

SENG 697 Final Project

Design Documents for U2-MPS
Using Tropos

By: Ben Law
Session: F06

October 29, 2006

Table of Contents

INTRODUCTION	4
SYSTEM SPECIFICATION	4
<i>EARLY REQUIREMENT PHASE: IDENTIFYING STAKEHOLDERS AND INTERESTS</i>	4
<i>LATE REQUIREMENT PHASE: SYSTEM OVERVIEW</i>	6
ARCHITECTURAL DESIGN	9
<i>MISSION EDITOR (OPERATIONAL CORE)</i>	10
<i>MISSION MANAGER (MIDDLE LINE)</i>	11
<i>OPTIMIZER (DECISION MAKER)</i>	11
<i>DATABASE (SUPPORT)</i>	11
<i>ANALYST (TECHNOSTRUCTURE)</i>	11
DETAIL DESIGN	12
<i>MISSION EDITOR AGENT</i>	12
<i>MISSION MANAGER</i>	14
<i>OPTIMIZER</i>	16
<i>DATABASE</i>	18
<i>ANALYST</i>	20
PROTOCOL	21
<i>MESSAGES WRAPPER</i>	21
<i>MISSION DESCRIPTION MESSAGES</i>	22
MESSAGE 101 – MISSION MESSAGE	22
MESSAGE 102 – WAYPOINT MESSAGE	22
MESSAGE 103 – LOITER ACTION MESSAGE	23
MESSAGE 104 – CAMERA ACTION MESSAGE	23
MESSAGE 105 – TARGET ACTION MESSAGE	24
MESSAGE 106 – MISSION TRANSFER STATUS	24
MESSAGE 107 – MISSION SUMMARY	25
SERVICE PROFILE MESSAGES	25
MESSAGE 201 – FLIGHT PARAMETERS MESSAGE	25
MESSAGE 202 – CAMERA SERVICES MESSAGE	25
MESSAGE 203 – WEAPON SERVICES MESSAGE	26
MESSAGE 204 – SERVICE PROFILE TRANSFER MESSAGE	26
INTELLIGENCE EXCHANGE MESSAGES	27

MESSAGE 301 – TERRAIN INFORMATION MESSAGE	27
MESSAGE 302 – WEATHER CONDITION MESSAGE	27
MESSAGE 303 – INTELLIGENCE TRANSFER MESSAGE	27
MESSAGE 304 – MISSION EXECUTION PROGRESS REPORT MESSAGE	28
MISSION 305 – UAV PERFORMANCE HISTORY	28
MISCELLANEOUS MESSAGES	29
MESSAGE 401 – UAV DISCOVERY MESSAGE	29
MESSAGE 402 – GENERIC PARAMETER LIST MESSAGE	29
MESSAGE 403 – GENERIC PARAMETER REQUEST MESSAGE	30
USE CASES	30
USE CASE #1: GATHER AND STORE UAV SERVICE PROFILES	30
USE CASE #2: REQUEST CAST RECOMMENDATIONS	31
USE CASE #3: UPLOAD MISSION	32
<u>REFERENCE</u>	32

INTRODUCTION

Conventional UAV (Unmanned Aerial Vehicle) systems allow the planning of UAV missions based on a specific vehicle type. A mission usually consists of a set of waypoints (coordinate locations) that a UAV is designated to reach; and a corresponding set of actions that the UAV is supposed to perform on arrival at each waypoint, known as waypoint actions. The type of waypoint actions supported varies greatly between different UAV systems. Depending on the capabilities of UAVs (e.g. – EO/IR cameras, weapons, chemical-sensor... etc.), UAV systems may have different waypoint actions. Consequently, conventional mission planning software is often tailored to the capabilities and constraints of one specific vehicle type only.

The conventional vehicle-specific approach in mission planning is viewed as a capability deficiency by the NATO alliance, as this approach does not support interoperability between UAV systems of different nationalities (i.e. – UAVs from different countries or manufacturers are unable to work together to execute a mission). There needs to be a system that allows mission commanders to plan missions involving different types of UAV systems, compliant to a common communication protocol, thereby taking advantage of the wide variety of capabilities available within NATO forces.

The Universal UAV Mission Planning System (U2-MPS) is an agent-based system that offers mission commanders with an integrated user interface for planning missions involving multiple UAV types. A common communication protocol known as STANAG-4586 (Standardization Agreement 4586) has been developed to allow compliant systems to communicate and thereby interoperate. A broadcast mechanism is built into this protocol to allow an instance of U2-MPS to enlist for services that it needs from the collection of UAV systems presented on a secured network. U2-MPS is to make recommendation as to the most cost-efficient and/or potent (level of confidence) deployment strategy to complete a mission. CDL Systems Ltd, a Calgary-based company specialized in VCS (vehicle control station) software is contracted to develop U2-MPS. The Tropos Agent-based Development Methodology is used to develop U2-MPS.

SYSTEM SPECIFICATION

Early Requirement Phase: Identifying Stakeholders and Interests

The stakeholders of this project are the NATO UAV-Interoperability Working Group (standardization organization), Mission Commanders (users), Defense Research and Development Canada (client) and CDL Systems Ltd. (contractor). In the early requirement phase, the actor and dependency model of Figure 1 is derived. Actors represent stakeholders and their interests are represented by associated goals.

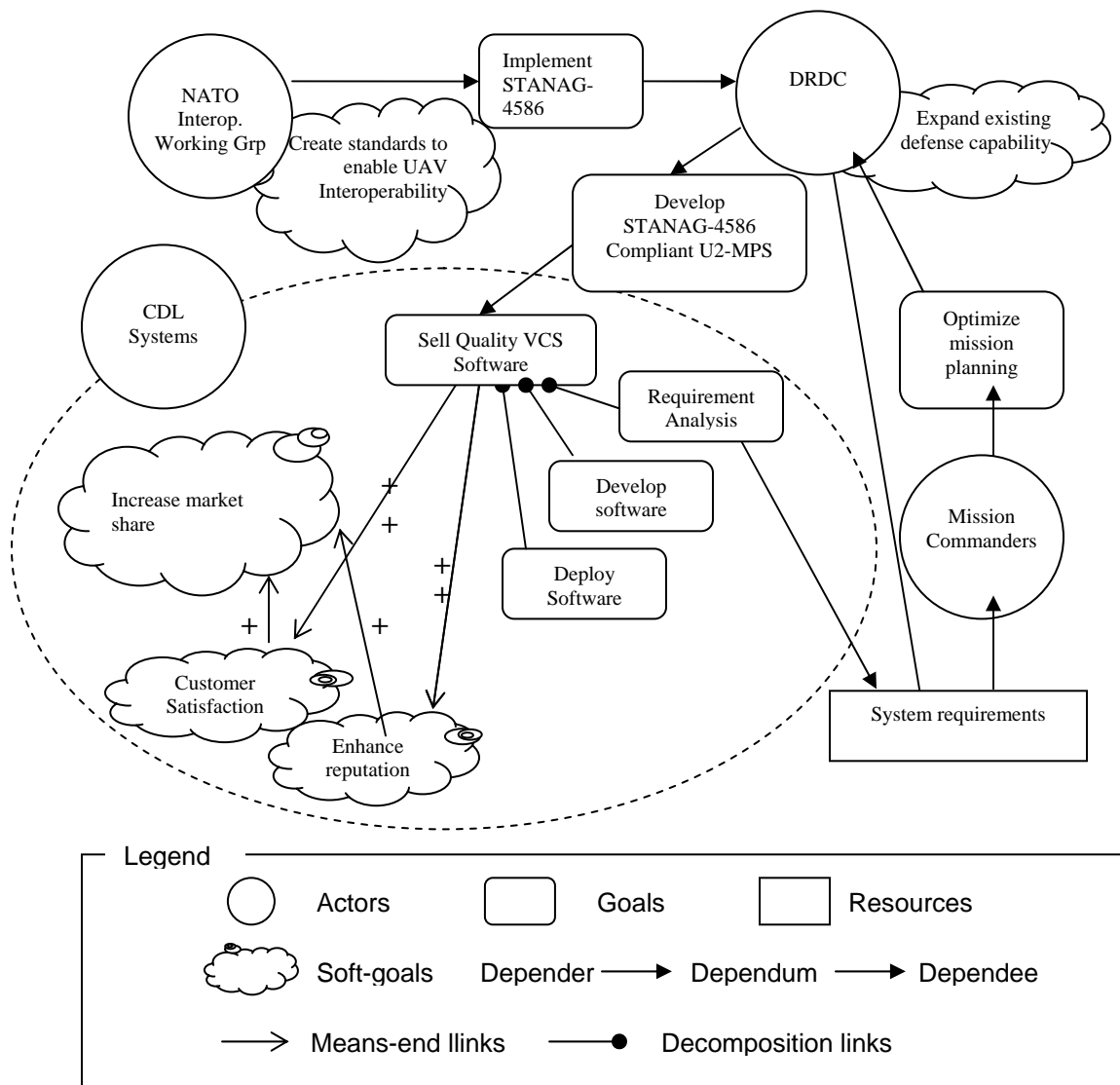


Figure 1: Early Requirement Phase Actor and Dependency Model

References to elements in the model are in *italic* format, a convention that will persist throughout this document. The *NATO Interoperability Working Group* is formed with the objective to *create standards to enable UAV interoperability*. To put new standards into practice, the working group depends on *DRDC*, among other defense research institutions to implement such standards. *DRDC* sees the opportunity to *develop a STANAG-4586 compliant U2-MPS* as a way to fulfill its goal to continually *expand existing defense capability*. *DRDC* has contracted *CDL Systems* to develop U2-MPS. The actor, *CDL Systems*, is analyzed in more detail in Figure 1 as its internal goals are illustrated within the dotted-line circle. *CDL Systems* wants to *increase market share* in the UAV control station market, and is set out to achieve this goal by *Enhance reputation* and achieve *Customer Satisfaction*. Both of these soft goals are achievable by the goal: *sell quality VCS software*. The goal to *sell quality VCS software* consists of three sub-

goals: *Requirement Analysis, Develop Software, and Deploy Software*. To achieve *requirement analysis*, *CDL Systems* depends on *Mission Commanders (user)* and *DRDC* to deliver *system requirements* as a resource.

