A flexible architecture and object-oriented model for space logistics simulation

A Flexible Architecture and Object-oriented Model for Space Logistics Simulation

Olivier de Weck[1], Nii Armar[2], Paul Grogan[3], and Afreen Siddiqi[4]
Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA, 02139

and

Gene Lee[5], Elizabeth Jordan[6], and Robert Shishko[7]
Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, 91109

This paper summarizes the motivation for and the resulting development of a flexible object-oriented software model for simulation and analysis of space exploration campaign logistics. The software model was designed to be applicable to several widely-varying use case scenarios including International Space Station resupply, support for a long-term lunar outpost and global lunar exploration, and Martian exploration by human and robotic agents. The model includes analysis capability for exploration sustainability topics including reusability, reconfigurability, commonality, and repairability. Key additional features include modular demand models that are easily interchanged, element-level models for fine-grained demand integration, and reconfigurable system operational states to dynamically change demand models. Other capabilities include generalized impulsive burn maneuvers, surface transportation, abstracted flight transportation, improved support for multi-destination scenarios, and multi-user collaboration via online databases.

Nomenclature

COTS = Commercial Orbital Transportation Services
EVA = Extravehicular Activity
GUI = Graphical User Interface
IP = International Partner
ISRU = In-situ Resource Utilization
ISS = International Space Station
KSC = Kennedy Space Center
LEO = Low Earth Orbit
LLPO = Low Lunar Polar Orbit
LNP = Lunar North Pole
LSP = Lunar South Pole
MOE = Measure of Effectiveness
NEO = Near Earth Object
OMS = Orbital Maneuvering System
PAC = Pacific Ocean Splashdown Site
RCS = Reaction Control System

[1] Associate Professor, Aeronautics and Astronautics, MIT, 33-410, MIT, AIAA Associate Fellow.
[2] Graduate Research Assistant, Aeronautics and Astronautics, MIT, AIAA Member.
[3] Graduate Research Assistant, Aeronautics and Astronautics, MIT, 33-409, AIAA Member.
[4] Postdoctoral Research Associate, Aeronautics and Astronautics, MIT, AIAA Member.
[5] Staff Engineer, Mission Systems Concepts, JPL, 301-180, AIAA Member.
[6] Staff Engineer, Mission Systems Concepts, JPL, 301-180, AIAA Member.
[7] Principal Engineer, Mission Systems Concepts, JPL, 301-180, AIAA Member.

I. Introduction

DEVELOPING an effective underlying model in novel applications often requires significant iterative work to realize the essential components and form an overall architecture. Applications typically start with specialized capabilities that are later expanded to more general cases through abstraction. This paper summarizes a general object-oriented software model used to capture logistics-related aspects of space exploration scenarios, and demonstrates its implementation in SpaceNet 2.5.

Object-oriented data structures helped to reveal a flexible architecture that forms the core of the software model. A fundamental change in the underlying software model from previous versions has enabled a vastly expanded capability in analysis. Some of the major additions include: OMS and RCS burn capability, support for inclusion of most planetary bodies, orbiting moons, and Lagrange points, surface vehicles and surface transportation, abstracted flight transportation for known flight architectures, multi-mission, multi-destination scenarios, reconfigurable element operational states, element-level demand models, common parts and spares scavenging, and repairable parts. By leveraging the flexible nature of the software model, new capabilities will be much easier to add in future development.

II. Motivation

The prime motivation for developing a flexible architecture and object-oriented model was for implementation in a new version of SpaceNet, an integrated space logistics analysis tool for space exploration first developed in 2005.[1] Further generalization of the existing model from SpaceNet had reached a point of decreasing returns due to a restricting bus-like architecture of the underlying procedural code. Object-oriented programming methods were desired to introduce higher levels of modularity and flexibility for ease of future development.

With a more flexible architecture for development, there is a desire to analyze the "-ilities" of space logistics: reusability, reconfigurability, repairability, and commonality, which have a large impact on outpost sustainability.[2] There is also interest to provide a improved support for building and analyzing more general exploration scenarios including long-duration, multi-flight campaigns, with multiple destinations and exploration sites at diverse locations in the solar system.[3]

A. Underlying Software Architecture

SpaceNet was first focused on the analysis of individual lunar sortie missions. SpaceNet 1.3 was publicly released in 2007 as a MATLAB program.[4] The underlying software model of SpaceNet 1.3 relied on a central file to access many of the sub-routines necessary to build and evaluate a scenario (See Figure 1a).

Further development broadened the analysis capability to include multi-mission campaigns at a single lunar outpost, resulting in SpaceNet 1.4.[2] The revised version of SpaceNet abstracted portions of the existing software model to create a more modular architecture created a more modular code structure, but the majority of sub-routines were still accessed by a one central file (See Figure 1b).

The file that forms the central "hub" in both SpaceNet 1.3 and 1.4 is the GUI. Under this architecture as the complexity ballooned, it became increasingly difficult to generalize portions of the software model or expand the functionality since changes could not easily be isolated to a specific module of code.

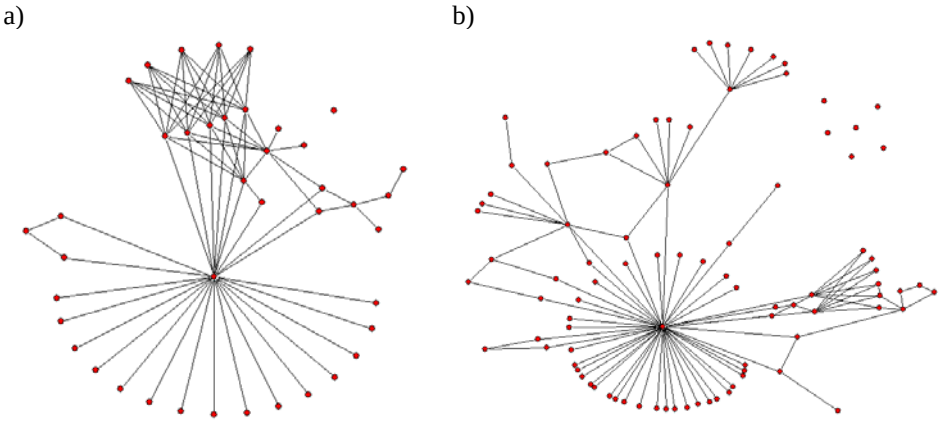

Figure 1. SpaceNet file dependency visualization. *SpaceNet 1.3 (a) and 1.4 (b) use a central file (the GUI) in a bus-like architecture to control user interaction. The GUI accesses sub-routines which assist in the construction of a scenario. With increasing code complexity, it is difficult to isolate changes under this type architecture.*

SpaceNet 2.0 served as an internal prototype to replace the existing procedural code with object-oriented programming, using the concepts of encapsulation, polymorphism, and inheritance to create a more modular and flexible architecture. Under a modular architecture, interfaces are used to abstract object definitions, allowing for a rich object hierarchy and flexibility for future development. SpaceNet 2.5 built on the concepts explored in SpaceNet 2.0 while expanding analysis capabilities and focusing on a diverse set of use case scenarios.

