Goal-Function Tree Modeling for Systems Engineering and Fault Management

Stephen B. Johnson[1]
Jacobs ESSSA Group, Dependable System Technologies LLC, and University of Colorado, Colorado Springs

Jonathan T. Breckenridge[2]
Jacobs – ESSSA Group / Ducommun Incorporated, Miltec Systems, MSFC, Huntsville, AL, 35763, USA

Abstract: This paper describes a new representation that enables rigorous definition and decomposition of both nominal and off-nominal system goals and functions: the Goal-Function Tree (GFT). GFTs extend the concept and process of functional decomposition, utilizing state variables as a key mechanism to ensure physical and logical consistency and completeness of the decomposition of goals (requirements) and functions, and enabling full and complete traceabilitiy to the design. The GFT also provides for means to define and represent off-nominal goals and functions that are activated when the system's nominal goals are not met. The physical accuracy of the GFT, and its ability to represent both nominal and off-nominal goals enable the GFT to be used for various analyses of the system, including assessments of the completeness and traceability of system goals and functions, the coverage of fault management failure detections, and definition of system failure scenarios.

I. Introduction

SYSTEMS engineering (SE) as currently practiced has a number of significant shortcomings. Many current SE methods and practices, such as development of requirements and specification of interfaces, are primarily based on natural language. For example, requirements traces attempt to connect individual requirements from semi-structured lists to each other, but this is hampered by the fact that they link inherently ambiguous natural language statements using individual judgment to determine the appropriate connections between them. It is thus difficult if not impossible to determine if the requirements specified for the system are complete, or to determine the accuracy of the relationships between them. Natural language ambiguities also make it difficult to map from requirements to functions, and functions to designs. Another problem with current SE is that to the extent it recognizes off-nominal perspectives at all; it poorly integrates nominal to off-nominal functions, goals, and designs. This problem can be stated in fairly simple terms: for each nominal goal or requirement to be achieved (in this paper we will consider the words "goal", "requirement", and "objective" to be synonyms), there is the possibility that this goal will not be achieved. In this situation, which is usually called "failure", what should the system to do? If the system is to predict or detect failure and then perform some action to prevent or mitigate the failure, then the system now has new off-nominal goals and associated functions, which we will identify as "fault management" (FM). In turn, FM is the operational subset of "system health management" (SHM) which is the set of system capabilities put in place to ensure that the system meets its intended goals.

A number of techniques to improve systems engineering move away from natural language and instead use models, under the rubric of "Model-Based Systems Engineering" (MBSE). Other efforts are ongoing to develop SHM and FM as the "dark side" of SE.[1] This paper will describe a new model-based representation, known as the Goal-Function Tree (GFT), which provides significant improvements over current SE practice by integrating nominal and off-nominal perspectives, and providing rigor to the classical notion of the functional decomposition.

The GFT representation, as its name implies, is not merely a functional decomposition, because it inherently integrates goals (requirements) and functions together. The GFT representation and its corresponding development methodology provides a means to represent, decompose, and elaborate system goals and functions in a rigorous manner that connects directly to design through use of state variables, which also enable translation of natural

[1] President, Dependable System Technologies, LLC; Associate Research Professor, Department of Mechanical and Aerospace Engineering, University of Colorado, Colorado Springs; Analysis Lead, Mission and Fault Management, Space Launch System Program, National Aeronautics and Space Administration; AIAA member.

[2] Assistant Fault Management Analysis Lead, Mission and Fault Management, Space Launch System Program, Marshall Space Flight Center (MSFC), AIAA Member.

language requirements and goals into logical-physical state language. The state variable-based approach also provides the means to directly connect FM to the design, by specifying the range in which state variables must be controlled to achieve goals, and conversely, the failures that exist if system behavior go out of range. This in turn allows for the systems engineers and SHM/FM engineers to determine which state variables to monitor, and what action(s) to take should the system fail to achieve that goal. In sum, the GFT representation provides a unified approach to early-phase SE and FM development.

This representation and methodology has been successfully developed and implemented using Systems Modeling Language (SysML) on the National Aeronautics and Space Administration (NASA) Space Launch System (SLS) Program. It enabled early design trade studies of failure detection coverage to ensure detection of all failure scenarios that can threaten the crew. The representation maps directly both to FM algorithm identification and to failure scenario definitions needed for design analysis and testing. The GFT representation provided a basis for mapping of abort triggers into scenarios, both needed for initial and successful quantitative analyses of abort effectiveness (detection and response to crew-threatening events).

Section II explains the utility of tree-like, hierarchical models in systems engineering, whose value is typically taken as a given. This section also describes some earlier attempts to create success-space models. Section III describes the motivations, problems and issues that inspired the development of the GFT on the SLS Program, and further issues and problems with systems engineering that it is well-suited to address and improve over standard practice. The details of the GFT representation and the methodology to construct a GFT model for a given system are presented in Section IV. Section V describes how the GFT is and can be used for a variety of analytical purposes for SE, SHM, and FM.

II. Tree Models and Intentionality

Success-space, hierarchical models have been researched and occasionally used in systems engineering for several decades. Despite this, their utility and purposes have not been universally accepted as part of the standard systems engineering process. We will describe in this section several approaches to develop top-down hierarchical representations along with some of the uses of these representations. Lastly, to understand why hierarchical models are important, but have some significant limitations, we must explore what these hierarchical models actually represent, given that a system design is at best only partially hierarchical.

The most common form of a success-space decomposition is the traditional functional decomposition. In it, the systems engineer attempts to define the functions of the system, usually in natural language, and arrange these functions into a top-down, inverted tree structure with the root (or perhaps the trunk) at the top and the branches below. In general, the top function or functions represent the overall activity that the system is to perform. This top function is then decomposed into other functions that must be provided for the top level function to successfully occur, and so on as the tree is developed further into many levels. The typical use of this decomposition is to determine the functions and the corresponding requirements needed to ensure these functions are provided.

The tree structure is useful as a method to trace system functions and requirements in an attempt to ensure completeness. However, it is difficult to be sure that the functions and related requirements developed in this way are complete, mainly due to the problems of interpreting natural language. Requirement trace and function trace tools provide means to track the tree structure and traceability, but the inherent difficulty lies in the interpretation of natural language, so as to know whether the trace is technically legitimate. The trace tools merely enable traces to be tracked and searched once the modeler decides that one requirement traces to another. The validity of that decision is not addressed. Note that the typical functional decomposition is distinct but somewhat ambiguously related to trace of its corresponding requirements, with the normal assumption being that there should be a requirement associated with each function. In systems engineering today, functional decompositions are often given lip service, but in practice they are not used consistently, implying that they do not always provide enough value to merit standard and consistent implementation.

Hierarchical decompositions are also used, and perhaps more frequently and consistently, in failure space models usually described as “fault trees.” Unlike success trees, fault trees attempt to determine how the system may fail, starting from the top level functions that the system must achieve and specifying how the system might fail to perform its highest-level function. These failure types are then decomposed further into lower level failures, once again leading to an inverted tree structure. Eventually the fault tree can be developed all the way down to the failure modes as described in a typical failure modes and effects analysis (which itself is a matrix). In practice a tree developed all the way to the failure modes becomes overwhelmingly large and hence the trees are not typically developed to this level of detail. As with success space representations, the ways in which functions can fail are described in natural language, though the trees can be associated with logical and mathematical attributes to enable probabilistic and logical operations on the fault tree model. However, the validity of these logical and mathematical operations depends at least in part on whether the "failure functions" in fact have the assumed tree-like relationships, which in turn depends on proper interpretation of the natural language descriptors. Fault trees have a long track record of use and success, ensuring that they are used more consistently for the design of highly reliable systems than success space functional decomposition.

Significant research in success space hierarchical methods has focused on goals. A recent goal methodology is described in van Lamsweerde's book, *Requirements Engineering: From System Goals to UML Models to Software Specifications*.[2] As implied by the title of the book, van Lamsweerde views goals as being directly related to requirements. Van Lamsweerde believes that goal tree specification using "goal diagrams" is essential, and that it is effective to use formal models in Unified Modeling Language (UML) to represent these goals. Conversely to the fault tree structures that use OR gate structures for the basic logic of the fault tree, with AND gates to model redundancy, in success space van Lamsweerde uses AND gate structures, with OR-gate modifications for redundancy. Van Lamsweerde's work emphasizes the utility of goal tree structures to understand and model requirements for complex systems controlled by software, so as to rigorously specify software requirements and link them to other UML software design models. Van Lamsweerde correctly notes that tree structures for representation of goals is common in artificial intelligence, going back at least to (Nilsson 1971).[3] Its introduction into requirements engineering, according to van Lamsweerde, came with the work of (Dardenne et al., 1991)[4] and (Mylopoulos et al., 1992).[5]

This link to artificial intelligence is clear in the work of Kim and Modarres (1986),[6] who introduced a Goal Tree-Success Tree representation for use in operator advisory systems. This approach developed in the mid-1980s for nuclear power systems modeling, and was tied to expert system development, which in turn is part of the artificial intelligence research tradition. For Modarres, the goal tree portion represented the top-level goals, and the success tree accounted for lower level goals that needed to be accomplished for the higher level goals to be achieved. It too used an AND-gate structure as primary, with OR-gates for alternative methods to achieve a goal. In the advisory system application, achievement of success of lower level goals (the lower level "success trees") was required to achieve high-level goals. Modarres (1999)[7] continued to develop the GTST approach, creating "master logic diagrams" of top level goals as columns and "support functions" as rows to begin assessing how failures affected goals. This approach developed into a highly complex methodology and model. Its application to other systems seems to have been quite limited. However, it does point to the difficult but critical issue of mapping from goals and functions to design, which is a many-to-many complex problem.