Late Requirement Phase: System Overview

The analysis of early requirement phase has concluded that *DRDC* and *Mission Commanders* and the two main sources for system requirements. After much correspondence with these two stakeholders, functional requirements of the *U2-MPS* system are obtained and presented in Figure 2.

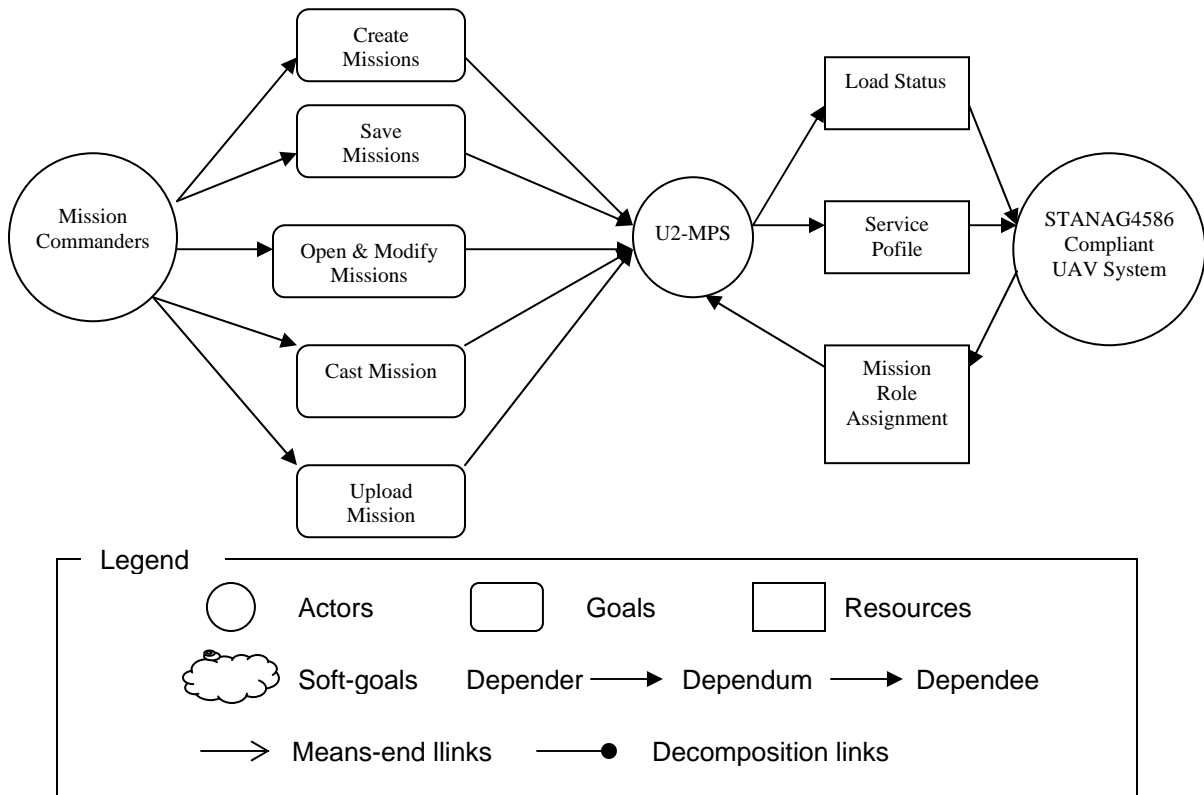


Figure 2: Late Requirement Phase Actor and Dependency Model

In the late requirement phase, the system-to-be is introduced and its dependencies with external actors identified. Figure 2 illustrates that the *U2-MPS* system is to fulfill five high-level goals for the *mission commander*:

- Create missions: Compose a list of waypoints and associated waypoint actions
- Save missions: Save the list of waypoints and associated waypoint actions to a file
- Open and modify missions: Open a mission file and modify its contents
- Cast mission: Fill the roles (capabilities) required for a mission by casting UAV systems that can best fulfill these roles
- Upload mission: Inform UAV systems of their roles in the mission

To achieve these high-level functional requirements, the *U2-MPS* system depends on instances of *STANAG-4586 Compliant UAV Systems* to provide *Service Profile* that details the services that are offered by a given UAV Systems, as well as the *Upload Progress* information while mission upload is in progress. The individual *STANAG-4586 Compliant UAV Systems* depends on the *U2-MPS* to “assign work” to them by granting them *mission role assignments*.

To further breakdown the 6 high-level requirements, these goals are further refined into smaller sub-goals within U2-MPS. This breakdown is achieved by deriving a goal model focusing on the system-to-be rather than external actors. After a series of means-end analysis performed on the model presented in Figure 2, a more refined goal model is produced and presented in Figure 3.

Figure 3 identifies 13 tasks that U2-MPS must implement to satisfy all its requirements, they are identified by ◊ in Figure 3.

1. Display coordinates in table
2. Display coordinates on map
3. Enter longitude and latitude in table
4. Pick coordinate from map
5. Open and save waypoint data
6. Display waypoint actions
7. Present available waypoint actions (as options)
8. Choose waypoint actions
9. Find available UAVs
10. Send mission role assignments (to UAV systems)
11. Sort casting options by potency
12. Sort casting options by cost
13. Display casting options

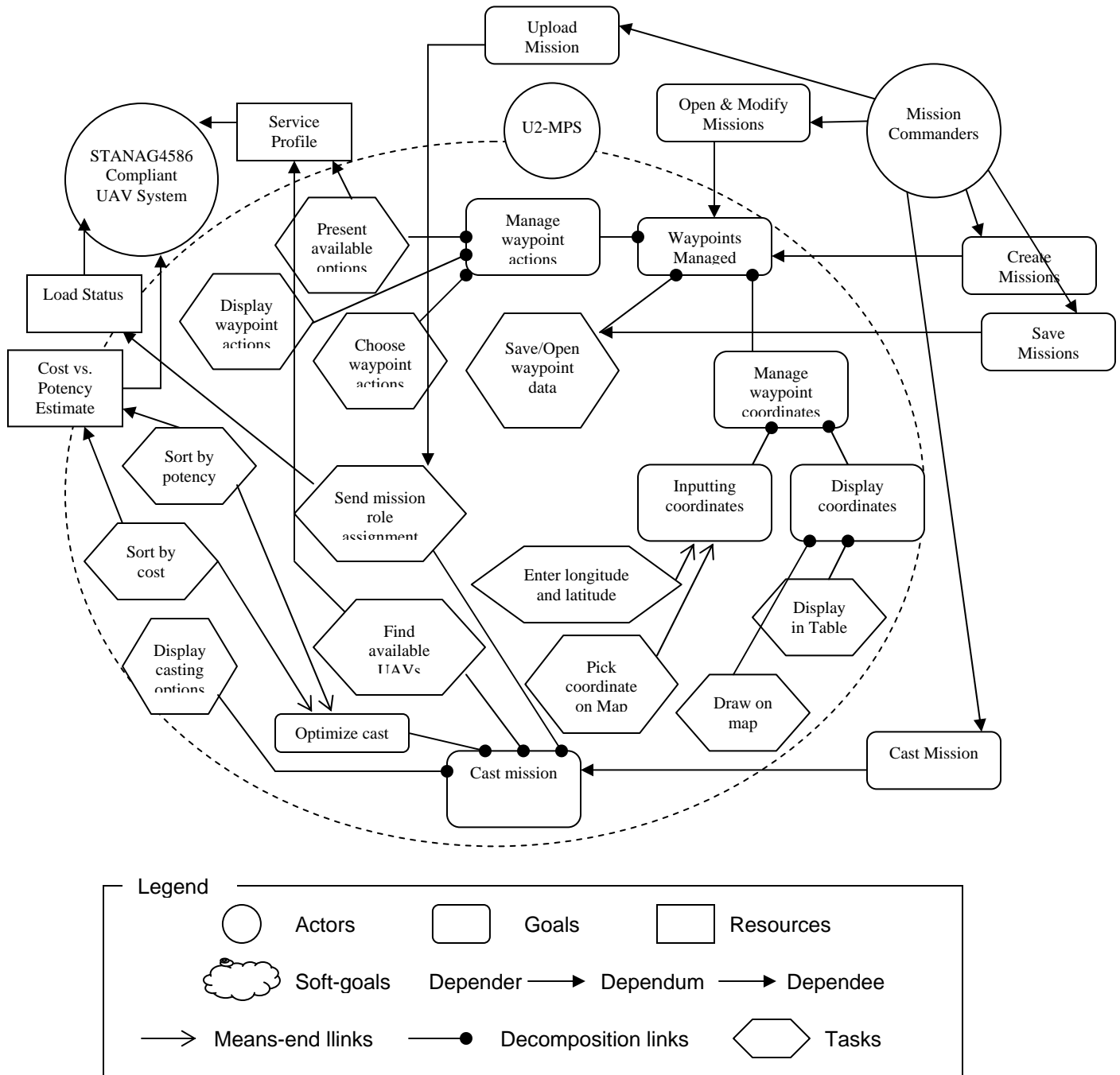


Figure 3: Late Requirement Phase – U2-MPS Goal Model

In order to sort casting options by potency/cost, the U2-MPS requires cost and potency estimates form each UAV system on the potential assignment(s), a newly discovered dependency.

