

# Computer-Aided Identification of System Vulnerabilities and Safeguards during Conceptual Design<sup>1,2</sup>

Jane T. Malin NASA Johnson Space Center 2101 NASA Road 1 Houston, TX 77058 281 483-2046 jane.t.malin@nasa.gov	David R. Throop The Boeing Company 2100 Space Park Drive Houston, TX 77058 281 460-8415 david.r.throop@boeing.com	Land Fleming Hernandez Engineering 17625 El Camino Real Houston, TX 77058 281 483-2055 land.d.fleming1@jsc.nasa.gov	Luis Flores Lockheed Martin Space Operations 2400 NASA Road 1 Houston, TX 77058 281 333-6423 luis.flores@lmco.com
--	--	--	--

*Abstract*—This paper presents an approach to helping engineers to begin to address safety risks and to capture information during conceptual design, for use in early hazard analysis. Complex system accidents are hard to understand when they happen and hard to identify before they happen. Our goal is to aid early identification of this type of potential accident. We describe progress in developing a prototype Hazard Identification Tool to help engineers capture design features of systems and components. We describe terminology for classifying and describing system functions, problems, vulnerabilities and safeguards. This terminology can be applied to hardware, software and human factors. Conceptual design information is mapped to a library of component models, to support generation and simulation of system accident scenarios. We describe a strategy for identifying potential system accident scenarios, based on an analysis of types of sequences of events in system accidents. We describe tools for scripting scenarios and mapping to a hybrid simulator. The generic component library of the simulator is used to construct component-connection models whose behavior can include a broad variety of types of performance problems and hazards. We illustrate the strategy with a design case with a biological water processing system.

## TABLE OF CONTENTS

<b>1. INTRODUCTION</b> .....	<b>1</b>
<b>2. HAZARD ANALYSIS USAGE CONCEPT</b> .....	<b>2</b>
<b>3. FUNCTION AND PROBLEM ONTOLOGIES</b> ..	<b>4</b>
<b>4. IDENTIFYING SYSTEM ACCIDENTS</b> .....	<b>7</b>
<b>5. SIMULATING SYSTEM ACCIDENTS</b> .....	<b>8</b>
<b>6. CONCLUSIONS</b> .....	<b>11</b>
<b>REFERENCES</b> .....	<b>11</b>

## 1. INTRODUCTION

In recent times, there has been increasing interest in predicting and preventing system accidents, as startling system accident cases have accumulated [19]. Accidents of

this type include an Osprey helicopter crash [9], Therac-25 failures [11] and the ValueJet 592 crash [10]. Complex system accidents are hard to understand when they happen and hard to identify before they happen.

A hardware designer is usually aware of the significant consequences of planned operations for the anticipated states of the system. It is more difficult to anticipate all reasonable sequences of operations to which the system might be subjected, or to anticipate all the off-nominal system states in which a planned operational sequence is inappropriate. Likewise, designers of operational procedures or control software and human operators of the implemented system may be far removed from the details of physical component interactions within the system. It is difficult to anticipate the full consequences of unplanned operations in response to component failures or consequences of normal operational procedures conducted in unusual circumstances.

Our goal is to aid early identification of potential accidents. System accidents are brought about by complex unexpected interactions between failures in systems during operations, and by inappropriate responses to developing accidents. In system accidents, complex system and process failures typically interact with failures of controls, safety systems, software and human operators. Analysis is needed of propagation of system threats in operations, due to interactions among subsystems and with resources (power, thermal, data). Analysis is also needed of safeguard subsystems for managing these threats, including controls, safety systems, software and human operations.

We are developing an approach to help engineers identify hazards, threats, counteractions and safeguards during conceptual design. Our goal is to use this information for graph analysis and simulation, for early hazard analysis that considers operations, with some features of operating and support hazard analysis (O&SHA) and hazard and operability analysis (HAZOP) [4].

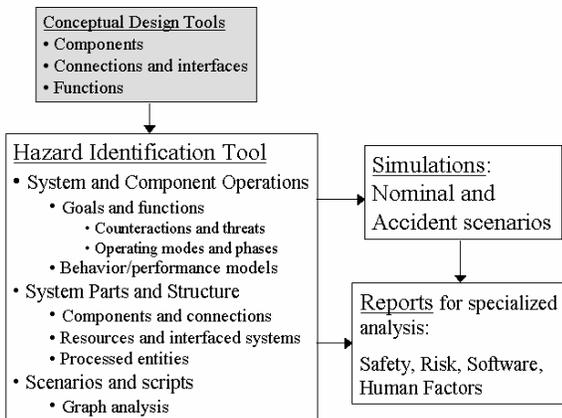
<sup>1</sup> 0-7803-8155-6/04/\$17.00©2004 IEEE

<sup>2</sup> IEEEAC paper #1181 Version 3.

The prototype Hazard Identification Tool (HIT) helps an engineer to capture design information top-down from the system level, with emphasis on system functions and structure. HIT provides vocabularies and library-based help for identifying and analyzing system functions, threats and safeguards. HIT supports developing component-connection models of systems, for model-based hazard analysis (MBHA). HIT libraries include component models that can be mapped to a simulation tool. By mapping selected HIT data to models in the simulation tool, it is possible to simulate accident event propagation during operations. The approach uses a hybrid modeling and simulation tool, which includes a library of generic component behavior models, with a broad variety of types of performance problems and hazards. HIT will also help engineers identify, develop and script candidate accident scenarios for simulation, based on types of sequences of events in system accidents. This paper provides a progress report on development of these concepts and prototypes.

## 2. HAZARD ANALYSIS USAGE CONCEPT

Figure 1 show as usage concept for HIT and the simulation. Using HIT and its supporting terminology and libraries, the engineer identifies system information that is relevant to operations and hazards. HIT data is used for simulation, and to generate reports for analysis by engineers specializing in software, safety and human factors.



**Figure 1 - MBHA System Usage Concept**

### *HIT Usage Concept*

HIT provides knowledge acquisition forms and supports selection of types of components and entities, functions and problems from libraries. Use of these libraries helps the engineer consider potential problems and safeguards that are associated with types of functions and entities. The following tasks illustrate step-by-step use of HIT.