Another goal-tree-related development is the state analysis and Mission Data System (MDS) under development at Jet Propulsion Laboratory (JPL). Ingham et al. (2005)[8] provides a description of this approach. The JPL work uses a state-based approach to develop an autonomous, on-board deep space computing architecture. This architecture employs a goal-tree-like structure to represent goals, for the same reasons that artificial intelligence applications use tree structured search spaces. Fragments of the tree can be shifted around based on changes to the vehicle configuration, representing changes to the mechanisms needed to operate system functionality. A separate set of models represents system constraints. As a state-based approach, the JPL representations utilize state variables as a key aspect of their models, which aim toward an operational, deep-space architecture.

Each of these approaches provide important clues as to the various attributes that can and/or should be modeled to provide benefits for systems engineering and fault management. Standard functional decompositions have a potentially useful relationship to problems of requirements completeness and traceability. Fault trees attempt to develop near-complete representations of failures in safety-critical systems, and quantitative methods and tools have been developed to further their utility. Formal modeling methods to represent goals have been developed to create more complete and higher quality requirements for software systems, and to support the development of expert systems in safety critical applications. When linked to state-based methods and artificial intelligence techniques, JPL's methods attempt to apply goal tree models to deep-space autonomous architectures. The GFT method described in this paper utilizes aspects of each of these techniques, but then integrates and applies them to systems engineering and fault management.

Unstated, but underlying all of these methods is a fundamental question: what do hierarchical structures, whether in success space or failure space, actually represent? After all, unless the designer is building an artificial tree, a tree structure does not match the actual system design. For systems in general, components link to each other in all kinds of ways, including feedback loops that do not divide cleanly into independent, separate branches. Put another way, what makes some goals and functions more important than others, such that other goals and functions merely support these "important" goals and functions?

What gives some goals and functions a "higher importance" than others is the intention of the designer or operator. Engineered systems exist to fulfill some purpose, whose intent is specified originally by the system designers, and later on by the system operators or users, who may decide to use the system differently from the designers. It is *intentionality* that creates hierarchy among the system's goals and functions. For example, the primary purpose of a launch vehicle such as the SLS is to transport a functioning payload of a certain volume and mass to a destination in space. This describes a primary set of goals, such as the specific destinations the launch vehicle is capable of providing to payloads of the relevant size and mass, and primary functions, which are the corresponding transport activities or operations the launch vehicle must perform to move the payload from Earth to its destination. All other goals and functions exist to support these primary goals and functions. In the actual design, the control of the vehicle's accelerations, velocities, positions, and attitudes are achieved by control loops, which are enabled by further open or closed loop control of power, computing, fluids and so on. In the design representation, all of these loops, components and structures are connected in complex ways that have little resemblance to a tree. *Intention* is what enables the flat, interconnected design to be fruitfully represented and analyzed as a hierarchy, even when the design is only partially hierarchical.

III. Motivations for the GFT

The GFT originated from concrete FM analytical and design problems that arose on NASA's Ares I launch vehicle project during its development from 2006-2011. One key problem was attempting to determine whether the abort detections (called "abort triggers") on Ares I actually detected all Ares I failures that threatened the crew, which was in the Orion crew capsule on top of Ares I. Another problem was determining the physical and logical relationship between the various abort conditions and triggers. Abort conditions are vehicle failure behaviors, which if they occur, would immediately or ultimately threaten the crew through their "downstream" failure effects. In most cases, failures produce a variety of effects, and in many cases, two or more abort conditions and triggers could be activated in a single failure scenario (one particular failure occurring, leading to many downstream failure effects). The design question was whether some abort triggers (detections) were superfluous, because the relevant failure effects (abort conditions) were already being detected by other abort triggers. In essence, the question is if there were gaps and/or overlaps in the detection coverage of crew-threatening launch vehicle failures. Lastly, were there any missing abort conditions? A common theme among the first three items (abort trigger detection coverage, physical and logical relationships of abort conditions, and potential missing abort conditions) is that they cannot be resolved using "bottom up" approaches.

On Ares I, candidate abort conditions were generated using engineering judgment and documented as a list containing the conditions and a variety of condition attributes. As the list was developed, a number of issues arose, which the FM team had great difficulty in addressing. One particularly troublesome issue was trying to determine what was meant by a "monitored condition". Some conditions were potentially monitored "at" the location of the abort condition, such as a Thrust Vector Control (TVC) Hardover abort condition, monitored by direct measurement of the gimbal angles. This condition was ultimately classified to be "non-monitored". This was because the gimbal angle position sensors were of insufficient criticality, meaning it would have been costly to upgrade them and add more sensors, and because failure effects of a TVC gimbal hardover would be visible and adequately monitored and detected by the Guidance, Navigation, and Control Abort Triggers: attitude error and attitude rate error in this case. These attitude and rate triggers were "causally downstream" from the gimbal position. A good case could be made that this meant the condition was actually monitored, though not directly.

This case was only one of many examples showing the existence of a physical, causal relationship between abort conditions, but in list form it was difficult to determine these relationships in a consistent and complete way. If there was more than one way to monitor an abort condition, how does one identify the several possible abort triggers that could monitor the condition, and determine which of these triggers should be implemented? If more than one trigger was implemented, is unneeded system complexity being created, adding to cost and perhaps decreasing reliability? Does monitoring mean "directly monitored" as opposed to "indirectly monitored", and if so, can one define more precisely what "directly monitored" means? Finally, could the relevant analysis be done early enough in the design process to select and develop the relevant hardware and software in a cost efficient manner?

The same kinds of problems arose on the SLS program as it began after the cancellation of Ares I and Constellation. To address these problem, a new analytical method and approach was needed. This is the genesis of the GFT (which on SLS is called the Goal Tree / Success Tree (GTST)). It was hypothesized that a top-down approach in success space may be able to address these questions. Unlike failure space, in success space it does not matter how a system goal is not achieved, and hence the analyst can ignore the specific failure modes that can cause failure of the goal. It matters only that the goal is not being achieved, because if the goal is not achieved, then higher-level goals will also be compromised, ultimately compromising the system's ultimate purpose(s). Given that one of the major problems with SHM and FM is the impossibility of identifying all possible causes of failure, a method that focuses on what must be achieved, as opposed to how the system might fail, has distinct advantages. Finally, a top-down approach would allow assessment of these issues well before the design was complete, which would then enable potential design changes to occur before they became "locked in". This has been identified as a major issue for FM in general, which typically has been designed after the nominal system has been determined.[9] Instead of this "FM Band-Aid", it is desirable to have an analysis and design method that enables the FM design to be performed in parallel with the nominal design, not after.

IV. The GFT Representation, Construction, and Nuances

The GFT can be seen as a direct outgrowth of aspects of the theory of SHM. In that theory, as described in *System Health Management: with Aerospace Applications*, SHM's primary goal is to preserve the system's ability to function as intended. SHM theory precisely defines functions in the standard way identified in mathematical texts, in terms of the equation $y = f(x)$, where x is the input state vector, y is the output state vector, and f is the function, which is the transformation that maps the input state into the output state. Functions are allocated to mechanisms (which can be hardware, software, or humans), such that the function f can be implemented in several possible ways. The core SHM principle of function preservation simply states that SHM exists to ensure the correct transformation of an input state vector x to the appropriate output state vector y. In turn, proper transformation ensures that the goals associated with the function are preserved as long as the input state vectors are within their nominal ranges.

