A Comparison of Fault-Tolerant GN&C System Architectures Using the Object Process Network (OPN) Modeling Language

Gregor Z. Hanuschak[1]
Massachusetts Institute of Technology, Cambridge, MA 02139

Nicholas A. Harrison[2]
Charles Stark Draper Laboratory, Cambridge, MA 02139-4307

Edward F. Crawley[3] and Steven R. Hall[4]
Massachusetts Institute of Technology, Cambridge, MA 02139

Alejandro D. Domínguez-García[5]
University of Illinois at Urbana-Champaign, Urbana, IL 61801

John J. West[6]
Charles Stark Draper Laboratory, Cambridge, MA 02139-4307

and

Cornelius J. Dennehy[7]
NASA Engineering & Safety Center, Greenbelt, MD, 20771

This paper summarizes the final results of a study analyzing different Guidance, Navigation and Control (GN&C) architectural approaches for fault tolerance in National Aeronautics and Space Administration's (NASA's) crewed and robotic exploration space systems. GN&C systems were decomposed into simple building block subunits of sensors, computers, and actuators and various forms of subunit interconnection were defined for investigation. The resulting subunit/interconnection construct was used as a top-level abstraction for building candidate GN&C system architectures. This model was implemented using Massachusetts Institute of Technology's (MIT's) Object Process Network (OPN) modeling language in order to more easily enumerate possible architectures and ultimately identify which of these architectures have optimal properties. Dual and triple redundant GN&C system architectures, employing different classes of components, were modeled using the OPN language. The model assumed perfect coverage – 100-percent accuracy in detecting and isolating a failure. Within the constraints of the model, all possible architectures were rigorously enumerated and the weight/reliability trade-offs of cross-strapping components and using more than one type of component were assessed. The study results indicate it is possible to produce nearly all potentially optimal GN&C architectures using generic connections between low-reliability components. The identified optimal architectures reveal a preference to increase GN&C system redundancy of lighter, less reliable components rather than using smaller numbers of more reliable, heavy components.

[1] Graduate Research Assistant, Department of Aeronautics and Astronautics, 77 Massachusetts Avenue, Room 33-409, and Student Member AIAA.
[2] Avionics Systems Engineer, Space Systems Department, 555 Technology Square.
[3] Ford Professor of Engineering, Department of Aeronautics and Astronautics, and Engineering Systems Division, 77 Massachusetts Avenue, Room 33-413, and Fellow AIAA.
[4] Professor, MacVicar Faculty Fellow, Department of Aeronautics and Astronautics, 77 Massachusetts Avenue, Room 33-313, and Associate Fellow AIAA.
[5] Assistant Professor, Department of Electrical and Computer Engineering, 1406 W. Green Street, MC-702, and Member AIAA.
[6] Program Manager, Space Systems Department, 555 Technology Square.
[7] NASA Technical Fellow for GN&C, NESC, Goddard Space Flight Center, Mail Code 590.

Nomenclature

CEV	=	Crew Exploration Vehicle
CLV	=	Crew Launch Vehicle
CxP	=	Constellation Program
GN&C	=	Guidance, Navigation and Control
IMU	=	Inertial Measurement Unit
MIT	=	Massachusetts Institute of Technology
NASA	=	National Aeronautics and Space Administration
NESC	=	NASA Engineering and Safety Center
OPN	=	Object Process Network

I. Background

AT the core of NASA's future space exploration is a return to the Moon, where we will build a sustainable long-term human presence. As the Space Shuttle approaches retirement and the International Space Station nears completion, NASA's Constellation Program (CxP) is designing and developing the next fleet of American space-faring vehicles to bring astronauts back to the Moon, and possibly to Mars and beyond. In order to meet their exploration goals, NASA's CxP will have to acquire and operate a number of new human-rated systems, such as the Orion Crew Exploration Vehicle (CEV), the Ares-1 Crew Launch Vehicle (CLV), and the Altair Lunar Lander, along with other elements for crew transportation (e.g., in-space propulsion stages), lunar habitation, and mobility. Robotic systems will include lunar robotic orbiter vehicles and robotic lunar landers. Commonality in exploration system hardware, and software elements offers the opportunity to significantly increase sustainability by reducing, both nonrecurring and recurring cost and/or risk. In particular the potential benefit of common GN&C avionics and flight software is considerable, not only in the initial development effort, but in validation and verification, and more importantly in the ongoing maintenance efforts and incremental upgrades that will occur over the life cycle of these exploration spacecraft. With commonality of the onboard components of this system, there is more likelihood that ground control and communications systems could be made more common, yielding a multiplier effect. This paper summarizes the final results of a comparative assessment of robotic and human-rated GN&C system architectural approaches. This study was performed by a combined MIT and Draper Laboratory team as part of a proactive GN&C "discipline-advancing" activity sponsored by the NASA Engineering and Safety Center (NESC).

This study effort was primarily driven by the observation, both on the part of NESC and MIT, that GN&C systems for exploration prominently stand out among all the future spacecraft systems, as an area where commonality might be of greatest benefit. This comparative assessment of robotic and human-rated GN&C system architectural approaches was undertaken as a fundamental step towards understanding the opportunities and limitations of GN&C commonality across the CxP flight elements.

II. Introduction

CxP has created a need to develop new robotic and human-rated space systems. In an attempt to influence the design of the most collectively reliable and cost-efficient systems possible, the NESC sponsored a commonality study for GN&C systems through the MIT and Draper Laboratories. By modeling, enumerating, and comparing simplified GN&C architectures using simple metrics, this resulting paper presents sound reasoning for making certain architectural choices which, when implemented, would further these reliability and cost-efficiency goals.

In the 2007 AIAA paper, "A Comparison of GN&C Architectural Approaches for Robotic and Human-Rated Spacecraft" (Ref. 2), different architectural approaches for fault tolerance in guidance, navigation, and control (GN&C) systems were analyzed at the topmost level. The study broke down the GN&C systems into simple subunits, i.e., sensors, computers, and actuators, and analyzed how the components were interconnected. This paper expands upon the previous 2007 paper written by the authors. It uses the previous paper's subunit/interconnection construct as a top-level abstraction for building a preliminary model of GN&C system architectures.

Although never before used to model GN&C system architectures, the OPN modeling language was used with great success to model mission and hardware architectures (see Ref. 3 and Ref. 5 for details). OPN is a visual and computable meta-language that assists with systems architecting tasks. OPN is typically used to describe and partition the space of architectural alternatives, generate and enumerate the set of instances of feasible system models, and then simulate and order the performance metrics of each model. This language combines visual representation on Pareto plots with mathematical modeling and provides a modeling framework in which it is relatively easy to add new options to understand the effect of new technologies and different configurations. Moreover, as proven in this study, the OPN language is applicable to many "levels" of a given architecture.

Candidate GN&C system architecture models were implemented using the OPN modeling language in order to more easily enumerate possible architectures and ultimately identify which architectures have optimal properties. Following the basic procedure employed in the above references and using Ref. 2 to provide the background for the top-level abstraction used in the model, OPN was successfully employed in the models described in this paper.

Partial 2 x 2 systems (i.e., systems with up to dual redundancy per component class for two component classes) and 3 x 2 systems (systems with up to triple redundancy per component class for two component classes) were modeled in OPN. Within the constraints of these models, all possible architectures were rigorously enumerated and the weight/reliability trade-offs of cross-strapping components and using more than one type of component were assessed.

The described models assume perfect coverage – 100-percent accuracy in detecting and isolating a failure. The models also assume that more reliable components tend to be heavier, more costly, and/or more complicated to deal with. Given these assumptions, it was found that more reliable components are only beneficial in single string systems or systems with single point failures. All optimal architectures employing component redundancy could be produced from generic connections and the least reliable type of component from each component class.

According to Ref. 2, a GN&C system can be represented with sensors, computers, actuators, and how these components are interconnected. Given this abstraction, the completed OPN model discussed in this paper represents all possible GN&C architectures within a given set of constraints. The constraints are defined as the number of component classes, the maximum component redundancy in each component class, and the number of component types for each class.

In this paper, sensors, computers, and actuators will be defined as "component classes." The terminology "I x J OPN model" will be used to describe a model with up to "I" redundancy per component class and up to "J" component classes. In other words, J = 2 could designate a model which only has sensors or which has both sensors and computers. J = 3 could designate a model with sensors, with sensors and computers, or with sensors, computers, and actuators. If J = 3 and I = 2, this could designate a system with up to two sensors, two computers, and two actuators. This paper will discuss OPN 2 x 2 and 3 x 2 models and touch on their applicability to a 3 x 3 model.

For the purpose of simplicity, it will be assumed that there are only three different types of components possible for each component class. In reality, three sensor types might include a sun sensor, star tracker, and an Inertial Measurement Unit (IMU). However, to be more generic, types will not be designated so specifically – they will instead be referred to as type A, type B, and type C.

As a first pass, all enumerated architectures are evaluated based on two specific metrics: reliability and weight. Note that, with some exceptions, both complexity and cost increase as weight increases; thus, weight is a good first order approximation for these metrics.

Section III of this paper will discuss the design of the simple 2 x 2 model, section IV will give further details on the model, and Section V will discuss the design of the more complicated 3 x 2 model. Section VI will examine the application of reliability and weight metrics to the enumerated architectures. Finally, Section VII concludes and describes the model's future iterations.

III. A "2 x 2" GN&C System

This section begins the discussion of the design of the 2 x 2 model. Even with just four components (two sensors and two computers), many architectures can be defined for a 2 x 2 system based on how the components are interconnected. Each of these architectures will have different total weight and different total reliability.

