Configurable Components for Health Management of Space Systems

Debra Schreckenghost, R. Peter Bonasso, and David Kortenkamp

TRACLabs, Inc
1012 Hercules, Houston, TX 77058
ghost@ieee.org, r.p.bonasso@nasa.gov, korten@traclabs.com

Abstract

Advanced computing techniques for system health management typically address problem detection and the determination of fault causality. We have designed a health management architecture that combines these techniques for problem detection and isolation with intelligent monitoring and control techniques for problem response to make humans more effective at problem management. This integration of technologies leverages a large body of advanced computing research while providing a solution that is fundamentally human-centric. Our approach is general in that the components are designed for use in a variety of applications, from single systems to complex spacecraft consisting of multiple, coupled systems. In this paper we define eight core health management components and use these components to design a health management architecture for NASA mission operations support.

Introduction

A variety of advanced computing techniques have been developed for health management of systems in space operations and the military [7, 11, 15]. These technologies address the detection of problems (including prognostics) and the determination of fault causality. With the exception of actions taken to safe systems after anomalies, however, problem response in these domains is essentially a human task. We have designed a health management architecture that combines health management techniques for problem detection and isolation with intelligent monitoring and control techniques for problem response to make humans more effective at problem management. The architecture supports human involvement throughout problem detection and response, and addresses areas outside of traditional health management, such (1) identifying the impacts of problems to system capabilities, resources, and mission, (2) determining and evaluating response options, and (3) coordinating the chosen response with other system and crew activities.

Our design is based on configurable health management components that are used to construct health management software targeted to the needs of different applications. These components include technologies derived from Integrated System Health Management (diagnosis and state determination), technologies derived from intelligent monitoring and control (real-time sensing & acting and execution), and technologies to support human jobs (mission management, capabilities determination, resource management, and planning). The functionality of these components will typically be provided by humans and software working together.

In this paper we define eight core health management components. We use these components to design a health management architecture for NASA mission operations. This architecture supports detecting and isolating problems, assessing mission impacts, planning problem response, and managing execution of the planned response.

Configurable Health Management Components

We have identified the following eight core health management components:

- State Determination
- Diagnosis
- Planning
- Execution
- Real-time Sensing and Acting
- Resource Management
- Capabilities Determination
- Mission Management

We documented our design using Universal Modeling Language (UML) diagrams for each component and interfaces in the Interface Definition Language (IDL). Interfaces are shown in the component diagrams. Page limitations prevent including the detailed design in this paper. We summarize our design for configurable health management components in this section.

State Determination

The State Determination component identifies states, state properties, and collections of states (i.e., configurations), monitors for the occurrence of pre-specified values of states (i.e., conditions), and manages the saving and loading of sets of states (i.e., a checkpoint). States are the truth-values of properties in the world. States can be one or more of these properties, limited only by the mathematical and logical relations among them. State Determination performs the functions below. See Figure 1 for a diagram relating these functions to the interfaces that can be called on the State Determination component.

State Identification: State Determination transforms signals (numeric values) as well as input states into new states. State Determination also can generate new values by propagating (e.g., inference) or projecting (e.g., extrapolation) old values. Statistical techniques may be used to extract properties of system elements. As well State Determination may be required to degrade the validity or accuracy of state values over time based on its source and the last time of update. Related to all of these functions is the ability to reconcile conflicting information, making use of time, sources and weighting factors

Condition Monitoring: State Determination can be requested to watch for specific state values, or conditions, and report when the conditions occur. This monitoring function ranges from simple limit sensing to monitoring for complex events, called episodes, which occur over time intervals and may involve pattern matching and curve fitting. A special case of condition monitoring is plan monitoring, wherein states of the world of concern to a plan are compared to the desired states in order for the plan to be updated and modified if necessary.

Checkpointing: State Determination can save a set of state values at some point in time (checkpoint) and reload the saved states at some later time. Checkpoints can be used to initialize the state determination process. Checkpoints are used to return to previous configurations which is useful for training, simulation, and system checkout. These archived sets of states also provide a record of states for diagnosis and troubleshooting.

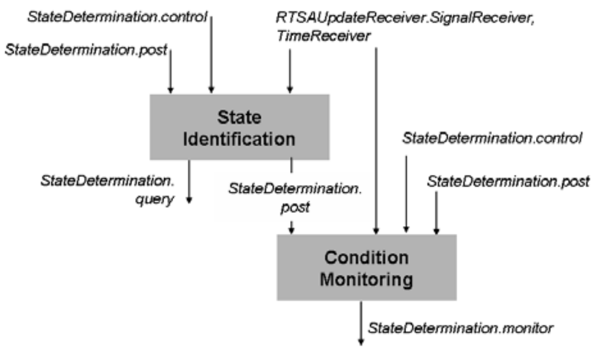

Figure 1. State Determination Component

Diagnosis

The Diagnosis component identifies system faults and isolates the cause(s) of the fault, when possible. Faults are anomalous changes in the state of a controlled device that may warrant some type of corrective action. Diagnosis can perform two main functions: identifying possible causes of faults (fault candidates) and proposing tests to reduce the set of possible faults (candidate refinement). Prognosis is a special case of Diagnosis, where faults are predicted before they occur instead of detected after they occur. Diagnosis performs the functions below. See Figure 2 for a diagram relating these functions to the interfaces that can be called on the Diagnosis component.

Fault Candidate Identification: Diagnosis provides models that map observed signals and states to faults. It uses State Determination to monitor for changes in command and sensor signals from hardware. It predicts the expected state changes from the observed commands and compares these predicted states to observations. When there is a discrepancy between observation and prediction, Diagnosis searches for faulty device states that will explain the discrepancy. Unless instrumentation makes the device fully observable, more than one explanation is possible. When multiple candidate causes are identified, Diagnosis can estimate the likelihood of each candidate cause. These estimates are typically based on a priori statistics about device failure. Diagnosis can occur at multiple levels – system, subsystem, and element (the parts and collections of parts that comprise systems and subsystems). Diagnosis components can be connected such that the candidate faults identified by one component serve as state input to another component. This permits progressive refinement of the fault candidate list as information moves through the different levels. Since the detection of mismatch is based on correct knowledge about the expected behavior of devices (called the reference model by Robinson [22]), it is necessary to update the reference model when devices degrade or are configured in new ways.