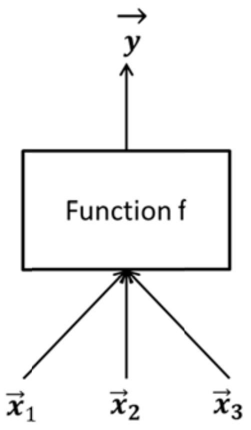
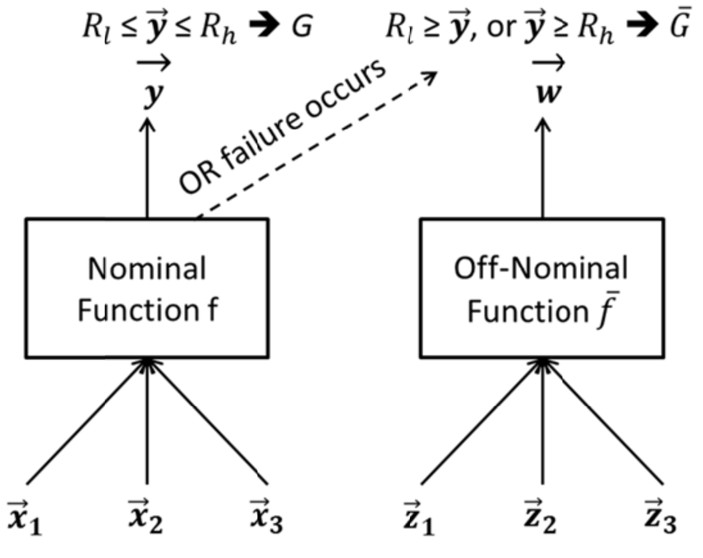


Figure 1: Function with Input and Output State Vectors

Alternatively, SHM can be defined as the system's set of capabilities that preserve the system's ability to achieve goals. How then are goals related to functions? Goals are defined as controlling the output state vector y within the intended range. Success of any function (its goal) is simply that its output remains within the intended range. That is, $R_l \leq y \leq R_h$, where R_l is the lowest value of the intended range, and R_h is the highest value of the intended range, where these intended ranges can vary over time. The output state vector y can be constructed from a variety of individual state variables, which implies that the upper and lower ranges are also vectors, with same number of variables as the output state vector y. Figure 1 shows a simple function block with the relevant input and output state variables. A goal is simply the control of the output state vector y within "success limits." The connection to off-nominal representation is simply to note that failure exists if the values of output state vector y stray outside of intended limits.

In the GFT, the state variables are the characteristics that must be controlled and transformed, of the entities being controlled. For example, for a launch vehicle, temperature, pressure, and mass (flow rate) are the state variables whose range must be controlled, of the liquid or solid propellants that are being used to provide thrust. The thrust itself is the state variable being controlled as the output of the rocket engines; the thrust is an attribute of the mass of propellants that are being expelled at a certain velocity. In the initial development of the GFT, these entities (the items whose attributes are the state variables) were not explicitly identified in the model, and in the examples to be shown in this paper, they will not be shown. However, in future updates to the GFT representation, these will be fully incorporated into the model.

For each and every goal, there is the possibility that the goal will not be achieved. If the system must perform some off-nominal action when the goal is not achieved, then this implies the existence of a new off-nominal goal that will be activated when the nominal goal is not achieved, and a corresponding off-nominal function to detect that the output state variable y is outside its nominal range and to take some action. Figure 2 shows the



$R_l \leq \vec{y} \leq R_h \Rightarrow G$
$R_l \geq \vec{y}$, or $\vec{y} \geq R_h \Rightarrow \bar{G}$
$\vec{y}$
OR failure occurs
$\vec{w}$
Nominal Function f
Off-Nominal Function $\bar{f}$
$\vec{x}_1$
$\vec{x}_2$
$\vec{x}_3$
$\vec{z}_1$
$\vec{z}_2$
$\vec{z}_3$

Figure 2: Off-Nominal Goal and Function if Nominal Goal is Not Achieved

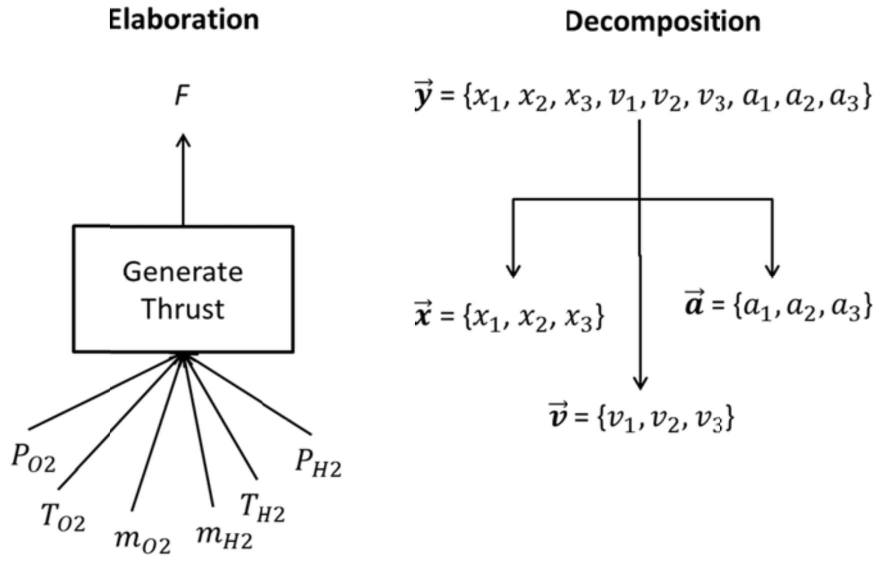
existence of an off-nominal goal and associated off-nominal function that are activated when the nominal goal is not achieved (that is, the state of the nominal output state vector *y* is outside of its intended range).

Construction of a GFT, just like any other hierarchical representation, is a progressive construction of each level of the tree. However, the construction of each level is tightly constrained and significantly improved by the heuristics imposed by the state variable methodology and the mathematical definition of a function. By definition, every function is simply a transformation or mapping of input state variables to output state variables. This mapping inherently defines the physics and logic to be manipulated by the function to enable control of the output state variables from the input state variables. As systems engineers and safety engineers know well, the progressive construction of the tree from the top to lower levels implies some assumption at each level about how the system will manipulate physical laws to achieve its goals.

For example, let us compare two hypothetical space transportation systems: a classical staged chemical rocket, and the other a space elevator. It is important to realize that without some basic concept of how the system will work, it is essentially impossible to create a functional decomposition. At each deeper level of functional decomposition, more detailed assumptions must be made, until ultimately the hierarchy has implicitly specified many details of the design.

For both systems, the objective is the same, to transport an object of some mass and volume to a point in space, although it is clear that the space elevator can only transport an object to ONE location in space, and going elsewhere requires further means of transportation. For the sake of argument, we will describe a launch vehicle that has the identical objective of going to the same location as the space elevator, simply to enable direct comparison. For both systems, the top level state variables and function is the same, to move a mass and volume from a location on Earth to a location in space. The state variables would represent the classical orbital elements of the destination, along with the projected mass and volume of the payload to be transported. If the payload requires power and life support, then the variables representing these characteristics must also be described. However, at the very next level of the tree, "level 2", the functions immediately begin to differ. For a space launch vehicle, the second level of the tree identifies the intermediate locations that each stage of the launch vehicle transports the payload to, with separation functions in between. Success at the second level means delivery of the functioning payload to that intermediate location for the separation event, and then successful separation events needed before going to the next stage's operation. We will assume that the launch vehicle has no capability to return the payload. For the space elevator, its second level of functional hierarchy represent modes of operation are quite different, and refer to elevator-like modes such as an ascent mode and a descent mode. For each system, the next level of hierarchy defines further functions that depend on their respective design concepts. For the launcher, this means the ability to control the attitude, velocity, and acceleration by controlling rocket engine thrust vectors, whereas for the space elevator, attitude is fixed while velocity and acceleration are controlled in the vertical only, using electrical or some other mechanism for power as the input, along with the passive or active control of the elevator structure itself.

A. Decomposition and Elaboration



Elaboration
F
Generate Thrust
P_{O2}
T_{O2}
m_{O2}
m_{H2}
T_{H2}
P_{H2}
Decomposition
$\vec{y} = \{x_1, x_2, x_3, v_1, v_2, v_3, a_1, a_2, a_3\}$
$\vec{x} = \{x_1, x_2, x_3\}$
$\vec{a} = \{a_1, a_2, a_3\}$
$\vec{v} = \{v_1, v_2, v_3\}$

Figure 3: Elaboration and Decomposition

The state variable methodology requires precise definition of the physics to be controlled, with a concept of how that control will occur. This leads immediately to two distinct ways in which the state variables can be subdivided and defined: *decomposition* and *elaboration*. *Decomposition* means that an output state vector is simply subdivided into smaller state vectors, which all use the same state variables as the output state vector. That is, no new state variables are created. For example, assume that the output state vector represents the position, velocity, and acceleration of the launch vehicle, with the nine relevant state variables: (x_1, x_2, x_3, v_1, v_2, v_3, a_1, a_2, a_3). This state vector can be decomposed into three state vectors, one for position, one for velocity, and one for acceleration. In this case, there is no function that is required to move down the tree structure, because there is merely a rearrangement and division of the state variables into different state vector groupings; there is no functional transformation.

