Prototype Framework for Dynamic Probabilistic Risk Assessment of Space-Flight Medical Events

Kirsti Pajunen*

Department of Mechanical Engineering, Milwaukee School of Engineering, Milwaukee, WI

Spencer Lane, Brian Moore

Abstract. Medical problems in space, while not necessarily common, can be life and mission threatening. Because of this, a model has been created to assess the risk of certain medical events under space mission conditions called the Integrated Medical Model (IMM). The model uses a type of risk analysis called Probabilistic Risk Assessment. An effort is being conducted to convert these static models to Dynamic PRA (DynaPRA) models. These models would integrate changing conditions in space missions such as human interaction, changing system components, and environmental factors and apply a risk analysis to find more complete and useful probabilities of certain medical events occurring to astronauts in space. To this end, a prototype DynaPRA framework was created during the first three months of the project using the existing IMM hip, wrist, and lumbar spine bone fracture models as a case study. After the initial implementation was complete, ideas were generated for applying this framework to the rest of the IMM.

Introduction

Probabilistic risk assessment. Probabilistic Risk Assessment (PRA) is a systematic process of estimating the probability of reaching an end-state, defined by the combination of individual event probabilities and their consequences. It is typically used for engineering applications, such as for determining the probability of reaching system failure after a valve breaks in a piping network. The PRA simulation is a static, quantitative process that estimates the probability of reaching certain intermediate or end-states resulting from initiating or critical events using a Monte Carlo simulation. The Monte Carlo process estimates the expected value or individual probabilities of the system end-states by conducting many independent trials of the simulation. It calculates through end-result estimation, a method which is much more efficient than direct calculation of the probabilities.

Dynamic probabilistic risk assessment. PRA may be a useful tool in many situations; however, some engineering systems are inherently dynamic. As the name implies, dynamic systems change over time, and the response of a dynamic system may change over time due to multiple external factors. Therefore, a static simulation like PRA does not always accurately represent a dynamic system. Dynamic probabilistic risk assessment, or DynaPRA, is an extension of the PRA concept that characterizes complex dynamic systems, unlike PRA. It accounts for the complex interactions between a system and its internal and external factors. All possible event chains, of which there are infinitely many, can be interrogated using DynaPRA. DynaPRA incorporates dynamic changes to a subsystem of the overall system that may affect other subsystems, like how a broken valve in a piping system can affect the chance that another valve will break. Because of the greater complexity and higher sensitivity of DynaPRA, it allows for a much more complex risk assessment than PRA.

*Funding provided by the Wisconsin Space Grant Consortium.

Integrated Medical Model. The Integrated Medical Model (IMM) is a PRA simulation created through the HRP headed at JSC and at Wyle Labs, JSC's subcontractor for the project. GRC has created several new modules for the IMM where data is sparse, such as the bone fracture and renal modules. It is an ongoing project that attempts to quantify medical risks associated with space travel, including those that have not been observed to date, such as bone fractures, kidney stones, and sleeping problems. For example, the risk associated with bone fractures increases as a mission proceeds due to the decrease in bone mineral density (BMD) in low gravity environments. One main purpose of the IMM is to guide space mission planners in selecting appropriate medical treatments for each mission, while improving the safety of the astronauts in flight (Griffin, 2012). The IMM uses PRA with Monte Carlo simulations to determine the likelihood of 87 different medical events occurring to astronauts while they are on space missions. The IMM is written in SAS, but the modules that GRC constructs are primarily written in MATLAB. The current static simulation nature of the IMM proves useful in some cases, such as finding the probability of a bone fracture occurring after a loading event with a fixed probability. In other cases, however, a more dynamic simulation is needed.

Bone model. There are three existing bone fracture models from the IMM that were used as a case study for converting the static PRA nature of the IMM to a DynaPRA nature. These models include the hip, wrist, and lumbar spine models. The hip model represents a posterolateral fall onto a hip from the side. A fracture occurring in this model is representative of a fracture in the proximal femur, including the femoral neck. The wrist model models a generic load applied to the wrist such as that from a fall forward to the side. The lumbar spine model models a fall from a ladder onto two feet (Nelson, 2009).

