An Advanced Computational Approach to System of Systems Analysis & Architecting Using Agent-Based Behavioral Model

Technical Report SERC-2013-TR-021-2

March 29, 2013

Principal Investigator: Dr. Cihan Dagli, Missouri Science & Technology

Team Members:

Air Force Institute of Technology: Dr. John Colombi,

Pennsylvania State University: Dr. Nil Ergin,

Systems Engineering Practice Office, MITRE: Dr. George Rebovich

Naval Postgraduate School: Dr. Kristin Giammarco

Missouri University of Science &Technology: Dr. David Enke, Dr. Abhijit Gosavi, Dr. Ruwen Qin, Paulette Acheson, Khaled Haris, Louis Pape

Contract Number: H98230-08-D-0171 Task Order: 0029, RT 044

Report No. SERC-2013-TR-021-2

The Systems Engineering Research Center (SERC) is a federally funded University Affiliated Research Center managed by Stevens Institute of Technology.

This material is based upon work supported, in whole or in part, by the U.S. Department of Defense through the Office of the Assistant Secretary of Defense for Research and Engineering (ASD(R&E)) under Contract H98230-08-D-0171 (Task Order 0029, RT 044).

Executive Summary

A major challenge to the successful planning and evolution of an acknowledged System of Systems (SoS) is the current lack of understanding of the impact that the presence or absence of a set of constituent systems has on the overall SoS capability. Since the candidate elements of a SoS are fully functioning, stand-alone Systems in their own right, they have goals and objectives of their own to satisfy, some of which may compete with those of the overarching SoS. These system-level concerns drive decisions to participate (or not) in the SoS. Individual systems typically must be requested to join the SoS construct, and persuaded to interface and cooperate with other Systems to create the "new" capability of the proposed SoS. Current SoS evolution strategies lack a means for modeling the impact of decisions concerning participation or non-participation of any given set of systems on the overall capability of the SoS construct. Without this capability, it is difficult to optimize the SoS design.

The goal of this research is to model the evolution of the architecture of an acknowledged SoS that accounts for the ability and willingness of constituent systems to support the SoS capability development. Since DoD Systems of Systems (SoS) development efforts do not typically follow the normal program acquisition process described in DoDI 5000.02, the Wave Model proposed by Dahmann and Rebovich is used as the basis for this research on SoS capability evolution. The Wave Process Model provides a framework for an agent-based modeling methodology, which is used to abstract the non-utopian behavioral aspects of the constituent systems and their interactions with the SoS. In particular, the research focuses on the impact of individual system behavior on the SoS capability and architecture evolution processes.

A generic agent-based model (ABM) skeleton structure is developed to provide an Acknowledged SoS manager a decision making tool in negotiating of SOS architectures during the wave model cycles. The model provides an environment to plug in multiple SoS meta-architecture generation multiple criteria optimization models based on both gradient and non-gradient descent optimization procedures. Three types of individual system optimization models represent different behaviors of systems agents, namely; selfish, opportunistic and cooperative, are developed as plug in models. ABM has a plug in capability to incorporate domain-specific negotiation modes and a fuzzy associative memory (FAM) to evaluate candidate architectures for simulating SoS creation and evolution. The model evaluates the capability of the evolving SoS architecture with respect to four attributes: performance, affordability, flexibility and robustness.

In the second phase of the project, the team will continue with the development of an evolutionary strategies-based multi-objective mathematical model for creating an initial SoS meta architecture to start the negotiation at each wave. A basic generic structure will be defined for the fuzzy assessor math model that will be used to evaluate SoS meta architectures and domain dependent parameters pertaining to system of systems analysis and architecting through Agent Based Modeling. The work will be conducted in consideration of the national priorities, funding and threat assessment being provided by the environment developed for delivery at end of December 2013.

The method is applied to an Intelligence Surveillance Reconnaissance (ISR) “acknowledged” SoS as an example domain. The agent-based model represents a System Program Office (SPO) personnel’s interactions with the acknowledged SoS manager, and with the other Systems’ representatives. An agent models each SPO’s negotiation and decision process and interactions.

Table of Contents

List of Figures

List of Tables

1 Introduction

The goal of this research is to model the evolution of the architecture of an acknowledged Systems of Systems (SoS) that accounts for the ability and willingness of constituent systems to support the SoS capability development. Since DoD SoS development efforts do not typically follow the normal program acquisition process described in DoDI 5000.02, the Wave Model proposed by Dahmann and Rebovich is used as the basis for this research on SoS capability evolution. The Wave Process Model provides a framework for an agent-based modeling methodology, which is used to abstract the non-utopian behavioral aspects of the constituent systems and their interactions with the SoS. In particular, the research focuses on the impact of individual system behavior on the SoS capability and architecture evolution processes.

A generic agent-based model (ABM) skeleton structure is developed to provide Acknowledged SoS manager a decision making tool in negotiating of SoS architectures during the wave model cycles. The model provides an environment to plug in multiple SoS meta-architecture generation multiple criteria optimization models based on both gradient and non-gradient descent optimization procedures. Three types of individual system optimization models represent different behaviors of systems agents, namely; selfish, opportunistic and cooperative, are developed as plug in models. ABM has a plug in capability to incorporate domain-specific negotiation modes and a fuzzy associative memory (FAM) to evaluate candidate architectures for simulating SoS creation and evolution. The model evaluates the capability of the evolving SoS architecture with respect to four attributes: performance, affordability, flexibility and robustness.

1.1 Problem/Motivation

1.1.1 Project Description

This research develops, validates, and pilots ABM Methods, Tools and Processes (MTPs) that support investigating the properties of an acknowledged SoS, and can be applied early in the life cycle when there is high uncertainty and ambiguity about SoS requirements, architecture, DoD Acquisition guidance and implementation technologies, based on the Wave Process Model.

1.1.2 Value Proposition

Generic ABM model provides a capability to model each system in the context of its environment, which is distinct from a more typical modeling of one system in the context of its environment. There are many moving pieces in such a model, and that complexity introduces many opportunities to select suboptimal designs for an Acknowledged SoS. We cannot afford to continue to make decisions "eyeballing" the SoS design and improving through trial and error. We need to use modern tools to model our understanding of the behavior of the SoS under difference circumstances (such as participation/non-particiaption of systems) so that we have clear visibility into the full range of feasible SoS designs as well as tradeoffs among those designs, and see the predictable impacts in advance of decisions. No structured, repeatable approach is consistently used within DoD for planning and

modeling proposed alternative Acknowledged SoS architectures, and then systematically prioritizing those architectures in order of the best to worst match with stakeholder needs like flexibility, robustness, performance, etc. This research takes a step towards achieving that capability by introducing a new analysis framework that uses modern modeling tools to expose foreseeable SoS level impacts for decision makers early in the lifecycle, when such impacts can be managed less expensively and more solutions to possible problems can be put on the table.

Early insight into likely properties of acknowledged SoS meta architectures and conditions that affect their implementation and effectiveness will inform technological and policy decisions about far reaching and expensive SoS acquisition decisions.

1.2 Research Objectives

1.2.1 Objectives

The goal of this research is to develop a proof of concept ABM tool suite for SoS systems simulation for architecture selection and evolution. An Intelligence, Surveillance and Reconnaissance (ISR) SoS, consisting of numerous individual systems is used as a domain example to demonstrate the framework of the ABM tool suite.

1.2.2 Methodology

Policies on architecting in the DoD continue to evolve, although perhaps at a slower pace than in the recent past. As a result, the modeling of architecture development and evolution is not settled science. This is particularly true in SoS settings (ASD(NII), 2009). Existing analysis methodologies and tools narrow the scope of the SoS problem space by invoking the assumption that there is a limited set of solutions, solely or primarily driven by technical performance considerations. However, the SoS problem boundary includes integration of technical systems as well as cognitive and social processes, which alter system behavior (Dauby & Upholzer, 2011).

As mentioned before, most system architects assume that SoS participants exhibit nominal (utopian) behavior, but deviation from nominal motivation leads to complications and disturbances in systems behavior. It is necessary to capture the behavioral dimension of SoS architecture to be able to represent the full problem space to guide the SoS architecting and analysis phase (Dauby & Upholzer, 2011). Evaluation of architectures also lends itself to a fuzzy approach because the criteria are frequently non-quantitative, or subjective, or based on unknowable future conditions, such as “robustness.” Finally, since one of the current problems with SoS composition is lack of imagination, and system participation or non-participation is conveniently binary, a genetic algorithm (GA) as the non-gradient based optimization approach can help to explore a wider architecture space more fully.

Agent based models (ABMs) consist of a set abstracted entities referred to as agents, and a framework for simulating agent decisions and interactions. Agents may have their own goals and are capable of perceiving changes in the environment (Acheson P. , Methodology for Object Oriented System Architecture Development, 2010). System behavior (global behavior) emerges from the decisions and interactions of the agents. The approach provides insight into complex, interdependent processes.

Agent-based modeling methodology has several benefits over other modeling techniques; it enables Acknowledged SoS manger to capture emergent patterns of system behavior, provides a natural description of a system composed of behavioral entities and is flexible for tuning the complexity of the entities (Bonabeau, 2002). The methodology is used in a wide range of application domains including financial markets (Kilicay-Ergin, Enke, & Dagli, 2012), homeland security applications (Weiss, 2008) and autonomous robots (Dudenhoeffer & Jones, 2000).

Agent-based modeling methodology is used to abstract behavioral aspects of the acquisition process. In this project, it is assumed that the Systems are the agents. The System Agents embody themselves and the people (individual stakeholders) responsible for them. The wave model applies to an Acknowledged SoS, thus there is also a specific agent responsible for the SoS, and that agent influences the other System Agents. An initial SoS mission is already determined and funds are allocated to the mission through a responsible organizational entity (Director Systems and Software Engineering, OUSD (AT&L), 2008). The structure of the wave model is depicted in Figure 1. (Dahmann, Rebovich, Lane, Lowry, & Baldwin, 2011)

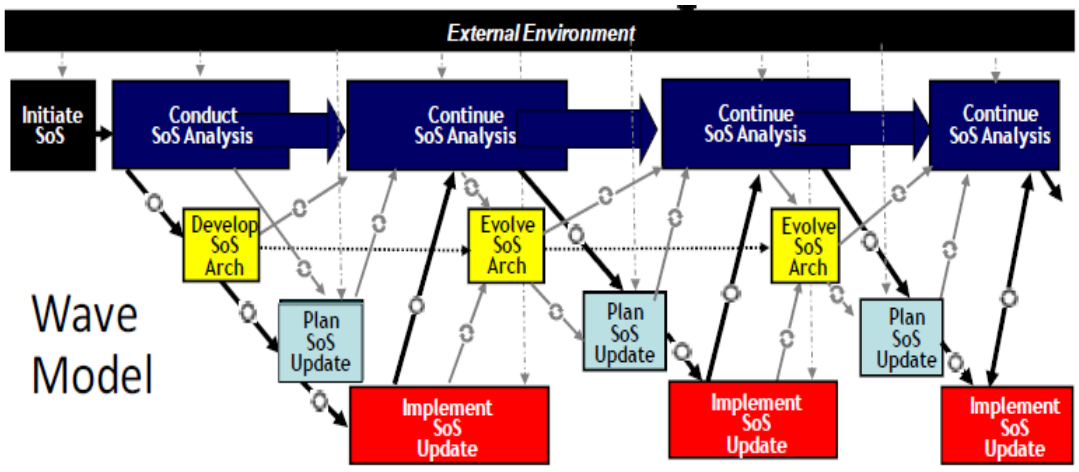


Figure 1. The Wave Model of SoS initiation, engineering, and evolution

2 Background and Literature Review

2.1 Systems of Systems Challenges

2.1.1 SoS Background and Challenges

Policies on architecting in the DoD continue to evolve, although perhaps at a slower pace than in the past. In the last 20 years, there has been a shift in government acquisition policy from being the systems integrator, to contracting out for this function, and now retaking the role as lead system

integrator for large, complex programs. Two trends meshed to make large, complex systems more likely to be needed, and more difficult to integrate. The developments of the theory and application of Network Centric Warfare, and the successive updates to the DoD Architecture Framework have now reached the point of adequately representing the desired integration policies for very large SoSs. Since these developments are relatively recent, the modeling of development and evolution of architectures for SoS is not yet practiced consistently throughout the community (ASD(NII), 2009).

Traditional system engineering methods and tools close the problem by scoping the system boundaries and allocating requirements to system components with the assumption that no suprises come from the outside. However, most SoS development problems involve open systems which lack clear boundaries. This is mainly due to the fact that SoS component systems are independent and have their own functionality, development processes, funding and operational missions. In addition changes in external environment such as funding, national priorities or threats can alter the dynamics of the acquisition process (Dombkins, 2007) (Kilicay Ergin & Dagli, 2008). Therefore, SoS engineering methods and tools need to consider change as an important dynamic beyond technical considerations.

The SoS engineering process spans through multiple abstraction layers and domains to foster collaboration among independent systems. Collaboration requires more emphasis on interface architecting which brings along many challenges such as interoperability, scalability and security issues.

Additionally, architectural constraints imposed by existing systems can have a major influence on overall SoS capabilities. Specifically, the amount and variety of the legacy systems impose interoperability constraints in SoS architecting processes. The existing component systems' life cycle and the recently added systems' life cycle complicate the SoS life cycle definition. The diversity of the engineering focus and the engineering environment bring up additional challenges. For example, dynamically changing requirements increase uncertainty and systems need to be designed for fuzzy attributes.

2.1.2 Acknowledged SoS

In an "Acknowledged SoS," an initial SoS mission is determined and funds are allocated to the mission with a responsible organizational entity employing the SoS Manager. SoS constitutuent Systems are approximately independent and have their own functionality, development processes, funding and operational missions, although they can have interfaces in operation. Constituent Systems are frequently in different phases of their own life cycles (Director Systems and Software Engineering, OUSD (AT&L), 2008). The SoS Manager has some, but not complete, authority over constituent Systems when they work within the SoS. She may solicit, offer funding, cajole, use the bully pulpit, beg, log roll, match make, and use her own influence to gain cooperation from the constituent Systems' personnel. Contributing Systems may have their own problems, interests and desires, goals, external influences, restrictions, image to protect or enhance, fears, and opportunities that make them cooperate more or less with the SoS. The combined ABM , GA and fuzzy approaches help model, explore, and analyze the influences of opportunity and social interaction on SoS development and performance.

2.1.3 SoS Viewpoint

The SoS manager views the SoS from a different perspective than any individual system is capable of appreciating. The task of the individual systems is to allocate funding and schedule within their system to each of the interfaces to offer the SoS manager a performance capability increment. However, Systems cannot force other Systems to participate either; both much agree to interfaces. Both SoS and System are concerned with the development funding, schedules, and sufficiency of achievable capabilities, but the SoS manager may be concerned with other, possibly higher level, non-linear, combined attributes of the SoS. These could include overall affordability, schedule, robustness of the delivered SoS capability, operating cost, his own development flexibility in combining the several Systems, the risk within development efforts, scalability, training impact, responding to threat changes, or looking good to his peers, superiors or other stakeholders in the SoS purpose, among many other possible concerns.

2.1.4 Social Aspect of Systems' Decisions to Participate

Most system architects attempt to design under the assumption that SoS participants exhibit nominal (utopian) behavior, but deviation from nominal motivation leads to complications and disturbances in systems behavior. It is necessary to capture the behavioral dimension of SoS architecture to be able to represent the full problem space to guide the SoS during the architecting and analysis phase (Haris & Dagli, 2011). Systems have internal states, in addition to environmental factors that influence their ability or willingness to participate in the SoS. Individual programs might have many reasons for either joining or refusing to join the SoS, and reflect these in their negotiation strategies. These reasons could run the gamut from having too little margin in their own performance to be able to afford any support to the SoS, to turning their entire program over to the new SoS purpose. They could be overrun and behind schedule, and not be able to spare the resources to develop a new capability or interface that would aid the SoS. The program could be nearing the end of its own life cycle, and want to jump on any new bandwagon as a reason to stay alive. They might need a reason (some may say excuse), and extra funding, to make a change to their program to fix an internal problem they just discovered, and can reasonably connect the new capability request and the internal fix together. They could even think the SoS is a wonderful idea and desperately want to be a part of it, whether they get new funds or not. At the very least, the Systems have different stakeholders than the SoS, who need to be consulted or informed about the "mission creep" that participation in the SoS entails. Program offices might have any number of similar reasons, each pulling them in different directions with regard to cooperation with the SoS agent.

2.1.5 Combined Models

Different architecting tools are required to support the SoS engineering process. A good balance of heuristics, analytical techniques and integrated modeling is necessary as architecting tools to support SoS engineering. Specifically, model-centric frameworks and executable models become important tools for SoS analysis and architecting as they provide insights to SoS architecture behavior. These tools are also needed to analyze emergent behavior of SoS engineering in a rigourous way.

Agent based models (ABMs) can help explore the dimensions of the social aspects of SoS development by abstracting behavior to simple interaction models. Independent agents following simple rules can model surprisingly complex behavior. Since one of the current problems with SoS composition is a limited, pre-conceived notion of a solution, a GA approach may help to explore the potential architecture space more fully (Haris & Dagli, 2011). In both the ABM and GA approaches, however, evaluation of the intermediate and final architecture fitness is necessary.

Evaluation of architectures lends itself to a fuzzy approach because the criteria are frequently non-quantitative, or subjective (Dauby & Dagli, 2011), or based on difficult to define or even unknowable future conditions, such as "robustness." Individual attributes may not have a clearly defined, mathematically precise, linear functional form from worst to best. The goodness of one attribute may or may not offset the badness of another attribute. Several moderately good attributes coupled with one very poor attribute may be better than an architecture with all marginally good enough attributes, or the reverse! A fuzzy approach allows many of these considerations to be handled using a reasonably simple set of rules, as well as having the ability to include non-linear characteristics in the fitness measure. The simple rule set allows us to tweak the model to see how seemingly small changes affect the outcome.

While this may not be a perfect modeling approach, it consists of reasonably small regions of approximation and simplification, but joined in ways that can lead to quite complex behaviors. It also provides a way to experiment by changing the rules and observing how the SoS changes.

Integrating various modeling approaches that address different aspects of the SoS engineering problem provides insights and systems thinking about the SoS engineering problem. It is envisioned that plug-and-play type of integrated modeling techniques capture the SoS engineering problem dimensions and support SoS acquisition managers in their decision making process.

2.2 Modeling Approach

This modeling framework is for an Acknowledged System of Systems (SoS), where each component system is a fully functioning, independently funded and managed System (or Program Office). A high-level manager sees the opportunity to achieve a needed, new capability by using existing systems in a new way, either unchanged except for the manner of utilization, or with relatively minor system changes. The SoS approach is only useful if it can achieve the new capability for either or both reduced cost compared to designing a separate, new "purpose built" system, and/or reduced time to field the new capability.

The new capabilities being sought from the SoS are achieved by combining mostly *existing* System capabilities and/or adding new capabilities that arise *in conjunction* with *other* Systems (i.e., through new interfaces). If simply throwing more Systems (with their individual capabilities) at the problem sufficed, there would be no need to create the SoS. The nature of the *Acknowledged* SoS though, means that the SoS manager does not have absolute authority to command participation, but must "purchase" the component Systems' participation and modifications not merely with funds but also through

persuasion, the strength of the vision of the SoS, quid pro quos, the bully pulpit and whatever other means are legitimate and effective. (Director Systems and Software Engineering, OUSD (AT&L), 2008) Individual Systems remain free to decide *not* to participate in the SoS development. Alternatively, they may not be available during a particular operational period of need. Some capabilities and interfaces already exist, meaning they are free and fast for development, but they may still have a cost to operate in the fielded SoS. Some systems may have enough capability that the SoS can tap their spare capability, so they are essentially free to operate as well. Other capabilities may need minor (compared to a major program) development, either within a system, or a new interface with another system. The Performance capabilities of the SoS will generally be greater than the sum of the capabilities of its parts. If this were not the case, there would be no need for the SoS. Changing the way the Systems interact without modification typically would not improve the capabilities possible from simply adding more individual Systems, in this simplified modeling approach.

We model the development of the SoS Capabilities with a three-part approach:

- A domain specific model is created with feasibility rules prohibiting some architectures, and Key Performance Attribute algorithms for evaluating an SoS architecture, using a fuzzy inference system (FIS)
- The GA develops optimized chromosomes from the domain meta-architecture for the SoS Agent to propose to the Systems by using the fuzzy assessor to determine the fitness of evolving chromosomes
- The Agent Based Model manages individual System negotiation models to produce a "realizable" architecture chromosome through their cooperation.

One cycle through the *proposal-negotiation-agreement-execution* development steps is an epoch, or wave, in the wave model (Dahmann, Rebovich, Lane, Lowry, & Baldwin, 2011). The domain model describes what the bits in the chromosome represent in terms of cost, capability contribution, and time to deliver,and the fuzzy assessor evaluates chromosomes for fitness. The GA manages the exploration of the architecture space and picking an optimum or ideal (from the viewpoint of the SoS agent) SoS chromosome, as well as the evolution from a negotiated, "achievable," (but possibly falling short of the ideal) SoS chromosome for the next epoch. There may be additional interface or minimum SoS feasibility rules within the domain model, the GA or the fuzzy assessor. The Agent Based Model lets the SoS Agent make cooperation requests to the individual Systems' independent Agents, which have relatively simple rules for their negotiation strategies. We currently have three types of system agent negotiation strategies: selfish, opportunistic-Markov, and cooperative. The ABM portion of the framework manages the negotiations, and returns an "achievable," Systems negotiated architecture/chromosome for re-evaluation by the FIS and SoS Agent as a starting point for submission to the GA for evolution to the next epoch.

The overall sequence of activities of this modeling approach is:

- Define key goals and attributes of the SoS

- Feasibility rules may be established for some interfaces between systems (i.e., certain minimum interfaces may be required, certain others may be impossible)
- Define a fuzzy method of determining, from a binary participation chromosome, a level of performance or a value for each of the SoS attributes
- Define membership functions for attribute levels and get agreement among stakeholders
- Define weights and rules in a fuzzy inference system for combining attribute values to an overall SoS fitness
- A method of allocating initially proposed funding, performance and deadlines among the Systems and capabilities is devised
- The SoS agent uses the GA to select an SoS chromosome/capability set to be proposed to the Systems
- The ABM manages the negotiations among Systems' agents and SoS Agent to return a realizable SoS chromosome for the epoch
- The realized chromosome is evaluated again to determine the SoS fitness for the epoch
- This chromosome guides the SoS Agent, using the GA, to find and propose an improved chromosome, funding, performance and deadlines as the SoS Agent's proposal for the next epoch and a new ABM managed round of negotiations.

The model provides a set of systems and their interfaces with other systems as a linear array of bits; zero indicates non-participation in a performance capability, and one represents participation. If a System does not participate, then no interface with that System is possible. Incorporating a fuzzy evaluation in the process allows substantial nonlinearity to be introduced, but controlled by a simple, short list of rules. The overall SoS fitness measure may be used within both the GA and the ABM portions of the framework to compare different chromosomes, or examine the impact of changes to negotiation rule sets or environment values for the same chromosome, and other products of analysis. Funding is proposed by the SoS Agent once per epoch to each System. The SoS agent may include information about the desired interfaces in the allocation to each System and Capability, but funding is not spread directly to the interfaces by the SoS Agent, only to the System. Deadlines and Performance are handled the same way, one for each System and Capability.

2.3 Searching for SoS Meta-Architecture Through Multi-objective Optimization Methods

Architecting is a hierarchical reduction of ambiguity and multi-criteria NP complete optimization problem. In addition, architecture optimization can involve multilayers. In this research, both multi-criterion and multi-layer optimization aspects of architecting are considered. To find a solution to the multi-criterion optimization, it is possible to have a model that is based on ambiguity, in representing the architecture attributes. On the other hand, models based on multi- objective functions are considered when the multi-layer aspects of architecting are involved. In theory, meta- architecture trades can be generated through mathematical programming models, heuristics and domain dependent algorithms

2.3.1 Optimization

In optimization, the goal is to obtain the best result under the circumstances. Optimization theory embodies mathematical results and numerical methods to obtain best solutions from a set of alternatives at hand (Ravindran & Ragsdel, 2006).

2.3.1.1 *Mutli-Objective Optimization Models For Architecture Generation*

Optimization problems can have a number of objective functions to be optimized. Such optimization, which may also be called multi-objective programing, the problem can be stated as follows:

$$\text{Find } X \text{ which minimizes } f_1(X), f_2(X), \dots, f_k(X)$$

This is subject to the following constraints

$$g_j(X) \leq 0, \qquad where\ j = 1,2, \dots, m$$

$f_1, f_2, \dots, f_k$ are objective functions to be simultaneously minimized.

However, when architecture generation is specifically considered, the problem may rather be to maximize certain objective functions and or minimize others. For example, the objective functions can involve funding, deadline, performance and other criteria. In such a case, the problem can be considered a maximization problem of certain functions. For the opposite cases f_k can be transformed by taking its negative. That is $-f_k$ so those functions can also be maximized.

In addition, in general, vector X cannot simultaneously minimize or maximize all the k objective functions. As a result, the concept of the Pareto optimum solution is used for multi-objective optimization. So among the several methods that have been developed to solve the multi-objective optimization problem, most of these methods basically use the Pareto optimal solution. In addition, they use a criterion or rule to select one of the Pareto optimal solutions as the desired solution (Rao, 2009).