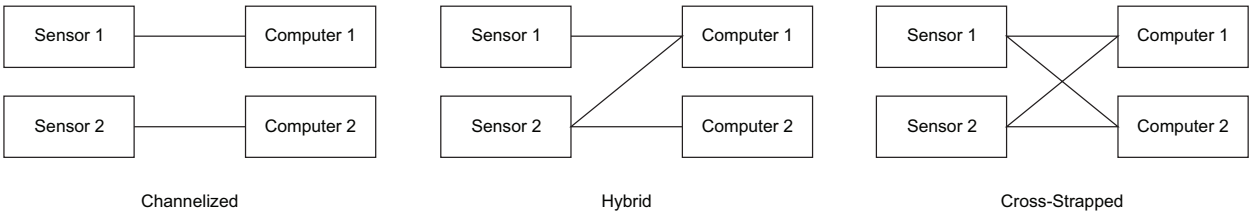


Figure 1. Three possible 2 x 2 systems.

Figure 1 depicts three possible 2 x 2 architectures. The reliability R of the three models is shown in Table 1 where s_j is the reliability of sensor j and c_k is the reliability of computer k.

Table 1. Reliability expressions for the 2 x 2 systems in Figure 1.

Architecture	Reliability
Channelized	$R = s_1c_1 + s_2c_2 - s_1c_1s_2c_2$
Hybrid	$R = s_1c_1 + s_2c_1 + s_2c_2 - s_2c_1c_2 - s_1s_2c_1 - s_1s_2c_1c_2 + s_1s_2c_1c_2$
Cross-Strapped	$R = s_1c_1 + s_1c_2 - s_1c_1c_2 + s_2c_1 + s_2c_2 - s_2c_1c_2 - s_1s_2c_1 - 2s_1s_2c_1c_2 + 2s_1s_2c_1c_2 - s_1s_2c_2 + 2s_1s_2c_1c_2 - s_1s_2c_1c_2$

It is important to note that, no matter what the architecture, the reliability of any 2x2 model can be generated by taking the cross-strapped expression for R and then eliminating terms from the expression for connections which do not exist and therefore do not contribute to system reliability.

Additional indicator variables are added to the cross-strapped reliability expression to specify which terms to eliminate. These indicator variables are correlated with the interconnections between components. A nonzero indicator variable represents a connection whereas an indicator variable equal to zero represents a missing connection.

Using the methodology described, the following general expression for R is obtained:

$$R = s_1i_{11}c_1 + s_1i_{12}c_2 - s_1i_{11}i_{12}c_1c_2 + s_2i_{21}c_1 + s_2i_{22}c_2 - s_2i_{21}i_{22}c_1c_2 - s_1s_2i_{11}i_{21}c_1 - s_1s_2i_{11}i_{22}c_1c_2 - s_1s_2i_{12}i_{21}c_1c_2 + s_1s_2i_{11}i_{12}i_{21}c_1c_2 + s_1s_2i_{11}i_{21}i_{22}c_1c_2 - s_1s_2i_{12}i_{22}c_2 + s_1s_2i_{11}i_{12}i_{22}c_1c_2 + s_1s_2i_{12}i_{21}i_{22}c_1c_2 - s_1s_2i_{11}i_{12}i_{21}i_{22}c_1c_2$$

where i_{jk} is the reliability of the connection between sensor j and computer k if such a connection exists and is 0 otherwise.

As a sanity check, the reliability expressions for the channelized and hybrid architectures above can be derived from the general expression. Assuming perfect connection reliability, i.e., $i_{jk} = 1$ for all connections in the architecture, the channelized and hybrid architectures would be represented by the indicator variables in Table 2. Plugging these indicator variables into the general expression gives the same reliability expressions in Table 1.

Table 2. Indicator for the channelized and hybrid 2 x 2 systems in Figure 1.

Architecture	i_{11}	i_{12}	i_{21}	i_{22}
Channelized	1	0	0	1
Hybrid	1	0	1	1

IV. Details on the Model

This section gives further detail on the 2 x 2 model. Like the previously mentioned 3 x 2 and 3 x 3 models, the 2 x 2 OPN model can be viewed as a sophisticated Petri net model. In a Petri net model, information-storing tokens move via directed arcs from transitions to places and from places to transitions. Note that there may be more than one directed arc feeding from or to a transition or place. Upon arrival at a transition, a token is consumed, some processing is done, and, if appropriate, new tokens are introduced in the places dictated by the directed arcs leading from the transition.

The sequence of transitions in any of the discussed OPN models is a sequence of decision points. At each decision point, a token is replicated with multiplicity equal to the number of possible decisions. The information stored in each token represents a unique possible architecture. Taken together, the tokens enumerate all possible

architectures given an initial set of constraints. All tokens are collected when they completely propagate through the model for analysis.

Figure 2 is a visual representation of the OPN decision tree for the 2 x 2 model and the following questions are the decision points:

- How many sensors?
  - 1 or 2
- Type assignment for sensors?
  - If only one sensor, choose SensorA, SensorB, or SensorC
  - If two sensors, choose two of the same type of sensor or one of each of two types (possible combinations: AA, AB, AC, BB, BC, and CC)
- How many computers?
  - 1 or 2
- Type assignment for computers?
  - If only one computer, choose ComputerA, ComputerB, or ComputerC
  - If two computers, choose two of the same type of computer or one of each of two types (possible combinations: AA, AB, AC, BB, BC, and CC)
- Which sensors are connected to computer 1?
  - Just sensor 1
  - Just sensor 2 (if sensor 2 exists)
  - Both sensor 1 and 2 (if sensor 2 exists)
- If computer 2 exists, which sensors are connected to computer 2?
  - Just sensor 1
  - Just sensor 2 (if sensor 2 exists)
  - Both sensor 1 and 2 (if sensor 2 exists)

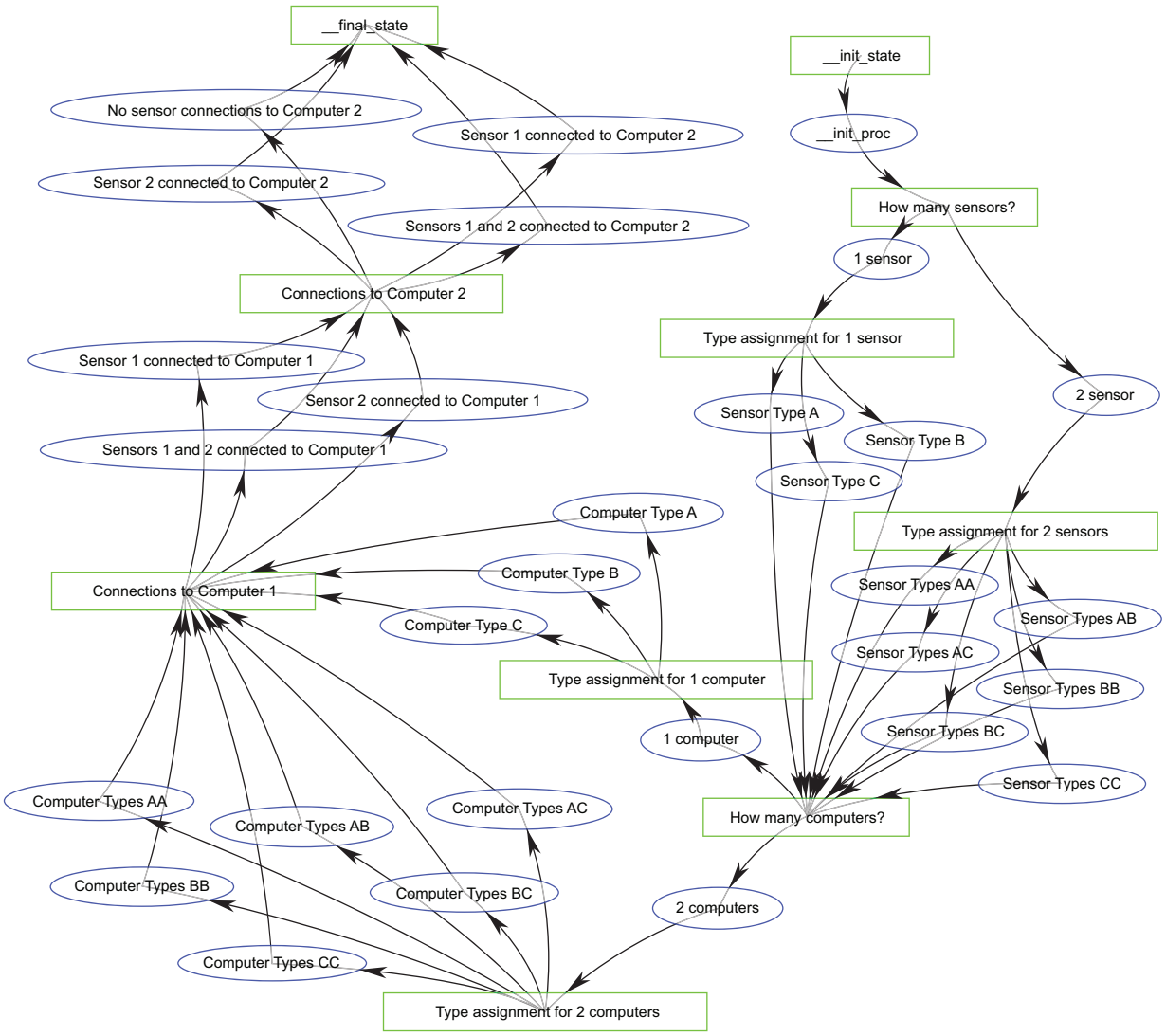


Figure 2. The 2 x 2 OPN decision tree.