#### Import system data from conceptual design tool

Optionally, import some system data from a conceptual design tool, such as DDP [2] or CUP [1]: goals/functions,

components/subsystems, and their structure of connections. While CAD-oriented tools such as CUP focus on low-level components and shapes, HIT, like DDP, emphasizes system-wide functions and interfaces, from the top down.

#### Specify system-level information

1. List system components/parts, interfaced systems and supporting resources (Some component-level information could be specified during this step.)
  - Import from a design tool or create instances
  - Classify components/parts with component types from library (with information used for simulation)
2. List major system operating modes, operations and operational phases
3. Specify system goals hierarchically (Identify related goals, super-goals, sub-goals)
  - Functional goals
  - Counteraction goals, to manage threats to or within system, from system, or passing through system
4. Define goals, using the Function class – both functions and threat counteractions
  - Name and classify function verb and objects, using the function and entity type hierarchies
  - For counteraction, classify as safeguard type: Isolate/protect, monitor/indicate, control/command, compensate/apply redundancy, or recover/maintain
  - Identify agents, participants and contributors (components/parts, materials, interfaces)
  - Identify conditions for activation and performance
  - Identify and evaluate function effects (importance, effectiveness, potential problems)
    - For performance, provide a model (expressions, equations or tables)
    - For problems, define using the Problem class structure

#### Graphically specify system structure

Define system structure: Connect components and interfaced systems to define relationships. (This supports generation of an interface file that is used to translate the connection information into a simulation model.)

#### Specify component level information

Component instances were identified and classified in the system-level specification. Component classes in the library include typical functions, potential problems and performance models.

- Some components were identified as contributors to specific system goals
- Some components were identified as involved in specific system threats (hazards and vulnerabilities)
- Specify further information for each component, same as for system (parts, operating modes, goals/functions)

#### Script nominal and accident scenarios

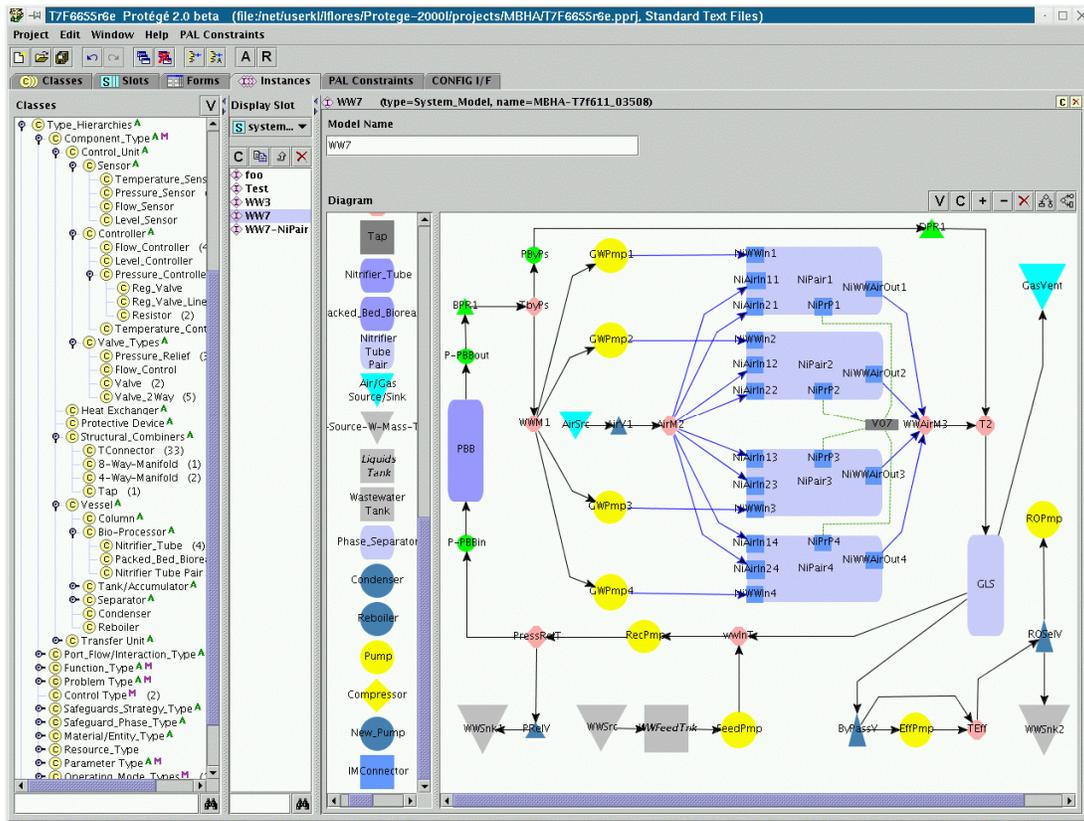
Develop scenarios and scripts to guide simulations focused on system accidents.

- Configure elements of potential accident scripts for operations, based on graph analysis and identified functions, problems and safeguards. Select paths and event sequences that include hazards and safeguards.

*Biological Water Processor System Example*

The HIT prototype is implemented in the open-source Protégé knowledge acquisition tool [18]; a system that provides intuitive information entry forms for knowledge acquisition. We have enhanced the basic Protégé tool with automatic data propagation and a tool for specifying simulation scripts. Because implementation follows design, some details of the example case have not caught up with the design described in this paper.

Figure 2 shows use of HIT to capture system components and structure for a biological water processor (BWP). The BWP is a subsystem of a Water Recovery System that was designed, built and tested at NASA Johnson Space Center [3]. The BWP takes in raw gray water and reduces the organic carbons and the ammonium. This water is further processed in downstream systems that include a Reverse Osmosis (RO) unit, a Post Processing Subsystem and an Air Evaporation Subsystem for recycling water from RO brine. The BWP consists of a continuous loop with two biological processors and a Gas Liquid Separator (GLS) in a single path. A recycle pump keeps the flow through the system at the desired level.



**Figure 2 - Biological Water Processor Components and Connections**

The Packed Bed Biological Water Processor (PBB) is a tank packed with ceramic saddles that support the growth of anaerobic microorganisms. These microorganisms use nitrates and nitrites provided by the nitrification process, next in the loop, to reduce the organic carbon in the wastewater (WW). The other biological processor, the Nitrifier, consists of four pairs of polypropylene tubes that contain aerobic microorganisms. These microorganisms convert the WW ammonium to nitrates and nitrites. Both air and water are pumped into the Nitrifier tubes. The GLS separates the Nitrifier effluent gases and liquids, and vents the gases. In steady state mode, WW is fed back to the PBB while the stream is tapped at the GLS to feed the RO. About

95% of the liquid is recycled back into the PBB. The inflow and the outflow are balanced to maintain a constant flow through the two biological processors.

