Fault management based on systems description as directed graph with absolute dependence relations

The article is accepted for publication in IEEE Systems Journal, Early Access link:
https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8779661

Fault Management based on Systems Description as Directed Graph with Absolute Dependence Relations

Recep Firat Tigrek

***Abstract*—A new systems description method that utilizes graph theory for systems modeling is presented. The new method represents the system in terms of a directed graph, where each system component is represented by a vertex and dependencies between system components are represented by directed edges, also called arcs. The absolute dependence rule is introduced as the criterion for the analysis of a system into the graph representation. Together with the concept of an elementary component of a system, the proposed systems description method is applicable to a wide variety of systems for fault diagnosis, reliability engineering and other system modeling and system architecture design purposes. The application of the new method for fault diagnosis is demonstrated in this paper by developing a novel fault management algorithm, and connections with several other systems description methods are presented.**

***Index Terms*— Fault diagnosis, reliability engineering, graph theory, systems modeling, systems architecture**

I. Introduction

Systems engineering can be described as designing the system behavior through the whole life cycle of a system [1]. The requirements describing the system performance and behavior throughout its lifetime are derived from the specifications provided by the customer. These requirements, in turn, consist of quantitative performance criteria, description of the interaction between the system and personnel, and the interfaces between the system and the other systems with which the designed system has to operate together. Systems engineering involves the interdisciplinary effort to formulate these requirements and break them down into specifications for system components such that the combined action of all constituents of a system realizes the desired system performance. This task is an optimization procedure in essence, because there are, in general, many components that may fulfill a specific function within a system. Each system component option requires specific interfaces and interactions with the other components. Certain components may rule out the use of certain other components or may otherwise have unique interfacing requirements. The system architecture brings together a viable set of components to realize the system. Hence, the system architecture is strongly connected with the system behavior. Fault management is an example of system behavior that is heavily influenced by the system architecture. During the system operation, any fault occurring within the system has to be reported promptly and accurately to initiate corrective action. The accuracy with which faults can be isolated depends on the organization of the system and the fault detection provisions proposed by the system designers.

Development of the method presented in this paper began in the course of systems engineering activities concerning active electronically scanned array (AESA) radars. Initially the method was developed as a means to improve fault management algorithms and conduct analysis of fault isolation accuracy. The approach for the fault detection and isolation for AESA radars focuses on the RF characteristics of the system, rather than proposing a system-wide fault diagnosis and isolation methodology [2]. Array calibration is a crucial stage in the operation of an AESA radar and it is an important means of fault detection; however the AESA is only part of the whole radar system which may incorporate thermal management, electromechanical, communication and power generation subsystems.

The contribution of the paper can be summarized as the introduction of the absolute dependence principle and the concept of an elementary component. The systems description method as directed graph with absolute dependence relations is a method that is rigorous and based on universal principles. Hence, this novel method is applicable to a wide variety of processes and systems besides AESA radars.

The wide-ranging coverage of the new method can be demonstrated better when compared with the system representation methods in the literature, such as [3], [4], [5] and [6], where graph representation emerges as the major tool for representing the flow of events leading to the failure of the system. An automation method for reliability analyses is proposed in [7], using the link between the fault tree analysis and Failure Modes and Effects Criticality Analysis (FMECA) through representation of the fault tree as directed graph. While these methods aim to describe the relation between system elements, the relations under consideration are event-based and can be considered as scenario specific. Hence, it still takes

The manuscript was submitted on July 03, 2018 for review. The method described in this work was developed by the author while working at Aselsan A.S., Ankara, Turkey.

R. F. Tigrek is with the Signal Processing Systems Group in the Electrical Engineering Department, Eindhoven University of Technology, Eindhoven, the Netherlands. (e-mail: r.f.tigrek@tue.nl)

significant labor to develop and fit the scenario into one of the graph representations proposed in the literature.

A goal of system reliability and maintainability effort is the prevention of ambiguous fault isolation and no fault found events. Ambiguous fault isolation refers to those cases where a fault in the system, perceived through the failure of the system to realize its intended functions, cannot be traced to a particular replaceable unit. In such cases the functionality of the system, which is failed in the particular case, is provided by a group of units that depend on each other for their individual functionalities. Built-in test equipment (BITE) is implemented to resolve such ambiguities, however the guidelines to implement BITE in a system show a strong dependence on the expert knowledge of the systems engineer [8], and quantitative analysis of the BITE efficiency is available only after design alternatives for the BITE structure of the system take shape. No fault found events can be associated with the ambiguous fault isolation, for one of the practices in the system maintenance is to replace all units that may contribute to the fault condition and test these units in the repair center, as described in [9] and [10]. A systems description method that facilitates analytical BITE coverage analysis may benefit the BITE design and reduce the system maintenance overhead.

The proposed methodology introduces a fundamentally different mindset compared to the techniques applied today. Rather than facilitating easier interaction of the design engineers with the system representation or focusing on the system behavior rather than the system architecture, the proposed method provides a coherent description of the system architecture that can be processed by a computer. To this end, consistent operation rules based on logic operators are proposed, and adherence to these operation rules is defined as the key measure of the accuracy of the system representation. In the final outlook, any fuzzy or gradually changing parameter aggregates into a binary decision; for fault tree analysis it is one of the conditions that should be satisfied for the application of minimal cut set analysis [11], while for logistics purposes it corresponds to the decision whether to replace a unit or not. The proposed technique functions by this paradigm, where the system organization is described in terms of absolute dependencies between elementary system components. Using this binary approach, a method is developed to test the integrity of systems description, isolate faults in a system, analytically assess the BITE coverage and generate proposals for BITE locations in a system. The method can be developed further to automate the FMECA and fault tree analysis, significantly reducing the need for the opinion and time of expert engineers.

The paper begins by introducing in Section II the new method that depicts a system as a directed graph. Elements of the graph description are the vertices and the directed edges; logic rules that govern the analysis of the system into graph elements are also presented in Section II. Section III focuses on utilization of the new method for system fault management, with investigation of specific cases in Section IV to demonstrate the rigor of the absolute dependence principle. One such case concerns the assessment of BITE placement in an AESA radar. Readings from the system have to be translated into binary fault states and fed into the graph representation, whereas the output of fault diagnosis algorithms have to be aggregated for reporting. A three tiered structure, described in Section V, is proposed to this end. In Section VI connections between the new methodology and two of the existing system representation and analysis methodologies, namely the design structure matrix (DSM) and unified modeling language (UML) are explored. The paper is concluded with a summary and remarks on the automation of reliability analysis.