IMM shortcomings and motivation. The IMM is not in itself intended to be accurate, which is a characteristic of PRA. Instead, it is designed to be inaccurate in that it is trying to capture the entire field of what could happen (the uncertainty in the outcome) because the inputs are uncertain. Thus, DynaPRA would provide the IMM a means to improve the estimate of uncertainty propagation and illustrate its impact as time moves forward. A space mission poses countless possibilities for medical events to occur, and a DynaPRA analysis of the mission would create a more realistic representation of risk variation over time as well as elucidate critical event series that may exacerbate the occurrence and outcomes of time dependent medical events. It would allow for a higher fidelity analysis and an analysis of the interaction effects between various medical events. Unlike the current IMM, with DynaPRA, a wrist fracture could affect a hip fracture in that after a wrist fracture, the astronaut has a higher probability of breaking his hip because he may no longer have a useable wrist to catch himself with as he falls. DynaPRA would be able to analyze how time and other external factors could affect medical risks, such as the space environment, the training and expertise of the astronaut, and changes in the mission (such as changing times for extravehicular activities, or EVAs). The progression of events would not be purely controlled by the logical process as defined by the developer, but would constantly be taking into account interactions that could change the course of events in unique ways. A dynamic IMM could be applied to any sort of mission environment, such as the International Space Station (ISS), Mars, or the Moon. With such a greater accuracy over the current IMM, a DynaPRA IMM could better help mission planners decide what treatments to bring on several different kinds of missions, and to provide more possibly necessary safety precautions for astronauts. It could also help mission planners optimize the crew assignment for

specific missions. Because of all of these reasons, a DynaPRA IMM simulation is being created at GRC. The objectives for this project are described in the next section.

Objectives

Project goals.

1. Construct DynaPRA framework, including credibility testing.
2. Integration of IMM, and then demonstration of IMM integration.
3. Finalize credibility assessment and demonstrate and examine need for additional applications.

Prototype objectives.

1. Create a conceptual model for DynaPRA implementation.
2. Develop a prototype DynaPRA framework using the existing IMM bone fracture models as a case study.
3. Devise ideas for integrating DynaPRA into the entire IMM.

DynaPRA IMM Prototype

Design reference mission. Based on the design missions created by the Constellation project, Mars was chosen as the case study reference mission for the prototype framework. The primary mission phases include the initial transfer to Mars, the stay on Mars, and the transit back to Earth. Both transits take 6 months. During this time the crew would be exercising, living on the ship and sleeping. EVAs would be rare and unlikely. The crew would also have a heightened rate of bone loss as compared to on Mars or Earth. The total stay length on Mars would be 18 months. The crew would spend their time exercising, living on the ship and sleeping and would likely spend a significant amount of time on EVAs (McElyea, 2007).

Event classification. Events are classified and separated into one of three classes. Class 1 events, as events that are not re-schedulable, include events such as launches and orbital insertion burns. With the critical mission events defined above, this makes the Class 1 events launch and entry, descent, and landing (EDL). Class 2 events, as required but re-schedulable events, include mostly mission events with a few medical events. These would include sleeping, EVAs, intravehicular activities (IVAs) and exercising in addition to certain medical conditions such as space adaptation sickness. Class 3 events are scheduled randomly and do not have to occur. This class contains the bulk of the medical events and a few mission events. The current Class 3 events are each of the bone fracture modules.

State variables. There are several variables that affect all parts of the simulation. These variables are called state variables because they account for the state, or status, of different parts of the system, such as the environment and the astronaut. The state of the environment, for example if the astronaut is on the Moon or on Mars, or the state of the astronaut, such as whether his hip is broken or not, affects probabilities of several events occurring along with the probabilities of the outcomes of those events.

