Human Interactive Mission Manager:
An Autonomous Mission Manager for Human Cooperative Systems

by

Jason M. Furtado

S.B. Computer Science, M.I.T., 2007

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

June 2008

Author_____
Department of Electrical Engineering and Computer Science
May 23, 2008

Certified by_____
Lauren J. Kessler
Principal Member of Technical Staff, C.S. Draper Laboratory, Inc.
VI-A Company Thesis Supervisor

Certified by_____
R. John Hansman, Jr.
Professor of Aeronautics and Astronautics
M.I.T. Thesis Supervisor

Accepted by_____
Arthur C. Smith
Professor of Electrical Engineering
Chairman, Department Committee on Graduate Theses

[This page intentionally left blank]

Human Interactive Mission Manager:
An Autonomous Mission Manager Framework for Human Cooperative Systems

by
Jason M. Furtado

Submitted to the
Department of Electrical Engineering and Computer Science

May 23, 2008

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Facilitating low level human supervisory control of mission management is highly challenging because of concerns regarding system stability and performance. Previous implementations of mission managers based on C. S. Draper Laboratory's All-Domain Execution and Planning Technology (ADEPT) are based on an architecture that can be verified to act deterministically with scripted human interaction opportunities. This thesis describes the Human Interactive Mission Manager (HIMM), a general software architecture to facilitate human supervisory control level of interaction based on ADEPT. The HIMM provides operator insight and mission designer interaction mechanisms. These features provide interaction in a controlled but asynchronous way as a baseline service of the HIMM system. The design separates the information used by the operator from the data used by the mission manager so that the addition of asynchronous human interaction will not adversely affect normal execution. To explore the interaction mechanisms and exercise the system, the software was applied to a space domain application. This prototype system facilitates asynchronous input from a human operator to the mission manager.

Thesis Supervisor: Lauren J. Kessler
Title: Principal Member of the Technical Staff, C.S. Draper Laboratory, Inc.

Thesis Supervisor: R. John Hansman
Title: Professor of Aeronautics and Astronautics

[This page intentionally left blank]

Acknowledgement

May 23, 2008

I would like to thank Lauren J. Kessler for her guidance and support as my Industry Supervisor at Draper Lab for the past three years. She has been both a mentor and friend, and has given me invaluable advice that will help me for years ahead.

I must also thank my MIT thesis advisor, Professor John R. Hansman, for challenging me throughout this project through lively discussions and debates which helped me to view my project from different perspectives. His views helped me frame and structure my thesis ideas in a logical and informative manner.

To Emily Braunstein and Michael Ricard: many thanks for taking the time to read and edit my thesis. Also, thank you to Brian Collins for his programming help on the LPR algorithm.

Finally, to my family and friends: I want to thank you for your love and support. And to my parents, I especially dedicate this thesis to you for all your years of hard work and the sacrifices made to make my achievements possible.

This thesis was prepared at The Charles Stark Draper Laboratory, Inc. under Internal Company Sponsored Research Project 21819, Human Interactive Mission Manager.

Publication of this thesis does not constitute approval by Draper or the sponsoring agency of the findings or conclusions contained herein. It is published for the exchange and stimulation of ideas.

_____

(author's signature)

[This page intentionally left blank]

Contents

[This page intentionally left blank]

List of Figures

[This page intentionally left blank]

List of Tables

[This page intentionally left blank]

Chapter 1

1. Introduction

1.1 Overview

As complex software systems move into all aspects of human life and work, the interfaces between man and machine gain more importance. Computation is superior to the human brain in rule based tasks, while humans are able to perform knowledge based tasks that computers cannot. As decision-making computer systems have become more reliable, people are accepting them as decision aids. However, these systems cannot yet perform the full functional spectrum of operation and are reliant on human operators as an integral part of the control loop. To grant a complex system the flexibility to adapt to outside human input in a manner that still ensures system stability is a serious design challenge. The Human Interactive Mission Manager is a system designed to make asynchronous interaction with autonomous vehicle systems seamless to a human operator.

1.2 Motivation

The purpose of the research presented in this thesis is to influence and control autonomy for vehicles in such a way that better leverages both human and computational strengths. Complex systems have been successfully deployed to plan and execute missions in uncertain environments; such systems have focused their design on reacting to unforeseen changes and events in a closed system where human interaction is sparse. No such Draper Laboratory

planning system has been designed where facilitating human interaction was a fundamental requirement. Therefore, the Human Interactive Mission Manager includes a simple way for an engineer to both enable and constrain the interaction to ensure system stability. Such a system adds a new layer of design freedom and responsibility for algorithm and mission designers alike. They will now have the freedom to explore opportunities for interaction to increase system robustness, as well as the responsibility to tackle the challenges that may require human and computer cooperation to achieve.

System stability is a major concern for software used to control autonomous systems in environmental domains including space, underwater, terrestrial and aerial applications. Autonomous systems are focused on controlling vehicles where the assets are expensive and instability could lead to mission failure, vehicle loss, or, in the case of a manned vehicle, even injury or death. Allowing for human interaction adds another variable to the system that could introduce a failure mode. Since the adoption and financing of such systems is contingent on minimizing errors, such systems are often designed with limited interaction. As these systems have matured, computational resources have increased and the problems autonomous systems are charged to tackle have become more complex. The advance of computational technology has made interaction become more feasible and will be necessary for success in future applications.

Extensibility is a basic requirement of the HIMM design, since the goal of human interaction will be achieved as an addition to the baseline functionality of the mission manager. The system will be easily applicable to different kinds of missions, vehicles, and domains without changing the underlying engine of the driving software. Such a generalized framework is powerful since it allows mission and algorithm designers to incorporate interaction into applications and better leverage available human resources. The baseline interaction model can

grow and become more robust as it is tested in different applications and as these designers become more comfortable and confident using this technology. A narrow design solution to the interaction problem would not hold much promise in creating a technology that supports exploring the incorporation of human interaction as a fundamental component of a vehicle mission.

The human decision making process is not well understood and seems to be influenced by intangible variables that are not quantifiable. Although computation can react almost instantaneously to its environment by analysis of multiple sensor inputs, computation does not handle non-quantitative decision making as naturally. An example of these non-quantifiable variables is the process of identification of an interesting area to explore. Autonomous software must make use of the information available and its own decision model to make choices, but could utilize human input to provide input in cases where the best choice is not clearly defined. In these areas, humans are better equipped to decide the course and should be given input into automation decisions.

1.3 Goal

The goal of the Human Interactive Mission Manager (HIMM) research is to augment an existing mission manager design to include human interaction as a baseline service of the system. The All-Domain Execution and Planning Technology (ADEPT) technology designed at the Charles Stark Draper Laboratory (Draper Laboratory) describes the functionality of a hierarchical mission manager that can control an autonomous system and react to changes in its environment and plan. The HIMM project does not aim to reinvent the control theory and structure for a mission manager but rather to adopt the ADEPT technology and leverage a

suitable existing implementation of ADEPT for extension. The HIMM implementation must be domain independent, and must have a modular and object oriented design. The implementation's baseline services must be independent of its operational domain so that there is separation between the code which is specific to the vehicle and mission, and the code that actuates the mission. Facilitating human interaction in a modular and extensible way as a fundamental service of the mission manager is the primary goal of the HIMM project. These goals must be achieved without significantly adding to the computational complexity of the baseline system in a way that would adversely affect system performance.

1.4 Structure of Thesis

- Chapter 1 motivates the thesis, describes project goals and explains the structure of the thesis.

- Chapter 2 focuses on the background and previous work relating to autonomy and the different levels to which humans can be involved in an autonomous system. A discussion of the ADEPT technology follows, describing the theory of the control structure for the autonomous system as well as where human interaction fits into the ADEPT technology description.

- Chapter 3 presents the requirements and design of the HIMM. These requirements motivate an in-depth exploration of the previous software implementations of ADEPT that led to the design chosen as the baseline mission manager implementation for this thesis work. Following, there is a high-level description of the HIMM system design including data flow descriptions, and information on how data is maintained and updated. This chapter then delves into a low level discussion of the Kernel and External

Coordinator implementation. Lastly, there is an explanation of how this system adds a new layer of design to the algorithm and mission creation tasks in order to take advantage of the new capabilities the system can provide.

- Chapter 4 describes the implementation and organization of the general system as well as the application of the framework to a specific example. There is a characterization of the HIMM system's interaction mechanisms in order to explore the costs of adding a layer of interaction to an ADEPT implementation. This chapter also illustrates how parameter changes can affect the plan depending on how the algorithm and mission are designed and implemented.

- Chapter 5 concludes the thesis and describes areas of future work to expand the capabilities of the HIMM.

Chapter 2

2. Background

2.1 Software in Autonomous Vehicles

Software's tasking in systems is along a vast spectrum. Some tasks are simple, such as automating data analysis that would otherwise be done by a human. Some tasks are computationally complex, but are not decision oriented, such as keeping an airplane at a constant altitude. The research presented in this thesis is focused on high level mission planning for an autonomous vehicle. These systems are tasked with complex missions made up of smaller tasks whose progress is analyzed to evaluate the status of higher level goals.

Autonomous vehicles must be able to deal with the uncertainty of the real world. As unexpected constraints arise and environmental changes occur, the system adjusts its plan or makes an abort decision. The system makes a decision to abort if system resources are insufficient or conditions are not safe enough for the vehicle to complete its task. Just as humans use their senses to sense their environment and judge their progress, these vehicles use multiple sensor inputs to provide information about the world around them. A decision on how best to proceed based on the information available is not always clearly defined by a simple equation or a set of rules. In a completely autonomous system, there is no human input and the computer must make a decision based on the logic programmed into the system. In some cases, it would be beneficial to the systems operation to allow for some open ended decision making to come

from a human. This thesis aims to explain how such software can incorporate human input and how this affects the process of algorithmic and mission design within the system.

Algorithms designed for autonomous systems are generally self contained black box utilities that can be used by mission planners for a particular calculation. A set of inputs are provided, and the algorithm provides some output. Many times there are a variety of parameters used by the algorithm that could take on a range of values but are pre-set prior to system operation. This is an example where a human could adjust system operation to produce valid but slightly different operation of a system. How much control and the mechanisms with which the human is given to influence the direction the software pursues fall into a number of categories. There are cases when each mechanism is uniquely appropriate, as is explained in the next section.

2.2 Levels of Autonomy

Thomas Sheridan of MIT studied the spectrum of control modes that exist to balance control between human and machine [1]. Figure 1 illustrates the range of roles that the human and computer can take on in a system. Computer systems can operate completely manually which grants the human control of all the activities the system performs. On the opposite end of the spectrum, systems can run completely autonomously where the system performs all of its actions without any human input. Between these extremes, there are multiple types of human interaction with autonomy that demonstrate a varying degree of human control leveraged over the actions of the autonomous software. The HIMM focuses on enabling a supervisory level of control which can manifest as human management by consent, timeout and exception. Each

version of supervisory control places a different amount of responsibility on the human operator [1].

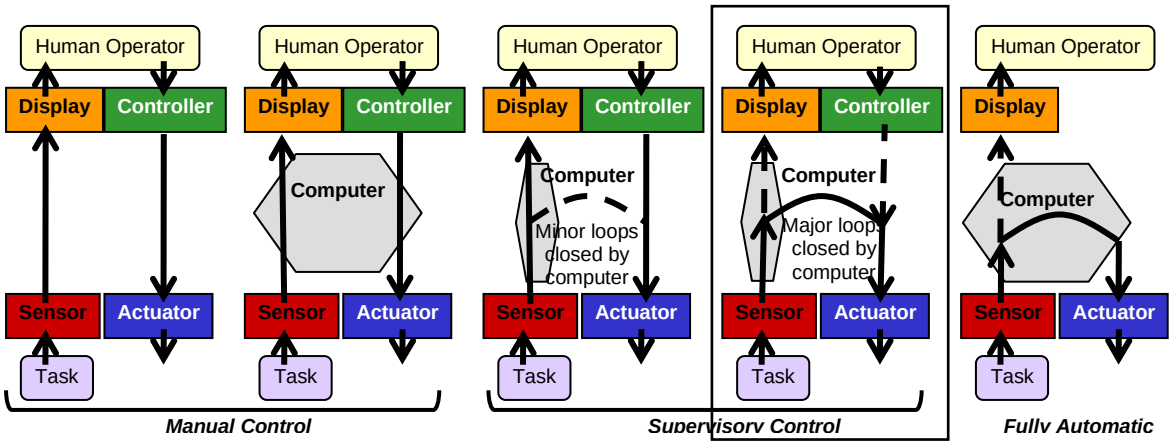

Figure 1 Levels of Autonomy [1]. Computer autonomy increases from left to right. The HIMM is aimed at facilitating a supervisory control level of autonomy highlighted by the box above. At this level, major control loops are closed by the computer.

There are system operations which must be monitored closely but may still rely on computer autonomy for success. An example of such an operation is whether or not to abort a mission. To provide the appropriate control to the human operator, the system prompts the operator for consent prior to performing its task. This is referred to as management by consent, and it grants decision making authority to the human, but leverages the abilities of the computer to perform intricate tasks. This level of autonomy is closer to manual control of the system since the computer can take no action without human input.

There are occasions when action is more important than human control and human indecision is more dangerous than computer autonomy. These occasions arise when time becomes a limiting factor. Operations where time can affect the success or failure of a mission warrant a different kind of human oversight on the system. In this form of interaction, an opportunity for human input exists in the form of a time window. The operator maintains

executive control of actions taken by the machine and only forfeits them when action is not taken within that window. Management by timeout gives the decision of how to proceed to the software only if the human fails to act in the predetermined time period. The balance of control shifts to the autonomy when the authority to act on its own without first asking for permission to do so is given to the computer.

Levels of control where the software has priority over decision making are autonomous, but there is still a place for a human operator beyond maintenance tasks. Some systems look for human input in exceptional cases when the correct course of action cannot be clearly determined based on numbers or equations. For instance, a nuclear power plant that runs into problems may give a human operator the opportunity to shut down the system if remedial activities are not ameliorating the situation. Management by exception is a level of autonomy where the system completes its tasks without human input, but it also has the option to defer control to a human operator if it deems it necessary. The human may also have the ability to abort an operation at any time for safety reasons. Not all instances clearly fit into a single control category and a hybrid distribution of control between human and computer may be employed to achieve a particular control structure [2].

The HIMM facilitates all the described types of interaction and lets the algorithm and mission designers decide what kind of control to leverage for a particular parameter of the system. Some systems may simply run autonomously while the human only monitors for safety purposes. The HIMM is designed for systems that will need human interaction at different levels of control for different types of decisions, and allows for input parameter adjustments by the operator. The ability to leverage multiple types of interaction allows appropriate control to be distributed in each case.

2.3 ADEPT Architecture

The All-Domain Execution and Planning Technology (ADEPT) architecture was developed at Draper Laboratory for autonomy. Autonomy is defined for the purpose of this architecture to be the ability to plan and execute in a rapidly changing environment. In order to achieve any task in such an environment, the system must be designed to recognize when changes to a plan must be made and adapt dynamically in real time. Missions are designed by a human, and are subsequently executed by adapting to the conditions in which it is operating [3]. The ADEPT technology is strongly associated with the military's Object-Orient-Decide Act (OODA) loop [4]. This loop describes planning and decision-making nodes that are made up of modules that perform situation assessment, plan generation, plan implementation and coordination. The ADEPT architecture incorporates these elements into its design but extends this idea into a hierarchical organization of such nodes. The basic building block (a planning and decision-making node) in the ADEPT framework is the ADEPT autonomous system [3].

2.3.1 Functional Decomposition

An autonomous system must have the capability to sense its environment, reason based on the information it gathers, and then take action to achieve its goals. The ADEPT technology attempts to encapsulate this process with functionality referred to by the following modules (parts of the functional block): Monitoring, Diagnosis, Plan Generation, Plan Selection, Plan Execution and Control, and Internal and External Coordination. Figure 2 displays the functional decomposition of an ADEPT autonomous system into its modules. Internal Coordination facilitates information flow between the different modules of a specific system, while External Coordination refers to the information flow with entities outside of the system (i.e. other

systems). The set of modules taken as a whole is the basic functional building block, out of which ADEPT complex autonomous systems are built.