II. The Systems Description as Directed Graph

A graph is a mathematical object that consists of vertices (also called nodes) connected to each other by edges. If an edge provides only a one-way connection between two vertices, the edge is directed. If all edges in a graph are directed, the graph is called a directed graph (or digraph). Thus, a directed graph consists of directed edges, also called arcs, between vertex pairs. More detail on directed graphs can be found in [12].

A functional block diagram can be considered as a directed graph, where each block corresponds to a vertex and connections between blocks correspond to directed edges. However, the types of analysis that can be conducted on a block diagram is quite limited. The nature of connections between blocks is not well defined, in the sense that edges connecting blocks can represent many different types of interfaces. In other words, the variety of connections cannot be handled by any single analysis technique. Consequences of cutting an edge, which corresponds to severing an interface, are not explicit and usually require further analysis of the behavior of the block which had the removed edge at its input. To investigate the behavior of a block in order to uncover the consequences of an interruption in the flow, the block can be analyzed further into subblocks and directed edges connecting these subblocks representing the dependence between subblocks, in a fashion similar to that in [13]. The analysis can be conducted in a recursive fashion, dividing every sub-block into further sub-blocks, until elementary components of a system are obtained.

A. Vertices and Directed Edges

The proposed method analyzes the system functions such that the elementary functions are revealed. These functions might be fulfilled by software or they might be associated with hardware, which corresponds to elementary system components. An elementary component is defined from the perspective of fault states: An elementary component is either operational or not operational, which is indicated by a Boolean fault state. Each such system component and function is assigned a vertex point. Depending on the analysis that will be conducted over the graph description, a number of other Boolean indicators can be assigned to each vertex.

Components of a system depend on each other to fulfill their functions, hence, remain operational. This dependence of elements is modeled as directed edges, which point from the providing vertex towards the dependent vertex. As such, the system is modeled as a directed graph, where providing vertices are predecessors to dependent vertices and dependent vertices are successors to providing vertices.

B. The Absolute Dependence Principle

The absolute dependency requirement forms the core of the proposed method: Any dependent vertex that is not properly serviced by a provider vertex must enter the fault state. In other words, if a vertex is at a fault state, any vertex that depends on the faulty one must also enter a fault state. The absolute dependence principle can be modeled as *OR-based fault propagation principle*.

Definition: *OR-based fault propagation principle* states that a vertex is faulty if any of the predecessor vertices is faulty. Fault condition for a vertex must be consistent with the logical OR operation over the fault conditions of the predecessor vertices.

The OR based fault propagation defines the mapping of the system behavior over the graph representation. If this condition is violated in the actual system behavior, the system must be analyzed further to make sure that no two distinct functionalities are represented by a single vertex. In other words, adherence to OR based fault propagation guarantees that elementary functions and components that constitute the system are revealed.

C. Systems Description Conditions and Graph Properties

The systems description as a directed graph with absolute dependence relations is a system analysis and representation methodology based on three fundamental conditions and the definition of the elementary component, which emerges from these conditions.

Condition 1: It must be possible to aggregate the operational status of each component into a binary fault state indicator, which is assigned to a vertex representing the component.

Condition 2: Directed edges must indicate the dependence relation between vertices, hence, components.

Condition 3: Dependence relations must be absolute. In other words, a vertex must necessarily be faulty if any of its predecessors is faulty.

Definition: A component is *elementary* if it satisfies these conditions with its incoming edges. In other words, an *elementary component* has a binary fault state indicator which is driven to faulty state if any of its predecessor components also has faulty state.

It is noted the term "component" does not only refer to hardware, but also to software and functional outputs and services. Any constituent and service of a system with measureable specifications is abstracted as a component of the system, and each such component has dependence relations with other components.

There are two additional rules that emerge from the definition of an elementary component:

Rule 1: An edge cannot start and end at the same vertex. In other words, a vertex cannot depend on itself. The statement that a vertex absolutely depends on itself is a tautology, which is avoided by adopting this rule.

Rule 2: Two vertices cannot have mutual dependence. Otherwise, the fault condition in one vertex would imply the fault condition of the other vertex. In that case, representing these vertices as separate entities is redundant, the two can be combined into a single vertex.

It is possible that a cycle is formed by a sequence of dependency relations between several vertices. At this point, it is not possible to rule out the occurrence of cycles in the graph representation of a system. However, detection of cycles is essential and care must be taken when analyzing the graph representation of a system.

III. The Fault Management Algorithm

The fault management can be considered from two different aspects: In the system design phase, the fault management is closely associated with the reliability analyses. The ability of the system design to successfully diagnose the source of a fault to a line or shop replaceable unit is assessed, and necessary BITE infrastructure is designed to facilitate the fault diagnosis capability. In the system operation phase, the fault management consists of algorithms that monitor the system outputs and BITE readings to detect any fault condition and diagnose its source. For both phases, the graph representation of the system with absolute dependencies provides new insights and automation opportunities.

A. Observable Vertices

For the fault management algorithm, not all vertices comprising a system are directly observable. While those vertices that model the system functions are generally observable, internal vertices that provide for these observable vertices are not observable unless specific measures are implemented.

Those components with which a communication link is present or from which messages that signal the activity of the component can be received are observable. Examples of such components are processing units and microcontrollers. Another set of observable vertices belongs to built-in test equipment (BITE)'s, which are specifically designed and placed to provide information over the distinct units within a system. BITE may measure a wide array of parameters, such as the thermal condition of a component, electrical characteristics of an interface, or mechanical effects such as shock and vibration. BITE outputs can be conveyed for fault diagnosis purposes in various ways, such as reading through messages received by a microcontroller or signaled through light and sound indicators. Translation of BITE readings into Boolean fault indicators is covered in Section IV.

B. The Fault Management Algorithm Description

Readings from an observable vertex determines the fault condition for that vertex. However, for unobservable vertices the diagnosis must be obtained according to a rule that is derived from the fault propagation principle.

Definition: *AND Based Fault Diagnosis Rule* requires a vertex to be diagnosed as faulty if and only if all other successor vertices are diagnosed or observed as faulty. In other words, fault diagnosis of an unobservable vertex is determined by the logical AND operation over the fault diagnosis or observations of the successor vertices.