B. "-Ilities" Analysis Capability

The improved analysis capabilities were focused on four "-ilities" of space logistics: reusability, reconfigurability, commonality, and repairability. The "-ilities" were identified as increasingly important for analyzing the future human exploration campaigns for a number of reasons discussed below. Since the existing software model could not elegantly handle these concepts, the improved analysis capabilities formed the core of the new requirements.

As human habitation duration is stretched to longer periods at remote outposts, utilizing pre-positioning of resources and infrastructure becomes essential. Reusability focuses on the reuse of elements across missions, including pre-positioning and re-positioning planetary assets for future use. The challenges of implementing reusability included decoupling elements from specific missions and persisting elements and their effects on demands throughout the entire simulation.

Faced with the enormous expense of delivering infrastructure to remote locations, infrastructure elements will have to perform more than one function to save on mass and upkeep.[5] Reconfigurability focuses on defining operational states for elements which can be dynamically changed according to mission plan. The operational state defines the element's duty cycle and models used to generate demands for resources. The challenges of implementing reconfigurability included sufficiently describing different operational states of elements and dynamically changing the underlying models generating demands.

With infrequent resupply missions, future exploration must make the most of existing spares stocks. Commonality focuses on creating common parts for elements so methods like spares pooling and scavenging from unused elements can be used to decrease demands for spare parts throughout the scenario. The challenges of implementing commonality included adding the ability to scavenge from existing elements and developing a metric to evaluate scenario commonality.

As human surface duration is extended and crew time becomes more available, repair rather than replacement may become a desirable action for broken or damaged parts. Repairability focuses on creating repairable parts for elements where crew time can be traded for reduced spares demands via corrective repair activities. The challenges of implementing repairability included adding the ability to choose repair actions and evaluating the impact on the overall scenario.

C. Use Case Scenarios

A set of use case scenarios was developed to aid the development of a flexible software model that could adapt to a wide range of interesting scenarios. The use case scenarios range in complexity from a single lunar sortie to an extended human exploration of Mars. Although the capability to completely model each scenario may not immediately be available, the hope is for easier future development with a broad range of applications in mind.

Table 1. Use case scenario summary.

Use Case Scenario	Primary System	Number of Nodes	Duration	Number of Missions
Lunar Sortie	Earth-Moon	5	14 days	1
ISS Resupply	Earth	6	5 years	40
Lunar Hub-Spoke	Earth-Moon	6	10 years	20
Lunar Spoke-Hub	Earth-Moon	12	10 years	20
Mars	Earth-Mars	8	20 years	16

The Lunar Sortie use case models a single Constellation-class mission for a seven-day duration exploration of the Lunar South Pole, serving as a validation scenario to match existing capability.

The ISS Resupply use case models the sustainment of the International Space Station post shuttle retirement (2010-2015) utilizing a combination of International Partners (IP) and Commercial Orbital Transportation Services (COTS). Challenges associated with this use case include a non-zero initial state and requirements for detailed models to accurately model demands for resources and spare parts.

The Lunar Hub-Spoke use case models the build-up and sustainment of an outpost at the Lunar South Pole. Challenges associated with this use case include a high level of element reuse across missions and an increasing dependence on ISRU technology.

The Lunar Spoke-Hub use case models a global exploration of the Moon including a multi-mission traversal from the Lunar North Pole to the Lunar South Pole using a mobile habitation element. Challenges associated with this use case include multi-destination missions and capturing surface-to-surface transportation.

The Mars use case models a precursor robotic exploration of the Martian surface, followed by human-crewed missions based on Mars Design Reference Architecture 5.0.[6] Challenges associated with this use case include a high variance of launch windows and flight duration associated with orbital dynamics, increasing concern for alternative propulsion methods, and increasing health concerns for long-term human space habitation