ARCHITECTURAL DESIGN

In the architectural design phase, sub-system actors internal to the U2-MPS system are introduced to take on different portions of the set of responsibilities assigned to U2-MPS. In this project, the Structure-in-5 agent architectural pattern is employed considering the system's conceptual similarity to a store. System is divided into 5 components: Operational Core, Strategic Apex, Middle Line, Technostructure and Support.

Structure-in-5 proposes that an agent system can be structured in a way that resembles how human organizations are structured. According to [1], Operational Core is responsible for carrying out the core services of the organization. The Strategic Apex is responsible for making executive decisions to guide the organization to achieve its overall goal more effectively given its environment. Middle Line is a collection of managers that supervises the Operational Core and forms a hierarchy of authority between the Operational Core and the Strategic Apex. Technostructure is responsible for optimizing certain operations within the organization. Finally, Support is responsible for providing services that are consider peripheral services that supports the core service of the organization. In the U2-MPS system, the primary services are:

1. Specification of mission
2. Cast for a mission in a way that optimizes cost or potency

Table 1 below shows the subsystem actors introduced to take on each of the 5 roles, and the primary responsibility of each.

Subsystem actor	Role	Responsibility
Mission Editor	Operational Core	Provides user interface for specifying mission requirements. It is also responsible for searching for available UAV systems who can potentially fulfill the role requirements of a given mission.
Optimizer	Strategic Apex	Decides on the best cast for a mission based on field environment, past experience and advertised potency/cost estimates. It draws on the capability provided by Analyst to reach decisions.
Mission Manager	Middle Line	Perform higher-level tasks such as uploading a mission to its cast and reporting summary of mission parameters.
Analyst	Technostructure	Provides algorithms and computational methods to evaluate service profiles provided by external UAV systems, calculate mission potency, calculate mission cost.
Database	Support	Database of service profiles, mission history and environment conditions. Serves as the source of information for other components.

Table 1: Architectural Components of U2-MPS

The interaction between these subsystem actors is depicted in Figure 4 below. Subsystem actors identified here will be implemented as individual agents.

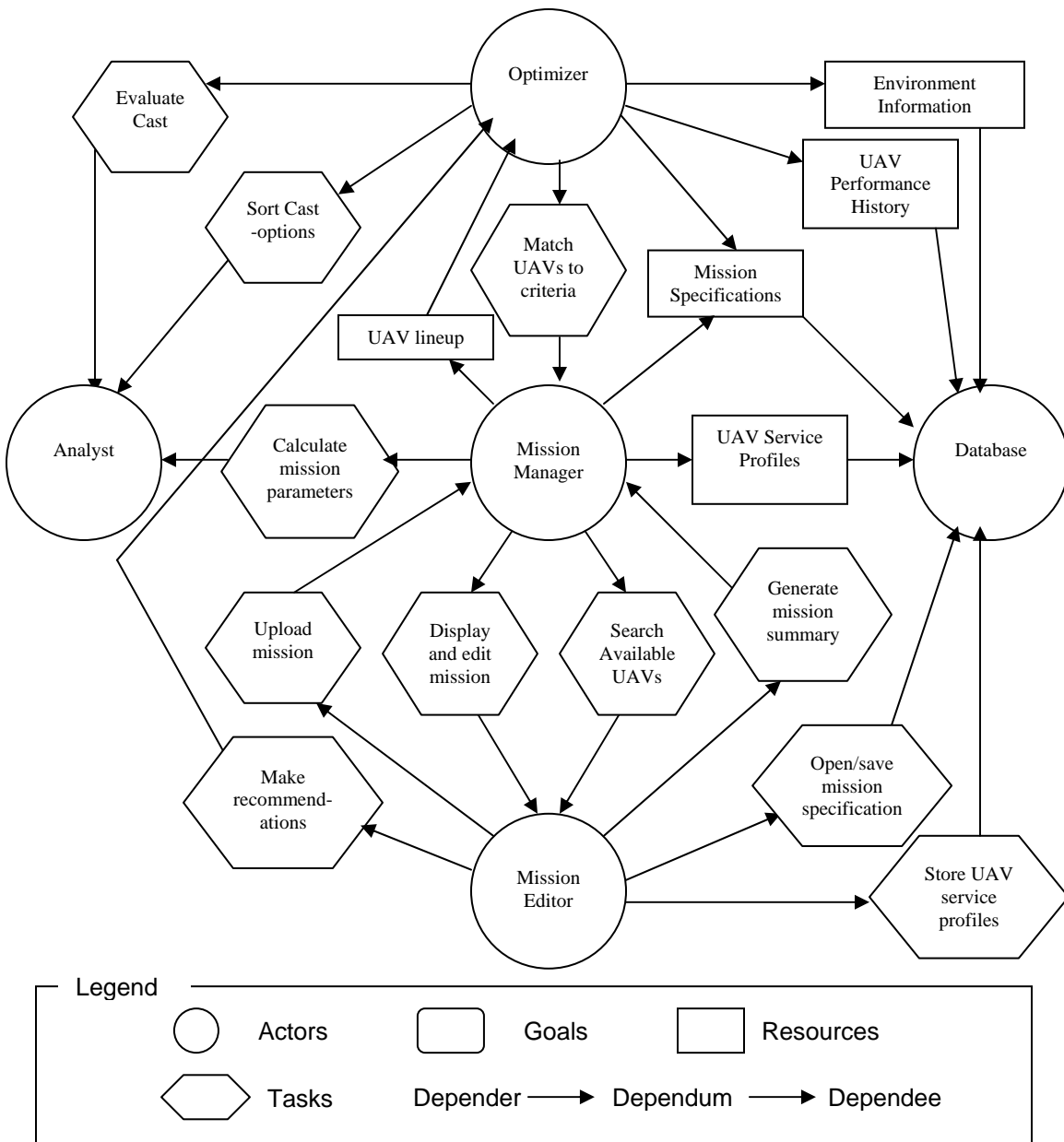


Figure 4: Architectural Design of U2-MPS

The interactions illustrated in Figure 4 will be discussed in detail focusing on one of the 5 subsystems actors at a time.

Mission Editor (Operational Core)

The mission editor is depended upon to *display and edit mission*. The *Mission Editor* is likely to present a GUI for the user to enter mission specifications. The *Mission Editor* is also asked to search for available UAV systems on the network. Search results are obtained in the form of service profiles provided by the external UAV systems. The *Mission Editor* depends on the *Database* actor to store received *UAV service profiles*. When the user chooses the save/open a mission, the *Mission Editor* relies on the *Database* to *open/save mission specifications*. *Mission Editor* relies on the *Mission Manager* to perform higher-level mission manipulation tasks including *Upload Mission* and *Generate Mission Summary*. When the user asks the system to determine a cast for the mission, *Mission Editor* calls upon the *Optimizer* to *make recommendations*.

Mission Manager (Middle Line)

Mission Manager needs to *generate mission summary* and to *upload mission* to the UAV systems that are chosen for the mission. Both of these tasks are triggered through the GUI of *Mission Editor*. In turn, it relies on the *Mission Editor* to *display and edit mission* and to *search available UAVs*. In order to generate mission summary, *Mission Manager* relies on *Analyst* to *calculate mission parameters*, such as mission duration and mileage. The *optimizer* depends on the *Mission Manager* to *match UAVs to criteria*, so that the optimizer can determine the UAV system that best matches a mission role description. The *Mission Manager* obtains *UAV Service Profiles* from *Database* to perform criteria matching.

Optimizer (Decision Maker)

Optimizer has one goal: to *make recommendation* as to the best cast (collection of UAVs) to use for a given mission specification. The *Optimizer* requires several inputs from the *Database* actor to make recommendations, these include: *Environment information*, *UAV performance history* and *mission specification*. *Optimizer* relies on the *Mission Manager* to match mission requirements to available UAVs, and analysis all inputs with the help of *Analyst*, who provides capability to *evaluate cast* and *sort cast options*.

Database (Support)

Database serves as the source of information for other components. The types of information made available to other components include: *Environment information*, *UAV performance history*, *mission specifications (from Mission Editor)*, and *UAV Service Profiles (from Mission Editor)*. The *Mission Editor* relies on the *Database* to *open/save mission specification* and *store UAV service profiles*.

Analyst (Technostructure)

The capabilities provided by *Analyst* include *evaluate cast* (in terms of potency and cost), *sort cast-options* (by potency or cost), and *calculate mission parameters* (required in mission summary report). *Analyst* does not depend on other subsystems.