For the fault management during system operation, readings

from observable vertices are translated into binary fault states. The fault diagnosis algorithm is implemented as a recursive tree search, where each vertex is evaluated for fault condition according to fault states of all neighboring vertices. The fault state of a vertex cannot be determined if all the successor vertices are not already evaluated for fault condition. Thus, the tree search algorithm runs in a depth-first fashion until vertices with no outgoing edges or those with all successors evaluated are encountered. After all vertices are evaluated for fault status, the core cause of the fault can be traced by checking for the faulty vertices that have no faulty predecessors. The pseudo-algorithm in Fig. 1 explains the use of absolute-dependence based systems description for fault management.

Care must be taken for detecting cycles in the directed graph describing the system. The recursive tree search will run on a cycle forever, since the fault state of each vertex will depend on a successor which lies on the same cycle. As the presented fault management algorithm treats observable vertices as evaluated, one way of breaking a cycle is to render observable one of the vertices in the cycle.

The operation of the fault management algorithm can be observed through a simple case presented in Fig. 2(a), where two observable vertices *D* and *E* are preceded by three unobservable vertices *A*, *B* and *C*. *A* and *C* precede solely one observable vertex each, while the vertex *B* precedes both observable vertices. Assume a fault is observed at the vertex *D*. The fault management algorithm initially assumes the faulty state to all unobservable vertices *A, B* and *C*, then proceeds to clear both *B* and *C* as *E* is observed as not faulty. However, there is no way of clearing the fault state of *A*, because the only observable vertex that succeeds *A* is faulty, even though the fault may actually be due to *D*. In other words, when a fault is observed at vertex D, it still means that D is possibly faulty. The algorithm searches for root causes for any fault observation amongst the non-observable vertices, and identifies any vertex as possibly faulty as long as they are not cleared by the presence of at least one dependent vertex with no fault status. The output of the fault management algorithm is depicted in Fig. 2(b).

Fault Management Algorithm

INPUT System Description as vertex{ID, obs, pre, suc} *(Directed graph input as vertex objects with attributes ID, observability condition and list of predecessor and successor vertex handles)*

For each vertex, assign Boolean state flt = 1 (*faulty)*

For each vertex, assign Boolean state cmp = 0 *(search not complete)*

For each vertex, assign Boolean state str = 0 *(search not started)*

For each vertex with vertex.obs = 1, assign vertex.cmp = 1 *(search complete)* and write the observed fault state to vertex.flt

For each vertex with vertex.cmp = 0, call **Search Algorithm with INPUT vertex{.}**

For each vertex with vertex.flt = 1, if all vertex.pre.flt = 0, append vertex.ID to FaultList

OUTPUT FaultList

Search Algorithm

INPUT vertex{.}

If vertex.str = 1, algorithm terminated, OUTPUT Error Report("a cycle is detected")

Set vertex.str = 1 for the vertex under investigation

For each successor vertex, if vertex.suc.cmp= 0, for that successor vertex call **Search Algorithm with INPUT vertex.suc{.}**

Set vertex.cmp = 1 for the vertex under investigation

Set vertex.flt = AND(vertex.suc.flt) *(assign the result of the AND operation over the fault states of all successor vertices to the fault state of the current vertex*).

Fig. 1. The fault management algorithm is explained as pseudo-code.

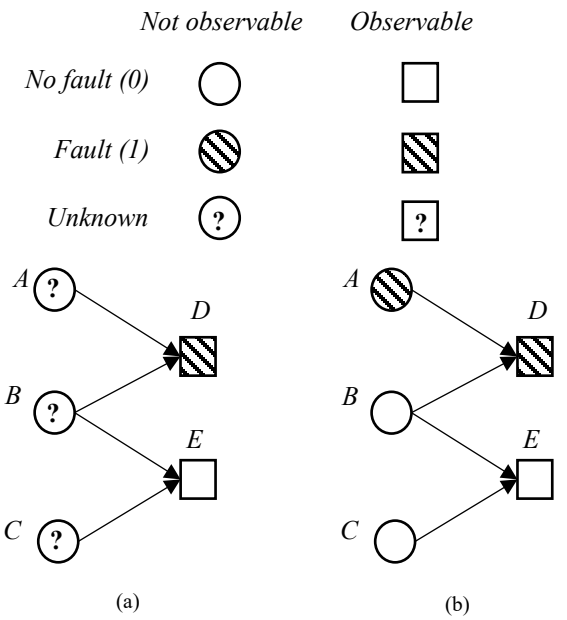

Fig. 2. A graph representation demonstrates the operation principle of the fault management algorithm. In (a) the input to the algorithm is given, and in (b) the output of the algorithm shows that the vertex *A* is identified as the possible cause of the fault observed in vertex *D*.

C. Computational Complexity of the Fault Management Algorithm

The algorithm as described in Fig. 1 is in essence a depth-first search algorithm, which keeps exploring along successor vertices until an observable vertex is reached. However, the processing of a vertex is concluded only after every successor vertex is also processed. Thus, it is possible to recognize the first vertex that is processed by the algorithm as the root of a tree, whereas all the observable or terminal vertices (i.e. vertices with no successors) constitute the leaves.

The proposed reorganization of the systems graph seems to imply an immense computational complexity, however, it is noted that once a vertex is processed for one search tree, it is also processed for all other trees of which it is a part. In other words, once a vertex is processed, it becomes a leaf for any other tree that includes that vertex, effectively pruning those trees and reducing the computational burden associated with their processing.

The computational complexity of the algorithm can be assessed by reorganizing the list of vertices according to the maximum depth of observable and terminal vertices from each vertex. According to this list, if the vertices with the least maximum depth are processed first, the remaining vertices are pruned, leading to reduced search depths. This approach actually leads to a sequence of search trees with depth equal to 1: Any depth greater than 1 implies that there is a subtree with lesser maximum depth, which can be processed to prune the original tree. This topological ordering transforms the depth-first search into a breadth-first search, where every vertex in the graph is explored according to the topological ordering, starting from the leaf vertices. It is noted that if the directed graph representation is pre-processed to obtain a topologically ordered graph, the search cycles can be eliminated from the algorithm in Fig. 1. The computational complexity is calculated under this assumption, as the topological ordering is an offline processing task that is undertaken before the system is deployed, while the fault management has to run in real time.