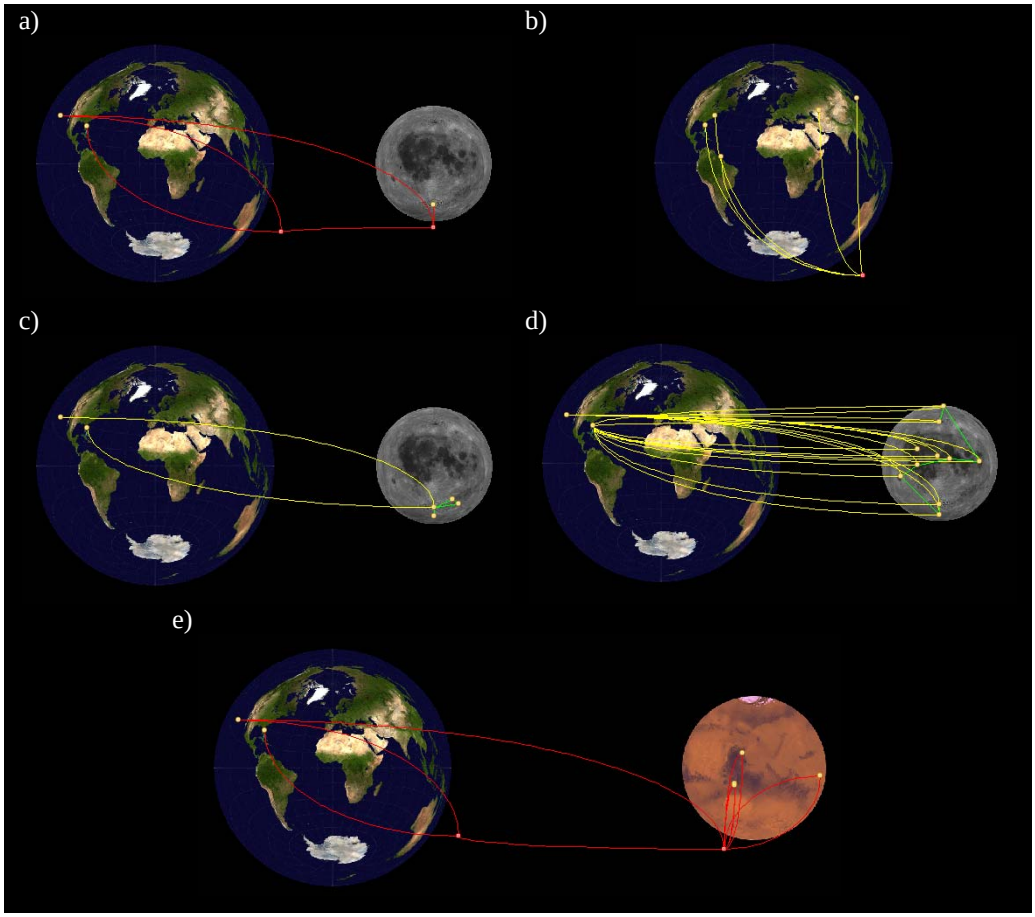

Figure 2. Use case scenario networks. *These network diagrams were generated by SpaceNet to illustrate the scope of the use case scenarios. (a) Lunar sortie validates existing analysis capability for a Constellation-class mission to LSP. (b) ISS resupply analyzes the IP and COTS resupply of ISS post shuttle retirement. (c) Lunar hub-spoke builds a central outpost at LSP with expeditions to nearby locations. (d) Lunar spoke-hub performs a traversal from LNP to LSP with several long-distance surface missions. (e) Mars exploration uses robotic precursor missions to select a site for human exploration.*

III. SpaceNet Software Model

SpaceNet uses a discrete event simulator to model the flow of elements and resources through a time-expanded network of nodes and edges. At the core, there are three high-level objects: the event stack, the network state and the simulator. The event stack is a listing of user-defined events that will drive the simulation by serving as input to the simulator. The network state tracks the elements' locations and other properties including nested elements and available resources. The simulator sequentially polls and executes events, altering the network state and possibly generating additional events, also maintaining and updating any relevant simulation parameters like the simulation time and measures of effectiveness (MOEs).

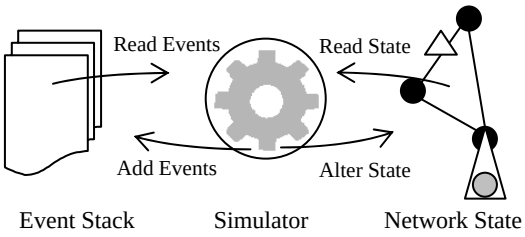

Figure 3. Discrete event simulator. *The three high-level objects include the event stack, the network state (comprised of nodes, edges, elements and resources), and the simulator.*

The lower-level object groupings in SpaceNet include domain objects which include the network components and objects which flow through the network, event objects which drive the simulation and associated logic, and demand objects which coordinate demand generation and satisfaction during the simulation. Note that many of the following Unified Modeling Language class diagrams are simplified for ease of understanding.

A. Domain Objects

Locations are components of the network that can contain elements. There are two branches in the hierarchy of locations: nodes representing time-invariant locations and edges representing paths between nodes. Surface nodes represent a location on a planetary body given by a latitude and longitude. Orbital nodes represent stable orbits about a planetary body given by an inclination, apoapsis, and periapsis. Lagrange nodes represent the five stationary solutions in a three-body system. Surface edges represent paths between two surface nodes. Space edges represent paths between two nodes using a series of impulsive burns. Flight edges represent a path between two nodes using a flight architecture that is known to close with a given cargo and crew capacity.

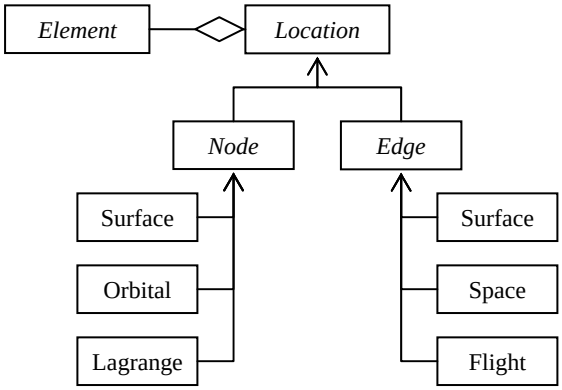

Figure 4. Network location hierarchy. *Network locations can contain elements and are either nodes or edges. There are three subclasses of nodes and three subclasses of edges.*

The hierarchy of locations is flexible for future development through the Node and Edge interfaces. New object classes, such as near-Earth object (NEO) nodes, time-dependent edges, and low-thrust (ion propulsion) edges would be interchangeable with the existing objects in any generic application, such as those that happen in most underlying simulation code (e.g. tracking demands or element locations).

Resources represent substances used to satisfy demands. There are three kinds of resources: continuous, generic, and discrete. Continuous resources represent substances such as propellants, liquids, and gases. Generic resources represent substances that are abstracted to continuous amounts in analysis such as food and hygiene equipment. Discrete resources represent indivisible objects such as spare parts and other whole supply items.

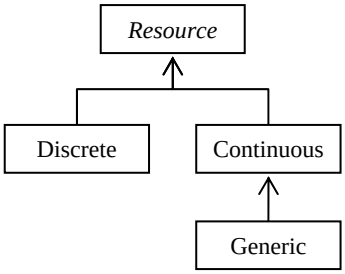

Figure 5. Resource hierarchy. *Resources are either discrete or continuous. Generic resources, meaning aggregated and abstracted as continuous for ease of analysis, are a subclass of continuous resources.*

The existing resource class architecture is accommodating for all current demands through customizable units of measure, unit masses, and unit volumes. For example, the extension to include energy

as a new resource could be accomplished by creating a new continuous resource with units of kilowatt-hours and a unit mass and unit volume of 0. Since the three resource types are united with the Resource interface, analysis can combine different levels of detail, for example using discrete resources for sparing, continuous resources for propellant, and generic resources for consumables.

Elements are the basic building blocks of scenarios and have a rich hierarchy to represent different functional capabilities. All elements have a mass and volume and can generate demands for resources. Human agents are elements that represent human crew members. Resource containers are elements that can contain resources up to a capacity constraint. Carriers are elements that can contain other elements up to a capacity constraint. Propulsive vehicles are carriers that can perform OMS and/or RCS impulsive burns. Surface vehicles are carriers that can navigate surface edges up to a maximum speed.

Any element may also contain parts, which are a combination of a resource (to determine commonality) and application-specific quantities such as a quantity, mean time to failure, and mean time to repair. Parts can be used as inputs to various demand models, identifiers for repair activities, or can be scavenged for the respective resource if the element is placed in a decommissioned state.