During the process of token propagation, the number of components, component types, and connections are continuously updated for later use in reliability calculations. In addition, the current weight of the system is updated at execution time.

Each component type is given its own unique reliability and weight based on the specific make and model of the component. These values were based on real components, but modified slightly to facilitate analysis.

Reliabilities are dependent upon the failure rate of the component and the desired operational time for the component. The relationship is governed by the equation $R = e^{-\lambda t}$, where λ is the failure rate and t is the operational time. Operational time is user defined and based on the length of the proposed mission. An operational time of t = 10 years was used in the models discussed in this paper. Other component properties are illustrated in Table 3 and Table 4.

Table 3. Component properties for sensor types A, B, and C.

Sensor Type	A	B	C
Failure Rate λ (/year)	0.00015	0.0001	0.00005
Reliability R	0.9985	0.999	0.9995
Weight (dimensionless)	3	6	9

Table 4. Component properties for computer types A, B, and C.

Computer Type	A	B	C
Failure Rate λ (/year)	0.0001	0.00004	0.00002
Reliability R	0.999	0.9996	0.9998
Weight (dimensionless)	3	5	10

Two additional weights and one additional reliability were also included in the model. A "connection weight" and a "dissimilar component penalty" were included to ensure that weight continues to approximate complexity and cost. Cross-strapping components may not add much physical weight to the overall system, but it surely adds to the complexity and cost of the system. Similarly, dealing with more than one type of sensor and/or computer also increases complexity and cost. Hence, adding these additional weights where appropriate worked as a first step toward reality.

The weights associated with connections and dissimilar components were chosen to be consistent with the weights of sensors and computers. To do so, assumptions had to be made. Connections were considered to be, at most, one-third of the complexity of the average computer. In addition, the weight penalty for dissimilar components was set such that it was not larger than the heaviest sensor or the heaviest computer.

These logical assumptions dictated a certain range of weight values used for connections and dissimilar component parameters. However, rather than presuppose exact values for these weights, multiple OPN runs were executed varying one of the parameters each time. Assuming the connection reliability would be greater than that of a computer, the nine OPN scenarios are illustrated in Table 5.

Table 5. Connection reliabilities, connection weights, and dissimilar component penalties for each OPN scenario run.

OPN Scenario	1	2	3	4	5	6	7	8	9
Connection Reliability	1	1	1	0.99995	0.99995	0.99995	0.9999	0.9999	0.9999
Connection Weight (dimensionless)	0	0	0	0.5	0.5	0.5	5 / 3	5 / 3	5 / 3
Dissimilar Component Penalty (dimensionless)	0	6	9	0	6	9	0	6	9

V. A "3 x 2" GN&C System

This section discusses the design of the 3 x 2 model. Implementation of the 3 x 2 OPN model is very similar to that of the 2 x 2 model with two notable exceptions. These exceptions relate to the reliability formula for the overall system and the removal of duplicate architectures to conserve memory.

The reliability formula is much more complicated for these larger models and must be handled differently. Although reliability is still calculated after OPN completes execution, it can no longer be easily calculated by hand for implementation in Excel. Instead, symbolic MATLAB was used to multiply out the formula and a MATLAB script was used to insert the correct "i" indicator values where appropriate. Only after this manipulation was performed could the reliability be imported back into Excel for implementation.

In addition, care had to be taken to ensure no architecture was represented more than once in the model. Running a larger 3 x 2 OPN model would take an inordinate amount of time and computer memory. It was found that certain architectures could be represented in multiple configurations and this was not taken into account by the 2 x 2

model.[8] By producing tokens for all possible configurations of the same architecture, the model took much more time and used much more memory than necessary.

An example duplicate architecture is shown in Figure 3. A1 and A2 represent the same architecture since, in both cases, one sensor of type A is connected to a computer of type A, the other sensor A is connected to a computer A and a computer B, and a sensor of type B is connected to a computer of type C. A3 represents a different architecture, however, since both sensors of type A are connected to a computer of type B.

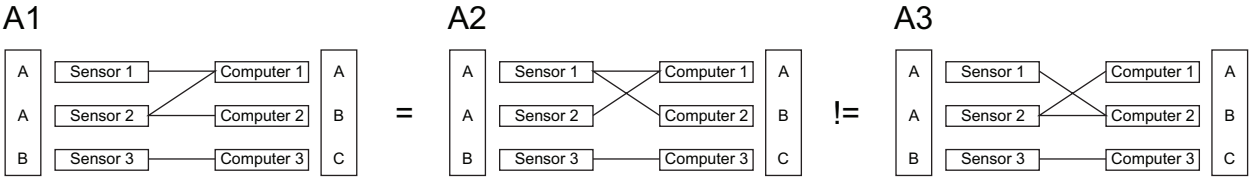


Figure 3. Determination of duplicate architectures.

The process of eliminating duplicate architectures began by choosing a representative set of sensor types and computer types. Ten possibilities were chosen as representative orderings of the three sensor types and also the three computer types: AAA, AAB, AAC, ABB, ACC, ABC, BBB, BBC, BCC, and CCC.

A1 in Figure 3 represents a connection pattern between three adjacent components. This connection pattern has four connections: Sensor 1 is connected to computer 1, sensor 2 is connected to both computer 1 and computer 2, and sensor 3 is connected to computer 3. Keeping this connection pattern fixed, we can give a "type" identity to the three sensors based on the ten possible orderings. For each possible ordering of the sensor types, there are ten possible orderings of the computer types, each of which defines a unique architecture. In other words, for any given connection pattern such as A1, there are 10 x 10 = 100 possible architectures. The OPN model iterates through all possible connection patterns and finds all 100 possible architectures for each one.

Note that orderings such as ABA and BAA are not taken to be representative orderings. When all possible connection patterns are taken into account, these additional orderings will fail to produce any architecture that cannot be produced by AAB. This is because ABA, BAA, and AAB are all equivalent – all represent two components of type A and one of type B. It does not matter in what order the letters are written as long as the case is represented.

Luckily, searching for duplicate architectures in OPN does not require checking 100 possible architectures for each connection pattern. The 100 possibilities for each connection pattern can be represented by just 16 representative architectures. As illustrated in Figure 4, the ten sensor type combinations and the ten computer type combinations can be further abstracted to just four representative combinations per component. First, AAA, BBB, and CCC all represent the case where all three components are of the same type. Next, AAB, AAC, and BBC all represent the case where the first two components are of the same type and the third component is of a different type. Furthermore, ABB, ACC, and BCC all represent the case where the second and third components are of the same type, but the first component is of a different type. Finally, ABC represents the case where all three components are of a different type.

[8] The 2 x 2 model is much smaller than the 3 x 2 model. As a result, there were no memory issues and duplicate architectures could be removed in post-processing – they did not have to be removed in OPN.

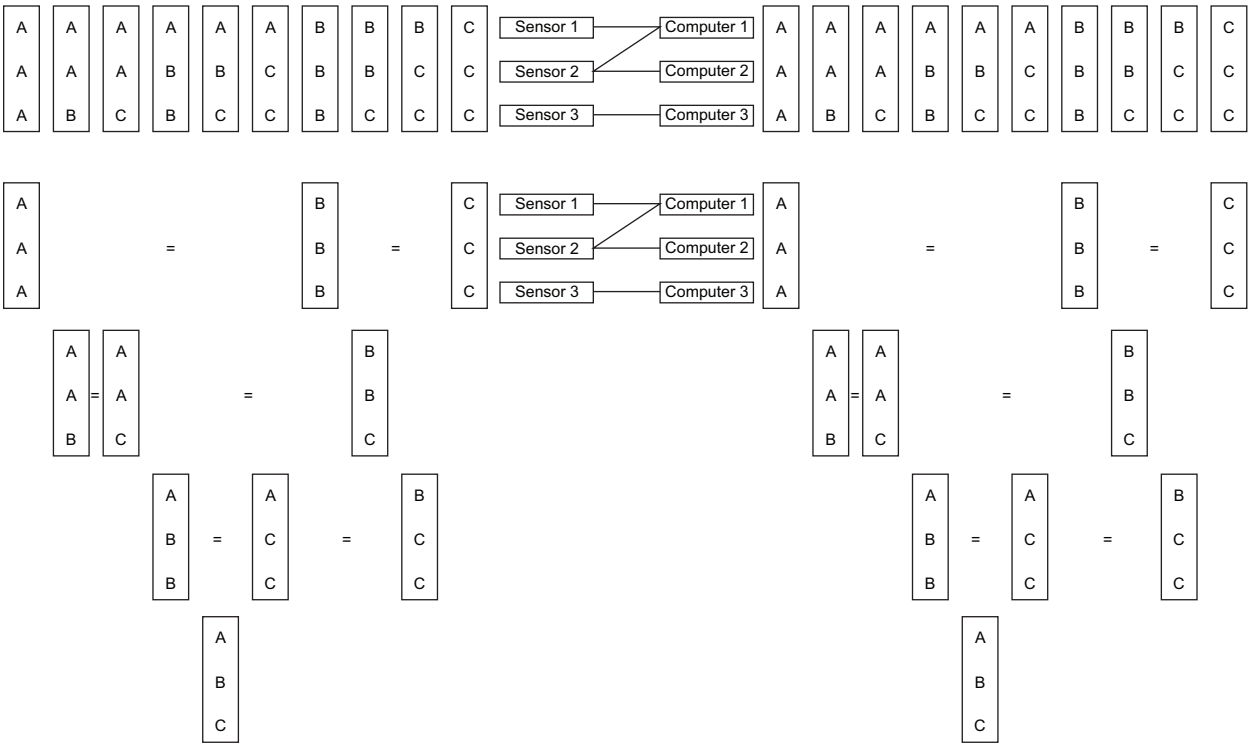


Figure 4. Finding representative architectures.