The operation for each vertex consists of running the AND operation over the fault states of all its successor vertices. To have a worst case figure for the computational complexity, the topological ordering concept can be explored further. For a directed acyclic graph of N vertices, the number of edges can be at most N(N-1)/2. This number is obtained when each vertex is a predecessor to every other vertex that comes after it in the topological ordering. Thus, the number of AND operations required to diagnose the faults is equal to (N-1)(N-1)/2.

IV. Specific Fault Management Examples

Several specific examples are inspected in detail for the fault management application to demonstrate the applicability of the new method and reveal possible shortcomings.

A. Distinction between Parameter Measurement and Thresholding

The first example in Fig. 3(a) considers a BITE vertex B monitoring a parameter of another vertex A. Another edge establishes a dependence relation between the observable vertex B and the unobservable vertex C. In this case in Fig. 3(a), the vertex B is not only BITE but also acts as a trigger to call vertex C into operation according to the measurement it takes. The problem with this representation is the ambiguity in the term ‘fault’. The sensor in vertex B may send readings that indicate a fault condition due to those readings exceeding certain limits, however such faulty readings do not necessarily affect the operation of vertex C. In other words, the graph representation in Fig. 3(a) does not reflect the dependencies in the actual system, which is also revealed by demonstrating that the vertices in Fig. 3(a) do not have an absolute dependence relation with each other. To better demonstrate the possible conflicts this representation may cause, we may add another observable vertex D in Fig. 3(b) which is represents a BITE that monitors vertex C. It is possible that vertices A and C are functional, which is indicated by a fault free vertex D, however the parameter measured at vertex A through vertex B may exceed the threshold for some other cause that is not represented in the graph, such as the ambient conditions in which the system operates.

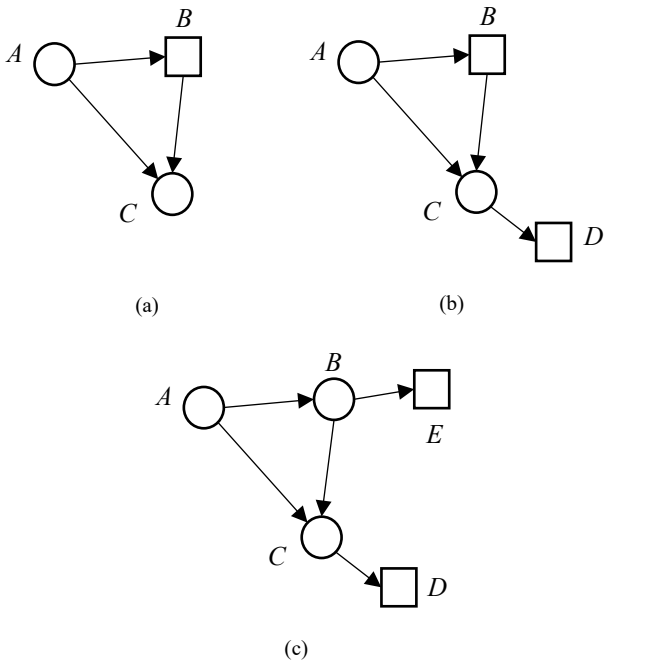

Fig. 3. Graph representations are given, where a parameter is measured by a vertex and compared by a threshold. The representation in (c) reflects the correct interpretation, which differentiates the measurement function and thresholding function as two separate vertices B and E respectively.

Thus, the actual source of a parameter, measuring a parameter and comparing that parameter to a threshold are three distinct functions that should be represented by separate vertices if necessary. In Fig. 3(c) the measurement function is represented by the vertex B while the threshold function is represented by the vertex E. It is noted that the operating conditions for the system, such as the parameters describing the ambient environment, should possibly be included in a graph representation.

B. Control Loops

One possible mistake when describing a control loop as a graph is to confuse the functional dependencies with the flow of information and material. A pool is considered as an example of a control loop, as depicted in Fig. 4(a) and two graph descriptions are given in Fig. 4(b) and Fig. 4(c). The control loop consists of the pool itself, a buoy to measure the liquid level and a pump that fills the pool as the liquid is drained by some other, uncontrolled process. The graph description in Fig. 4(b) is provided as an example of a possible misrepresentation, where the pool, represented by A, is the predecessor of the buoy (B), which in turn is the predecessor of the pump (C) and which is the predecessor of the pool (A) again. The reading is assumed to be coming from the buoy itself. There are several issues with this representation.

First; the output function, that is the liquid level, is depicted as the reading of the observable buoy vertex. However, the buoy is just a measurement device. A faulty liquid level does not have to be caused by the buoy, as explained previously.

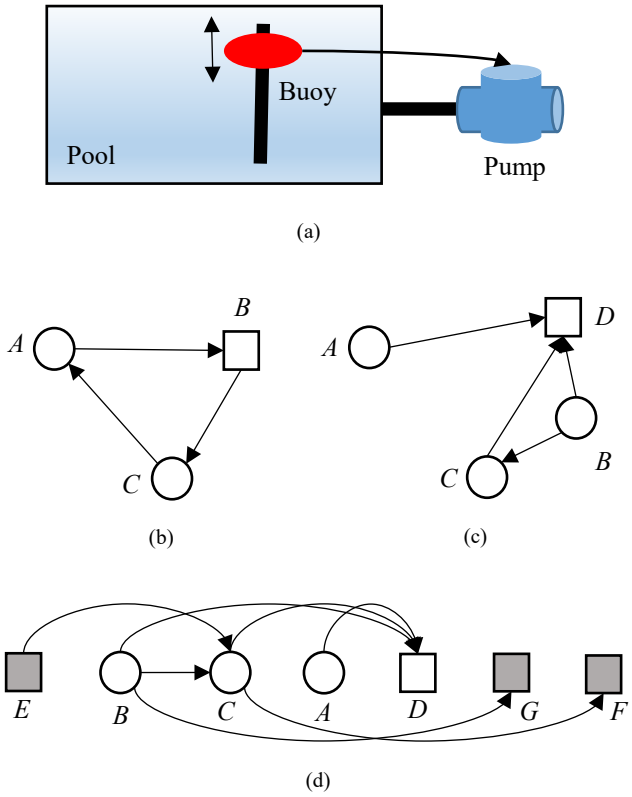

Fig. 4. A pool where the liquid level is measured by a buoy and controlled by a pump is depicted in (a). Graph representation in (b) demonstrates a faulty interpretation, where measurement and thresholding functions are represented by a single vertex and dependencies are misinterpreted to produce a loop. Graph representation in (c) shows the correct interpretation of the control loop. The rearranging of the graph into a topological ordering in (d) can provide a clearer view and reveal the need for extra BITE vertices, also shown on the graph as shaded vertices.