Pareto optimization can be expressed as follows:

- There are k objective functions to be maximized in order to select the best architectures
- The decision variables are expressed as a decision vector $(x_1, x_2, \dots, x_n)$ in the decision space X
- A function $f: X \rightarrow Y$ evaluates a specific solution expressed in objective space $(y_1, y_2, \dots, y_n)$ in objective space Y
- Assume the objective space to be a subset of the real numbers. That is $Y \subseteq R$
- In a single-objective optimization problem, a solution $x^1 \in X$ is better than $x^2 \in X$ if $y^1 > y^2$, where $y^1 = f(x^1)$ and $y^2 = f(x^2)$. Single optimization can work well for a model that is based on ambiguity for finding a trade-off in architectures
- In a case of a vector-valued evaluation function $f: Y \subseteq R^k$ and $k > 1$, to compare two solutions x^1 and x^2, the Pareto dominance is applied
- An objective vector y^1 dominates another vector y^2, expressed as $y^1 > y^2$ if no component of y^1 is smaller than the corresponding component y^2 and at least one component is greater

- Accordingly a solution x^1 dominates x^2 $(x^1 > x^2)$ if $f(x^1) > f(x^2)$
- The optimal solution in decision space can be expressed as the Pareto set $X^* \subseteq X$. Its image in objective space is denoted as the Pareto front $Y^* \subseteq Y$.

Optimal solutions in multi-objective multi-criteria optimization can apply one of the aggregation-based method, criterion-based methods or Pareto-based methods (Zitzler, Laumanns, & and Bleuler, 2004).

2.3.1.2 *Aggregation-based Method*

- In this method, trade-off surfaces are generated to aggregate the objectives into a single one.

2.3.1.3 *Criterion-based Methods*

- One method is to use one objective function at a time to determine the best among the architectures in the current set.
- Another method, assigns a different function to evaluate a certain portion of the set of architectures.
- A third one assigns probabilities for using different objective functions to assign the best. The probabilities can be user defined or random.

2.3.1.4 *Pareto-based Method*

Building on the above discussion, an optimal Pareto set P^* and Pareto front PF^* can also be defined as follows (Coello, 2004):

- Let $F(\vec{x}) = (f_1(\vec{x}), \ldots, f_k(\vec{x})$ be a specific solution for a minimization multi-objective problem, where $\vec{x} \in X$ given $\vec{x}$ is an n-dimensional decision variable vector $(\vec{x} = \mathrm{x}_1, \mathrm{x}_{2,} \ldots, \mathrm{x}_\mathrm{n})$, and $F(\vec{x}) \in Y$. An optimal Pareto set P^* in feasible region $X" \subseteq X$, where there is no $\vec{x}' \in X"$ for which $F(\vec{x}')$ dominates $F(\vec{x})$ can be defined as follows:
- $P^* := \{\vec{x} \in X | \neg \exists\, \vec{x}' \in X \;\; F(\vec{x}') > F(\vec{x})\}$
- The corresponding Pareto front PT^* can be defined as follows:
- $PT^* := \{F(\vec{x}) \mid x \in P^*\}$
- The set P^* or P_{true} corresponds to 'non-inferior', 'admissible' or 'efficient' solutions. For a given Pareto set P^* in search space, the objective function vector components cannot be improved simultaneously. The set PT^*or PT_{true}, also known by 'non-dominated' vector, is found based on phenotype operations or criterion space. In addition, the PT^* implicitly indicate acceptable genotypes for selection operations.
  -

2.3.2 Optimization Methodologies

Classical optimization theory uses differential calculus to determine the maximum and minimum points for functions that are unconstrained and constrained. The theory does provide the basis for most nonlinear programming algorithms to include the Jacobian and Lagrangean methods and Karash-Kuhn-Tucker conditions, which provide for almost all nonlinear programming problems. However, the classical optimization theory methods may not be suitable for efficient numerical calculations (Taha, 2008).

Optimization can be defined as a process that provides the conditions for determining the maximum or minimum of a function. There are ever increasing optimization techniques that are modern or nontraditional to include genetic algorithms, simulated annealing, ant colony optimization and particle swarm optimization. These recent methods provide powerful means for solving complex optimization problems by basically employing computerized search and optimization algorithms (Rao, 2009).

2.3.3 Evolutionary Methodologies For Solving Multi-Objective Functions

Various methods can be used to solve multi-objective functions. However, evolutionary methodologies, namely genetic algorithms, evolutionary programming and evolution strategies have proved themselves to be robust and powerful. They also possess characteristics that make them desirable for optimization problems that have multiple conflicting objectives with large and complex search spaces (Rao, 2009).

2.3.3.1 *Genetic Algorithms*

Many design optimization problems include mixed continuous and discrete variables. This is in addition to being characterized by design spaces that are discontinuous and non-convex. System architecting can be considered a design problem with similar characteristics. Using standard nonlinear programming techniques for such problems will be inefficient and computationally expensive. Furthermore, in most cases, a relative optimum is rather reached close to the starting point. On the other hand, genetic algorithms (GAs) are well suited for solving such problems with a high probability of finding the global optimum (Rao, 2009).

Holland demonstrated the tremendous advantage of using genetic algorithms to certain optimization problems that can exploit its procedure (Holland, 1973). Genetic algorithm technique is one alternative of various evolutionary computational methods of simulating natural evolution. It is basically a technique that incorporates a population of individuals formed of chromosomes mainly in a binary representation. Then, it propagates copies of the individuals based on fitness criteria, and finally new individuals are formed through the process of cross-overs and mutation (Fogel, 2006).

The principles of data structures and operators of genetic algorithms are tied to the genetic knowledge sprouting from Mendel's work. The genetic algorithm method parallels the workings of the genotype of an organism, which is a collection of genes, and the phenotype, which reflects the visible characteristics of the biological organism (Iba, 2012).

In genetic algorithms, the solution of an optimization problem starts with a population of size n, which is usually fixed. The initial population consists of a number of randomly generated strings. Each string or chromosome can denote a design vector. In this research, a string depicts an architecture vector. A

fitness value is determined for each string (design or architecture vector) when evaluated. Three operators, namely reproduction, crossover and mutation generate a new population of points (architectures). The fitness value is determined each time a new population is generated until the fitness values converge based on a certain criterion. Genetic algorithms have certain differences from the traditional optimization methods. These differences can be summarized as follows (Rao, 2009):

1. An initial population of values is used as a trial design vector, whereas traditional methods start with a single point. Due to the many points used in the initial solution, it is less likely that GAs get trapped in a local optimum
2. No derivatives are needed in genetic algorithms. The method uses only the values of the objective function.
3. Being represented in chromosomes of binary variables, genetic algorithms are well suited for solving discrete and integer programming problems. This property makes the method well suited for searching system architecture space for individual designs.
4. The fitness in natural genetics is replicated in genetic algorithms in the form of the objective function values.
5. Genetic algorithms work in an iterative process through many generations. A new set of genes is a result of random parent selection, cross over and mutation. As a result, the new combinations are efficiently explored based on available knowledge to find a new generation with better fitness. That is a better objective function value.

2.3.3.2 *Using Pareto-based Methods in Multi-Objective Algorithms*

In multi-objective problems (MOPs), the objective of multi-objective evolutionary algorithms (MOEA) is to converge into an optimal Pareto front that consists of a diverse set of points representing trade-offs in decision space (Coello, 2004).

In the Pareto-based methods the population can be ordered based on one of the following ways:

- Dominance Rank Approaches. Here the fitness value is determined based on the number of individuals by which an individual is dominated.
- Dominance Depth Approaches: Here the population is divided into several fronts with depth determined based on an individual's front.
- Dominance Count Approaches. Here the fitness value is based on the number of individuals dominated by an individual.

Recent research has focused on using evolutionary algorithms as a means for a good approximation because generating the Pareto set by traditional methods can be computationally expensive and often infeasible as its complexity prevents the application of exact methods.

However, approximating the Pareto set itself is a multi-objective problem, which can include:

- Minimizing the distance of the generated solution to the Pareto set.
- Maximizing the diversity of the achieved Pareto set approximation.

- Other objectives can be hyper-volume, spread and cardinality.

In addition, using evolutionary algorithms to solve multi-objective optimization is mainly motivated by the computational characteristic of evolutionary algorithms. In fact, the population used in the evolutionary algorithms provides a set of possible solutions simultaneously. This can result in finding several members of the Pareto optimal set in a single run as opposed to mathematical programming techniques where several separate runs would be needed (Coello, 2004).

Additional literature review for phase II will focus on the Pareto front.

2.4 Fuzzy Assessment for Multi-Attribute SoS Architectures

The analysis question addressed by this method is as follows: Given a set of participation / non-participation decisions made for each system in the proposed SoS, how well will a resulting candidate SoS perform? Any number of architecture attributes may be used for evaluation, but peoples' cognitive processing power, as well as the number of independent, orthogonal attributes, typically cannot sustain large numbers (Miller, 1956) (Pedrycz, Ekel, & Parreiras, 2011).

In DoD acquisition management, a program's (system's) current attribute status is often presented as a color code, as in the Contractor Performance Assessment Report (Department of the Navy, 1997); for example, plotted on stop light charts, or displayed on Kiviat charts. These reporting methods primarily allow reviewers to compare evaluation of alternative architectures, or a program status, from one quarterly review to the next. They gloss over many fine details to get to the attribute summary color or single word description. An overall evaluation, that combines several attributes, is still largely a gestalt of component attribute values, especially when not all areas are weighted equally. Attribute evaluations themselves are well suited to fuzzy logic approaches because of the difficult nature of boundaries between subjective evaluation ranges. A particular SoS architecture (chromosome) may fall partially into an Acceptable, and partly into a Marginal set, as any point on the quality scale between 2.3 and 2.8 does, shown in Figure 2. This region of confusion, ambiguity, or uncertainty can be due to

- Differing interpretation of components' status facts by the reviewing subject matter experts (SMEs)
- The energy of the presenter of the facts biasing the impression of the facts in the reviewers' minds
- Unconscious or even conscious bias of the evaluators.

The normal method of accounting for this in program management is to have multiple reviewers and do some kind of averaging and/or throwing out high and low values; similar to what is done in judging sporting events, like ice skating or gymnastics. This is basically what the fuzzy associative memory or fuzzy inference system does by summing the contributions of partial membership functions at an evaluation point in the judgment range.

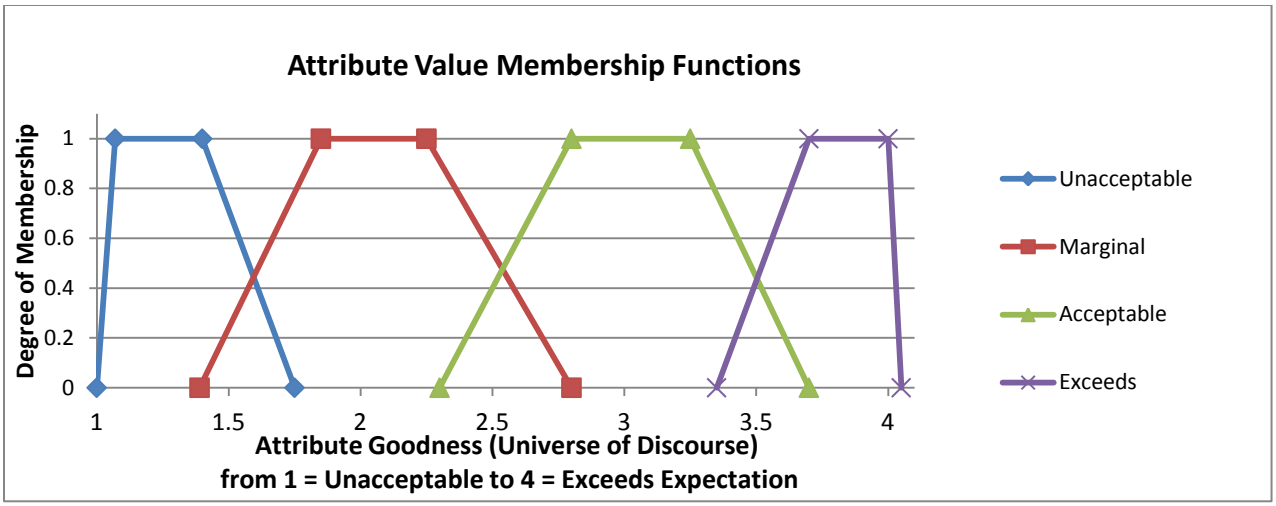


Figure 2. Fuzzy Attribute Value Membership Functions from RT-37

2.5 Domain Independent Architecture Assessment Method

The steps to create an SoS domain model for experimentation are largely independent of the domain.

- The attributes against which the SoS is to be evaluated must be identified, defined, compared and checked for independence or correlation between attributes. Having correlation decreases the value of the similar attribute to discriminate among chromosomes
- Methods to evaluate the SoS against these attributes must be developed, assessed, and most importantly, agreed to by the stakeholders
- To accomplish this, a series of guided interviews with stakeholders are conducted by subject matter experts with architecting skills
- Some of the required information is gleaned by simply letting stakeholders talk about their needs, their imagined or desired solutions and the characteristics of those solutions
- Some information comes from guided questions exploring the stakeholders' needs and current operations, state of technology, capabilities of existing systems, impact of changes in hardware, software, procedures, processes, and tactics, and outside influences such as competition or threats
- Several trial methods of mapping issues, features and attributes to architecture structures may be created and tested
- Validity and range of applicability of existing models for predicting performance, cost or other attribute values are explored
- Relevance and usefulness of existing performance models are compared with proposed simplified models
- The process must result in a well defined model for use in the SoS analysis.

The purpose here is to have a reasonably simple evaluation process that depends on binary decisions about participation and interfaces between systems in the SoS, as represented in the chromosome. It must also correlate with some impact to the SoS quality (fitness) through changes in the participation

model (the chromosome). Then it can be used as a fitness function in the optimization process of the GA, and also be handed off to the ABM to exercise the behavior and cooperation rule model. Our fuzzy modeling subgroup of SMEs acted as both interviewers and stakeholders for the development of the domain *dependent* model we used for the ISR SoS, discussed in section 3.3.

2.6 Modeling and Simulating Evolving SoS

Model Based Systems Engineering (MBSE) is becoming a more popular approach to system development as it provides a communication and verification that transcends the levels of development. MBSE uses a model or set of models to document and communicate from the system requirements levels down to the hardware or software implementation level. When a set of models is used, the models are connected and dependent on each other so that changes in one model automatically require the update of the set of models. This interdependence among the models provides a built in level of verification that may be costlier in document-centered system development, but is still the best practice in system development programs. The joining together of systems in the field has not yet been reflected in a movement to joint, major systems models. Instead, when this is necessary, typically much abstracted, ad hoc models are used, with the loss of fidelity that comes with that approach.

SoS development does not follow the normal system development process. An SoS typically does not have a normal program office organization. SoS capability is based on the contributions of the individual systems that comprise the SoS. This interdependence between the SoS and the individual systems makes a document-centric development less practical as an extensive effort is required to maintain the SoS development documentation. On the other hand, if SoS development is characterized by small extensions of the individual systems' capabilities, or the modeling the SoS may be achievable through small extensions of the existing program level models. This route may be risky and expensive. It has not been extensively pursued to date.

This research provides an Agent-Based Model (ABM) of SoS development to go beyond the purely technical joining of hardware and software. The ABM is based on the Wave Model (Dahmann, Rebovich, Lane, Lowry, & Baldwin, 2011) and can be executed using many initial SoS architectures with different input parameters. With an ABM the result is not a simple output as is found with discrete event models, but rather the ABM can be used to explore how different behaviors by different systems, with external parameters such as funding, priorities, performance, etc. can affect the resulting SoS capabilities, parameters, funding, etc..

2.6.1 Background

System of Systems development for the DoD does not follow the normal DoD acquisition process as defined in DoDI 5000.02. Instead, SoS development has been shown to follow a Wave Model process as shown in Figure 1 . The Wave Model of SoS development is like a bus-stop where the bus picks up passengers at a specific time interval. A passenger arriving at the bus-stop after the bus has left, must wait for the next time the bus arrives. Similarly, individual systems provide capabilities to the SoS but those capabilities must arrive at the height of the wave in order for the system capability to integrate into the overall SoS. A system that can provide a capability that arrives after the SoS integration point

(or height of the wave), must wait until the next wave for incorporation into the SoS. (Dahmann, Rebovich, Lane, Lowry, & Baldwin, 2011)

There are multiple types of SoS, such as Directed or Acknowledged, and the type of SoS determines the role and authority of the SoS manager in the SoS development. An Acknowledged SoS is by definition dependent on the individual systems to provide the capabilities that the SoS requires. The SoS has objectives, management, and funding but does not have the strong influence over the individual systems that a Directed SoS would have (Director Systems and Software Engineering, OUSD (AT&L), 2008). So the actual capabilities and performance of the SoS is determined by the cooperation of the individual systems with the SoS and with each other. In this research, an Acknowledged SoS is chosen as the overarching focus of the model based systems engineering product.

In this research, the SoS Architecture is a key to the SoS development. The importance of architecture to MBSE was shown in (Acheson, Dagli, & Kilicay-Ergin, Model Based Systems Engineering for System of Systems Using Agent Based Modeling, 2013). The ABM begins with an initial proposed SoS Architecture and this architecture reflects the systems providing capabilities as well as the system interfaces, with initial estimated budget, performance and schedule

2.7 Modeling and Simulating Systems Contributing to SoS

System of Systems (SoS) architecting poses challenges, as the solution space of the design is much more open compared to a standalone system (Kilicay Ergin & Dagli, 2008). Existing analysis methodologies and tools scope the SoS problem space by assuming that there is a limited set of solutions (Dagli, 2011) (NDIA, 11 October 2011). However, the SoS problem boundary includes integration of technical systems as well as cognitive and social processes, which alter system behavior (Dauby & Upholzer, 2011). As mentioned before most system architects assume that SoS participants exhibit nominal behavior (utopian behavior) but deviation from nominal motivation leads to complications and disturbances in systems behavior. It is necessary to capture the behavioral dimension of SoS architecture to be able to represent the full problem space to guide SoS analysis and architecting phase (Dauby & Dagli, 2011).

Agent based models (ABM) consist of a set abstracted entities referred to as agents, and a framework for simulating agent decisions and interactions (Brazier, C.M., & Truer, 1977) (Brazier, Dunin-Keplicz, Jennings, & and Treur, 1966). Agents have their own goals and are capable of perceiving changes in the environment. Simplified agent interaction rules may result in interesting group behavior. System behavior (global behavior) emerges from the decisions and interactions of the agents. The approach provides insight into complex, interdependent processes. Agent based modeling methodology has several benefits over other modeling techniques, such as Discrete Event modeling or System Dynamic modeling: it captures emergent patterns of system behavior, provides a natural description of a system composed of behavioral entities and is flexible for tuning the complexity of the entities (Bonabeau, 2002). A key characteristic of an SoS is the independence of the individual systems that comprise the SoS (Dahmann, Rebovich, Lane, Lowry, & Baldwin, 2011). The ABM has agents implemented as independent processes that more accurately reflects real world SoS development. The methodology is used in a wide range of application domains including financial markets (Kilicay-Ergin, Enke, & Dagli,

2012), homeland security applications (Weiss, 2008) and autonomous robots (Dudenhoeffer & Jones, 2000).

The goal of this research was to model SoS architecture evolution and acquisition based on the Wave Process Model and test the concept on the DoD Intelligence, Surveillance, and Reconnaissance (ISR) domain. The idea of Wave Planning was developed by Dombkins (Dombkins, 2007) and applied to the Trapeze Model of SoS Systems Engineering in order to illustrate the incremental and iterative process that characterizes SoS development (Dahmann, Rebovich, Lane, Lowry, & Baldwin, 2011). Agent based modeling methodology is well suited to abstract behavioral aspects of the acquisition process in the special case of SoS. In this project, the SoS and the individual Systems are embodied in agents. The System agents represent themselves (e.g., Program Manager) as well as any other individual stakeholders. The wave model applies to acknowledged (Director Systems and Software Engineering, OUSD (AT&L), 2008) SoS, thus there is a specific agent responsible for the SoS; that agent influences the cooperation of other System agents. An initial SoS mission is already determined and funds are allocated to the mission through a responsible organizational entity. The details of this work are detailed in the Complex Adaptive Systems Conference paper (Acheson, Pape, Dagli, Kilicay-Ergin, Columbi, & Harris, 2012).

2.8 Negotiation Between SoS and Systems Providing Specific Capabilities to SoS

The SoS development cycle starts with an initial SoS architecture that supports a set of capabilities (C_i). Each capability (C_i) has an associated weight (w_i). The SoS agent has a set of desired capabilities and associated performance levels and deadlines for each system. In order to encourage the system to provide the desired capabilities and performance levels, the SoS agent can offer some additional funding to the system or can adjust the deadlines when the system must deliver the capabilities. The system has its own priorities, schedules, funding, and goals and returns to the SoS agent what performance levels and deadlines it can support for what amount of funding. When this happens the SoS must assess the resulting SoS architecture and decide if the resulting architecture is “good enough”. If the resulting architecture is not good enough, then the SoS begins a negotiation cycle with the systems.

In order to support the negotiation cycle the SoS agent performs a Fuzzy Decision Analysis (FDA) that incorporates a Fuzzy Negotiation Model. The FDA calculates the gap between what was requested of each system and what the systems agreed to provide. The gap in performance, funding, and deadline is calculated for each system. These gap values are input to the Fuzzy Negotiation Model which outputs the adjustments to be made to the funding and deadline for each system. The details of the FDA are published in the International Journal of Soft Computing and Systems Engineering paper (Acheson, Dagli, & Kilicay-Ergin, Fuzzy Decision Analysis in Negotiation between the System of Systems Agent and the System Agent in an Agent Based Model, 2013).

3 Proposed Model Elements

3.1 ABM Framework

The overall ABM consists of 3 major elements; SoS acquisition environment, SoS agent, and a system agent. Each agent has its own set of behavior patterns. The ABM has one instance of the SoS agent and multiple instances of the system agent. The number of instances of the system agent corresponds to the number of systems in the SoS. The SoS instance and the individual system instances are embedded in the SoS acquisition environment model and are influenced by the changes in this environment. The ABM model user, in this case the SoS Acquisition Manager, provides domain dependent parameters as input to the ABM. The SoS agent is responsible for generating SoS meta-architecture based on these input parameters and evaluating the overall quality of the SoS architecture. The SoS agent behavior is abstracted based on the Wave Process model. Each instant of the individual system agent has its own decision model which can exhibit selfish, opportunistic or cooperative behavior. The ABM simulates the negotiation dynamics among SoS agent and individual system agent in order to analyze the SoS architecture evolution summarizes the overall model architecture.

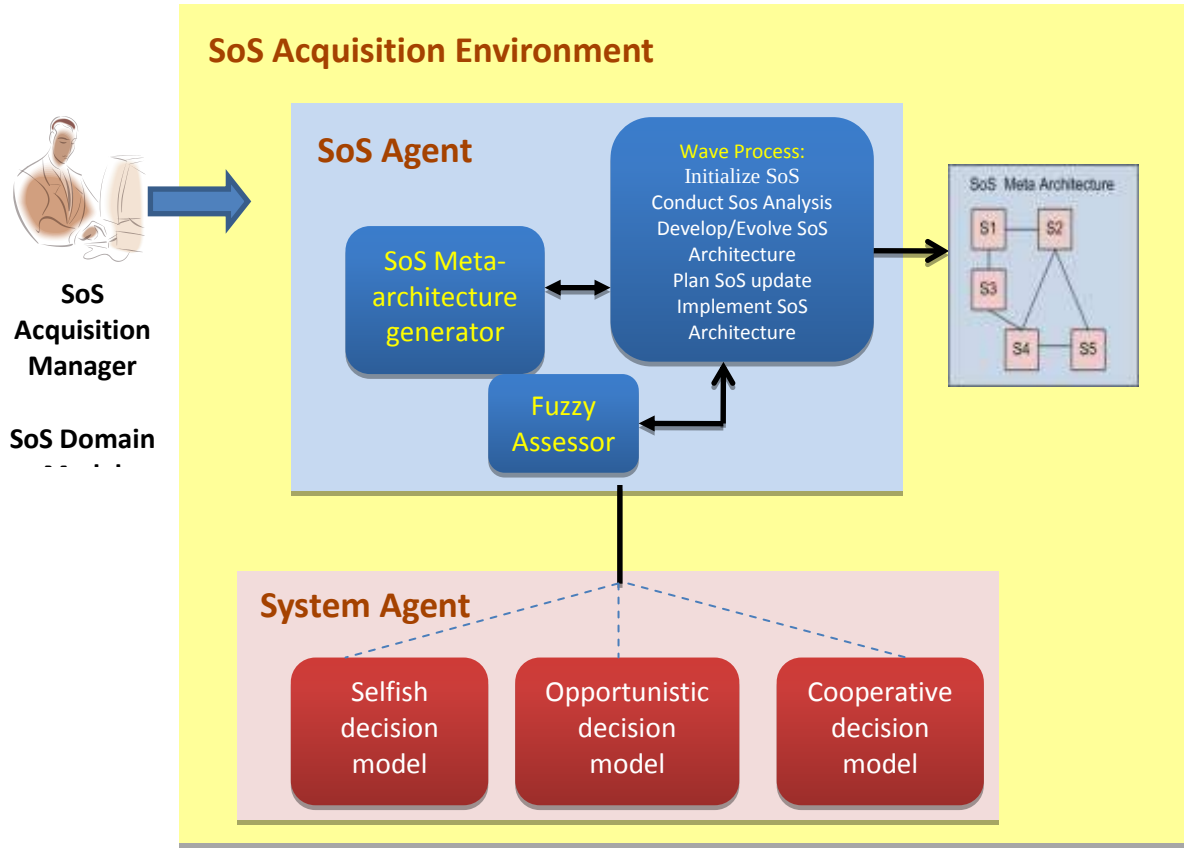


Figure 3. Overall Agent-based Model Architecture

The overall mathematical framework of the ABM is described based on the main elements of the model; SoS acquisition environment, SoS agent behavior and individual system agent behavior.