Figure 5 helps demonstrate why these representative architectures work for finding duplicate architectures. To use representative architectures to find duplicates is to claim that if architecture A1 is equivalent to architecture A2, but not A3, then architecture B1 is equivalent to B2, but not B3. This is clearly the case.

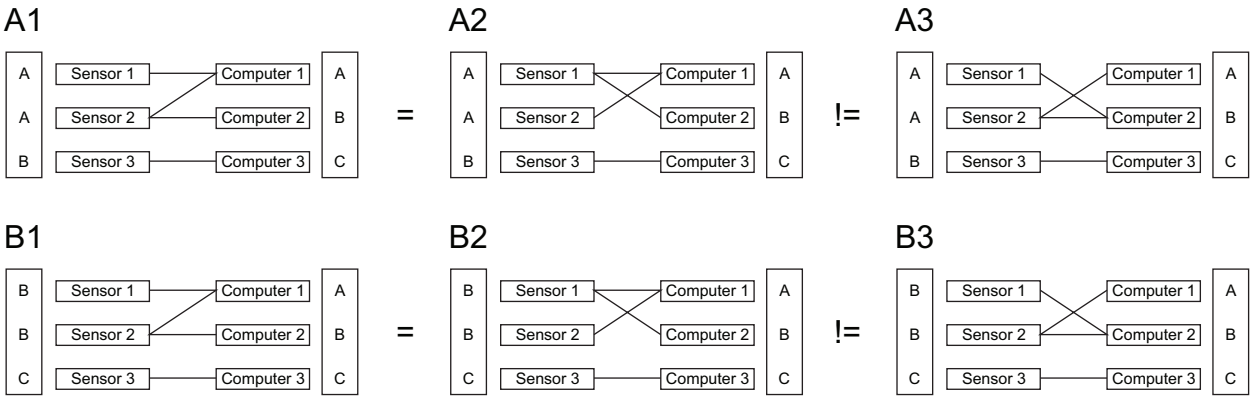


Figure 5. Using representative architectures to find duplicates.

As previously discussed, A1 and A2 represent the same architecture even though they have different connection patterns. It is arbitrary which form of an architecture is chosen as the primary form and which is a duplicate – either A1 or A2 could be considered the duplicate.

Implementing duplicate detection into the model turned out to be a very involved process. The project had a limited time horizon and the development time necessary for automating the duplicate detection process was uncertain. It was therefore decided that a surefire yet brute force method would be used to implement duplicate detection. All possible representative architectures were drawn by hand and duplicate architectures were circled. In all, over 100 pages of architectures were drawn and compared.

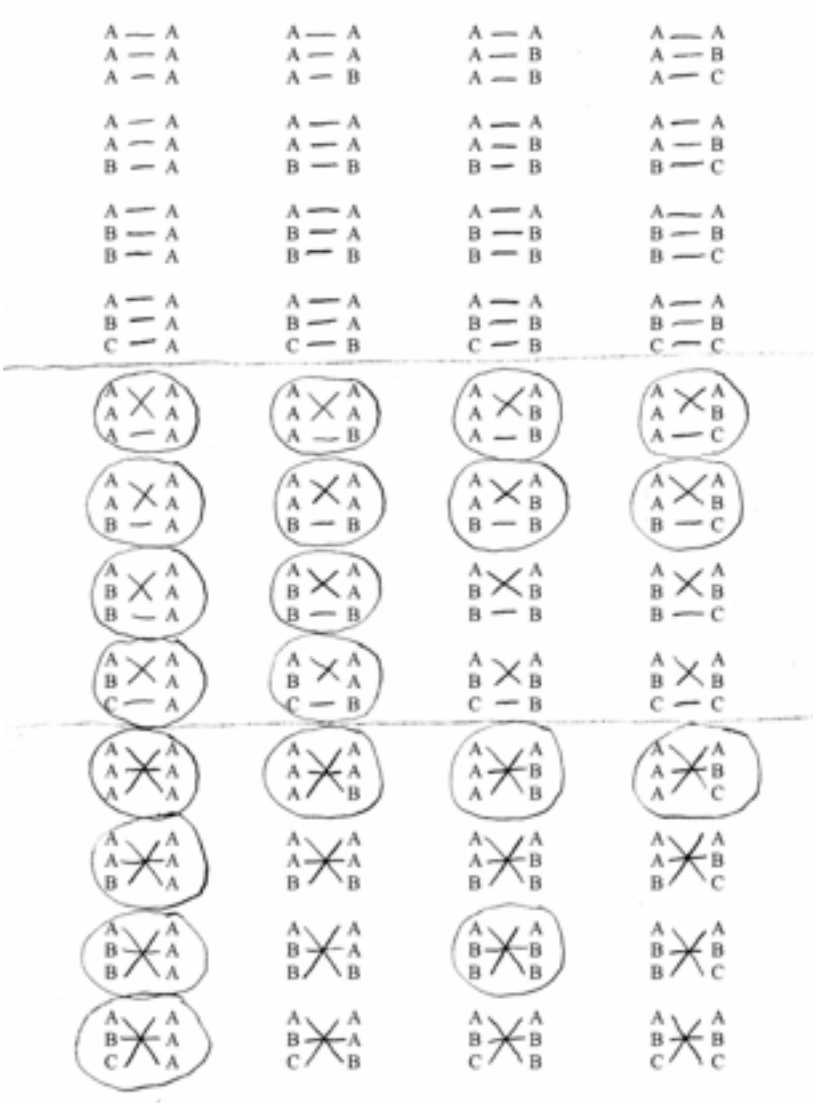


Figure 6. One page of hand-drawn architectures.

Based on the circled representative architectures, rules were created and inserted into OPN to keep any tokens that will produce duplicate architectures from propagating. Note that all rules are in the form of Boolean expressions starting with "not if" instead of "if". Although all work was double-checked, it is conceivable that an incorrect rule was entered due to human error. By using "not if" instead of "if", the default is to pass the token. It is better to retain a duplicate architecture rather than exclude a potentially optimal architecture.

The Boolean rules are inserted into the OPN model on the transitions from:

- Which sensors are connected to computer 1?
- If computer 2 exists, which sensors are connected to computer 2?
- If computer 3 exists, which sensors are connected to computer 3?

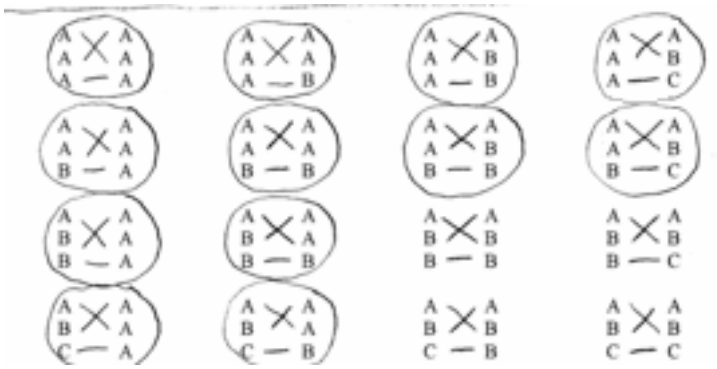
To the places:

- Just sensor 1
- Just sensor 2 (if sensor 2 exists)
- Just sensor 3 (if sensor 3 exists)
- Just sensors 1 and 2 (if sensor 2 exists)
- Just sensors 1 and 3 (if sensor 3 exists)
- Just sensors 2 and 3 (if sensor 3 exists)
- Sensors 1, 2, and 3 (if sensor 3 exists)

Trivial rules govern which sensors are connected to computer 1. If a particular token represents an architecture with only 2 sensors, it is not possible to make a connection to a computer from a nonexistent sensor 3. Therefore, in the 2-sensor case, no new tokens are introduced into the places representing, “just sensor 3,” “just sensors 1 and 3,” “just sensors 2 and 3,” or “sensors 1, 2, and 3.” Similarly, if a particular token represents an architecture with only 1 sensor, no new tokens are introduced into the places representing, “just sensor 2,” “just sensor 3,” “just sensors 1 and 2,” “just sensors 1 and 3,” “just sensors 2 and 3,” or “sensors 1, 2, and 3.”

Rules governing connections to computer 2 and computer 3 are more complicated. If a token represents an architecture with only two computers, it is known what the final system architecture will be after creating the sensor connections to the second computer. If a token represents an architecture with three computers, it is known what the final system architecture will be after creating the sensor connections to the third computer. Connections that will form duplicate (circled) architectures should not be allowed to propagate. Hence rules are put in place to block introduction of these tokens.