Second; the meaning of the pool vertex (A) is ambiguous. It might be representing the physical integrity of the pool, but then it should not be the predecessor of the buoy vertex: The buoy will fulfill its function and give a faulty reading if there is a crack at the bottom of the pool and the liquid is leaking.

Third; the observable vertex B representing the buoy should not be the predecessor of the pump, as the pump and the buoy may be functioning correctly even if the reading is faulty. However, if the buoy is faulty, this will certainly mean that the pump will not function correctly. Hence, the reading and the buoy must be represented separately.

Another representation given in Fig 4(c) eliminates these issues by clearing the ambiguities about what each vertex corresponds to. The functionality of the pump depends on the functionality of the buoy, which is not observable immediately. For the readings received from the buoy, the vertex D is declared. The fault condition of the readings depends on the buoy vertex B, the pool vertex A, which now corresponds to the integrity of the structure, and the pump C. The functionality of the buoy is not preceded by the pool vertex anymore. Thus, the control loop does not appear as a circle in the graph representation of the system. This new graph also shows the inadequate BITE replacement, because for a failure in the liquid level threshold no single reason can be isolated: Both the pump,

the buoy and the structural integrity of the pool are suspects in this case.

C. BITE Placement

If we consider the same example presented in Fig. 4(a), the inability to locate the source of the failure in the case of a faulty reading is obvious. More BITE vertices are necessary to successfully isolate the cause of the problem. The graph representation is reorganized into a topological ordering in Fig. 4(d) and BITE vertices are placed to assess the error isolation capability.

Focusing on the pump, two BITE vertices can be introduced, namely the power BITE (E) and the flow BITE (F). However, the power BITE necessitates declaring a power vertex. At this point the system operation and maintenance scenario should be considered: If the power supply is part of the system solution and requires monitoring for maintenance purposes, a power supply vertex distinct from the power BITE vertex is requited. Otherwise, the BITE vertex can be considered as the predecessor of the pump vertex. Flow BITE vertex, which measures the amount of liquid that passes through the pump, is the successor of the pump vertex and actually completes another missing component from the system description: The liquid inflow is a function of the system that is fulfilled during the successful operation, and it has to be observable if the functionality of the system is to be assessed by a fault manager. With both observable BITE vertices in place, the fault isolation is now possible for vertex (A).

It is noted, however, that introducing an observable predecessor to an unobservable vertex does not improve the fault isolation capability. Power failure can be considered as a very probable fault source for this particular example, and observing the power input to the pump may reveal in many cases that the failure in the flow rate is due to a power failure. However, a fault observed in buoy readings in vertex (D) can still originate from either the buoy vertex (B) or the pump vertex (C) if there is no failure observed in the power BITE vertex (E). A failure in vertex (C) is expected to cause a failure in both vertices (D) and (F) because of the absolute dependence principle. Moreover, a failure in vertex (B) will propagate through vertex (C) and again cause a failure in vertices (D) and (F). Thus, a new observable vertex (G), which depends exclusively on the buoy vertex (B), is required to isolate the failures in vertices (B) and (C).

To further demonstrate the utility of the proposed method, the Active Electronically Scanned Array (AESA) radar is considered as another example for explaining the BITE replacement. An AESA consists of multiple antenna elements with independent signal transmission and reception components. There are plenty of diverse array architectures described in the literature. For the BITE placement example, an AESA architecture with transmitted signal phase control and power amplification for each antenna element is adopted. Similarly, the received signal is digitized separately for each antenna element in the chosen architecture. The block diagram for a single transmission/reception (TX/RX) module of this generic AESA radar is given in Fig. 5, where external connections to central units providing power, control and reference RF signals are also visible.

The BITE example concerns monitoring the status of the RF signal power transmitted from the antenna at the output of the TX/RX module. The RF power at the output of various components in the TX and RX circuits can be monitored by introducing directional couplers to the circuit. However, directional couplers take up space and introduce RF signal power loss. Using a minimum number of such monitoring devices without compromising the fault isolation capability is the goal for this particular example. To this end, the directed graph representation of the TX/RX module given in Fig. 6, is utilized.

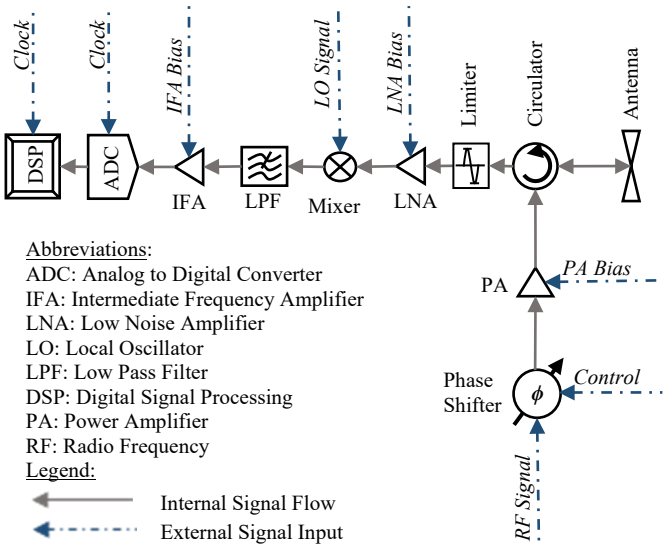

Fig. 5. The block diagram of a generic TX/RX module for an AESA radar consists of multiple active RF components. To ensure the radar signal is transmitted from the antenna with the required power, the signal level at the output of the power amplifier has to be monitored.

For this particular example, it is assumed that the power transmitted from the antenna is also coupled to the RX line such that it can be sampled by the ADC and used to monitor the transmitted power. Following the directed graph representation, this scenario amounts to monitoring the vertex which corresponds to the ADC. The dependence relation between the power amplifier output power, the antenna transmission power and the signal power at the output of each RX component indicates that a fault at the power amplifier output can be detected by monitoring the signal sampled by the ADC. However, this approach fails to isolate the fault to the power amplifier output; a fault at the sampled signal may be caused by a fault at any one of the components coming before the sampling. Hence, in order to isolate the fault at the power amplifier, it is unavoidable to introduce BITE dedicated to monitoring the power amplifier output.

