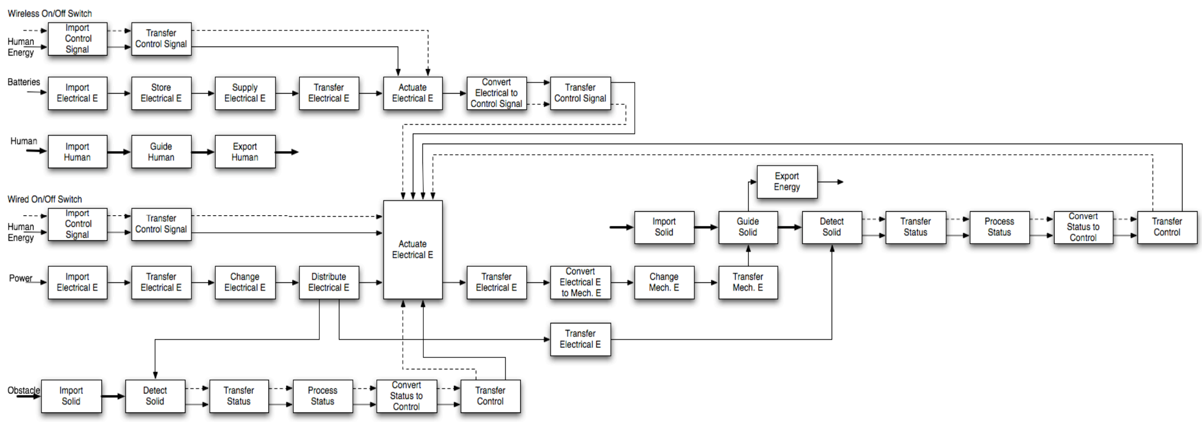

Figure 5. Aggregated functional model of an automatic garage door opener

Each of the signal based function chains identified in the garage door and provided Figures 3 and 4 are built from the templates provided through syntax following the morphology rules. The combined grammar aids in the identification of necessary flows for desired functionalities, which not only aids in the development of independent function-flow chains, but also aids in aggregation of the overall model. Flows fit together better in the final functional model and are more clearly represented with the inclusion of carrier flow information, which might otherwise be excluded from the functional model.

6 CONCLUSIONS

The Functional Basis Modeling Language is a start to evolve the Functional Basis toward a formal language for functional modeling. Analogies are drawn between FBML and formal languages, and a grammar consisting of syntax and morphology is presented. The signal grammar for modeling signal flows allows for more consistent functional modeling among designers. This uniformity helps to ensure understanding and helps to maintain consistent archival of design information. The signal usage grammar provides a framework for the application of Functional Basis terms, which is demonstrated on an example electromechanical product. A structured modeling language with a clear morphology, syntax, lexicon, graphology, and semantics aids in the automated and manual functional model generation techniques.

Usage grammar addresses the consistency of model structure, and when paired with a consistent taxonomy like the Functional Basis, increases the consistency of functional modeling. Consistency of models improves model-to-model communication among designers and helps to develop the synergy needed to develop an engineering solution for automated design where components must communicate and function across domains.

Future work will focus on the further evolution of the Functional Basis into the Functional Basis Modeling Language and on verification of the validity of the proposed signal grammar. Refinement of the grammar and the development of an analogous syntax and morphology for energy and material flows will be developed. Once grammar consisting of syntax and morphology has been developed for the entire Functional Basis, experiments will be performed with students who are learning and familiar with functional modeling utilizing the Functional Basis to verify the uniformity among models when the refined grammar is applied.

REFERENCES

[1] Ulrich, K.T. and Eppinger, S.D. *Product Design and Development*. (McGraw-Hill/Irwin, Boston, MA, 2004).
[2] Cutherell, D. Chapter 16: Product Architecture. In Rosenau Jr., M., ed. *The PDMA Handbook of New Product Development* (Wiley and Sons, 1996).
[3] Fenves, S. A Core Product Model for Representing Design Information. (National Institute of Standards and Technology, Gaithersburg, MD, 2001).
[4] Hundal, M. A Systematic Method for Developing Function Structures, Solutions and Concept Variants. *Mechanism and Machine Theory*, 1990, 25(3), 243-256.
[5] Miles, L. *Techniques of Value Analysis Engineering*. (McGraw-Hill, New York, 1972).
[6] Pahl, G. and Beitz, W. *Engineering Design: A Systematic Approach*. (Springer-Verlag, London, UK, 1996).
[7] Otto, K. and Wood, K. *Product Design: Techniques in Reverse Engineering, Systematic Design, and New Product Development*. (Prentice-Hall, New York, 2001).
[8] Dieter, G. *Engineering Design: A Materials and Processing Approach*. (McGraw-Hill, New York, 1991).
[9] Ullman, D.G. *The Mechanical Design Process 3rd Edition*. (McGraw-Hill, Inc., New York, 2002).
[10] Stone, R. and Wood, K. Development of a Functional Basis for Design. *Journal of Mechanical Design*, 2000, 122(4), 359-370.
[11] Hirtz, J., Stone, R., McAdams, D., Szykman, S. and Wood, K. A Functional Basis for Engineering Design: Reconciling and Evolving Previous Efforts. *Research in Engineering Design*, 2002, 13(2), 65-82.
[12] Sridharan, P. and Campbell, M.l. A Grammar for Functional Structures. *Design Engineering Technical Conferences and Computers and Information in Engineering Conference* (ASME, Salt Lake City, Utah, 2004).
[13] Sridharan, P. and Campbell, M.l. A study on the grammatical construction of function structures. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 2005, 19, 139-160.
[14] Nise, N.S. *Control Systems Engineering*. (John Wiley & Sons, 2004).
[15] Chen, D. and Törngren, M. A Systematic Approach for Identifying Operational

Relationships in Embedded Computer Control Systems. *EUROMICRO Conference* (IEEE Computer Society, Rennes, France, 2004).
[16] Chen, L., Jayaram, M. and Xi, J.F. A New Functional Representation Scheme for Conceptual Modeling of Mechatronic Systems. *Design Engineering Technical Conferences and Computer and Information in Engineering Conference* (ASME, Montreal, Canada, 2002).
[17] Jayaram, M., Chen, L. and Xi, J.F. Functional Modeling of Complex Mechatronic Systems. *Design Engineering Technical Conferences and Computer and Information in Engineering Conference* (ASME, Chicago, Il, 2003).
[18] Rajan, J.R. A Robust Functional Modeling Method in Product Design. pp. 73-135 (The University of Texas at Austin, Austin, 2002).
[19] Rajan, J.R., Stone, R.B. and Wood, K.L. Functional Modeling of Control Systems. *International Conference on Engineering Design*Stockholm, 2003).
[20] *The New Oxford American Dictionary*. (Oxford University Press, 2005).
[21] Little, A., Wood, K. and McAdams, D. Functional Analysis: A Fundamental Empirical Study for Reverse Engineering, Benchmarking and Redesign. *Proceedings of the 1997 Design Engineering Technical Conferences*Sacramento, CA, 1997).
[22] Millward, C.M. *A Biography of the English Language*. (Harcourt Brace College Publishers, Fort Worth, TX, 1996).
[23] Quirk, R., Greenbaum, S., Leech, G. and Svartvik, J. *A Comprehensive Grammar of the English Language*. (Longman, London, 1985).

Contact: Robert L. Nagel, Author
University of Missouri-Rolla
G-5 Interdisciplinary Engineering Building
1870 Miner Circle
Rolla, MO 65409-0201
USA
1.573.341.6064
rlnkc7@umr.edu
http://web.umr.edu/~rlnkc7