Figure 7 shows an example rule based on a hand-drawn architecture. This rule determines whether or not a connection should be made between sensor 3 and computer 3. Note that, by the time a token reaches the given rule, the connections between sensor 1 and computer 2 as well as between sensor 2 and computer 1 have already been defined. No connection should be made if the type definitions for the sensors and computers match those represented by the circled architectures – such tokens will result in the formation of duplicate architectures. In other words, the connection between sensor 3 and computer 3 should not be made if sensor 1’s type is the same as sensor 2’s type or computer 1’s type is the same as computer 2’s type.



```
!(Computer_Redundancy < 3) &&
!(Sensor_Redundancy < 3) &&
!(
      (i11 == 0 && i12 == 0) ||
      (Sensor_Redundancy > 1 && i21 == 0 && i22 == 0) ||
  (
            (i11 == 0 && i12 > 0 && i21 > 0 && i22 == 0 && i31 ==
            0 && i32 == 0) &&
            (
                  (Sensor1_Type == Sensor2_Type) ||
                  (Computer1_Type == Computer2_Type)
            )
  )
)
```

Figure 7. An example rule based on a hand-drawn architecture.

Eliminating duplicate architectures in the 3 x 2 model significantly reduced the number of tokens produced. Before duplicates were removed, the OPN produced 51,902 tokens. After duplicates were removed, the model produced only 9,795 tokens.

VI. Results

Despite eliminating the unnecessary duplicate architectures from the model, attempts to run a 3 x 3 model still resulted in a memory shortage. Although it is unfortunate that the larger model could not complete, it is important to note that results from the 3 x 2 model can still be applied to the 3 x 3 case.

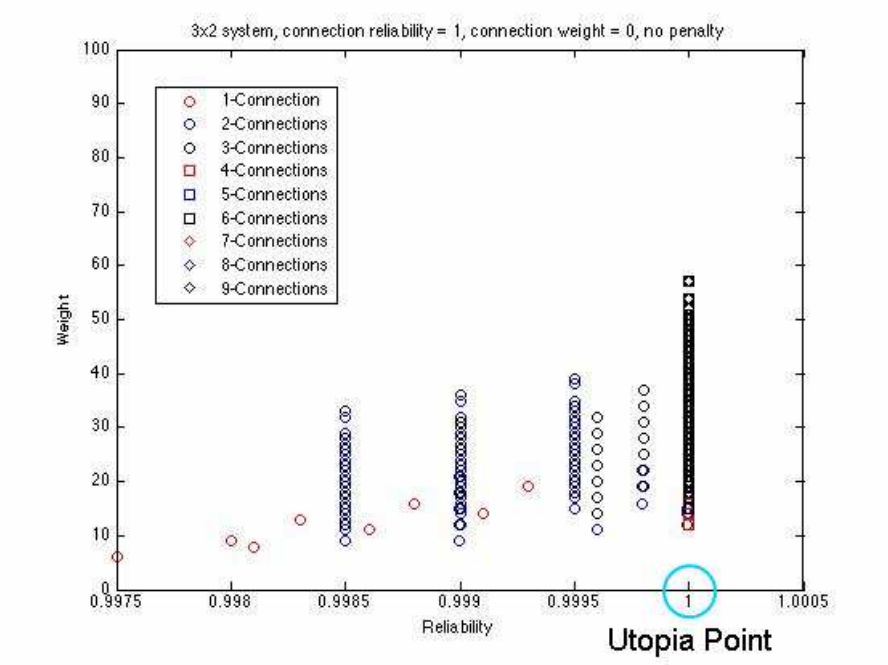
Taking such a top-level view of a GN&C system, the interaction between adjacent sensor and computer components is identical to the interaction between adjacent computer and actuator components. Just like sensors and computers, there are different types of actuators, each with a unique set of properties, and a system architect can choose different redundancies for each of these types. Furthermore, the connection patterns already found between sensors and computers are the same as those between computers and actuators. Finally, the metrics of weight and reliability can be calculated in exactly the same way.

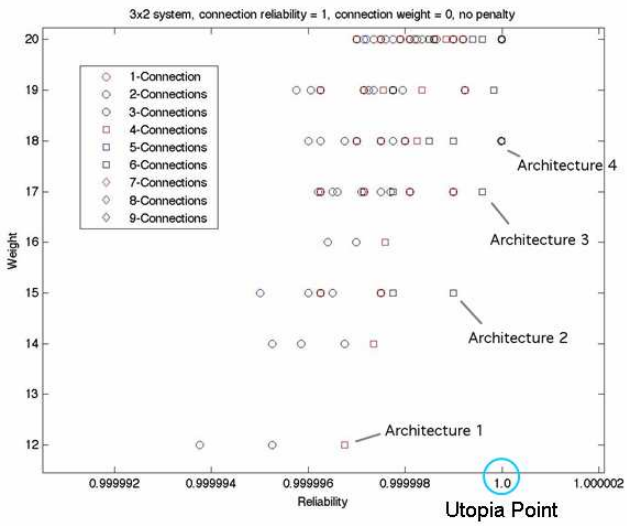
The nine scenarios outlined in section IV were run and Pareto plots were produced for each. Representative plots for each of the nine scenarios are reproduced in Figures 8 – 11. In each scenario, the architectures that simultaneously had both lowest weight and highest reliability were identified. These architectures are "on the Pareto front." The identified Pareto-front architectures for each scenario are reproduced in figures 8 – 11 as well. These architectures were found by zooming in on the "utopia point" at the lower right hand corner of the plot.[9] Note that, in most cases, there are multiple identified architectures for each scenario since it is somewhat subjective which architectures are closer to the utopia point. Is an architecture with weight = 17 and reliability = 0.999999596912468 (six "9"s) better than an architecture with weight = 18 and reliability = 0.999999995634079 (eight "9"s)? The answers to such questions would be made clear in a mission requirements context. For a human-rated mission, perhaps a reliability of 0.9999999 (seven "9"s) is required for safety. If this were the case, the architecture with weight = 18 would clearly be better, since the one with weight = 17 does not meet the requirement.

In the zoomed out (top) plot of each of the nine scenarios, there appear to be six clusters of architecture data points. Although the clusters appear to be columns of data points, they are not; the architectures in each cluster have nearly identical, but not completely identical reliabilities. Looking from left to right, the first five clusters – the five clusters with the lowest reliability – are driven by single point failures of any of the six component types. For example, in an architecture that contains one sensor and two computers or an architecture that contains one sensor and three computers, the single sensor present in the architecture must remain reliable in order for the overall system to remain reliable. Sensor type A has a reliability of 0.9985. Therefore, a one-sensor two-computer or one-sensor three-computer architecture that contains sensor A can have a maximum reliability of 0.9985. Such architectures, which have a single point failure at sensor A, define the first (least reliable) cluster of data points. Both sensor type B and computer type A have the same reliability: 0.999. Therefore, architectures that have a single point failure at a sensor of type B or a single point failure at a computer of type A will both fall into the second cluster. Similarly, sensor type C's reliability of 0.9995 defines the third cluster, computer type B's reliability of 0.9996 defines the fourth cluster, and computer type C's reliability of 0.9998 defines the fifth cluster.

The sixth and final (most reliable) cluster contains all other architectures – architectures free from single point failures. The few additional points that do not fall into any of the six clusters are single-string architectures, i.e., architectures that contain just one sensor and one computer. These architectures contain not one, but two single point failures and are therefore significantly less reliable than an identical architecture with additional computers or additional sensors.

[9] The utopia point represents the ideal architecture which is 100 percent reliable with weight = 0.





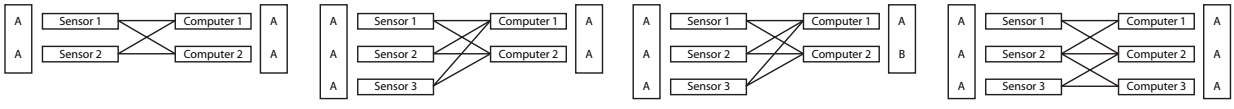
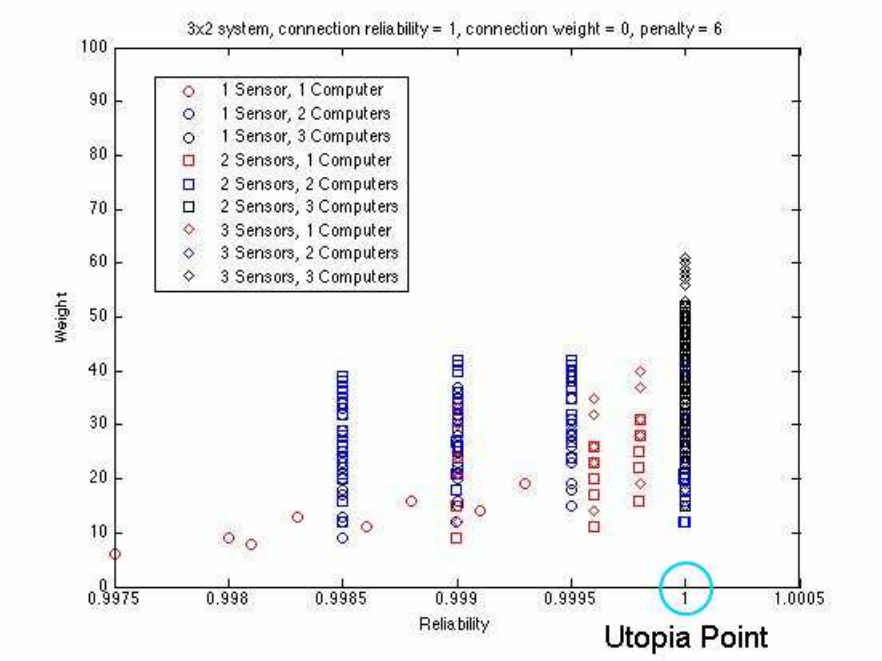
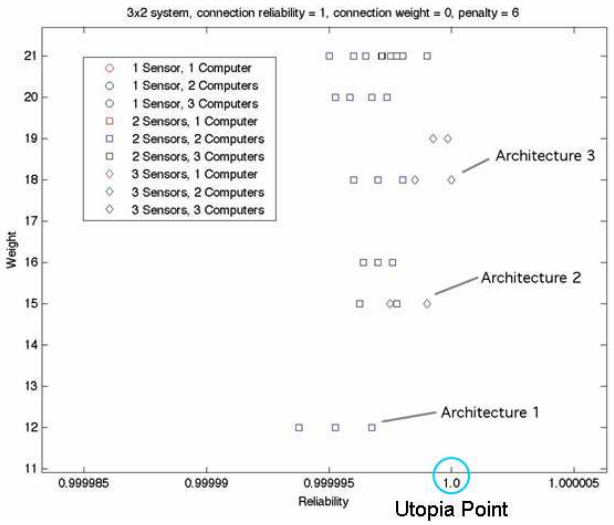