Fault Candidate Refinement: Active diagnosis can include running tests on faulty devices to glean additional information useful in identifying faults. A second function of the Diagnosis component is to propose tests of the affected device that will help disambiguate the fault candidate list by either reducing the set of possible faults or adjusting the likelihood of candidates on the list. Because these tests can require reconfiguring the affected devices, diagnostic tests must be coordinated with other ongoing system tasks. This is accomplished by scheduling diagnostic tests in the activity plan when possible and performing tests using the Execution component. Once a test has been performed, Diagnosis components monitor signal changes resulting from the test and compare them to models of faulty behavior to refine the candidate list.

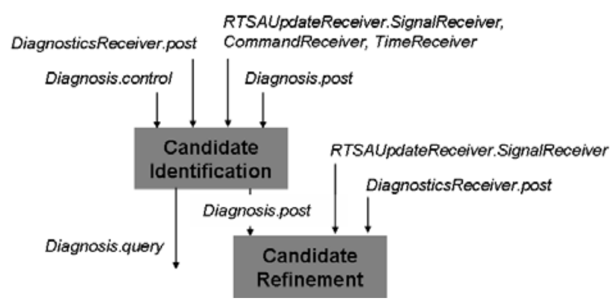

Figure 2. Diagnosis Component

Planning

Planning generates and evaluates one or more sets of tasks (plans) that will move the controlled system from an initial state to a desired state, selects a plan for execution, and possibly modifies the selected plan as it executes. Thus, the Planning component performs the functions plan generation, plan evaluation, plan selection, and plan modification. We consider plan monitoring to be a special case of State Determination (see previous discussion).

Plan execution is described in the Execution component. Planning performs the functions below. See Figure 3 for a diagram relating these functions to the interfaces that can be called on the Planning component.

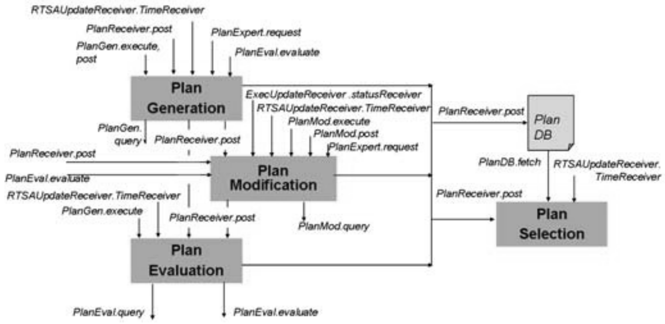

Figure 3. Planning Component

Plan Generation: The Plan Generation function searches through the space of tasks or states to find a time-ordered set of tasks that will bring about a desired world state from an initial world state. Plan Generation must handle planning for multiple, possibly conflicting goals. If plans cannot be found for all the goals, Plan Generation should be able to report such cases to the plan requester along with the goals that failed and the reasons for failure. Plan Generation begins with loading the set of world-states, application domain ontologies -- including information about agents available to perform tasks -- and constraints that must hold when generating a plan for the goals in question. The Plan Generation function then builds a plan using techniques such as the following:

- *theorem proving techniques* that prove the existence of a sequence of states from the starting set of states to the desired final set of states, or
- *operator-based techniques* that use goals and state conditions associated with plan operators to select and temporally link sets of operators in sequence of states from the starting set of states to the desired final set of states. A plan operator is a set of tasks that will achieve the associated goal under the prescribed conditions, which can include expert computation from outside Plan Generation.

For both techniques, the derived sequences of states must not violate constraints placed on the goals. Constraints are the allowable combinations of state values (e.g., resource levels) that must hold at specified times in the final plan. Sometimes, Plan Generation will be given a partial plan that meets some goals and constraints, and must use it to generate a final plan that meets all the goals while satisfying the pertinent constraints. When planning in domains with more than one agent (person, robot, or process that can execute tasks), Plan Generation identifies which agent will carry out each task in the plan. When supporting distributed planning, Plan Generation must be able to request an agent's ability to perform a task or set of tasks (see Plan Evaluation) and to use the resulting information to assign tasks to agents during planning.

Plan Evaluation: The Plan evaluation function can provide three methods of measuring a plan for comparison to other plans. The first evaluation method computes a set of measures for the overall plan, such as the amount of time required to execute, the minimum resource usage, or how well the plan stays within overall constraints, e.g., the fuel budget. The second evaluation method measures how well the plan stays within the constraints on the individual tasks of the plan. This second method is important when evaluating the capability of an agent to accomplish specified tasks in a plan, a query that is made during Plan Generation (and possibly Plan Modification). The various constraints can have weights associated with them to specify the importance of each constraint with respect to the others. The third method computes the probability of success, a measure most often looked at by humans during Mission Management to eliminate certain plans from consideration.

Plan Selection: The Plan Selection function determines the preferred plan in a set of candidate plans (e.g., as stored in a plan database) that match the world state and achieve the required goals. These plans can be the result of Plan Generation or can be hand generated as standard operating procedures. In some cases, a full plan is selected for execution. In other cases, only the next task to perform is selected in an already executing plan. In still other cases, as in the middle tier of a three-tiered architecture [12], the variables in the task are instantiated and the task is decomposed into action groups at the next level of abstraction (method selection or spreading activation among plan nodes). In all cases, plan selection provides one or more tasks to the Execution component for execution. When selecting the mission plan, we expect humans to manage Plan Selection. They will use their superior knowledge of the overall mission with the measures provided by Plan Evaluation to select the plan best suited to the mission objectives and current circumstances. The Mission Management component will provide functions and interfaces to support humans in designating the mission plan.