3.1.1 SoS Acquisition Environment

The SoS agent is influenced by the changes in the SoS acquisition environment. Let E_0 be the initial environment model which represents the SoS acquisition environment at wave time T=0. The SoS acquisition environment is influenced by several external factors including changes in the national priorities, changes in the SoS funding and changes in threats to the nation. Thus the initial environment model E_0 can be represented as a function of these variables.

$$E_0 = f(National\ priorities, SoS\ funding, threats)$$

As the SoS acquisition progresses through wave cycles, these variables are updated by the SoS Acquisition Manager to reflect current acquisition environment. Thus the environment model E_T at wave time T:

$$E_T = E_0 \sigma_T$$

where σ_T is the change in external factors at wave time T.

The environment model also includes two types of agents, the SoS agent and the individual system agent. The SoS agent is responsible for the development of the SoS architecture and the instances of the individual system agent represent independent systems that can provide the capabilities necessary for the SoS agent.

3.1.2 SoS Agent Behavior

SoS agent is responsible for the overall SoS engineering activity and coordinates with individual system agents to achieve the desired SoS mission. In the model, the SoS Acquisition Manager, also called the user of the ABM model determines the initial SoS mission, the SoS target measures based on the SoS domain. This information is provided as input to the SoS agent. The SoS agent follows the six core SoS engineering activities outlined in the Wave Process Model (Dahmann, Rebovich, Lane, Lowry, & Baldwin, 2011) in order to develop the SoS architecture. The SoS architecture evolves based on the behavior of individual systems as well as changes in environment model.

3.1.2.1 *Initiate SoS: SoS Domain Model and Parameters*

In this step, the SoS agent uses the domain model provided by the SoS acquisition manager to initialize model parameters. During the initialization phase, the wave interval, the time interval from one wave to the next, is set to a specific value. Let this value to be *epoch* and simulation time be *t*. Then wave time, T, is a multiple of wave interval and simulation time:

T= epoch* t

At T=0, the desired SoS capabilities are determined by the SoS Acquisition manager based on the SoS domain. Since some of these capabilities may have higher priority to achieve the SoS mission, weighted value for each capability is also determined at the initialization phase. Let:

$SoS.C_i = (C_1, C_2, \dots C_n)$ represent the set of desired capabilities to achieve the SoS mission,

$SoS.P_i = (P_1, P_2, \dots P_n)$ represent the set of desired performance parameters for these capabilities

$SoS.w_i = (w_1, w_2, \dots w_n)$represent the weighted values for each of the SoS capabilities

$SoS.M_0 = [a_{ij}]$ represent the initial SoS target measures where $[a_{i1}] = SoS.C_i$, $[a_{i2}] = SoS.P_i$ and $[a_{i3}] = SoS.w_i$

The set of individual systems for generating the SoS architecture are domain dependent. Thus a domain dependent model generates a feasible set of individual systems, $System.S_i = (S_1, S_2, \dots S_n)$ and interfaces $System.S_{ij} = (S_{11}, S_{12}, \dots, S_{nn})$ that can be integrated to meet the desired SoS target measures.

The capabilities of individual systems can be identified as a matrix, $System.c_i = [a_{ij}]_{n\times m}$ where $[a_{ij}] = 1$ if $System.S_i$ can provide $SoS.C_i$, else $[a_{ij}] = 0$

The performance parameters of individual systems can be represented as a matrix, $System.p_i = [a_{ij}]_{n\times m}$ where $[a_{ij}] = System.p_i$ if $System.S_i$ can provide $SoS.C_i$, else $[a_{ij}] = 0$

In a similar way, the deadline and funding for each SoS capability are also determined. Let the deadline for each capability be represented as $SoS.d_i = (d_1, d_2, \dots., d_n)$ which are increments of wave intervals and funding for each capability to be represented as $SoS.f_i = (f_1, f_2, \dots., f_n)$

3.1.2.2 *Conduct SoS Analysis*

Once the set of feasible systems are identified, a combination of SoS architecture alternatives are generated using the meta-architecture generation model where the SoS architecture is represented as a chromosome, $SoS.C_{g,n}$. The chromosome contains information on individual systems and interfaces between individual systems that are necessary to generate the SoS architecture. Thus the SoS architecture is represented as n×n matrix chromosome

$$SoS.C_{g,n} = [a_{ij}]_{n\times n}$$

where the interface between system i and System j is represented as $[a_{ij}] = 1\ if\ System.S_i\ has\ interface\ with\ System.S_j$, otherwise$[a_{ij}] = 0$. Since a system cannot connect to itself $System.S_i \neq System.S_j$.

The meta-architecture generation model uses the Fuzzy Assessor and Fuzzy Associative Memory (FAM) to evaluate the fitness of each alternative SoS architecture chromosome.

Let F represent the Fuzzy Assessor which maps individual system performance measures to SoS performance measures

$$F, SoS.C_{g,n}: System.p_i \rightarrow SoS.P_i$$

Let FAM represent the set of fuzzy association rules that map SoS performance measures to SoS architecture score

$$FAM, SoS.C_{g,n}: SoS.P_i \rightarrow ArchitectureScore.SoS.C_{g,n}$$

Once the SoS architecture score of each alternative chromosome is determined by the FAM, the meta-architecture generation model selects the chromosome that maximizes SoS architecture score, minimizes SoS funding and minimizes SoS deadline as the initial SoS baseline architecture:

$$SoS.A_0 = f(max(ArchitectureScore.SoS.C_{g,n}), min(SoS.d_i), min(SoS.f_i))$$

The initial SoS baseline architecture as well as the funding and deadline information is passed to the individual systems as SoS connectivity request to the SoS. Thus, the connectivity request $SoS.R_i$ is a function of

$$SoS.R_i = f(SoS.A_0, SoS.f_i, SoS.d_i)$$

Individual systems should evaluate whether they can develop the requested interface with other systems and the requested capabilities with the given funding at the requested deadline.

3.1.2.3 *Develop and Evolve SoS Architecture*

The SoS agent receives response from the individual systems. The individual systems may decide to cooperate with the SoS agent at the requested deadline and funding or they may request change in performance Δp, deadline Δd and funding Δf depending on their motivation. The SoS agent uses fuzzy association rules to govern the negotiation dynamics between individual systems.

Let $System_i.Gap$ represent the gap in performance, deadline and funding for individual systems.

Let FNM represent the set of fuzzy association rules for negotiation that map SoS capability weights, $SoS.w_i$ and $System_i.Gap$ to architecture update factor, $Beta_t$

$$FNM: SoS.w_i \; and System_i.Gap \rightarrow Beta_t$$

The current SoS architecture based on the negotiation is

$$SoS.A_t = SoS.A_0 + Beta_t$$

The SoS agent uses FAM to determine the architecture score for the current SoS architecture

$$FAM, SoS.A_t : SoS.P_i \rightarrow ArchitectureScore.SoS.A_t$$

The negotiation continues until a specified number of cycles are reached or a certain SoS architecture quality is reached.

3.1.2.4 *Plan SoS Update*

At the end of the wave cycle, the SoS agent evaluates changes in the external environment. Changes in national priorities, funding and threats have an impact on weighted values for each of the SoS capabilities, $SoS.w_i = (w_1, w_2, \ldots w_n)$. The weights for each capability are updated based on the external environment E_T. Thus SoS target measures update factor, $SoS.Alpha_T$, is a function of the environment at wave time T:

$$SoS.Alpha_T = f(E_T)$$

And at wave time T, the SoS target measures is updated as:

$$SoS.M_T = SoS.M_0 + SoS.Alpha_T$$

The updated SoS target measures are used as an input for the next wave cycle to update the SoS domain model.

3.1.2.5 *Implement SoS*

At the end of the wave cycle, the SoS agent updates the initial SoS baseline architecture based on the negotiation dynamics. Thus the expected SoS architecture at wave time T is

$$SoS.A_T = SoS.A_0 + Beta_T$$

3.1.2.6 *Continue SoS analysis*

The next wave cycle of the SoS development starts with the current SoS architecture, $SoS.A_T$once the SoS target measures are updated.

3.1.3 Individual System Behavior

Individual systems receive request for connectivity to SoS architecture, $SoS.R_i$. Since each system is independent and has its own goals and motivations, the system has the option to cooperate or negotiate with the SoS agent to request more funding, deadline or performance change. The individual system behavior depends on the decision model the individual system uses to negotiate with the SoS agent. The decision model can range from full cooperation to opportunistic to selfish model to maximize profits. The decision model that will be used by the individual system agent is determined by the SoS Acquisition Manager through the user interface for the SoS ABM. For details of the decision models refer to Section 4.6. Thus individual system $System.S_i$ responds back to the SoS agent with the change in funding, deadline, performance necessary to achieve the requested capability based on its decision model.

$$System.S_i = \{cooperative, opportunistic, selfish\}$$
$$System.Information_i = f(\Delta d, \Delta f, \Delta p)$$

3.2 Modeling and Simulating Evolving the SoS Through the Wave Model

The model development followed a spiral development approach. Multiple versions of the model were planned with increasing levels of capability and complexity.

3.2.1 First Version of the Model

3.2.1.1 *Description*

The first version of the model was developed in AnyLogic as agent-based model. This version of the model was an agent-based model that had one SoS agent and one system agent. The model had one instance of the SoS agent and ten instances (representing ten individual systems) of the system agent. The SoS instance would read in a chromosome (representing an initial SoS architecture) from an Excel file. The SoS would send out request for cooperation to all the individual systems. Each system would send back a message to the SoS indicating whether the system would cooperate or not. The determination whether the system would cooperate was based on a probability set by the user through the user interface. The model would repeat this process for ten chromosomes read from the same Excel file but in different worksheets.

The model included code to write data out to Excel files and text files from both the SoS agent and the system agent. This data included enough information to indicate what the SoS was doing as well as what each system was doing.

3.2.1.2 *What it Demonstrates/Shows*

The purpose of the first version of the model was to provide a proof of concept that the SoS could read the chromosome from the Excel file, send the request for cooperation to the systems, and receive response messages from each of the individual systems.

The purpose of this version was to provide a proof of concept that an agent-based model could be created that had:

- One SoS agent and one system agent
- Multiple instances of the system agent
- Communication between the SoS instance and all the system instances
- The initial SoS architecture in the form of a chromosome read in from an Excel file
- Results written to Excel file and to text files.

3.2.1.3 *Validation*

The validation for this version of the model was commensurate with the purpose of this version of the model. The validation was running the model and verifying that the items mentioned in the purpose

above were satisfied. Output files containing data from the model simulation showed that these objectives were satisfied and the proof of concept was validated.

3.2.2 Second Version of the Model

3.2.2.1 *Description*

The second version of the model was also a proof of concept as well as adding additional capability to the model. The overarching SoS development model is actually a set of models developed by different individuals independently. This version of the model built upon the first version of the model which was developed in AnyLogic. This agent-based model developed in Java needed to integrate with other models developed in Matlab (which is C code). The other models consisted of the following:

- Genetic Algorithm to determine the best SoS initial architecture
- Fuzzy Assessor used as the fitness function within the genetic algorithm and as the mechanism within the SoS agent to determine SoS architecture quality
- Three different system negotiation models (selfish, opportunistic, and cooperative) to provide the behavioral characteristics of the individual systems.

The three system negotiation models were developed in Matlab and Matlab was used to create three executable files which are called by the system agent. For this proof of concept model, system 1 calls the opportunistic system negotiation model which is implemented as a Markov stochastic process. System 2 calls the selfish negotiation model and system 3 calls the cooperative negotiation model.

3.2.2.2 *Differences from First Version*

3.2.2.2.1 Negotiation Model added to System Agent

Three different negotiation models have been added to the System Agent. The system 1 instance calls the Matlab executable that is the Opportunistic System Negotiation Model. System 2 instance calls the Matlab executable that is the Selfish System Negotiation Model. The system instance 3 calls the Cooperative System Negotiation Model. These three system negotiation models read and write read and write system information from Excel files. The name of the Excel file is passed to the Matlab executable when the Matlab executable is called. The System Agent waits for the Matlab executable to finish execution before proceeding to the next steps. This ensures that the system negotiation models have finished processing and that the results have been written to the Excel files.

3.2.2.2.2 Genetic Algorithm added to SoS Agent

The SoS Agent was updated to call the Genetic Algorithm (GA) executable. The GA was developed in Matlab and generates the initial SoS architecture used to start the first wave of the SoS development cycle. The SoS uses this initial SoS architecture to query each system to provide the capabilities in the initial SoS architecture. The GA also sets the initial values for performance, funding, and deadlines that are passed to the systems for the first cycle of the negotiation.

3.2.2.2.3 Other Changes

This version of the model changed the number of instances of the System Agent to 22. The first version of the model had ten instances of the System Agent. This version of the model now reads the initial SoS Architecture from the SoSArchitecture.xlsx file which is created by the GA Matlab executable. The first version of the model passed the Funding and Deadline to the System Agent in the SoS Request for Connectivity message. This version of the model no longer cycles to read in ten chromosomes as initial SoS Architectures but cycles for one epoch of the wave.

3.2.2.2.4 How do System Agent Environment variables affect the System architecture and the SoS architecture

The ABM does not have System Agent Environment variables at this time. The system environment for this version is inherent in the initial performance, funding, and deadline passed to each system. These initial values are determined by the GA at the beginning of the simulation.

3.2.2.3 *What it Demonstrates/Shows*

This second version of the model shows that the overall set of models can be integrated into one overall model by having the agent-based model call the executables created from Matlab. Matlab was used to generate executable files for each of the models developed in Matlab. The Matlab run time library was installed so that the Matlab executables would run without a Matlab license. The version of the Matlab run time library had to match the version of Matlab used to generate the Matlab executables. The agent-based model developed using AnyLogic then calls the Matlab executables at the appropriate time during the simulation.

3.2.2.4 *Validation*

The validation of this second version was commensurate with the purpose of this version of the model. The validation was running the model and verifying that the items mentioned in the purpose in 3.2.1 were satisfied. This validation occurred in two ways. The first validation was to make sure that the Matlab executables were actually running during the simulation. Checking the Windows Task Manager while the simulation ran, verified that the GA and Systems Negotiation Model executables were called and executed at the appropriate times during the simulation. The second validation was to make sure the Matlab code provided the same results when the Matlab executable was called from the ABM. This was accomplished by running the Matlab code in Matlab and comparing the results (both in the Matlab console window and in the Excel files) to the results when the Matlab executable was called from the ABM. The same output from the Matlab executables was shown in the AnyLogic Console window that can be seen in the Matlab window when the Matlab code is executed in Matlab. The outputs in the Excel files from running the ABM simulation was verified with the outputs in the Excel files when the Matlab code was executed in Matlab.

3.2.3 Future Versions of the Model

3.2.3.1 *Description*

The next version of the model will incorporate the following capabilities:

- Allow the user to determine which systems represent selfish, opportunistic, or cooperative behavior by selecting which systems will call which of the three system negotiation models.
- Implement the same Fuzzy Assessor used by the Genetic Algorithm as the fitness function
- Cycle for multiple epochs
- Start with new SoS architecture from the Genetic Algorithm at the beginning of each epoch
- Add the negotiation model into the SoS Agent

3.2.3.2 *What can be done to support other research*

Run the model multiple times using different combinations of system negotiation models and analyze the data to determine if conclusions can be drawn that would aid SoS managers in making acquisition decisions.

3.3 Domain Dependent Model Parameters and Analysis

3.3.1 Historical Example

A guiding physical example is taken from history. During the 1991 Gulf War, Iraqi forces used mobile SCUD missile launchers called Transporter Erector Launchers (TELS) to strike at Israel and Coalition forces with ballistic missiles. Approximately 50-60 TELs were hidden in the western Iraqi desert, from which Iraqi forces launched over 200 missiles during the 30 day conflict. The Iraqi forces had developed new techniques called “shoot and scoot” that allowed them to reduce the TEL vulnerability time to half an hour to set up for launch and return to their hiding places. This was only one third of pre-war estimates, and a great surprise to Coalition planners (Thompson, 2002). While the inaccurate Scuds were not a tactically significant factor in the war, they did have a significant strategic impact on morale and cohesiveness of the Coalition. Therefore, the TELs became a “high value, fleeting” target.

Existing intelligence, surveillance, and reconnaissance (ISR) assets were inadequate to find the TELs during their vulnerable setup and knock down time. The “uninhabited and flat” terrain in the desert was in fact neither of those things, with numerous Bedouin goat herders and their families, significant traffic, with thousands of wadis with culverts and bridges to conceal the TELs and obscure their movement. In addition, Iraqi forces produced some very fine camouflage and realistic decoys (Rosenau, 1991). Even though thousands of sorties were flown, during hundreds of opportunities, TELs were spotted only 11 times, and the contacts were lost before completing an attack eight of those 11 times. The average time between spotting and arriving at a potential target was 90 minutes, which might have been marginally acceptable before development of the shoot and scoot tactic (Thompson, 2002). This offers a prime example of existing systems being inadequate to address a mission, but some relatively low cost, quick changes, and joining together of existing Systems might be used to create an SoS capability to achieve the mission.

3.3.2 Inputs to A Domain Specific Model

- The set of existing, potential systems

- The set of capabilities that could be contributed by each system through small changes, such as adding a new interface
- A short list of rules for combining the systems, interfaces, and their capabilities
- A short list of what feasible changes (typically adding an interface) could (or could not) be made
- The method by which each *contribution* from the systems and interfaces is *measured* against the desired capabilities (Key Performance Atrributes – KPA)
- Estimated deadlines and funding for each System in the SoS

3.3.3 Domain Capabilities, Performance, Budgets and Deadlines

The SoS agent delivers individual plans to each system with proposed participation and interfaces in each capability, and proposed budget, performance contribution, and deadline for delivering the capabilities. The four attributes are evaluated at the SoS level, but negotiations about individual systems' participation are at their level. Not until the SoS agent has the answers from all the systems is the new chromosome worth evaluating again to the same initial attribute list. The domain model is represented as the performance budget and deadline in each capability from each system and for each of its interfaces. We are currently using a simple extension of a system capability to each interface, but there is room for adjustment to this quick approximation. The system budget is simply the sum of estimated development costs for each change or interface added. Estimates for these domain specific values are shown in Table 1

3.3.4 Domain Model Development: Feasibility

Before we know which systems participate in the SoS, we made a decision that the interfaces should be controlled by the way they interface through the communications systems. Most of the valuable real time interactions in the ISR SoS occur through communications links, so we counted them as the key to performance of the SoS. Rather than assuming that any random chromosome offered for evaluation is "feasible and correct" in its own right, we instead assume that only the communications interface portion is "correct." This is not too far fetched. It gives us a method for saying some chromosomes cost more or less and perform better or worse than other near by architectures. This borrows a concept from network centric warfare: that interconnections are key to increasing combat power. (Alberts, Garstka, & Stein, 1999)

Therefore, under the assumption that the communication interfaces are correct in the chromosome (from whatever source, whether randomly generated, evolved, or negotiated), that chromosome is input to the fitness function for evaluation. If a system is not present, then no interfaces with that system are possible. However, if two systems are present, and the chromosome says there is an interface between them, we first check if the communications systems interfaces allow both those systems to communicate for the interface to exist. If the chromosome does not allow communications, then that interface is not feasible. The domain model could easily be built with different rules for different systems, but this represents one way to impose this type of rules.

- IF S_g and S_h share a comm link, then they are *allowed* to be connected, or interfaced, with each other. So the first step is checking to see if any of the communication links (System 17-20) have an interface with S_g.
- IF it does, then check if that communication system ALSO has a link with S_h.
- Then, and only then, the interface between S_g and S_h is *feasible*. This continues for all four communication systems, "OR-ing" the results of each communications link (the source of interfaces in this model) feasibility check to create the initial feasibility matrix for the chromosome. (These connecting systems could be communities of interest, or classification levels, or languages in an allied force, just as easily
- If the chromosome also chooses to have an interface between S_g and S_h, that is a useful interface that increases performance.
- If the chromosome chooses *not* to have an interface between g and h, then it wasted an opportunity to have a useful interface, and is therefore penalized in performance.
- If an interface is *not* feasible but the chromosome does have a 1 in that bit position, it is penalized in cost for wasting resources by attempting to have an interface where it is not possible
- If an interface is not feasible and the chromosome also says it is not using that spot, the architecture is rewarded for wise use of resources

Therefore, the first step in the creation of the feasibility matrix is to find all the comm links' interfaces with every other system and record them in the interface bit position in the chromosome. A small complication to feasibility for the ISR domain is that only two RPAs can be controlled by a single ground station. So IF there are not enough control stations for the number of RPAs, we find the worst RPAs in terms of other interfaces, then delete an RPA until the participating control station(s) can handle them. The final limitations on feasibility are that the CONUS Exploitation station can only contribute if it has a BLOS link, U-2 can only contribute if there is at least one Exploitation Center, and DSP can only contribute if there is at least one (Exploitation *or* Control center).

The feasibility matrix is then compared with the remaining chromosome selected systems and interfaces. The fuzzy assessor has no way to control the evolution of individual bits in the chromosome directly. The only way it can contribute to selection of better systems and interfaces is to make *infeasible* choices costlier in funding terms, and weaker in performance terms. The GA should be able to figure out what works better as it evaluates fitness of new, evolved generations of chromosomes with the fuzzy assessor. To implement this concept, if the chromosome does have infeasible elements, it pays a fractionally higher cost for each one, and similarly, if the chromosome does not include a feasible interface as we just described, it pays a slight penalty in performance for every bit in the wrong place in the chromosome. In our demonstration implementation, both cost and performance are bumped up for feasible and down for infeasible elements. After SoS Performance is evaluated, Affordability, Flexibility and Robustness are calculated from the Domain Model and the Chromosome.

3.3.5 ISR Domain Model Detail

The ISR Domain Model contains the following types of Systems, with existing capabilities, cost bogeys for development and operation, and likely development times as follows:

- Fighter aircraft with ElectroOptic/Infrared (EO/IR) pods,
- Fighters with Synthetic Aperture Radar (SAR),
- U-2 aircraft and Satellites with EO/IR,
- DSP missile launch detection satellites,
- JSTARS aircraft with SAR, 6) Remotely Piloted Aircraft (RPA),
- Line of sight (LOS), that is, tactical; and Beyond Line of Sight (BLOS) data links,
- Exploitation centers in theater as well as in Continental United States (CONUS) for analyzing ISR data, and
- Control stations for the RPAs.

The five types of capabilities ($SOS.C_j$) provided by Systems and interfaces listed below are how we build up the SoS capabilities:

- EO/IR
- SAR
- Communications links
- Exploitation Centers
- Control Centers

A summary of postulated ISR SoS component system properties, capabilities, costs, and deadlines is in Table 1. It might be easier to add "weapons" as a separate capability, and to split Comm links into two different capabilities, LOS and BLOS, but we will just keep track of this separately for now. The rules for combining these Systems into a feasible SoS are relatively simple, but you can imagine that although they remove very large numbers of the total possible $2x10^{72}$ (for 22 systems, $2^{m(m+1)} = 2^{253}$) chromosomes from consideration, there are still many, many architectures to explore:

- There must be at least one System from each of the 5 types of capability (and one with weapons)
- Fighters carry several weapons, RPAs have one; nothing else has weapons
- All aircraft have at least one LOS comm; at least one aircraft must have BLOS comm, because the area is too large for purely LOS comms
- Bandwidth limits on LOS comm interfaces sort of match the lower capabilities of the Theater exploitation, making them about 10% as fast as BLOS and CONUS exploitation
- CONUS exploitation requires BLOS comm from the data collector or a connecting relay, but if the relay connects with the collector, it largely moots the timeliness improvement of CONUS exploitation
- RPAs and Control stations must have a common comm interface

Table 1. Domain model of SoS with 22 Systems: Capabilities, Costs, and Schedules

System	Type Sub-System	Capability #	Coverage sq mi/hr;	Bandwidth Mb	Attack Speed, MPH or process time, sec	$ Develop $M/ epoch/ interface	$ Operate $K/hr per system	Time to Develop, Epochs	Number possible	System Number
Fighter	EO/IR	1	500		350	.2	10	1	3	1-3
RPA	EO/IR	1	2000		150	2	2	2	4	4-7
Fighter	Radar	2	3000		350	.7	10	2	3	8-10
JSTARS	Radar	2	10000			.1	18	0	1	11
Theatre	Exploit	4	5000		90	2	10	1	2	12-13
Control Station/ AOC	C4I	5	1		30	1	2	1	2	14-15
CONUS	Exploit	4	25000		120	.2	0	0	1	16
LOS Link	Comm	3		.25	-	.2	0	1	2	17-18
BLOS Link	Comm	3		2	-	0.5	3	0	2	19-20
U-2	EO/IR	1	50000		-	0	15	0	1	21
DSP	IR	1	100000 *.01			1	1	0	1	22

Feasibility checks on the SoS architecture:

- System participation rules; if a system doesn't participate, then no interfaces to that system are feasible and any relevant chromosome interface bits are penalized.
- Interfaces to the comm links are taken as given by the GA with random or evolving changes; Interfaces between systems must be through comm systems; if two systems don't have a common comm link, there is no interface feasible between them.
- RPA must have at least one comm link to a control station; same rule as fighter interfaces. If enough control station systems (determined by the GA processes) do not participate, the RPA systems are limited in performance.
- U-2 and DSP don't interface with aircraft, but with ground centers.
- CONUS exploitation center must interface through a BLOS comm link. It automatically links with control centers.