Figure 8. (Top) Pareto plot. with added details for number of connections between sensors and computers, (Middle) zoomed in version of the same plot, (Bottom) potential optimal architectures for this scenario: (From left to right) Architecture 1 has weight = 12 and reliability = 0.99999675437371 (five 9s), architecture 2 has weight = 15 and reliability = 0.99999899763201 (five 9s), architecture 3 has weight = 17 and reliability = 0.99999959691247 (six 9s), and architecture 4 has weight = 18 and reliability = 0.99999999563408 (eight 9s).





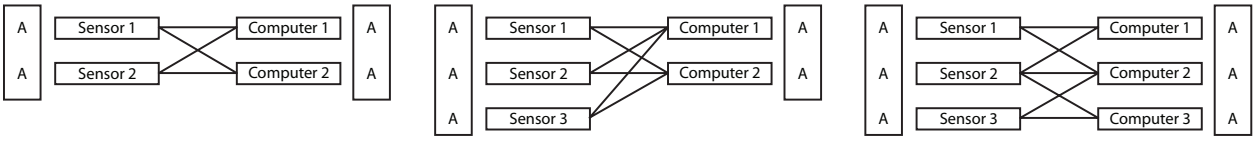
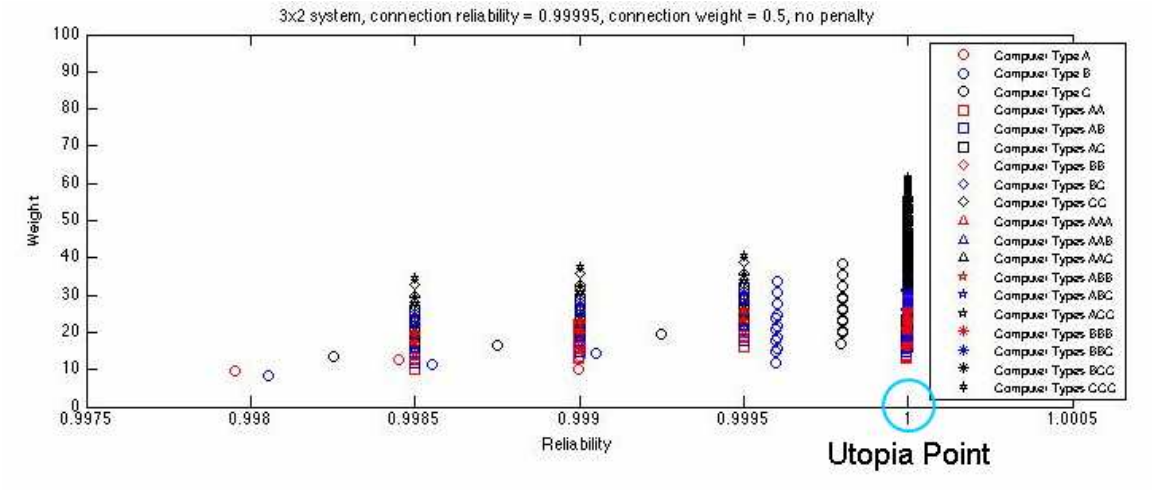
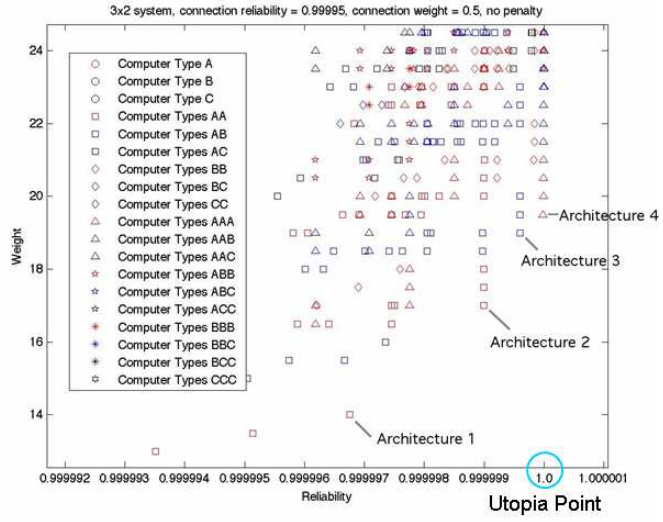


Figure 9. (Top) Pareto plot with added details for both number of sensors and number of computers, (Middle) zoomed in version of the same plot, (Bottom) potential optimal architectures for this scenario: (From left to right) Architecture 1 has weight = 12 and reliability = 0.99999675437371 (five 9s), architecture 2 has weight = 15 and reliability = 0.99999899763201 (five 9s), and architecture 3 has weight = 18 and reliability = 0.99999999563408 (eight 9s).





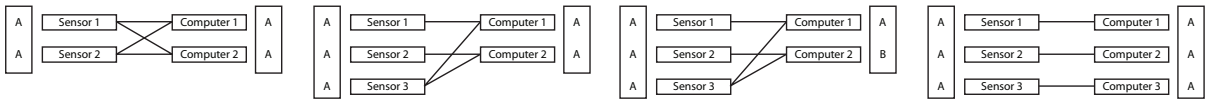
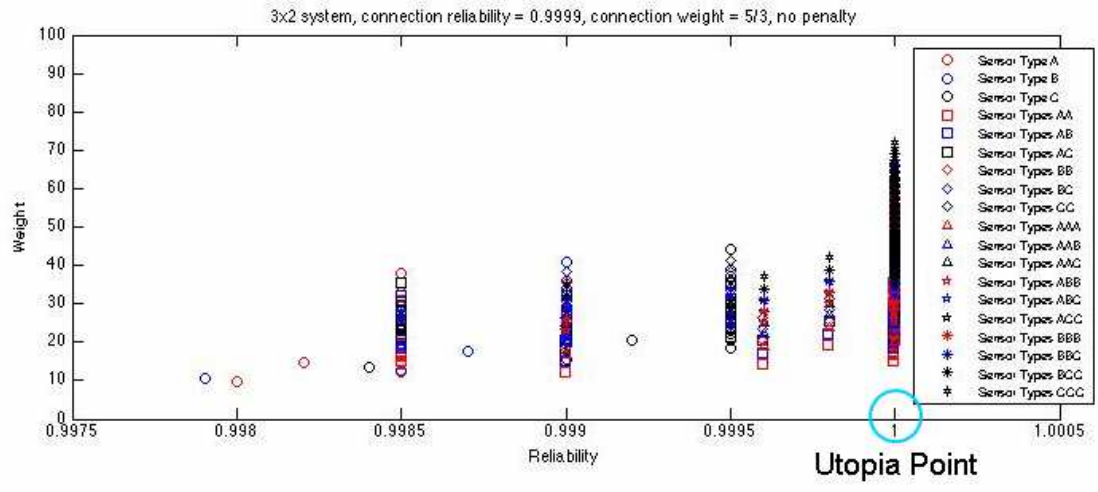
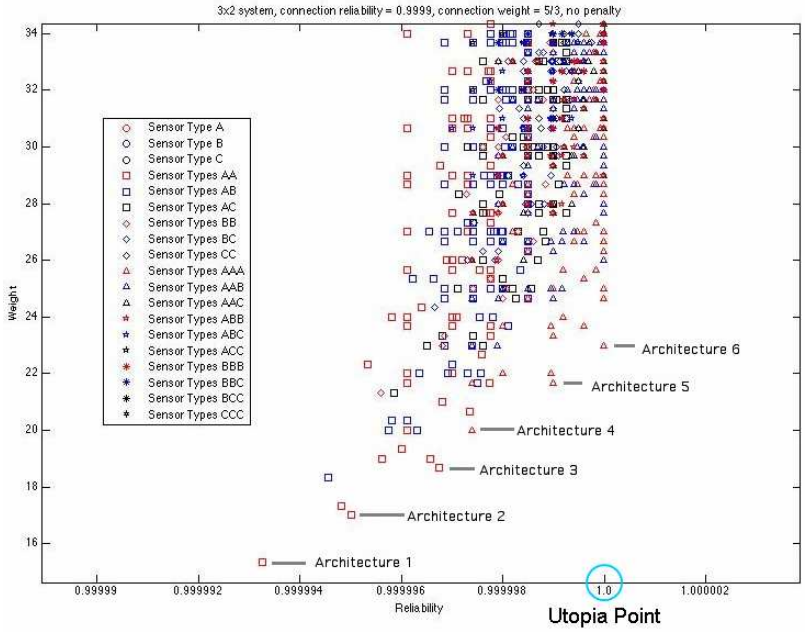


Figure 10. (Top) Pareto plot with added details for both number and types of computers, (Middle) zoomed in version of the same plot, (Bottom) potential optimal architectures for this scenario: (From left to right) Architecture 1 has weight = 14 and reliability = 0.999996754062388 (five 9s), architecture 2 has weight = 17 and reliability = 0.999998992620742 (five 9s), architecture 3 has weight = 19 and reliability = 0.999999593334442 (six 9s), and architecture 4 has weight = 19.5 and reliability = 0.999999983481890 (seven 9s).