A new design can be quickly outlined with the Protégé diagram capability. The system presents a list of components that may be dragged from the vertical palette into a canvas to create an instance. Dragging the mouse pointer from one component to another in the graph creates connections. Figure 2 displays the BWP graph, the palette, and part of the hierarchical component library display in Protégé. HIT supports development of submodels, called “inner models”. The Nitrifier is made up

of four paired nitrifier tubes with common structure. Figure 3 displays a submodel of a nitrifier pair.

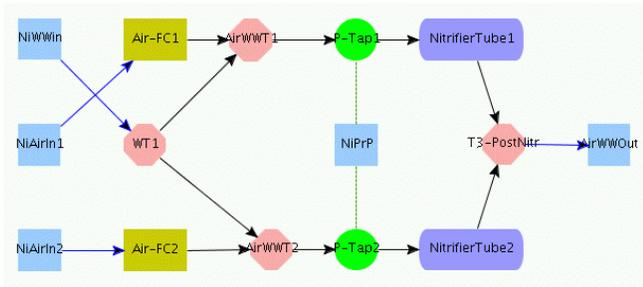


Figure 3 - Inner model of Nitrifier Pair

Figure 4 shows a graphical interface that is being developed for specifying system characteristics for the BWP. Windows

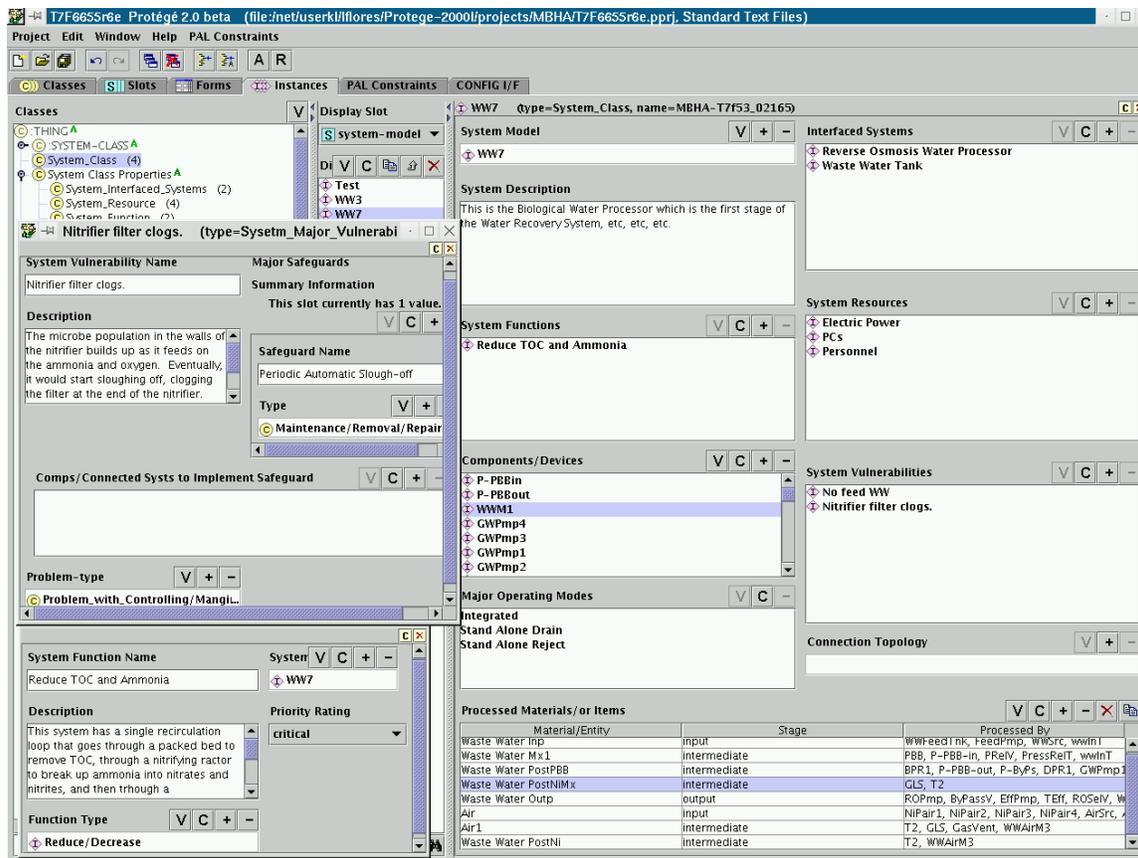


Figure 4 - Specifying Water System Attributes, with Functions and Vulnerabilities

#### Goals Describe Functions, Vulnerabilities and Safeguards

Specifying counteraction goals helps engineers identify safeguards, hazards and vulnerabilities. Counteractions (safeguards or mitigations) are designed to protect the system against vulnerabilities, which are unacceptable system level weaknesses or occurrences. The three types of counteraction goals clarify the distinctions between mishaps, hazards and threat propagation:

open on the left illustrate forms for specifying attributes for a function and for a vulnerability (threat), with its safeguards. These windows contain selection boxes for mapping system attributes into types in the appropriate class hierarchies in the ontology.

### 3. FUNCTION AND PROBLEM ONTOLOGIES

We are developing ontologies for capturing system design information. Ontologies are knowledge structures with standard terminology. The ontologies provide hierarchies of types/classes with default information to define the classes. The type hierarchies and class structures provide standardized terminology and a standard framework for information capture in early design.

1. Counteract Mishap or Functional Failure: manage threat to or within vulnerable entity
2. Counteract Hazard: manage threat from entity
3. Counteract Threat propagation: manage threat passing through entity

Functions and safeguards, mishaps and hazards can be classified and described by using the ontologies. The Function class (summarized in the system-level specification in Section 2) provides a structure for defining

both functions and safeguards. The Problem class has an attribute structure that closely resembles the function class, defining conditions and effects for problem occurrence, and evaluation of their prevalence and severity. Designers consider hazards and off-nominal behaviors, and both standard and unusual problems.