3.3.6 SoS Attributes

We continued with the attributes from RT-37: **Performance**, **Affordability**, Developmental **Flexibility**, and Operational **Robustness**. We have upgraded the definitions through our SME discussions as follows:

3.3.6.1 *Performance Attribute*

Performance – for the ISR Domain example is made up of surveillance coverage in area per hour and wavelength region, combined with ability to reach the site of a discovered but fleeting high value target before it disappears.

- Background Assumptions: 100,000 square miles in which to hide; 30 minutes from start to finish for an operational launch; on the order of 60 TELs operational; an individual TEL might hide for several days, so the probability of an individual TEL popping out to make a launch is only about 10% per day.

Rules for combining capabilities into performance:

- Fighters can provide modest capability in non-traditional ISR with on board sensors, *and* deliver several weapons, but they cost more to operate than many other systems
- Remote Piloted Aircraft (RPAs) can provide better ISR capabilities with somewhat less speed and single weapon capabilities, but also require a control station for each 2 RPAs. They are considerably cheaper to operate than fighters
- JSTARS can provide considerable radar ISR, and LOS and BLOS relay, but no weapons
- DSP can provide reliable notice of an actual launch, which means there definitely was a TEL in the open at that launch point, but it is not very precise localization of that point, meaning some search is still required upon an armed vehicle's arrival in the vicinity, and it takes a couple minutes to receive the data from DSP. The TEL can hide quickly after launch, leaving not much time to arrive there, find and attack it before disappears again. In the performance model, I multiplied the DSP coverage by 0.01 to account for the likely lack of closure from a DSP detection
- U-2 or Satellite can cover large area with high resolution, but turnaround time is hours; participation of U-2 or Satellite effectively decreases total area to be searched by other ISR platforms by a reasonable percentage, but does not affect real time surveillance. DSP is basically free, because it is used for other purposes
- Area to be covered is divided into sectors among the participating surveillance systems
- Time to arrive is proportional to the square root of the sector area being covered by each type of system, plus some time for transmitting data to, and double checking by, the exploit systems to insure the target is valid and not in a restricted area
- Probability of successful engagement is *defined* as 50% if coverage is the total area in half an hour, and time to arrive after detection is less than 10 minutes. Fighters or RPAs making the discovery are able to attack relatively quickly, transit time is less than 5 min for fighters, 10 min for RPAs; other types of detection require transit time for the attack vehicle which may be longer if it is in a different sector.

3.3.6.2 *Performance Evaluation*

Performance grading of the SoS will be by probability of a successful targeting of a TEL per day, given six opportunities per day: (this matches Iraqi Scud employment during the Gulf War). These ranges are the fuzification of the performance variable. (Sumathi & Surekha, 2010)

- Unacceptable: probability of successful engagement per day less than 23%;
- Marginal is between 23 - 45%;
- Acceptable is between 45 - 60%;
- Exceeds expectation is greater than 60%.

The following algorithm gives the performance model:

- Search area = total area * (0.9^(number of U-2 and Surveillance Satellites))
- Coverage by SoS per hour = (500* num EO/IR Fighters) + (3000* num SAR Fighters) + (2000*num RPA)+ (.01*100000 * num DSP)+(10000 * num JSTARS)
- Exposure time = 0.5 hours
- Coverage percentage = 100*(Search area/(coverage area per hour * exposure time +1))
- Transit time if discovery is not by armed System (i.e., JSTARS or DSP)= [sqrt(search area / sum of armed systems +1)]/(average speed of armed systems+1) (avg speed = (num fighters *350 mph + num RPAs* 150 mph)/ (num fighters + num RPAs +1)) (extra 1 in the denominators to avoid dividing by zero)
- Probability of detecting & targeting a TEL in the open (Pdt) =Coverage percentage due to armed system + coverage percentage due to JSTARS, DAP *(1-(transit time+3 min exploitation)/exposure time))
- Probability of finding and targeting one TEL per day =1 –((1 – Pdt)^number of TEL launches). Shown in Figure 4.
-

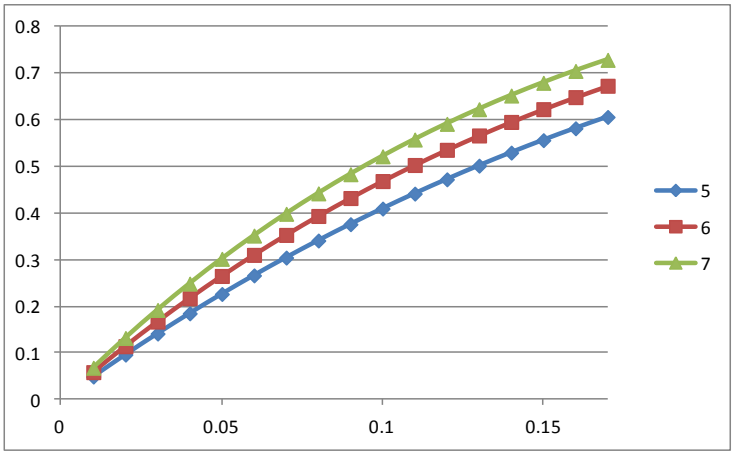


Figure 4. Graph of probability of success per day versus coverage Pdt and number of launches

3.3.6.3 *Costs/Affordability Attribute*

Affordability – counts both development cost and operations cost. Operations cost is evaluated for one month of 24/7 operation. The development and operations costs for types of systems and interfaces are found in Table 1. The development costs are split up by the ratios of the estimates in Table 1. The fuzification of the costs are as follows:

- Unacceptable: more than $125M sum of development and operating cost
- Marginal: between $90M d and $125M sum of development and operating cost
- Acceptable: between $50M and $90M sum of development and operating cost
- Exceeds affordability expectations: less than $50M sum of development and operating costs.

3.3.6.4 *Flexibility Attribute*

Flexibility – is based on the number of different *types of capabilities* on systems performing development to achieve the new capabilities. Fewer types of systems give the SoS manager less flexibility to rearrange his choices and funding to replace a system that chooses not to participate, for whatever internal reason. If there are more types of systems making small contributions, the manager has more flexibility to manage, and it reduces development risks, should one interface modification prove to be more difficult to achieve.

Flexibility grading fuzzification is:

- Unnacceptable: single System for multiple types of capabilities allows multiple single point failures in development;
- Marginal: no more than one type of capability being supplied by only a single System;
- Acceptable: No type of capability has only one source;
- Exceeds: more than one capability has multiple sources and no capability has a single source.

3.3.6.5 *Robustness Attribute*

Robustness – is based on the maximum percentage of delivered performance capability lost from the loss of *any* individual system and all its interfaces during operation. This requires repeated evaluation of the performance of the SoS for the loss of each participating system; the development cost is sunk, but the operating cost changes along with the performance, so it is possible that affordability could change with performance. Flexibility does not change, because that is measured only during development. Not sure at this stage if we should use Affordability, since the fuzzy assessor uses all four attributes to evaluate the SoS; since we don't know what Robustness is for this calculation, it would probably be more self consistent to use only performance change for Robustness measure.

Robustness grading fuzzification is:

- Unnacceptable: degradation greater than 33%;
- Marginal: Maximum capability degradation less than 25%;
- Acceptable: Maximum capability degradation less than 12%;
- Exceeds: Maximum capability degradation less than 12%.

These estimates are merely the starting point; basically the SoS manager's initial estimates of the individual system's capabilities, costs, etc. They may need to be adjusted during the negotiation process, but we're keeping it simple so far. Currently we do not adjust the evaluation criteria after negotiation, and the fuzzy assessor does not evaluate the deadline or schedule data.

Using the domain model in Table 1 we constructed relationships for each of four SoS attributes from the chromosome. We updated the calculation of Performance, Affordability, Flexibiilty, and Robustness from the very simple model of RT-37 that basically only counted ones in different parts of the chromosome.

3.3.7 Performance of the Fuzzy Assessor

Performance of the SoS now includes dividing the required total search area by the number of ISR vehicles in the SoS, using their inspection area per hour and speed to evaluate their probability of finding an individual target, also using speed to get to a target identified by another system in an adjacent sector. The measure of performance (MOP) for individual targets, or for individual searching systems, is not high, but the final performance of the SoS is the likelihood of finding and hitting one target per day with all the elements of the SoS cooperating. This reduces the correlation between number of systems and performance. Performance evaluations of random chromosomes seem to be reasonably distributed between around 10% to 80% chance of finding a target per day, and varies quite reasonably with the number and type of systems present in the chromosome and the domain model. A range of performance values for thousands of random chromosomes is shown in the third subplot of Figure 6. These chromosomes were completely random, not developed in the GA, but only to test the fuzzy evaluation algorithms.

The cost naturally increases with number of systems (with differing costs for each system/capability) so better performance is slightly correlated with worse Affordability. We included an operations cost element as well as development cost in the affordability attribute measure, shown in columns 7 and 8 ofTable 1. This adds variability to the value of the attribute not entirely under the control of the participating System Program Offices, who only control the development funding. Selecting systems with high operational costs IS somewhat under the control of the SoS Agent through the GA for the initial chromosome, but it is less subject to influence by the System Agents during their negotiations on development costs and deadlines. The last 3 subplots of Figure 6 show the range of Affordability at the SoS level, and the development and operations costs for many random chromosomes.

The Flexibility attribute changed from the simple number of systems to look at the variety of system types that can provide each capability. If only one system type, such as fighter, is present in the chromosome to provide Radar capability, there is less flexibility than if both fighters and JSTARS are in the SoS. The point is to provide two opportunities to get a Radar capability. RPA control was only available from the control stations, so we excluded that type from the flexibility calculations. If all the remaining four capabilities are provided by more than one system type, that is excellent SoS Flexibility; if more than two capabilities are provided by only a single system type, that is much worse flexibility. It is still correlated with the total number of systems, but now you can have excellent flexibility with only 9

total systems of the 22 possible, if they are chosen wisely. On the other hand, you can have poor Flexibility with as many as 13 participating systems.

The Robustness calculation was also changed significantly in this increment. Now we take the chromosome and its performance, then cycle through every participating system, removing one at a time, along with all its interfaces, and recalculate the performance. Using the baseline of all selected bits in the chromosome, as we remove each system successively, the performance of the SoS could actually improve for some removals, due being penalized for that system having many infeasible interfaces. But typically, the performance degrades for removal of a system from the architecture. The loss of actual performance is measured and stored for the removal of each of the systems; if the maximum loss is large, then the SoS is not very robust. A large loss means that a lot of performance is contributed by only one system; if something happens to that one system, such as enemy development of a countermeasure, or a fleet logistics problem that sidelines that system until it is fixed, then your overall SoS capability is greatly reduced at that time. Instead, if the SoS performance is more evenly distributed over many systems, that is a more robust SoS architecture. On the other hand, if the performance attribute of the SoS architecture is not very good to begin with, then there is not much to be lost through removal of individual systems. Therefore the robustness of this otherwise poorly performing architecture may be adjudged pretty good in this case, but the performance attribute already captures the poorness of the SoS architecture. The fuzzy combination rules prevent much compensation from one attribute by another. Even with excellent robustness, poor performance trumps it in the SoS fitness evaluation.

We believe the changes to the definitions of attributes are a significant improvement over the initial feasibility demonstration in RT-37. Another refinement is that we departed from the purely trapezoidal membership functions, which were simple enough to understand, but also provided limited bands of fitness evaluations. This does not provide a useful range of fitness values for the GA to choose one chromosome over another, or rank them among a population of chromosomes. We chose to move to another membership function model from the MatLab Fuzzy Logic Toolbox™ that tracks the trapezoidal functions, but rounds the corners a bit with a Gaussian combination. The reason we did this is because the trapezoidal functions resulted in “chunky” evaluation distributions shown in Figure 6 on the left, compared to the more helpful distribution shown on the right.

Our original concept of the fuzzy membership functions for the goodness of SoS attributes from RT-37 is shown in Figure 2. The corresponding membership functions from the MatLab Fuzzy Toolbox are shown on the left half of Figure 5, to show we’re being faithful to it. Figure 6 shows the plots for 100 randomly generated chromosomes with the following subplots:

1) The total number of ones in the chromosome of possible 253 (for 22 systems), and (to see it on the same scale) five times the number of Systems in that chromosome (first 22 bits only)
2) The output of the fuzzy evaluation of the entire SoS chromosome against the ISR domain model, with a vertical scale from 1=unacceptable, 2=marginal, 3=acceptable, and 4=exceeds expectation. The crisp numbers are rounded to the nearest integer to get the final evaluation

3) The Performance of each SoS architecture
4) The Flexibility attribute evaluation
5) The Robustness
6) The Affordability
7) One component of the Affordability attribute: the development cost
8) The second component of the Affordability: the operations cost.

It is interesting to note that both good and bad architectures appear across a range of number of participating systems and interfaces. This is true both for the overall SoS fitness, and for individual attributes scaled values. There is some correlation between performance and number of systems up to a point; there is generally anticorellation between number of systems and affordability, and the others are a bit more random.

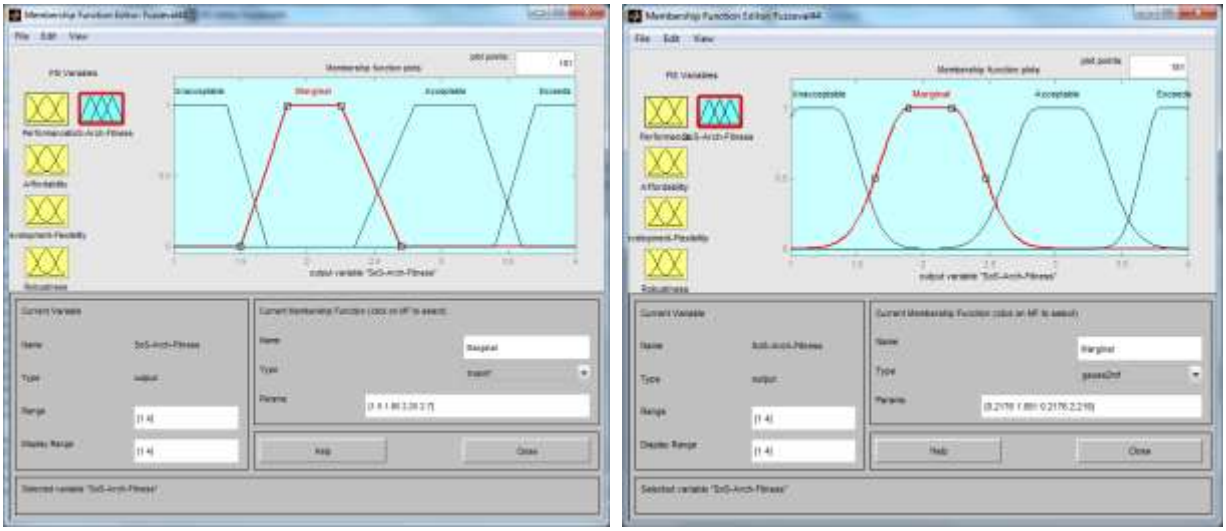


Figure 5. MatLab Fuzzy Toolbox membership function display on the left equivalent to Figure 2, and with Gaussian rounding of corners on the right

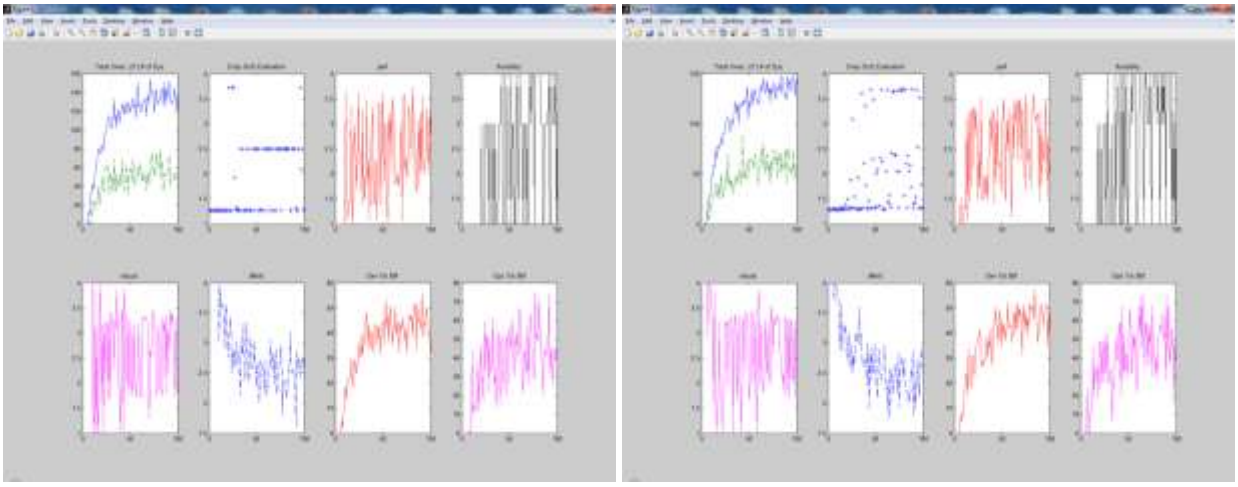


Figure 6. Fuzzy evaluations of random chromosomes, using trapezoidal membership functions on left, and the Gaussian rounded function on right. Note fewer "exceeds" (>3.5 on the vertical scale) on the left and significant clustering at 1.2 and 2.5, and far less clustering as well as greater number of "exceeds" in the crisp evaluations on right

The SoS evaluations (shown in the second subplot of Figure 6), for the original trapezoidal membership functions, while acceptable, tended to clump significantly around low and medium values, also resulting in few architectures being graded above 3.5, or "exceeds" in our overall schema. This result *could* work in the GA, but could require many generations to converge. A wider variation in the fitness value of different chromosomes would aid the sorting of improved chromosomes in each generation. The MatLab Fuzzy Logic Toolbox™ provides the gentler, rounder membership function shape on the right side of Figure 5. By changing to these Gaussian rounded membership functions, while keeping to the spirit of our SME derived membership functions, we were able to get far more tractable evaluations for purposes of the GA, shown in the right side of Figure 6.

Because visualizing the chromosome in its linear form is difficult, we can arrange the systems on the diagonal of an upper triangular matrix, and the interfaces between S_i and S_j in row i and column j of the matrix. In this display, as shown in Figure 7.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	F EO/IR
	1	1	0	0	0	1	0	1	0	0	0	0	0	1	0	1	1	0	1	1	0	2	F EO/IR
		1	0	0	0	1	0	1	1	0	0	0	0	1	0	0	1	0	0	1	0	3	F EO/IR
			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	R EO/IR
				0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	R EO/IR
					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	R EO/IR
						1	0	1	1	0	0	0	0	0	0	0	0	0	1	1	0	7	R EO/IR
							0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	F RAD
								1	1	0	0	0	0	1	0	1	1	0	1	0	0	9	F RAD
									1	0	0	0	0	1	0	1	1	0	1	0	0	10	F RAD
										0	0	0	0	0	0	0	0	0	0	0	0	11	J RAD
											0	0	0	0	0	0	0	0	0	0	0	12	T EX
												0	0	0	0	0	0	0	0	0	0	13	T EX
													0	0	0	0	0	0	0	0	0	14	CONT
														1	0	0	1	0	1	1	0	15	CONT
															0	0	0	0	0	0	0	16	C EX
																1	1	0	1	1	0	17	LOS
																	1	0	1	1	0	18	LOS
																		0	0	0	0	19	BLOS
																			1	1	0	20	BLOS
																				1	0	21	U2
																					0	22	DSP

Figure 7. Matrix view of the architecture chromosome with participation highlighted in blue. Note that there are no interfaces with non-participating systems(on the diagonal) in this example. System types andnumbers are labeled on the right

In Figure 8, we get the same view from a trial random chromosome in the MatLab output of the fuzzy assessor checkout; the light blue squares represent a participating interface (a one) this time, while the yellow squares are the participating Systems on the diagonal, and dark blue are zeroes. Later, we'll explain how if a system does not participate, any interface with that system cannot be there either.

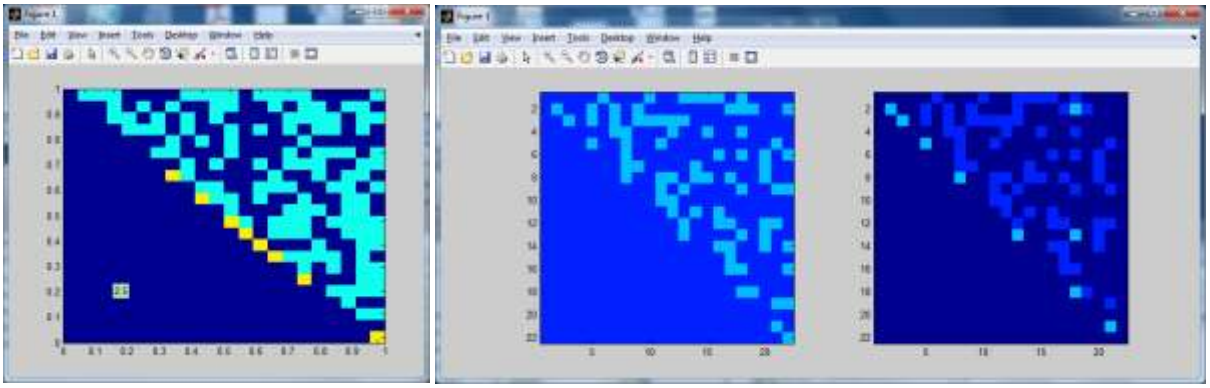


Figure 8. Upper diagonal matrix look at a chromosome. Darker blue is zero, light color is interface ones, yellow is participating systems (along the diagonal) in the left sample

Finally, we still use the fuzzy assessor (or fuzzy associative memory or fuzzy inference system) to combine the individual attributes for the overall SoS fitness evaluation. This combines the effect of the membership functions discussed above with the rules for combining attributes shown below inTable 2. The evaluation is used both within the GA, to sort generations of chromosomes to produce the

optimized architecture (given assumed capabilities, costs, deadlines, etc.) for the SoS Agent starting point, and again to evaluate the negotiated architecture at the end of a round of negotiations, or one wave model epoch. We retained the simple rule structure from RT-37 for combining attribute values for the overall SoS evaluations. This framework is set up to allow changes to the relative weights between attributes by very simple rule changes. Defining new attributes is quite simple within the fuzzy inference system framework in the MatLab Fuzzy Logic Toolkit.

Table 2. Simple Fuzzy SoS Evaluation Rules

Plain Language Rule	Fuzzy Rule Definitions from MATLAB Fuzzy Toolbox
If ANY attribute is Unaccptable, then SoS is Unacceptable	• If (Performance is Unacceptable) or (Affordability is Unacceptable) or (Developmental_Flexibility is Unacceptable) or (Robustness is Unacceptable) then (SoS_Arch_Fitness is Unacceptable)
If ALL the attributes are Exceeds, then the SoS is Exceeds	• If (Performance is Exceeds) and (Affordability is Exceeds) and (Developmental_Flexibility is Exceeds) and (Robustness is Exceeds) then (SoS_Arch_Fitness is Exceeds)
If ALL the attributes are Marginal, then the SoS is Unacceptable	• If (Performance is Marginal) and (Affordability is Marginal) and (Developmental_Flexibility is Marginal) and (Robustness is Marginal) then (SoS_Arch_Fitness is Unacceptable)
If ALL the attributes are Acceptable, then the SoS is Exceeds	• If (Performance is Acceptable) and (Affordability is Acceptable) and (Developmental_Flexibility is Acceptable) and (Robustness is Acceptable) then (SoS_Arch_Fitness is Exceeds)
If (Performance AND Affordability) are Exceeds, but (Dev. Flexibility and Robustness) are Marginal, then the SoS is Acceptable	• If (Performance is Exceeds) and (Affordability is Exceeds) and (Developmental_Flexibility is Marginal) and (Robustness is Marginal) then (SoS_Arch_Fitness is Acceptable)
If ALL attributes EXCEPT ONE are Marginal, then the SoS is still Marginal	• If (Performance is Marginal) and (Affordability is Marginal) and (Developmental_Flexibility is Marginal) and (Robustness is Acceptable) then (SoS_Arch_Fitness is Marginal)
	• If (Performance is Marginal) and (Affordability is Marginal) and (Developmental_Flexibility is Acceptable) and (Robustness is Marginal) then (SoS_Arch_Fitness is Marginal)
	• If (Performance is Marginal) and (Affordability is Acceptable) and (Developmental_Flexibility is Marginal) and (Robustness is Marginal) then (SoS_Arch_Fitness is Marginal)
	• If (Performance is Acceptable) and (Affordability is Marginal) and (Developmental_Flexibility is Marginal) and (Robustness is Marginal) then (SoS_Arch_Fitness is Marginal)

Figure 9 is a plot of thousands of runs for random chromosomes with varying numbers of ones. This is still using the trapezoidal membership functions. The number of ones in the chromosome (five times the number of systems, in red) is shown on the first subplot, the second subplot shows that the vast majority of SoS chromosomes are low scoring on the scale of 1=unacceptable to 4=exceeds performance. The other Attributes are also plotted, as well as the development and operations costs.

The purpose of these runs was to examine the shape of the performance model to insure that various chromosomes could reach all regions of the four attributes. We think this is amply demonstrated.