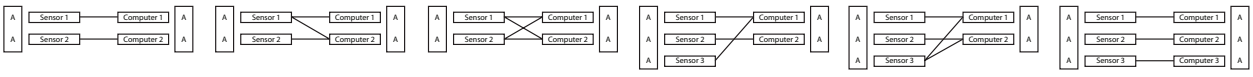


Figure 11. (Top) Pareto plot with added details for both number and types of sensors, (Middle) zoomed in version of the same plot, (Bottom) Potential optimal architectures for this scenario: (From left to right) Architecture 1 has weight = 15.333 and reliability = 0.999993257523472 (five 9s), architecture 2 has weight = 17 and reliability = 0.99999501059889 (five 9s), architecture 3 has weight = 18.667 and reliability = 0.999996753726173 (five 9s), architecture 4 has weight = 20 and reliability = 0.999997398039817 (five 9s), architecture 5 has weight = 21.667 and reliability = 0.999998992071849 (five 9s), and architecture 6 has weight = 23 and reliability = 0.999999983481890 (seven 9s).

Many interesting aspects can be found in Figure 8, the ideal case where there is no penalty for connection weight or using dissimilar components. In this scenario, all identified architectures are fully cross-strapped. This is to be expected since additional connections increase reliability yet cost nothing. Figure 8 also demonstrates that the weights of an architecture's component plays a major role in determining the optimality of the architecture. Recall that components of type "A" are the lightest and least reliable and that components of type "C" are the heaviest and most reliable. Even though components of type "A" are the least reliable, sensors of type A and computers of type A are by far the most prevalent component types in all the optimal architectures. In addition, although components of type "C" are the most reliable, no components of this type appear in any of the optimal architectures. Furthermore, there are not even any sensors of type B in the optimal architectures. Architecture 3 contains a computer of type B, but this architecture is no longer optimal once the "dissimilar components penalty" is increased from penalty = 0 to penalty = 6 (see Figure 9, bottom). Since the optimal architectures for penalty = 6 contain no dissimilar components, increasing the penalty again to penalty = 9 results in no further changes to the optimal architectures.

Figure 10 gives a baseline for what is realistic. In this scenario, there are no longer perfect connection reliabilities of 100 percent and there is an actual cost for producing connections. Now, the connection reliability = 0.99995 and the connection weight = 0.5. As a result of the 0.5 connection weight, only one fully cross-strapped architecture (architecture 1) is among the optimal architectures – much different than the scenario where connection weight = 0. The other optimal architectures have just three or four connections. From these, architectures 2 and 3 are the most interesting. Each has three sensors and two computers with two of the sensors having one connection to a computer and the last having two.

The connection weight = 0.5 scenario still has quite a bit in common with the connection weight = 0 scenario. Once again, the component types in the optimal architectures are predominantly of type A and never of type C. Only one optimal architecture (architecture 3) contains a component of type B and this component is again a computer. This architecture is no longer optimal when the dissimilar components penalty is increased to penalty = 6. There is no change in optimal architectures for connection weight = 0.5 when this penalty is increased from penalty = 6 to penalty = 9.

Figure 11 depicts the first scenario where connection reliability = 0.9999 and connection weight = 5/3. This scenario produces very similar optimal architectures to the connection weight = 0.5 scenario, but there are some notable exceptions.

Even with the dissimilar components penalty set to penalty = 0, connection weight = 5/3 is sufficiently high as to eliminate any architecture that contains a component type heavier than type A. This means that architecture 3 from Figure 10, the only connection weight = 0.5 architecture which contains a component of type B, is not an optimal architecture for connection weight = 5/3. This also means that the optimal architectures for connection weight = 5/3 will not change if the dissimilar components penalty is increased to penalty = 6 or penalty = 9.

A connection weight of 5/3 also makes it significantly more desirable to have architectures with even fewer connections. Although optimal architectures 1, 2, and 4 from Figure 10 are still optimal architectures when the connection weight is increased to 5/3, the value of these architectures is diminished with the heavier connection weight. As a result, these architectures are no longer significantly closer to the utopia point than architectures 1, 2, and 4 from Figure 11.

The six potentially optimal architectures for connection weight = 5/3 produce an interesting set. All legal architectures with four or fewer connections that contain two to three sensors of type A and two computers of type A are optimal architectures for this scenario. In effect, a system architect is directly trading an increase in weight for additional reliability when connection weight = 5/3.

When reviewing a subset of all the possible architectures, specifically a subset in which all members have the same number of connections, the effect of the dissimilar components penalty on the optimal architectures of the subset is nearly identical to the penalty's effect on the optimal architectures of the superset. Figures 12 and 13 depict the optimal architectures for 3 x 2 systems with 1, 2, 3, 4, 5, 6, 7, 8, and 9 connections when connection reliability = 1 and connection weight = 0 (the reliabilities and weights for these architectures can be found in Tables 6 through 8). Again, as the penalty is increased from penalty = 0 to penalty = 6, nearly all architectures containing sensor type B or computer type B cease to be optimal. Note that, for architectures with the same number of connections, the exact same architectures which are optimal for penalty = 0 will be optimal no matter what the connection weight. Similarly, architectures which are optimal for penalty = 6 or penalty = 9 will also remain optimal no matter what the connection weight. The reasoning goes as follows. Although the overall system weight of any member of a subset will change if the connection weight is changed, this change will be identical to the change seen by any of the other systems in this subset. This is because all systems in each subset have, by definition, the same number of

connections. Therefore, among architectures with the same number of connections, the architectures closest to the utopia point will remain closest to the utopia point no matter what change is made to the connection weight.

Table 6. The reliabilities and weights for the 1-, 2-, 3-, 4-, 5-, 6-, 7-, 8-, and 9-connection architectures closest the utopia point given a 3 x 2 system with connection reliability = 1, connection weight = 0, and no penalty for dissimilar components.

Connections	Reliability	Weight	On Pareto front?
1	0.999100405	14	
	0.999300245	19	
2	0.999993766	12	
	0.999995260	14	
	0.999996397	16	
	0.999997344	19	
	0.999997754	20	
	0.999998292	22	
	0.999998741	25	
3	0.999995260	12	
	0.999996756	14	
	0.999997499	15	
	0.999998996	17	
	0.999999984	18	
4	0.999996754	12	YES
	0.999998993	15	
	0.999999594	17	
	0.999999988	18	
5	0.999998995	15	
	0.999999596	17	
	0.999999992	18	
6	0.999998998	15	YES
	0.999999597	17	YES
	0.999999996	18	
7	0.999999994	18	
8	0.999999996	18	
9	0.999999996	18	YES

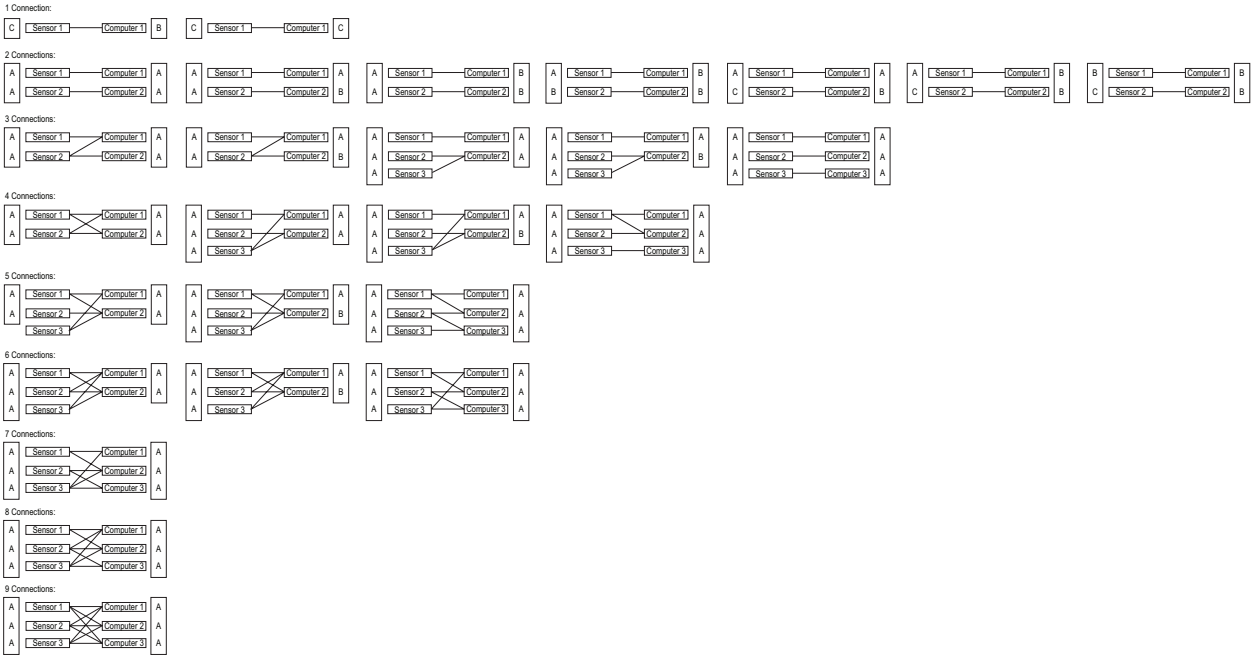


Figure 12. The architectures described in Table 6.