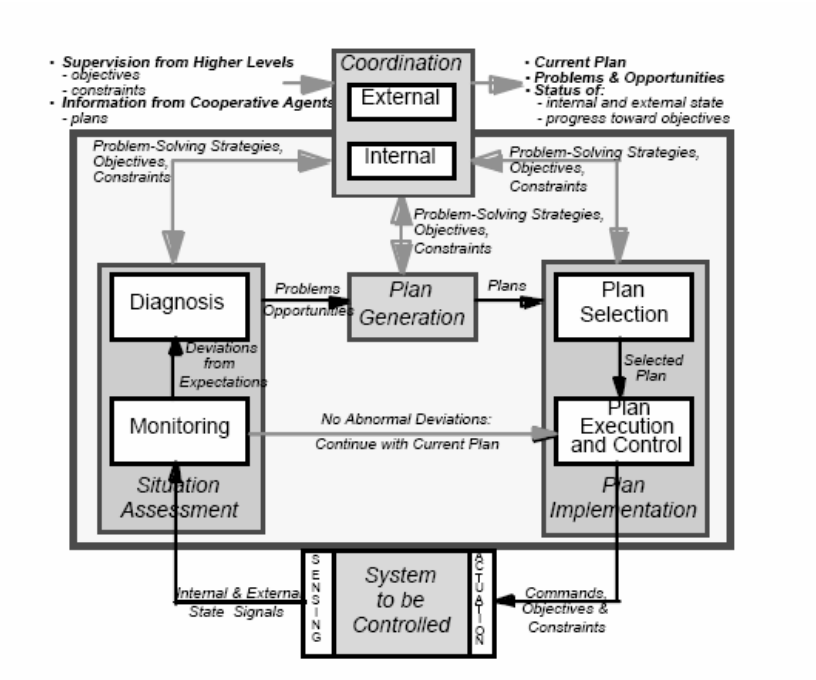

Figure 2 Functional Decomposition of a building block for an Autonomous System [3]

When observing an autonomous system, its behavior performing its mission can be captured by a single functional building block as described in Figure 2. However, each smaller sub-task of the larger mission can also be encapsulated by an individual functional building block. A single building block that encapsulates all the necessary actions to complete a complex mission would be intractable to create. Instead, the problem is broken down into smaller parts which can each be handled by different functional blocks. These functional blocks working together to form the mission will be referred to as activity objects for this thesis. Each activity object encapsulates the reasoning necessary to accomplish its task by itself, or the knowledge to break down its problem into smaller tasks [3]. The mission designers are responsible for breaking down a high level task into sub-tasks and specifying what and how each activity object

will attempt to achieve its objective. The algorithm designer, another important person in activity object implementation, concentrates on creating software modules that actually do the processing to perform the task of the activity object.

2.3.2 ADEPT Activity Object

The basic architecture of an ADEPT activity object, also known as a "four box activity" is shown in Figure 3. This figure shows decomposition of an activity object into four of the functional modules mentioned previously, namely: monitoring, diagnosing, planning and execution. The relationship between these modules is described below. The individual activity objects do not completely encompass the remaining two functional modules, Internal and External Coordination. Those responsibilities lie within the kernel which maintains the structure between activity objects and actuates the four boxes of an activity object.

After an activity object is given its task, it can then create a plan to accomplish its work. Figuring out a path of travel or creating sub-activity objects to achieve higher level tasks are examples of the types of planning completed by an activity object. Execution then performs the plan by sending commands to the system to actuate. Monitoring then takes the data from the system's sensors and decides if the system is progressing towards the activity object's goals as expected and looks for new opportunities. If the current plan is not sufficient to achieve its goals on time, the diagnosis component determines the limiting factors. This information is then forwarded to the planner, which uses the data to create a new plan [3].

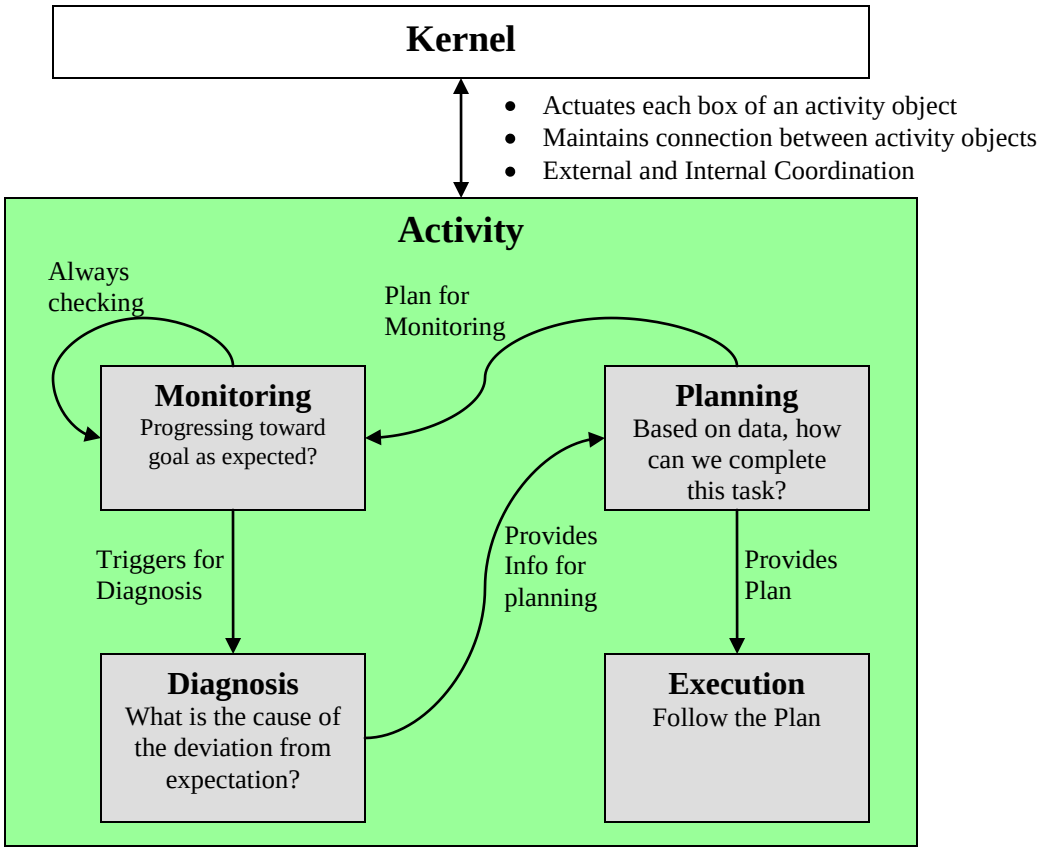

Figure 3 Four Box Activity and Kernel Responsibility Distribution.

2.3.3 Hierarchical and Temporal Decomposition

The ADEPT technology breaks down complex problems into smaller sub-problems to solve the larger problem. The task accomplished by an activity object is accomplished solely by the actions of one specific activity object if the task is simple enough. On the other hand, if the task is complex, it will be broken down into smaller parts that are assigned to sub-activity objects. High level activity objects perform their tasks by sending commands and monitoring progress, and by spawning children activity objects that will perform the sub-parts of the task. Lower level activity objects perform their tasks by sending commands and monitoring progress locally. The parent activity object will then monitor the status of the children and perform higher

level decision making for the conceptually higher task. Figure 4 shows an example of a decomposition to perform the everyday task of doing laundry.

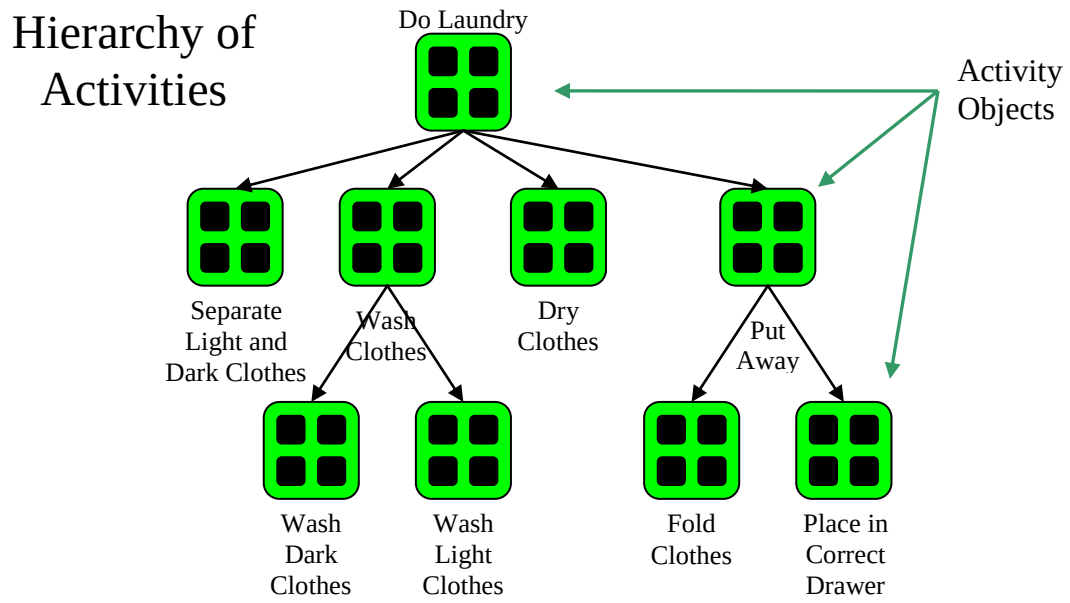

Figure 4 Task Hierarchy – Activity Object Breakdown. An illustration of a problem decomposition of the example high level task of doing laundry broken down into smaller sub-problems that make up the higher level activity object.

Activity objects are combined into a hierarchy, where the top most activity objects have the greatest temporal scope (planning horizon), but the least solution detail. Contrarily, the lowest level activity objects have the smallest temporal scope (planning horizon), but the greatest solution details. The higher level activity objects are responsible for monitoring the progress of their children in the context of their higher level objectives. Figure 5 graphically depicts these relationships between levels in the activity hierarchy or tree with solution detail and planning horizon. Lower level activity objects are generally concerned with more concrete tasks that translate into commands to the system. Its high level counterparts focus on organizing and planning complex tasks made up of smaller parts.

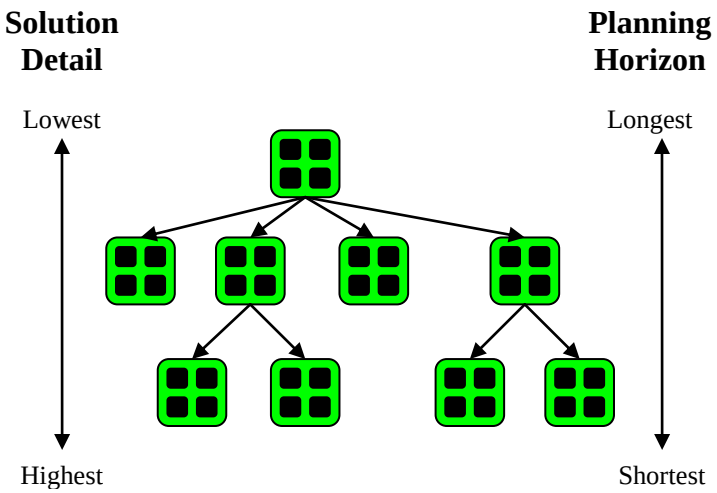

Figure 5 Characteristics of Solutions at Different Levels of the Hierarchy.

The activity hierarchy is a temporal organization of the activity objects of the mission. At each level of the tree, activity objects are ordered chronologically based on when the activity will be executed (becomes active). The activity objects that are active send commands to the vehicle and monitor progress to see if they have completed their task. The activity objects in the left most branch of the hierarchy are active and when an activity object completes its execution, it is removed from the tree thereby making a new set of activity objects active [3].

2.3.4 Previous ADEPT Implementations

The ADEPT architecture enables high level reasoning and adaptive behavior which is used to control autonomous vehicles; the technology can also support human interaction during execution. This architecture has been applied successfully to vehicles in the air, undersea and space domains. The basic framework of the technology has been in development since the mid 1980's, focusing on improving planning to support autonomy [5]. ADEPT has been utilized by a NASA program for satellite operations [6], by a DARPA program for mission planning and execution of multiple aircraft on multi-day campaigns [7, 8], by the Office for Naval Research (ONR) for ship mission based ISRT (Intelligence, Surveillance, Reconnaissance, and Targeting)

missions [9] and more [3]. The focus of such projects has been on functional mission execution and to verify stability and safety of the software. Activity objects in these systems must react to changes in the vehicle's environment such that the software system keeps the vehicle out of danger. Current applications of the Draper Laboratory mission management technology have not required incorporating the human into asynchronous, real-time decision making. The HIMM project builds upon previous work to explore the problem of creating a human-interactive mission manager that provides the mechanisms for an operator to dynamically participate in real-time mission planning. The findings will then be used to support a different set of missions where a human operator and automation software can cooperate.

2.4 ADEPT Adaptation for Human Interaction

External Coordination within the ADEPT framework is specified in broad terms, but does mention input from a cooperative agent. The HIMM system leverages the human operator (cooperative agent) as a source of external input as part of the responsibility of external coordination. External coordination also covers the communication between ADEPT activity objects in the hierarchy. This type of intra-activity communication has been accomplished in previous implementations by having the activity objects be able to send information up the hierarchy to its parent. The HIMM is focused on allowing for human input into an activity object's parameters to affect the mission and to utilize the knowledge and understanding of the human operator. The ADEPT technology description does not discuss what mechanisms should be employed to enable interaction or what form it should take, but it does mention that outside input is a part of the responsibilities of coordination in the ADEPT design. The HIMM extends outside input to include human input [3].

Chapter 3

3. Design of System

3.1 Approach

In order to design and implement the Human Interactive Mission Manager (HIMM), a structured approach was used to ensure a successful system. The first step was to determine exactly what the HIMM project must accomplish by specifying system requirements. This list of requirements guided the direction of the research and molded the final system design. With the requirements completed, a design was created of the HIMM system structure and interaction with the Draper Laboratory's ADEPT mission manager technology. Solving this interaction design problem before deeply exploring existing ADEPT system implementations allowed for the problem to be solved without the past overly biasing the solution. These two foundational steps led to an exploration of existing technologies to find an ADEPT implementation that could be leveraged to create the HIMM system. Using the metrics defined by the requirements and the HIMM high level architecture, an ADEPT implementation was chosen. The work on the final design and implementation of the HIMM proceeded, and was subsequently evaluated against the requirements.

3.2 System Requirements

The Human Interactive Mission Manager's (HIMM) design is motivated and guided by a set of high level goals and requirements. The baseline goal is to implement a mission manager that allows for asynchronous human interaction in a structured way, inclusive of the three types of human interaction management methods. The ADEPT-based activity object in the HIMM system handles human interaction through the registering of parameters for interaction as a form of external coordination. These parameters must be wrapped in metadata which limits if and when human interaction is enabled and specifies numerical constraints of parameter changes. Registration is supported as a baseline service of the HIMM's kernel framework. Previous implementations did not have interaction mechanisms during mission execution as a fundamental requirement of their baseline execution. The addition of metadata wrapped registered parameters for interaction as a baseline service must not interfere with the services previously available in ADEPT implementations.

The kernel must still adhere to the ADEPT architectural structure which breaks down a mission into a hierarchy of activity objects with four boxes and include all the functionality expected from an ADEPT-based system. These ADEPT functionalities require that the kernel of the system will provide a mechanism for an activity object to plan its execution, execute its plan, monitor its execution and diagnose issues that arise during execution of a plan. Supporting on-the-fly mission replanning when requested (replanning occurring during mission execution) by an activity object is still the responsibility of the kernel as well. The ability to replan an activity object due to outside input, lack of progress toward mission goals or arrival of new information creates the capability to adapt to evolving mission objectives and sequences. A mechanism must exist to facilitate communication between activity objects. These are the fundamental

functionalities provided by previous ADEPT implementations to allow a mission to control a vehicle and adapt to unknown factors in its environment. Building the HIMM off of previous implementations of the ADEPT technology allowed the focus of the research to be the interaction problem.

In order to facilitate interaction, a user interface (UI) must be able to easily access the information to be adjusted and the constraints associated with that data. The decision of what data is available and how that data can be changed by the operator/UI must be decided by human factors engineers, with the help of mission and algorithm designers. The stability of the system is more important than giving the operator unrestricted access to data. Mission designers have an area of knowledge between mission and interaction design, which gives them the insight to distinguish valuable interaction from system destabilizing access. Interaction must be constrained so that changes are achieved within safe pre-set boundaries. Constraints to changes must be available to the UI so that the operator can be given all the information needed to efficiently provide input to the system.