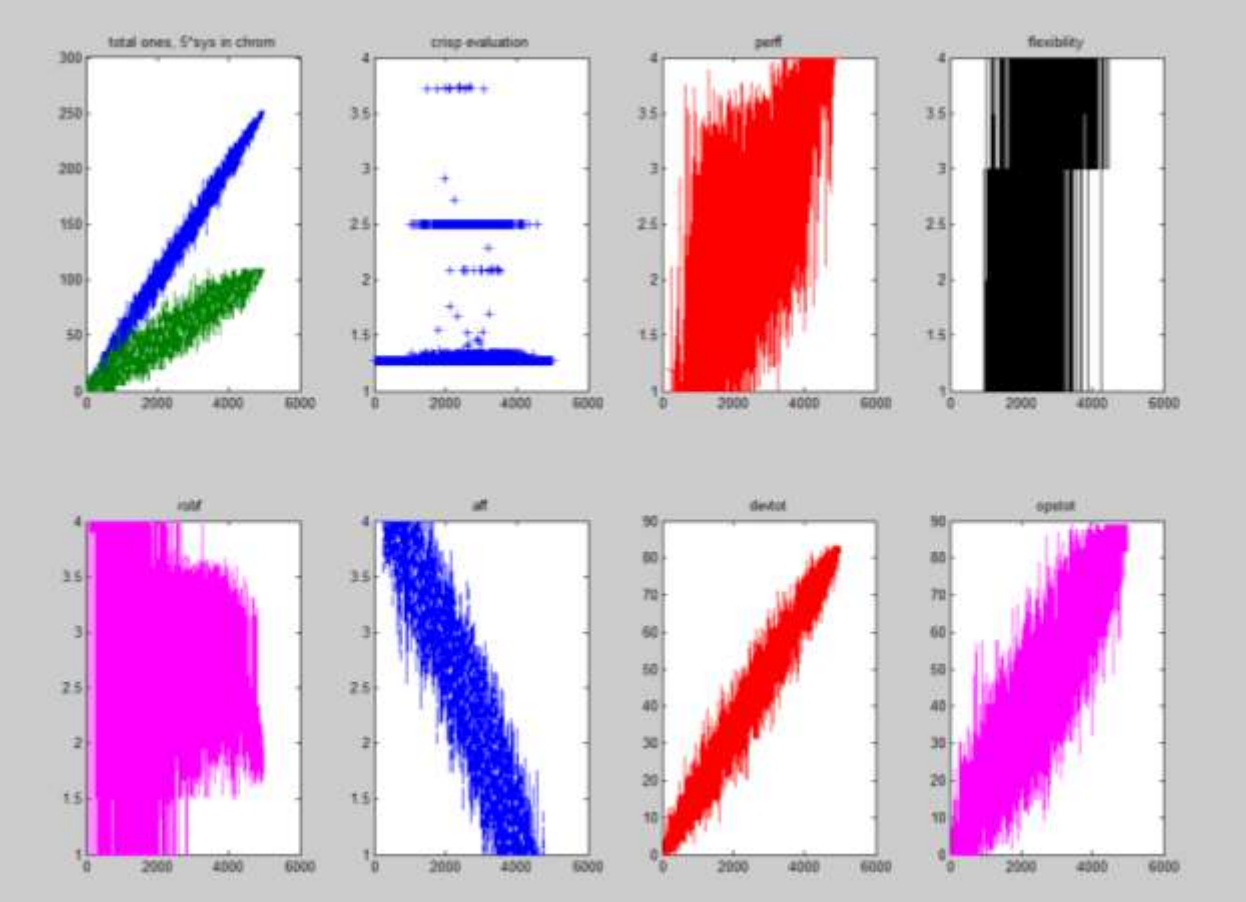


Figure 9. Range of Attribute Values for Random Chromosomes

Figure 10 is another look at the range of values produced by numerous random chromosomes, but this time they are sorted by the SoS evaluation in the second plot. This helps us to understand where the "sweet spot" of numbers of systems and interfaces is, in this version of an ISR model. This led us to examine the rounded corner membership functions. It required only one click in the Fuzzy Logic Toolbox to change the shape of the membership functions as shown in Figure 5. Finally, Figure 11 shows the smoother evaluations of slightly different chromosomes which we chose to go forward with.

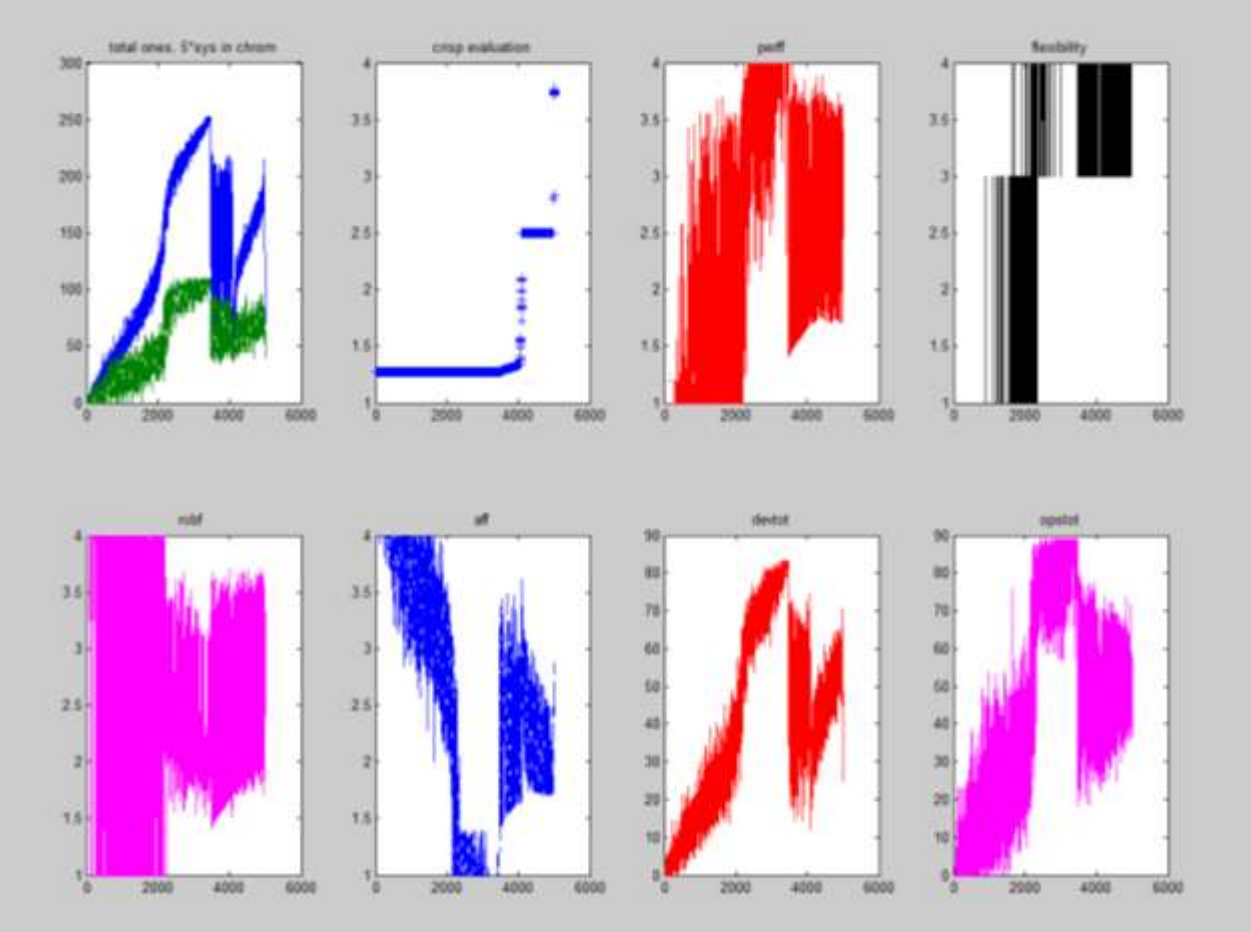


Figure 10. Chromosomes sorted by SoS Fitness, trapezoidal membership functions

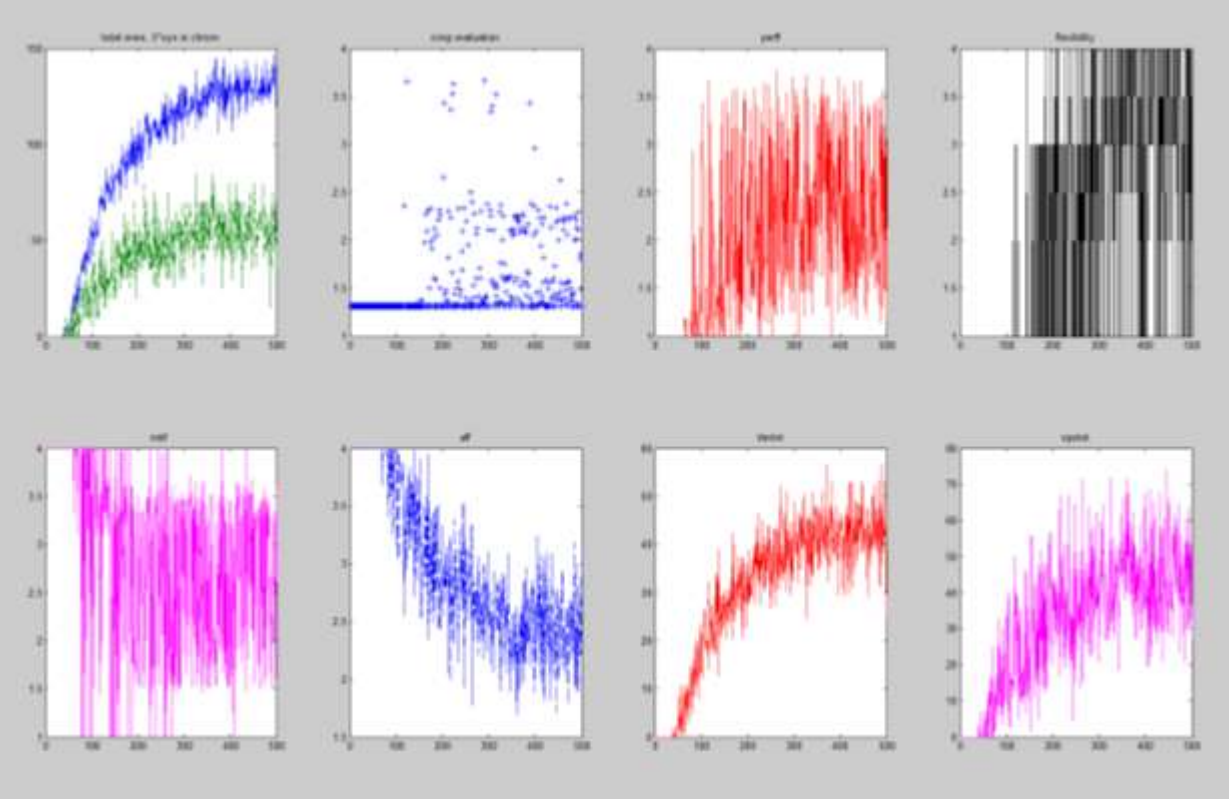


Figure 11. Improved variation in fitness level of random chromosomes with modified membership functions

3.3.8 Generating Capability, Performance, Cost, Schedule Matrices

After evolving and selecting a chromosome based on SoS fitness, the information to provide the individual systems is prepared. An Excel file with 4 sub matrices equivalent to (SOS.M, and SOS.C_g) for capability allocated to systems and interfaces, SOS.P for performance to each capability, SOS.f for funding, and SOS.d for deadlines, is prepared in the format of file "domainParameters.xlsx." This is created from the domain specific model coded in the Matlab file "DomainModel.m" (in Appendix Par 9.2) for the GA to process into individual system files for the SoS agent to provide to each system agent.

3.4 Domain Independent Multiple Objective Meta architecture Generation Models Based on Domain Inputs

3.4.1 Multi-Objective Meta-Architecture Optimization

The meta-architecture aims at generating architectures for the Systems of Systems (SoS) and selecting the best architecture with optimal results for inputs to the Agent Based Model (ABM).

3.4.1.1 *Variables*

The meta-architecture model uses the following variables.

- n: Number of capabilities in the SoS

- m: Number of systems in the SoS
- l: Number of possible interfaces in the SoS
- C_i: Capability i where $i = 1, 2, \dots n$
- S_j: System j involved in the SoS architecture where $j = 1, 2, \dots m$
- I_k: Interface k involved in the SoS architecture where $k = 1, 2, \dots l$
- s_{ij}: A variable indicating assignment of system S_j to capability C_i
- r_{ik}: A variable indicating assignment of interface $I_k$to capability C_i
- p_{ij}: Level of performance of system S_j in providing capability C_i
- f_{ij}: Cost of acquiring capability C_i on system S_j
- d_{ij}: Time for acquiring capability C_i on system S_j
- p_{ik}: Level of performance of interface I_k in providing capability C_i
- f_{ik}: Cost of acquiring capability $C_i$on interface I_k
- d_{ik}: Time for acquiring capability C_i on interface I_k
- P_i: Level of performance expected of capability C_i
- F_i: Budget allocated for capability C_i
- D_i: Deadline to build capability C_i
- P_{SoS}: Overall performance of the SoS
- F_{SoS}: Overall funding requirement of the SoS
- D_{SoS}: Total time required for the SoS
- Φ_{SoS} ( ): A fuzzy function that involves various ambiguous decision variables, and fuzzy assessor parameters
- A_{SoS}: System of systems architecture
- A_i: System architecture to provide capability i where $i = 1, 2, \dots n$
- g: Number of sets of architectures in architecture space
- h: Number of architectures in an architecture set
- $\overrightarrow{A_{SoS,b}}$: Set b of system of systems architectures
- $\overline{A_{SoS,\bar{a}\,b}}$: System of systems architecture $\bar{a}$ with best fitness in set b where $\bar{a} \leq h$ and $b = 1, 2, \dots g$
- $\overline{\overline{A_{SoS}}}$: Optimal architecture with best fitness value in architecture space
- $\overrightarrow{A_{ib}}$: Set b of system architectures to provide capability i where $i = 1, 2, \dots n$ and $b = 1, 2, \dots g$
- w_p: Value for providing relative weight to SoS performance in calculating fitness
- w_f: Value for providing relative weight to SoS funding in calculating fitness
- w_d: Value for providing relative weight to SoS deadline in calculating fitness
- φ_{SoS}: Fitness value of SoS architecture
- $\varphi_{SoS,ab}$: Fitness value of SoS architecture a in set b of SoS architectures where $a = 1, 2, \dots h$ and $b = 1, 2, \dots g$
- $\overline{\overline{\varphi_{SoS}}}$: Optimal fitness value for architecture space
- $\overline{\varphi_{SoS,b}}$: Best fitness value among SoS architectures in set b of SoS architectures where $b = 1, 2, \dots g$

3.4.1.2 *Mathematical Model*

All the n capabilities are required for the SoS architecture to be generated. The SoS architecture is formed by selecting from the pool of m systems and possible l interfaces, where

$$l = \frac{m(m-1)}{2}$$

Consider S_j a system j where $j \in \{1,2,\dots,m\}$ and m is the total number of systems. In addition, consider I_k be the interface between the systems $S_j\ with\ S_{j'}$ where $j' \in \{1,2,\dots,m\}$ and $j' \in \{1,2,\dots,m\}$. Now, consider the set of all interfaces a graph G of size m. Then, it can be represented by its adjacency matrix, whose elements are given by the following:

$$I_k = I_{jj'} = I_{j'j} = \begin{cases} 1, if\ exists\ an\ edge\ between\ vertices\ j\ and\ j' \\ 0, otherwise \end{cases}$$

Since an interface cannot connect a system to itself then itself $j \neq j'$. This is $I_{jj'} = 0 \quad \forall j = j'$.

That is the diagonal of the adjacency matrix $S(G)$ will have the values zero. In addition, since an interface needs to be considered only once for the connection of two systems, only the upper triangle of the matrix needs to be considered, whereas the remaining elements of the matrix take the value of zero. The following illustrates 3×3 adjacency matrix representing interfaces in the upper triangle, which depicts existing interfaces between three systems:

$$S(G) = \begin{Bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{Bmatrix}$$

Consequently, for vector architecture representation, the lower triangle and diagonal are not considered.

The model uses row vector representation of architectures with all systems $S_j\ \forall j$ involved in the SoS architecture where $j \in \{1,2,\dots,m\}$ represented side by side, then next to them all first system interfaces I_k where $k \in \{1,2,\dots,m-1\}$ depicted in the first row of the upper triangle of the adjacency matrix. These are followed by all second system interfaces I_k where $k \in \{m, m+1,\dots,2m-3\}$ as depicted in the second row of the adjacency matrix. This is repeated for all $(m-1)$ system interfaces depicted by the corresponding rows in the adjacency matrix. The element to the very right of the architecture vector is for the interfaceI_l connecting systems $(m-1)$ and m. This corresponds to the second from the bottom of the adjacency matrix.

In the architecture vector structure, each system or interface found in a possible architecture is represented by a binary digit, with connected taking the value '1' while not connected taking a '0'.

The various capabilities are assigned to the different systems and interfaces as provided below.

$$s_{ij} = \begin{cases} 1\ if\ \ capability\ C_i\ is\ assigned\ to\ system\ S_j \\ 0\ if\ not \end{cases}$$

$$r_{ik}=\begin{cases}1\ if\ \ capability\ C_i\ is\ assigned\ to\ interface\ I_k\\0\ if\ not\end{cases}$$

If a system S_j can provide an interface as a surrogate to systems that are not connected, the system itself is still considered a system and not an interface. If in this case the system S_j itself can no interface, then its respective interface assignment to all other systems will take the value '0'. That is $r_{ik} = 0 \quad \forall k$.

In addition, there is at least one system assigned to each capability. This can be expressed as follows:

$$\sum_{j=1}^{m} s_{ij} \geq 1 \qquad \forall i$$

So if a system is assigned to one or more capabilities then it will be included in the SoS architecture. Otherwise, it will not. This can be formulated as shown below:

$$S_j=\begin{cases}1 \text{ if } \sum_{i=1}^{n} s_{ij} \geq 1\\0 \text{ if } \sum_{i=1}^{n} s_{ij} = 0\end{cases} \quad \forall j$$

Similarly, if an interface is assigned to one or more capabilities then it will be included in the SoS architecture. Otherwise, it will not. This can be formulated as shown below:

$$I_k=\begin{cases}1 \text{ if } \sum_{i=1}^{n} r_{ik} \geq 1\\0 \text{ if } \sum_{i=1}^{n} r_{ik} = 0\end{cases} \quad \forall k$$

However, an interface can be considered for the SoS architecture if and only if both systems connected by the interface are also included, which can be expressed by the following technical feasibility constraint:

$$I_k=\begin{cases}0 \text{ or } 1 \text{ if } both\ S_j\ and\ S_{j'} = 1\\0 \text{ if } \ either\ S_j\ or\ S_{j'} = 0\end{cases}$$

This is true for all interfaces, where I_k is interface of $S_j\ with\ S_{j'}$. Furthermore, a system cannot have an interface to itself $j \neq j'$.

In addition, technically feasible interfaces can happen only if the two systems under consideration can functionally, physically or logically be connected. For example, if a sender communication system needs to connect to another communication system, the other system needs to have a receiving capability to be considered for the interface. Where an interface is not feasible the capability assignment to the interface takes the value '0'. That is $r_{ik} = 0$.

Finally there are multiple objectives that need to be simultaneously optimized as follows:

1. The overall SoS architecture performance: P_{SoS}

Here, the objective is to maximize the performance for an effective SoS architecture. For this, the individual capabilities are required not to exceed minimum acceptable performance levels P_i.

To meet this objective, the performance of each capability is determined by the maximum performance of all systems and interfaces providing the capability.

Performance of all systems providing a certain capability:

$$max_{\forall j}(p_{ij}s_{ij}) \quad \forall i$$

Performance of all interfaces associated with the same capability:

$$\max_{\forall k}(p_{ik}r_{ik}) \quad \forall i$$

From this, the performance of the SoS can be measured as the maximum among all capabilities constituting the SoS:

$$\underset{\forall i}{max}[\underset{\forall j}{max}(p_{ij}s_{ij}), \underset{\forall k}{max}(p_{ik}r_{ik})]$$

2. The overall funding requirement for SoS: F_{SoS}

The objective is to find the minimum funds that are needed for the SoS architecture development. For this, the individual capabilities are required not to exceed their respective budgets F_i.

To meet this objective, the cost of providing each capability is determined by finding the total cost of all systems and interfaces providing the capability as follows:

Cost of all systems providing a certain capability:

$$\sum_{\mathrm{j}=1}^{m} f_{ij}s_{ij} \qquad \forall i$$

Cost of all interfaces associated with the same capability:

$$\sum_{\mathrm{k}=1}^{l} f_{ik}r_{ik} \qquad \forall i$$

Total cost for the capability:

$$\sum_{\mathrm{j}=1}^{m} f_{ij}s_{ij} + \sum_{\mathrm{k}=1}^{l} f_{ik}r_{ik} \quad \forall i$$

From this, the overall cost of the SoS can be found as the sum of costs for all the n capabilities constituting the SoS:

$$\sum_{\mathrm{i}=1}^{\mathrm{n}}(\sum_{\mathrm{j}=1}^{m} f_{ij}s_{ij}+\sum_{\mathrm{k}=1}^{l} f_{ik}r_{ik})$$

3. Total time required for the SoS: D_{SoS}

Similarly the objective is to find the shortest time needed for the SoS architecture development. For this, the individual capabilities are required not to exceed specific deadlines D_i.

To meet this objective, the time for providing each capability is determined by finding the maximum time of all systems and interfaces providing the capability, assuming all capability development start at the same time, as follows:

Time of all systems providing a certain capability:

$$max_{\forall j}(d_{ij}s_{ij}) \quad \forall i$$

Time of all interfaces associated with the same capability:

$$\max_{\forall k}(d_{ik}r_{ik}) \ \forall i$$

From this, the time required for the SoS can be found as the maximum time among all capabilities constituting the SoS:

$$\max_{\forall i}[\max_{\forall j}(d_{ij}s_{ij}), \max_{\forall k}(d_{ik}r_{ik})]$$

3.4.2 Optimization Model

Having defined all variables and formulated all equations, an optimization model can be integrated as follows:

Maximize $P_{SoS} = max_{\forall i}(P_i)$

Minimize $F_{SoS} = \sum_{i=1}^{n} F_i$

Minimize $D_{SoS} = max_{\forall i}(D_i)$

Subject To:

$\sum_{j=1}^{m} f_{ij} s_{ij} + \sum_{k=1}^{l} f_{ik} \iota_{ik} \leq F_i \quad \forall i$

$max_{\forall j}(d_{ij} s_{ij}) \leq D_i \qquad \forall i$

$max_{\forall k}(d_{ik} r_{ik}) \leq D_i \qquad \forall i$

$max_{\forall j}(p_{ij} s_{ij}) \geq P_i \qquad \forall i$

$max_{\forall k}(p_{ik} r_{ik}) \geq P_i \qquad \forall i$

$\sum_{j=1}^{m} s_{ij} \geq 1 \qquad \forall i$

$$S_j = \begin{cases} 1 \text{ if } \sum_{i=1}^{n} s_{ij} \geq 1 \\ 0 \text{ if } \sum_{i=1}^{n} s_{ij} = 0 \end{cases} \qquad \forall j$$

$$I_k = \begin{cases} 1 \text{ if } \sum_{i=1}^{n} r_{ik} \geq 1 \\ 0 \text{ if } \sum_{i=1}^{n} r_{ik} = 0 \end{cases} \qquad \forall k$$

$$I_k = \begin{cases} 0 \text{ or } 1 \text{ if } both\ S_j\ and\ S_{j'} = 1 \\ 0 \text{ if } either\ S_j\ or\ S_{j'} = 0 \end{cases} \forall k$$

where I_k, is the interface of $S_j\ with\ S_{j'}$; and $j \neq j'$

3.4.3 Solution of the Model

A simple example has been solved numerically in iterations is provided in Appendix 7.1.

Genetic algorithms (GA) have been used to solve the optimization model for the meta-architecture.

3.4.3.1 *Data and File Structures Used in Integrating the Models to Generic ABM*

1. Individual System Data Outputs

An output file is created for each system S_j involved in the SoS architecture where $j = 1, 2, \ldots m$. The file is constituted of a table of multiple rows. Each row depicts one capability C_i where $i = 1, 2, \ldots n$. This includes whether the system S_j has been assigned to the capability C_i. If assigned, s_{ij} shows the value '1' indicating assignment of system S_j to capability C_i. Otherwise the value shown is '0'. Similarly, the row shows the assignment status of all the system interfaces r_{ik}. In addition, it provides the aggregate capability performance, deadline and funding for each capability C_i.

Figure 12 below shows the partial Excel file table for individual systems specifying capability participation of the system and interfaces in up to ten various capabilities are used. The deadline, funding and performance for each capability are also provided. More specifically the figure shows part of an output sheet for system 1. A complete spreadsheet includes all the respective interfaces. There are 22 files that are provided as output.

n= 5 capabilities
m= 22 systems
j= 1

	Architecture (the portion related to system j)					
	S_j	$I_{j,1}$	$I_{j,2}$	$I_{j,3}$	$I_{j,4}$	$I_{j,5}$
C_1	**0**	**0**	**0**	**0**	**0**	**0**
C_2	0	0	0	0	0	0
C_3	**1**	**0**	**0**	**1**	**1**	**0**
C_4	**1**	**0**	**0**	**0**	**0**	**0**
C_5	0	0	0	0	0	0

Figure 12. Individual System Capability Participation

•

2. SoS Data Output

An output file of the system of system A_{SoS} architecture is created . Each cell depicts a system S_j has been selected to be part of SoS architecture. If selected, S_j shows the value '1'. Otherwise the value shown is '0'.

Figure 13 below depicts the output including 22 systems and respective interfaces totaling 253 systems and interfaces. The sheet has been truncated for visibility.