The state variables are calculated at the beginning of the simulation before scheduling any Class 2 events. The astronaut state is first generated with a function called GenerateAstronaut. The state of the astronaut is stored in two structure arrays, or structs. The first is astronautState and the second is astronautAnthropometrics. The astronaut state was split into two structs due to the two distinct types of astronaut variables present. The struct astronautState contains astronaut

variables that can change; i.e., they are not static. Such variables include the BMD, current fractures, and organ states of the astronaut. The fractures and organ states are sub-structs of the main struct and contain the states of specific bones and organs. The organ state sub-struct does not currently exist, but would need to be implemented as the prototype is formed into the final simulation. After an event is executed, certain states of the astronaut can change, and these are updated within the astronautState struct. BMD is constantly decreasing with time until the minimum BMD is reached, and it is always updated. The current fractures are stored as Boolean variables and are updated as fractures occur. For example, if an astronaut hits his knee and shatters his kneecap, the fracture state for 'knee' is updated to broken. This could have serious repercussions for future events. The organ states could be stored as integers that represent the seriousness of the organ injury. For example, 0 could mean that the organ is functioning perfectly whereas 10 would mean that the organ is completely dysfunctional. The organ states would be updated as medical events execute and organs change condition.

The second struct for the state of the astronaut is astronautAnthropometrics. The anthropometrics are the static variables of the astronaut that either do not change or vary very little throughout the mission. Such variables are the height and weight of the astronaut, damping coefficients of certain parts of the body, and spring constants for certain fracture areas. These variables certainly affect the probabilities of events, however they change relatively little as the mission progresses so they are not constantly updated. They are, however, constantly called during events to calculate probabilities and to schedule appropriate events.

The BMD of the astronaut is an important state variable. It is one of the variables where time is a very important factor. The current BMD of the astronaut is constantly decreasing with time due to the less than 1 g environments. The rate of the bone loss is slowed due to the high amount of exercise that astronauts get in space. For the prototype, it was decided to make the rate of bone loss linear based on time, which can be changed for the final model. There are several factors that affect the rate of the bone loss, including where the astronaut is located for the mission (Mars, spaceship, etc.) and whether the astronaut is exercising regularly. If the exercise machine broke, for example, the rate of bone loss would increase due to the lack of exercise. It is important for the BMD to be updated before and after each event, because BMD is an important factor for such models as the bone models, where the chance of breaking a bone is increased when the BMD gets smaller.

The function BoneDensity constantly updates the BMD. It takes into account the state of the environment, such as where the astronaut is or the state of the exercise machine, in order to calculate the rate of bone loss. First, it is checked whether the minimum BMD has been reached. Each part of the body has a minimum BMD that cannot be gone below. If the minimum BMD for that part of the body has not been reached, then the BMD loss rate is found using the environment state variables. This rate is then used to calculate the current BMD. The current BMD for each specific area of the body is then outputted into the astronautState struct and the BMD has been updated for use in the next event.

The final state variable that is currently developed is the environment state. Of course, more state variables would need to be added as the prototype is converted to the final simulation. The environment state contains important information, such as where the astronaut is (Mars,

spaceship, etc.), the current gravitational acceleration, and the state of the exercise machine (whether it is being used or not). The environment greatly affects the probabilities of events throughout the entire simulation. It affects things like the rate of bone loss, the speed at which impact occurs for loading events, and whether or not certain events take place. The environment state is stored in a struct called environmentState and is not currently generated by a function at the beginning of the simulation. A generation function, however, is desired, and it probably would be wise to create one for the final model. This way, it would be easy to create all of the needed environment variables right away and have them at disposal for as soon as they are needed. Whenever events occur, the environmentState could be updated, such as if the astronaut moves his location from Mars to the spaceship or if the exercise machine goes from broken to fixed.

Framework. The frameworks for each event classification are separated into different functions. The Class 1 framework is the main program that is used to run the model. During initialization, the initial astronaut state and anthropometrics as well as the initial environment state are generated. The Class 1 events are then scheduled and the initial event is executed. The Class 2 framework is then run between the current time and the next Class 1 event time. After the Class 2 framework executes, the next Class 1 event is executed. This is repeated until there are no more Class 1 events to execute.

Class 2 events are scheduled between the start and end time of the function. The Class 3 framework is then executed between the current time and the next Class 2 event time. This determines if a random event occurred between the two time periods. After running the Class 3 framework, the program investigates the system to see if there were any significant consequences that occurred during the time period. Significant consequences for the models could be actual fractures occurring. This changes what types of Class 2 events will be executed. For instance, if an astronaut breaks their hip, all EVAs most likely will be postponed until the wrist heals. EVAs would then have to be re-scheduled, canceled, or given to another astronaut. If no significant consequences occurred during the time period, the next scheduled event is executed. If a significant consequence does occur, different Class 2 events are scheduled during the remaining time period based on the consequence. This process repeats until there are no more events to execute in the level 2 event queue.