In a real-time application of an ADEPT implementation, the kernel of the system typically actuates each of the four boxes of an ADEPT activity object at a frequency of one hertz. This frequency depends on the domain of execution but will be used as a baseline for this project. The extra processing for the interaction mechanisms of the HIMM system must still ensure that the kernel can actuate the activity hierarchy with limited overhead. The HIMM system must perform a kernel Actuation loop in less than one second and should leave at least 99 percent of that time period for the activity objects to perform their monitoring, executing and diagnosing. The goal of the HIMM system is to provide a layer on top of an ADEPT

implementation, but this goal cannot interfere with the timely functioning of the system toward accomplishing its mission. Table 1 summarizes the major requirements of the HIMM system.

Table 1 HIMM Requirements Summary.

Type	Requirement
System	Interaction Mechanisms must be a baseline service of the HIMM.
Interaction	Mechanisms must exist to constrain interaction to allow a mission designer to ensure safety. This includes when interaction can occur and what the limits are on parameter changes.
Kernel	The kernel must adhere to ADEPT architectural structure (activity hierarchy of four box activity objects)
Kernel	The kernel must provide mechanisms for activity objects to plan, execute, monitor and diagnose.
Kernel	The kernel must support on-the-fly mission replanning.
Kernel	The kernel must facilitate communication between activity objects.
Interface	The HIMM must support direct access by a UI to facilitate access to registered parameters data, constraints and interaction specification.
Resources	The HIMM system must perform a kernel actuation loop in less than a second.
Resources	Kernel interaction mechanism processing must not exceed 1 percent of a one second kernel actuation loop, leaving 99 percent of actuation loop time for activity object processing.

3.3 Evaluation of Previous ADEPT Implementations

The Human Interactive Mission Manager (HIMM) builds off of the ideas from previous implementations of the ADEPT technology at Draper Laboratory. The Maritime Reconnaissance Demonstration (MRD) implementation provided the basis for the Maritime Open Autonomy Architecture (MOAA) implementation, currently under development. The Framework for Autonomy is another ADEPT implementation explored as a possible basis for the HIMM. The

MOAA program focused on creating a modular, extensible and usable implementation of ADEPT for the mission programmer based on the lessons learned from MRD and the Framework for Autonomy. While neither of these systems provides a flexible environment for a mission programmer to incorporate the high-bandwidth asynchronous human interaction, they each proved to have particular design features that could serve as a baseline to the HIMM.

3.3.1 MRD Implementation

MRD is an implementation of the ADEPT technology developed to perform mission planning control of an unmanned underwater vehicle. It includes all the functionality expected from an ADPET implementation. Design work was focused on algorithms specific to the domain of underwater vehicles. Figure 6 displays how the MRD Autonomous Controller system is designed. The Mission Manager is called the Mission Planner (MP) in this system. The Vehicle Subsystem Controller (VSSC) is responsible for all communication with the vehicle and its sensors by receiving data and sending commands. Shared memory is used to communicate data between the MP and VSSC. Situational Awareness maintains representations of the vehicle's surroundings for use by the activity objects.

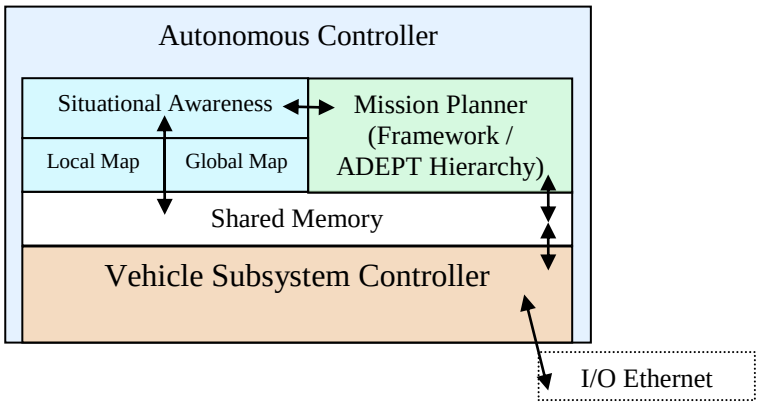

Figure 6 MRD Autonomous Controller System

The MRD system was designed specifically to control an underwater vehicle. In such an environment, communication outside the vehicle is limited to occasions when the vehicle is on the water's surface. Therefore, the system enabled outside interaction with the system to occur at predetermined times in the mission. Interaction in this system was synchronous with the plan (scripted), and the types of interaction were extremely limited. Human interaction was not viewed as a source of constrained input but rather as special case input to be built on top of the system. The types of input were limited to a small number of commands and each command had a different implementation at the system level, rather than having an interaction design as part of a baseline service provided by the MRD framework. MRD was not required to support direct human manipulation of algorithm parameters; the HIMM design is focused on supporting such manipulation.

The architecture of the MRD system is focused on providing activity objects as much flexibility and access to the framework as possible. A modular implementation was created using the 'C' programming language. This language does not have the feature set which facilitates object oriented implementations, and therefore generally makes any implementation of a system more difficult to extend. The MRD framework that manages the ADEPT hierarchy is tied to its operational environment since the backbone of the system (kernel) is itself made up of activity objects. The backbone provides services to manage the hierarchy to actuate execution and monitoring of each activity object as well as support diagnosis and replanning when needed. Tight integration of framework and mission/vehicle code in the system makes the existing implementation not an ideal baseline to extend toward new functional capabilities to the framework design in a structured manner [10]. However, a higher level view of the MRD design

does incorporate the functionality necessary for the HIMM, and, therefore, MRD was used as a basis for the ADEPT implementation used as a baseline for the HIMM system.

3.3.2 Framework for Autonomy

The "Framework for Autonomy" is a Draper Laboratory project with an academic focus to explore the possibilities of creating a general ADEPT infrastructure with which an autonomous system could be built. The focus was not on human interaction; rather it was a different perspective on the ADEPT technology. The framework was implemented in both Smalltalk and in the C programming language. There was an emphasis on an object oriented design, but neither of these implementations was easily extensible to the goals of the HIMM research. This translation is difficult because some of the services leveraged by the system are unique to the Smalltalk runtime environment. Both the Smalltalk and the C versions of the Framework for Autonomy were evaluated as possible baselines for the HIMM. The design of the ADEPT hierarchy and activity objects contained some interesting departures from the MRD-style ADEPT implementation.

The activity object concept exists in the framework, but as a three box activity object: planner, monitor and diagnoser. Additionally, activity objects in this system were not a packaged entity with a fixed monitor, planning and diagnosis algorithm for each instance of a particular activity object. During planning, a version of a planner, monitor and diagnoser are chosen by the activity object, and only then they are associated with that activity object. This freedom in algorithm selection could be useful, for example, to allow an activity object to choose a less accurate but faster path planning algorithm if the system needs to avoid an obstacle. There are many other interesting scenarios where this ability could be leveraged. Commands in the

system take the form of the leaves of the activity hierarchy and are executed by the backbone framework. Therefore, the hierarchy's leaf activity objects generate the plan in the form of commands that the framework then executes. There is no execution box in an activity object, since the plan consists of the command leaves hanging at the bottom of the activity hierarchy. This is a large departure from the MRD design because the activity hierarchy contains non-activity objects, and the activity object is missing one of the baseline functions (execute).

Coordination in this framework has a much broader meaning. The backbone of the system is expected to facilitate communication between the boxes (planner, monitor, and diagnose) of the activity object and between activity objects. Coordination also includes input/output functionalities. The Framework for Autonomy is a prototype that was never applied to a deployed system, and therefore the design and implementations are less mature. Although many of ideas in this design are interesting, the focus of the HIMM research is not to further explore new ADEPT designs but to add interaction as a baseline service. Since this framework was experimental and time was limited, there is very little documentation of the lower level implementation which made the system difficult to understand at the code level but the ideas in the Framework for Autonomy were considered when the HIMM was designed [11].

3.3.3 MOAA Implementation

The MOAA system was designed using the MRD system as a basis, but focused on creating a general, modular design while leaving open the ability to build new functionality into the baseline mission manager. Coded in the C++ programming language, MOAA utilizes a fully object oriented design. The framework of the system is domain independent with vehicle and mission code separated from the kernel of the system. All the functionality expected in an

ADEPT implementation are present: a planning and replanning capability, 4 box activity objects, and activity objects that are stored in a hierarchical temporal tree. It is an attractive option for the HIMM system because of its well-documented, modular and easy to augment design.

The kernel classes are separate from code specific to its operational environment. The kernel class operates on the tree, but is not itself an extension of the Activity base class or any part of the tree itself. Figure 7 shows the modules of the MOAA system. Code relating to the mission is separate from the services specific to the vehicle, those specific to the domain, and those which are provided by the MOAA kernel. The kernel provides the base class for all activity objects created as part of the mission. This ensures that each activity object in the mission has the same services available from the kernel. The ADEPT hierarchy of mission activity objects is controlled by the kernel. The modularity of the system makes it easy to see what the underlying kernel provides to the system and how to add a new baseline service.

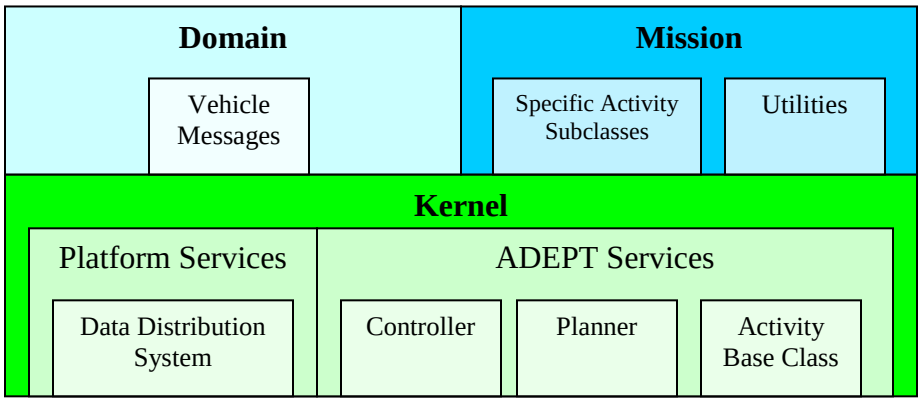

Figure 7 MOAA System. There are three major modules to the MOAA mission manager which are separate. The Kernel and Domain classes provide services used by activity object implementations in the Mission Module. The kernel owns and controls the activity tree.

3.3.4 Implementation Selection

The HIMM is built on the MOAA technology because of its modular, easy to follow structure and its programming language choice. The C++ language is object oriented yet still

highly applicable to deployable systems. The MOAA design is easily extendible for the new services necessary to enable human interaction into a mission manager. The MRD structure was too integrated to use as the basis for an extensible mission manager, and the Framework for Autonomy's design is so distributed and open, it becomes difficult to comprehend and control. MOAA's design borrows from the positive aspects of the MRD design ideas and added modularity making a good baseline system on which to build the HIMM. The HIMM system was built on a scaled down version of the MOAA design programmed in the Java programming language. Although Java is not considered a real time language due to its uncontrollable background processes and virtual machine, the language was chosen due to superior tools, easy prototyping tools and similarity to C++ syntax. Future implementations of the HIMM design could then be translated into a more accepted real-time language.

3.4 Human Interactive Mission Manager Design

3.4.1 High Level Architectural Design

The HIMM system is divided into five major modules: Mission Manager (MM), External Coordinator (EC), Common Data Repository (CDR), User Interface (UI) and the vehicle itself. Each module provides a service to the other modules it communicates with. This thesis section will describe how each module's service supports the functionality of the larger system. An overview of the HIMM's system architecture is illustrated in Figure 8.

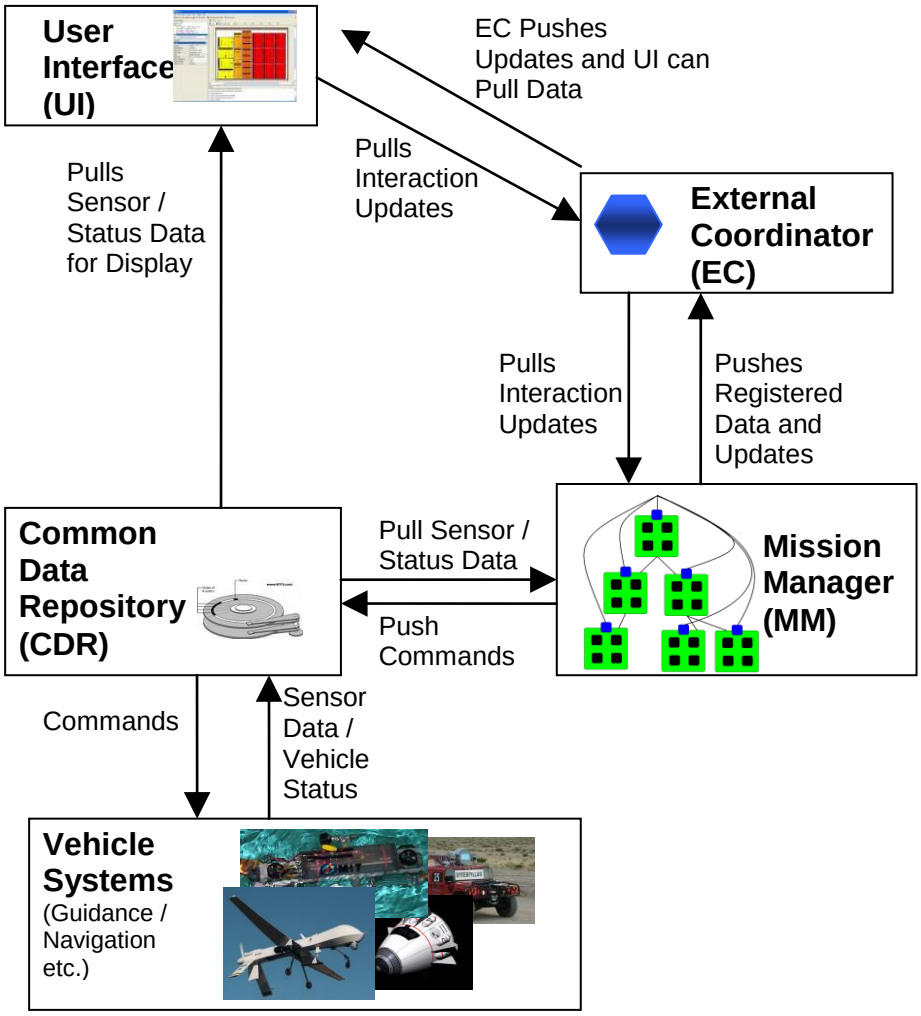

Figure 8 HIMM Architectural Overview

For the operator, the window into the system is the UI. The UI has access to the sensor data and basic status information coming from the vehicle via the CDR. Data from the CDR is read by the UI and cannot be modified or written. The CDR is the repository for data coming from the vehicle's sensors and also is where other vehicle systems (Guidance, Navigation, etc.) retrieve data for its systems. Data associated with the plan resides in the MM; however, the UI accesses data through the EC which acts as a buffer between the UI and the MM. Any data that the EC has access to must first be explicitly registered by the MM. Registered Data Objects store which activity object it pertains to, constraints on any changes to the data and whether interaction with the data is available. If interaction is enabled, the operator can use the UI to send requests for changes. A response from the MM comes from the EC to indicate whether the change was applied. The EC is tightly coupled with the UI, and the EC acts as an interface to and from the MM for the UI.

Registered Data Objects are created and managed by the MM, of which the EC has a copy and provides to the UI. Manipulation of that data is specified and enforced in the MM. Legitimate changes to data by the operator using the UI are sent to the EC and then to the MM to be applied. Changes to the parameter can only be applied if the Data Object is registered as interactive and if the change passes the constraints associated with the parameter. This parameter registration service supports human interaction and the ability to bring out the internals of the plan to the operator. The Data class is described in detail in 3.4.4. The MM provides all of the essential ADEPT mission manager functionality. The MM communicates with the other parts of the system, such as guidance and navigation software, by sending commands through the CDR.

The HIMM still provides the functionality of the ADEPT technology, but with the additional capability of parameter registration with constraint and interaction specification through the EC as mediator. The CDR allows the HIMM system to receive sensor data and communicate with other software systems. The EC allows the UI to communicate with the MM through the EC mediator to ensure that the UI never has direct access to the data used in the MM. This decoupling allows the MM to run as it normally would, with periodically applied updates, provided the updates meet their data constraints. Each module helps support the goals of the system to provide ADEPT functionality with interaction without sacrificing system stability.

3.4.2 Parameter Registration/Data Flow

Data parameters and updates to that data drive the information flow in the HIMM system. There are three types of data that are accessed by the UI and the MM: Local, Activity-wide and External. External data is associated with the CDR and is accessed by the MM and UI through the CDR interfaces. This section will focus on the two remaining types of data, both of which are associated MM interaction.