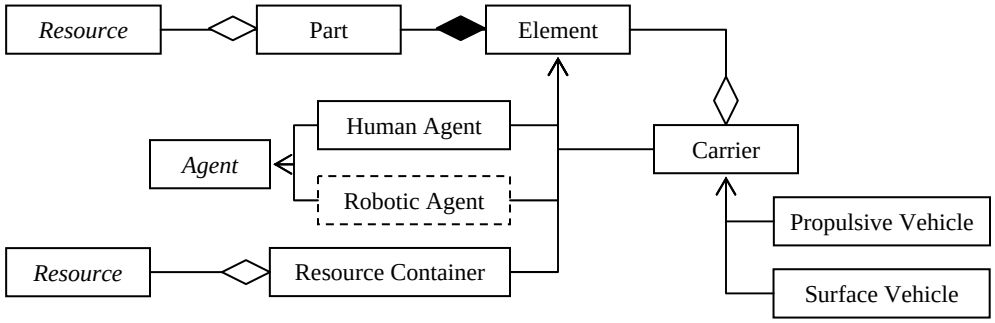

Figure 6. Element hierarchy. *Elements may contain parts, which as associated with a resource. Subclasses of elements include human agents (crew), robotic agents (reserved for future expansion) resource containers that can hold resources and carriers that can hold other elements. Subclasses of carriers include propulsive vehicles which can perform impulsive burns, and surface vehicles which can drive along surface edges.*

The hierarchy of elements is likely to be expanded with future development. The core interfaces Element, Agent, Resource Container, and Carrier provide a framework that is easy to manage. For example, if low-thrust (ion propulsion) vehicles were desired, a new sub-class of Carrier could be created similar to the existing Propulsive Vehicle.

B. Event Objects

Events are objects that are executed by the simulator. There are four low-level events governing elements: Create, Move, Remove, and Reconfigure. Create events instantiate new elements at a location. Move events change the location of an element. Remove events remove an element from the simulation. Reconfigure events change the operational state of an element. There are three low-level events governing resources: Add, Transfer, and Use. Add events add resources to an existing resource container. Transfer events move resources to a co-located resource container. Use events remove resources from an existing container to satisfy a demand.

Although simulation could be accomplished using only these low-level events, higher-order events ease the burden to define complex events and processes. For example, a Burn event, which models an impulsive burn, is comprised of several Use events to model propellant usage, and Remove events to model staging of expended elements. A Space Transport event, which models movement on a space edge, is comprised of several Move and Burn events. Similarly, an EVA event is comprised of several Move events to model crew movement, and Reconfigure events to model reconfiguration to higher EVA demands. An Exploration event schedules EVA events at a specified frequency for a long-term site exploration.

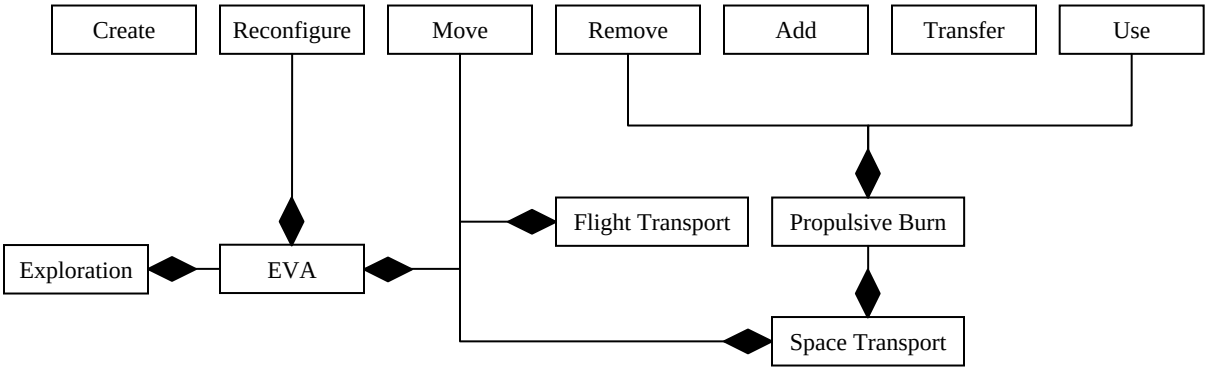

Figure 7. Event compositions. *Seven core events perform low-level functions for elements (Create, Reconfigure, Move, Remove) and resources (Add, Transfer, Use). Higher-level events use combinations of core events to model more complex interactions.*

Future development will likely introduce new higher-order events to model new processes such as low-thrust transportation, preventative maintenance activities, and infrastructure assembly; however the low-level events are not likely to change, isolating the change propagation.

C. Demand Objects

States are closely tied with element-level demands, describing an element's operational capability. State are classified as either Active, Quiescent (partially active), Dormant (not active), or Decommissioned (available for scavenging). Once an element enters a decommissioned state, it cannot be reconfigured to any other state because its parts availability cannot be guaranteed. An element can have any number of possible states, and one current state, which can be changed by undergoing a reconfigure event. Each state specifies a set of demand models that generate demands for the element while it is in that state.

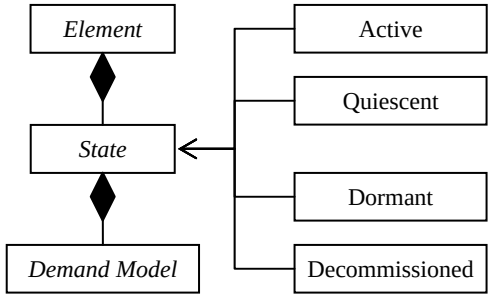

Figure 8. State hierarchy. *Elements' states are either Active, Quiescent, Dormant, or Decommissioned, and each state can have a set of demand models.*

Demand models are objects that generate a set of demands for resources during simulation. In SpaceNet, there are two functional classes of demand models, mission-level models and element-level models, although there is no operational difference. Mission-level models represent the traditional demand modeling method, and are used on a per-mission basis to generate and aggregate demands for the entire mission to one point in time. Mission-level models typically use parameterized inputs such as the number and type of instantiated elements, the number of activities like EVAs, and other related information.