Systems Contributing to the SoS Architecture

1 means system is contributing
0 means system is not contributing

	Systems						
	S_1	S_2	S_3	S_4	S_5	S_6	S_7
Final Architecture	**1**	**0**	**1**	**1**	**1**	**1**	**1**

Figure 13. Output chromosome

3. Domain Parameter Data Inputs

An Excel file with four tables specifying the domain parameters required for the genetic algorithms process is used.

I. Figure 14 specifies the assignment s_{ij} for each system S_j, and the assignment r_{ik} for each interface I_k. This is provided for each capability C_i. The following truncated sheet clarifies.

DOMAIN PARAMETERS
m= 22 systems
n= 5 capabilities
Capability Provided by Systems 1 means capability provided by system 0 means capability not provided by system

	Systems										
	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}
C_1	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**
C_2	0	0	1	1	1	1	1	0	0	0	0
C_3	**1**	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**1**
C_4	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**
C_5	0	1	0	1	1	1	0	1	1	1	1

Figure 14. Capabilities

Figure 15 specifies the performance p_{ij} for each system S_j , and the performance p_{ik} for each interface I_k. This is provided for each capability C_i. The following truncated sheet clarifies.

System Performance for Each Capability

	Systems						
	S_1	S_2	S_3	S_4	S_5	S_6	S_7
C_1	**0**	**0**	**5**	**2**	**0**	**2**	**0**
C_2	0	0	5	5	1	2	1
C_3	**4**	**0**	**7**	**3**	**0**	**10**	**0**
C_4	**3**	**0**	**0**	**0**	**0**	**3**	**0**
C_5	0	5	0	3	3	3	0

Figure 15. Performance

- II. Similarly the third table (not shown) depicts the funding f_{ij} and f_{ik} for each system and interface respectively.
- III. Finally the last table (not shown) provides the deadline values d_{ij} and d_{ik} for each system and interface respectively.

3.4.3.2 *Optimization*

The multi-objective genetic algorithm optimization (MOGAO) used for RT44 utilizes the weighted method and starts by reading the inputs file and gets the domain parameters that are described above, namely $s_{ij}$the initial data for the variable indicating the possible assignments of systems S_j to capabilities C_i, r_{ik} the initial data for the variable indicating the feasible assignments of interfaces $I_k$to capabilities C_i, p_{ij} the level of performance of system S_j in providing capability C_i, f_{ij} the cost of acquiring capability C_i on system S_j, d_{ij} the time for acquiring capability C_i on system S_j, p_{ik} the level of performance of interface I_k in providing capability C_i, f_{ik} the cost of acquiring capability $C_i$on interface I_k, and d_{ik} the time for acquiring capability C_i on interface I_k.

In addition, the algorithm generates an initial random population having g chromosomes for each capability. This also corresponds to $\overrightarrow{A_{ib}}$ the set b of system architectures that provide capability C_i where $i = 1, 2, \ldots n$ and $b = 1, 2, \ldots g$. It also specifies deterministic values for the weights of the objective functions namely, w_p the value for providing a relative weight to SoS performance in calculating fitness, w_f the value for providing a relative weight to SoS funding in calculating fitness, and w_d the value for providing a relative weight to SoS deadline in calculating fitness. It also specifies the number of generations h.

3.4.3.3 *Phenotype Operations*

Based on the initial set of architectures $\overrightarrow{A_{bi}}$ that is generated at random for each capability as described above, the capability performance, funding and time needed are found. Criterion evaluation is repeated

for each in the set of capability architectures leading to corresponding system of system (SoS) evaluation. Then, the process is repeated for all architecture sets generated throughout the genotype operations. The genetic architecture generation is described in the next section. The evaluation of the different criteria uses the optimization model described in previous sections and is outlined below.

The following finds the performance of certain SoS architecture a in set b of SoS architectures where $a = 1, 2, \dots h$ and $b = 1, 2, \dots g$

$$P_{SoS,ab} = \max_{\forall i}[\max_{\forall j}(p_{ij}s_{ij}), \max_{\forall k}(p_{ik}r_{ik})]|_{ab}$$

The funding required for the SoS architecture is calculated for certain SoS architecture a in set b of SoS architectures where $a = 1, 2, \dots h$ and $b = 1, 2, \dots g$ below:

$$F_{SoS,ab} = \sum_{i=1}^{n}(\sum_{j=1}^{m} f_{ij}s_{ij} + \sum_{k=1}^{l} f_{ik}r_{ik})|_{ab}$$

The deadline that can be met by the SoS architecture a in set b of SoS architectures where $a = 1, 2, \dots h$ and $b = 1, 2, \dots g$ is found by:

$$D_{SoS,ab} = \max_{\forall i}[\max_{\forall j}(d_{ij}s_{ij}), \max_{\forall k}(d_{ik}r_{ik})]|_{ab}$$

Finally, the evaluation of the different criteria is combined in a fitness value to be used in the genotype operations. The fitness value of the architecture is calculated as the weighted sum of the above functions.

$$\varphi_{SoS,ab} = w_p P_{SoS,ab} + w_f F_{SoS,ab} + w_d D_{SoS,ab}$$

Initially, scaling factors were applied in order to address the different units and magnitudes. However, this did not seem to provide any benefit.

The set of SoS architectures is designated $\overrightarrow{A_{SoS,b}}$. The best fitness value among SoS architectures in set b is found as follows:

$$\overline{\varphi_{SoS,b}} = \max_a(\varphi_{SoS,ab})$$

The optimal fitness value among SoS architectures in architecture space is found as follows:

$$\overline{\overline{\varphi_{SoS}}} = \max_b(\varphi_{SoS,b})$$

A SoS architecture a in set b of SoS architectures, designated $A_{SoS,ab}$, is found by integrating the systems and interfaces used in the respective architectures of all capabilities as follows:

$$A_{SoS,ab} = \sum_{i=1}^{n} A_{iab} \mid \exists\{S_j \in A_{SoS,ab} = \begin{cases} 1 \ if \sum_{i=1}^{n} s_{ij} \in A_i \geq 1 \\ 0 \ otherwise \end{cases} \forall j\} \cup \{I_k \in A_{SoS,ab} = \begin{cases} 1 \ if \sum_{i=1}^{n} r_{ik} \in A_i \geq 1 \\ 0 \ otherwise \end{cases} \forall k\}$$

The best SoS architecture $\overline{A_{SoS,\bar{a}\,b}}$ with the best fitness in set b of SoS architectures is found by integrating the systems and interfaces used in the respective architectures of all capabilities as follows:

$$\overline{A_{SoS,\bar{a}\,b}} = \sum_{i=1}^{n} A_{i\bar{a}\,b} \mid \exists\{S_j \in \overline{A_{SoS,\bar{a}\,b}} = \begin{cases} 1 \;\; if \sum_{i=1}^{n} s_{ij} \in A_i \geq 1 \\ 0 \;\; otherwise \end{cases}_{\forall j}\} \cup \{I_k \in \overline{A_{SoS,\bar{a}\,b}} = \begin{cases} 1 \; if \sum_{i=1}^{n} r_{ik} \in A_i \geq 1 \\ 0 \; otherwise \end{cases}_{\forall k}\}$$

The optimal architecture $\overline{\overline{A_{SoS}}}$ in architecture space is one with optimum fitness $\overline{\overline{\varphi_{SoS}}}$.

3.4.3.4 *Genotype Operations*

The genotype operations are conducted on the capability architectures for all capabilities. The architectures are represented in chromosomes in the genotype operations. The chromosomes are constitutedcomprised of binary strings. Thus, the chromosome alphabet is limited to two letters. A system or interface is represented by genes or cells. Each individual gene can take a value of one if used to form the corresponding architecture or zero if it is not. A set of architectures of a certain capability corresponds to a respective population of chromosomes. In addition, the architecture space is explored by working on one capability at a time. This is done through many generations of chromosome populations. The chromosome representation is shown in Figure 16 below

S_1	S_2		S_i			S_m	I_1	I_2		I_k		I_l

Figure 16. Chromosome representation for the system meta-architecture

This representation corresponds to the structure shown in Figure 17 based on the adjacency matrix discussed in the model section.

S_1	S_2		S_i			S_m	I_{12}	I_{13}		$I_{jl'}$		$I_{m-1,m}$

Figure 17. Semantic chromosome representation for the system meta-architecture

Certain methods use about 70 to 80 percent of the population in the mating process. In this research, elitism is applied by selecting 25 percent of the population chromosomes that have the best fitness values to replace a corresponding number of chromosomes from the entire population in each generation. The chromosomes to be replaced are identified at random.

The second step in the process of selection of mating population is using the random roulette method. The individual chromosomes constituting the chromosome population are designated as parents at this point. Parents with higher fitness values are assigned a higher probability of being selected based on their relative fitness as follows:

Using the following formula, the roulette density value ρ_{iab} is calculated for parent a in population b of generation c for capability i where $i = 1, 2, \ldots n;\ a = 1, 2, \ldots h;\ b = 1, 2, \ldots g$ and $c = 1, 2, \ldots e$. The roulette value is found as the ratio of the parent fitness φ_{iabc} to T_{abc} the total fitness of all parents in the population.

- $\rho_{iabc} = \frac{\varphi_{iabc}}{T_{abc}}$

Parents are selected for mating based on cumulative roulette density values in comparison to random numbers. Thus, those with higher roulette density values than a random value end up being selected more frequently, whereas those with lower values have a lower of being selected. Parents with very small roulette density values are not likely to be selected at all. The selected chromosomes become mating parents.

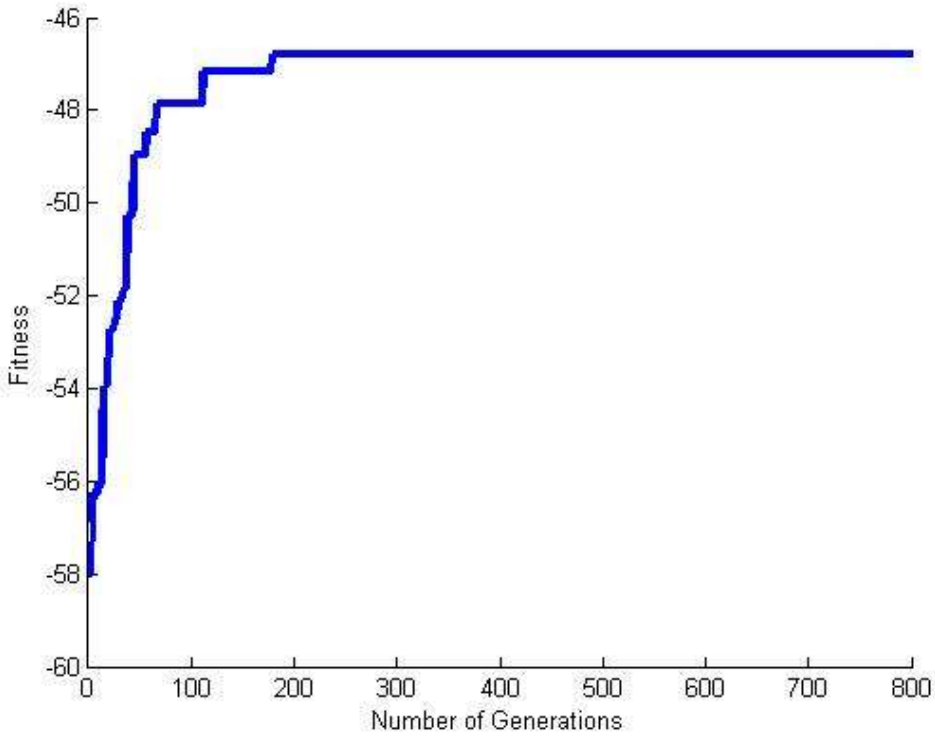
Cross over is used for providing new off-spring. The cross-over takes a random number of genes at random chromosome partitions. In order to ensure not being trapped in local maximum points, mutation is used. It is applied at a very low rate of 0.01 by switching the value of a random gene. That is, the gene value changes from one to zero or from zero to one.

The process is then shifted back to phenotype operations, which starts by checking the technical feasibility of the resulting population architectures. In the process infeasible interfaces are replaced by feasible ones having the highest performance ratio. Then, the fitness is found for the new solutions to help select the new parents for the next generation as described above.

3.4.4 Running of Genetic Algorithms

For genetic algorithms, all systems and interfaces are represented side by side in a chromosome. In the chromosome structure, each system or interface found in a possible architecture is represented by a binary digit, with cooperation taking the value “1” while inability to cooperate taking a “0”.

The model was run multiple times using a population $g = 200$ and various numbers of generations h. The following graph shows an example of genetic algorithm convergence when running with $h = 800$ and the weights used for aggregating the objective functions had equal values $w_p = w_f = w_d = 1$. In addition, for these tests, pseudo-domain parameters have been used.



3.4.5 Initial Testing of Concept for Meta-Architecture Generation and Selection

The resulting fitness output has no indicative value as it provides a weighted summation of the three objective functions with deadline and funding subtracted as the goal is to minimize them, whereas the performance objective function is to be maximized.

The optimal architecture and system data are obtained as described under outputs above. They will be used as inputs to the Agent Based Model (ABM). The ABM is described in section 3.1.

3.4.6 Alternative Optimization Model

As an alternative model rather than separate multiple objectives the criteria are combined in a single ambiguous multi-objective function that depicts the SoS architecture quality.

- The overall SoS architecture quality: Φ

The objective is to find best level of overall SoS architecture quality that can be obtained given ambiguous key performance attributes that are required. This can be obtained through $\Phi(\ )$, which is a fuzzy function that involves the various ambiguous criteria. The fuzzy assessor is defined in a separate section.

Maximize $\Phi_{SoS}(\ )$ …. **Fuzzy assessor**

3.4.6.1 *Inputs and Outputs for Fuzzy Evaluation*

An Excel file with interim architectures has be used for output to the fuzzy assessor and called by the genetic algorithms program. The architectures are similarly constituted of 22 systems and respective interfaces and can take up to 200 architectures. The same file is used for input of fuzzy evaluation of the architectures. The following partial spreadsheet illustrates.

m=	22	systems
Number of Architectures =	200	Architectures (max 200)
Architecture Size=	253	systems and interfaces

	Arch	Systems															
	Quality	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}	S_{12}	S_{13}	S_{14}	S_{15}	S_{16}
Architecture 1	**2.5**	0	0	0	1	1	0	1	1	1	0	1	1	1	1	0	0
Architecture 2	**2.1**	0	1	0	0	1	0	1	0	0	0	1	0	1	1	0	1

Figure 18. Interim Architectures wtih Fuzzy Assessment

3.4.6.2 *Initial Testing of Concept for Meta-Architecture Generation and Selection Using Ambiguous Criteria*

For this phase, a simplified concept has been tested as provided below. However, this concept does not address the constraints at the capability level. Consequently, no initial system data are provided with this model at this time. The intermediate inputs are not used either.

Initial testing was done using 10 pseudo-systems and one objective function employing a fuzzy assessor. Please see the Appendix 7.1 for more details.

3.5 Negotiation Between SoS and System Providing Capability to SoS

3.5.1 Selfish System Model

3.5.1.1 *Model overview*

A model of an individual system *k* is built, which is capable of providing both capabilities to a system of systems (SoS) and interfaces with other individual systems in the SoS. The request for participation is sent from SoS to the individual system, including:

- Requested capabilities, C_i
- Requested interfaces between the system *k* and other individual system *j* on capability *i*, χ_{ij}
- Deadline of delivery, $SoS.d_i$
- Funding for providing the requested capabilities, $SoS.f_i$
- The performance requirement on each of requested capabilities, $SoS.p_i$

The inputs from the SoS listed above are saved in an n by (m+4) matrix shown inFigure 19:

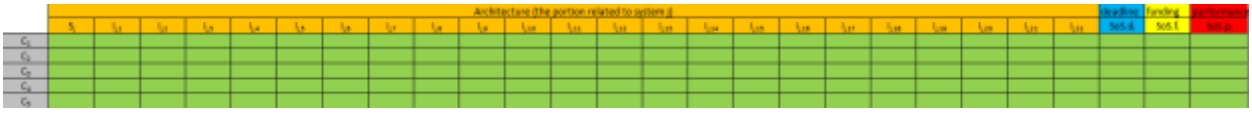


Figure 19. Inputs from SoS to System

This model can be termed a "selfish" model in that the necessary condition for the individual system *k* to collaborate with the SoS is that the incremental profit from the participation is nonnegative. Therefore, a resource allocation problem is formulated to model the decision behavior of the individual system. The optimization problem is solved with considerations of the capabilities, resources and efficiency of the system. Moreover, the market condition is modeled so that the system agent has a rational assessment of the incremental profit provided by the SoS.

The outputs sent from the system k to the SoS include:

- Provided capabilities, C_i
- Provided interfaces between the system *k* and other individual system *j* on capability *i*, χ_{ij}
- Delivery time deviation (additional time needed), $SoS.\Delta d_i$
- Funding deviation (additional fund needed), $SoS.\Delta f_i$
- Performance deviation (under performance is any), $SoS.\Delta p_i$

The outputs listed above are saved in an n by (m+4) matrix shown inFigure 20:

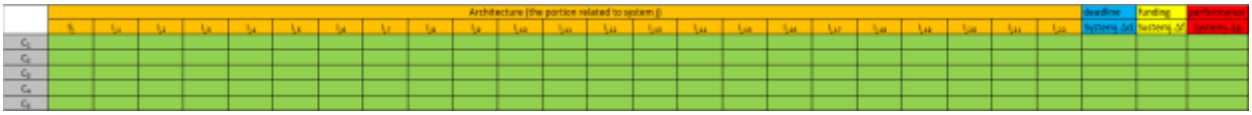


Figure 20. Outputs from System to SoS

Two linear programming (LP) models are built to provide two scenarios of negotiations with the SoS, which are solved using the optimization tool of Matlab. In the first scenario, the SoS is informed possible performance deviation (if any) at the provided funding and time. In the second scenario, the SoS may be provided the capabilities and interfaces as it desires yet is asked to provided additional funding and/or time.

3.5.1.2 *Model setting*

The following is the setting of the individual system *k*.

i: the index of capabilities, and $i = 1, 2, \ldots, n$.

I: The set of capability indices, and $I = \{1, 2, \ldots, n\}$.

j: the index of individual systems, and $j = 1, 2, \ldots, m$.

J: The set of system indices and $J = \{1, 2, \ldots, m\}$.

C_i: binary variable indicating whether the capability *i* is requested, $\forall$ *i.*

I_{req}: the set of requested capability indices; $I_{req} \subseteq I$.

p_i: performance requirement on capability *i*, $\forall\ i \in I_{req}$.

d_i: deadline to deliver capability *i*, $\forall\ i \in I_{req}$.

f_i: funding provided to capability *i*, $\forall\ i \in I_{req}$.

χ_{ji}: binary variable indicating the requested interface between system *j* and *k* ($k \neq j$) in forming capability *i*, $\forall$ *i* and *j.*

n_i: the number of interfaces that system *k* is requested to provide on capability *i*, $\forall\ i \in I_{req}$.

y_i: the system *k*'s throughput of capability *i* in a unit of time, $\forall\ i \in I$.

I_y: the set of indices of capabilities that system *k* is able to build, $I_y = \{i \mid y_i > 0\}$ and $I_y \cap I_{req} \neq \varnothing$.

I_{reqf}: the set of indices of capabilities that is requested by the SoS and the system *k* is able to build, and $I_{reqfsbl} = I_y \cap I_{req}$.

$I_{reqifsbl}$: the set of indices of capabilities that is requested by the SoS but the system *k* is not able to build, and $I_{reqifsbl} = \bar{I}_y \cap I_{req}$.

$Z_{\min}$: the system *k*'s minimum available resource per unit of time.

$Z_{\max}$: the system *k*'s maximum resource per unit of time.

T: planning time horizon. $T = \max\{\text{int}\lceil \sum_{i \in Ireq} p_i/(y_i \times Z_{\min}) \rceil, \max_{i \in Ireq}(d_i)+1\}$. $\text{int}\lceil \sum_{i \in Ireq} p_i/(y_i \times Z_{\min}) \rceil$ is the time needed (in integer) if the system *k* has only the minimum resource and build the requested capabilities in sequence. $[\max_{i \in Ireq}(d_i)]$ is the relaxed upper bound of deadlines. The model assumes the planning time horizon is no shorter than these two.

t: the index of time; $t = 1, 2, \ldots, T$.

Z_t: the system *k*'s resource available at time t, and $Z_{\min} \le Z_t \le Z_{\max}\ \forall\ t$.

Z_{ti}: the resource allocated to build capability *i* at time *t*. $z_{it} \ge 0\ \forall\ i \in I_{reqfsbl}$ and $\forall\ t$. $z_{it\ is}$ are the decision variables.

c_{ci}: the cost of consuming one unit of resource in providing capability *i* at time zero, $\forall\ i \in I_y$.

c_{li}: the cost of consuming one unit of resource in providing interfaces associated with capability *i*, as a percentage of c_{ci}.

g: the inflation rate of unit cost over time.

c'_{ti}: the cost of consuming one unit of resource in providing capability *i* and the associated interfaces at time *t*, for $t = 0, 1, \ldots, d_i$ and $i \in I_y$. $c_{ti} = c_{ci}(1+c_{li}n_i)\exp(g(t\text{-}0))$.

pm: profit margin.

c_{ti}: the market price of one unit of resource for providing capability *i* at time *t*. $c_{ti} = (1+\text{pm})c'_{ti}$.

c''_{ti} (>0): virtual penalty on exceeding the deadlines, for $d_i < t \le T$ and $i \in I_y$.

cp_{ti} $(= c'_{ti} + (c_{ti} - c'_{ti})^+ + c''_{ti})$: the cost of consuming one unit of resource for providing capability *i* and the associated interfaces at time *t*, after the adjustment for the opportunity cost, $\forall\ t$.

w_i: the weight that system *k* put on capability *i*, $\forall\ i \in I_{reqfsbl}$; and $w_i = [f_i/d_i]/\ (\sum_{i \in Ireq} f_i/d_i)$.

3.5.1.3 *Linear Programming (LP) models for decision support*

The first problem (P1) helps identify the best performance of system *k* on the requested capabilities and interfaces with the given funding and deadlines. First, $z_{ti} = 0$ for $i \notin (I_{reqfsbl})$; and $\forall i \in I_{reqfsbl,}\ z_{ti}$'s are determined by the following problem:

(P1)

$$\min \sum_{i \in I_{reqfsbl}} \left(-w_i \sum_{t=1}^{d_i} y_i z_{ti} \right)$$

subject to:

$$\sum_{t=1}^{d_i} y_i z_{ti} \le p_i,\ \forall i \in I_{reqfsbl};$$

$$\sum_{i \in I_{reqfsbl}} z_{ti} \le Z_t, \text{ for } t = 1, 2, \cdots, \max_{i \in I_{reqfsbl}} \{d_i\};$$

$$\sum_{i \in I_{reqfsbl}} \sum_{t=1}^{d_i} c_{ti} z_{ti} \le \sum_{i=1}^{n} f_i;$$

$$z_{ti} \ge 0,\ \forall i \in I_{reqfsbl} \text{ and for } t = 1, 2, \cdots, d_i. \qquad (4.6.1\text{-}1)$$

The objective of (P1) is to minimize the weighted sum of under performances. The first constraint means the performance on a requested capability does not exceed the required performance. The

second constraint means the resource consumed at time *t* should not exceed the resource available at then. The third constraint means the total costs should not exceed the total funding provided. The fourth constraint indicates that the resource relocated to building a requested capability and the associated interfaces is nonnegative.

Denote by $\{z_{ti}^* \mid t = 1, 2, \ldots, d_i, \text{ and } i \in I_{reqfsbl}\}$ an solution to (P1). Let $z_{ti}^*=0$ at $t = 1, 2, \ldots, d_i$ and for $i \notin I_{reqfsbl}$ so that z_{it}^* is defined at any *i*. The minimized performance deviation is calculated by

$$\Delta p_i^* = y_i \sum_{t=1}^{d_i} z_{ti}^* - p_i \,. \tag{4.6.1.-2}$$

The minimized performance deviation is calculated based on the assumptions of no extension of deadlines and/or no additional funding:

$$\Delta d_i^* = 0 \,, \tag{4.6.1-3}$$

$$\Delta f_i^* = 0 \,. \tag{4.6.1-4}$$

The second problem (P2) helps determine the minimum additional funding and/or minimum additional time to meet the goal of forming all the capabilities and interfaces that system *k* is capable of providing. First, $z_{ti} = 0$ for $I \notin I_{reqfsbl}$; and $\forall\, I \in I_{reqfsbl,}$ z_{ti}'s are determined by the following problem:

(P2)

$$\begin{aligned} &\min \left\{ \sum_{i \in I_{reqfsbl}} \sum_{t=1}^{T} cp_{ti} z_{ti} \right\} \\ &\text{subject to:} \\ &y_i \sum_{t=1}^{T} z_{ti} = p_i, \forall i \in I_{reqfsbl} \\ &\sum_{i \in I_{reqfsbl}} z_{ti} \le Z_t, \forall t \\ &z_{ti} \ge 0, \forall t \text{ and } i \in I_{reqfsbl} \end{aligned} \tag{4.6.1-5}$$

The objective of (P2) is to minimize the total costs, including the additional fund used and the virtual penalty on additional time used. The first constraint means the performance on a requested capability is equal to the required performance. The second constraint means the resource consumed at time *t* should not exceed the resource available at then. The third constraint indicates that the resource relocated to building a capability and the associated interfaces is nonnegative.