DETAIL DESIGN

Each subsystem actors (agent) identified in the architectural design phase will be further decomposed into smaller actors to take on specific tasks. The approach is to first extract the dependencies associated with a subsystem actor (e.g. - Mission Editor), and tackle each task dependency with a collection of sub-actors. The actor and dependency models built with these sub-actors will eventually be translated into class diagrams.

Mission Editor Agent

First we extract all the dependencies related to the Mission Editor actor from Figure 4 into Figure 5 below. Each dependency is color-coded with a unique color, this is so that we can map each of these external dependencies into one or more dependencies between the sub-actors within *Mission Editor*, as will be illustrated soon.

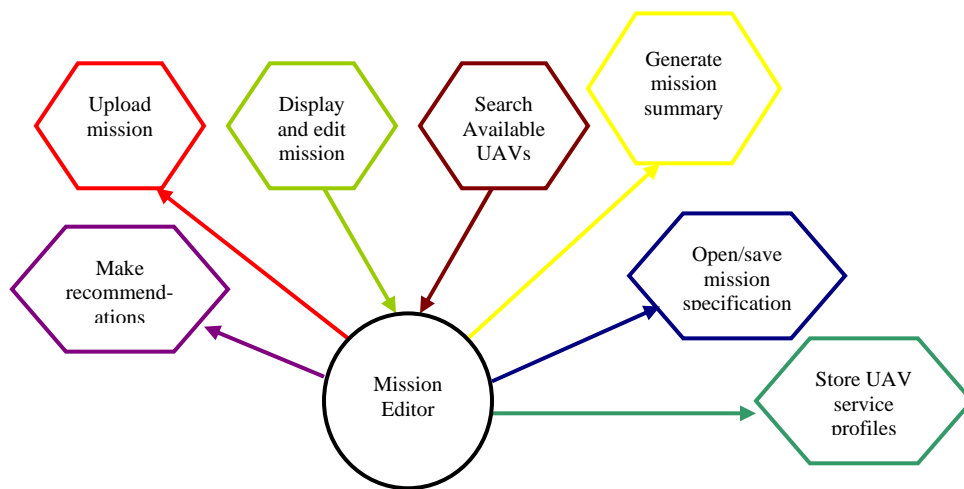


Figure 5: Mission Editor External Dependencies

To address the *Display and edit mission* dependency (in light-green), we employ the model-view-controller (MVC) pattern. The *WaypointDataModel* actor is introduced as the data storage model of all waypoint related data, and it is shown near the center of Figure 6. The *WaypointListView* actor is responsible for displaying the waypoint data in *WaypointDataModel*. *WaypointController* is responsible for servicing user commands to open, save and edit mission data. The *WaypointListView* depends on the *WaypointDataModel* to emit notifications whenever data changes, so that the display can be updated accordingly. As implied by the color coding, the *Display and edit mission* external dependency is sufficiently satisfied by the combination of *update model, notify changes and display data* dependencies between the sub-actors in figure 6, also presented in light-green.

To address the *Open/save mission specification* dependency (blue), we let the user initiate this action through the *WaypointListView* actor, who will relay on *WaypointController* to service these requests. *WaypointController* subsequently asks the *WaypointDataModel*

to store (save) or recall (open) mission data from the *Database* actor. Here, we have introduced the *DatabaseIO* actor as the primary point of contact with the *Database* actor identified as one of 5 architectural actors.

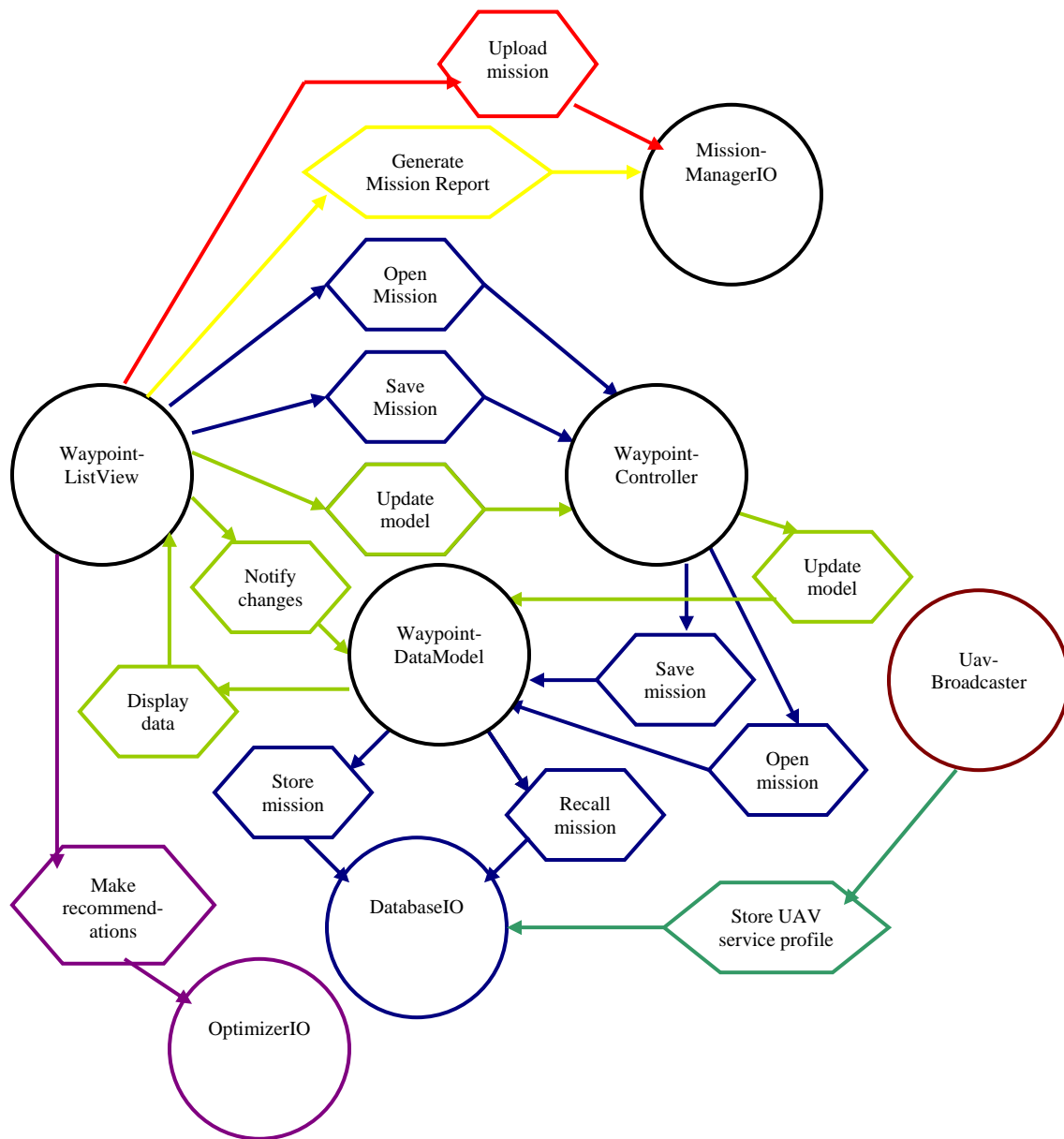


Figure 6: Decomposed Mission Editor

The Mission Editor depends on the Mission Manager for two tasks: *Upload Mission* and *Generate Mission Report*. These two tasks are being handled by the *MissionManagerIO* actor, who serves as the primary point of contact with the Mission Manager. Mission Editor also depends on the Optimizer to *make recommendations* to the user as to the UAVs to employ for a mission for optimized cost/potency. The *OptimizerIO* actor is introduced as the primary point of contact with the *Optimizer* actor.

Finally, the Mission Editor is responsible for finding available UAVs on the network, and stores this information via the *database* actor. The *UAV broadcast* actor is introduced to take on this task, and it depends on *DatabaseIO* to store the *UAV service profiles* it receives.

The sub-actors identified in Figure 6 are translated into class diagrams for later implementation. This is illustrated in Figure 7, Mission Editor class diagram.