Sampling the IF amplifier output to monitor the transmitted power at the antenna is not a viable solution due to a second failure mechanism: The connection between the power amplifier and the antenna might be severed or the antenna itself

might be physically damaged. In that case, the output of the power amplifier is not coupled to the antenna but reflected back into the RX channel. To prevent damage to the low noise amplifier, an RF limiter component is installed before the amplifier. The limiter is a non-linear device that reflects back the RF signal when the power level is above a threshold. The circulator, in turn, conducts the reflected power back to the power amplifier, which may damage the amplifier. The directed graph representation in Fig. 6 covers this failure mechanism as well. If the antenna failure is considered as a possible scenario and its detection is necessary, another BITE for monitoring the reflected power is required.

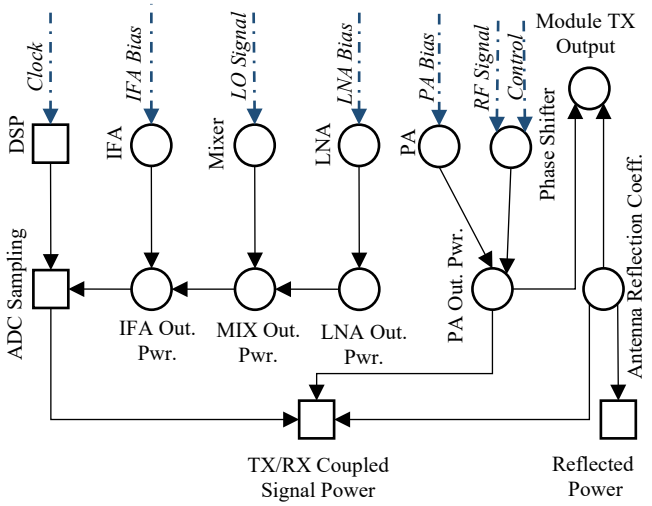

Fig. 6. The directed graph representation of the TX/RX module shows the absolute dependency relations between the components and functions realized by each component. The need for additional BITE can be determined by the directed graph representation, which reveals the fault isolation capability provided by the observable vertices.

It is noted that multiple such TX/RX modules are present in AESA radars. The fault state of the external signals depicted in Fig. 5 and Fig. 6 can be determined to a great extent by the overall fault states of the modules. All modules that are connected to the same external signal source has to enter a fault state before the external signal source itself is diagnosed as faulty.

D. *Graceful Degradation*

Some systems have more than one component that fulfill the same function, which enables a smooth change from the fully functional state to failure stage as more of components fail gradually. Such systems are said to have the graceful degradation feature. A detailed explanation of graceful degradation can be found in [14]. The Active Electronically Scanned Array (AESA) radar, which transmits and receives with multiple antennas simultaneously is an example of such systems. The combined action of multiple antennas enables the AESA radar to steer transmit and receive beams in any direction required by the task at hand, and each antenna contributes to the power of the transmit beam. Modern long range air defense radars can contain thousands of antenna elements. Out of thousands of antenna elements, if a fraction are out of order, the radar continues to function albeit with lower total transmit power and reduced beam steering capabilities. However, this does not necessarily take the radar out of operation. The radar may continue its operation until the aforementioned performance items are degraded to a point that is deemed unacceptable.

The graceful degradation provided by multiple elements involves a random process, where each element serving the function of the system may malfunction according to a probability distribution function. Thus, at any given time, a random selection of elements will be in a faulty state. If an output vertex is declared as the successor of all vertices representing these elements, the fault propagation rule will necessarily drive it to the faulty state most of the time even though the system is functioning within the expected specifications. For example, an observable vertex representing a minimum transmit power cannot be declared as the successor vertex to all other vertices representing the AESA elements. Any faulty element vertex would be a predecessor of the total output power vertex, which would be observed as not faulty and yield a contradicting error diagnosis.

Hence, graceful degradation behavior cannot be expressed in the graph representation of the AESA radar system, as it is provided by the combined action of multiple elements. An aggregation rule distinct from the fault propagation rule of the graph representation is necessary to handle the graceful degradation. In the next section, the two aggregation layers, one that translates the BITE readings into binary fault states and the other that evaluates the diagnosed errors are presented.

E. *Dynamic System Organization*

The directed graph representation of the system gives the snapshot of all dependency relations between system elements. In the course of reliability analyses, the reliability of a system for a given dependency configuration can be determined along with those components that are identified as single points of failure. To eliminate such critical points, it is possible to incorporate within the system a number of the same element. In such settings either the combined outputs of multiple elements provide the necessary input for the successor elements, which may correspond to graceful degradation, or redundant elements are taken into operation upon the failure of the primary element. The dynamic system organization refers to the second case, which amounts to modifying the dependency relations between elements.

Consider, as an example, a radar system where the transmit power is generated by a single solid state amplifier, which is determined as a possible single point of failure. As a solution, a second amplifier module, which is in stand-by mode during the normal operation can be introduced into the system. Upon failure of the first amplifier, the second amplifier is taken into operation by switching the RF circuitry to the amplifier. In the case of a failure in the first amplifier, the directed graph representation of the system is modified upon the switching of the second amplifier in place of the first one.

The significant point regarding the switching to the redundant element is the assessment of the necessity to

undertake the action. An automated fault management system can take the decision to switch to the redundant element upon evaluation of the fault diagnosis. It is noted that the directed graph representation can cover only one of the system configurations. Two separate graph representations are necessary to represent the change in the system configuration. This, however, is a matter of introducing a new vertex and edge list into the algorithm, without changing the functioning of the fault management algorithm. The aggregation layer described in the next section is capable of monitoring such single points of failure, making decisions regarding the system organization and reporting the results of the fault management algorithm even when the graph representation is modified.

V. Three Tiered Systems Analysis Structure

The graph description of systems is based on absolute dependence between constituents of systems and binary fault modeling. The fault propagation and fault diagnosis rules form a consistent logic rule set which can be processed by a search algorithm that runs over the directed graph representation of the system. To cast the readings from a system into binary fault states, an input layer that aggregates and evaluates those readings is required.