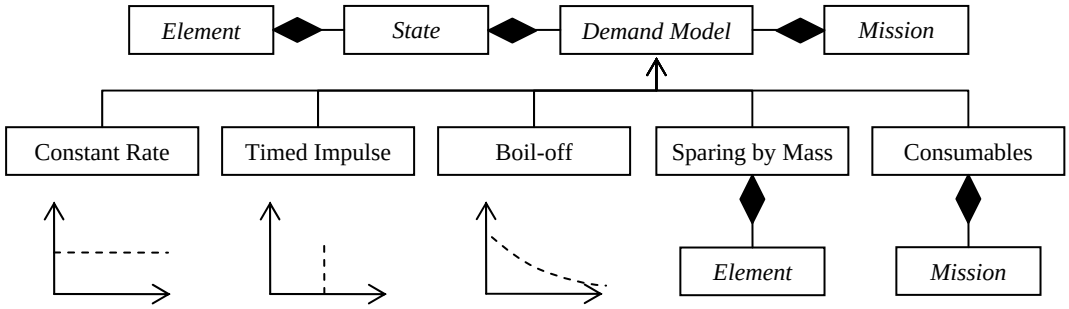

Figure 9. Demand model hierarchy. *There are a wide variety of modular demand models suitable for a range of applications. Some models (Constant Rate, Timed Impulse, Boil-off) do not depend on any external objects, making the demand profile over time easy to visualize. Other models (Sparing by Mass, Consumables) depend on parameters from the associated objects and may vary widely by scenario.*

Element-level models are used in association with individual element states to generate demands throughout simulation based on the elapsed time since the last event. Since all demand models implement the same interface, they are easily interchangeable and can be combined to create highly customizable composite models. Current demand models include a mission-level consumables model, constant rate demands, timed impulse demands, boil-off demands, sparing by mass demands. Future demand models could represent ISRU production, advanced sparing models, and resource shelf-life.

IV. Demonstrative Examples

To aid understanding of the new software model and architecture, three specific examples of improvements are demonstrated as implemented in SpaceNet 2.5. The first example investigates the improved flexibility for transportation, focusing on the usage of flight transports as an abstraction of space transports in a notional Lunar Sortie scenario. The second example investigates the usefulness of flexible, modular demand models in a notional scenario modeling the build-up of an outpost at LSP. Finally, the third example showcases the potential for user collaboration via online databases.

A. Transportation Flexibility

The existing SpaceNet model only allowed one method of transport between nodes: impulsive burns utilizing OMS engines. In the revised model, the concept of transportation was abstracted to three events: space transport events utilizing propulsive vehicles on space edges, flight transport events utilizing notional vehicles on flight edges, and surface transport events utilizing surface vehicles on surface edges.

In a long-duration exploration encompassing several dozen missions, building the burn-by-burn space transportation from Earth to the destination becomes increasingly burdensome for the user. As an abstraction, flight architectures can be developed that are known to achieve propulsive closure so detailed analysis is not required after the initial assessment. These known flight architectures have been abstracted with a flight edge (and associated flight transport event), which allows the transportation of elements from an origin node to a destination node in a specified transfer duration up to a set mass and crew capacity.

Using flight transports with the baseline Constellation architecture, the number of events required to model a single Lunar Sortie scenario is reduced from twelve to six. The abstraction does not take into account propellant, rendezvous, crew transfer, and undocking so notional carriers are typically used for delivery and/or return instead of realistic vehicle models. Although some detail is inevitably lost, the method of flights has been very useful in feasibly modeling long-duration exploration missions. In this team's experiences with long-duration explorations, flight transportation has been critical to rapid development and iteration of scenarios.

Lunar Sortie with Space Transports (see Figure 10)

1. Create Ares I launch stack
2. Create crew members inside crew module
3. Launch Ares I stack for transport to LEO
4. Create Ares V launch stack
5. Launch Ares V stack for transport to LEO
6. Perform TLI/LOI transport to LLPO
7. Move crew from crew module to ascent module
8. Perform lunar descent transport to LSP
9. Explore at LSP
10. Perform lunar ascent transport to LLPO
11. Move crew from ascent module to crew module
12. Perform TEI transport to PAC splashdown

Lunar Sortie with Flight Transports (see Figure 11)

1. Create notional departure and return vehicles
2. Create crew members inside departure vehicle
3. Perform flight transport to LSP
4. Explore at LSP
5. Move crew to return vehicle
6. Perform flight transport to PAC splashdown

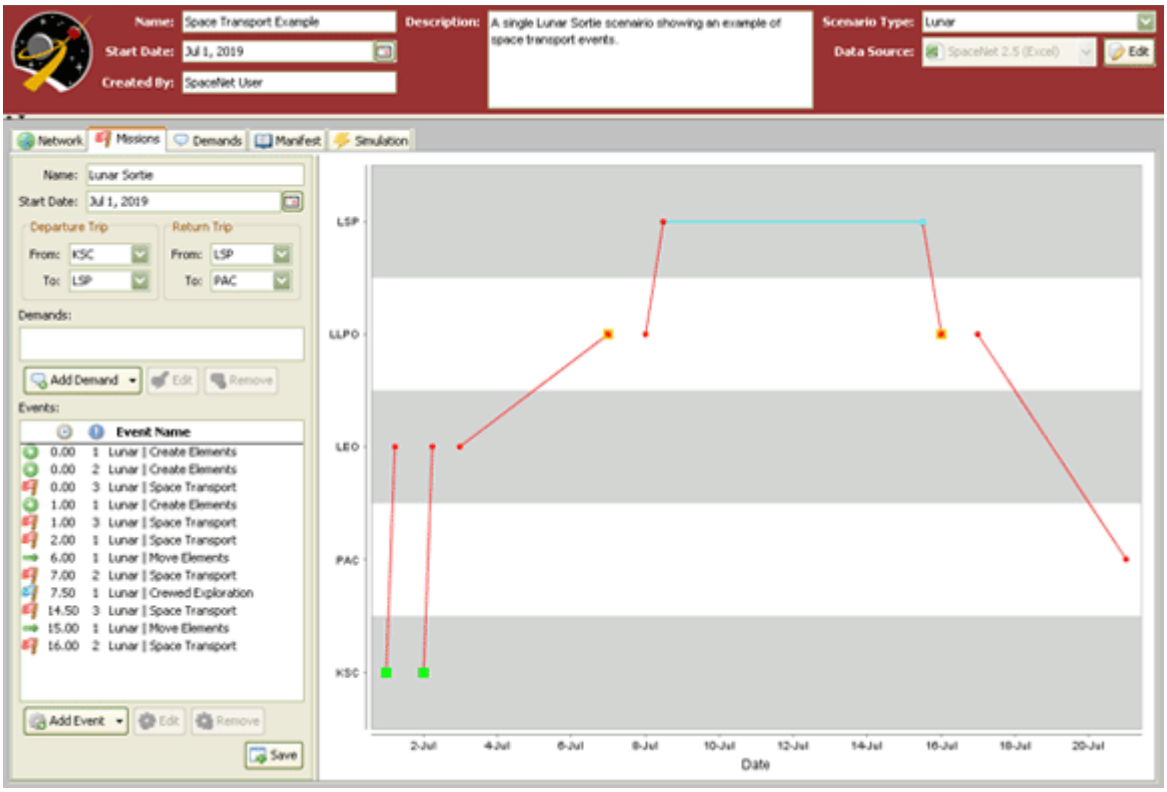

Figure 10. Scenario with space transports. *Space transport events capture details of the dual Ares I / Ares V launches and check logistics closure for propellant demands during all orbital maneuvers.*