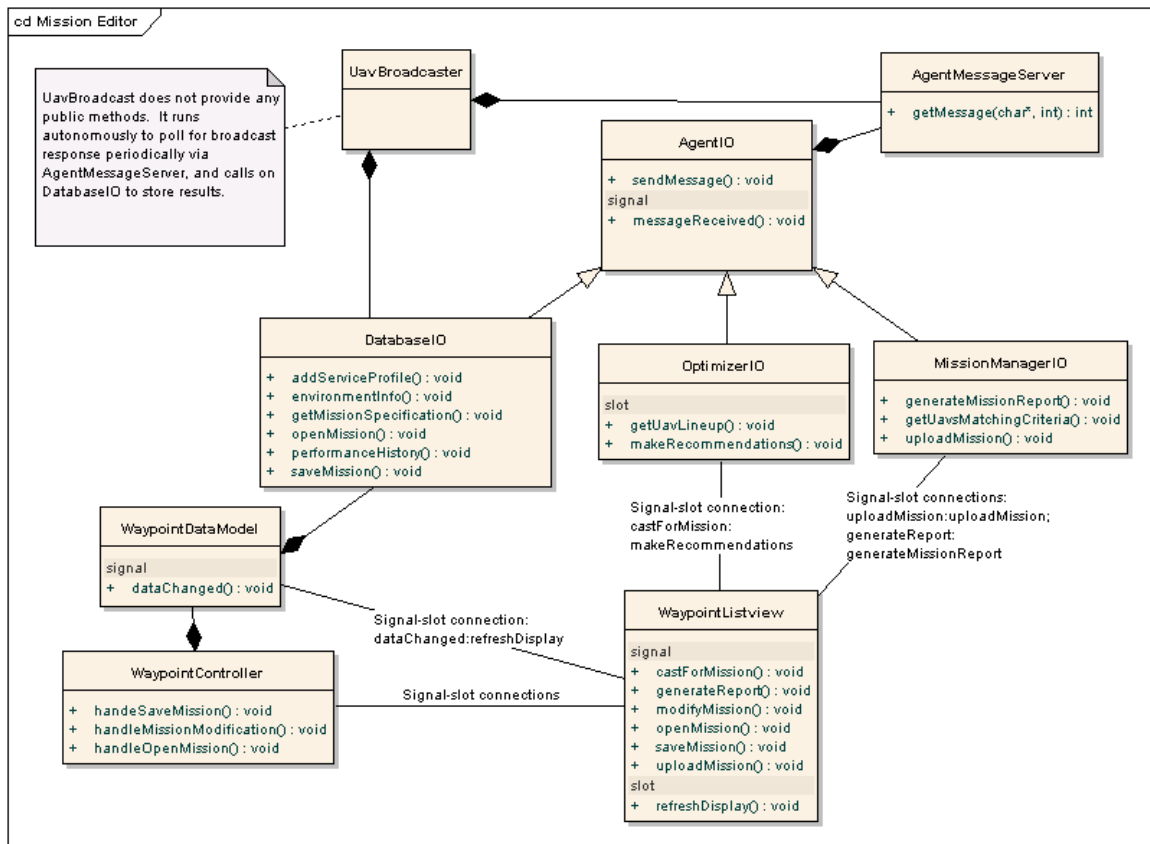


Figure 7: Mission Editor Agent Class Diagram

In the Mission Editor class diagram above, notice that there are several instances of a specially labeled associations called “Signal-slot connection”. This is a feature provided by the Qt development tool kit, where classes that subclasses from a Qt-defined QObject will be able to emit signals and catch signals with slots. Both signals and slots are implemented as public methods, and will be invoked by the Qt meta-object architecture. It is assumed that Qt will be used as our development tool kit in this project and therefore we can rely on the signal slot mechanism to be available throughout the design process.

Mission Manager

Using the same approach used for refining Mission Editor, we first extract the external dependencies of *Mission Manager*, as shown in Figure 8.

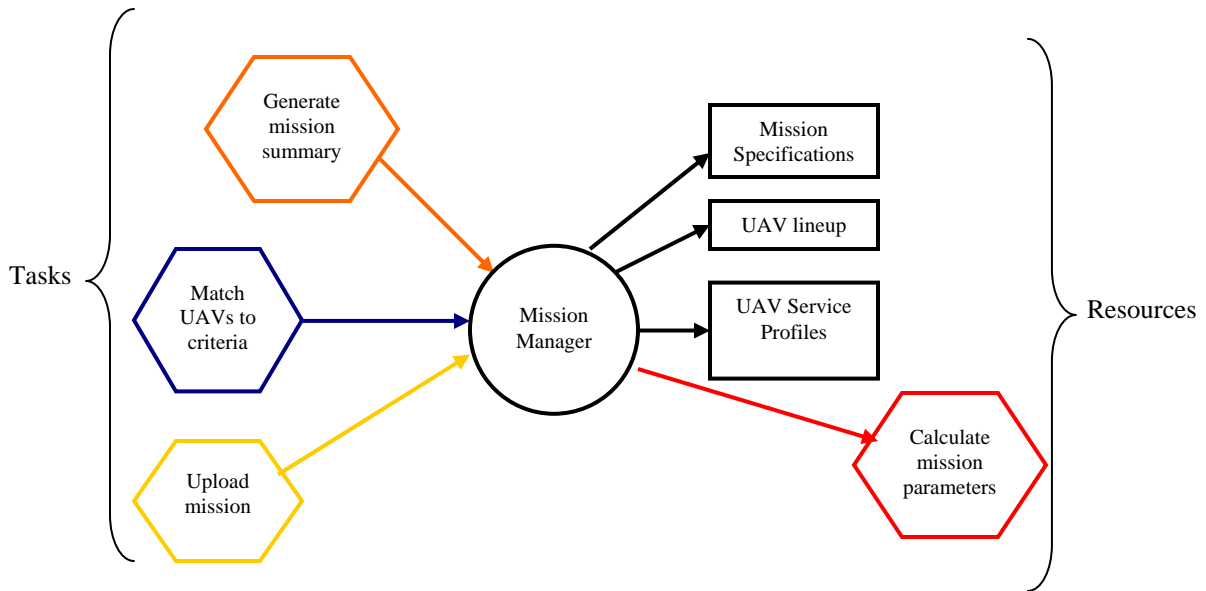


Figure 8: Mission Manager External Dependencies

The Mission Manager is assigned three main tasks as shown on the left side of Figure 8: *Generate mission summary*, *Match UAVs to criteria*, and *Upload mission*. To fulfill these tasks, it has 3 resources at its disposal all provided by the *Database actor*. It may also depend on *Analyst* to *calculate mission parameter*. Note that the *Display and edit mission* and *Search available UAVs* tasks are omitted in Figure 8. This is because these two tasks are handled autonomously by *Mission Editor*.

To be able to obtain the crucial resources from *Database*, another instance of the *DatabaseIO* actor is reused here to communicate with the *Database actor*. Specifically, the *MissionReporter* actor is to obtain *Mission specifications* from the *Database actor*.

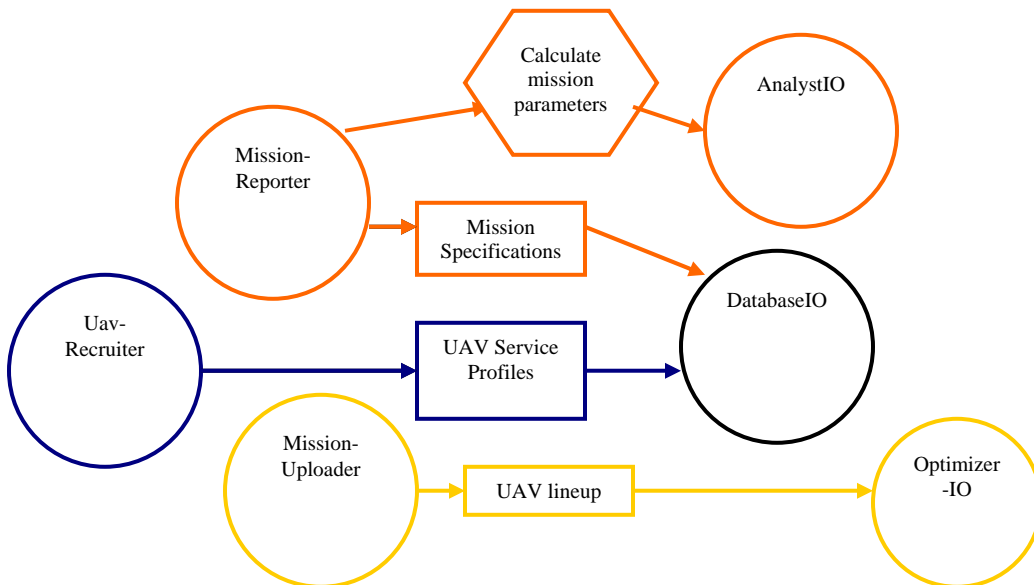


Figure 9: Decomposed Mission Manager

The *UAVRecruiter* obtains the list of *UAV Service Profiles* that matches certain criteria (given by *Optimizer*) from *Database*. On the same token, an *OptimizerIO* actor is reused here to obtain the *UAV lineup* resource, which is the finalized UAV lineup that is chosen by the user from the list of recommendations generated by the *Optimizer*. The *MissionUploader* actor depends on *OptimizerIO* to obtain the finalized lineup in order to determine the properly recipients of mission assignments. The *AnalystIO* actor is the central point of contact with the *Analyst* actor to tap into the collection of computational capabilities provided by *Analyst*. In this particular case, the *Mission Reporter* actor is to solicit help from *Analyst* to *Calculate mission parameters*, and report these results in the report that it generates. The decomposed Mission Manager model can be translated into the class diagram in Figure 10.