Plan Modification: The Plan Modification function resembles Plan Generation in that it can generate a new plan in response to changing world states, as determined by Plan Monitoring. Thus it must be able to generate multi-agent plans for multiple goals using theorem proving or operator based techniques, and to refine plan fragments. An important difference between Plan Modification and Plan Generation is that the request for planning is from the Plan Modification function itself. Plan Modification begins when the completion status of the executing parts of the plan are updated from world states obtained from Plan Monitoring. If the new statuses are as expected from Plan Generation, Plan Modification need only update what tasks are completed and then update the preconditions those tasks affect for other tasks in the plan. If the statuses are not as expected, Plan Modification tries to repair the plan by identifying what part of the plan did not have the expected effect (called the locus of repair) and trying to find an alternative way to achieve the expected effect. This repair should minimize the disruption of the executing plan when possible. Additionally, plan modification must be able to relax constraints not based on physical limits (i.e., soft constraints) in order to find a feasible repair.

Execution

The Execution component carries out the tasks specified by a plan. A plan specifies one or more tasks needed to achieve effects in the world that will accomplish a set of goals. The Execution component transforms tasks into commands to and responses from the Real-time Sensing and Acting (RTSA) component. Since a plan can consist of complex tasks comprising multiple commands, it may be necessary to use multiple layers of Execution and Planning to decompose the plan recursively into simpler tasks until it consists of commands the hardware can interpret. The Execution component has two primary functions: (1) dispatching tasks as commands to the RTSA component, or to other levels of control for further task decomposition, and (2) dispatching tasks or plans to other Execution components. While carrying out tasks, the Execution component must transition the tasks through the various stages of execution, e.g., ready, active, waiting, finished as well as be able to report these stages to interested requesters. The Execution component also must be able to manage the concurrent execution of tasks and determine the order of execution of competing tasks via a system of priorities. Finally, the Execution component must be able to load archived states or query State Determination to obtain the context for execution. See Figure 4 for a diagram relating these functions to the interfaces that can be called on the Execution component.

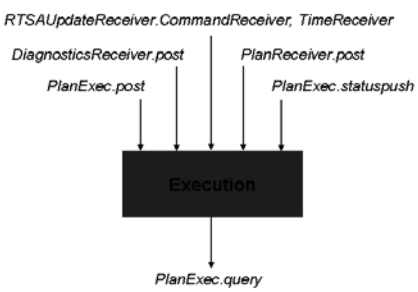

Figure 4. Execution Component

Real-time Sensing and Acting

The Real-Time Sensing and Acting (RTSA) component connects the Health Management component software to the controlled system. It directs (commands) the hardware actuators to carry out operations. It receives numeric values (signals) from both hardware sensors and actuators. It also distributes time signals to other components. For spacecraft, RTSA functions are often delivered by command and control in the avionics system of the craft. In such applications, our architecture specifies the functional and information requirements for compatibility with it but does not mandate how these functions are implemented. RTSA performs the functions below. See Figure 5 for a diagram relating these functions to the interfaces that can be called on the RTSA component.

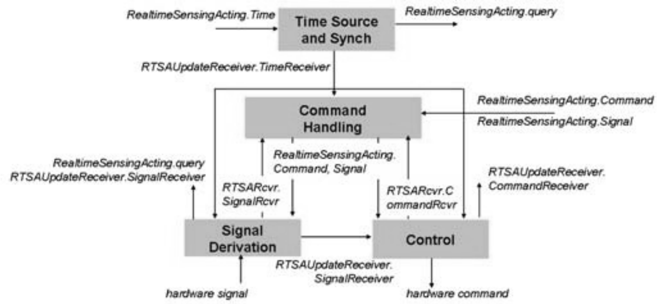

Figure 5. Real-time Sensing and Acting Component

Command and Signal Handling: RTSA provides the control algorithms and the sensor processing algorithms that manage the hardware. It is responsible for command handling and arbitrating conflicting commands. The command handling functionality also verifies proper command format, routes commands to other components, and returns responses to commands. RTSA also provides sensor processing algorithms for manipulating sensed data from the hardware. These manipulations include derivation of values and identification of signal features. RTSA also maintains the coherency of signals distributed to other components in the architecture. To support our objective of providing configurable components useful in a variety of applications, the RTSA will provide for encapsulation of hardware-specific interfaces. This will permit other functions in the RTSA component to manipulate the hardware using standard interfaces and will reduce the effect of hardware upgrades on these interfaces.

Time Source and Synchronization: RTSA is the source of time for synchronizing functionality with the rest of the architecture. It provides a means of calibrating the time source and distributing timing signals e.g., atomic time clock or hardware system time.

Resource Management

Resources are the supply of assets available to the systems connected to the Health Management architecture. All devices in these systems can be considered resources. Additionally, the people, computers, or robots performing tasks using these systems can be resources (e.g., resources needed to perform a task). The designation of the assets that will be managed as resources is specific to an application. For example, manned spacecraft considers power, data, and thermal to be the core resources. Resources have properties that affect their availability and how they can be allocated. Resources may be consumable or reusable. Consumable resources are depleted by use while reusable resources can be used multiple times without depletion. Resources also may be dedicated or sharable. Dedicated resources must be allocated to one task at a time, while sharable resources can be used by multiple tasks simultaneously. For example, power is a consumable, sharable resource while tools are a reusable, dedicated resource. Finally, some applications provide the ability to generate resources during the course of the mission (e.g., power). This must be considered when allocating these resources. The Resource Management component administers the allocation, use, and optimization of resources during a mission. The resource allocation function determines what resources are allocated to the activities of the mission plan. The resource tracking function matches resource utilization rates with allocated resources to detect differences. Various other components are expected to communicate with Resource Management. Mission Management will coordinate its goals with Resource Management to ensure that enough resources are available to accomplish the goals. Planning will iterate with Resource Management as resources are assigned to tasks in the plan to avoid conflicts. Capabilities Determination will inform Resource Management when faults have reduced the available resources. State Determination and RTSA will provide the raw data that Resource management uses to track resource levels and compare them against the allocation. Resource Management performs the functions below. See Figure 6 for a diagram relating these functions to the interfaces that can be called on the Resource Management component.