The alternative situation is when new state variables are required. For example, in the launch vehicle case, the thrust of a liquid rocket engine is generated from the mass, pressure, and temperature of the fuel and oxidizer injected into the combustion chamber. We have thus defined a function f that generates thrust from the propellant inputs. The creation of a new function and new state variables we call *elaboration*. Figure 3 compares a decomposition to an elaboration.

B. Passive and Active Functions

Functions can be either passive or active. Given a thrust, a mass, and an environment, a launch vehicle will move because physics inherently makes it happen. There are other passive functions, such as is often the case with structural functions, which provide control of the relevant state variables (stress, strain) through design margins. For many structures, there is no active control by a control system. If we drive the GFT further, we find that the amount of thrust and the direction of thrust are determined by measurement of translational and rotational rates and accelerations, determination of the difference between the desired and actual translational and rotational rates, and then commands sent to control the direction and magnitude of the thrust vector. The functions to perform these are active, as they require an operational activity of the system to transform the relevant state input variables, such as measured accelerations and rotations, into controlled output state variables of the thrust magnitude and direction.

C. Physically Correct Model Structure

In cases of active control, there is a preferred order to constructing the tree, where the highest level represents the variable to be controlled, such as the thrust vector direction. The next levels represent the steps required to achieve that control, such as the function to command the direction, then the function to compute the required direction, and finally, to measure or estimate the current direction. This ordering reflects that the goal is to control the thrust vector direction, and the levels below it are the goals and corresponding functions required to achieve that control.

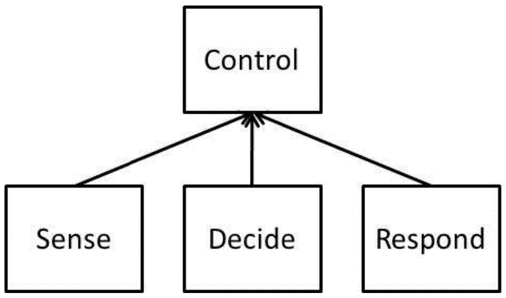


Figure 4: Parallel Mapping of Control Loop in Success Tree is Logically Correct but Causally Invalid

When modeled in this way, the tree structure created closely matches the physics and logic required for the system to achieve its ultimate goals. Unlike a traditional functional decomposition using natural language, the state variable methodology requires that the tree structure matches the system's true paths of causal behaviors. To see why a traditional natural language-based tree can easily be built that does not match the physics, let us look at a typical control system, in which there is a sensor, a processor to determine how to command the actuator, and the actuator that implements the control. A typical natural language decomposition can easily create the model in which the sensor, processor, and actuator are all on the same level. In many purely logical and/or probabilistic models, this yields probabilities that are correct, because the typical assumed structure for a success tree uses AND gates as the default assumption between levels, meaning that all functions at the lower level must be successful for the higher level function to be successful (conversely, failure space models use OR gates as their default). However, the model is physically invalid because the causal behaviors are not independent of each other. By this, we mean that a sensor failure does NOT cause a processor failure, which causes an actuator failure. These are all independent in this model. Put another way, it is possible to construct a success or fault tree in a manner that is logically correct, but physically inaccurate.

By contrast, the state variable GFT requires that the sensor, processor, and actuator are in serial, which also yields the correct probabilities and logic because all functions must succeed for the higher level function to succeed. However, it also ensures the correct behavioral causality. In other words, sensor success is required for processing success, which is required for actuation success, in that order. It is not merely true that all of the functions must succeed for the higher level function to succeed; in addition, the flow of causation must also be valid since these are not

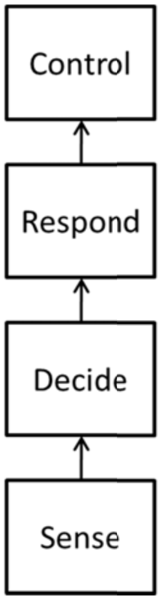


Figure 5: Serial Mapping of Control Loop in Success Tree is Logically and Causally Valid

independent functions. This is the correct model to represent the flow of actual physical operations in time. Figures 4 and 5 shows these two distinct ways to represent control loop functions in a tree hierarchy, with only the series representation being physically correct. For our purposes, correct physical causality is an essential property of the GFT, not just probabilistic and/or logical consistency.

D. Tree Fragment Replication: Similar but Distinct Requirements

Another feature of tree representations in general is the fact that some tree fragments are replicated in several locations, because these fragments represent a common set of functions that support several other functions. These fragments appear in different parts of the tree. The GFT also shares this feature, but with its state variable formulation, it distinguishes subtle but important differences between these otherwise identical tree fragments. With a human-rated launch vehicle, for example, accelerations must be controlled for several different reasons: to prevent breakage of the launch vehicle structure, to prevent breakage of the crew vehicle structure, and to prevent damage to the humans in the crew vehicle. The fact that accelerations must be limited shows up in three different locations in the GFT, and with acceleration state variables. Their appearance in three different locations indicates that these are really three different requirements, which in the GFT appear as three distinct ranges associated with the output acceleration state vector that appears in the three different locations in the tree. These must be tracked and traced as three distinct goals; that is, three distinct requirements.

E. Goal Types

It is often convenient to differentiate different kinds of nominal goals. For example, in the SLS GFT, it was found useful to define three nominal goal types: achievement, maintenance, and prevention. Achievement goals represented requirements to achieve a specific objective at a single point in time, such as attaining the appropriate altitude at the end of the first stage burn. Maintenance goals, on the other hand, are goals that must be maintained over long periods of time so as to attain an achievement goal. So in the first stage achievement goal just mentioned, the launcher must provide thrust and attitude control from the moment it lifts off the ground until it reaches the end of the first stage burn in preparation for first stage separation. Prevention goals are requirements to prevent something from happening. In the same case described above, separation of the first stage from the second stage must be prevented prior to the appropriate separation time. This leads to specific functions such as command inhibits and lockouts to prevent premature separation.

F. Requirements Capture, Completeness and Traceability

A typical feature of systems engineering work is the capture and traceability of requirements. The GFT improves this process in four important ways. The first is its identification of requirements as the control of the ranges of the output state variables of system functions. This precise manner of goal / requirement specification enables the translation of natural language requirements into a precise mathematical, physical, logical language. A second significant improvement is that the GFT's state variable methodology provides a much more structured and rigorous way of ensuring that the goals generated at each level of the tree are a complete representation of the state space that must be provided as inputs to functions so as to enable the control of the output state variables within their nominal ranges. This methodology enables a much better chance that the system engineers will determine a complete or near-complete set of requirements for GFT-modeled functions. A third significant improvement over typical systems engineering practice is that the GFT inherently creates a hierarchy of goals and requirements through its very construction, not just a hierarchy of functions as with a classical function decomposition. Finally, that same hierarchical structure means that requirement traceability is inherent in the GFT model. It is no longer a guessing game as to which requirements decompose into other requirements. In terms of physical relationships, functions, and requirements tied to these relationships, the hierarchy and traceability is guaranteed within the representation itself.

However, it is not obvious whether the functional hierarchy and physical relationships specified by the GFT are the "right kind" of hierarchy desired by the systems engineers or program managers. This is because the requirements decomposition that they desire is based not on function, but on the organizational and institutional structure of the project. Because organizations build the system's components, it is usually true that the interfaces between components are also the interface between organizations. Requirements must generally be specified at the interfaces between components, and hence at the interfaces between organizations. It therefore appears that there are at least two or three distinct kinds of hierarchical representations that are of relevance to systems engineering. The issue of how the GFT maps to the design components, and how these components relate to organizations, becomes important. This paper will address the former in the next subsection, but will not attempt to address the latter; this is a topic for future research.

G. Mapping of Requirements and Functions to Design

Functions describe the operations or physical transformations (which, as described earlier, can be passive or active) that the system must perform to achieve its objectives. The design describes the means by which these physical operations and transformations are achieved. Using the GFT to describe the system's goals and functions, the mapping to the design is based on the state variables and the entities being controlled, which are common to both. Mapping from the GFT to the design is merely a case of searching the GFT and the design for the common entities and the state variable attributes of those entities that must be controlled. The mapping simply links the common entities and state variables from the design model to the GFT and vice versa.