The Class 3 framework functions in a similar manner to the Class 2 framework, and is run over a time period. It starts by scheduling level 3 events between the start and end time periods. It then executes these events until there are either no more events to execute or a significant consequence occurs. Figure 1 shows the overall sequence of events for the entire framework.

Scheduler. The scheduler is the primary component to a functional DynaPRA model. One of the key attributes of DynaPRA is its ability to pursue all possible avenues of investigation by scheduling its own event trees dynamically. This requires a recording and sampling of its current and past state, combined with a robust algorithm that most accurately reflects reality when planning future events. The structure of the scheduler is divided into three tiers that each plan a class of events in a hierarchy based on requisite information. The first layer of the scheduler plans the Class 1 events. These are assigned at the mission beginning based on a mission plan.

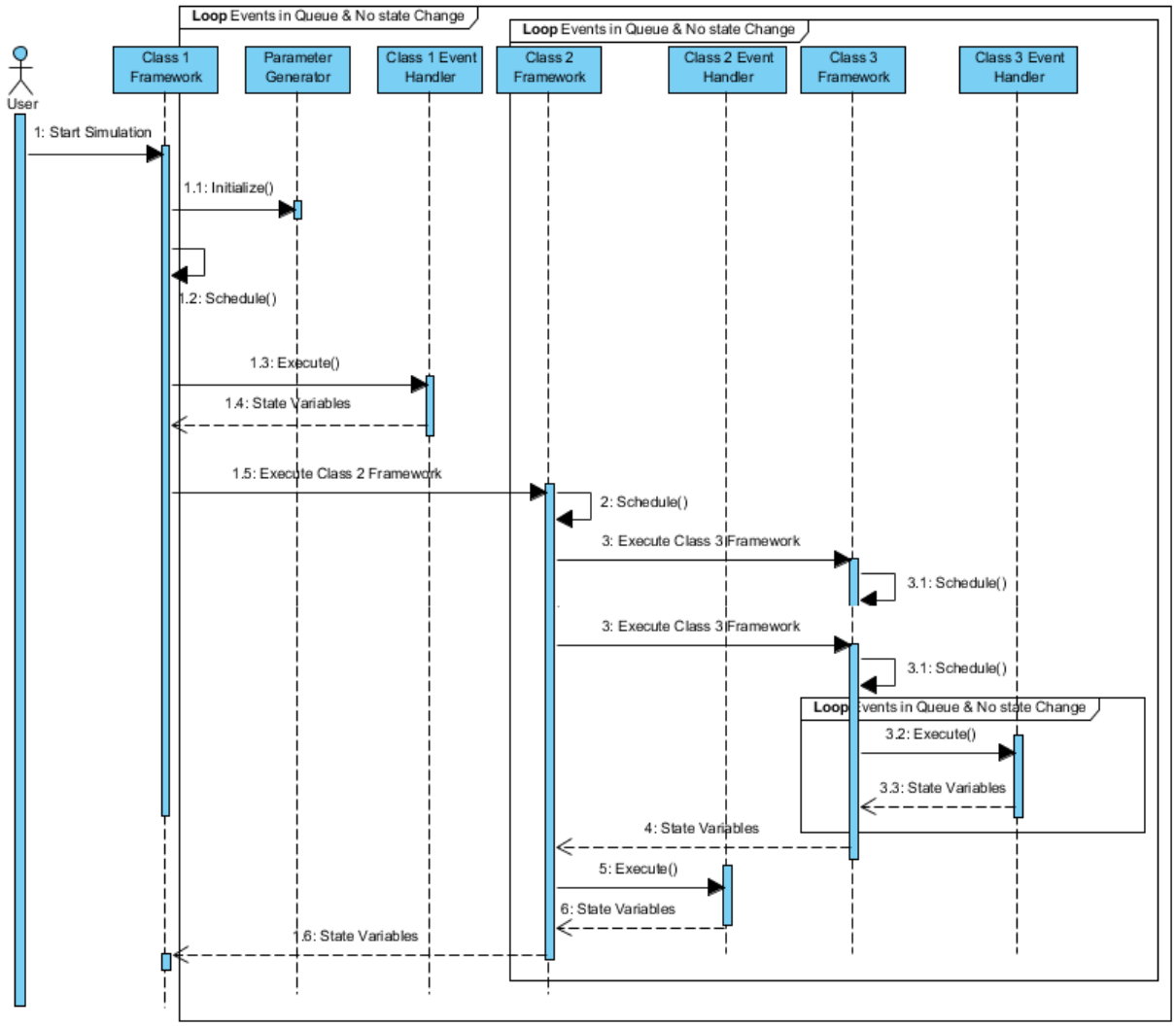

Figure 1. Framework Sequence Diagram - UML Sequence Diagram Showing the Flow of the Overall Framework.

The discrete event occurrences do not actually effect a change in the health of the astronaut. Rather, it changes the environment state in a way that is fixed until the next Class 1 event. Therefore the first layer is scheduling ranges or zones of fixed environmental state parameters within which you can schedule Class 2 and Class 3 with confidence, because between the Class 1 events the parameters affected by Class 1 events are not altered. The Class 2 scheduler works in much the same way as the Class 1 scheduler, in that the events denote a change in state, between which Class 3 events can be scheduled with confidence.

Class 3 events are presently discrete medical events that denote a change in the astronaut's state. The structure of the hierarchy was built from the core outward, based on what was needed at the fundamental level to properly schedule medical events, and what was required from the outer layers. The Class 3 events needed to be scheduled randomly, both in their occurrence and the time at which they occur. Multiple methods were considered, and are discussed further below. For the initial structure, the Class 3 events are built from modules - in this case the 3 bone fracture models. These models are then converted into two separate modules. One module is a

stripped down version of the former bone fracture model which calculates the probability of a bone fracture from the given astronaut and environmental parameters, such as body mass, BMD, fall height, gravity, etc. and generates a probability of fracture given the impact event. This percent chance is then evaluated, and the module returns with whether or not a bone fracture occurred.