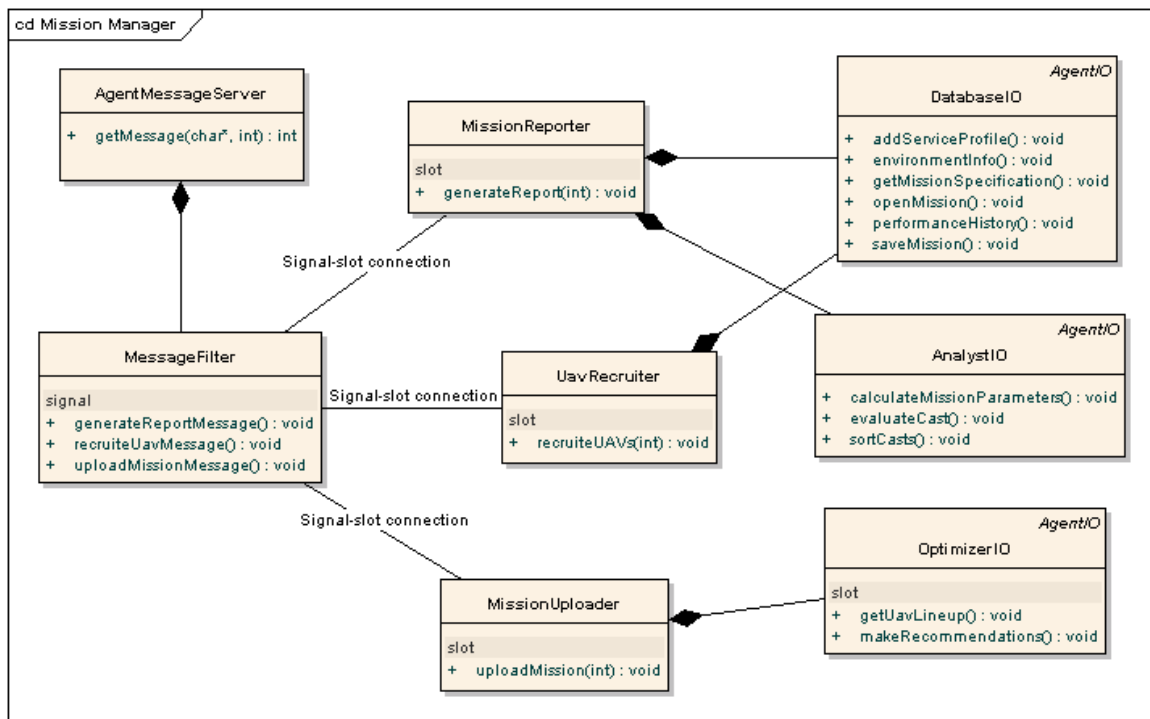


Figure 10: Mission Manager Class Diagram

The *MessageFilter* class receives all external agent messages via *AgentMessageServer*. If the received message is a *generateReportMessage*, *recruiteUavMessage* or an *uploadMissionMessage*, the *MessageFilter* is to emit signals notifying the proper handler. The handlers are *MissionReporter*, *UavRecruiter* and *MissionUploader*. They make use of subclasses of *AgentIO* to solicit help from external actors to complete their respective task.

Optimizer

External dependencies associated with *Optimizer* are shown in Figure 11.

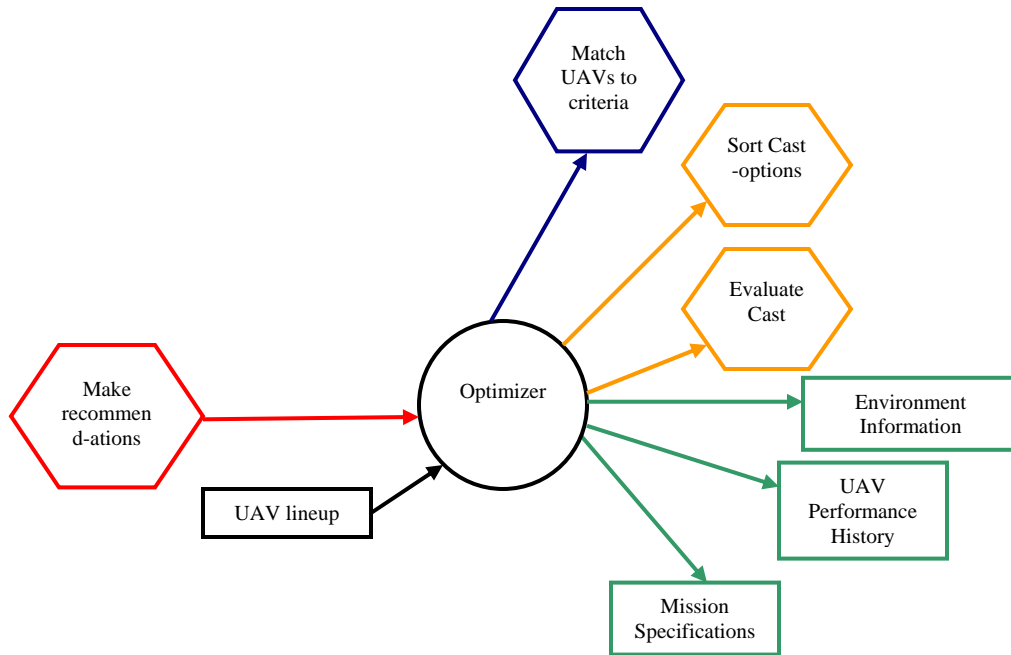


Figure 11: Optimizer External Dependencies

The *Optimizer* has one task, which is to make recommendations as to the best casting for a mission in a way that optimizes cost or potency. Such request for recommendations is sent from the *MissionEditor* actor. Upon reception of this request, the *Optimizer* first obtains the *Mission Specifications* from *Database*. Based on *Environment Information* obtained from *Database*, the *Optimizer* then breaks down the *mission specifications* into a set of UAV selection criteria. For example, weather condition may have an effect on UAV selection. The *Optimizer* then asks the *Mission Manager* for a list of UAV matching the selection criteria. Given a list of UAVs, the *Optimizer* composes all possible combinations of UAV selection, and calls upon the *Analyst* to evaluate each casting option. Finally, the *Optimizer* sorts the casting options with the help of *Analyst* and returns the final recommendations to *Mission Editor*. Figure 12 shows the sub-actors inside *Optimizer* and their dependencies.

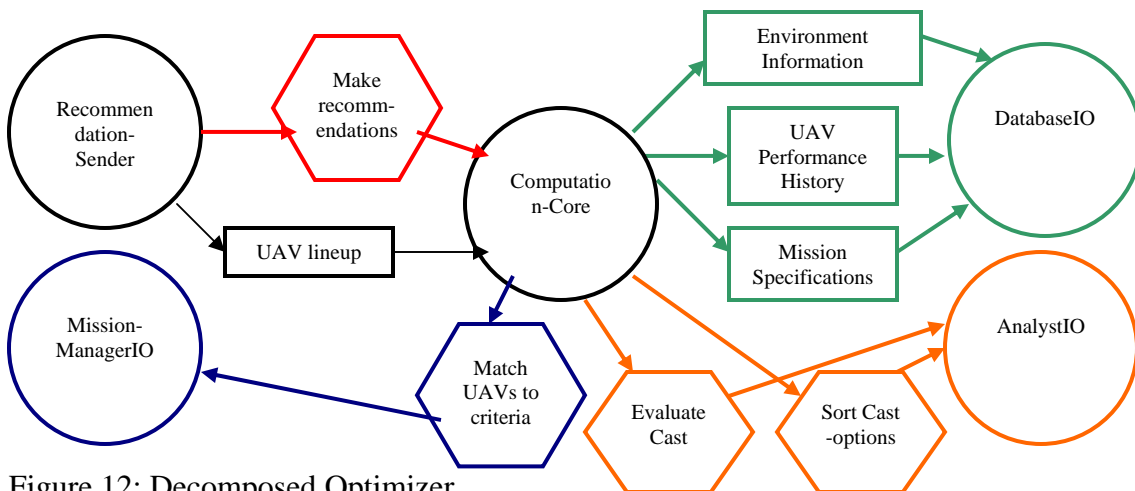


Figure 12: Decomposed Optimizer

The model in Figure 12 is further translated into the class diagram of Figure 13.

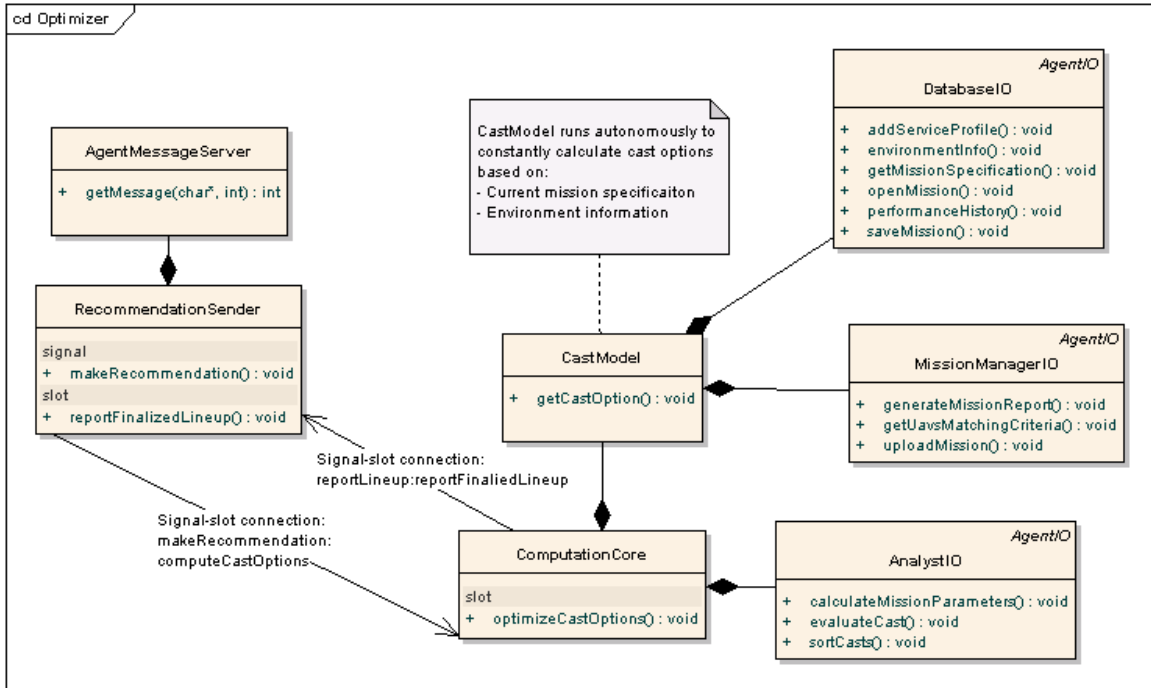


Figure 13: Optimizer Class Diagram

The RecommendationSender is responsible for receiving requests for recommendations (from Mission Editor) and/or finalized cast option (from Mission Manager). The RecommendationSender subsequently emits signals to the ComputationalCore where a large number of cast options stored in the CastModel will be evaluated and sorted with the help of Analyst via AnalystIO. The CastModel stores a comprehensive list of cast options given the current environment information and mission specifications. CastModel constantly updates its list of cast options, with input data supplied by DatabaseIO and MissionManagerIO.