In success and fault trees, the same or closely related functions are identified in many locations of the tree. Thus, in a launch vehicle, for example, "thrust vector control functions" appear in several locations to control attitude so as to facilitate proper trajectory, to ensure structural integrity, control structural temperatures, and to protect payload integrity or human safety. These are repeated in several major tree branches because these same functions are necessary for several system activities. Thus the thrust vector state variables appear tens of times in the overall success tree, but these all map to the same mechanism that performs these functions. It is also true that several mechanisms could map to the same function. There is in general a many-to-many mapping from a GFT to the corresponding system design.

One of the major reasons for developing a functional model is to determine what the system must do (its functions), with some independence from how the system will perform the function. This enables problems to be encapsulated, to enable trade studies of different methods and designs to perform the function. Within a GFT, the more precise definition of functions implies that the input and output state variables should be the same for all design options to perofrm the function. To enable a trade study for the function, the GFT modeler needs to consider the proper level of granularity for the function. As an example, consider a trade study of the potential use of reaction wheels or thrusters to perform spacecraft attitude control, that is, to provide rotational force. To set up the trade study, the function boundaries should be set with common inputs and outputs for both alternatives as much as possible. In this case, the output state vector is the rotational force vector, and a common input state vector, which would include electrical power for the actuator (whether reaction wheels or thruster valves), and the desired amount of rotational correction. In the case of thrusters, this would command a certain amount of thrusting, and for reaction wheels, a command to accelerate or decelerate the relevant wheels. Assuming that propellant is necessary for spacecraft trajectory maneuvers already, propellant would be necessary as an input for thrusters. The trade study would, among other things, have to determine if this would need to be added as an input to the rotation function.

H. Fault Management Goals / Requirements

Fault management comes into play in the GFT by the identification of off-nominal goals to be achieved should a nominal goal not be achieved. For every nominal goal, there is the possibility that the goal is not achieved, which simply means that the values of the state variables of the entity being controlled (i.e. the output state vector of the function), either are deviating, or are predicted to deviate out of range. Deviation out of nominal range is a failure, whereas prediction that deviation out of nominal range will occur in the future, based on a degradation occurring now, is a prognostication. If the systems engineer decides that nothing will be done if that state variable(s) goes out of range, then there is no off-nominal requirement at that location. If something will be done based on an estimate or measurement of that state variable(s), then an off-nominal goal is placed at that "monitored state variable" location.

The off-nominal goal can be of two distinct types insofar as the GFT is concerned. The first type is that the goal of the fault management is to maintain the current set of nominal goals for the system, such as a message that simply notifies a system operator that a problem exists, without actually taking any action to change the system's nominal functions. Another example is when a failure is produced from an external cause, and the system is able to fix the failure effects without having to change system goals. Because the failure was caused from an external event such as wind or cosmic rays that produce Single Event Upsets that flip software bits, if the system can detect and respond appropriately the mission may be able to continue. Examples of appropriate responses would be temporary activation of a different control algorithm to maintain control, or use of error detection and correction (EDAC) code to flip changed bits back to their correct state.

The second type is when the off-nominal action changes the system's nominal functions. In general, these actions produce goals and functions that differ from the nominal GFT, because by definition, new, off-nominal functions are activated. Once activated, these new functions produce new required system behaviors, and these mean that there is a new and different GFT to represent that new set of required goals and functions. In general, because the off-nominal functions use parts of the existing nominal functions (and corresponding designs), parts of the existing nominal GFT are used in the new off-nominal GFT. Thus the new GFT is an amalgamation of parts of the old GFT, typically re-arranged in some manner, and with some new goals and functions that enable this rearrangement. Different systems may choose to create system-specific categories of off-nominal goals, but if they are anything other than notifications, they will lead to the creation of new trees or sub-trees to address the off-nominal functions.

This re-arrangement of the GFT for systems engineering purposes can be seen in another, operational light. In the development of highly autonomous systems and of mission planners, artificial intelligence (AI) techniques are frequently used. These techniques typically include hierarchical tree structures to represent the implementation of system goals, which are searched and re-arranged by the AI system as the intentions for the system change as it reacts to novel circumstances. The GFT can therefore be seen as the equivalent of the AI planning model of system goal priorities, when translated into an operational, autonomous environment. A future research topic is how to relate the systems engineering GFT to the autonomous goal structures such as JPL's Mission Data System, briefly mentioned above in section II.

I. Tree Structure and Causality

Hierarchical tree structures highlight the causal independence of mutually supporting functions. Returning to the structural control described in section B above, in which structures are typically controlled in a passive manner, we find that structural integrity for a launch vehicle is controlled both passively and actively. For example, in a launch vehicle GFT, one finds that to maintain vehicle attitude (rotational) control, we must maintain structural integrity. However, it is also true that if we lose attitude control of the vehicle, we lose structural integrity. For one branch of the tree, maintaining vehicle attitude control is a top function, and a supporting function is to maintain structural integrity. However in other branches of the tree, maintaining structural integrity is the high level function, and maintaining vehicle attitude control is a supporting function! This is not a mistake; these alternative situations represent two distinct causal paths of mutually supporting functions (another example, we need power to perform computing, but we must compute properly to control power). In the GFT path where maintaining vehicle attitude control is the high level function and structural integrity is supporting, a failure of structural integrity will cause loss of vehicle control, but not the other way around. In the other branch, where maintaining structural integrity is the top function and vehicle attitude control supports it, a failure of vehicle attitude control will create loss of structural integrity (that is, the vehicle will break up), but not vice versa. Both paths together capture all possibilities.

This example shows an important feature of the GFT: its ability to be used for physically accurate analysis. In the case described in the previous paragraph, one sees that introducing failures into the system will lead to an accurate analysis of the potential outcomes in terms of how they physically and logically propagate to compromise system goals. Although this analysis is accurate for what it does model, it does not model all possible failure behaviors, as will be described below.

V. Analyses Using the GFT

The GFT enables a variety of assessments and analyses of the system for both nominal and off-nominal purposes.

A. Nominal Requirement and Function Completeness and Traceability

The GFT is inherently a top-down hierarchical representation of both goals and functions, with precise relationships between them. Both goals and functions are represented in a hierarchical manner, and the state variable methodology provides great value in ensuring that as the GFT is constructed from the top down, that the resulting goals and functions trace correctly and completely. Functional and goal traceability is guaranteed within the tree structure, to the extent that the modelers properly and precisely model those relationships as guided by the state variable elucidation of the physics to be controlled. Since goals are requirements, requirements traceability is guaranteed by definition. If requirements were developed separately from the GFT, the GFT can be developed as an independent check of requirements completeness and traceability. If the GFT was the mechanism for requirements development, the structure of the GFT model and the state variable methodology provide strong evidence of completeness. For functions, the same logic applies as it does for goals / requirements to assess function completeness and traceability.

However, this happy story is complicated by the fact that a functional decomposition is not the same as an institutional decomposition. Because requirements and functions need to be specified at component and institutional boundaries, the functional representation of the GFT must be mapped to component (design) and institutional hierarchy representations. Determining the proper method to achieve this is future research.

B. Failure Detection Coverage

One of the major questions of FM design is the coverage of failure detections. Historically, to the extent this has been addressed in FM design analyses, coverage has been assessed from the bottom up. In this bottom-up methodology, design models are built that show the location and relationship of all components (at a relevant level of component decomposition), including sensors, and also including the failure modes of the modeled components. Diagnostic models using directed graphs are typical examples; one is described in Kurtoglu et al. (2008).[10] Failure detection coverage is assessed and measured as the number and identity of the failure modes that can be potentially detected by a failure detection algorithm that uses a given set of sensors. There are a number of uses for this sort of coverage assessment, but one thing that it cannot directly do is to determine whether the failure modes being detected are important in terms of the system's goals. Failure detection coverage is perhaps better assessed by whether failure detections "cover", or detect all possible failures that can lead to compromise of critical system goals. In current typical methodologies, failure modes are assigned criticality values that appear in the failure modes, effects, and criticality analysis (FMECA) matrix. These criticality values represent the result of an informal analysis of the propagation of the failure effects to compromise critical goals such as human safety and success. Thus it is in theory possible with a complete model of failure modes with assigned criticalities to indirectly assess coverage of system goals. Unfortunately, by the time all of these failure modes are known and modeled, if any problems are found it is very far along in the design process, making any fixes quite expensive or impossible.

Another issue is that for any complex system, it is impossible in practice to determine all possible ways in which the system to fail. History shows that many system failures that have actually occurred were for a variety of reasons not considered or ruled out as extremely improbable. In a few cases, the failure mechanisms were not identified. In others, these failure mechanisms were considered, but judged not credible because pre-predicted probabilities of failure indicated that the probability of failure occurrence was trivially small, such as cases of multiple coincident failures. However the problems nonetheless occurred because the design or manufacturing assumptions behind the probabilistic estimate were violated. In yet other cases, the failure mechanism was understood, but the consequences underestimated or not understood at all. Any approach that relies purely on identification of failure modes, whether from the top down or bottom up, is doomed to failure (pun intended) due to the lack of imagination of the analysts and/or the vagaries of how seemingly trivial problems can slip through the nets of testing and analysis and lead to system failure.