The second module is an "event chance" module which divides an arbitrary span of time into discrete slices of arbitrary resolution relevant to the medical model in question. The module determines the probability of an impact capable of producing a fracture during each one of those time slices based on the environment state and astronaut state. A simple vector of random values, of length equal to the number of time slices is created, is compared with the vector of probabilities. At every point at which the random number is less than the probability percentage, an event is determined to occur and the medical event is scheduled. This combines the ability of time-dependent event probability, such as fatigue, to be referenced as a function and overlaid on otherwise uniform probability distributions. The choice of this system is discussed further below in comparison with a Poisson distribution. This process is paralleled by every medical event model, creating a timeline of all medical events that occur during an arbitrary time. All medical events determined to occur are placed into a queue system and executed by their respective medical modules in chronological order.

If a medical event is determined to occur, or if the astronaut or parameter state is changed in some way, then the remaining schedule of queued events may no longer reflect the reality of the situation. At this point, after the state is changed from the consequence of the medical event, the event probabilities for the remaining time are recalculated and a new timeline of events are queued up to be executed.

While the Class 3 scheduler can work for any arbitrarily small span of time, it is more efficient to schedule as far ahead as possible at once. This efficiency is countered by the growing likelihood of a need to recalculate the probabilities due to a state change, which renders the scheduled events beyond that point in time potentially invalid and therefore unusable. With only these two factors, the arbitrary amount of time scheduled would be determined by the sum of the probabilities of medical events multiplied by the chance of the medical event having consequence. Due to the very minute chance of a medical event with our current number of models, and empirically demonstrated to be the case no matter how many models are integrated by the low medical event rate in space, a third factor that defines the arbitrary time pops up long before an equilibrium of efficiency can be reached. Class 2 events, which by their nature are guaranteed to change the environment state, and therefore require recalculation for every event than follows, occur several times per day.

Class 2 events act as a bridge between the completely rigid Class 1 events and the completely fluid Class 3 events. While Class 3 events can be executed, and require rescheduling of Class 2 and Class 3 events on relatively rare occasions, Class 2 events require constant analysis of scheduling. The complexity introduced is that Class 2 events are beholden to a third set of parameters or 'mission state' which would be made up of other relevant factors influencing the scheduling of EVAs, sleep, exercise, and other day to day activities. Because far more abstract factors go into this scheduling, several assumptions have been made to facilitate a plausible

Class 2 scheduler with room for improvement or overhaul while allowing the rest of the simulation to be tested in the interim.