Parameters are classified as Local or Activity-wide. Local data is associated with a particular activity object instance and is deleted when that activity object completes or is replaced during a replan. Activity-wide data is associated with a particular activity object type, but persists even when the activity object is removed from the operating tree. Once a Data Object has been registered with the system, the EC must be notified of the registration and updates to that data object are communicated between the MM and UI.

Data created by the MM and registered to be available in the EC are wrapped in metadata that specify constraints on that Data, and whether or not the Data is interactive. An example of a

constraint on a numerical value is a range constraint that limits changes to a parameter to a specified range of values. The values specifying the range could be another registered parameter or a static number. Changes to a value will only be confirmed by the MM if the change request passes all constraints and if the parameter is registered as interactive. Since the EC keeps a separate library of registered parameters from the MM, information about new parameters and updates to those parameters must be passed between the MM and EC.

Information exchange between the MM, EC and UI are accomplished through the use of message passing. Data flows toward the UI in the form of MMUpdate Objects, which contain parameters that have been registered or changes to the data made by the Mission Manager. Interaction change requests initiated by the operator come toward the MM in the form of UIUpdate Objects. If the parameter associated with the UIUpdate is specified as non-interactive, the EC will not provide the UIUpdate to the MM. The flow of data is summarized in Figure 9.

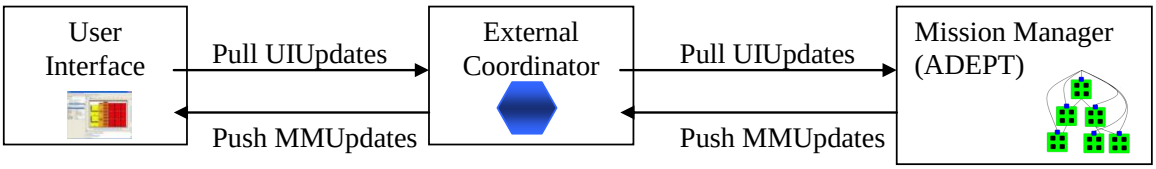

Figure 9 Data Flow with Messages

The update architecture allows the UI to request data from the EC as necessary without interrupting the execution of the MM. Additionally, maintaining a separate EC structure allows the system to be distributed with the UI/EC running on a separate machine from the MM as long as a protocol for message transfer can be established (e.g. Ethernet). The metadata wrapping enables the enforcement of constraints and limits changes to registered parameters. These characteristics of the data system make safe interaction possible by constraining changes and separating the MM's data from the data used by the UI.

3.4.3 External/Internal Coordinator Design

Coordination is necessary to keep data in the HIMM consistent between the MM's data and the data stored in External Coordinator (EC), which serves as a database for the UI. The EC lies between the MM and the UI and allows the UI to have access to copies of the parameters registered by the MM. Internal Coordination refers to the tasks that the MM must undertake to deal with updates coming from the UI, to send out updates generated by itself and to enable communication between activity objects. Coordinators generate and apply updates, keeping the stored data available and accurate for the MM and UI.

Inside the MM, coordination is built into the kernel and the individual activity objects which together make up internal coordination. Figure 10 illustrates the activity hierarchy with internal coordination in each activity object reaching toward the MM kernel backbone. The kernel contains a coordination loop which sends a buffered list of the MMUpdate Objects from all the activity objects in the MM to the EC and also pulls UIUpdate Objects from the EC and delivers them to the individual activity objects to handle. The kernel is also responsible for informing the EC of activity objects that have completed or have been replaced in the activity hierarchy, and of all newly registered data. The individual activity object is responsible for registering its data with the kernel and then handling incoming UIUpdate Objects for the parameters it has registered. When a change is made to a parameter's value, an object is generated to inform the UI of this change, but other activity objects in the active branch might also be interested in changes to this particular parameter. The kernel is responsible for forwarding Activity-wide parameter updates to activity objects that registered as listeners to changes for that parameter. The EC must perform similar tasks to keep its data updated for UI

use.

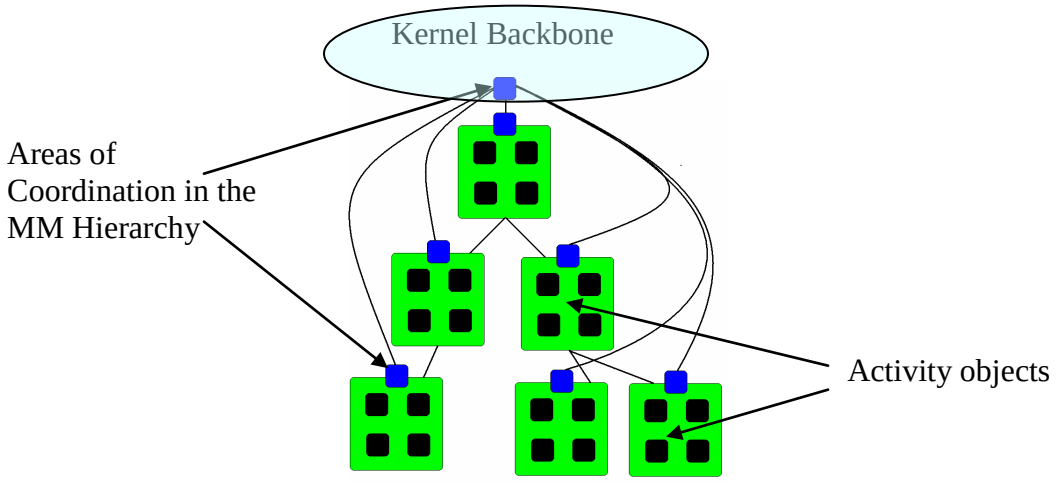

Figure 10 Areas of Coordination in the MM. Activity objects are organized into a hierarchy and are actuated by the kernel. The kernel provides services to the individual activity objects, and is also responsible for managing internal coordination for the MM. Blue areas represent coordination.

The External Coordinator (EC) is responsible for keeping a local copy of the data registered by the MM, and handling the updates it receives from the MM. The only time the EC changes the value of a parameter it stores is when it receives an update from the MM alerting the EC to do so. In this way, the EC simply stores a reflection of the data contained in the MM. A change requested by the UI is just forwarded to the MM, but the EC does not apply the change until the EC receives a change update from the MM. The EC also handles updates from the MM indicating activity object removal, which signals the EC to remove the activity object's registered local parameters. Locally storing the data from the MM in the EC gives the UI standardized access to the value and its metadata (constraints and interaction information).

The HIMM design of coordination creates a delay between changes applied in the MM and the reflection of the change in the EC and UI. This delay is acceptable because the human operator takes a comparatively long time period to process information while the MM is

constantly using the data at a higher rate to ensure the system is progressing toward completing its mission. If the opposite approach were used, and the MM received delayed data instead, it might delay the MM in recognizing a need to replan, thereby rendering any input the operator made to the old plan irrelevant. This design for coordination in the HIMM gives the UI access to the data needed without interrupting the MM directly. Coordination thereby supports human interaction by providing MM coordination procedures (message passing) as a kernel service. The external data storage in the EC is necessary to avoid exposing the MM to asynchronous data requests and adjustments from the UI that could destabilize the MM.

3.4.4 Data Class Design

The encapsulation of parameter values beneath a wrapper class of metadata that specifies how and if the data can change, is a fundamental addition to previous ADEPT implementations because it allows the HIMM system to support interaction. This design also acts as a standard mechanism for the UI to access copies of the underlying MM data. Therefore, the design of the Data class is fundamental to the success of the HIMM project as a step forward for ADEPT implementations.

At the MM level, the wrapper class adds a layer of indirection in order to access and change the underlying value. This indirection is necessary since a change to data must meet constraints contained within the Data class and must also generate an update object for the EC if a change is made. The algorithm and mission designers do not need to worry about the extra steps needed to keep the EC updated, as long they use the methods provided for changing the value of registered parameters. The Data class and the kernel of the MM are responsible for the updating mechanisms. When a parameter change is initiated by the MM, the change must still

adhere to the constraints that were attached to the parameter, or the change will be rejected. This view is slightly different than when the change is initiated by the operator.

An operator initiated change to a registered parameter reaches the MM in the form of a UIUpdate Object received from the EC. There are two steps that the Data Class will apply before making the change: applying the interaction specification and applying the constraints. The first step is to verify that the parameter passes the specifications associated with the interaction. Examples of interaction specifications are: no interaction allowed, management by timeout, management by consent or management by exception. A parameter with interaction enabled at the level of management by timeout would include the length of time the MM will wait before moving forward. In this case, the request from the UI for a change would have to be made within the time period specified for the change to be applied. Management by consent would include an indicator as to whether the operator had given consent to proceed. Management by exception would include an indicator of whether the MM is currently requesting input from the operator. The next step would apply the constraints associated with the data. If the requested change passes the validation checks, the change is applied to the MM's version of the data. The Data class then generates an Update Object to alert the EC and thereby alert the UI of the change.

The design of the Data class allows changes to be made by the MM and the operator without the mission/algorithm designers being responsible for specifically generating updates for each change or locally keeping track of whether or not interaction is enabled. Mission and algorithm designers can utilize outside interaction with minimal added programming overhead since the Data class is a resource with embedded metadata to automate updating. The Data class

wraps the value of the parameter with knowledge about when changes can occur and what the limits are for value changes.

3.4.5 Interaction Specification

The interaction specification encapsulates information about whether or not human input on registered data is available. This specification acts like an added constraint on parameter changes originating from the operator. There are several different kinds of interaction: no interaction, open to interaction, management by exception, management by timeout, and management by consent. Each specification for interaction puts a different limit on interaction with a parameter.

The HIMM supports a human supervisory level of control. Management by timeout specification gives the operator a window during which interaction with the parameter is available. The amount of time is predetermined by the mission designer and begins when the activity objects associated with the parameter becomes active. An activity object is active if it is in the left-most branch of the activity hierarchy because those are the activity objects being executed in the plan. The Interaction Specification, management by consent, can be applied to a parameter that acts as a signal to the system to proceed. The system should not proceed until the operator provides the signal. Management by exception in this system means that the parameter cannot be changed unless the system decides it wants such input. These three types of interaction provide different levels of cooperation between human and computer at the supervisory level of control, but occasions arise when more exclusive control is needed by the MM or operator.

The two remaining interaction specifications provide the more extreme versions of human access options: none and complete control. The no interaction specification should be applied to parameters that are specific to the MM, and should not be available for human input but are important for the operator to monitor for situation awareness (what the system is doing). Open to interaction specification is the last form of interaction supported, and means that the parameter is always open to changes. This specification is useful for parameters that are Activity-wide (shared by activity objects of the same type) and used in an activity object as an algorithm constant.

These specifications allow the operator to apply temporal and decision based boundaries for interaction while constraints provide numerical boundaries to data manipulation. The HIMM system gives the mission designer the tools necessary to incorporate interaction in their design and constrain it appropriately to ensure mission objectives are maintained as well as preserving the integrity of the algorithms which use the parameters. The Interaction Specification enables human interaction with the mission in a managed and pre-determined manner by the designers of the software.

3.5 HIMM Implementation Design Description

The Human Interactive Mission Manager (HIMM), as described in the architectural overview, consists of four major software modules: the User Interface (UI), Common Data Repository (CDR), the Mission Manager (MM) and External Coordinator (EC). The CDR is the database and message center between the MM and the vehicle but that module is not at the center of this research project. This section will go into detail about the class structure of the MM and EC because these modules support the HIMM project's goal of human interaction.

The MM is concentrated on the structure, actuation and maintenance of the activity hierarchy; the External Coordinator is concentrated on keeping a copy of the parameters and their metadata, which have been registered by the activity hierarchy, updated and available for the UI. Figure 11 shows the central classes of the MM and EC, and the interfaces that connect them to each other and to the UI. The interfaces between modules allow parameter updates to be forwarded around the system. The EC is centered on a single class that processes updates from the MM and is responsible for maintaining a copy of registered parameters for UI access. The External_Coordinator class signals the UI which parameters have been added, updated or deleted. The External_Coordinator receives updates because it implements the MM_Listener interface which facilitates message passing between the MM and UI.

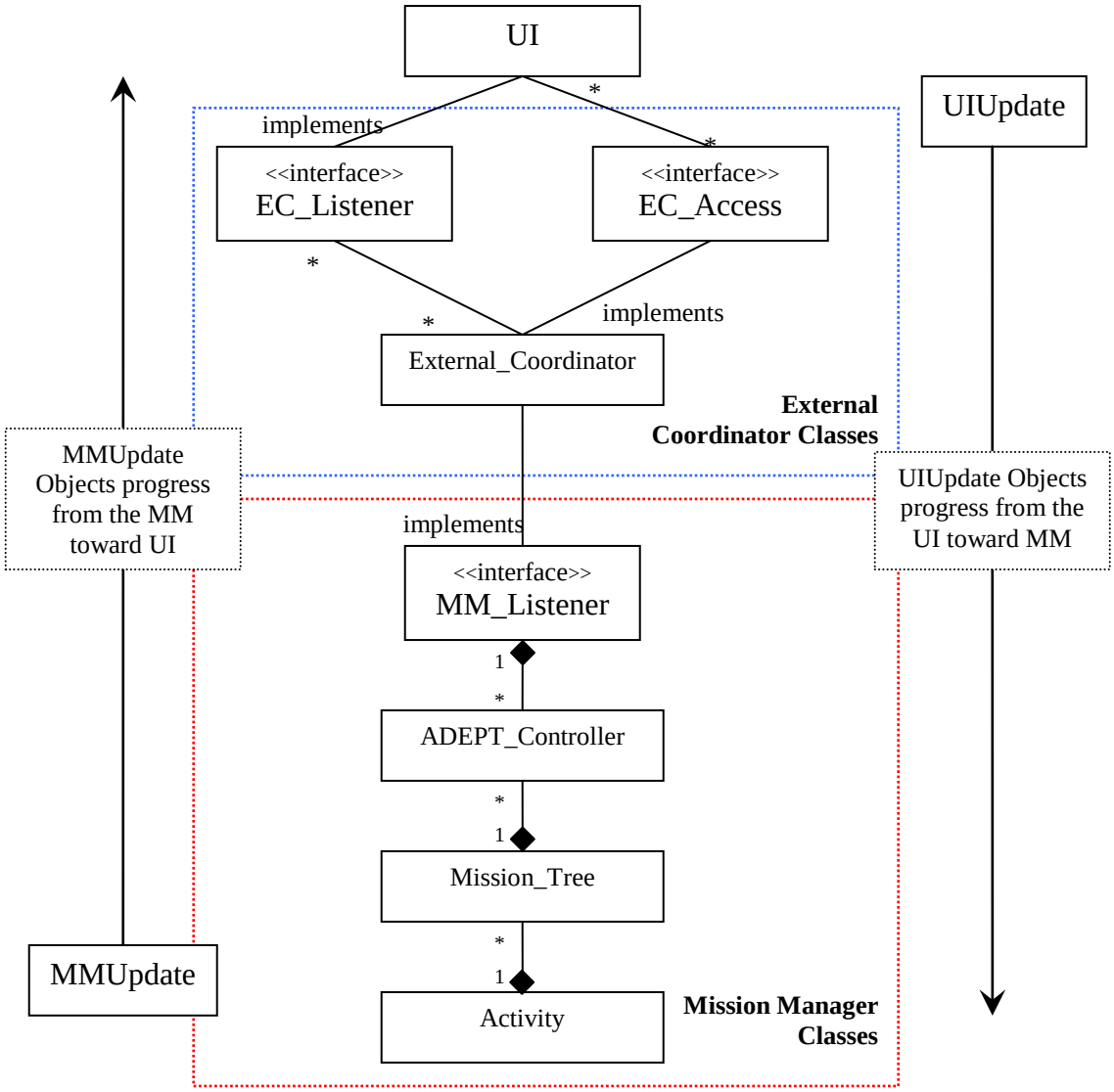

Figure 11 High Level Class Structure of the MM and EC. Interfaces are used to send MMUpdate Objects from the MM toward the UI and UIUpdate Objects from the UI toward the MM.

At a high level, the MM is composed of three classes that contain the major functionality of the ADEPT technology, as shown in Figure 11. The ADEPT_Controller class is at the heart of the kernel of the MM and actuates the four box activity objects in its hierarchy. It is also responsible for receiving and forwarding UIUpdate Objects from the UI and sending MMUpdate Objects to the EC. The Mission_Tree class maintains a hierarchy of activity objects and contains methods to manipulate the tree (graft a new branch to the tree, prune a branch of the tree, etc). The ADEPT_Controller contains the activity objects in the hierarchy in the form of a

Mission_Tree Object. The kernel's Activity class is a base class that is extended by each Activity subclass in the mission. It contains the expected scaffolding so that the kernel can actuate and manipulate the activity object. These three classes encapsulate the ADEPT functionality, but the next level of MM detail shows the additions to the baseline MM of the HIMM to support human interaction.