A better approach is enabled by the GFT. Failure detection coverage can be directly assessed by determining whether there are any paths from the bottom to the top of the tree that do not have at least one failure detection goal (and hence a corresponding detection response function) along that path. A quick visual scan, or for large models, automated searches can quickly discover any "uncovered paths."

One nuance to this rather simple approach is when there exist several locations in the GFT where the same state variables exist and are monitored, but with potentially different success (and hence failure) ranges. As described in IV.D above, for a launch vehicle, control of acceleration state variables is required to prevent breaking the launcher, the crew capsule, and the crew. These state variables each have different ranges, and hence a failure detection to monitor for vehicle acceleration could have thresholds based on any of these three possibilities. Only one of them is the most stringent range. If this most-stringent range is not selected, then it is possible that a system failure can occur without detection because an incorrect, looser range was selected. As an example, if the most stringent range is to protect the astronauts, but the acceleration detection instead uses a range that is valid for protecting breakage of the launch vehicle, then the safety of the astronauts can be put at risk without the acceleration detection occurring.

C. Failure Scenario Generation

Since the GFT is a causally correct representation, it can be used to identify many, though not all failure scenarios. We define a failure scenario to be a unique combination of failure effects, failure responses, and system configuration and/or time in which the failure occurs. Within the GFT, the failure effects are represented by the changes to the values of state variables from the bottom of the tree as functions perform incorrectly along the failure effect path (which corresponds to the nominal path) up the tree. If the failure can be contained, it will be stopped at a certain location along this path. Otherwise it will continue to proceed up to the top of the tree, compromising the system's primary goal(s). Every path up the tree corresponds to a unique set of failure effects. Potentially different responses are shown by the existence of one or more failure detections (which then link to different responses) along that path. Different mission times and configurations are represented by the existence of several sub-trees in the GFT, each sub-tree representing a unique system configuration and/or time (these create different tree structures). In short, each path up the tree is directly tied to a set of failure scenarios.

Because each path of the tree shows the state variables being affected by the failure effects, it gives important clues about the inherent physics and hence to the timing of the spread of failure effects. Different kinds of physics imply different characteristic times by which the physics propagates. For example, radio signals propagate near the speed of light, while fluid flows propagate much more slowly. Failure effects propagate at the rate dictated by the physics of that propagation. These timing effects are of relevance to analyzing the effectiveness of potential failure responses, as these must operate more quickly than the spread of the failure effects.

However, these GFT-defined scenarios do not define the total set of failure scenarios. This is because a GFT, as a success-space representation, can only represent the system's *nominal* causal connectivity. If a failure creates new connections between functionss, new failure paths that do not follow the system's nominal paths, then the GFT cannot and does not represent these new paths. It is in fact true that some failures create new failure paths, which in the GFT represent "sneak paths" from one part of the tree to another independent set of branches. Examples of these include electrical short circuits and breaks of the physical structure from fires and explosions.

The implication of this is that fault trees and success trees for a given system are NOT complete logical complements of each other, though parts of the respective trees are. Both success trees and fault trees provide value for systems engineers and system health management engineers, but for different kinds of analyses.

VI. Conclusion

The GFT model, with its rigorous state variable-based methodology, extends the concept and methodology of the standard systems engineering functional decomposition, and makes it physically and causally valid. This enables it to be used for a variety of analytical uses and significantly improves the quality, comprehensiveness, and traceability of both nominal and off-nominal requirements generation and maintenance. This utility has been demonstrated on the NASA Space Launch System Program, providing significant value in the design and assessment of Fault Management for this human-rated launch vehicle. The model and associated methodology enable direct analysis of FM detection coverage of various combinations of abort triggers, the mapping of various failure detections to failure scenarios, and enabled early assessments of expected failure response effectiveness.

Acknowledgments

Much of this work was performed on the NASA Space Launch System Program, contract NNM12AA41C. The authors thanks Jonathan Breckenridge, Ben Hager, Bradley Biehn and Michael D. Watson at Marshall Space Flight Center, Christian Neukom and Eric Barszcz at Ames Research Center, and Bill Maul from Glenn Research Center for their many insights in the initial development of the SLS Goal Tree / Success Tree and the state variable methodology. The author also thanks John C. Day, Bob Rasmussen, and Dan Dvorak at the California Institute of Technology's Jet Propulsion Laboratory for many fruitful discussions of fault management, success trees, and their relationships to the operational success/planning trees under development with the Mission Data System, which is the closest operational analog to the GFT approach to systems engineering described in this paper.

References

[1] Johnson, S.B. et. al., eds. *System Health Management: with Aerospace Applications*. John Wiley & Sons, Chichester, United Kingdom, 2011. Johnson, Stephen B., and John C. Day, "System Health Management Theory and Design Strategies," for AIAA Infotech@Aerospace Conference 2011, 29-31 March 2011, St. Louis, Missouri. AIAA paper 977233. Johnson, S. B., and J. C. Day, "Conceptual Framework for a Fault Management Design Methodology," AIAA Infotech Conference, Atlanta, Georgia, April 2010; AIAA paper 227006.

[2] Van Lamsweerde, A. *Requirements Engineering: From System Goals to UML Models to Software Specifications*. John Wiley & Sons, Chichester, UK, 2009.

[3] Nilsson, N.J. *Problem Solving Methods in Artificial Intelligence*. McGraw-Hill, New York, 1971.

[4] Dardenne, A., Ficklas, S., and van Lamsweerde, A. 'Goal-Directed Concept Acquisition in Requirements Elicitation," *Proceedings of IWSSD-6 – 6th International Workshop on Software Specification and Design*, Como, 1991, 14-21.

[5] Mylapoulos, J., Chung, L., and Nixon, B. 'Representing and Using Nonfunctional Requirements: A Process-Oriented Approach', *IEEE Transactions on Software Engineering*, Vol 18, No. 6, June 1992, 483-497.

[6] Kim, I.S., Modarres, M. *Application of Goal Tree-Success Tree Model as the Knowledge-Base of Operator Advisory Systems*. Chemical Process Systems Engineering Laboratory, Department of Chemical and Nuclear Engineering, University of Maryland. Submitted to *Nuclear Engineering and Design Journal*, October 1986.

[7] Modarres, M., and Cheon, S.W. "Function-Centered Modeling of Engineering Systems using Goal Tree-Success Tree Technique and Functional Primitives," *Reliability and Safety Engineering* 64, 1999, 181-200.

[8] Ingham, M., Rasmussen, R., Bennett, M., and Moncada, A., "Engineering Complex Embedded Systems with State Analysis and the Mission Data System", *AIAA Journal of Aerospace Computing, Information and Communication*, Vol. 2, No. 12, Dec. 2005, pp. 507-536.

[9] NASA Fault Management Handbook, NASA-HBK-1002, Draft 2, April 2, 2012.

[10] Kurtoglu, T., Johnson, S. B., Barszcz, E., Johnson. J., and Robinson, P.I., "Integrating System Health Management into Early Design of Aerospace Systems using Functional Fault Analysis," 2008 International Conference on Prognostics and Health Management, Denver, Colorado, October 2008.

Goal-Function Tree (GFT) Modeling for Systems Engineering and Fault Management

Dr. Stephen B. Johnson
sjohns22@uccs.edu and stephen.b.johnson@nasa.gov

Jacobs ESSSA Group, and Dependable System Technologies LLC
NASA Marshall Space Flight Center
EV43 Integrated System Health Management and Automation Branch
and
Mechanical & Aerospace Engineering Department
University of Colorado, Colorado Springs

19 August 2013
AIAA Infotech@Aerospace
Boston, Massachusetts

What is a GFT?