The function and entity hierarchies work together to classify verb-object function phrases, which are used in function analysis during system design. The current type hierarchies for functions and entities are shown in Table 1 and Table 2.

*Developing the Ontologies*

The ontologies and libraries for HIT include interrelated hierarchies of types of functions, entities, problems and safeguards. To support use by several specialty disciplines through the life cycle, the ontologies cover not only equipment and materials but also control and operations by

software and humans. The terminologies are drawn from diverse sources. The National Institute of Standards and Technology has contributed to development of a unified function ontology, and this ontology of functions and “flows” has served as starting point for our work on functions and entities [5]. We have also used concepts from Modarres [16] and Kitamura and Mizoguchi [8] and Norman [17]. The starting point for our problem ontology was HAZOP terminology [4,21]. Work by Hollnagel [6] served as a starting point for our safeguard representation. When possible, we have converted specialty terms into terminology that would be understandable to a design engineer.

These ontologies have been refined and mapped to the space domain by parsing and analyzing International Space Station (ISS) documents. One source is the ISS Reliability Block Diagrams (RBDs).

**Table 1.** Function/Capability Type Hierarchy

Level 1	Level 2	Levels 3, 4 (:) and 5 (--)	Example Mappings
Process	Convert		transform, process, condition
	Increase or Decrease	Produce/increase; Reduce/decrease; Use	generate, create; remove; consume
	Combine or Separate	Combine; Separate	mix, blend; extract, divide, remove
	Damage	Damage; Give way/destabilize	destroy, break; collapse, yield
Place	Hold	Store; Carry; Support/Stabilize; Secure	contain; channel; reinforce; restrain
	Shift	Send: Provision, Export; Release; Free	provide; disperse; spill; disengage
		Receive/import/collect; Shift in place; Transfer	capture; circulate; move, transmit
	Arrange	Isolate; Expose	shield, block; unlock, provide access
		Position; Displace	place, align, orient; dislocate, slip
		Assemble/order/set up; Disassemble	install, configure; unpack, unstow
		Connect; Disconnect	attach, fasten; demate, undock
		Cover; Uncover	coat, line; strip, remove
Serve/support (object can be function)	Provide (service)		support, accommodate, supply
	Withhold (service)		retain, hold back, withhold
	Require (service)		need, require, demand
	Forego (service)		waive, give up, relinquish
Energize/drive (object can be function)	Propel/pressurize	Increase force; Reduce force	pressurize, pump; decompress
	Heat	Increase heat; Reduce heat	warm; cool, refrigerate
	Power	Increase power; Reduce power	energize, charge; discharge
	Illuminate/radiate	Increase radiation; Reduce radiation	light, irradiate; dim, darken
Control/manage/perform (object can be function)	Inform/decide	Communicate: Indicate -- Conceal; Interact -- Intercept	show; hide; request; block
		Process or Ignore: Monitor; Understand; Determine; Ignore/mistake	sense; interpret; decide; dismiss
	Control/maintain (state)	Achieve; Maintain; Avoid; Recover	attain; preserve; prevent; correct
	Direct (act on)	Manage; Assign; Regulate/guide/modulate	supervise; designate; adjust
		Command: Select; Enable—Disable; Start--Stop	switch; allow, inhibit; actuate, cancel
	Perform/act/respond	Execute; Slip/Err	operate; mistake, omit
	Coordinate	Correlate or Conflict: Correlate; Conflict	synchronize; clash, compete
		Assist or Interfere: Assist; Interfere	help; interrupt
	Reward or Punish: Reward; Punish	motivate; discourage	

The RBDs contain a directed graph of functions and entities, from high-level functions to individual pieces of equipment. Each functional node has a description, which is usually an imperative sentence – e.g. “transmit voice communication” or “introduce O<sub>2</sub>”. Another source is the ISS Flight and System data books [7] that contain a large set of procedures. Many of the procedure titles are imperative sentences, e.g. “apply electrical power to specified load” or “configure p4 power module for proximity operations”. Using an English-language parsing program, the descriptions and titles were broken into noun and verb phrases. We generated lists of nouns and verbs that were not mapped to types in the ontology. We generated charts showing the parent/child relationships of the hierarchy, and counted the number of times each type appeared, along with a listing of examples and counts of the associated entities. This showed us which portions of the hierarchies were most heavily used. This information was used to selectively expand the word mappings for types, and to reorganize and expand the coverage of the type hierarchies.

#### Function Verb Type Hierarchy

Parts of this function verb hierarchy can be two to five levels deep. Example words that map to the deepest levels are shown in the right hand column.

The primary function types Process or Place entities. Three other Level 1 types, Serve/support, Energize/drive and Control/manage/perform, can be meta-functions, because they can act on function entities. The type hierarchy is designed to include both positive and negative versions of capabilities where it is appropriate (e.g., collapse, dislocate, hide, omit, compete). We expect these contrasts to be useful in identifying some types of functional failures. The Control/manage/perform ontology is designed to cover hardware, software, human and organizational domains.

#### Entity Type Hierarchy

Entities can serve as objects in function phrases. Parts of this hierarchy can be two to four levels deep. Example words that map to the deepest levels are shown in the right hand column. The functional type hierarchy influenced the definition of types of equipment entities. A library of specific instances such as materials can include hazard attributes, such as Material Safety Data Sheet information. We are exploring the interrelationships between the function and entity type hierarchies.

**Table 2.** Entity Type Hierarchy

Level 1	Level 2	Levels 3 and 4 (:)	Example word mappings
Objects	Life forms		microbe, person, plant
	Physical objects	Systems as units	hardware, payload, kit
		Parts	accessory, plate, shaft
		Equipment:	
		:Processors	filter, reactor, extinguisher
		:Storage/carriers	bottle, platform, wire
		:Tools/arrangers	rope, clamp, gripper, wrench
		:Control Equipment	sensor, marker, switch
		:Energy Equipment	pump, radiator, battery, thruster
	Material units/flows		oxygen, ammonia, moisture
	Mixtures/collections		air, gray water, trash, oil
	Object structures		assembly, array
Locations/places			module, zone, path, worksite
Information/signals	Information units/flows		number, pixel, setpoint
	Information groups		limits, defaults, band, sample
	Information structures		vector, software, procedure
Energy/power			heat, force, electricity, light
States	Variables/parameters		rate, pressure, level, duration
	Events/states		bump, exposure, arrival
	Behaviors/performances		growth, discharge
Functions	Services/capabilities		communication, access, handling
	Demands		load, requirement, need

### Problem Type Hierarchy

A designer can use the problem type hierarchy to consider and classify threats, vulnerabilities and failure modes. The current type hierarchy for problems is shown in Table 3. This hierarchy can be three to four levels deep. Example words that map to the deepest levels are shown in the right hand column.