Database

Figure 14 shows the list of external dependencies associated with Database.

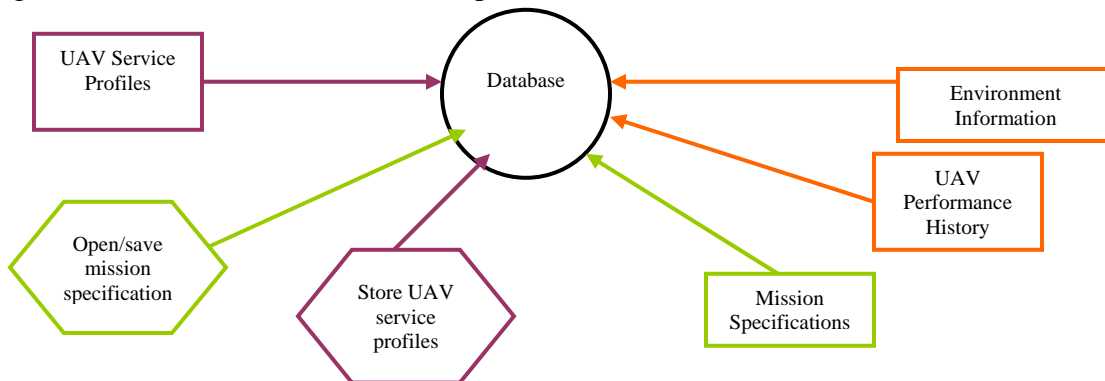


Figure 14: Database Agent External Dependencies

To address the need to provide access and storage to a variety of information, the information broker pattern is employed. The central *Information Broker* actor services external requests for information, and searches for the requested information in one of 4 database actors: *ServiceProfilDb*, *MissionDb*, *EnvironmentDb* and *HistoryDb*. Each database actor is populated via different IO actors. The *ServiceProfileIO* actor receives new service profiles from the *Mission Editor* and stores them into the *ServiceProfilDb*. The *MissionSpecificationIO* actor receives mission specifications from the *Mission Editor* and stores them into the *MissionDb*. The *IntelligentFinder* actor receives intelligence from external UAV systems and store data into the *EnvironmentDb* or *HistoryDb* depending on the nature of the data.

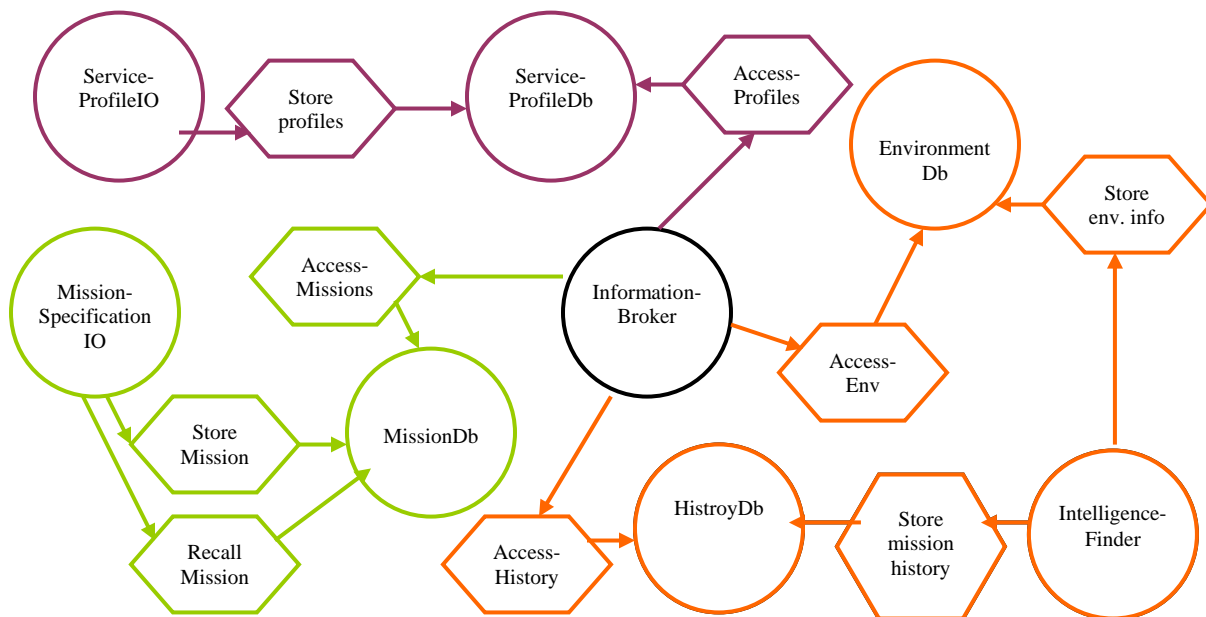


Figure 15: Decomposed Database Agent

The model presented in Figure 15 is then translated into the class diagram of Figure 16. *ServiceProfileIO*, *MissionSpecificationIO* and *IntelligenceFinder* are all able to receive messages from external agents via the *AgentMessageServer* class. As new data are received from the message server, signals are emitted to the corresponding database class so that the information can be stored. *InformationBroker* is able to receive requests for information from external agents, and will get the information out of the properly database class and return the requested data via the *reply()* method.

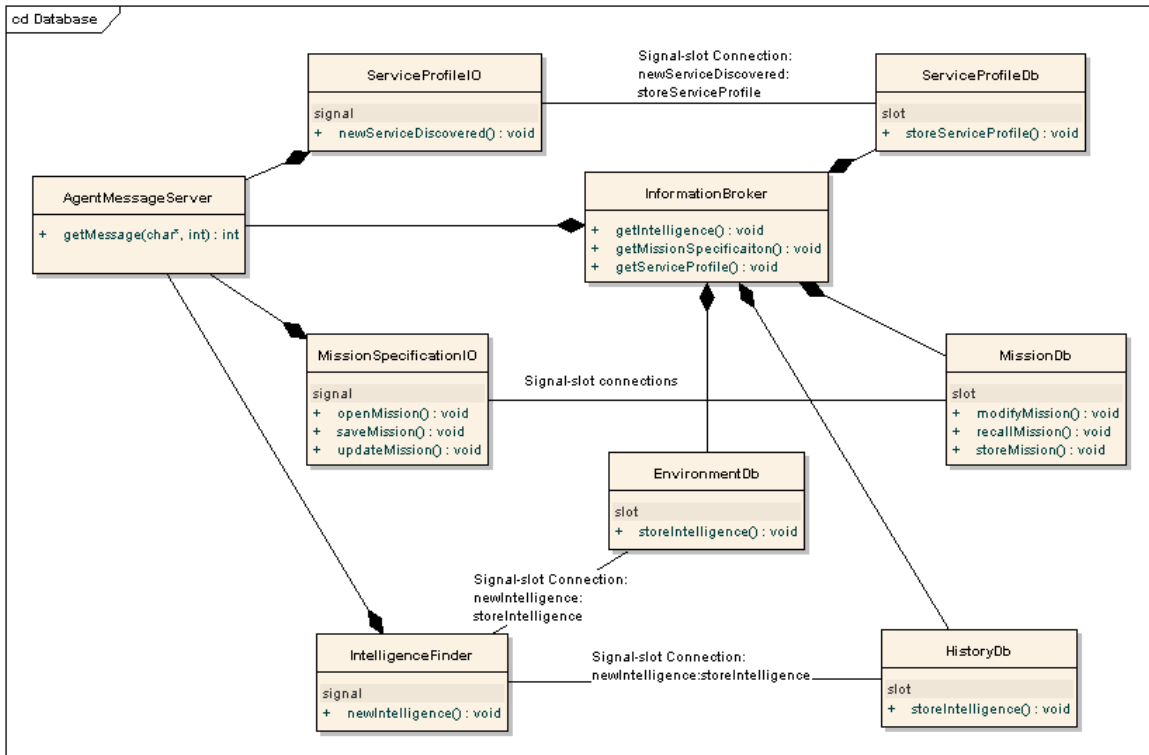


Figure 16: Database Agent Class Diagram

Analyst

Figure 17 shows the external dependencies of the *Analyst* actor.