- **A top-down hierarchical decomposition of system goals and functions --- hierarchies are models of “intention”**
  - Arranged by major system phase/configuration,
  - Defines the functions the system must perform and goals the system must achieve for the system to successfully perform its mission/objectives
  - Relationship between goals and functions defined through rigorous use of state variables

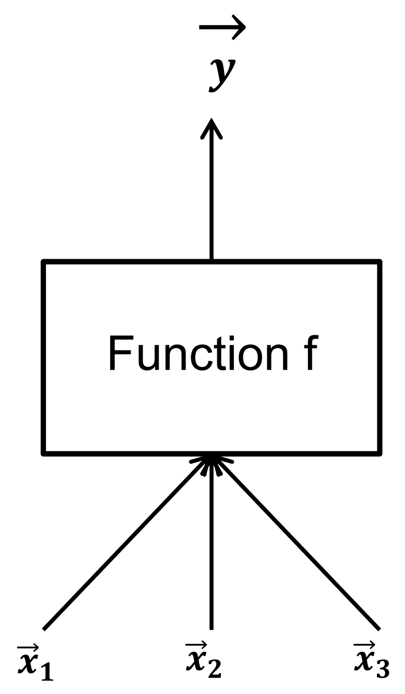
- **The GFT extends the classical function decomposition through the use of state variables to explicitly contain goals, and makes the GFT causally, physically correct**

- **It is based on function, not design, though inherently you *must* make some design assumptions**
  - **Example: Goal is to place humans into trans-lunar trajectory**
  - **Two possible solutions:**
    - **Star Trek-style quantum transporter**
    - **Space Launch System-style chemical rocket**
  - **These two solutions yield very different function decompositions, SLS has stages, the transporter does not, etc.**

State Variable Methodology

- State variables are defined as inputs and outputs to functions: y=f(x)
  - x = inputs to the functions f
  - f transforms the inputs into the outputs y
- Goals = Requirements = define intended range of the output state variables y
- Failure = state (value) of output state variable y is out of intended range
- State variables enforce strong connection of the functional decomposition to the system's physical laws and causation
- The state variables are the connection between function and design—exist in both function and design representations

$R_l \leq \vec{y} \leq R_h \Rightarrow G$



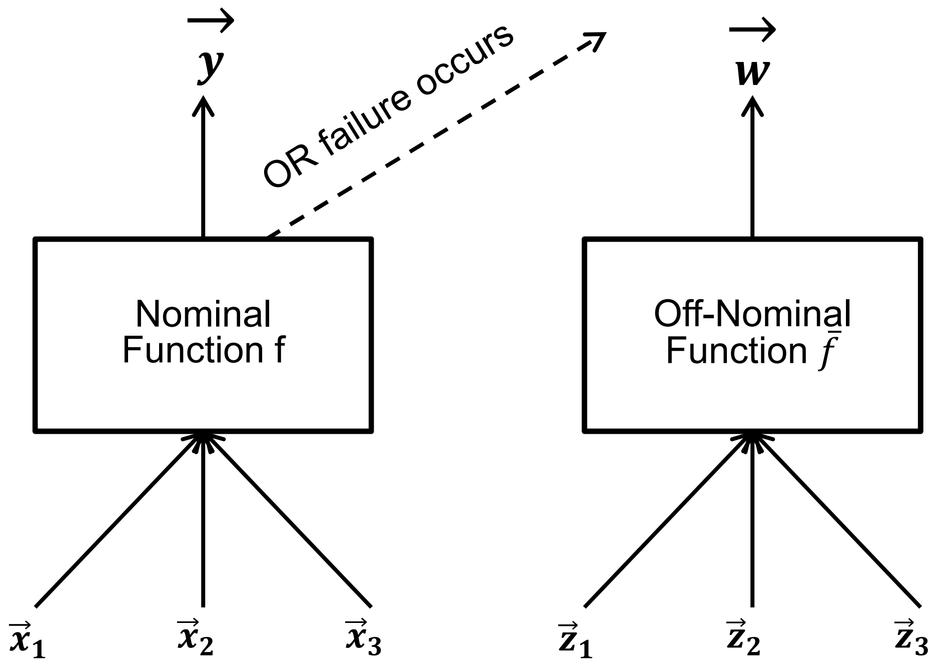
JACOBS
ESSSA Group

Off-Nominal Goals

- **For every nominal goal there is the possibility of an off-nominal goal, which is activated if the nominal goal is not achieved**
  - Value (state) of state vector ***y*** goes out of intended range

- **If a new off-nominal goal is identified, this is an operational Fault Management goal, with associated off-nominal functions to detect or predict failure, and respond**

$R_l \le \vec{y} \le R_h \Rightarrow G$ $R_l \ge \vec{y}$, or $\vec{y} \ge R_h \Rightarrow \bar{G}$

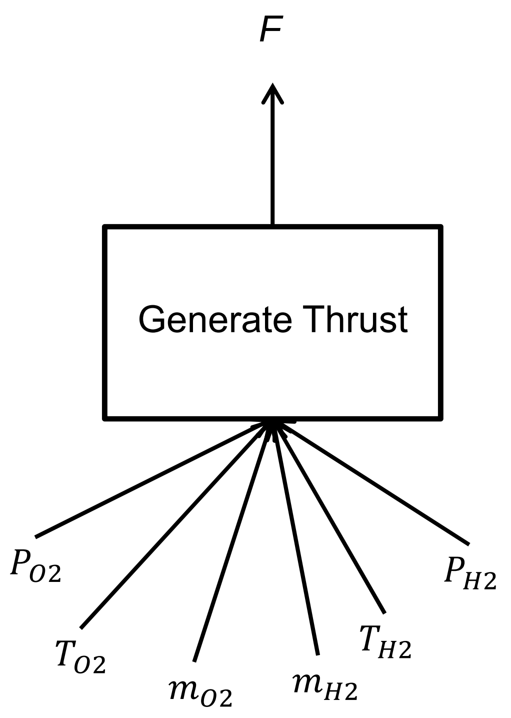


JACOBS ESSSA Group

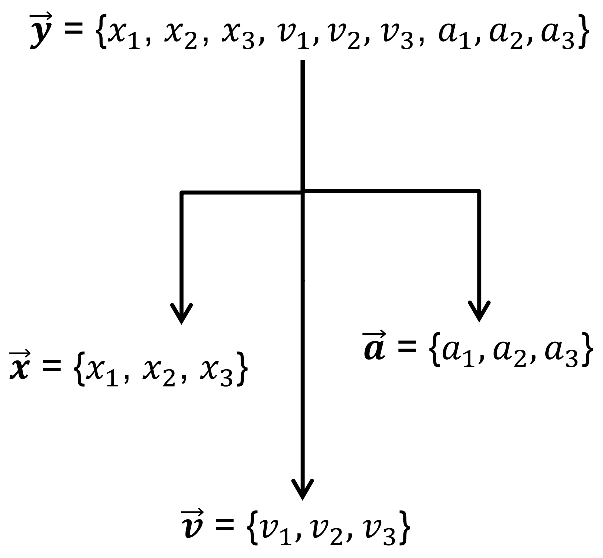
Elaboration vs. Decomposition

- **Decomposition merely re-arranges a state vector's variables into several state vectors containing the same variables**
  - No function exists
- **Elaboration creates new variables via a function**

Elaboration



Decomposition



$\vec{y} = \{x_1, x_2, x_3, v_1, v_2, v_3, a_1, a_2, a_3\}$
$\vec{x} = \{x_1, x_2, x_3\}$
$\vec{a} = \{a_1, a_2, a_3\}$
$\vec{v} = \{v_1, v_2, v_3\}$

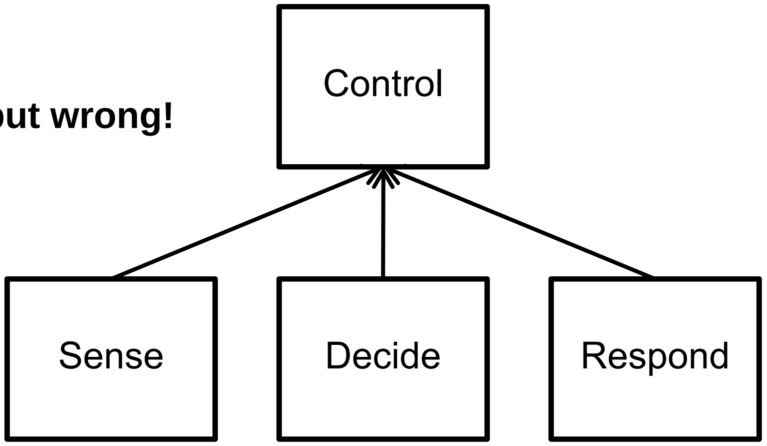
JACOBS ESSSA Group

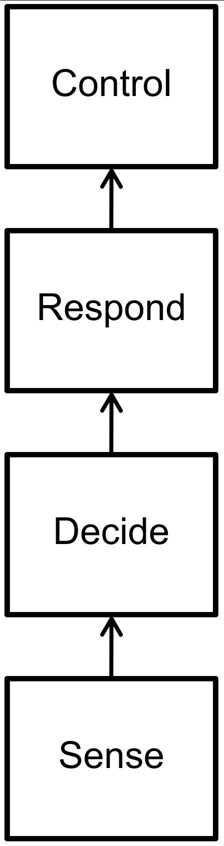
Local Motivation for the GFT

- **NASA Space Launch System – Fault Management Analytical Issues**
- **Does SLS detect failures properly in all SLS-caused failure scenarios that threaten the crew? [i.e.: failure detection coverage and failure scenario definition]**
- **What is the relationship of crew-threatening behaviors to each other, to system goals, and to potential detections?**
- **What is the optimal distribution of FM detections and responses?**
- **For a complex system, it is impossible to determine all possible ways the system can go wrong, but we can and should be able to specify completely what must go "right"!**
    - Need to base SHM/FM design in part on protecting the functions needed to succeed, regardless of how they might fail
- **Early in a program, detailed design and related FMEAs do not exist, but need to start FM early in design phase**
    - This must be a top-down analysis based on goals and intended functions, not just on design (SLS is only partly heritage)

Modeling for Logic and Causality in a Control Loop

Typical, but wrong!