The two main types of problems are Conditions or causes (which can be used as model input in accident scenarios), and Results or effects (which can be produced in accident event sequences as model output). The Condition problem types include both hazardous exposures and deviations from conditions needed for performance of functions. The Result types include both state consequences and problems with performing functions.

**Table 3.** Problem Type Hierarchy

Level 1	Level 2	Levels 3 and 4 (:)	Example word mappings
Condition/cause	Exposure/deprivation	Electrical/electromagnetic stress	load, shock
(can lead to bad result)	(too much, too little)	Thermal stress	load, shock
		Radiation stress	load, shock
		Acoustic stress	load, shock
		Mechanical/vacuum/microgravity	load, abrasion, impact
		Chemical/biological	load, shock, toxic accumulation
		Signal/information	overload, shock
	Deviation from required	Equipment property	vibrating, too small, misaligned
	(too much, too little, wrong)	Materials property	inadequate, excess, wrong temp.
		Energizing/driving property	
		:Pressure, Thermal, Power, Radiation	inadequate, excess
		Management/controls property	
		:Communication	missing, obscured
		:Processing	inadequate, misinterpretation
		:Action/operation/response	wrong, delayed
		:Coordination	conflicting, unsynchronized
Result/effect	Problem with State	Equipment function or structure	block, deform, leak, corrode
	(damage, failure, degradation)	Materials	noise, contamination, conductivity
		Life forms	injury, infection, aging
		Environment/locations	fire, explosion
		Energy/drives	outage, surge
		Information	erasure, corruption
	Problem performing function	Error (act when inappropriate)	unauthorized, improper, wrong
		Omission (when permitted/commanded)	failed, missed, left out, skipped
		Insufficiency (rate, time, quantity, quality)	slow, too little, degraded

## 4. IDENTIFYING SYSTEM ACCIDENTS

We are developing a strategy for finding challenging system accident sequences. It will be based on graph analysis of effects propagation, and features that are typical of system accidents. A typical system accident involves complex behavior that leads to misinterpretation, which then leads to damaging responses.

In typical system accidents, a primary failure interacts unexpectedly with other system events, defeating a safeguard that was not designed (or maintained in readiness) to handle the complex version of the failure. Human operators, software and instrumentation are frequently implicated in the accident. Their tools and procedures are not prepared to help identify or block the complex problem.

Our analysis of system accidents has led to a three-phase theory of system accidents.

1. Complexity surrounding failures
  - Combinations and synergistic effects: common causes, canceling failures, side effects, command combinations and timing
  - Interactions in dynamic complex trajectories or histories: causes and effects that are distant in time (e.g., interacting systems, stored potentials), or compensating mechanisms
2. Misinformation and misinterpretation, due to complexity
  - Missing information, either concealed or ignored (possibly due to overload)
  - Wrong information, due to a misleading situation or misinterpretation of information
3. Damaging response, from complexity and misinformation
  - Misapplied or not applied, due to ‘silently’ violated assumptions (operational, coordination), failure to

stop when not helping or shortsighted response to wrong priority

- Applied but not available (deactivated or ‘silently’ unavailable due to failures of build, maintenance, configuration)

We plan to construct accident scenarios by elaborating safeguard scenarios that the engineer identifies. A designer of hazard countermeasures can identify cases that test the safeguard. We plan to begin with scenarios that test the successful operation of a safeguard, then complicate them with types of complexity, misinterpretation and safeguard failures. Graph analysis will help identify combinations of failures and failed counteractions that could potentially result in a system accident. Selection of complicating factors can be influenced by prevalence and importance ratings for functions and threats, and by search backward in the model structure from mishaps.

## 5. SIMULATING SYSTEM ACCIDENTS

### *Simulation and Models for Analyzing System Accidents*

Simulations for early design should provide a quick look at what could happen in important operations. System accidents are characterized by propagation and combination of threats during system operation. Sequences of system events can be investigated in fast scenario-based simulation of operations. Scenarios should focus on interactions between functional behaviors of major subsystems, components and resources. Simulations should support investigating threat propagation that crosses system boundaries and connections, and even changes system connections or structure.

Experience from past simulation projects [14] has shown that it is difficult for engineers to anticipate consequences of operations that change the state of fluid flows in a system. Unanticipated and counterintuitive consequences may be spatially remote from the component directly affected by such an operation. Potential system flows can be dependent on the historical context of the operation, so the consequences may be temporally as well as spatially remote.

The behavior of components should be modeled in HIT as abstractly as possible, focusing on functions and functional failures. Structurally, systems should be represented as graphs. Models should be able to represent the types of safeguards and simulate the chains of events involved in system accidents: cascading failures, inappropriate system reconfigurations, and undesirable global or indirect effects.

### *CONFIG hybrid device models*

The CONFIG hybrid simulator is well suited to meet these specialized needs. CONFIG is a modeling and simulation tool with reconfigurable models of complex systems. It is intended to support high-level analysis of the behavior of complex systems during operations, and is designed for exploration of cascading failure sequences. The simulation

and modeling approach allows for model selection in simulation scripts, including failure insertion and selective fidelity adjustments. The approach to simulating flow permits qualitative capture of remote interactions between components that otherwise could go undetected in complex systems. It also permits analysis and simulation of distributions of static potentials that are dependent on the history of operations such as a sequence of valve openings and closures.

CONFIG extends discrete event simulation with capabilities for approximate and qualitative modeling of continuous system behavior [12, 14]. The tool supports investigation of the sequencing of system events in fast scenario-based simulation of operations, and is well suited for analysis of hazards and effects of failure modes and fault tolerance.

To evaluate operation concepts prior to implementation of actual control software, or to evaluate manual procedures that unfold over very long periods of time, procedures may be modeled as components of the larger system model in such a way that the procedure models and hardware component models interact synchronously. CONFIG capabilities for scripting and for modeling activities (control, procedures, schedules) can be used for early dynamic analysis of operational problems. In discrete event simulation, generation of events and time-advances can be random, supporting probabilistic analysis. CONFIG has been used primarily for deterministic analysis of specific state configurations and inputs, but can support probabilistic analysis.