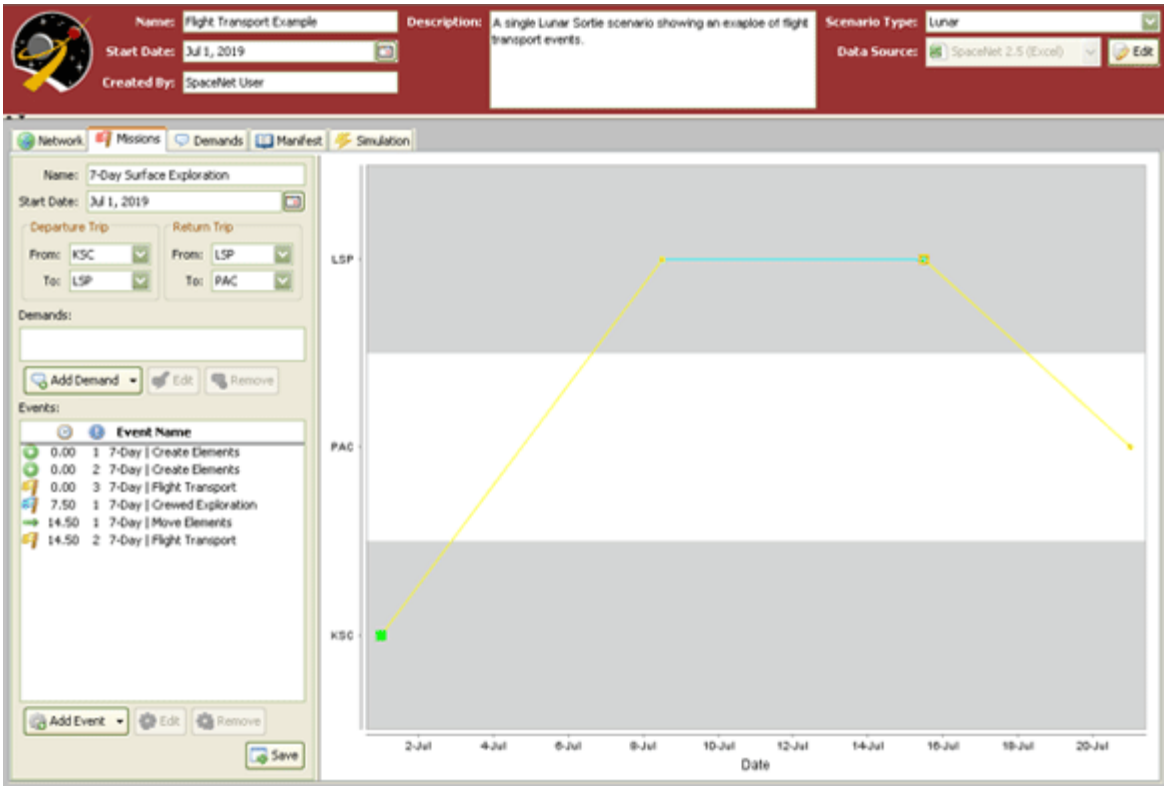

Figure 11. Scenario with flight transports. *Flight transports abstract the detail of getting from the launch pad to the destination, losing detail but gaining the advantage of much faster modeling.*

B. Demand Model Flexibility

Flexible, modular demand models have led to a new method of capturing demands for resources by using element-level models. Past efforts have used parameterized mission-level demand models to capture demands for resources during one mission based on its duration, number of crew, number of EVAs, and other relevant information. In long-duration campaigns with a high degree of element reuse between missions and multiple operational levels, this method starts to break down.

In this example, a five-mission scenario investigating the initial build-up phase of a lunar outpost is analyzed. Abstracted flights are used to build the following mission sequence:

1. Automated check-out without crew
2. 3-day crewed exploration at LSP
3. Cargo mission to deliver habitat module and power supply element
4. 7-day crewed mission to unpack and set up habitat
5. 14-day crewed mission at habitat

Within each crewed mission, traditional mission-level models are used for the easily-parameterized demands categorized as crew provisions, crew consumables, and waste management. The parameters used as inputs to the mission-level model include number of crew, surface exploration duration, number and duration of EVAs, airlock volume, and crew habitat leak rate.

After the habitat module and power supply element are delivered, modeling the demands for spare parts via parameterization becomes more complicated due to the split time between active and dormant states. When the crew members are present at the outpost, the habitat and power supply should be reconfigured to an active state, generating spares at a presumably higher rate than the dormant state between crewed missions.

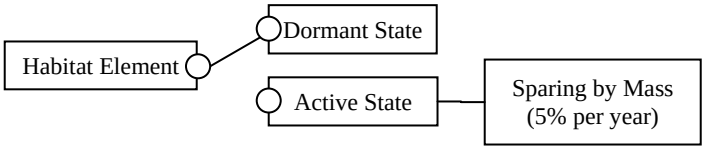

Figure 12. Habitat operational states. *The active state for the habitat element includes a demand model to generate spares demand at a rate of 5% of the element's dry mass per year.*