In (P2), the funding constraint is relaxed and the deadline for finishing any capability *i* is "extended" to *T*. We choose *T* to be

$$T = \max\left\{ \text{int}\left\{ \sum_{i \in I_{reqfsbl}} \left\lceil \frac{p_i}{Z_{\min} y_i} \right\rceil \right\}, \max_{i \in I_{reqfsbl}} (d_i) \right\}, \tag{4.6.1-6}$$

$\text{int}\left\{ \sum_{i \in I_{reqfsbl}} \left\lceil \frac{p_i}{Z_{\min} y_i} \right\rceil \right\}$ is the time needed if building capabilities in a sequential manner and the system *k* has just the minimum resource. The SoS may provide a very long deadline that is not needed by the system k. Considering this possibility, we choose such a value of *T* so that a feasible solution to (P2) is assured.

$\{c''_{ti} | t = d_i + 1, \ldots, T; i \in I_{reqfsbl}\}$ is the virtual penalty on exceeding the deadlines. For any capability *i*, the penalty satisfies

$$c_{(d_i+1)i} \geq \max_{1 \leq t \leq d_i} c_{it}, \tag{4.6.1-7}$$

and

$$c''_{ti} > c''_{si}, \ \forall t > s > d_i . \tag{4.6.1-8}$$

The virtual penalty like such indicates that the cost to build any capability *i* after the desired deadline, d_i, is extremely high and the marginal cost grows with the prolonged time. Consequently, the objective function of (P2) discourages the use of the resource after the deadlines unless system *k* has to. Therefore, the objective function of (P2) effectively penalizes the usages of both the extra funding and time.

Denote by $\{\hat{z}_{ti} | \ t = 1, 2, \ldots, T; i \in I_{reqfsbl}\}$ an solution to (P2). Let $\hat{z}_{ti}$=0 at any time *t* and for $i \notin I_{reqfsbl}$ to make $\hat{z}_{ti}$ be defined at any *i*. Since the system *k* guarantees the performance on the capabilities that it is able to provide, the minimum deviation of performance is either zero or $-p_i$. The minimum additional time needed is found to be

$$\Delta d_i^* = \begin{cases} \hat{d}_i - d_i & i \in I_{reqfsbl} \\ \inf & i \in I_{reqifsbl} \\ 0 & i \in \overline{I}_{req} \end{cases} \tag{4.6.1-9}$$

where

$$\hat{d}_i = \max_{1 \leq t \leq T} \{t | \hat{z}_{ti} > 0\}, \forall \ i \in I_{reqfsbl} . \tag{4.6.1-10}$$

The total minimum additional funding is

$$\Delta f^* = \left(\sum_{i \in I_{reqfsbl}} \sum_{t=1}^{d_i} c_{ti} \hat{z}_{ti} - \sum_{i=1}^{n} f_i \right)^+, \tag{4.6.1-11}$$

which is further split as the additional funding needed for each requested capability:

$$\Delta f_i^* = \lambda_i \Delta f^* \tag{4.6.1-12}$$

Where λ_i is determined by

$$\lambda_i = \begin{cases} \left(\sum_{t=1}^{\hat{d}_i} c_{ti}^* z_{ti} - f_i \right)^+ \Big/ \sum_{i \in I_{reqfsbl}} \left(\sum_{t=1}^{\hat{d}_i} c_{ti}^* z_{ti} - f_i \right)^+ & i \in I_{reqfsbl} \\ \text{Inf} & i \in I_{reqifsbl} \\ 0 & i \in \bar{I}_{req} \end{cases}. \tag{4.6.1-13}$$

3.5.2 Negotiation with the SoS

The solutions to (P1) and (P2) generate the foundation of two negotiation scenarios for the system *k*. We assume these two scenarios occur with equal chance. The system *k* may not share the complete information (e.g., the best performance, the capabilities not capable of providing, cost information) with the SoS during the negotiation for some reasons (e.g., business secret, for better negotiation outcomes).

The first scenario of negotiation is derived from the solution to (P1). The agent of system *k* provides the SoS the performance deviation defined as:

$$System_k.\Delta p_i = \begin{cases} \Delta p_i^* & i \in I_{req} \\ 0 & \text{otherwise} \end{cases}, \tag{4.6.1-14}$$

with the given funding and time:

$$System_k.\Delta d_i = \left[0, 0, \cdots, 0\right]^T, \tag{4.6.1-15}$$

$$System_k.\Delta f_i = \left[0, 0, \cdots, 0\right]^T. \tag{4.6.1-16}$$

Δp_i^* in Eqn. (4.6.1-14) has been defined in (4.6.1-2). In this scenario of negotiation, the system *k* has strong motivation to participate and, therefore, its shows the minimum deviation of performance to SoS, as it is indicated by Eqn. (4.6.1-14). Since the system *k* does not want to let the SoS know what capabilities it is not capable of providing, the system *k* does not update the C_i's and χ_{ji}'s.

The second scenario of negotiation is derived from the solution to (P2). The agent of system *k* provides the SoS the performance deviation defined as:

$$System_k.\Delta p_i = \begin{cases} 0 & i \in I_{reqfsbl} \\ -p_i & \text{otherwise} \end{cases}, \qquad (4.6.1\text{-}17)$$

that is the system k fully meets the performance requirement on the capabilities that it is capable of providing. The system agent adds a random nonnegative number above the minimum additional fund needed in order to not share the private information with the SoS. If the additional fund needed is zero, the agent will still ask for an additional funding equal to a small portion of the provided funding. On those capabilities that the system k is not capable of providing, the agent asks for a very large amount of additional funding equal to Mf_i. Therefore,

$$Systemk.\Delta f_i = \begin{cases} \Delta f_i^*(1+q_i) & i \in I_{reqfsbl} \text{ and if } \Delta f_i^* \neq 0 \\ q_i f_i & i \in I_{reqfsbl} \text{ and if } \Delta f_i^* = 0 \\ Mf_i & i \in I_{reqifsbl} \\ 0 & i \in \vec{I}_{req} \end{cases}. \qquad (4.6.1\text{-}18)$$

The agent adds a random nonnegative integer above the minimum time in order to not share the private information with the SoS. If the system k is not capable of providing a capability, then the agent asks for a very long additional time,

$$\Delta d_i = \begin{cases} \Delta d_i^* + e_i & i \in I_{reqfsbl} \\ Md_i & i \in I_{reqifsbl} \\ 0 & i \in \bar{I}_{req} \end{cases} \qquad (4.6.1\text{-}19)$$

Again, since the system *k* does not want to let the SoS know what capabilities it is not capable of providing, the system *k* does not update the C_i's and χ_{ji}'s.

Technically, we can produce a family of models by modifying the model parameters. For instance, the model will become more "selfish" if we increase the profit margin of the decision model; the performance of the system *k* can be varied by modifying the yield and resource level of the system; and the delivery time and funding can be affected by the cost structure of the system *k*.

3.5.3 Examples

3.5.3.1 *Example 1:*

The input data from the SoS is as follows:

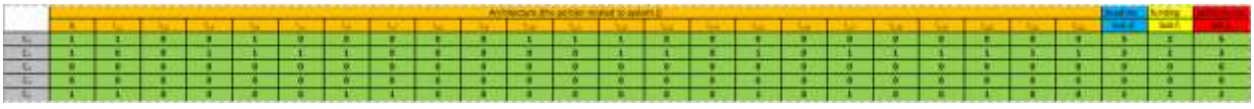


Figure 21. Sos Input Data

The SoS asks the system k to participate by providing capabilities C1, C2, and C5, and associated interfaces shown in the table.

The following output data is associated with the first scenario of negotiation (gives the performance deviation at given finding and deadline). The result shows that the performance on capability C1 meets the requirement. The performance on capability C2 is 3 units below the requirement of 3. That is, the realized performance on capability C2 is 0. Similarly, the realized performance on capability 5 is zero. These performances are associated with no additional funding and/or additional time. The SoS may guess that the system k is not a good candidate for providing capability 2, but he is not sure about it as the system k does not say it explicitly.

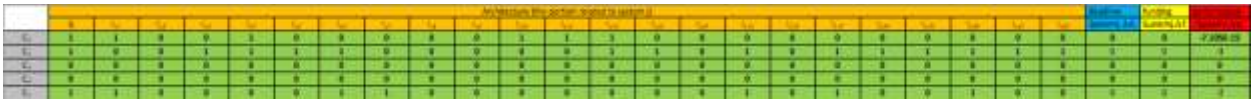


Figure 22. System Output Data 1

The following output data is associated with the second scenario of negotiation (ask for additional funding and/or time on the capabilities that the system k is capable of providing). The result shows that the system k can fully meet the performance requirement except for capability 2. The performance on capability 2 is 3 units below the requirement of 3. That is, the performance on capability 2 is zero. Also, to provide the capabilities, the system k requests extensions of deadline: 10 units on capability 1, 30 unties on capability 2, and 9 units on capability 5. The system k also requests additional funding: 0.46997 on capability 1, 20 on capability 2, and 1.9915 on capability 5. Again, The SoS may guess that the system k is not a good candidate for providing capability 2, but he is not sure about it as the system k does not say it explicitly.

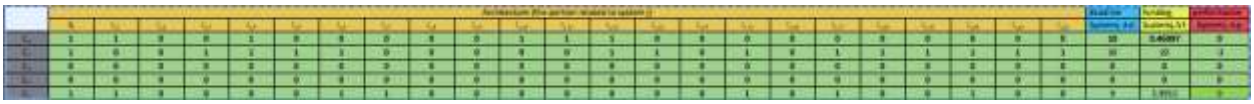


Figure 23. Output Data 2

Example 2:

The input data from the SoS is associated with the second scenario of negotiation (ask for additional funding and/or time on the capabilities that the system k is capable of providing).

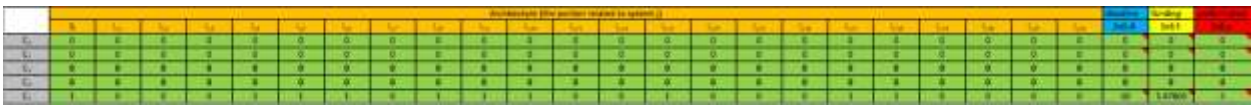


Figure 24. SoS Input Data, Ex. 2

The SoS only asks the system k to participate by providing capability C5. A very long deadline is given, but the provided funding is limited.

The following output data is associated with the first scenario of negotiation (gives the performance deviation at given finding and deadline). The performance deviation is -4.4945 on the capability 5 at the given funding and deadline.

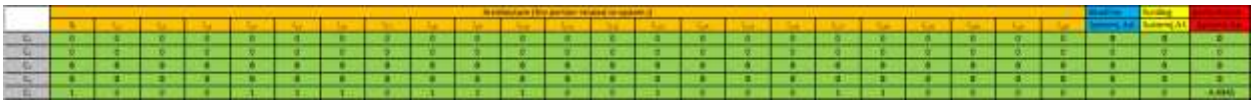


Figure 25. Output Data, round 1 of negotiation

The following output data is associated with the second scenario of negotiation (ask for additional funding and/or time on the capabilities that the system k is capable of providing). The system k can meet the performance requirement on the capability C5 but requests an extension of deadline for one unit of time and an additional funding of 16.341 units.

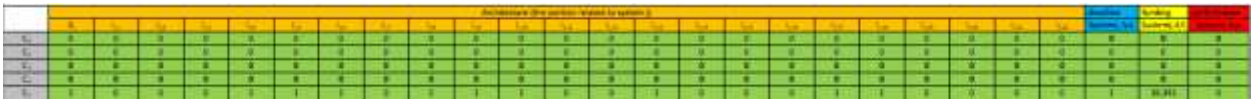


Figure 26. Output Data, round 2 of negotiation

3.5.4 Opportunistic - Markov Chain Model

The model presented here for the system in question is an "opportunistic" model, i.e., that the system can behave selfishly as well as unselfishly (or selflessly). In other words, by tweaking a certain tunable parameter, η, an entire spectrum of behavior – ranging from extremely selfless to extremely selfish – can be obtained from the system. Hence in a sense the system's behavior can be characterized as "opportunistic" because there is no fixed pattern of behavior that this system will exhibit. It needs to be understood that how the system behaves can be controlled by varying the tunable parameter (η), and thus it is possible to have a large number of systems, using differing values of η, ranging from very selfish to extremely selfless. An example is provided at the end to illustrate the interaction process between the SoS and the system concerned.

The system will interact with the SoS as follows. When the SoS provides via a data structure the following information: an architecture, a desired performance level, the level of funding, and the deadline, the system will perform internal calculations to develop outputs for the following:

- Willingness of the system to cooperate with the SoS
- The performance level that the system can deliver
- The funding it will need
- The deadline by which it will be able to complete its task

A project management model based on Markov chains will be used for estimating the above mentioned outputs. The mathematical details of this model are now below. The opportunistic behavior of the system will be dependent on an adjustable *opportunistic parameter η*. The SoS will have the ability to

change this parameter in order to develop a class of systems with differing behavior. Further, the sluggishness of the system will be defined by additional parameters (*l* and *m*), which also the SoS will have the ability of adjusting.

3.5.4.1 *Methodology*

Essentially, the work assigned by the SoS to the system will be assumed to be a ``project'' that takes a random amount of time and a random amount of resources (funding) to complete. The internal phases of this project will be modeled as a Markov chain, and the Markov chain will be to estimate the expected (mean/average) amount of time and funds needed by the system. The willingness to cooperate will be based on how quickly the system will be able to complete the task (project) at hand and its own workload. In what follows, we provide the details of how these calculations will be performed to generate outputs to be supplied to the SoS.

The three internal phases in the project will be: "initial," "intermediate," and "completion." Each of these will be considered to be states in an absorbing Markov chain in which the absorbing state is completion. We will assume that after unit time (e.g., one cycle), the project will move from one state to the next with a given probability. The input chromosome will be converted into a matrix of 0s and 1s, which will essentially represent the nature of interaction and the workload imposed on the system under consideration. These inputs will be used to compute the number of systems with which the current system must interact. This number will be used to compute the probabilities of the Markov chain. From the one-step transition probabilities of the Markov chain, it is possible to estimate the mean amount of time needed for completion of the project. If the resources are assigned according to the worst-case scenario, the mean time of the project completion will be used to estimate the level of cooperation.

Notation:

- n: number of systems in SoS
- i: index of the system under consideration, where $i = 1,2, \ldots n$
- p_i: *p*erformance desired from system *i* by SoS
- C: system architecture (interface): chromosome (matrix of 0s and 1s)
- d_i: delivery deadline set for system *i* by SoS
- f_i: funds allocated to system *i* by SoS; values for $i = 1,2, \ldots n$ will be provided to the *i*th system
- Δp_i: the difference in the performance that system *i* expects to deliver
- Δf_i: the difference in the funding that system *i* requires
- Δd_i: the difference in the deadline by which system *i* expects to deliver
- η: selfishness parameter in the interval (0,1] that SoS can adjust to obtain a spectrum of participating systems ranging from very unselfish to very selfish; 0 being very unselfish and 1 being very selfish

- *l* and *m*: additional system behavior parameters, each taking values in the interval (1,2], which the SoS can change to obtain a whole spectrum of participating systems ranging from very fast to very sluggish; 1 being very fast and 2 being very sluggish.

We will need some additional notation required in the internal calculations of the model that we will define later. The Markov chain of the internal system dynamics will be defined by the following transition probability matrix:

$$\begin{bmatrix} \frac{k}{ln} & 1-\frac{k}{ln} & 0 \\ 0 & \frac{k}{mn} & 1-\frac{k}{mn} \\ 0 & 0 & 1 \end{bmatrix}$$

where state 1 is the initial phase, state 2 the intermediate phase, and state 3 the completion phases. In the above, *k* will denote the number of systems with which system *i* will interact. The parameters *l* and *m* will be additional system behavior parameters that the SoS will be able to change at will. The value of *k* will be computed from the chromosome C. The, using standard analysis of an absorbing Markov chain (Ross, 2003), the number of expected cycles needed to complete the project starting at the initial phase will be computed. This number will be denoted as τ_i for the *i*th system. We now explain how the absorbing Markov chain analysis is performed. We construct the so-called Q-matrix from the transition probability matrix as follows eliminating the completion phase:

$$Q = \begin{bmatrix} \frac{k}{ln} & 1-\frac{k}{ln} \\ 0 & \frac{k}{mn} \end{bmatrix}$$

Then, the following operation is performed:

$$M = I - Q$$

where *I* denotes the identity matrix. Let *N* denote the inverse of *M*. Then, the expected number of cycles to completion are computed as follows:

$$T = N \cdot C$$

where T and C are column vectors. C is a column vector of ones, and T(1), the first element in the column vector, will equal τ_i. Setting *k = n* and repeating the calculations above, we will obtain an upper limit on the expected number of cycles needed for completion; this limit will be denoted by τ_{max}. The willingness to cooperate will then be computed as:

$$\Phi = 1 - \frac{\tau_i}{\tau_{max}}$$

Then, the actual performance of the system will be given as

$$p_i \, \Phi$$

which would imply that the change in performance will be given by:

$$\Delta p_i = p_i(\Phi - 1)$$

Further, the deadline, i.e., the number of cycles, by which the system will be able to deliver will be computed as

$$\frac{d_i}{\Phi}$$

The above scalar will be rounded up to the next higher integer. The above implies that the requested change in the deadline will be:

$$\Delta d_i = d_i(\frac{1}{\Phi} - 1)$$

And finally, we now show how the requested change in the budget will be computed. We use $\bar{B}$ to denote the upper limit on the budget, which will effectively serve as its own estimate of the upper limit on the total amount of funds that SoS can potentially allocate to all systems combined. This upper limit will be computed as:

$$\bar{B} = \sum_{i=1}^{n} f_i$$

Then, the maximum rate at which the SoS can fund projects will be computed as:

$$\rho = \frac{\bar{B}}{\tau_{max}}$$

Then, the actual funding needed by the system will be computed as

$$\eta \tau_i \rho$$

where η is the *opportunistic* parameter. For a very selfish system (η =1), the funds needed will be $\tau_i\rho$, i.e., the actual amount needed, while an unselfish system (whose selfishness is η <1) will need $\eta\tau_i\rho$ amount of funding. This would imply that the change requested in the funding will be:

$$\Delta f_i = \eta \tau_i \rho - f_i = \eta \tau_i \frac{\bar{B}}{\tau_{max}} - f_i$$

Setting η to a very small positive number close to 0 produces a system that is very selfless, while setting it to 1 produces a very selfish system. Thus, a large number of systems can be generated during the negotiation process by assigning a range of values to η, e.g., 0.01, 0.02, 0.03, 0.1, 0.3, 0.5, 0.6, 0.7, 0.8, 0.9, 0.99, 0.999. Also, by varying the values of *l* and *m* in the range (1,2], one can obtain a range of systems whose ability to meet deadlines and deliver performance levels varies from very fast (e.g., 1.01) to very sluggish (2). An example is now provided in regards to how the interaction occurs between the SoS and the system. The following the input provided to the system via an EXCEL file. Note that there are five different sets of capabilities provided to the system, one capability in each row. In all there are 22 systems.

Architecture (the portion related to system j)																						Deadline	funding	performance
S_j	$l_{j,1}$	$l_{j,2}$	$l_{j,3}$	$l_{j,4}$	$l_{j,5}$	$l_{j,6}$	$l_{j,7}$	$l_{j,8}$	$l_{j,9}$	$l_{j,10}$	$l_{j,11}$	$l_{j,12}$	$l_{j,13}$	$l_{j,14}$	$l_{j,15}$	$l_{j,16}$	$l_{j,17}$	$l_{j,18}$	$l_{j,19}$	$l_{j,20}$	$l_{j,21}$	$l_{j,22}$	$Systemj.\Delta d_i$	$Sustemj.\Delta f_i$	$Systemj.\Delta p_i$
1	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**			
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
1	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**1**	**1**	**1**			
1	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**0**			
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			

Figure 27. Capability Inputs

The system will read the relevant inputs and will generate the following outputs from its calculations in the Matlab code:

deadline	funding	performance
$Systemj.\Delta d_i$	$Sustemj.\Delta f_i$	$Systemj.\Delta p_i$
6	**-0.307692**	**-2.82051282**
0	1.5	0
7	**0.1290323**	**-2.12903226**
3	**-0.263158**	**-1.15789474**
0	1.5	0

Figure 28. System Deadlines, Funding & Performance Outputs

In Figure 28 above, η is set to 0.5, and both *l* and *m* are set to 2. The above implies that the system will need no change in the deadline or performance for capabilities 2 and 5 but it will require an increase in

funding by 1.5 units; for capability 1, 3 and 4, the changes in deadlines will be 6, 7 and 3 respectively. For the capability 1, 3, and 4, the system performance will be reduced by -2.82, -2.12, and -1.15 respectively, while funding needed will also be reduced by -0.31 (capability 1), increased by 0.13 (for capability 3) and reduced by -0.26 (capability 4). However, in the negotiation process, the values of η, l and m can be changed at will to obtain a large number of systems, and thereby the SoS has the ability to select a suitable system that is sufficiently selfless and fast in its actions. Now if the parameters l and m are changed to 1.1, i.e., the system is much faster (and we use η = 0.5 as before), we have the following response:

deadline	funding	performance
Systemj.Δd_i	Sustemj.Δf_i	Systemj.Δp_i
1	**-1.65625**	**-0.57291667**
0	0.2727273	0
1	**-1.410714**	**-0.58928571**
0	**-1.637363**	**-0.24175824**
0	0.2727273	0

Figure 29. Better System Deadlines, Funding & Performance

Figure 29 above implies that no changes are required for the deadlines for capabilities 2, 4, and 5 (note that previously this was the case only for capabilities 2 and 5). Further, the changes in deadlines required are significantly reduced in their magnitudes. Overall, we also see reduced needs for funding and reduced declines in expected performance. Finally, now if improve the selfishness parameter to η = 0.1 and keep both *l* and *m* at 1.1, we have the following output:

deadline	funding	performance
Systemj.Δd_i	Sustemj.Δf_i	Systemj.Δp_i
1	**-1.93125**	**-0.57291667**
0	0.0545455	0
1	**-1.882143**	**-0.58928571**
0	**-1.927473**	**-0.24175824**
0	0.0545455	0

Figure 30. Selfish Ouptuts from System

Figure 30 above shows the effect of the selfishness parameter. It reduces, as expected, the needs for funding for each capability. Further note that the performance levels and the deadlines are not changed, since the selfishness parameter does not have an impact on those characteristics. The

performance levels and the deadlines are dependent on the parameters *l* and *m* that dictate how quickly the system can respond. Cooperative Model

3.5.4.2 *Input of SoS*

At first, the individual system will receive the requirements from SoS. The target of SoS is to obtain n capabilities by m systems and some connections between these systems. The input information from SoS will include deadline, funding and performance demands for each capability, and those interface need to be construct by the *i*th system with others. Then if SoS has m systems and needs n capabilities, the inputs is an n by m+4 matrix as following.

Table 3. SoS Output format

		Interface between Sj and Si				deadline	funding	performance
	Sj	Ij,1	Ij,2	….	Ij,m	SoS.di	SoS.fi	SoS.pi
C1	1	0	1	…	1	1	1	5
C2	0	0	0	…	0	0	0	0
…	…	…	…	…	…			
Cn	1	1	0	…	0	3	2	2

3.5.4.3 *Basic supposes for the individual systems decision making*

We suppose the individual systems will make decisions based on the following considerations:

1. The establishment of each demanded capability is an independent task for the individual system
2. Building the interfaces with different other systems has different technique difficulty coefficients and different costs. And the total difficulty coefficient and cost are depended on the sum of difficulty coefficients and different costs for all demanded interface
3. Each task for a demanded capability can be seen as an input-out system and can be described by a production function as : performance = *f*(deadline, funding) with coefficients of technique difficulty and cost
4. Individual system will makes decision by the three rules:

a. If the individual system can reach the demand of performance by the given deadline and funding, then the system will accept the task without require adjustments
b. If the coefficients of technique difficulty and cost for building some interface in the task are infinite, then the system will refuse the task
c. If the task is feasible that the individual system can reach an acceptably adjusted performance with acceptably adjusted deadline and deadline, and then the system will accept the task with adjustments, otherwise the system will refuse the task.

3.5.4.4 *The steps of the decision making of individual systems*

Step 1. Receive the input data from SoS

Step 2. Initialization:

a. Given weights for adjustments of deadline, funding and performance depends on the important of them. The larger the weight is, the more flexible the adjustment of the requirement is.

b. Given the up bounds of adjustments

b. Given a coefficient matrix for the individual system *j*. An element in the matrix measures the difficulty and cost of the interface between system j and i for a capability. (For example, the element C_{23} in the matrix means the difficulty and cost coefficient of the interface between the studied individual systems with system 3 for capability 2 is $C_{23.}$

c. Suppose that the system with deadline and funding as input, and performance as output can be described with Cobb-Douglas production function. The function is as following: $\text{performance} = \text{A} * \text{deadline}^{\alpha} * \text{funding}^{\beta}$

Step 3. Make decision based on multi objective optimization for each task

Agent system will face the optimization problem as following for each capability task

- $Min\ (\Delta\ deadline, \Delta\ funding, \Delta\ performance)^T$

$$subject\ to\ performance \geq f(deadline + \Delta\ deadline, funding + \Delta\ funding, performance - \Delta\ performance)$$

Then we will transform the problem to a single objective optimization problem based on following idea: we can see the (deadline, funding, performance) given by SoS as an ideal point in the 3 dimensions space, and we just need to move the point into the feasible domain to satisfy the restricted condition that performance >= *f*(deadline, funding).