Discrete event simulation provides a framework for causal modeling of states and outcomes. The continuous time base of discrete event simulation [22] supports both event-stepped time advances and discrete-time-step approaches to continuous simulation. In CONFIG, a discrete-time-step approach can be used to periodically update continuous variables in a component [20]. Discrete event models of systems are composed of coupled component models. In CONFIG, components models can simulate multiple behavior modes. Each mode has state equations that generate behavioral effects, and conditions that govern mode transitions. System behavior emerges from the coupled behavior of the components. In CONFIG, the model structure can be “recomposed” during a simulation as the direction and activation of the couplings changes. Parts of the model are activated or deactivated as operating modes of components change or closed off areas of the system are brought into the working system configuration.

Simulations of biological water processing and air revitalization systems for spacecraft life support systems [15] requires representations of discrete events such as the opening of a valve and continuous changes. Continuous changes can include pressure changes in gas storage tanks as gas flows into or out of the tank. Gas pressures within a fluid container may be computed using either a linear (Eulerian) approximation or by projections of average rates

of change over a time step assuming exponential decay of the rate of pressure change. The exponential method is necessary when running simulations at high ratios of simulated to real time for systems in which time constants are large. The concentrations of the constituent chemicals in a fluid are updated by similar techniques.

During the course of a simulation, the magnitudes of pressure sources and resistances change and the system may be reconfigured periodically by events such as the opening and closing of valves. In response to such events, the simulator updates flow rates and potential drops across affected model components using graph analysis and linear circuit analysis of the model configuration. Prior to running a simulation, the model is partitioned into clusters of components so that that recomputation of flows and potentials down to the component level are necessary only for those clusters affected by a given simulation event.

In addition to the “dynamic” potentials associated with flows across resistances, the analysis determines static stresses (or static potentials) generated by flows at points where they are in contact with blockages such as closed

valves [13]. Unlike other flow-related properties, static potentials cannot be represented by a set of state equations. The distribution of static potentials is dependent on the specific history of operations performed on a system. The distribution of static potentials in a system may be difficult for a human operator to understand and anticipate because it is dependent on the history of operations performed on the system (e.g., the order in which valve and pumps are operated). In a hydraulic system, an undesirable distribution of static potentials could produce effects such as the unintended opening of a relief valve, with catastrophic consequences.

#### Generic Thermo-hydraulic Library

A generic component library is being developed to support system accident simulation in thermo-hydraulic domains. This library serves as a pathfinder for characteristics of models that can be coordinated with HIT design information. It is used to construct component-connection models whose behavior can include a broad variety of types of performance problems and hazards. Figure 5 shows the component class hierarchy in the current library.

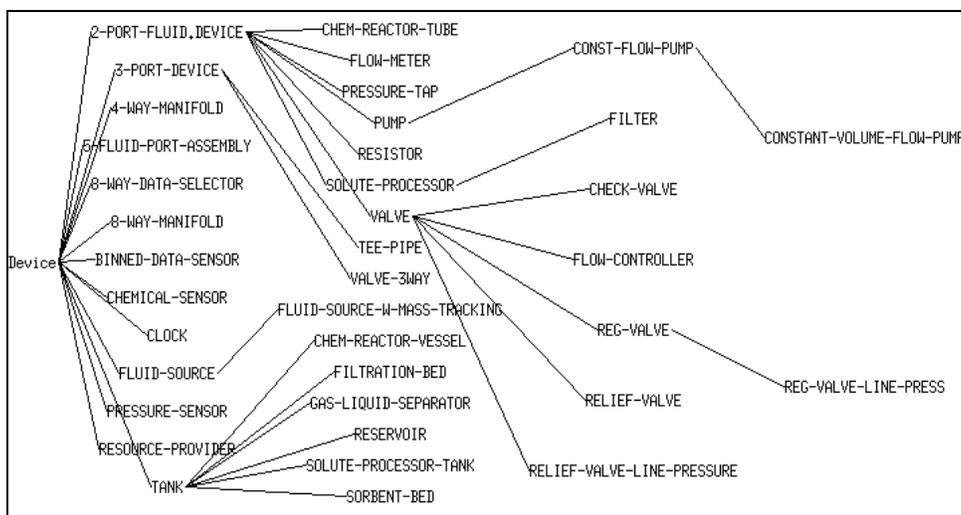


Figure 5 - Generic Components in CONFIG Library

#### Configurable Component Failures

The components in the generic library can exhibit a wide variety of hazardous states and failures. These include immediate or delayed discrete changes to state, behavior mode or control regime. These changes can be triggered by script input, failures and problems in connected components. These factors can also trigger continuous degradations. Nontemporal algebraic relations can define how performance levels are affected by conditions, and degradation and regeneration of performance rates can be modeled. Stuck flags can control failures to operate or change upon input. Measures or input can be randomly varied. Resource providers have alternative methods for reacting to excessive demands from multiple loads. Reactors and separators models with multi-component mixtures can

handle rapid fluid composition changes for introduction of contaminants, imbalances triggered by feed or flow reversal problems, and partial separation with migration of products to the wrong outflow. At the system simulation level, buildup and release of static stresses can be simulated and leaks can be specifiable additions to simulation scenarios.

CONFIG helps the modeler to locally and selectively complicate a model for a specific simulation scenario. The modeler can change component parameters rather than revising behavior descriptions. For example, changing the default valve resistance from “infinity” to a finite number easily simulates leakages across closed valves.

For simulation scenarios with leakage out of a specific

component, the user need only set a flag in that component. CONFIG then regenerates the computational network for flows with only the marginal increase in the complexity needed to simulate the leakage of interest.

### Mapping from HIT Specifications to Models

The Hazard Identification Tool is being developed to provide users with a means of constructing system models based on a functional view of the components. HIT, however, is not intended for running simulations. In order to reveal how a system performs its intended function over time, simulations are useful. To support such simulations, a capability is being developed to automatically translate HIT models into models usable by CONFIG. CONFIG is primarily oriented toward simulating the physical behavior of components. The structure of the hierarchy of components types in the CONFIG generic library is influenced more by the physical structure and behavior of the components than by their functions. Further, the names of component types will not always be identical in CONFIG and HIT, even when they are sufficiently similar for the

correspondence to be obvious to the user. Therefore, the correspondence between components selected in building a HIT model and the components available in a CONFIG library will not always be straightforward. To automate the translation process as much as possible, the translation utility contains tables explicitly mapping the correspondences between HIT and CONFIG component representations. Figure 6 shows part of the graphical interface for a running simulation of the biological water processor.