The input layer to the directed graph representation of systems may utilize many functions to translate and when necessary aggregate the readings into binary fault states. A commonly used function might be thresholding, which is already mentioned in Fig. 3. In that example the fault state of the observable vertex was explained as the result of the thresholding of a reading from the BITE. The translation to binary fault states may also include latching, which will hold an error state until a specific action is taken besides the reading returning to nominal operation point. Inconsistent readings from one or more sensors may be aggregated over time to obtain reliable fault state information. Any operation that requires a statistical analysis or nonlinear behavior such as latching is placed in this input layer.

The two tiers described so far can be used to diagnose a faulty vertex in the system's graph representation; afterwards, this diagnosis has to be conveyed to end-users for maintenance purposes. Moreover, for those systems which present ambiguous fault isolation due to system organization, i.e. inadequate BITE coverage, other mechanisms that take into account the maintenance logs and assign probabilities amongst possible points of failure can be implemented. The graceful degradation can be handled by aggregating the fault states of the many vertices that represent the outputs of the system. Such mechanisms are located in an output layer. Together with the input layer, the directed graph layer and the output layer, a three-tiered fault management architecture is formed, which is depicted in Fig. 7. It is possible to aggregate more than one reading from a system in tier 1 to assign fault state to a vertex. It is also possible to aggregate the fault diagnosis for multiple vertices into a final fault assessment in tier 3. Incidentally, the representation for the inner connections of tier 1 and tier 3 resemble the representation of an artificial neural network, which may be a possible method to aggregate readings into binary fault states.

It is also noted that each vertex in the graph representation can be assigned other properties besides the ones necessary for the fault management. Class information such as hardware, software or service can be utilized on tier 3 to provide decision criteria for maintenance actions. Similarly, information on the line-replaceable and shop-replaceable parent units can be a vertex property. Tier 3 and all the auxiliary information on the vertices form a translation mechanism between the system representation on tier 2 and the established processes of the systems integration and maintenance enterprises.

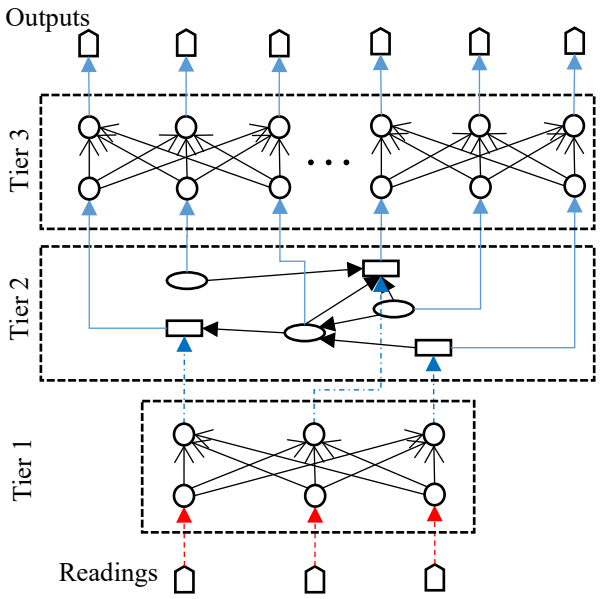

Fig. 7. The three tiered fault management structure has the directed graph representation of a system in the middle tier. The first tier translates readings from the BITE into binary fault states, while the last tier aggregates results of the fault management algorithm.

VI. Connections with the Other Systems Representation Methods

In the literature there are a variety of systems representation methods with emphasis on different aspects of systems engineering activities. The proposed method shares certain features with these systems representation methods. Two methods are emphasized here; a comprehensive comparison of different methods with the proposed technique and with each other requires the volume of a whole manuscript.

A. Design Structure Matrix (DSM)

DSM is proposed as a method for managing the design of complex systems by representing the precedence of the specifications of a system's components with respect to each other. In other words, the goal is to determine the order with which specifications for each system component have to be fixed and point out early in the design effort the necessary iterations which arise due to mutual precedence between system components [15]. The relation between the directed graph representation of systems and the DSM arises from the fact that a directed graph can be represented by an adjacency matrix. In other words, a DSM where precedence is represented by 1 and

the rest of the matrix elements equal to zero is equivalent to a directed graph.

DSM can represent the feedback relations between constituents of a system; analysis methods are present to group such constituents with feedback relations together into subsystems [16]. The DSM method presents certain insights, such as the inclusion of system constituents and outputs in the design structure. These system outputs are not limited to system functions; they may also cover built-in test results. Compared to the DSM technique, the new method proposed here establishes the absolute dependence principle as a guideline for determining the analysis level which yields elementary constituents of a process or a system design. Moreover, based on the fault management algorithm presented in this paper, DSM representations can now be analyzed to determine where process monitors should be placed in an organization, similar to determining BITE locations in a system for unambiguous fault diagnosis. The graph representation may augment the DSM modeling by providing a user-friendly toolset which facilitates automated manipulation and analysis tools through the establishment of the absolute dependence principle, which are observed as missing from the current implementations of the DSM according to [17].

B. Unified Modeling Language (UML)

UML is proposed as a modeling tool for complex real-time systems in [18] and [19]. The vertex presented in this paper corresponds to a capsule in UML, however the capsule implements the complete behavior of a part of the system by connecting together the ports of a capsule by a script or through equations modeling the energy and materials flow and transform. Capsules are connected by connectors, which correspond to edges presented in this paper.

The graph representation with absolute dependence relation differs from the UML by the lack of a complete system model and the presence of a constraint that limits the size of a vertex. A capsule may model a wide array of functions realized by a portion of a system, however there is no criteria to claim that a proposed capsule is "elementary". A number of capsules can be merged into a single capsule as long as the ports and connectors are tractable by the designer. Capsule size and contents are likely determined by the work practices of design teams within a company, which may differ radically between different companies. This point, that UML does not facilitate communication between stakeholders in a project as it lacks common building blocks, is presented in [20] as one of the reasons that prevent a wider adoption of UML in the industry. A reason behind this observation might be the lack of a criterion for declaring a capsule as elementary. The criterion for an elementary vertex, which is the fulfillment of the absolute dependence and fault propagation principles, may augment UML and contribute to wider adoption of UML and similar tools in systems engineering practice.