The parameters and arbitrary factors behind scheduling Class 2 events represented a large degree of complexity that need to be simplified. Class 2 events are more difficult than Class 3 events for two primary reasons. The first reason is that the best reflection of reality of the schedule of class 2 events relies on some form of intelligent decision making, rather than probabilities that only account for the chance of an event. The second reason is the interdependence between Class 2 events. Class 3 events did not influence each other while being scheduled in mass, because they only alter each other based on a rarely realized potential of consequence once executed. Class 2 events by contrast must always reflect the occurrence of all other Class 2 events, because every Class 2 event has consequence simply by occurring. Whether or not an astronaut has an EVA on a given day is always going to affect his sleep schedule or exercise routine. This creates the difficulty of scheduling Class 2 events in mass because of the nondeterministic nature of the schedule. Modeling, or at least simulating, a realistic scheduling process is a difficulty that has yet to be achieved.

The Class 2 Scheduler currently works as an aggressive schedule of predetermined goals based on a continuous model of time. Class 2 events currently include both in-transit and Mars variants of Sleeping, Exercising, and EVAs, as well as IVAs which are equivalent to a default position on non-specific activity. At the beginning of the mission, a data queue specific to each Class 2 activity is filled with occurrences of the desired event, including a specific alphanumeric denomination of the event, the duration of the event, the first point in mission time the event can be scheduled, and the last point in time during the mission it can be scheduled.

The queue automatically takes each list of each type of event and organizes it by priority. Priority is given first to the events whose window of opportunity for scheduling closes first, which minimizes the number of missed events. Secondary priority is given to the longest duration events, thereby front-loading work. Events whose windows of opportunity have closed are deleted. When the queue is prompted with a specific command, it will then output the top priority event for the Scheduler to schedule, and delete it from memory.

This structure is well suited for events like EVAs, which are pre-planned, finite, and have a window of opportunity in which they can occur. The structure can also easily be manipulated for events like Sleep, which are generally homogeneous and have no unique windows of opportunity. The Sleep queue is preemptively filled with an excessive number of sleep events, with a window of opportunity stretching from mission start to mission end and equal durations. Without creating extra framework, this allows sleep to be scheduled by no other factors that the convenience and desire to sleep, where the number of events achieved is unimportant. Exercise is handled similarly. The queue system also leaves room for additional Class 2 events to be scheduled during the mission in future iterations of the model.

The queue structure permits the Scheduler to now determine if “Astronaut would like to sleep now” or “Now is a good time for an EVA on Mars”. Once determined, the queue can be called, and an event is loaded into the events for the day.

The second part of the Class 2 Scheduler is deciding what the priority activity is at the particular moment in time. Each type of Class 2 event is assigned variables for a sigmoid function; (1/(1+exp(-x)). This function simulates the priority of events as a decimal from 0 to 1 based on x time after the last event of its type began. In the case of sleep, immediately following a sleep there is virtually no priority to engage in another 8 hours of sleep. This non-priority starts to become a low priority after 18 hours or so, where it becomes a small but slight priority. The next several hours have a steep slope of increasing priority, where after 24 hours it becomes a high priority. As time goes on to be 28 or 32 hours, the priority maxes out at asymptotically approaching 1.000, reflecting the relatively absolute need to sleep. A sigmoid function is able to model this very well, and can be adjusted by several variables. Each Class 2 event type has a set of parameters that define when and how quickly the priority for the event changes.

After the values are achieved from the sigmoid function for each event type, the Scheduler determines what events occur for the next week. The top priority event is scheduled first; time is moved forward by the duration of that event, and then the new highest priority event is chosen, and the process repeats. Every time an event is scheduled, it is pulled from its queue, and that data is placed into a Class 2 schedule queue. The scheduler schedules out an arbitrary amount of time, currently events to fill a week, which executes them in chronological order, updating the time and environment state as they are triggered. Between each execution, the Class 3 scheduler is called, and the random events are analyzed before returning to the Class 2 Scheduler for the next Class 2 event to occur. If a Class 3 event changes the state, the remaining Class 2 events in the queue are returned to their respective queues, and the Class 2 scheduler re-plans the events from that point forward.