3.5.1 Mission Manager Kernel and Mission

The MM, at the next level of detail, is broken into two major packages: the kernel and the mission. The kernel is the core of the MM which remains unchanged from application to application and provides services to the mission. The mission consists of activity objects that have extended the Activity base class of the kernel. Figure 12 shows this breakdown in the MM as well as the major classes in each package. By extending the Activity base class, each activity object implements the scaffolding of methods expected by the kernel, which include: specific planning, execution, diagnosing and monitoring functionality for the activity object; the registration of its local and Activity-wide parameters; and its handling of UIUpdate Objects. The kernel can then call these methods to actuate the activity object when appropriate.

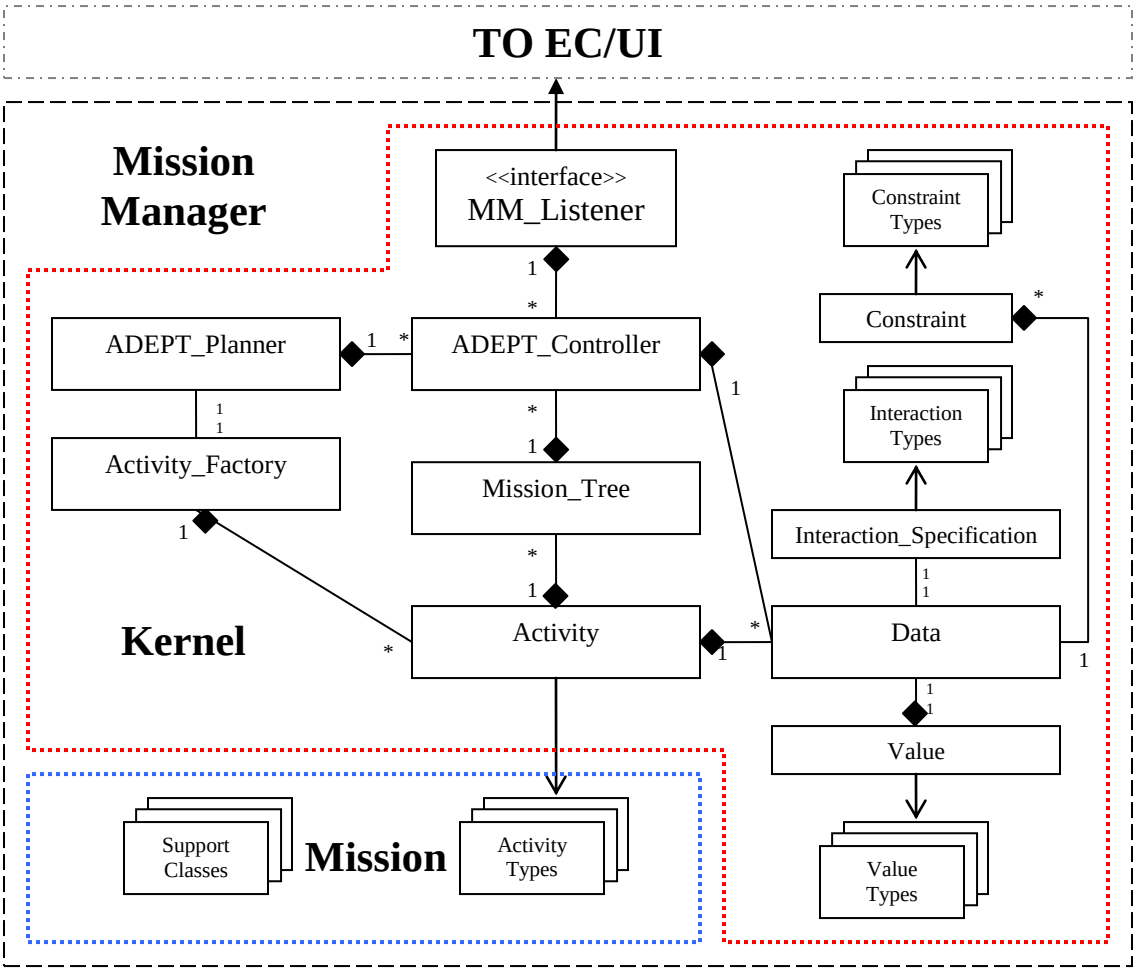

Figure 12 Low Level view of Mission Manger Kernel and Mission Packages.

The kernel package breakdown in Figure 12 includes the classes that control the activity hierarchy (ADEPT_Controller and MissionTree) as well as the support classes that provide the functionality of the kernel. When planning or replanning is necessary, the ADEPT_Controller creates an ADEPT_Planner object that is responsible for planning an activity object in a separate thread of execution. The result of planning is a new branch of the tree that replaces the unplanned activity object or the activity object that requested a replan. The information necessary to plan an activity object is passed into the ADEPT_Planner, which then uses the Activity_Factory to retrieve an instance of the activity object to be planned, and uses the

initialization information to initiate planning. The kernel also provides the services necessary for parameter registration and maintenance, message handling, and internal coordination.

The additional kernel services to support human interaction include parameter registration, sending MMUpdate Objects and handling UIUpdate Objects. Parameters are registered through methods in the ADEPT_Controller which create MMUpdate Objects for the new parameters and place them in storage. Local parameters are stored in a Map data structure in each activity object instance, while Activity-wide parameters are stored in a Map data structure in the ADEPT_Controller. Access to change or read a registered parameter is accomplished through accessor functions in the ADEPT_Controller. UIUpdate Objects pulled through the MM_Listener interface are sorted and sent to the individual activity objects. After the activity object reacts to the updates, the UIUpdate Object is given to the specified parameter for application. The Data class contains the code to apply an update or reject it if it does not meet its constraints. It can then create the necessary MMUpdate Objects to alert the MM_Listener of any changes. All MMUpdate Objects are batched together in the ADEPT_Controller and are sent to the MM_Listener after all the UIUpdate Objects have been applied and internal coordination of updates is complete. The MM kernel is responsible for packaging and forwarding MMUpdate Objects, pulling and applying UIUpdate Objects and supporting parameter registration and maintenance. Maintenance consists of the creation of MMUpdate Objects for changes to parameters to make sure the EC has accurate copies of registered parameters. These kernel services are necessary to ensure the parameters the operator is using are as accurate and up-to-date as possible.

3.5.2 External Coordinator

A deeper look at the EC highlights its focus on message passing and parameter maintenance. Figure 13 shows the lower level representation of the class structure which highlights message passing.

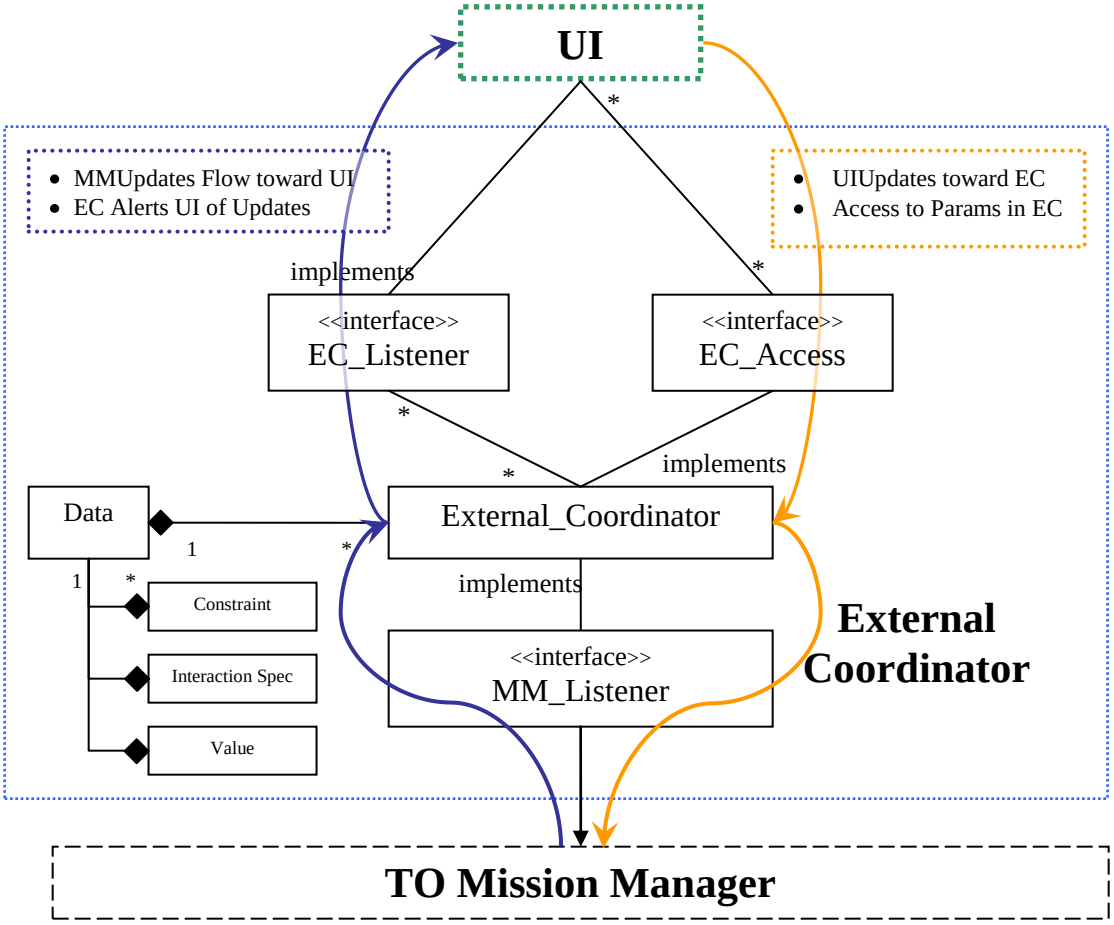

Figure 13 Low Level View of External Coordinator Structure. Shows how update messages pass through the EC.

The EC implements the MM_Listener interface which allows it to receive MMUpdate Objects from the MM and forward UIUpdate Objects from the UI to the MM. The UI implements the EC_Listener interface so that it can be alerted to parameters that have been registered, deleted or updated, and can then access those parameters from the EC through the EC_Access interface. The Data Objects themselves are structured exactly as in the MM, but are

stored slightly differently. The Local parameters are stored in a single Map data structure and are added and removed when the EC is signaled of a Local parameter registration or of the deletion of the activity object that the parameter is associated with. All of the code to parse messages from the EC and UI and forward them appropriately is located in the External_Coordinator class.

3.6 Effect of HIMM on User Interface Design

The User Interface for any mission using the HIMM system can use the same backbone code to access and process parameters since it will be using the same system interfaces. All the possible interaction with the HIMM takes place through manipulation of parameters. The Data Class encapsulates all the metadata needed to display the information. This metadata will guide the designer to illustrate the parameter so that the operator understands what the parameters represents, and what kinds of changes can be made. By making the limits of interaction explicit within the HIMM software, the responsibility for designing and limiting how much an algorithm can be adjusted lies with the algorithm and mission designer. This adjustment is advantageous since the algorithm and mission designers have the best understanding of what can be adjusted and the effects of such changes. The UI designer can now focus on the best means of bringing the information and interaction out to the user rather than figuring out which and how data in the system should be adjusted.

3.7 Effect of HIMM on Algorithm/Mission Design

Algorithm and Mission designers for the HIMM system can now add human interaction as a component of their design. Such a design requires a deep understanding of the inner workings of algorithms and how changes to parameters of an algorithm affect the output of the system. If operators are to act as cooperative agents to the system, they must have context of how their changes will affect the output. Consequently, simply designing an algorithm which computes an output using certain inputs is not sufficient without identifying what those inputs mean in the mission in human terms. Giving an operator access to change every parameter of the system is not useful either since it simply confuses the operator or makes the algorithm unwieldy. It will be a new specialty to understand how best to allow an operator to affect an algorithm or mission in order to provide the biggest gain in system effectiveness and insight into what the system is doing. To be successful in this area of design, an understanding of human factors or collaboration with a human factors engineer, as well as a technical understanding of algorithms and mission operations, will be necessary to recognize and take advantage of useful opportunities for human computer collaboration in the mission environment.

Chapter 4

4. Example Mission Application

4.1 Introduction

A Mission Manager (MM) framework that enables the design and implementation of mission logic and algorithms, which include human interaction, can be difficult to grasp in isolation. Description of the design and software implementation of such a system has an inherent lack of concreteness due to its abstract nature. This chapter uses a concrete application of the framework on an example mission and uses the example to highlight the overall structure and capabilities of the HIMM, including human interaction. Chapter 3 focused on the Human Interactive Mission Manager (HIMM) baseline system design including the interaction methods embedded in the kernel of the MM and External Coordination (EC). This chapter will demonstrate how these modules can be applied.

The example application of the HIMM system is in the Space domain controlling a Lunar Landing vehicle. Specifically, the focus of the mission for the purposes of demonstrating the HIMM architecture is in choosing a viable landing point for the landing vehicle, which is in the Landing Point Redesignation (LPR) stage of the lunar landing descent. This mission portion includes processing a variety of map information from a LIDAR (Light Detection and Ranging) sensor and finding a list of candidate safe aim points using the data. The criteria for comparing possible landing points used in the algorithm include tolerance constants as well as specification

of a target landing area. These parameters are perfect examples of data which a human should have constrained access.

This chapter will describe the Lunar Landing mission focusing on the Landing Point Redesignation (LPR) stage of the mission at a high level and then describe how the mission translates into a set of activity objects. There will be a discussion of each activity object's responsibilities and what services of the kernel the Activity subclasses utilize. Since registered parameters are central to human interaction in the HIMM system, the registered parameters of the activity objects in the hierarchy will be examined as an example of what kind of parameters are registered and for what reasons.

4.2 Lunar Landing Mission Description

NASA plans to have a manned mission to the Moon by 2020 as part of their Constellation Project. Draper Laboratory has been selected as one of the contractors to work as part of this multifaceted effort. One project associated with this effort is the Autonomous Landing and Hazard Avoidance Technology (ALHAT) project [12]. Both the Lunar Landing scenario and the LPR algorithms from this project serve as models for the HIMM demonstration. Both the scenario and the algorithms have been modified to suit the HIMM architecture demonstration. The HIMM project has taken the ALHAT LPR algorithm and adapted it to the HIMM system to take advantage of its human interaction mechanisms for the purposes of human input in the landing point choice.

The Lunar Lander is a manned vehicle with the goal of safely landing on the moon. During its descent toward the surface, there will be a period of time where an opportunity will exist to view the surface and adjust the location where the vehicle will land. This period of time

is during the Pitch Over mission stage which is when the vehicle positions itself for its final descent to the surface. Using surface elevation data received from the vehicle's sensors about, the human and computer work together to find a suitable landing aim point for the guidance software to target during the Terminal Descent phase. The operator is able to adjust a set of parameters to affect the algorithm's choice of the best set of candidate landing aim points. The operator (astronaut) can then choose a final aim point from the list of prioritized options or change the parameters to recalculate the list of candidate aim points. The vehicle will then adjust its descent to land in that selected location on the lunar surface. Figure 14 illustrates the Powered Descent phase of the mission during which the landing point decision is made.