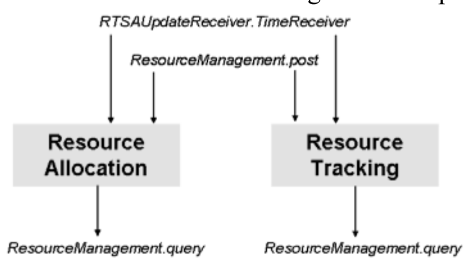

Figure 6. Resource Management Component

Resource Allocation: Resource Management assigns resources to tasks by comparing predicted resource needs to available resource quantities. Resource budgets and profiles define the maximum permissible quantities of a resource that can be used by a system. Budgets and profiles can vary at different points in time. Resources also can be allocated to tasks within the mission plan. Resource allocations for tasks specify the resources required to perform a task. Resource allocations for tasks must comply with the system allocations defined in resource budgets and profiles. If needed for an application, Resource Allocation can optimize the allocation of resources to tasks. Resource optimization computes how well different allocations meet predetermined optimization criteria and selects the allocation that best matches these criteria. In resource constrained environments like space, Resource Allocation must ensure the highest priority tasks receive adequate resources. For example, task priorities in space are driven by flight rules categorizing task criticality according to crew safety, vehicle integrity, and mission completion. Since failures and configuration changes can affect resource availability, Resource Allocation must be able to adjust allocations when availability changes.

Resource Tracking: Resource Management derives the current level of resources and resource utilization rates. These computations are based on information from State Determination and RTSA. It compares these resource utilization rates to allocated resources to detect over-utilization of resources. When Resource Tracking detects a resource shortfall, it notifies State Determination and Capabilities Determination about the shortfall. Resource Tracking also can predict resource levels to support planning. This prediction is based on knowledge of utilization rates and planned activities. Finally, Resource Tracking reports resource levels and utilization rates to other Health Management components.

Capabilities Determination

Capabilities Determination identifies the functions available in a given system configuration. Functions are the device behaviors used to achieve mission tasks. The functions of a system depend upon the states of the system and the supply of available assets (i.e., resources). When the system state becomes anomalous (i.e., fault), the functionality provided by the system may be affected. Similarly, when a system experiences a shortfall in required resources, system functionality may be affected. Capabilities Determination maps the state of system elements and the resources allocated to the system to the functions available in that state. Capabilities Determination can compute the current functions of a system element upon request. It also can predict system functionality, given a hypothetical state. This is necessary to determine if the functions required to perform tasks in the mission plan will be available when needed. When faults occur, Capabilities Determination re-computes the functionality of the system affected by the fault and notifies components with updated functional information. Similarly, when a resource shortfall is detected, system functionality is re-computed in light of the change in resource. Capabilities Determination then compares the latest functionality to the previous functionality to determine what functionality was lost due to the fault or resource shortfall. Function information from Capabilities Determination is used by Mission Management to assess the impacts of faults on the mission and by Resource Management to adjust resource availability after a fault has occurred. See Figure 7 for a diagram relating these functions to the interfaces that can be called on the Capabilities Determination component.

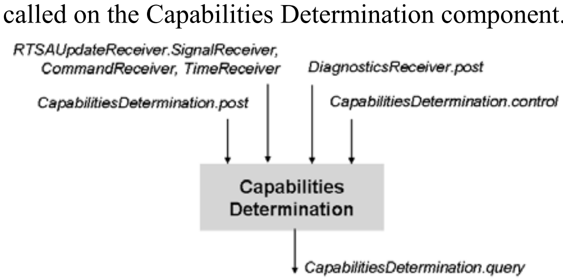

Figure 7. Capabilities Determination Component

Mission Management

Mission Management sets operational objectives, plans to achieve those objectives, and determines the impacts of problems to those plans. Mission Management is primarily a human task. Thus, the Mission Management component, in conjunction with the Planning component, provides functions useful to humans when planning the mission and assessing the impacts of problems to the mission. Mission Management provides five functions: (1) Goal Determination - determining and adjusting the mission objectives, (2) Constraint Modification - setting and modifying the constraints on the mission, such as flight rules, (3) Designation of Mission Plan - selecting a plan to achieve mission objectives, (4) Plan Impacts – assessing the impacts of faults to the mission plan, and (5) Criticality Impacts – determining the changes in the criticality of functions after failure. The function of building, modifying, and evaluating plans using goals and constraints from Mission Management is performed by the Planning component. The Mission Management component interacts with the Capabilities Determination component and the Planning component when assessing mission impacts. We describe the Mission Management functions below. See Figure 8 for a diagram relating these functions to the interfaces that can be called on the Mission Management component

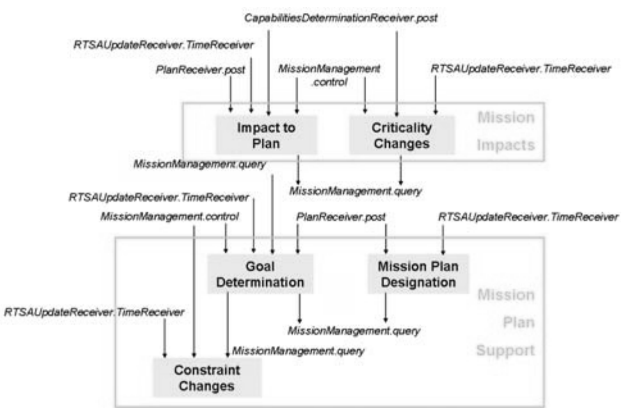

Figure 8. Mission Management Component

We describe the Mission Plan Support sub-functions below:

Goal Determination: The initial step in Mission Plan Support is to identify the goals of the mission. A goal is the purpose which a task is intended to accomplish. The Mission Management component supports humans in identifying and adjusting mission goals. A mission goal has a priority indicating the importance of accomplishing the goal. These priorities can be used by the Planning component to make choices about which goals to plan first. Priorities also are used when assessing the severity of failure impacts to the mission. Mission goals often are selected well before the mission is conducted. It is necessary, however, to support adjusting these goals during the mission in response to problems or opportunities. Faults can affect the ability to achieve mission goals by impacting available system functionality. Additionally, faults can introduce new goals required to recover or workaround the problems resulting from them.