Prior to running a simulation, values of device initialization parameters may be required in addition to those the user supplies in HIT. In order to accomplish this without requiring the HIT user to be familiar with the details of CONFIG device representations, each CONFIG component has an interactive dialog interface. This interface can be activated to elicit the necessary initialization data and guide the user in tailoring the generic models for the specific design.

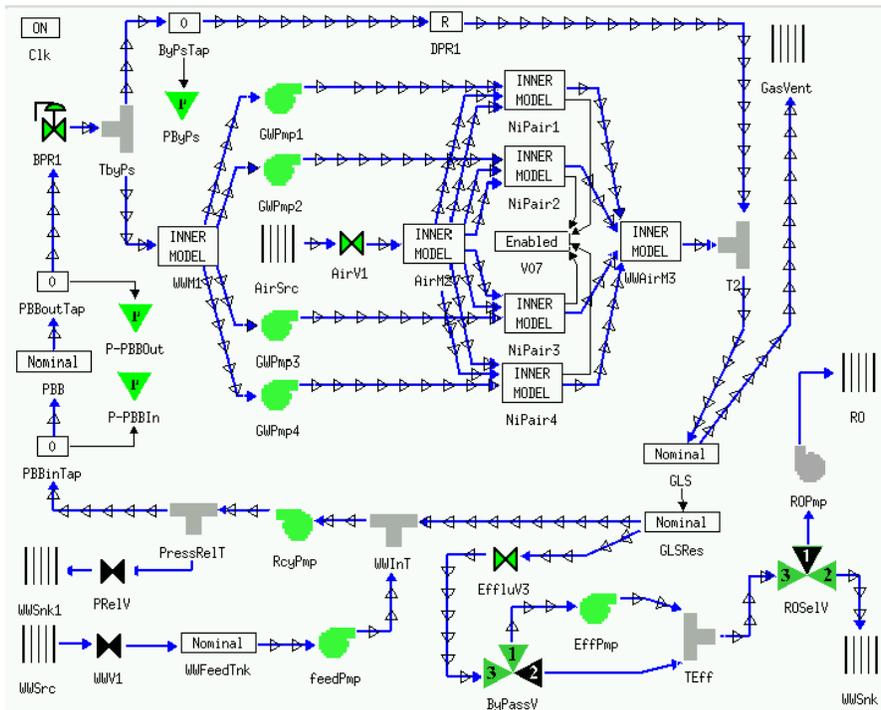


Figure 6 - Mapped CONFIG Model of BWP

### Scripts and Script Mapping

The HIT model will provide a static view of the relationships of the functions of components in a system. A HIT scripting capability (Figure 7) is being developed for specifying changes over time to the functional status of model components during simulations. The HIT user enters information such as initial conditions and the set of events for a simulation run. The tool queries the HIT knowledge base for model structure and component device information. In the HIT script, events may be specified to occur at a

given time or under a set of conditions. These conditions may represent either nominal operations or off-nominal induced failures. For example, a feed pump may supply water to a reservoir that must maintain a certain amount of water in order for the system to operate properly. To evaluate how effectively system safeguards respond to a pump failure, a script could specify that the pump will fail just before the reservoir becomes empty.

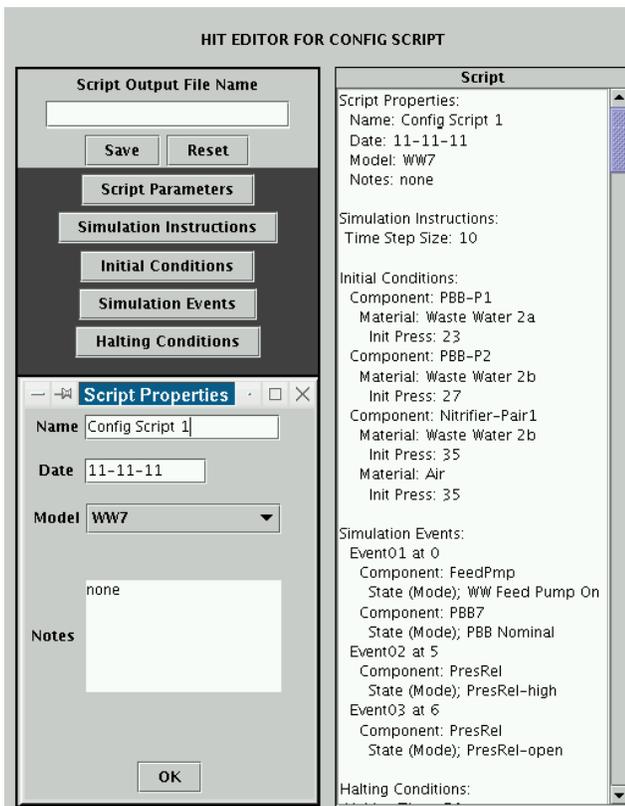


Figure 7 – Script Editor

A utility has been implemented for translating HIT scripts into a CONFIG simulation script of discrete events. It is complementary to the model translation capability. A directive to change the functional status of a component, such as a valve opening to permit flow, is translated into a directive in the CONFIG script to set the value of a specific variable that will cause the CONFIG representation of the same valve to transition from the CLOSED mode to the OPEN mode.

## 6. CONCLUSIONS

We have made substantial progress in designing and prototyping elements of a system for aiding design engineers in model-based hazard analysis. Our initial work indicates that it is probably feasible to anticipate and analyze system accidents that would otherwise elude designers. It is also possible to develop terminology and tools that are consistent with the concerns of the designer, but that can also provide information to support analysis by specialists concerned with safety and risk. We will continue to refine, develop and integrate all these elements of a prototype system for model-based hazard analysis.

## ACKNOWLEDGMENT

This work is funded by the System Reasoning and Risk Management thrust area in the NASA program, Engineering for Complex Systems.