So the problem can be transformed to

$$Min\ w_1 * (\Delta\ deadline)^2 + w_2 * (\Delta\ funding)^2 + w_3 * (\Delta\ performance)^2$$

$$subject\ to\ performance \geq f(deadline + \Delta\ deadline, funding + \Delta\ funding, performance - \Delta\ performance)$$

Step 4. Output

If the solution set of the optimization problem is empty or the optimum adjustments over the up bounds of adjustments, the individual system will reject the task of the capability. Otherwise the individual system will sent the adjustments to SoS system.

4 ABM Integration Framework and Sample Problem

There were two challenges to the Agent-Based Model: Generality and Integration. Generality meant creating and ABM that would be applicable and adaptable to any domain. Integration meant interfacing the ABM with other models developed in Matlab by other independent researchers.

4.1 Generic Agent-Based Model

As mentioned in (Dahmann, Rebovich, Lane, Lowry, & Baldwin, 2011), the SoS development follows a Wave Model. The ABM was built based on this Wave Model and the ABM reflects the behaviors inherent in an Acknowledged SoS (Director Systems and Software Engineering, OUSD (AT&L), 2008). This research showed that the domain specific area in an Acknowledged SoS development was in the evaluation of an SoS architecture. This SoS architecture evaluation is done in the Fuzzy Assessor and the domain specific part of the Fuzzy Assessor are the Fuzzy Rules. In order to make the ABM generic to any domain, the Fuzzy Assessor was implemented in the SoS agent using a Fuzzy Associative Memory (FAM). The SoS agent reads in the FAM from an Excel file during initialization. In this way, a user can specify the domain specific Fuzzy Rules in an Excel file and the ABM can simulate the SoS development for that domain. Figure 31 shows the Fuzzy Assessor that is in the SoS agent.

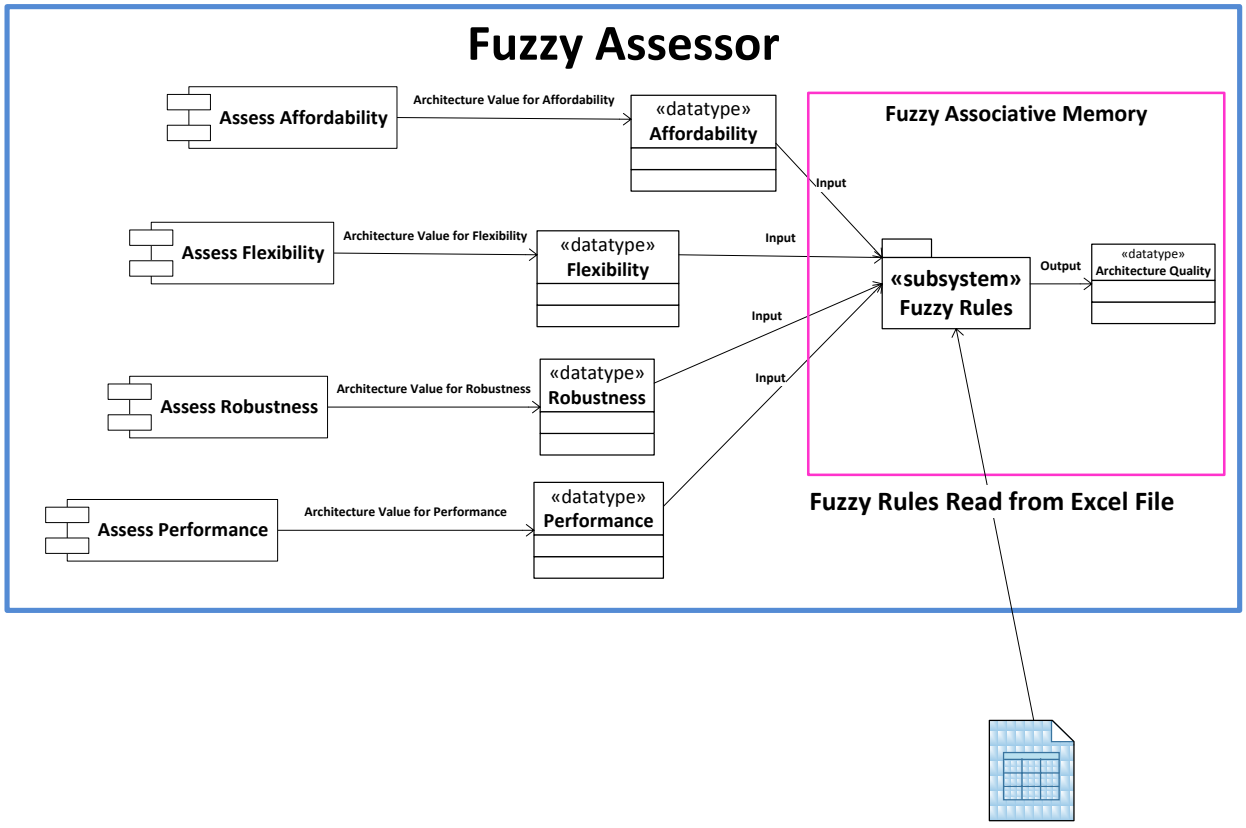


Figure 31. Fuzzy Assessor for the SoS Agent for Assessment of SoS Architecture

4.2 ABM Integration

Multiple models were developed in Matlab independently by different researchers. The second challenge was how to integrate these models with the ABM. The models developed in Matlab were:

- Genetic Algorithm (GA)
- Selfish System Negotiation Model
- Opportunistic System Negotiation Model
- Cooperative System Negotiation Model

In order to allow each researcher to work independently, the decision was made that the interface between these models and the ABM would be through Excel files. Matlab was to used to create executable files for each of the Matlab developed models. The ABM calls the Matlab executable files.

At the beginning of the epoch, the ABM calls the GA Matlab executable file to get the initial SoS architecture which is placed in an SoS architecture Excel file by the GA Matlab executable. The GA Matlab executable also writes the initial performance, funding, and deadline for each system to an Excel file. In order to prevent possible file access contentions, there is one Excel file for each system in the simulation that contains the performance, funding, and deadline for that system.

After the SoS reads the initial SoS architecture from the SoS architecture Excel file, the SoS then sends the Request for Connectivity message to each system in the simulation. There is one instantiation of the system agent for each system in the simulation. Once the system has received the Request for Connectivity message from the SoS, the system calls the Matlab executable file that contains the system negotiation model assigned to that system. For this research phase, the system 1 calls the Opportunistic System Negotiation Matlab executable file, system 2 calls the Selfish System Negotiation Matlab executable file, and system 3 calls the Cooperative System Negotiation Matlab executable file. The name of the Excel file is passed to the Matlab executable when the Matlab executable is called. Figure 32 shows the integration.

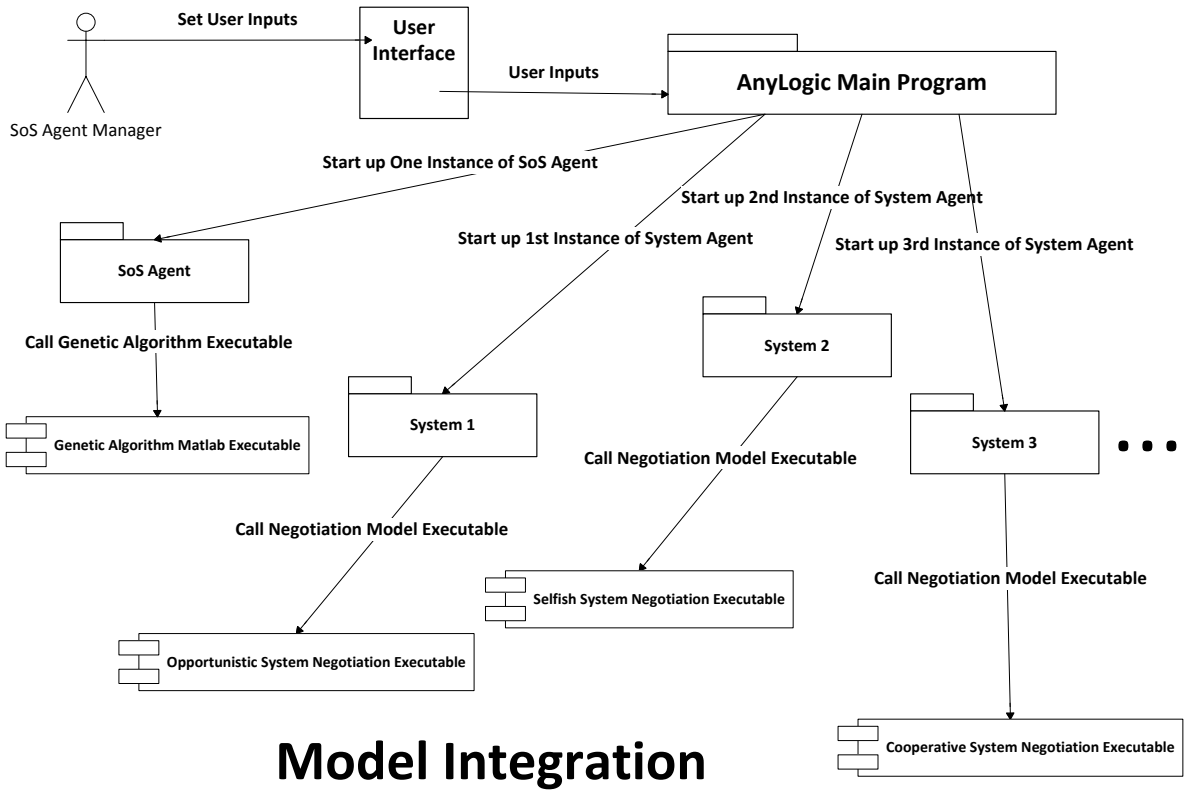


Figure 32. Overall Agent-Based Implemented Model Architecture

5 Concluding Remarks and Future Work

In the second phase of the project the team will continue with the development of evolutionary strategies based multi-objective mathematical model for creating initial SoS meta architecture to start the negotiation at each wave and basic structure of the math model that will be used in creating the fuzzy assessor for evaluating given meta SoS architectures and domain depended parameters that will be used in the system of systems analysis and architecting through Agent Based Model together with the national priorities, funding and threat assessment being provided by the environment developed during until end of December 2013.

6 Bibliography

Acheson, P. (2010). Methodology for Object Oriented System Architecture Development. *IEEE Systems Conference.*

Acheson, P., Dagli, C., & Kilicay-Ergin, N. (2013, March). Fuzzy Decision Analysis in Negotiation between the System of Systems Agent and the System Agent in an Agent Based Model. *International Journal of Soft Computing and Software Engineering [JSCSE]*.

Acheson, P., Dagli, C., & Kilicay-Ergin, N. (2013). Model Based Systems Engineering for System of Systems Using Agent Based Modeling. *Conference on Systems Engineering Research.* Atlanta GA.

Acheson, P., Pape, L., Dagli, C., Kilicay-Ergin, N., Columbi, J., & Harris, K. (2012). Understanding System of Systems Development Using an Agent-Based Wave Model. *Proceedings of the Complex Adaptive Systems Conference.* Washington, DC.

Alberts, D. S., Garstka, J. J., & Stein, F. P. (1999). *Network Centric Warfare: Developing and Leveraging Information Superiority, 2nd Edition.* Washington DC: C4ISR Cooperative Research Program.

ASD(NII), D. (2009). *DoD Architecture Framework Version 2.0 (DoDAF V2.0).* Washington DC: Department of Defense.

Bonabeau, E. (2002). Agent based Modeling: Methods and Techniques for Simulating Human Systems,. *Proceedings of the National Academy of Sciences, Vol. 99*, 7280-7287.

Brazier, F., C.M., J., & Truer, J. (1977). Formalization of a cooperation model based on joint intentions. In M. W. J.P. Muller, *Intelligent Agents III (Proc. Of the Third International Workshop on Agent Theories, Architectures and Languages, ATAL'96), Lecture Notes in AI, Vol. 1193* (pp. 141-155). Springer Verlag.

Brazier, F., Dunin-Keplicz, B., Jennings, N. R., & and Treur, J. (1966). DESIRE: modeling multi-agent systems in a compositional framework. *International Journal of Cooperative Information Systems, M Huhns, M Singh (eds), special issue of Formal Methods in Cooperative Information Systems*.

Coello, C. A. (2004). AN INTRODUCTION TO MULTI-OBJECTIVE EVOLUTIONARY ALGORITHMS AND THEIR APPLICATIONS. In C. A. Coello, *Applications of Multi-Objective Evolutionary Algorithms; Advances in Natural Computation — Vol. 1* (pp. 1-28). Singapore: World Scientific Publishing Co.

Dagli, C. (. (2011). *Complex Adaptive Systems, Procedia Computer Science Volume 6.* Elsevier.

Dahmann, J., Rebovich, G., Lane, J. A., Lowry, R., & Baldwin, K. (2011). An Implementers' View of Systems Engineering for Systems of Systems. *Proceedings of IEEE International Systems Conference.* Montreal.

Dauby, j. P., & Dagli, C. H. (2011). The canonical decomposition fuzzy comparative methodology for assessing architectures. *IEEE Systems Journal, vol. 5, no. 2*, 244-255.

Dauby, J. P., & Upholzer, S. (2011). Exploring Behavioral Dynamics in Systems of Systems. In C. (. Dagli, *Complex Adaptive Systems, Procedia Computer Science, vol 6* (pp. 34-39). Elsevier.

Department of the Navy. (1997). *Contractor Performance Assessment Reporting System (CPARS).* Washington DC.

Director Systems and Software Engineering, OUSD (AT&L). (2008). *Systems Engineering Guide for Systems of Systems.* available from http://www.acq.osd.mil/se/docs/SE-Guide-for-SoS.pdf.

Dombkins, D. (2007). *Complex Project Management.* South Carolina: Booksurge Publishing.

Dudenhoeffer, D., & Jones, M. (2000). A Formation Behavior for Large Scale Micro-Robot Force Deployment. *Proceedings of the 32nd Conference on Winter Simulation.*

Fogel, D. (2006). *Evolutionary Computation.* New Jersey: John Wiley and Sons.

Haris, K., & Dagli, C. (2011). Architecture Trade-off Analysis and Reconfiguration. *Proceedings Conference on Systems Engineering Research.* ISBN -978-0-9814980-1-0.

Holland, J. (1973). Genetic Algorithm and the Optimal Allocation of Trials. *SIAM Journal on Computing, vol 2, June 1973*, 88-105.

Iba, H. a. (2012). *New Frontier in Evolutionary Algorithms.* London: Imperial College Press.

Kilicay Ergin, N., & Dagli, C. H. (2008). In M. (. Jamshidi, *System of Systems: Innovations for the 21st Century.* Wiley & Sons, Inc.

Kilicay-Ergin, N., Enke, D., & Dagli, C. (2012). Biased trader model and analysis of financial market dynamics. *International Journal of Knowledge-based Intelligent Engineering Systems, Vol. 16*, 99-116.

Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review, 63*(2), 81-97.

NDIA. (11 October 2011). *Best Practices Model for SoS Systems Engineering (SE) and Test & Evaluation (T&E), Draft for NDIA Strategic Initiative: Best Practices Model for SoS T&E.*

Pedrycz, W., Ekel, P., & Parreiras, R. (2011). *Fuzzy Multicriteria Decision Making; Models, Methods and Applications.* West Sussex: John Wiley & Sons.

Rao, S. S. (2009). *Engineering Optimization Theory and Practice.* Hoboken, NJ: John Wiley & Sons.

Ravindran, A., & Ragsdel, K. a. (2006). *Engineering Optimization Methods and Applications.* Hoboken, NJ: John Wiley & Sons.

Rosenau, W. (1991). *Coalition Scud Hunting in Iraq, 1991.* RAND Corporation.

Ross, S. (2003). *Introduction to Probability Models, 8th Edition.* Academic Press.

Sumathi, S., & Surekha, P. (2010). *Computational Intelligence Paradigms: Theory & Applications Using MATLAB.* Boca Raton FL: CRC Press.

Taha, H. A. (2008). *Operations Research - An Introduction, 8th Edition.* Upper Saddle River, NJ: Pearson Prentice Hall.

Thompson, M. (2002, December 23). Iraq: The Great Scud Hunt. *Time Magazine*, pp. http://www.time.com/time/magazine/article/0,9171,1003916,00.html.

Weiss, W. (2008). Dynamic Security: An Agent-based Model for Airport Defense. *Proceedings of the Winter Simulation Conference.*

Zitzler, E., Laumanns, M., & and Bleuler, S. (2004). A Tutorial on Evolutionary Multiobjective Optimization. *Metaheuristics for Multiobjective Optimisation.* Springer.

7 Appendix

7.1 Optimization

7.1.1 Initial Testing of Concept for Meta-Architecture Generation and Selection Using Ambiguous Criteria

A set of initial architectures is generated at random for the genetic algorithm (GA). Out of a demonstration universe of ten systems, a portion is selected for participating in the SoS architecture with a probability of 0.7. In addition, out of 45 possible interfaces between the ten systems, the probability of an interface being selected is 0.8.

Then, an interface verification process is used to ensure that the selected interfaces are feasible. In this implementation, an interface that attempts to connect between two systems of which at least one is a non-contributing system is rejected. This is true for all interfaces. Thus, the interface random generation process is repeated until a feasible solution is reached.

The result is a population of chromosomes inhabiting the meta-architecture space. Each chromosome represents an alternative SoS architecture. This is done by assuming a vector with binary numbers, where a one stands for a contributing system or interface and a zero for the opposite. The collection of all contributing and non-contributing systems and possible interfaces form a chromosome. Figure 33 illustrates a chromosome representation. The chromosome structure incorporates a number of genes, where a gene S_i refers to a system and I_{ij} to an interface between any two systems i and j. S_i , I_{ij} can take the value of either one or zero.

S_1	S_2		S_i			S_n	I_{12}	I_{13}		I_{ij}		$I_{9,10}$

Figure 33. Chromosome representation for the ten system meta-architecture

In this research, the meta-architecture offers a space that is spanned by all possible SoS architecture configurations with specific information about the set of architectures under consideration. This information is released as outputs for assessment and tradeoff analysis.

Architecture assessment takes place in the fuzzy assessor as explained above, whereas tradeoffs and negotiations among System Agents and with the SoS Agent take place in the agent-based model. Each proposed configuration of a SoS architecture is constituted of ten systems and the underlying interfaces are evaluated and negotiated.

The results of the above processes give a score for each of the systems and interfaces under consideration. These scores are provided in the form of crisp numerical values. Together they bring about a vector with scores covering the set of SoS architectures.

From the meta-architecture perspective, the score vector feeds back an input for rating of the architectures. This also represents chromosome fitness values for the analyzed population. The fitness values would then be used as inputs in the genetic algorithm to determine the next generation of chromosomes.

7.1.2 Discussion of Results of Initial Concept Testing

Matlab was used to code and run the genetic algorithm. The initial population was generated at random with a probability of 0.7 for systems to contribute and a probability of 0.8 for the interfaces to connect. Then, all infeasible interfaces were removed such that connected interfaces remain only when the corresponding systems are contributing. Table 1 illustrates an example of an initial population consisting of only ten chromosomes.

Table 4. Example of a population of ten feasible chromosomes

No	Chromosome Genes	Contributing Systems	Connected Interfaces
1	0 1 1 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 1 0 1 0 0 0 1 1 0 0 0 0 0 0 1 0 0 1 1 0 0 1 1 0 0 0 0 0 0	0.6	0.27
2	0 0 1 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 1 1 1 0 1 1 1 1 0 1 1 0 0 1 0 0 0 0 1 0	0.7	0.33
3	1 0 0 1 0 1 1 0 1 1 0 0 1 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 0 0 1 0 0 1 0 1 1 0 0 1	0.6	0.29
4	0 1 0 1 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 1 1 0 1 1 0 1 0 0	0.6	0.29
5	1 0 1 1 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0	0.5	0.22
6	1 1 0 0 1 1 0 1 1 1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0 1 1 1	0.7	0.40
7	1 1 0 1 1 0 1 1 0 1 0 0 0 0 0 1 1 0 1 0 1 1 0 1 1 0 1 0 0 0 0 0 0 0 1 0 1 1 0 1 0 0 1 0 1 0 1 0 0 0 0 1 0 1 0 1 0	0.7	0.38
8	1 0 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0 0 1 1 0 1 1 0 1 0 1 0	0.8	0.60
9	0 0 1 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 0 1 1 1	0.6	0.27
10	0 0 1 1 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1	0.7	0.38

Table 4 also shows the potion of contributing systems in each chromosome and the corresponding feasible connections. Taking the averages provides a system contribution factor of 0.65 and connectivity of 0.34. It is evident from the results that the connectivity is much reduced when only the feasible chromosomes are maintained. This compares to initial chromosome interfaces of 0.8, which include the infeasible ones.

Figure 34 illustrates the conversion of the GA using pseudo-QFD matrix and membership functions, and domain fuzzy rules. This was based on a population of 30 and on 200 generations. The conversion occurred after about 130 generations after which the chromosome that corresponds to the highest fitness value was selected.

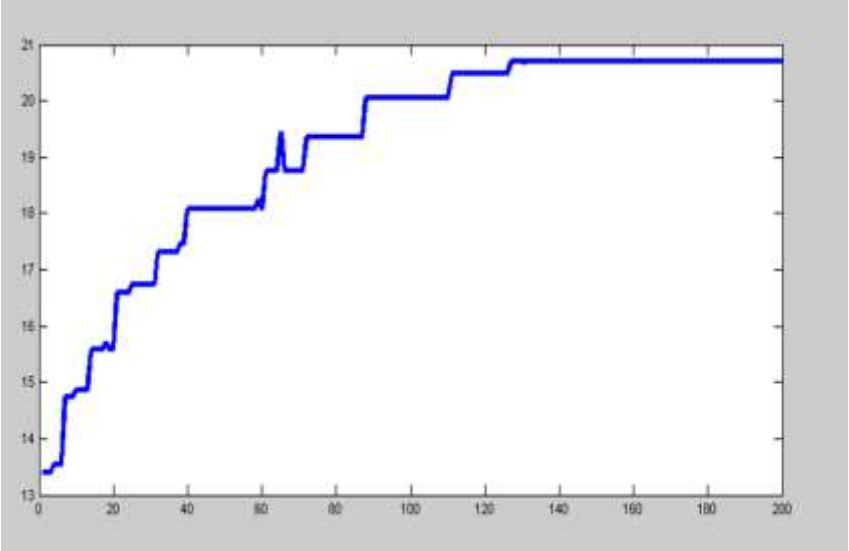


Figure 34. Model conversion using Matlab

The selected chromosome is shown in Table 5. This chromosome represents the recommended SoS architecture. The colored cells contain genes with values of one, which correspond to systems that contribute to the SoS, whereas the bordered cells contain genes, which correspond to interfaces between systems with mutual connection. It can be noticed that the chromosome can be considered a sparse vector. This has happened in spite of the higher probabilities of the initial population as described above. Similarly, the portion of contributing systems is only 0.4, and the ratio of interfaces to the total possible interfaces is only 0.1, whereas the probabilities used to generate the initial population were 0.7 and 0.8 respectively, with feasible interfaces of 0.34.

Table 5. Chromosome for recommended system of systems architecture

Gene No	Chromosome Gene Values										
1-11	1	1	0	0	0	1	1	0	0	0	0
12-22	0	0	0	1	0	0	0	0	0	0	0
23-33	1	1	0	0	0	0	0	0	0	0	0
34-44	0	1	0	0	0	0	0	0	0	0	0
45-55	0	0	0	0	0	0	0	0	0	0	0

The corresponding best SoS architecture is shown in Figure 35. The color coding of the architecture systems in Figure 35corresponds to that of the chromosome genes in table 2. On the other hand, the system connectors or interfaces in the architecture in figure 5 correspond to the chromosome genes shown in the bordered cells in Table 5. From the figure, it can be readily seen that systems one, two, six and seven contribute to the architecture. In addition, system 1 interfaces with system 6, system 2 also interfaces with system 6 and interfaces with system 7, and system 6 interfaces with system 7.

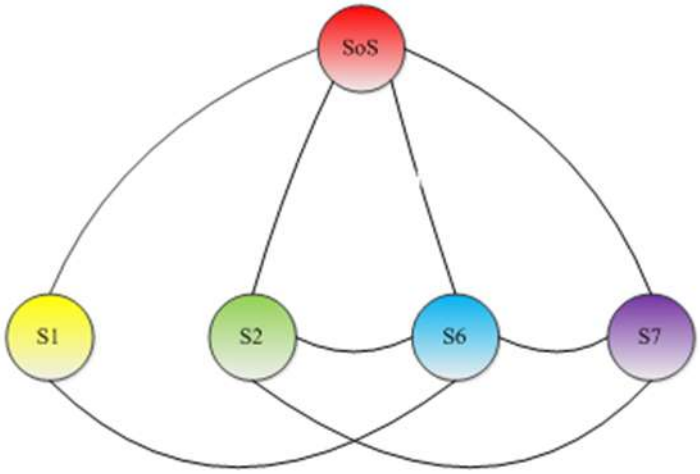


Figure 35. Selected of system of systems architecture

7.1.3 Concluding Remarks on Initial Concept Testing

It was possible to produce a system of systems architecture using genetic algorithms with fuzzy logic providing a fitness assessor. In addition, the solution converged after an acceptable number of iterations of 200 when a relatively small population of 30 is used, which demonstrates the usefulness of the methodology. However, the recommended solution was not always repeatable. That is, different architectures may result when multiple runs are performed.

As expected and discussed the final architecture is not dependent on the initial conditions. It is more a function of the QFD values designated for the different attributes with respect to the various systems and interfaces, membership functions and fuzzy rules put in place.

It is also noticed that the recommended number of interfaces is small compared to the total possible ones. More specifically, only four interfaces have been recommended out of possible 45 interfaces.