Constraint Modification: Prior to building mission plans it is also necessary to identify the constraints on how the mission goals are achieved. A constraint defines the allowable combinations of values for a subset of signals, states, and/or their properties. The Mission Management component aids humans in defining and modifying mission constraints. It defines data structures for mission constraints, provides capability to check whether sets of constraints are consistent, and supports humans in adding, deleting, or revising constraints. The Planning component uses constraints to guide its selection of tasks to accomplish mission goals. Mission constraints can be general, applying to all goals, or specific to a single goal. For example, all goals might be constrained to stay within power budgets while a single goal to check status with Ground Operations might be constrained to occur at a specific time. For manned space operations, flight rules capture many of the constraints that the mission plan must adhere to. The Mission Management component provides functions for inspecting and waiving flight rules.

Designation of Mission Plan: Because it is possible that more than one plan can be built to accomplish mission goals, it is necessary to evaluate different plans and select one to execute. The Mission Management component works with the Planning component to evaluate and select a mission plan. Plan evaluation permits assessing the utility of alternative plans. Humans work with the Mission Management component to review these utilities and designate one plan for execution. Mission goals, constraints, and plans are stored in a Mission Database. The Mission Management component interfaces to this database to retrieve and update these data structures.

Plan Impacts: Plan Impacts are determined by taking the losses in system functionality due to faults provided by Capabilities Determination and querying the Planning component to identify which tasks in the mission plan use the lost capability. Knowledge of available redundant functionality also is needed to determine impacts. When redundancy is available, impacts are limited to the cost of reconfiguring to and operating with the redundant capability. When redundancy is not available, impacts range from performing additional tasks to recover lost capability to being unable to accomplish a mission goal.

Criticality Impacts: Criticality Impacts are determined by taking the losses in system capability due to faults provided by Capabilities Determination and applying mission constraints (e.g., flight rules) to determine changes in criticality of the affected functions. For example, flight rules define three levels of criticality: 1 – crew safety, 2 – vehicle integrity, and 3 – mission. Loss of system capability can change the criticality level of a function. Loss of system capability also can affect redundancy. Finally, to prepare for quick response in the event of a further problem, the loss of system capability may be mapped to the next worse failure that could occur.

The severity of impacts depends upon a number of factors [6]. Loss of function is typically worse than loss of redundancy. If the lost functionality was in use, it may be necessary to reconfigure for proper operation while, if the lost functionality was a backup, immediate action is usually not required. The loss is more severe if the lost functionality is planned for use in current or upcoming activities. Some impacts may not manifest immediately, so the time it takes for a loss of functionality to have an adverse effect (time to effect) and the time that adverse effects will persist (time of effect) should also be determined. Finally, the severity of the impact depends upon the ability to workaround or mitigate the effects of the problem. Some effects cannot be mitigated and some mitigation actions may compromise other functions. Because the Diagnosis component relies on correct knowledge of operations, it also is necessary to map loss of system capability to changes in the reference model used by the Diagnosis components.

Health Management Architecture Design

Our Health Management Architecture design consists of four modules corresponding to the phases of health management: (1) Problem Detection and Isolation, (2) Mission Impact Assessment, (3) Mission Planning for Problem Response, and (4) Execution of Problem Response, shown in Figure 9.

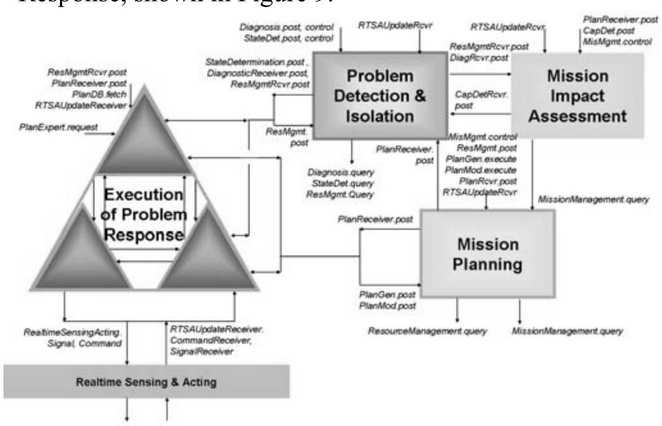

Figure 9. Health Management Architecture

Each module is constructed from the core components defined in the previous section. The Problem Detection and Isolation module detects problems, diagnoses the cause when possible, and passes fault states, resource shortfalls, and fault causes to Mission Impact Assessment and Execution of Problem Response. It requests Execution of Problem Response for diagnostic tests, if they can be identified. Mission Impact Assessment maps states from Problem Detection and Isolation and RTSA to loss of system capability. The human interacts with Mission Impact Assessment to determine how these losses impact the mission plan and change the criticality of remaining functions. These plan impacts and criticality changes are passed to Mission Planning. The human interacts with Mission Planning to identify if mission goals and/or constraints need to change. If changes have occurred, the mission plan is updated to reflect the changes. The modified plan is passed to Execution of Problem Response, which executes the response. The modified plan also is passed to Problem Detection and Isolation for use in identifying resource shortfalls in the future. We describe how the core components are used to create the architecture in the remainder of this section.

Problem Detection and Isolation

The purpose of Problem Detection and Isolation is to map the symptoms of a problem to sufficient information about the cause of the problem to permit a response. NASA has pioneered many innovative approaches to fault detection and isolation. The Beacon-based Exception Analysis for Multimission (BEAM) [13] software developed for unmanned spacecraft characterizes nominal operations statistically, and alerts problems as signal departures from nominal. BEAM determines system modes, integrates numeric and symbolic results, performs causal reasoning and interprets results by using Spacecraft Health Inference Engine (SHINE) [24]. Livingstone [28] and Livingstone 2 (L2) are a family of model-based fault diagnosis and recovery tools developed at NASA Ames Research Center. L2 uses a set of high-level, qualitative models to characterize a system's behavior in propositional logic form. The most recent addition to this suite of model-based tools is HyDE [19]. HyDE uses candidate generation and consistency checking for diagnosis of discrete faults in stochastic hybrid systems. MEXEC [1] is a model-based diagnosis engine developed at the Jet Propulsion Lab that combines ideas from Livingstone with the knowledge compilation techniques for diagnosis and planning. It compiles propositional logic models into Decomposable Negation Normal Form (DNNF) equations that can be evaluated in linear time.