VII. Summary and Conclusions

A novel systems representation method based on directed graphs with absolute dependence principle is developed and its utilization in the implementation of the fault management algorithm is presented in this paper. The concept, which is initially developed in the course of systems engineering activities concerning (AESA) radars, has applicability over a wide variety of systems and processes. Many ideas underlying the novel method were present in the literature in distinct times and aimed at different applications. The novelty of the technique presented in this paper is in the synthesis of these diverse ideas through the clarification of the nature of graph representation, introduction of the absolute dependence principle and the definition of the elementary component of a system. The fault management algorithm proposed in this paper is readily useable in the existing systems.

The systems description as directed graph provides a rigorous framework for representing a system. The rules of the systems representation method are consistent and there is no ambiguity in the denotations of graph elements. As such, this systems description can be manipulated by algorithms with minimal intervention from designers, paving the way to automation of a wide range of tasks that are currently handled by reliability engineers and systems engineers manually. Instead of case by case definition of relations between various scenarios, only the absolute dependence relationships between the system elements and functions can be defined according to the method described in this paper. Then, the fault tree analysis and FMECA can be obtained automatically utilizing the directed graph representation based on absolute dependencies. Methods for conducting such automated reliability analysis will be developed in the future. Connections between the graph representation with absolute dependence and the other systems representation methods will also be investigated in greater detail.

References

[1] *Systems Engineering Guide*, MITRE Corporation, 2014, ISBN 978-0-615-97442-2

[2] Kuan-Min Lee, Ruey-Shi Chu, Sien-Chang Liu, "A built-in performance-monitoring/fault isolation and correction (PM/FIC) system for active phased-array antennas," *IEEE Transactions on Antennas and Propagation,* vol. 41, no 11, Nov 1993.

[3] S. A. Lapp, G. J. Powers, "Computer-aided synthesis of fault-trees," *IEEE Trans. Reliability*, vol. R-26, no. 1, pp. 2–13, Apr. 1977, 10.1109/TR.1977.5215060.

[4] H. E. Lambert, "Comments on the Lapp-Powers computer-aided synthesis of fault trees," *IEEE Trans. Reliability*, vol. R-28, no. 1, pp. 6–9, Apr. 1979, 10.1109/TR.1979.5220452.

[5] S. A. Lapp, G. J. Powers, "Update of Lapp-Powers fault-tree synthesis algorithm," *IEEE Trans. Reliability*, vol. R-28, no. 1, pp. 12–15, Apr. 1979, 10.1109/TR.1979.5220455.

[6] S. Colombano, L. Spirkovska, V. Baskaran, G. Aaseng, R. S. McCann, J. Ossenfort, I. Smith, D. L. Iverson, M. Schwabacher, "A system for fault management and fault consequences analysis for Nasa's Deep Space Habitat," *AIAA SPACE 2013 Conference and Exposition, AIAA SPACE Forum,* (AIAA 2013-5319), 10.2514/6.2013-5319.

[7] F. Mhenni, N. Nguyen, J.Y. Choley, "SayeSysE: A safety analysis integration in systems engineering approach," *IEEE Systems Journal*, vol. 12, no. 1, Mar. 2018, 10.1109/JSYST.2016.2547460.

[8] P. Luthra, "BIT analysis: how to approach it," in *Proceedings 1990 Annual Reliability and Maintainability Symposium,* Los Angeles, CA, USA, 23-25 Jan. 1990.

[9] S. Khan, P. Phillips, I. Jennions, C. Hockley, "No Fault Found events in maintenance engineering Part 1: Current trends, implications and organizational practices," *Reliability Engineering and System Safety*, Vol. 123, Mar. 2014, pp. 183–195.

[10] S. Khan, P. Phillips, C. Hockley, I. Jennions, "No Fault Found events in maintenance engineering Part 2: Root causes, technical developments and future research," *Reliability Engineering and System Safety*, Volume 123, March 2014, pp. 196-208.

[11] *Guidelines for Chemical Process Quantitative Risk Analysis, 2nd ed. pp. 661-670,* Center for Chemical Process Safety, American Institute of Chemical Engineers.

[12] J. L. Gross, J. Yellen, "Handbook of graph theory," CRC Press, ISBN 978-1-58488-090-5.

[13] F. Balmas, "Displaying dependence graphs: a hierarchical approach," *in Proceedings Eighth Working Confrence on Reverse Engineering*, 2-5 Oct. 2001, 10.1109/WCRE.2001.957830.

[14] T. Saridakis, "Design patterns for graceful degradation," *Transactions on Pattern Languages of Programming*, vol. 5770, Springer, Berlin, Heidelberg 2009, 10.1007/978-3-642-10832-7_3.

[15] D. V. Steward, "The design structure system: A method for managing the design of complex systems," *IEEE Transactions on Engineering Management,* vol. EM-28, no. 3, Aug. 1981, pp. 71-74, 10.1109/TEM.1981.6448589.

[16] T.R.Browning, "Applying the design structure matrix to system decomposition and integration problems: a review and new directions," *IEEE Transactions on Engineering Management*, vol. 48, no. 3, Aug. 2001, pp. 292-306, 10.1109/17.946528.

[17] T.R.Browning, "Design structure matrix extensions and innovations: A survey and new opportunities," *IEEE Transactions on Engineering Management*, vol. 63, no. 1, Feb. 2016, pp. 27-52, 10.1109/TEM.2015.2491283.

[18] B. Selic, J. Rumbaugh. "Using UML for modeling complex realtime systems," ObjecTime Limited/Rational Sofware Corp. white paper, March 1998.

[19] C. Secchi, C. Fantuzzi, M. Bonfe, "On the use of UML for modeling physical systems," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, 18-22 April 2005, 10.1109/ROBOT.2005.1570731.

[20] M. Petre, "UML in practice," in *2013 35th International Conference on Software Engineering*, 10.1109/ICSE.2013.6606618.

R. F. Tigrek received the B.Sc. degree in 2002 and M.Sc. degree in 2005 in electrical and electronics engineering from The Middle East Technical University, Ankara, Turkey, and the Ph.D. degree in electrical engineering from the Delft University of Technology, Delft, the Netherlands in 2010.

From 2002 to 2005 he was Test Engineer at Aselsan A.S., Ankara, Turkey, and from 2011 to 2018 Radar Systems Engineer again at Aselsan A.S. He was responsible for developing system scenarios and interface control documents, algorithms for calibration and fault management, and integration, verification and validation of radar systems with active electronic scanning antennas. Since 2018 he is postdoc at the Signal Processing Systems Group in the Department of Electrical Engineering, Eindhoven University of Technology, Eindhoven, the Netherlands.