Physically and logically correct; causation is correctly modeled.

In both constructions, the basic logic and related probabilities of success are correct, because S-D-R all must succeed for Control to succeed.

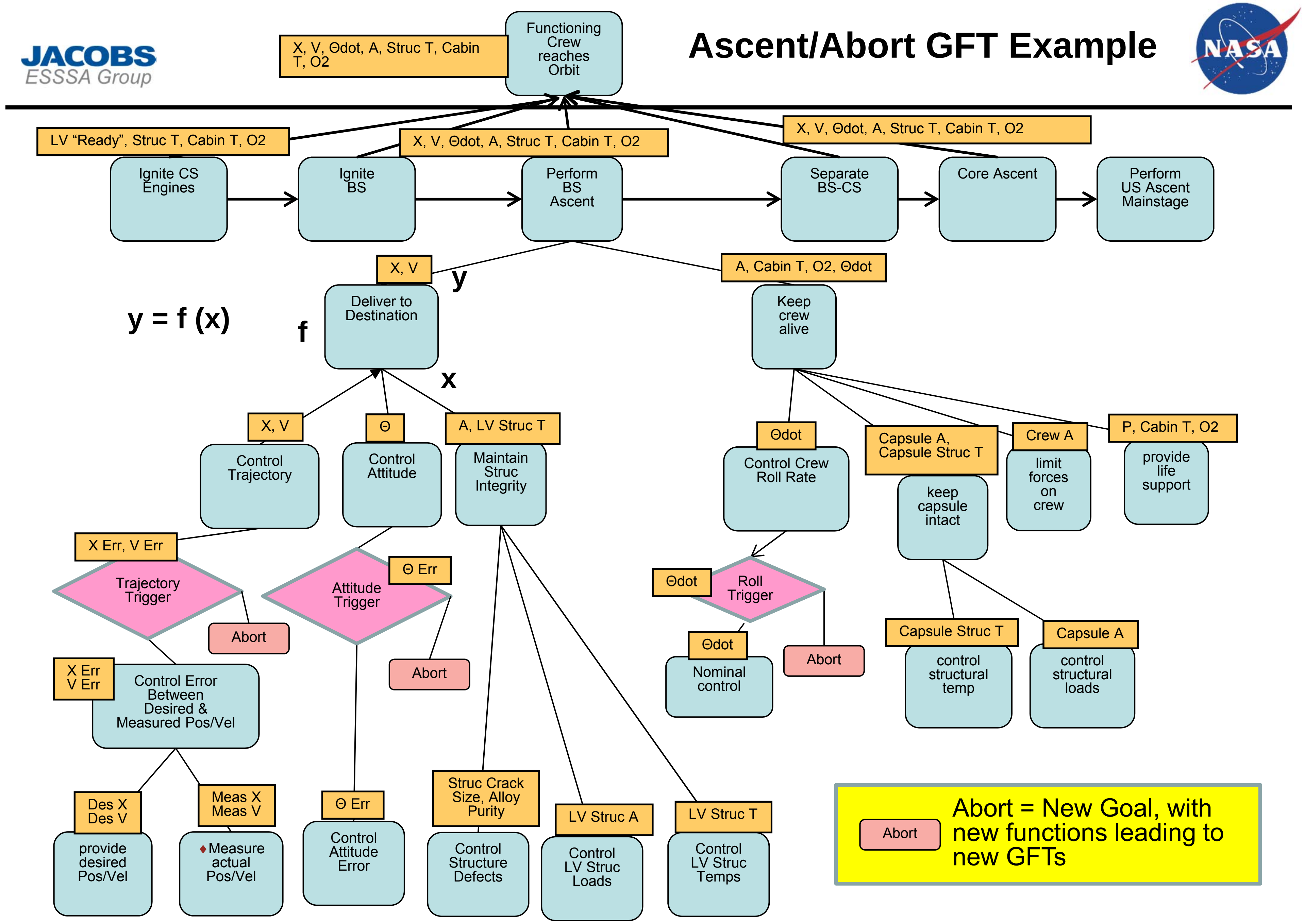
However, only the serial model is physically-causally correct, because if S fails, then D and R will both fail. S, D, and R, are NOT logically independent!

We require the GFT to be causally correct, not merely logically correct.

The State Variable methodology FORCES the model to be causally correct, and hence it can be used for physical analyses of many kinds!!

JACOBS ESSSA Group

Ascent/Abort GFT Example



Abort = New Goal, with new functions leading to new GFTs

GFT Analysis: Nominal Requirements Generation and Traceability

- **Every GFT Goal is a system functional requirement.**
  - A typical functional decomposition is only about functions; the relationship to their associated requirements is vague.
  - The GFT inherently associates goals to requirements.

- **All requirements are represented in mathematical-physical language, with constraints placed on state variable ranges.**

- **The state variable methodology ensures correct physical traceability and connectivity between functional requirements**
  - Natural language ambiguities make determining traces between natural language requirements problematic. It is easy to miss or misinterpret proper traces.

- **The state variable methodology provides much better evidence of requirements completeness than typical natural language requirements.**
  - The physics and logic being required are directly represented.

Nominal Requirements Traceability and Application Issues

- Does a functional decomposition, even using a GFT, create the "right kind" of traceability needed for systems engineering?
- Issue: requirements must be specified on interfaces between components and hence on organizations, but how do these relate to functions and functional requirements as represented in classical function decompositions and the GFT?
- Implication: function-based trees in general do NOT easily represent the kind of hierarchy desired, which is the institutional / component hierarchies in which requirements must be specified and levied on organizations to build components.
- Future research: building the organizational and component hierarchies and relating these to the GFT.

JACOBS ESSSA Group

GFT Fault Management Analyses #1

- **Every unique path up the GFT from bottom to top is a failure scenario**
  - Though this does not generate all possible failure scenarios
  - GFT paths are only nominal paths, but failures can create new off-nominal paths, such as structure breakage or electrical short-circuits
  - Scenario definition also requires a fault tree, not just a success tree, but this fault tree must be based on the same state variable methodology as the GFT
- **Failure detection coverage is determined by identifying all paths that have at least one failure detection along the path, and conversely for any non-covered paths**
  - Nuance: when the same state variable exists in more than one GFT location, it indicates a different requirement / range constraint levied on the same state variable: example---acceleration constrained by need to protect the crew, the crew capsule structure, and the launch vehicle structure
  - When this occurs, it is possible that the threshold values set for the failure detection associated with the state variable can be set to the wrong value; it needs to be set to the tightest requirement

JACOBS
ESSSA Group

GFT FM Analysis #2

- **Optimization of failure detection & response based on the distribution and relationship of failure detections along GFT paths**
  - At least one failure detection should exist along every GFT path
  - When more than one failure detection exists along a GFT path, then there is redundant detection for a given scenario. These could indicate excess redundancy, and this should be assessed. It could be that the detections are needed to cover other scenarios.
  - Failure response effectiveness is based on the race condition of the FM response speed compared to the failure effect propagation rate. The latter are related to the types of physics indicated by the state variables along the GFT paths (note not all failure paths exist in the GFT, see previous page). Examples: electrical state variables indicate electron flows with characteristic speeds; these differ from pressure and temperature state variables and their characteristic times for fluid flows.
  - Some paths have failure probabilities higher than other paths. For these it is appropriate to have detections “lower down” in the GFT to make more response time available before compromise of high-level goals.
- **Identify the “sneak paths” created by off-nominal paths.**

JACOBS
ESSSA Group

Conclusion

- GFTs extend the classical “functional decomposition” to include goals (requirements) and state variables
- Requirements / goals are translated from natural language into constraints on the range of controlled state variables, which in turn are the outputs of functions
- The rigor applied through state variables ensures the GFT is physically and causally valid, and hence can be used for a variety of physical analyses for both nominal and off-nominal purposes
  - Requirement and function traceability and completeness
  - Failure detection coverage and optimization
  - Failure scenario generation
- The GFT has been successfully applied to the NASA SLS Program, and used for off-nominal analysis of abort conditions and detections, using SysML as the modeling language