Table 7. The reliabilities and weights for the 1-, 2-, 3-, 4-, 5-, 6-, 7-, 8-, and 9-connection architectures closest the utopia point given a 3 x 2 system with connection reliability = 1, connection weight = 0, and penalty = 6 for dissimilar components.

Connections	Reliability	Weight	On Pareto front?
1	0.999100405	14	
	0.999300245	19	
2	0.999993766	12	
	0.999996397	16	
3	0.999995260	12	
	0.999997499	15	
	0.999999984	18	
4	0.999996754	12	YES
	0.999998993	15	
	0.999999988	18	
5	0.999998995	15	
	0.999999992	18	
6	0.999998998	15	YES
	0.999999996	18	
7	0.999999994	18	
8	0.999999996	18	
9	0.999999996	18	YES

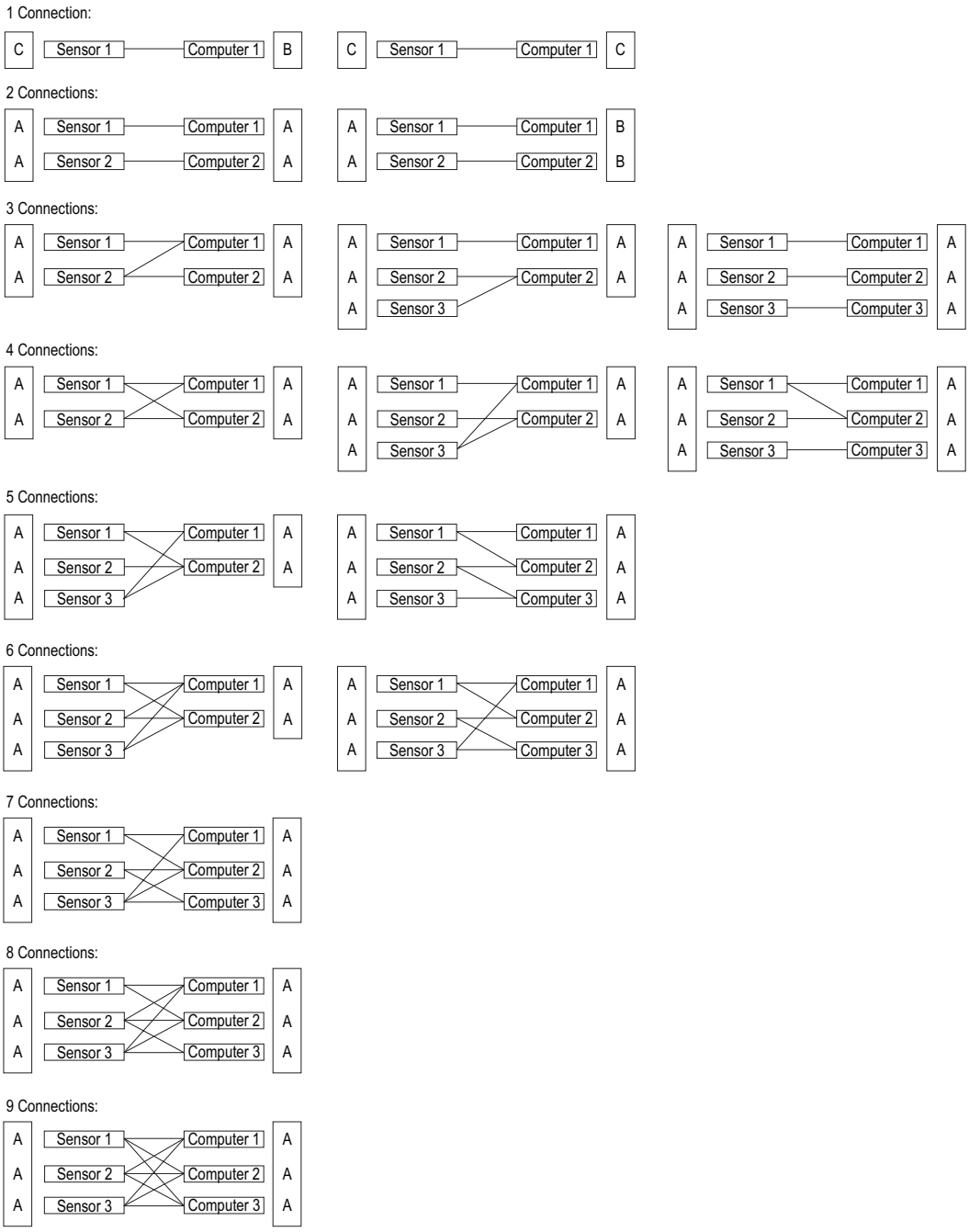


Figure 13. The architectures described in Table 7.

VII. Conclusion

If the only materials on hand were sensors of low-reliability type A, computers of low-reliability type A, and generic connections, it is still possible to produce nearly all potentially optimal architectures. The identified optimal architectures show this and, more generally, that it is preferable to increase redundancy of lighter, less reliable components rather than to use smaller numbers of more reliable, heavy components.

Due to the extremely lightweight nature of "sensor A" and "computer A", these component types appeared in the optimal architectures in every scenario and almost always with redundancy > = 2. Furthermore, the very heavy yet very reliable "sensor C" and "computer C" never appeared as an optimal architecture in any scenario, not even with redundancy = 1.

At a certain point, there are diminishing returns to adding redundancy, connections, or components with greater reliability – the system starts to experience only minimal increases in overall reliability for the large gains in weight. This fact becomes more apparent through the use of the connection weight penalty. Looking at the optimal architectures in each scenario, the number of connections drops dramatically as the connection weight penalty is increased from connection weight = 0 to connection weight = 0.5. When connection weight = 0, there are, as expected, optimal architectures with as many as nine connections. When connection weight increases to 0.5 however, this number drops to three or four.

The impact of the "dissimilar components penalty" is more subtle, but still apparent. In the scenarios where connection weight = 0 and connection weight = 0.5, architectures containing both computer type A and computer type B were identified as potentially optimal when the penalty = 0. However, when the penalty is increased to penalty = 6, these architectures no longer appear to be better than other architectures.

The created OPN models the building blocks of a GN&C system. Now that the base model is complete, more detail can gradually be added until the entire system is encapsulated. First, the OPN language should be made more efficient so it does not run out of memory when trying to run the 3 x 3 model. Next, other architectural layouts should be investigated in which the component redundancy for any component can be greater than three. In order to do so, however, the process of eliminating duplicates and creating rules (which was done by hand) should be automated. Also, it is important that more than three types of sensors, computers, and actuators be incorporated into the model and that further details be included for each component. For example, requirements might suggest that a system needs six degrees-of-freedom of attitude and position knowledge. This requirement cannot be addressed with the current top-level mentality. Finally, further metrics besides reliability and weight should be implemented in the model.

Having such a model in place in OPN would serve as an excellent tool for system design. System requirements may dictate a given reliability and such a model would easily be able to find the lowest weight/lowest cost option for that reliability. Although less likely, if, rather than overall reliability, the system requirement called for a certain number of connections between adjacent components, the OPN model could easily find the optimal high reliability/low weight option for this architecture as well.

In addition to reliability, OPN can be made to find the best option for satisfying other metrics as these metrics are added to the model. As additional component redundancy and component types are added to the model, all possible architectures could eventually be both enumerated and evaluated by OPN.

A detailed consideration of the findings in this paper could be extremely beneficial to the development of the CxP's robotic and human-rated systems; clearly exploring commonality in GN&C components can reduce both nonrecurring and recurring cost and risk.

Acknowledgments

The authors would like to thank their colleagues at MIT, Draper Laboratory and NASA for their support, technical insights, and help in reviewing this paper. Ralph Roe and the NASA Engineering and Safety Center management team are also to be acknowledged for their support and sponsorship of this GN&C discipline advancing activity.

References

[1] Cameron, B., Crawley, E., Loureiro, G., and Rebentisch, E., "Value flow mapping: Using networks to inform stakeholder analysis" Acta Astronautica, Volume 62, Issues 4-5, February-March 2008.

[2] Dominguez-Garcia, A., Hanuschak, G., Hall, S., and Crawley, E., "A Comparison of GN&C Architectural Approaches for Robotic and Human-Rated Spacecraft" AIAA Guidance, Navigation and Control Conference and Exhibit, 20-23 Aug 2007, Hilton Head, SC.

[3] Hofstetter, W., Wooster, P., Nadir, W., and Crawley E., "Affordable Human Moon and Mars Exploration Through Hardware Commonality" AIAA-2005-6757 Space 2005, Long Beach, California, Aug. 30-1, 2005.

[4] Sahner, R., Trivedi, K., and Puliafito, A.. Performance and Reliability Analysis of Computer Systems: An Example-Based Approach Using the SHARPE Software Package, Kluwer Academic Publishers, 1995.

[5] Cameron Simmons, W., Koo, B., and Crawley, E., "Architecture Generation for Moon-Mars Exploration Using an Executable Meta-Language" AIAA-2005-6726 Space 2005, Long Beach, California, Aug. 30-1, 2005.

[6] Chapman, G. T., and Tobak, M., "Nonlinear Problems in Flight Dynamics," NASA TM-85940, 1984.