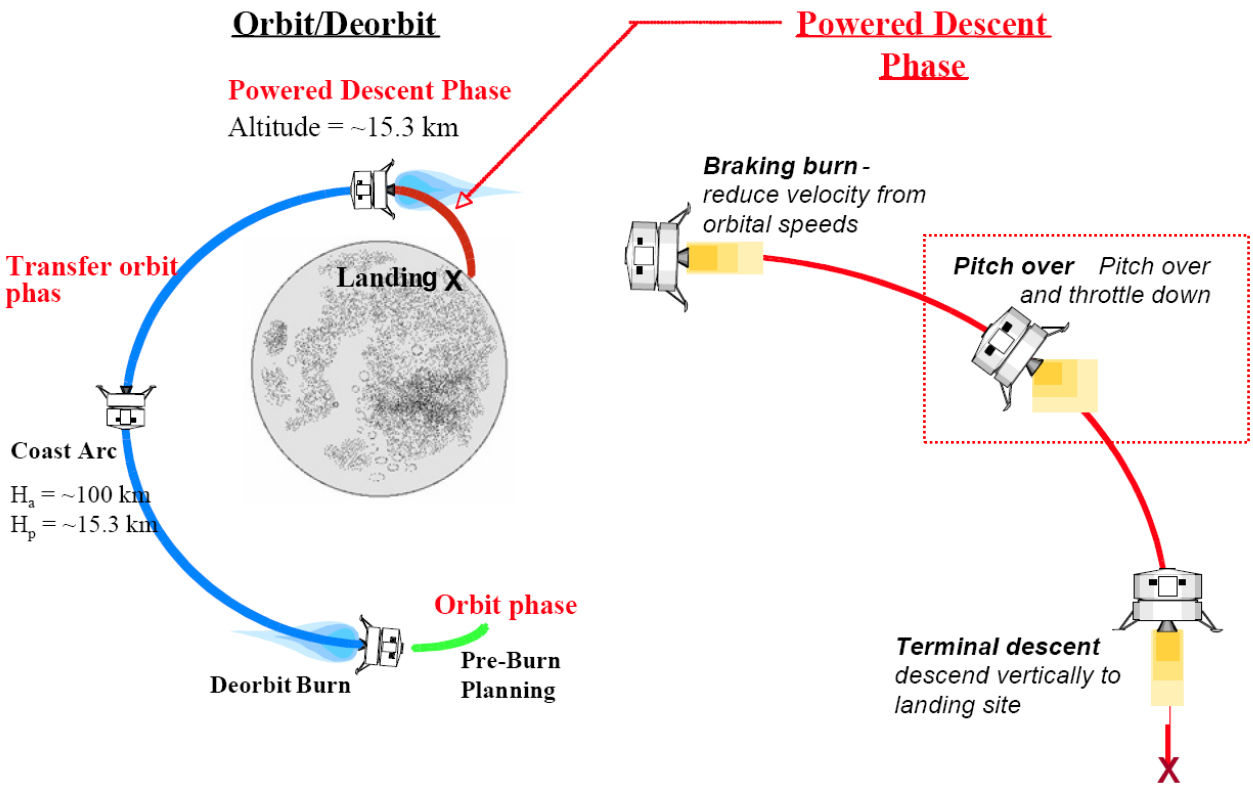

Figure 14 Lunar Landing mission phases and maneuvers [12]. Boxed portion is the Pitch Over mission phase. This is when the Landing Point Redesignation task is performed.

4.3 HIMM Implementation of Lunar Landing Mission

The HIMM system requires an extra level of engineering for mission and algorithm design before the activity object subclasses associated with the mission are finalized. A human-systems collaboration engineer must be involved in the development of the algorithms in the system and as a result, determine which parameters the operator should access. There must also be a discussion of the pros and cons of displaying certain parameters and what types of access should be available to the operator. The correct balance of information access will give the human understanding and comfort with the actions of the software without overwhelming the operator with too many decisions and data.

Implementation of the Lunar Landing Mission using the HIMM system was undertaken after one generation of the LPR algorithm had been designed and analyzed for human interaction. Each parameter that is registered with the HIMM has a specific purpose and has value in operation clarity or to give greater control to the operator to affect how the automation chooses the list of candidate aim points. The software must ensure that the landing point chosen is safe, but there is freedom within parameter settings to affect the final choice of landing aim point.

4.3.1 System View of the Task

The HIMM implementation of the Lunar Landing Mission decomposes the mission into sections by task to create its activity hierarchy. This scenario will focus on the Pitch Over Activity and its sub-activity objects: Cost Map Calculation and LPR. Figure 15 shows the entire Mission hierarchy and highlights the activity objects that will be the focus of our application example.

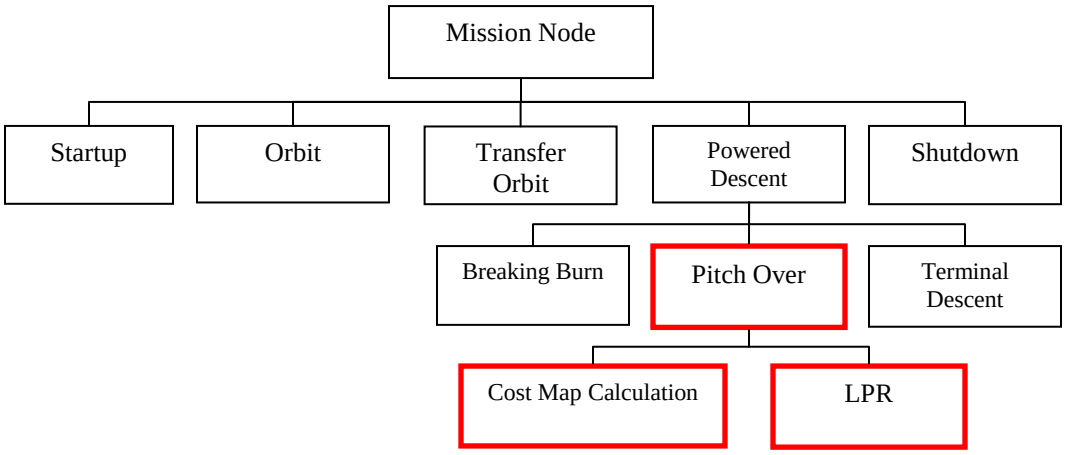

Figure 15 Activity Hierarchy for Lunar Landing Application. Red outlined activity objects are the focus of this application example.

In the context of the entire system, the activity objects themselves are only part of the solution to accomplishing the tasks they are charged with solving. Guidance software, sensor software, the CDR, as well as the human operator are all vital parts of the success of finding a suitable landing point for the vehicle. Figure 16 describes the relationship between the major actors in the Landing Point Redesignation scenario. The goal of the Pitch Over Activity is to choose an aim point on the lunar surface for landing that is both feasible for the Lander to reach given speed and fuel constraints as well as safe given the features of the surface in that area. When the decision is made, the chosen aim point will be sent to the CDR for Guidance Software use to adjust the vehicle's trajectory to land in the chosen area. In order to achieve its task, the Pitch Over Activity spawns the Cost Map Calculation and LPR Activity Objects. The Cost Map Calculation Activity is tasked with calculating cost maps by retrieving sensor data from the CDR that was set by software controlling the LIDAR sensor. The raw elevation data will be processed to create three cost maps: Distance to Nearest Hazard Cost Map, Distance to Target Cost Map and DV Cost Map. Section 4.3.2 will discuss how human interaction can affect the calculation of these cost maps as well as the final list of aim points. When the maps have been calculated, they are made available to the rest of the HIMM system by registering them as parameters.

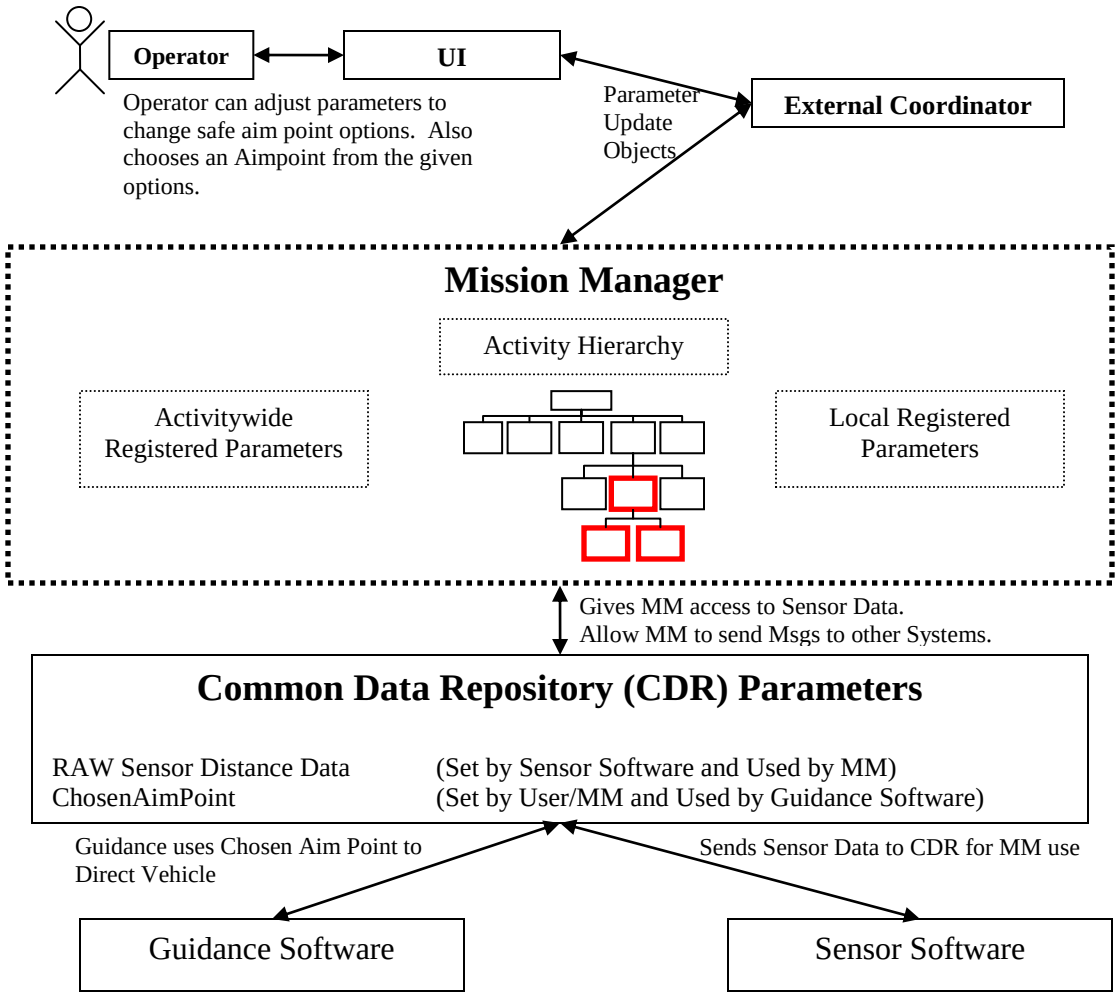

Figure 16 System View of Landing Point Redesignation Scenario.

The cost maps calculated by the Cost Map Calculation Activity are then used by the LPR Activity to achieve its task. The task of the LPR Activity is to use the cost maps to come up with a ranked list of the best locations on the surface of the moon to direct the vehicle to land. The LPR Activity uses the three cost maps and integrates them into a single cost map. This weighted cost map is then used to find a list of candidate aim points that are the least costly. The algorithm takes spacing of the aim point options and other issues into account when coming up with its list. There are also human interactive registered parameters that can affect the options in the final list of aim points.

The list of viable vehicle aim points is a registered parameter that the UI displays to the operator. The operator may choose one option from the list to become the aim point sent to the Guidance software. During the decision period, the operator may choose to change registered parameters which will force the system to recalculate the list of landing aim point options. When the calculation completes, the operator will have a new set of landing aim points to choose from. The amount of time to make a decision is not unlimited, and if the operator fails to choose an option, the Pitch Over Activity will choose the current best option.

4.3.2 HIMM Registered Parameters in LPR

Interactive registered parameters in the HIMM are important to the activity objects that are part of the Landing Point Redesignation scenario. Changes to these parameters will affect the list of candidate landing aim point options. Figure 17 shows the parameters registered by the activity objects and whether or not human interaction is enabled. During the mission, there is only a limited amount of time to make a decision, so all the registered parameters that allow for human interaction are of the type 'Management by Timeout'. The Pitch Over Activity contains an Activity-wide parameter specifying how much time remains in the interaction period. This section will describe the affects of changes to interaction enabled registered parameters in this mission.

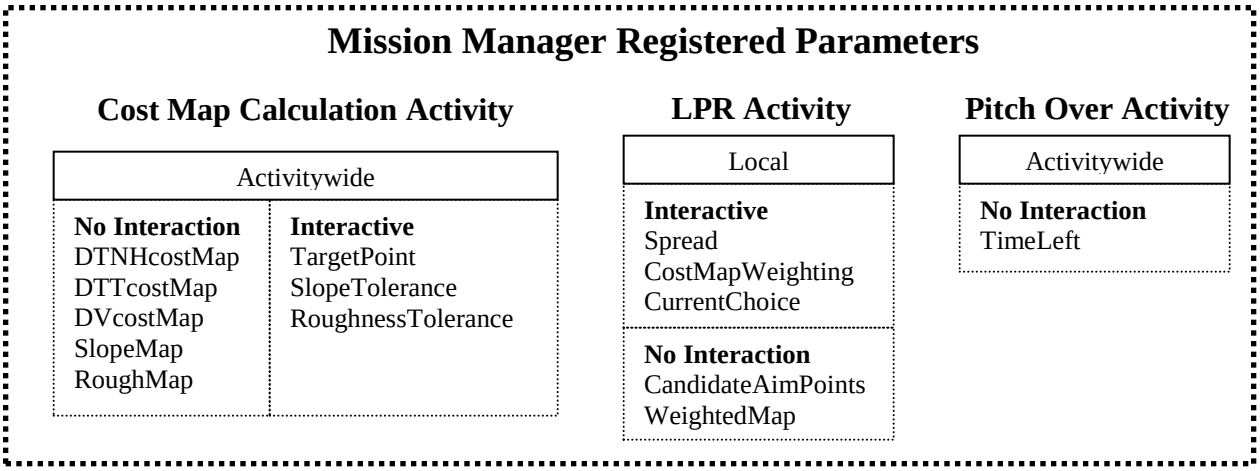

Figure 17 Registered Parameters in Landing Scenario

In the Cost Map Calculation Activity, the registered parameters are all Activity-wide so that any activity object can access the parameters and they persist even when an instance of the activity object completes. The non-interactive parameters are the cost maps that are calculated in the Cost Map Calculation Activity for use by the LPR Activity. The interactive parameters are used in this activity object to create these cost maps. Slope Tolerance and Roughness Tolerance are parameter thresholds that represent what is considered capability limitations of the landing vehicle. There is an added constraint on these tolerances to limit the range of changes to only safe options. The parameter Target Point, allows the operator to specify a location where the operator would like to land which is used to calculate the Distance to Target cost map. Changes to these parameters during the decision period will force a replan of this activity object and the LPR Activity. The LPR Activity is dependent on the cost maps created in the Cost Map Calculation Activity so both activity objects must be replanned. The Pitch Over Activity is active for the entire period that interaction is enabled for these parameters, so it is a parameter listener that monitors for changes by the user to these parameters and schedules a replan if sufficient time remains for extra calculation.

In the LPR Activity, all of the registered parameters are local in scope to the particular instance of the activity object. The non-interactive parameters are the combined single cost map and the list of aim point choices. The interactive parameters correspond to the weighting between the three cost maps used for integration, the current aim point choice, and the spread, which is how disparate the aim point options should be. A change to the spread parameter does not cause a replan, since the aim points associated with any value of spread are pre-calculated. Such a change to the spread will change the list of aim points, which in turn will change the current aim point to be the highest ranked aim point in the new list. A change to the current aim point must be in the list of current options and that value is stored until a final choice is made or time runs out. When the interaction period is nearing its end, the LPR Activity will send the current aim point value to the CDR to be accessed by the Guidance software. The weighting parameter adjusts how the cost maps are integrated together to a single cost map. A change to the weighting parameter will force the LPR Activity to replan in order to recalculate the aim point options, but the Cost Map Calculation Activity will not have to be replanned based on this change.

The design of the algorithms in activity objects affects if, and what, changes to parameters will cause activity objects to replan. In the activity objects associated with the Landing Point Redesignation scenario, there are three levels of replans that can be triggered by parameter changes. If a replan is triggered in the Cost Map Calculation Activity, the LPR Activity must also be replanned since it is dependent on the output of the Cost Map Calculation Activity. It is actually the parent activity object, Pitch Over, which performs the combined replan of the LPR and Cost Map Calculation Activity. A change in the LPR Activity parameters could simply require a parameter update or it may trigger a replan of only the LPR Activity. A

change to a LPR Activity parameter will force it to replan itself. Depending on how algorithms are designed to react to changes to registered parameters, a different cost in time will be incurred depending on how much processing must be redone.

4.3.3 Activity Object Integration with HIMM Kernel

All Activity subclasses in the HIMM (i.e. the LPR Activity, or Cost Map Calculation Activity) extend the abstract Activity base class in the kernel. The abstract methods that must be implemented in a subclass are expected by the ADEPT_Controller class, which actuates the Activity hierarchy and handles message passing to allow for internal and external coordination. Figure 18 shows the LPR Activity class's major methods and breaks down which methods are local to the activity object and which were inherited as abstract methods to implement from the Activity base class.