Our approach to Problem Detection and Isolation combines functions from the State Determination, Diagnosis, and Resource Management components. The State Determination component identifies states and configurations of the systems managed by the architecture, based on signals from the RTSA component and its own state information. It also monitors for conditions manifested in states and signal that indicate a problem has occurred. The Resource Management component tracks the levels of resources and identifies when resources fall short of the expected levels. The Diagnosis component map states from State Determination and signals from the RTSA component to system faults by predicting the expected state changes from the observed commands, comparing these predicted states to observations, and identifying possible faulty states that explain mismatches between predictions and observations. A human works with the Diagnosis component to identify diagnostic tests that could be performed to eliminate some of the possible faults. These tests result in new signals and state changes being input to the Diagnosis component. The Diagnosis component uses these inputs to revise its list of possible faults. Since the diagnostic tests needed to isolate faults can require systems to reconfigure, they must be coordinated with other ongoing tasks. The requested tests are passed to humans and the Mission Management component to determine if the test can be conducted immediately or must wait for other tasks to complete.

In manned space operations today, most diagnosis is performed by executing malfunction procedures. These procedures describe the tests that will help in isolating the fault to the level needed to resolve the problem. Malfunction procedures also can include information to help humans determine the capabilities remaining after a fault, a necessary step in mission impact assessment (see later section). The Diagnosis component is able to perform some of the tests typically accomplished by malfunction procedures. Defining the interface between the Diagnosis component and the malfunction procedures executed by humans and the Execution component is one of the innovations of our design.

In some applications of the architecture, it may not be necessary or desirable to isolate the cause of the fault during the mission. In such applications, it can be sufficient to diagnose until a response to the problem can be identified. For example, short duration missions, such as Shuttle missions or early CEV lunar missions, are designed for most repair tasks to occur after the mission. In these cases, it is important to collect and archive sufficient data about the fault conditions to aid repair after the mission. It is not important, however, to isolate the cause of the fault for on-orbit repair.

Mission Impact Assessment

The purpose of Mission Impact Assessment is to map the results of Problem Detection and Isolation to their impacts on the mission, specifically how loss of functionality impacts the mission plan and changes the criticality of functions. There has been considerable research in modeling functions for system design [14, 26] or deriving functional models from existing designs [25], but little on using functional models during operations. Our approach uses such models for Mission Impact Assessment.

Our approach to Mission Impact Assessment combines functions from the Capabilities Determination and Mission Management components. When a fault or resource shortfall occurs, Capabilities Determination maps signals from RTSA, states from State Determination, and resources from Resource Management to the functions available from the associated systems. It compares this functionality to the previous functionality to determine what functionality was lost due to the fault or shortfall.

Impacts to the mission plan are determined by a human and the Mission Management component. The lost functions and resources identified by Capabilities Determination are passed to the Planning component. It identifies which tasks in the mission plan use that lost capability. Once the affected tasks have been identified, the availability of redundant functionality is assessed and the resource margins are identified. If redundant capability is available, plan impacts are limited to the cost of reconfiguring to and operating with the redundant capability. Similarly, if resource margins permit, plan impacts are limited to the cost of accessing those margins. If redundant functionality is not available or resource margins are constrained, the mission plan must be modified. Either new tasks to recover lost capability or resource must be added to the plan, or the mission goals using the lost functions or resources must be removed from the plan. Mission Management passes these impacts to the Mission Planning module described in the next section.

Criticality changes are determined by a human and the Mission Management component. The lost functions and resources identified by Capabilities Determination are evaluated using mission constraints that categorize function criticality (e.g., flight rules). Loss of system capability can eliminate or reduce redundancy, as well as make a function more critical. The severity of these impacts is assessed by comparing what has been lost or impacted to what is needed for current and future operations. In some cases, the impacts may not manifest for a period of time, or may not persist. Finally, to prepare for quick response in the event of a further problem, the loss of system capability may be mapped to the next worse failure that could occur.

The human can use Mission Management functions to predict the effects of a hypothetical problem. These functions are used to predict the next worse failure that could occur, given a state of the system. Such prediction also can be used to determine if the functions required to perform tasks in the mission plan will be available in the event of a hypothesized fault.

Mission Planning for Problem Response

The purpose of Mission Planning after a problem is discovered is to map the impacts from Mission Impact Assessment to a plan for responding to the problem. There has been extensive research into algorithms for building and modifying plans [5, 9, 17, 27]. Recent research has investigated how to extend these techniques for human interaction during planning [3, 10, 18]. We leverage this research in mixed initiative planning when implementing the Mission Management functions for interacting with Planning.

Our approach to Mission Planning for problem response combines functions from the Mission Management, Resource Management, and Planning components. Mission Impact Assessment passes plan impacts of lost capability to the human and Mission Management. The human interacts with Mission Management to identify if mission goals need to change – either add new goals for workaround or recovery of lost capability or abandon goals that are no longer achievable. The human and Mission Management may need to adjust the goal priority. The human also interacts with Mission Management to see if constraints used when building the mission plan need to change. These constraints can apply to all goals or can be specific to a single goal. Finally the human and Mission Management interact with Resource Allocation to adjust resource constraints affected by the lost system capability. The resulting mission goals and constraints are then passed to Plan Modification in the Planning component.

Plan Modification responds to changes in mission goals or constraints by trying to repair the plan. It identifies what part of the plan is affected by these changes. It removes tasks related to abandoned goals, adds tasks for new goals, and adjusts tasks affected by the changed constraints. If Plan Modification is unable to repair the plan in response to these changes, it may be necessary to rebuild the mission plan using the Plan Generation functions. When more than one plan satisfies the changes, the human interacts with Planning through Mission Management to evaluate the different plans and select one to execute. Once a revised mission plan is selected, it is passed to the Problem Response module for execution and to the Problem Detection and Isolation module for use in identifying resource shortfalls in the future. The Mission Management component stores changes in mission goals, constraints, and plans in the Mission Database.