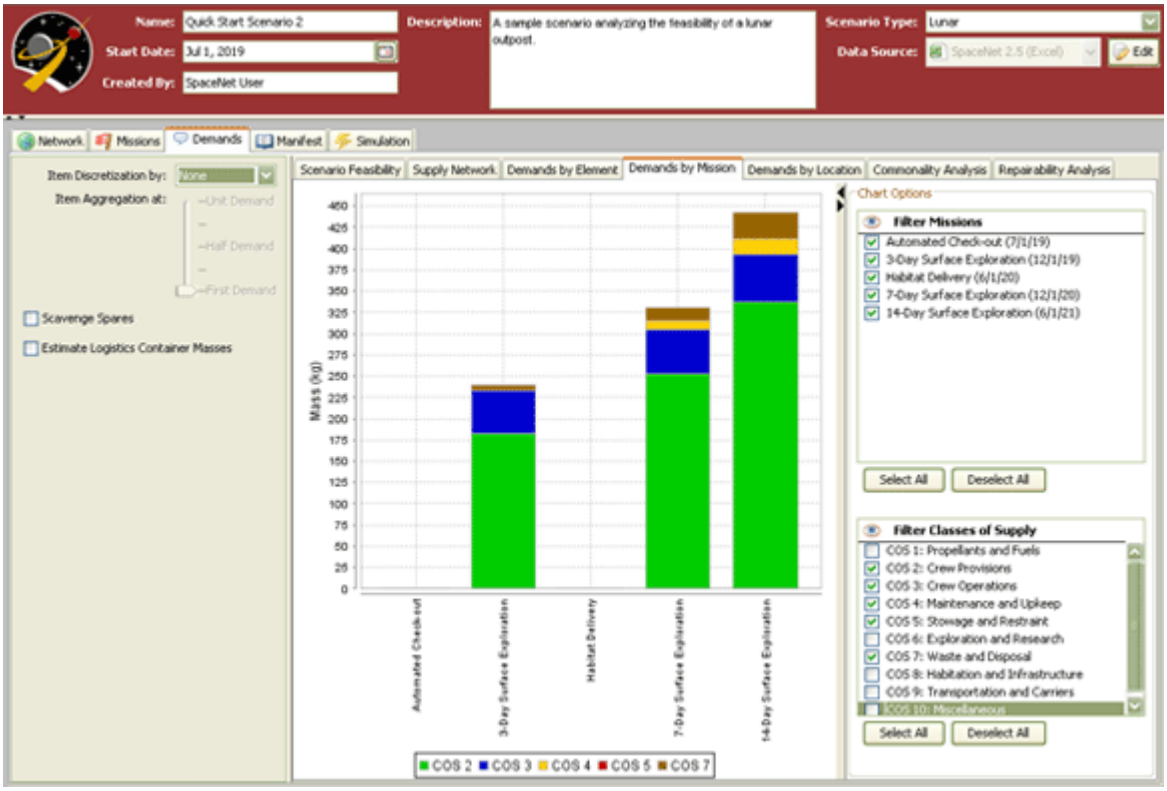

Figure 13. Demands by mission. *Total demands can be seen, grouped by mission and demand category (class of supply). This aggregated view of demands includes demands from the mission-level model for crew provisions (COS 2, green), crew operations (COS 3, blue), and waste (COS 7, brown) and from the element-level models for spares (COS 4, yellow).*

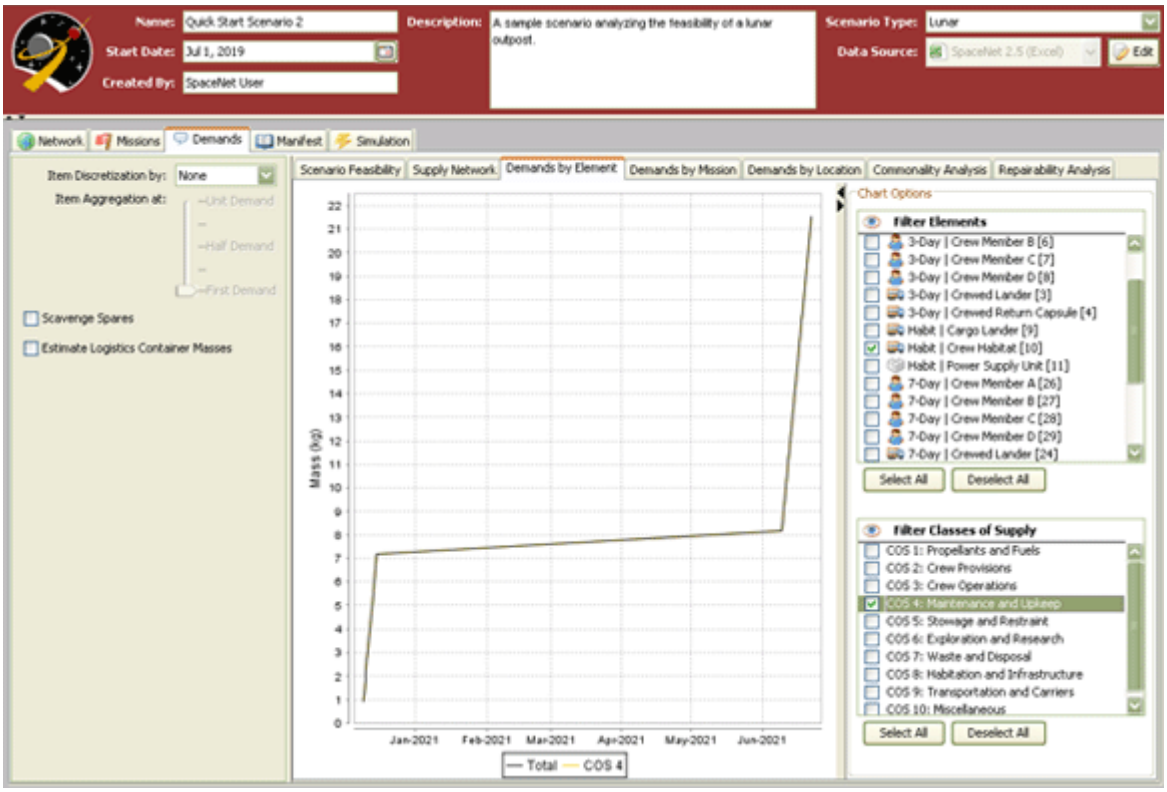

Figure 14. Demands by element. *Demands can be broken down to individual elements using element-level demand models, in this case the habitat element. The bi-model demand rates can clearly be seen, differentiating the time when crews are using the habitat and time between missions.*

C. Data Source Independence

Although not directly tied to any use case scenario, the improved software model in SpaceNet 2.5 allows for multiple formats of data sources. This change has been motivated from inefficiencies in managing multiple file-format databases between users in past analyses. While support has been maintained for an expanded format of Excel spreadsheet database similar to SpaceNet 1.3, online relational databases have also been created to aid in the collaboration of multiple users.

Relational databases are a natural method of storing data for object-oriented programming, as the data structures can be conveniently represented using tables and primary/foreign key relationships. For example, the SpaceNet database has tables for nodes, edges, elements, states, and demand models. The elements, states, and demand models tables are related with primary / foreign key relationships.