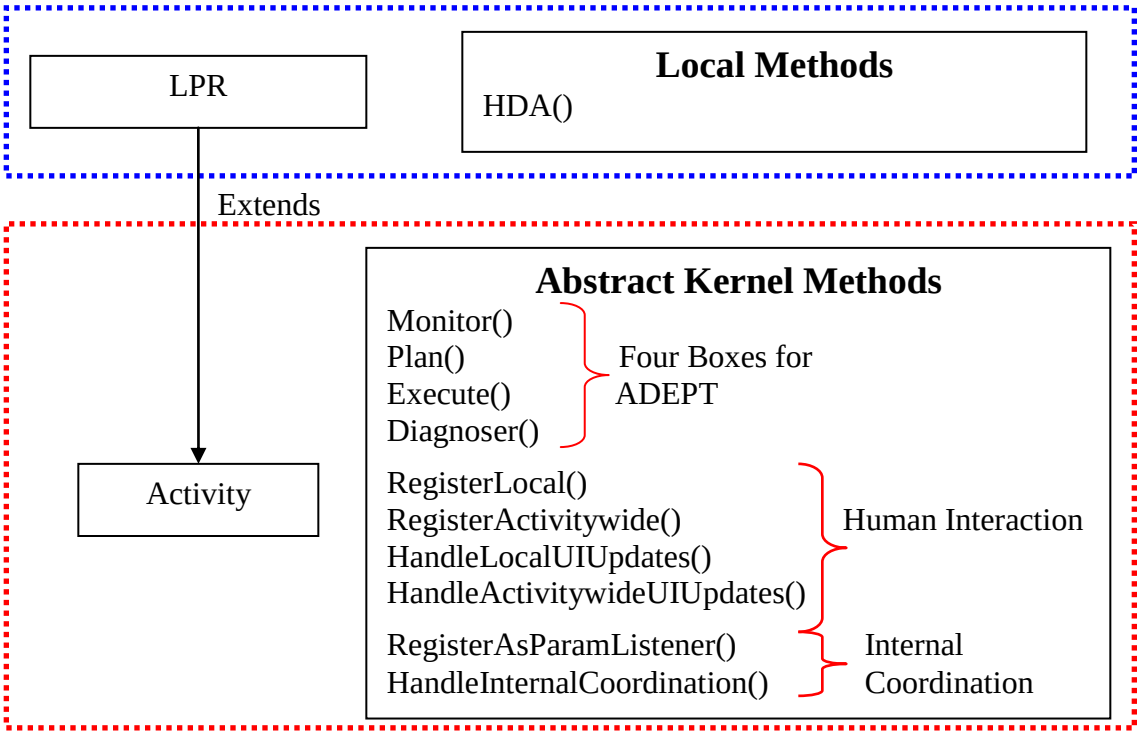

Figure 18 Activity Base class in Kernel connects Activity subclasses to the MM Kernel.

The LPR Activity is charged with using the cost maps calculated by the Map Processing Activity to find a list of aim points that the operator may choose from. The four box ADEPT functions are inherited as abstract method signatures from the Activity base class and are implemented by the LPR Activity and actuated by the ADEPT_Controller class. One of the tasks that internal coordination undertakes in the Pitch Over Activity is to receive any updates to parameters that force a replan using the HandleInternalCoordination() method. The Pitch Over Activity then prepares for and requests the replan. The final set of methods, labeled Human Interaction in the figure above, pertain to the interaction mechanisms needed to support the HIMM.

The kernel's human interaction functions are present in the form of registration functions (registerActivitywide() and registerLocal()), which the activity object uses to register its parameters as well as methods to handle UI updates to parameters (HandleLocalUIUpdate() and HandleActivitywideUIUpdate()). UIUpdate Objects received from the UI encapsulate input from the operator. Any update that is associated with a local parameter attached to this activity object will be passed into the HandleLocalUIUpdate() method before being applied to the parameter itself. For example, when an update to the Current Aim Point Local parameter arrives, the LPR Activity will ensure that the requested change is in the list of aim point options before forwarding the update to the Data class itself. Updates to Activity-wide parameters are directly forwarded to the Data class if there is no active instance of the activity object associated with the parameter in the active branch of the tree. Otherwise, the Activity-wide UIUpdate Object is sent to HandleActivitywideUIUpdates() before being forwarded to the Data class associated with the parameter. The only method in the LPR that is not inherited is the HDA() method. This method

performs the analysis using the cost maps to integrate the maps into a single cost map and come up with the list of safe aim points.

4.4 Test Scenario of HIMM Capabilities with LPR Mission

4.4.1 Testing Environment

The HIMM system is exercised using the Lunar Landing mission to examine the system's functionality and efficiency. Timing data recorded was averaged over 1000 runs of each test. The HIMM was programmed in the Java programming language, executing in a Java Virtual Machine with a garbage collection thread that is not controllable by the programmer. The garbage collection thread can take up significant processing time. Additionally the software was not run on a dedicated machine, so there were background operating system tasks also using processing cycles. Timing information is not completely accurate because of the unavoidable background processing. Therefore the data is most useful in comparison to other scenarios run on the same system. The discussion will focus on how times compare to a baseline scenario although the actual times will be presented as well. The HIMM software was run on a Microsoft® Windows XP Machine on an Intel Pentium® single-core 3.2 GHz processor with 1GB of RAM.

4.4.2 Message Passing Characterization

The HIMM system is centered on its registered parameters and the passing of messages from the MM toward the UI and vice versa. This section examines the duration of a single actuation of the activity hierarchy by the kernel, without the layer of the interaction mechanisms, and compares it to the delay in an interaction enabled system between requests for a change and the receipt of a response. This comparison is informative because update requests are forwarded

and applied once per kernel actuation loop; so a change to a parameter cannot be applied any faster than a single actuation loop. A kernel actuation loop includes initiating monitoring, executing, planning and coordinating (applying messages and forwarding responses) of the activity hierarchy. A requirement of the system is to ensure the delay between request and response is no more than 1 second to be consistent with the 1Hz frequency with which the kernel will perform an actuation loop of the activity hierarchy.

In order to get a baseline characterization of the HIMM system in action, the kernel actuation loop was timed without any interaction mechanisms enabled. The activity hierarchy had a depth of three and no processing was performed in the activity objects beyond decrementing an internal counter. Over 1000 runs, a kernel actuation loop took 0.006 milliseconds, which leaves almost all of the one second processing time for activity objects to perform their processing.

The first test examined the time for the HIMM system with interaction enabled to push an update from the MM to the UI. The timing period began with the request to change a registered parameter by an activity object in its monitoring loop and ended with the receipt of the update by the UI. The activity hierarchy size is the same as in the baseline characterization with only one activity object having a single registered parameter being changed. Delay for this task was 0.0544 milliseconds which is extremely small compared to the 1Hz time frame. The same timing test was performed when two, three and five parameters were changed during a single kernel monitoring run of the hierarchy. The added processing for each additional update to process should be linear since the update Objects are batched and sent together after all the activity objects have been actuated by the kernel. Results show that he increase was close to

linear and that delays were on the order of a hundredth of a millisecond per update which is also small compared to a 1 second time frame.

In order to perform an end to end test on the HIMM parameter system, the second test examined the HIMM system with interaction enabled to apply a UI request and then for the UI to receive a response. This test was performed with the same depth activity hierarchy, with no processing during monitoring or execution in the activity objects. The timing period began with the forwarding of the UI request and ended when a response Object is received by the UI from the MM. This is a useful test because the application and forwarding of parameter updates occurs once per kernel actuation loop. Therefore, this test shows us if there is a large amount of extra processing necessary to incorporate message passing into the HIMM. The average delay over 1000 runs of the test was 0.05 milliseconds which is much less than the one second limit. Even though this delay was significantly more than the baseline kernel actuation loop without interaction, it still leaves more than 99 percent of the one second of processing time to the activity objects. The test is repeated changing two and then three parameters at the same time and waiting for a response from the MM. Results of these test showed again that extra parameter changes only add a linear amount of extra processing on the order of 0.01 milliseconds added per request. The results are presented in Table 2 and Figure 19. Even if there were hundreds of change requests to registered parameters during an actuation loop, the aggregate processing time would be on the order of a few milliseconds, leaving 99.9 percent of available processing resources for activity objects.

Table 2 Comparing Kernel Actuation with Interaction Mechanism Overhead.

Test	Time (Milliseconds Averaged over 1000 Trials)	Standard Deviation
Baseline		
Kernel Actuation Loop without Interaction	0.0060	0.0027
Time to Push Data from MM to UI		
One Change	0.0544	0.0252
Two Simultaneous Changes	0.0661	0.0247
Three Simultaneous Changes	0.0736	0.0238
Five Simultaneous Changes	0.0844	0.0243
Round trip from Request from UI to MM Response received by UI		
One Request	0.0519	0.0224
Two Simultaneous Requests	0.0672	0.0225
Three Simultaneous Requests	0.0738	0.0209
Five Simultaneous Changes	0.0833	0.0211

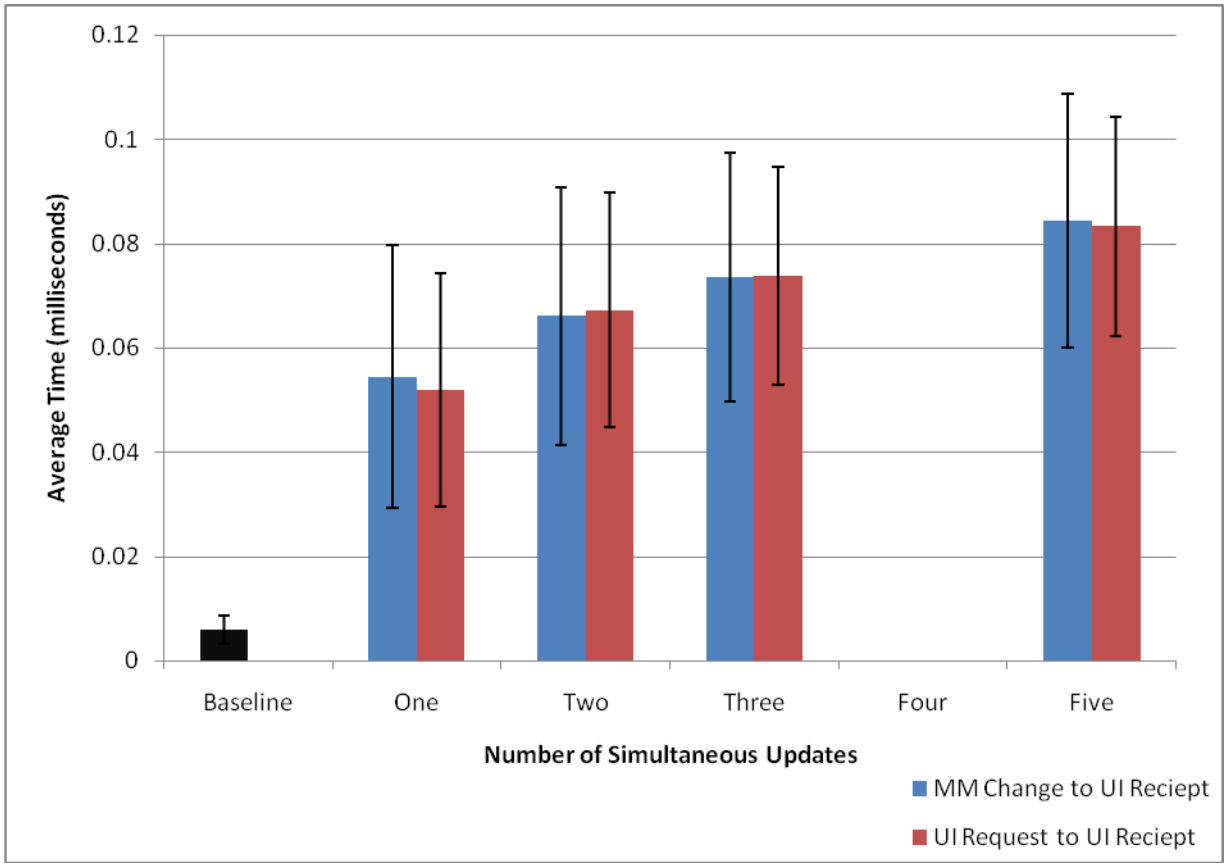

Figure 19 Graph: Comparing Kernel Actuation Loop Time with Interaction Mechanism Overhead.

The data shows that the HIMM system's extra layer of message passing places a minimal burden on resources, leaving the overwhelming remainder of a one second time block to the processing of activity objects. Much less than one percent of the time is used on kernel overhead processing. Comparing roundtrip time to a one-way timing from MM to UI, the differences are within a standard deviation of each other. This illustrates that all the processing time of updates occurs from MM application of an update to the UI receipt of the response and not during the time from UI request to the MM receipt. The HIMM provides a layer of interaction mechanisms to the MM at minimal resources costs.

4.4.3 HIMM System-level Characterization

The HIMM system is an ADEPT implementation with an added layer supporting parameter registration and outside input. Adding an extra layer of functionality should not add significant computational overhead such that it creates a resource issue for the mission designer. The HIMM mission designer must understand how changes to a parameter might cause the need to replan all or part of the mission, but should not worry about the processing overhead of message passing to keep parameters updated in the MM and the UI. In section 3.4.2, the thesis discusses that parameters in the HIMM are either classified as Activity-wide or Local, and based on this distinction are stored and retrieved in different ways. To examine the costs associated with the interaction mechanisms of the HIMM, the system was run 1000 times without any interaction mechanisms. The test was repeated with all the functionality of the HIMM in place and with three parameters registered per activity object. The same activity tree with depth three was used for all tests, and each parameter has a numerical value. The leaf activity objects in the hierarchy replan once halfway through their cycle to show that the replanning mechanisms are

also functional. Each time an activity object is monitored, it changes one parameter's value which then generates a MMUpdate Object. This test translates to an activity object changing a parameter's value every fraction of a millisecond since an activity object is monitored once per kernel Activation Loop.

The system is timed with all registered parameters classified as Activity-wide and then the test is repeated with Local parameters to explore the differences in how the system processes the two types of registered parameters. The results are illustrated in Table 3 and Figure 20. Mission timing differences between interaction mechanisms enabled versus disabled shows that there is extra processing necessary to incorporate any kind of interaction. Additionally, Activity-wide parameters are more costly than Local parameters. The difference was a 40 percent to 80 percent increase in time to run the mission if a change to the registered parameters occurred during every kernel actuation loop. This difference is expected because Activity-wide objects are stored based on the name of the activity object they are associated with and the name of the parameter while the Local parameters are retrieved based on only their name. The extra search to retrieve an Activity-wide parameter from a Map explains the difference.

The resources needed to run a pseudo-mission that included more than 70 kernel actuation loops ran in close to 4.5 milliseconds, which is in itself small in comparison to the one second requirement for just a single kernel Actuation loop. This test was a stress test, running the kernel actuation loops in quick succession to illustrate the difference in resource usage with the HIMM's interaction mechanisms enabled. The extra processing less than doubled the resource usage for background processing in the worst case, which would still leave more than 99 percent of processing time to activity objects. Therefore, the HIMM system does not increase

computational complexity by more than a constant factor (dependent on the number of simultaneous updates).

Table 3 Comparing Mission Timing with and without Interaction Mechanisms Enabled.