Execution of Problem Response

The purpose of Execution of Problem Response is to take action to recover or workaround the problem. A number of intelligent monitoring and control architectures have been designed to handle problems by executing a problem response plan [2, 16, 20]. Recent research in adjustable autonomy [4, 8, 23] focuses on how humans can interact with these control architectures to adjust the degree of control automation. Our design is innovative in that it integrates technologies for adjustable autonomy with ISHM technologies for problem detection. Additionally, our architecture is unique in combining hierarchical control authority with peer to peer negotiation about task allocation during task execution. We recognize that some applications may choose to use existing system software (such as flight software) combined with human procedures for Execution of Problem Response. In such applications, it is sufficient to meet the functional and interface requirements of our design without utilizing advanced software techniques to implement these requirements.

Our approach to Execution of Problem Response combines functions from the Planning, Execution, and State Determination components into a Planning-Execution-Monitoring (PEM) module. A revised mission plan is provided by Mission Planning described in the previous section. The Task Selection function in the Planning component identifies what task(s) in the plan are ready to be executed. These tasks are passed to the Execution component. The Execution component dispatches these tasks to either another Execution component for further task decomposition or, if tasks correspond to commands, to the RTSA component that interfaces to system hardware. Once the hardware command has been executed, the Condition Monitoring function in State Determination monitors for signals indicating the command has its expected effect. Once these effects are observed, Plan Monitoring notifies Execution and it marks the task as complete. Execution passes the task completion status to Task Selection in the Planning component, which determines the next task to be executed. This cycle continues until all tasks in the plan have been executed or until the execution of a task does not have the intended effect on the system.

When Plan Monitoring detects that a task did not have the intended effect, it notifies the Plan Modification function in the Planning component. Plan Modification tries to repair the plan by identifying what part of the plan did not have the expected effect and trying to find an alternative way to achieve the expected effect. If an alternative is found, the adjusted plan is passed to the Task Selection function. Task Selection determines the next task to be executed and nominal operations continue. If an alternative is not found, then Plan Execution fails which requires that a new plan be generated by Plan Generation.

Humans participate in the execution of problem response. In all cases the human supervises the response. For some actions, human consent is required before the response can be taken. And some actions always are done by the human. The PEM module is designed to support setting the level of automation appropriate to the risk associated with the response.

For some applications, the PEM modules interact as peers when responding to a problem. Peer interactions supported by the Health Management design include (1) querying a PEM module for information about a task or plan it is executing, (2) requesting a PEM module to perform a task or plan, (3) declining the request to perform a task or plan, and (4) negotiating among PEM modules to determine which module will perform a task .

Negotiation to allocate tasks is accomplished using a contract net approach [21]. The task to be allocated is posted to all PEM modules with a request to evaluate whether the module can perform the task. All modules receiving the proposed task either (1) decline to perform the task, or (2) make a bid describing how it would perform the task. Bids consist of the task originally passed to the module with the variables in task constraints filled in by the bidding module. For example, if a task must be completed by some deadline, the PEM module fills in when it can complete the task. These constraints can be weighted according to the importance assigned by the requesting module. The Plan Modification function in the responding module interacts with its Plan Evaluation function to construct a bid. All bids are posted to the module requesting the task be performed. The Plan Modification function in the requesting module interacts with its Plan Evaluation function to evaluate a bid. These bids are evaluated by comparing the constraint bindings among the different bids, since they represent how well an agent can meet the required constraints. The bid is awarded to the module meeting the constraints most effectively. The task is passed back to the winning module with the direction to execute the task. Results from executing the task are returned to the requesting agent. At any time the requesting module can query for an update on the status of the task performance.

Conclusions and Future Work

The need for increased crew autonomy for exploration missions will require automating aspects of systems health management so the crew can respond effectively with less reliance on Earth. Health management requires (1) detecting and diagnosing problems, (2) taking immediate action in response to problems to minimize impacts to safety and mission, (3) determining how to recover from a problem, and (4) implementing the recovery. Yet most health management technologies address problem detection and diagnosis, leaving impact assessment and problem recovery to humans. The health management architecture we propose to develop addresses this technology gap by developing integrated software that supports every phase of problem detection and response. Humans still direct problem response, but they are better supported in doing their job.

Our approach to providing health management is novel in that it integrates health management technologies in support of problem detection with technologies for intelligent monitoring and control and human-computer interaction in support of problem response. This integration of technologies leverages a large body of advanced computing research while providing a solution that is fundamentally human-centric. Our approach is general in that the components are designed for use in a variety of applications, from single systems to complex spacecraft consisting of multiple, coupled systems.

A defining feature of our approach is that the functions of these components can be performed by humans as well as software. We expect that different application domains will have different requirements for human participation in health management. Our design specifies the necessary functions and interfaces, but does not require that these capabilities be delivered by software.

Another defining characteristic of our approach is that the functions of these components can be performed by software already implemented in the application domain. For example, spacecraft avionics systems will provide many health management functions. In such applications, we specify the functional and information requirements for compatibility with our design but do not mandate how these functions and interfaces are implemented.

The health management architecture that we have described has not yet been implemented. We plan to implement and evaluate our architecture design using a combination of commercial, government-provided, and custom software. The interfaces defined for these components will be implemented using the commercial middleware software Common Object Request Broker Architecture (CORBA). Once we have implemented the core components, we will apply them to example problems for spacecraft and robotic health management. A key aspect of this integration is providing capability for humans to interact with the architecture. Finally we plan to evaluate how well the implemented architectures support managing problems in spacecraft and robotic systems.

Acknowledgements

This work was funded under a NASA Small Business Innovative Research (SBIR) contract. We would like to recognize the contributions of Lui Wang at Johnson Space Center in designing this architecture.