Emulating this type of decision making efficiently is something not yet achieved by our model. This framework may have potential for the final Class 2 Scheduler, but currently it acts as a place-holding function that facilitates the necessary functions of the Class 2 scheduler so that the rest of the model can be run for testing purposes. The Class 2 schedule produced by the scheduler cannot be verified as reflective of reality, or even of a structure that is capable of reflecting reality. Many questions still remain with exactly how the Class 2 Scheduler should function. The resolution of detail, such as whether an EVA occurs at a specific hour, or that it occurs in the morning or the afternoon or simply that it occurs today, are questionably relevant to a useful model. As such we predict the Class 2 model will be the most volatile scheduling function as this project is continued, where it will likely be adapted based on what time-specific medical events are introduced and how complex of mission scheduling algorithms are desired to be employed.

Discussion and Future Work

General improvements and changes. The efficiency of the model should be taken into account both during the design process and during an optimization process following the completion of a working DynaPRA model. A large magnitude of variability due to the near-infinite potential event sequences becomes problematic for analysis. Currently the model is designed to create results that can be analyzed as a Monte Carlo simulation. It is likely that the model itself will be run hundreds of thousands of times for a useful distribution of results. As the DynaPRA model is expanded with further IMM models, the complexity will grow to the point where several more magnitudes of mission trials are necessary, potentially in the realm of

millions or billions of trials. Within each mission is a repetitive structure of scheduling events, followed by executing events, followed by rescheduling events. Each function within the program will be called hundreds or thousands of times, and potentially more. This program was never meant to be able to deliver quick results, and at its full potential will likely spend days or weeks to deliver useable distributions of medical events and mission risks. With functions potentially being called on the order of thousands of times, every small change in execution time for each function will yield a large change in runtime for the program, and a massive change in runtime for the Monte Carlo results.

Framework. While the framework that was created this summer is a functional prototype, there are still improvements that could be made. First, most of the required functionality should be left up to the developer of the module. A common framework should be developed and inheritance utilized to allow for common class definitions and function calls. Each module could then be stored in a data structure for easy access and processing. For instance, each module could have a schedule method that either schedules that event occurring or returns the time that it occurred. The function in the framework that calls the schedule method could then iterate through the data structure calling the schedule method for each item. The framework would not need any knowledge of the inner workings of each module and would therefore be generic. Adding new modules would be simplified to updating the state variables and adding the new module object to the data structure.

Challenges with data requirements. Due to the complex nature of DynaPRA techniques, accurate DynaPRA models require a large amount of data on event occurrence and interactions to be collected and analyzed. In physiological systems, this is even more true as every internal and external factor has an effect on the state of the body. In order to have a valid system, data would need to be collected on all of the interactions between the environment and medical events that are being modeled. Additionally, because of the low number of medical events that occur in space and the limited amount of time spent in space, this data cannot be collected on the actual system. Some of the data can be collected from medical studies of the population at large but that wouldn't take into account the effects of microgravity. Eventually, estimations, assumptions, and abstractions will need to be made in order to compensate for missing data.

Integration with the IMM. The next step for this project is to analyze additional IMM models and plan for their integration. The current framework is structured with versatility in mind, but the only tested models are the bone fracture models. Continuous aspects, such as a continuous medical model like the Renal Stone model, have yet to be extensively considered. It is likely an expansion of the framework will be necessary to accommodate continuous processes. Integrating them with the Scheduler and the more discrete medical events will likely be handled by discrete appearances of symptoms, which will be tracked in the astronaut state and considered when planning Class 2 events.

References

Griffin, DeVon. *ISS and Human Health Office: Exploration Medical Capability, IMM.* n.d. http://microgravity.grc.nasa.gov/SOPO/ICHO/HRP/ExMC/IMM/ (accessed June 2012).

McElyea, Tim. *Project Constellation: Moon, Mars and Beyond.* Burlington: Apogee Books, 2007.

Nelson, Emily S., Beth Lewandowski, Angelo Licata, and Jerry G. Myers. *Development and Validation of a Predictive Bone Fracture Risk Model.* National Aeronautics and Space Administration, 2009.