Test	Time (milliseconds Averaged over 1000 Trials)	Standard Deviation
Run Mission without Interaction mechanisms	2.467	0.012
Run Mission with Interaction (Local Parameters)	3.494	0.031
Run Mission with Interaction (Activity-wide Parameters)	4.373	0.026

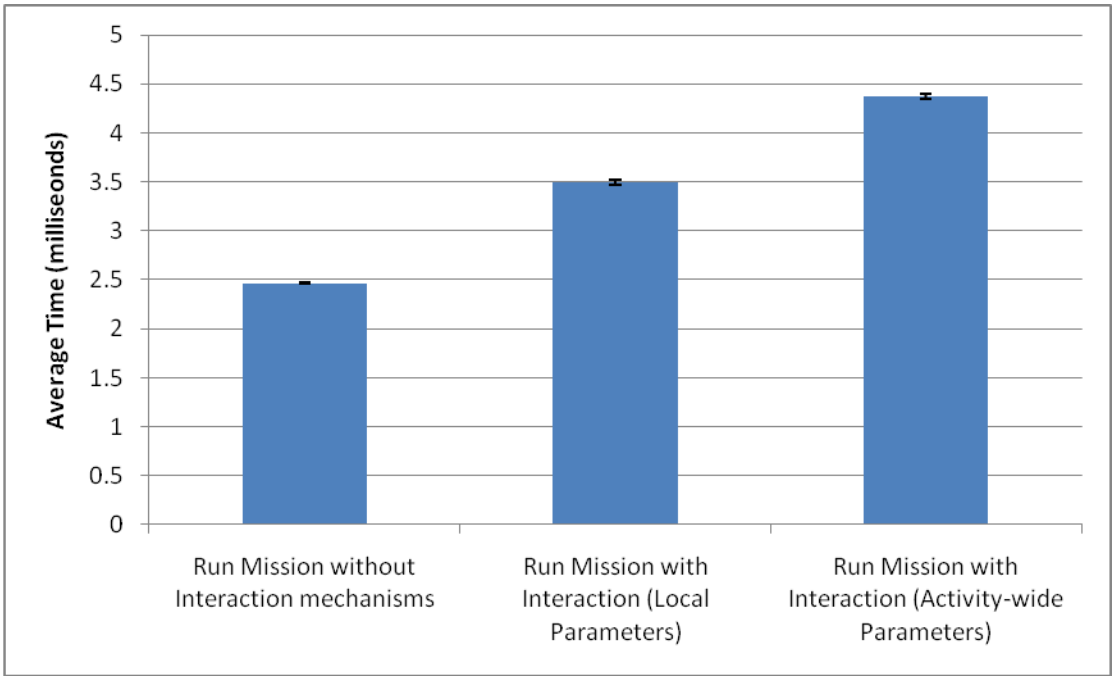

Figure 20 Graph: System Timing of a Mission with and without Interaction Mechanisms

4.4.4 LPR Testing Scenarios and Rationale

Timings taken using the Lunar Landing mission are focused on showing how parameters used within the same algorithm can have different costs based on the design of the algorithm. In this scenario, the operator has a limited amount of time with which to make a decision on a

landing point. If each change to a parameter results in a replan that causes a significant delay, the operator will only be able to make a limited number of changes before a decision must be made. This test is focused on illustrating that a mission and algorithm designer must consider the amount of extra processing necessary for the UI to receive the output expected from a parameter change initiated by a human operator.

The Landing Point Redesignation portion of the Lunar Landing mission has simple logic and does not have any large background processing occurring outside of computing cost maps and generating landing point options. The reason for performing timing calculations on the parameters in the landing mission is to highlight the costs of different parameters in replanning time versus the human's added control associated with the parameter's registration. As discussed in Section 4.3.2, changing the parameters in the landing point mission will have different costs attached to them in the form of replanning an activity object or just added calculation. This section will reinforce that claim by examining the duration between a UI request for a parameter change and when the expected effect of the change reaches the UI. Changes in the Spread, Cost Map Weightings, and Slope Tolerance parameters will be explored and average timings taken over 1000 such parameter changes to illustrate the delay between the change and the expected result.

Changes to parameters will have difference effects based on how the algorithm and mission are designed. In the Lunar Landing mission, changing the Spread parameter causes the LPR Activity to access a pre-calculated array and choose another entry of landing point options. The LPR Activity then updates the parameter associated with the Aim point options. Since there is no heavy calculation, the expected time to update the UI with a set of aim points is on the order of the delay to send an update to the UI from the MM and perform an array copy operation.

The results show that the time associated with this change is less than a millisecond (0.2659 milliseconds) which is consistent with a small amount of processing and a message passing delay.

The Landing Point Redesignation algorithm has many points of manipulation and the amount of time from changing a parameter to receiving the outcome of the change may vary. A change to the Cost Map Weightings parameter results in a replan of the entire LPR Activity. The time between a request from the UI change the Cost Map Weightings parameters and the resulting replan and processing of a new set of aim points took 172.09 milliseconds with a standard deviation on the order of a hundredth of a millisecond over 1000 changes. Changing the Slope Tolerance parameter has an even greater effect on the system since changes to this parameter means that both the Map Processing Activity and the LPR Activity must be replanned. This operation took 525 milliseconds with a standard deviation also on the order of a hundredth of a millisecond. The computation to replan an activity object is much greater than the overhead of message passing but even these replans take less time than the one second time period for a single Actuation Loop. The operator would consequently not lose a great deal of time by forcing a replan since the processing needed would be small. Figure 21 and Table 4 show the results of the timings from change request to the receipt of the expected result.

Changing parameters can force replans that take a substantial amount of time and must be characterized so the costs of a change can be understood. This example shows that the delay between request and response can vary greatly and could affect the number of changes an operator can make during an interaction time window. However, the costs can be characterized, and with coordination between designers, these costs can be minimized. If costly replans are necessary, this information must be brought out to the operator to aid in making good use of the

interaction window. The HIMM system adds a layer of processing to the baseline ADEPT implementation and a layer of extra design to the job of an algorithm and mission designer.

Table 4 Landing Point Parameter Timings until output is received.

Parameter Name	Expected Result	Average Time (milliseconds)	Standard Deviation
Spread	New Set of Aim Points	0.2659	0.0026
Cost Map Weightings	New Set of Aim Points	172.0906	0.0120
Slope Tolerance	DTNH Cost Map / New Set of Aim Points	525.4434	0.0176

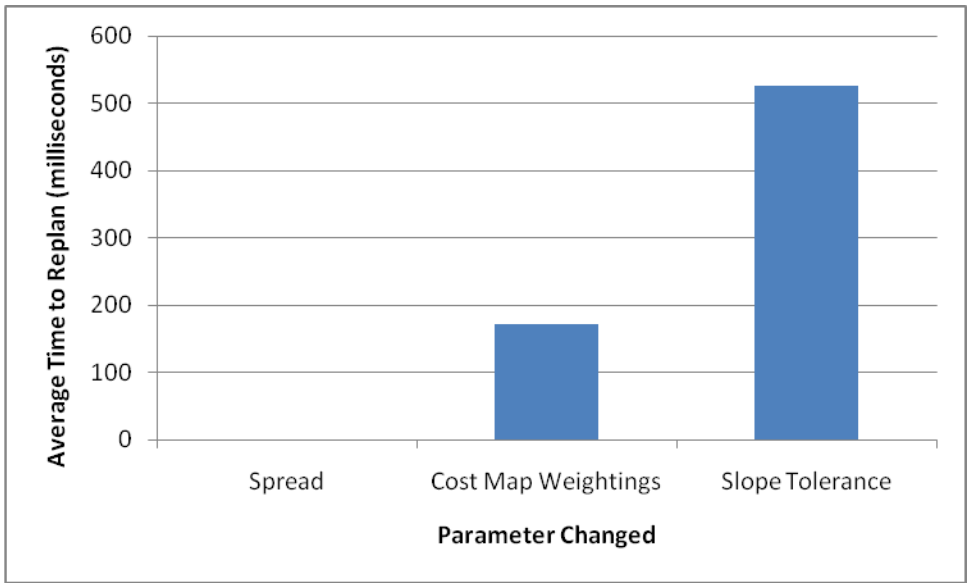

Figure 21 Graph: Delay between Parameter Change and UI Receipt of Expected Result.

Chapter 5

5. Conclusion

5.1 Summary and Contributions

The Human Interactive Mission Manager (HIMM) project enhances Draper Laboratory's ADEPT capability towards becoming a more human interactive execution and planning technology. Human interaction is enabled in this system through the use of registered parameters in the mission that are wrapped in constraints that control when parameters can be manipulated and limit parameter changes. The layered design of the Maritime Open Autonomy Architecture, the baseline ADEPT implementation used for the HIMM system, simplified the modification of the existing structure to support interaction mechanisms. The HIMM project provides the expected functionality of an ADEPT implementation, and succeeds in adding interaction mechanisms without excessive computational overhead.

5.1.1 HIMM Implementation

The requirement on the HIMM implementation of the ADEPT technology is that the computational overhead must take up less than one percent of the computational resources of a one second kernel actuation interval. Characterization of the system shows that with interaction enabled, overhead is linear on the order a hundredth of millisecond per UIUpdate Object received during a single actuation loop. Therefore, the performance of the system is consistent with its performance requirement. These tests were run on a desktop computer running the

Microsoft ® Windows XP operating system in the Java programming environment which has a number of background processes taking up computing resources at intervals outside of the control of the programmer. If the system were run on a dedicated machine, performance would improve.

The reference architecture implementation for the HIMM is programmed in the Java programming language which provides automatic garbage collection and runs in a virtual machine. These utilities are background processes that are not controllable by the programmer. To apply the HIMM to a real-time system, it may be advantageous to translate the architecture to another programming language without as many uncontrollable utilities. Translating the HIMM to other object oriented languages (like C++) is possible with some design adjustments. The ability of the Java programming language to make a method protected allows classes within a package to have access to these methods but not classes from outside. This ability was used to separate the kernel functionality from the activity object subclasses. In order to allow such a conversion, there would still need to be a separation between the public methods provided by the kernel and the methods that are internal to the functioning of the kernel structure. This could be accomplished through the use of more interfaces. Such nuances between the Java programming language and other object oriented languages would have to be identified and solved before a successful conversion could occur.

5.1.2 Landing Point Mission

Exercising the abilities of the HIMM system was achieved through the application of the HIMM on the Landing Point Redesignation portion of the larger Lunar Landing mission. This mission allowed the registration of a variety of parameters with constraints on value changes and

an opportunity to interact with the data. Changes to these parameters could result in replanning of the activity object or multiple activity objects. There were a limited number of activity objects in this exercise, but the example mission still exercised the main functionality of the HIMM system. The Landing Point Redesignation mission served to illustrate the new level of design and cooperation necessary between algorithm designers, human factors engineers and user interface specialists to be able to create a system for a human to understand, interact and be comfortable with automation processes.

The Landing Point Redesignation mission did not fully explore the possibilities of the HIMM system. All levels of human interaction supported by the system were not tested in this mission since the parameters were all part of a single task to choose a landing aim point for the Lunar Lander in a preset amount of time. Such a small test also did not need to support a great deal of coordination between activity objects within the activity hierarchy. Overall, the mission did illustrate the new capabilities of the HIMM system to support human interaction with a mission manager.

5.1.3 Mission and Algorithm Design Tasks

In the past, mission designers could view algorithms as a black box and algorithm designers did not have to worry much about the overall context that their algorithms would be used. The HIMM system encourages a deeper collaboration between these designers in order to decide what information needs to be brought out to the user to understand the context of the algorithms. This thesis applied the HIMM system to the Lunar Landing mission scenario to explore the delay that can occur between parameter change and the desired result arriving in the UI. The test showed that the amount of time can vary greatly depending on the side-effects of a

parameter change which can include the replan of multiple activity objects. Therefore, a mission designer, human systems collaboration engineer and algorithm designer must agree on what parameters are important, what the costs are to making a change to the parameter and what constraints on changes are necessary for the algorithm to function deterministically and provide useful results.

5.1.4 Summary

The HIMM design succeeded in achieving its objective to enable human interaction with a mission using registered parameters as an input mechanism to the mission's plan without a prohibitive amount of added computation. Implementation of the system is an object oriented programming language (Java) and it maintains a layered architecture that separates the data used by the autonomy software from the data being manipulated by the human operator. All parameters that a human can access have been explicitly registered and have been wrapped in metadata describing the limits of how and when the parameter can be manipulated. Results of testing showed that kernel processing with interaction was small enough that more than 99% of available processing time is available for activity objects. The HIMM system design is expected to be adapted to the mission manager that will be used in the Autonomous Landing and Hazard Avoidance Technology (ALHAT) project at Draper Laboratory for NASA.

5.2 Future Work

The HIMM project succeeded in achieving its objective of enabling interaction, but there are limitations to the system. With more time it would have been possible to make the kernel functionality more complete, to explore what the HIMM can support, and to examine how best to create missions that are easy to use and understand. Such an exploration encompasses algorithm

design, human systems collaboration engineering, mission design and user interface design. Designing algorithms that allow a human to easily adjust its operation, understanding which and how parameters should be registered, generalizing the responsibilities of different types of activity objects in a mission, creating guidelines to incorporate registered parameters in a complex mission, and designing usable interfaces for such a dynamic system are all important areas of future research.

The HIMM system is a tool that supports the implementation of missions and algorithms that can provide human interaction in decision making. Future work is vital in researching how interaction points in a mission should translate into the activity objects registration of parameters for input. As a UI problem, the operator must be able to quickly recognize what parameters are there for information versus interaction purposes, how changing parameters will affect a mission, how much of a change is permitted, and the length of delay caused by the extra computation associated with the change of that value. For the mission designer, activity objects are no longer solely responsible with coordinating with other activity objects, but also must make their operation transparent to an operator. Developing guidelines for deciding what parameters to register and how they should be registered is vital to the utility of the system and mission.

Applying the HIMM to new domains may bring out the need for further evolution of the ADEPT architecture to support the kinds of functionality necessary. The Lunar Landing mission is an example of a mission in need of a dynamic decision aid. The HIMM system could also be integrated into a semi-autonomous personal automobile in which the driver can select a destination and then interact in real-time to determine and execute the best route. A different domain might explore the differences between creating a mission for a manned system versus one controlled remotely. An interesting research question in this area would explore if the

control dynamics change the amount of information that must be displayed so the remote operator can understand the present state before interacting. Each domain presents its own set of challenges and it would be enlightening to examine whether the HIMM supports missions in all these domains without alteration.

The HIMM system is an enabling tool to better support human computer collaboration in decision making, and much work must be undertaken to fully utilize the applications made possible by such a system. This software system is capable of supporting development and testing of dynamic human-automation mechanisms. Adding human interaction mechanisms as a fundamental service of a mission manager encourages the exploration of domain and mission areas from Lunar Landers to our own automobiles. As human's better leverage computation, there is a need to better leverage the abilities of computation while avoiding burdening ourselves with an unnecessary cognitive burden because of a lack of design and foresight in such systems. Computation is a tool to serve us and make our lives easier, efficient and more enjoyable. The HIMM system is an attempt to provide a framework to enable a designer to spend more time creating usable and exciting applications rather than becoming muddled in the limitations of a system that was not designed with the human operator as a fundamental actor in the operation of the overall system.

Bibliography

[1] Sheridan, Thomas. *Telerobotics, Automation and Human Supervisory Control.* Cambridge, MA: MIT Press, 1992.

[2] Sarter, N. "Making coordination effortless and invisible: The exploration of automation management strategies and implementations," presented at the *CBR Workshop Human Interaction Automated Systems*, June 1, 1998.

[3] Ricard, M. and S. Kolitz, "The ADEPT Framework for Intelligent Autonomy," *VKI Lecture Series on Intelligent Systems for Aeronautics*, von Karman Institute, Brussels, Belgium, May 2002.

[4] Boyd, J. "A Discourse on Winning and Losing." Maxwell AFB, AL: *Air University Library*, Document No. M-U 43947, August 1987.

[5] Adams , M. B., Deutsch, O. L., and Harrison, J.V., "A Hierarchical Planner for Intelligent Systems," In Proceedings of *SPIE - The International Society for Optical Engineering*, Vol 548, Arlington, VA, April 9-11, 1985.

[6] Abramson, M., et al., "The Design and Implementation of Draper's Earth Phenomena Observing System (EPOS)," in *Proceedings of the AIAA Space 2001 Conference*, Albuquerque, NM, August 2001.

[7] Adams, M, et al., "Closed-Loop Operation of Large-Scale Enterprises: Application of a Decomposition Approach," in *Advances in Enterprise Control*, Minneapolis, MN, July 2000.

[8] Barth, C.D., "Composite Mission Variable Formulation for Real-Time Mission Planning," SM Thesis, Massachusetts Institute of Technology , 2001.

[9] Abramson, M.; Cleary, M.; Kolitz, S., "Steps Toward Achieving Robust Autonomy," in *Proceedings of the AAAI Spring Symposium*, Stanford, CA, March 2001.

[10] Ricard, Michael, Margaret Nervegna, Michael Keegan and Marilyn Jarriel, "Autonomy and Control Architectures", in *Proceedings of the AUVSI Unmanned Systems 2005 Conference*, Baltimore, MD, June 2005.

[11] Hildebrant, Richard. "A Framework for Autonomy." in Proceedings of the *SPIE Boston Robotics Conference SA110: Intelligent Robots and Computer Vision XXIV: Algorithms, Techniques, and Active Vision - The International Society for Optical Engineering*, Boston, MA, October 2006.

[12] Epp, C.D. and Smith, T.B. "The Autonomous Precision Landing and Hazard Detection and Avoidance Technology (ALHAT)." In Proceedings of the *2007 IEEE Aerospace Conference - Institute of Electrical and Electronics Engineers*, Big Sky, MT, 2007.