References

1. Barrett, A. Model Compilation for Real-Time Planning and Diagnosis with Feedback, *Proceedings of IJCAI 2005*, Edinburgh, Scotland, July 2005, pp. 1195-1200.
2. Bonasso, R. P., Firby, J. R., Gat, E., Kortenkamp, D., Miller, D. P., and Slack, M. G. Experiences with an Architecture for Intelligent, Reactive Agents. *Journal of Experimental and Theoretical Artificial Intelligence*, 9 (1997). 237-256.
3. Bresina, J., A. Jónsson, P. Morris, & K. Rajan. Mixed-Initiative Planning in MAPGEN: Capabilities and Shortcomings. *International Conference on Automated Planning and Scheduling*. Monterey, CA. June 2005.
4. Brookshire, J., S. Singh, and R. Simmons. Preliminary Results in Sliding Autonomy for Coordinated Teams. *Proceedings of the 2004 Spring Symposium Series*, March, 2004.
5. Chien, S., G. Rabideau, R. Knight, R. Sherwood, B. Engelhardt, D. Mutz, T. Estlin, B. Smith, F. Fisher, T. Barrett, G. Stebbins, D. Tran , "ASPEN - Automating Space Mission Operations using Automated Planning and Scheduling," *SpaceOps. 2000*.
6. Crocker, A. *ISS EPS Power Bus Reconfiguration Operational Scenario*. JSC Internal Document. June 18, 2002.
7. Dixon, R. W., T. Hill, W. Kahle, A. Patterson-Hine, K.A. Williams, S. Hayden. Demonstration of an SLI Vehicle Health Management System With In-Flight and Ground-based Subsystem Interfaces. In *Proceedings 2003 IEEE Aerospace Conference*, Big Sky, Montana, Mar 2003.
8. Dorais, G. A., Bonasso, R. P., Kortenkamp, D., Pell, B., and Schreckenghost, D. Adjustable Autonomy for Human-Centered Autonomous Systems on Mars. In *Proceedings of Mars Society Conference*, 1998
9. Elsaesser, C., & MacMillan, T.R. 1991. *Representation and Algorithms for Multiagent Adversarial Planning*. Technical Report MTR-91W000207. MITRE, Wash. D.C.
10. Ferguson, G. and Allen, J. F. TRIPS: An Integrated Intelligent Problem-Solving Assistant. In *Proceedings of 15th National Conference on Artificial Intelligence*. Madison, WI, 1998
11. Fletcher, D. P., R. Alena. A Scalable, Out-of-Band Diagnostics Architecture for International Space Station

Systems Support. *2003 IEEE Aerospace Conference*, Big Sky Montana, March 8-15, 2003.

12. Gat, E. 1998. Three-Layer Architectures. In *Mobile Robots and Artificial Intelligence*, Kortenkamp, D., Bonasso, R. P., and Murphy, R., Eds. Menlo Park, CA: AAAI Press, 195-210..

13. Mackey, R., James, M., Park, H. , and Zak, M. "BEAM: Technology for Autonomous Self-Analysis", *IEEE Aerospace Conference*, 2001, vol. 6, March 2001, pp. 2989-3001.

14. Malin, J.T. Throop, D.R. Fleming, L. Flores, L. Computer-aided identification of system vulnerabilities and safeguards during conceptual design. *Proceedings of IEEE Aerospace Conference*, 2004. 6-13 March 2004. Vol 6. pp 3873- 3884

15. Malley, M. E.. *A Methodology for simulating the Joint Strike Fighter's (JSF) Prognostics and Health Management System*. Thesis, Air Force Iinstitute of Technology, Wright-Patterson Air Force Base, OH, 2001.

16. Muscettola, N., P.P. Nayak, B. Pell, and B. Williams. Remote Agent: To Boldly Go Where No AI System Has Gone Before. *Artificial Intelligence*, Vol 100. 1998.

17. Muscettola, N., G. Dorais, C. Fry, R. Levinson, and C. Plaunt, "Idea: Planning at the core of autonomous reactive agents," in *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*, Oct 2002.

18. Myers, K. , Tyson, M.., Wolverton, M., Jarvis, P., Lee, T., and desJardins, M. PASSAT: A User-Centric Planning Framework. In *Proceedings of 3rd International NASA Workshop on Planning and Scheduling for Space*. Houston, TX. 2002.

19. Narasimhan, Sriram*.* *Hybrid Diagnostic Engine*. Briefing at Ames Research Center. October 2005.

20. Nesnas, I.A., R. Simmons, D. Gaines, C. Kunz, A. Diaz-Calderon, T. Estlin, R. Madison, J. Guineau, M. McHenry, I. Shu, and D. Apfelbaum, "CLARAty: Challenges and Steps Toward Reusable Robotic Software," *International Journal of Advanced Robotic Systems*, Vol. 3, No. 1, pp. 023-030, 2006.

21. Reeves, D. M., Wellman, M. P., and Grosof, B. N. 2001. Automated Negotiation from Declarative Contract Descriptions. In *Proceedings of Fifth International Conference on Autonomous Agents* (Agents-2001), 51-58. Montreal, QC, Canada: ACM Press.

22. Robinson, P. *A First Cut of ISHM Requirements for CEV*. Briefing at Ames Research Center. May 2006.

23. Scerri, P., Pynadath, D. V., and Tambe, M. Adjustable Autonomy in Real-World Multi-Agent Environments. In *Proceedings of Autonomous Agents* (Montreal, Canada, 2001). ACM Press. 300-307.

24. Schaefer, P., Colgren, R., Abbott, R., Park H., Fijany, A., Fisher, F., James, M., Chien, S., Mackey, R., Zak, M., Johnson, T., and Bush, S. Reliable Autonomous Control Technologies for Uninhabited Air Vehicles, *IEEE Aerospace Conference*, 2001, vol. 2, March 2001, pp. 677-684.

25. Shirley, M., and T. Cochrane. SimStation: A knowledge-integrating virtual vehicle. *NASA Workshop on Virtual Iron Birds*. Monterey, CA. April 2004

26. Tumer, I.Y., and R. B. Stone. Mapping Function to Failure during High-Risk Component Development. *Journal of Research in Engineering Design*, 14:25–33, 2003.

27. Wilkins, D. & Myers, K. A Multiagent Planning Architecture. In *Proceedings of Artificial Intelligence Planning Systems* (Pittsburg, PA, 1998). 154-162.

28. Williams B. C., and Nayak, P. P. A Model-based Approach to Reactive Self-Configuring Systems, *Proceedings of 13th National Conference of Artificial Intelligence*. (AAAI-96), vol. 2, 1996, pp. 971-978.