## REFERENCES

- [1] L. Anthony, W. C. Regli, J. E. John and S. V. Lombeyda, "An Approach to Capturing Structure, Behavior and Function of CAD Artifacts," *ASME J. Computer and Information Science in Engineering*, **1**, 186, 2001.
- [2] S. L. Cornford, M. S. Feather, and K. A. Hicks, "DDP – A Tool for Life-cycle Risk Management," *2001 IEEE Aerospace Conference Proceedings*, Big Sky, March 2001.
- [3] Crew and Thermal Systems Division. *Advanced Water Recovery System (WRS) Integrated Test Plan*. Crew and Thermal Systems Division, Life Support and Habitability Systems Branch, NASA Johnson Space Center, Houston TX, February 2000.
- [4] D. C. Hendershot, R. L. Post, P. F. Valerio, J. W. Vinson, D. K. Lorenzo and D. A. Walker, "Putting the 'OP' Back in 'HAZOP'," *MAINTTECH South '98 Conference and Exhibition*, December 2-3, 1998.
- [5] J. M. Hirtz, R. Stone, S. Szykman, D. A. McAdams and Kristin L. Wood, "Evolving a Functional Basis for Engineering Design," *Proceedings of DETC01, ASME Design Engineering Technical Conference*, September 2001.
- [6] E. Hollnagel, *Accident Analysis and Barrier Functions*. Halden, Norway: Institute for Energy Technology, 1999.
- [7] International Space Station Program, NASA Johnson Space Center, *Assembly and Operations Support Plan Systems Data Report*, Volumes 1-16, 1997-2003. [http://iss-www.jsc.nasa.gov/ss/issapt/oddi/sys\\_book.html](http://iss-www.jsc.nasa.gov/ss/issapt/oddi/sys_book.html)
- [8] Y. Kitamura and R. Mizoguchi, "Meta-functions of Artifacts," *Papers of the 13th International Workshop on Qualitative Reasoning (QR-99)*, 136-145, 1999. <http://citeseer.nj.nec.com/387684.html>
- [9] P. Ladkin, "Software Direct Cause of December 2000 Osprey Crash," *The Risk Digest 21*, Issue 33, April 2001.
- [10] W. Langewiesche, "The Lessons of ValueJet 592," *The Atlantic Monthly*, 15: 81-98, March 1998.
- [11] N. Leveson, *Safeware: System Safety and Computers*. Reading, Mass.: Addison-Wesley, 1995.
- [12] J. T. Malin and L. Fleming, "Enhancing Discrete Event Simulation by Integrating Continuous Models," *Hybrid Systems and AI. Working Notes for AAAI 1999 Spring Symposium Series*, AAAI, Menlo Park, CA., March 22-24, 1999.
- [13] J. T. Malin, L. Fleming and D. R. Throop, "Predicting System Accidents with Model Analysis during Hybrid Simulation," *Proceedings of Business and Industry*

*Symposium, Advanced Simulation Technologies Conference, Simulation Councils, Inc., pp. 155-160. April 2002.*

[14] J. T. Malin, L. Fleming and D. R. Throop, "Hybrid Modeling for Scenario-Based Evaluation of Failure Effects in Advanced Hardware-Software Designs," *Model-Based Validation of Intelligence*, Technical Report SS-01-04, AAAI Press, Menlo Park, CA, 2001.

[15] J. T. Malin, L. Flores, L. Fleming and D. R. Throop, "Using CONFIG for Simulation of Operation of Water Recovery Subsystems for Advanced Control Software Evaluation," *Proceeding of 32nd International Conference on Environmental Systems*. SAE Paper No. 02ICES-114, July 2002.

[16] M. Modarres, "Functional Modeling of Physical Systems Using the Goal Tree Framework," *AAAI-98 Workshop on Functional Modeling and Teleological Reasoning*, July 1998.

[17] D. A. Norman, "Categorization of Action Slips," *Psychological Review* **88(1)**, 1-15, 1981.

[18] N. F. Noy, W. Grosso, and M. A. Musen, "Knowledge-Acquisition Interfaces for Domain Experts: An Empirical Evaluation of Protege-2000." *Twelfth International Conference on Software Engineering and Knowledge Engineering (SEKE2000)*, 2000.

[19] C. Perrow, *Normal Accidents: Living with High Risk Technologies*. Princeton, NJ: Princeton Univ. Press, 1984 & 1999.

[20] D. R. Throop, J. T. Malin and L. Fleming. 2001. "Automated Incremental Design FMEA," *2001 IEEE Aerospace Conference Proceedings*, March 2001.

[21] V. Venkatasubramanian, Jinsong Zhao and Shankar Viswanathan. "Intelligent Systems for HAZOP Analysis of Complex Process Plants," *Computers and Chemical Engineering*, **24**, 2291-2302, 2000.

[22] B. P. Zeigler, *Theory of Modeling and Simulation*. New York: Wiley, 1976.

## BIOGRAPHIES

**Jane T. Malin** is Technical Assistant in the Intelligent Systems Branch in the Automation, Robotics and Simulation Division in the Engineering Directorate at NASA Johnson Space Center, where she has led intelligent systems research and development since 1984. She has led development of the CONFIG simulation tool for



evaluating intelligent software for operation of space systems. She has led research on intelligent user interface and intelligent agents for control of space systems, and on teamwork tools for anomaly response teams. Her 1973 Ph.D. in Experimental Psychology is from the University of Michigan.

**David R. Throop** has been an Artificial Intelligence Specialist with The Boeing Company since 1992. He provides engineering software support in the Intelligent Systems Branch in the Automation, Robotics and Simulation Division in the Engineering Directorate at NASA Johnson Space Center. He oversaw development of FMEA modeling software and its use for the International Space Station. His 1979 Bachelors of Chemical Engineering is from Georgia Tech. His 1992 Ph.D. in Computer Science is from the University of Texas, with a dissertation on Model Based Diagnosis.



**Land D. Fleming** is a Computer Systems Specialist supporting the NASA Johnson Space Center Automation, Robotics, and Simulation Division since 1990. He has been involved in both the development of computer simulation tools and their application to space systems. His 1987 M. S. in Computer Science is from De Paul University.



**Luis Flores** is a systems software engineer supporting the NASA Johnson Space Center Automation, Robotics, and Simulation Division since 1985. He has been involved in design and development of software using knowledge-based, intelligent control and computer simulation tools for space systems applications. His 1967 Ph.D. in Physics is from Texas A&M University