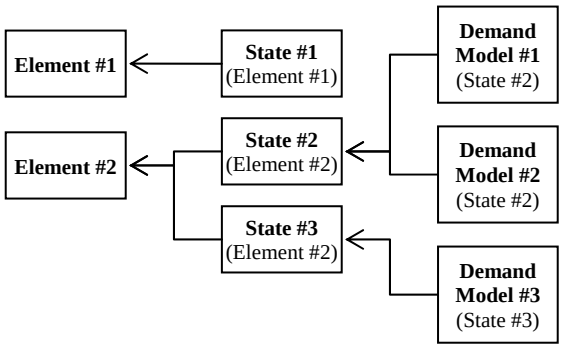

Table 2. Database relationships. *Direct user interaction with the database can be frustrating due to compound relations between tables.*

Although useful from a programming prospective, direct user interaction with relational databases can be frustrating due to a waterfall of key relationships (see Figure 14). One of the goals in the near future is to create a database editing client to provide easier user interactions with the databases.

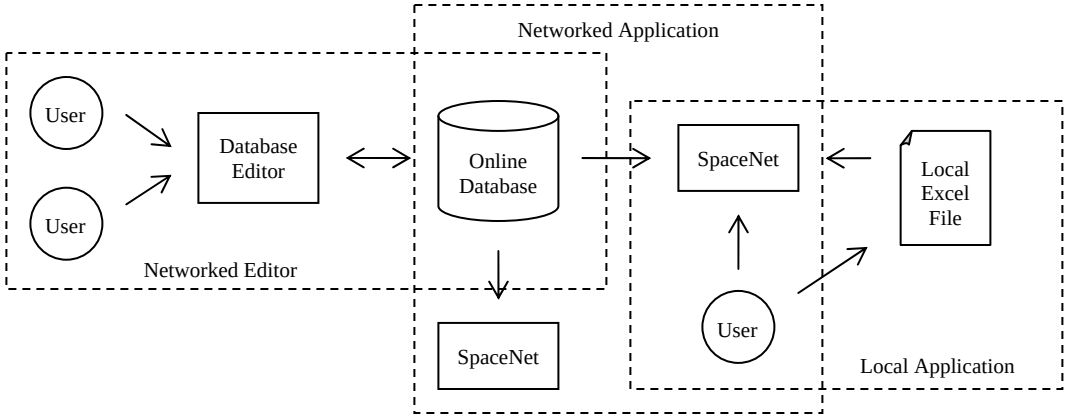

Figure 15. Online user collaboration. *The local application utilizes an Excel spreadsheet as a database, while the networked application can connect to an online relational database which can be edited by multiple users through an associated data editor.*

As in any integrated analysis, the results are only as good as the input information – by providing an easy-to-use collaborative environment, the hopes are that it will become easier to perform more detailed analysis. Although nearly operational in basic capacity, the details of user management and data security have yet to be fully determined and are important for the sensitive nature of some information.

V. Conclusion

The improvements to the SpaceNet software architecture and underlying model have produced greatly expanded capabilities and a more flexible environment for future development. The upgrades have enabled the analysis of the "-itilies" of space logistics in a wide-array of potential use case scenarios. Improvements in transportation flexibility, demand modeling flexibility, and data source flexibility have already positively affected the process of building and analyzing scenarios.

A. Assumptions and Limitations

Although improved, the modeling effort in SpaceNet still has several assumptions and limitations that should be considered. First, as in SpaceNet 1.3, resources are shared among all co-located elements (no "micro" logistics) and there is no way to guarantee which resources will be consumed in demands – this sometimes causes minor resource insufficiencies when resources are inadvertently consumed from a container scheduled for transport to another location.

Second, methods to optimally (or near-optimally) pack and manifest resources for multi-mission, multi-destination scenarios are still under development.[7,8] The current manifesting module is not heavily tested, and the developers have encountered some limitations that are more explicitly outlined in the SpaceNet user's guide. Manual manifesting may be needed to properly guarantee that resources are properly located as the current auto-manifesting option uses simple brute-force tactics.

Finally, SpaceNet does not currently perform stochastic analysis of probabilistic events. The primary roadblock to these types of analysis is the auto-manifesting problem discussed above. Human-in-the-loop manifesting is not acceptable for performing the large number of Monte-Carlo simulations required to model the range of potential low-probability events. A more sophisticated framework will be required to properly handle re-manifesting lost or incapacitated elements and dynamically update demand projections.

B. Future Work

There are a lot of areas of future work that are now readily accessible given a flexible software model. Current efforts are focused in the following directions:

- Time-dependent trajectories utilizing orbital dynamics
- Low-thrust (ion propulsion) travel
- More detailed crew time analysis including preventative maintenance
- Developing a database editing client

- General access to online databases, including user management and data security
- Improved manifesting heuristics to for multi-mission, multi-destination scenarios

Acknowledgments

Financial support for this research was provided by the Jet Propulsion Laboratory (JPL) under the Strategic University Relations Program (SURP) and contract number 1344341. The authors thank Takuto Ishimatsu for his help with the Mars use case scenario and guidance with time-dependent trajectories.

References

[1] Cirillo W.M., Earle K.D., Goodliff K.E., Reeves J.D., Stromgren C., Andraschko M.R., and Merril R.G., "Strategic Analysis Overview", AIAA-2008-7778, *AIAA Space 2008 Conference and Exposition*, San Diego, California, September 9-11, 2008.

[2] Lee G., de Weck O.L., Armar N., Jordan E., Shishko R., Siddiqi A., and Whiting J., "SpaceNet: Modeling and Simulating Space Logistics", AIAA-2008-7747, *AIAA Space 2008 Conference and Exposition*, San Diego, California, September 9-11, 2008.

[3] Shull, S. "Integrated Modeling and Simulation of Lunar Exploration Campaign Logistics", M.S. Thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, Massachusetts. 2007.

[4] de Weck O.L., Simchi-Levi D., Shishko R., Ahn J., Gralla E., Klabjan D., Mellein J., Shull A., Siddiqi A., Bairstow B, Lee G., "SpaceNet v1.3 User's Guide", NASA/TP-2007-214725, January 2007.

[5] Siddiqi A., de Weck O., "Spare Parts Requirements for Space Missions with Reconfigurability and Commonality", Journal of Spacecraft and Rockets, 44 (1), 147-155, January-February 2007.

[6] Drake, B., "Human Exploration of Mars Design Reference Architecture 5.0," *NASA Website*, URL: http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20090012109_2009010520.pdf [cited 11 August, 2009].

[7] Siddiqi A., de Weck O.L., and Lee., G, "Matrix Modeling Methods for Space Exploration Campaign Logistics Analysis", AIAA-2008-7749, *AIAA Space 2008 Conference and Exposition*, San Diego, California, September 9-11, 2008.

[8] Armar, N. "Cargo Revenue Management for Space Logistics", AIAA-2009-6723, AIAA Space 2009 Conference & Exposition, Pasadena, California, September 14-17, 2009.