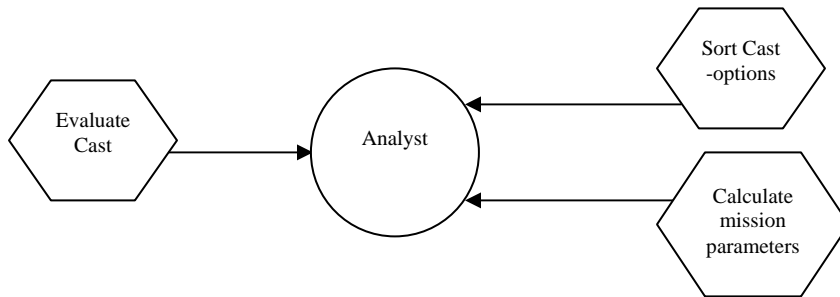


Figure 17: External Dependencies of the Analyst Actor

Analyst needs to take input data from service-requesting agents, perform computations and send the output parameters from the computation back to the requesting agent. The *ParameterIO* actor is introduced to receive input parameters and send output parameters to and from the external agent requesting a particular service. Three service provider agents are introduced, one for each computational need required of *Analyst*. Because of the simplicity of the *Analyst* actor, we jump straight to developing a class diagram, and the result is shown in Figure 18.

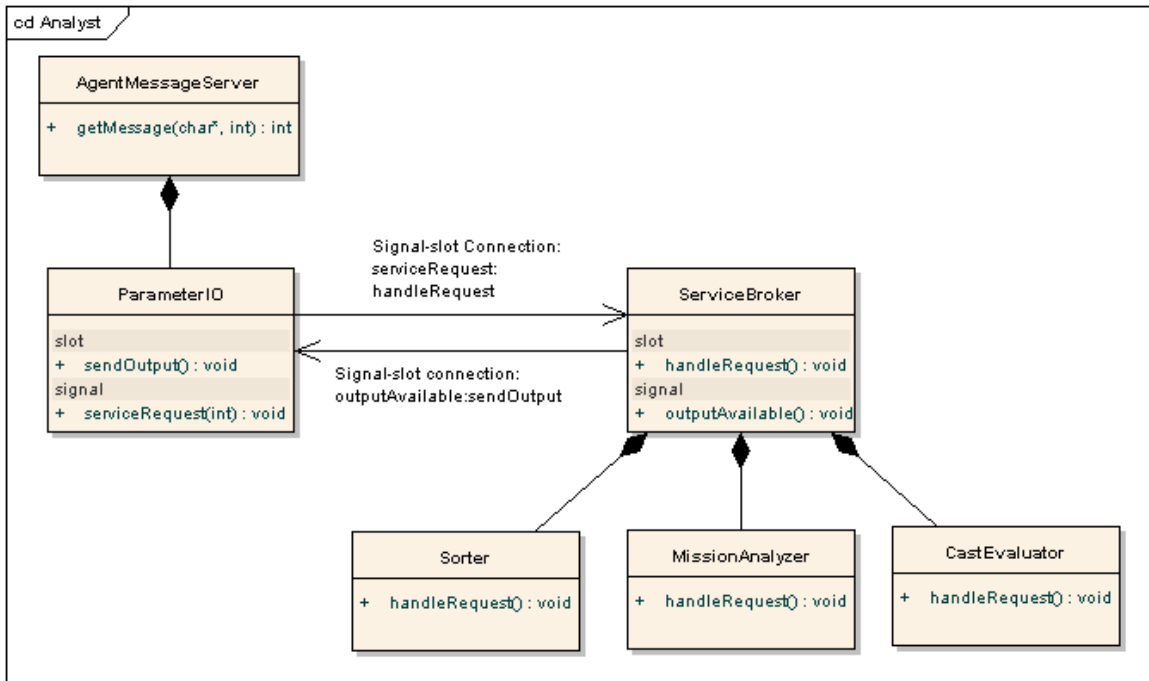


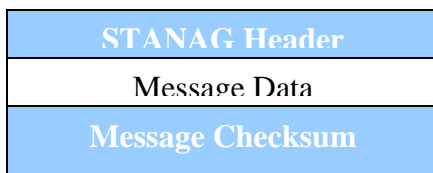
Figure 18: Analyst Class Diagram

PROTOCOL

The STANAG-4586 protocol is the communication protocol that all agents within the U2-MPS system and external agents representing external UAV systems will use. An external UAV system that is able to communicate with the STANAG 4586 protocol is said to be a STANAG-4586 compliant UAV System. In this section, the protocol will be presented in detail.

Messages Wrapper

All STANAG 4586 messages shall be wrapped between a STANAG Header and Message Checksum as illustrated in Figure 19:



The STANAG Header is detailed in Table 2 below:

Field Name	Type	Valid Values	Description
Destination agent ID	Unsigned integer	Min: 0 Max: 4294967295	ID identifying the agent to which the message is addressed
Source agent	Unsigned	Min: 0	ID identifying the agent from which the

ID	integer	Max: 4294967295	message is sent
Message instance ID	Unsigned integer	Min: 0 Max: 4294967295	This number indicates the instance number of a particular message sent from the sender. For example, the first message #1 sent has an instance number of 0, the second instance has a number of 1, and the instance number is incremented for each subsequent instance. Each message would have a set of instance number.
Time stamp	Double	Positive real number	Number of seconds after the epoch (midnight UTC, January 1, 1970).
Message number	Unsigned integer	A valid message number as listed in this protocol document	Indicate the message number of the message data.
Message length	Unsigned integer	Min: 0 Max: 63365	Length of message data

Table 2: STANAG 4586 Message Header

An integer checksum must follow the message data, which is calculated as the arithmetic sum of all byte values in the message from header to the end of message data.

STANAG 4586 messages have unique data and are detailed in the rest of this protocol document. STANAG 4586 messages can be divided into 4 general categories: *Mission Description (100-series)*, *UAV Service Profile (200-series)*, *Environment Information (300-series)* and *Miscellaneous (400-series)*.

Mission Description Messages

The 100-series collection of messages combine to completely specifies a multi-UAV mission. Each message in this section must be completed with the message wrapper specified above.

Message 101 – Mission Message

This message provides a mission ID and the waypoint number of the first waypoint of this mission. Subsequent waypoints are determined via the “Next waypoint number” field of message 102.

Field Name	Type	Valid Values	Description
Mission ID	Char[50]	Null-terminated string within 49 characters long (+ '\0' character)	Identification string of the mission
First Waypoint	Unsigned integer	Min: 0 Max: 4294967295	The waypoint number of the first waypoint of the mission

Message 102 – Waypoint Message

This message describes a particular waypoint. The “Next waypoint number” field is key to linking waypoints to form a complete mission. The “Waypoint action ID list” field is a

variable length list of action IDs indicating the list of actions to be performed at this waypoint.

Field Name	Type	Valid Values	Description
Assigned agent ID	Unsigned integer	Min: 0 Max: 4294967295	The UAV system that is assigned the task of achieving this waypoint and the associated actions
Waypoint number	Unsigned integer	Min: 0 Max: 4294967295	Waypoint number unique within one mission
Waypoint longitude	Double	Min: -PI Max: PI	Radians value of the waypoint longitude
Waypoint latitude	Double	Min: -PI Max: PI	Radians value of the waypoint latitude
Waypoint altitude	Double	Natural double range	Altitude to be achieved at this waypoint
Number of waypoint actions	Unsigned integer	Min: 0 Max: 10	The number of waypoint actions associated with this waypoint
Waypoint action ID list	List of Unsigned integers	Min: 0 Max: 4294967295	A list of integer IDs indicating the waypoint actions to be achieved. The number of integers in this field should be consistent with the value in “Number of waypoint action” field.
Next waypoint number	Unsigned integer	Min: 0 Max: 4294967295	Next waypoint in the sequence

Message 103 – Loiter Action Message

This message describes a loitering action. Multiple waypoints can reference this action via the “Waypoint action ID list” field of message 102.

Field Name	Type	Valid Values	Description
Action ID	Unsigned integer	Min: 0 Max: 4294967295	Unique identifier of this action
Loiter pattern	Enum	0 – Circular 1 – Hover 2 – Figure 8 3 – Race track	Loiter pattern to be executed
Duration	Integer	Min: 0 Max: 4294967295	Time duration for which to loiter at the waypoint in seconds

Message 104 – Camera Action Message

This message describes where the camera should be pointed when the UAV reaches a waypoint. There is an option to turn on/off the strobe light. Multiple waypoints can reference this action via the “Waypoint action ID list” field of message 102.

Field Name	Type	Valid Values	Description
Action ID	Unsigned integer	Min: 0 Max: 4294967295	Unique identifier of this action
Stare-point	Double	Min: -PI	Radians value of the longitude component of

longitude		Max: PI	the coordinate at which the camera will stare
Stare-point latitude	Double	Min: -PI Max: PI	Radians value of the latitude component of the coordinate at which the camera will stare
Strobe light state	Enum	0 – Off 1 – On	Indicates whether to turn on the strobe light shining toward the stare-point
Zoom	Double	Min: 0 Max: PI	Radians value of the field of view (i.e. – how zoomed in is the camera?)

Message 105 – Target Action Message

This message describes target related actions to be performed at a waypoint. Multiple waypoints can reference this action via the “Waypoint action ID list” field of message 102.

Field Name	Type	Valid Values	Description
Action ID	Unsigned integer	Min: 0 Max: 4294967295	Unique identifier of this action
Target profile size	Unsigned integer	Min: 0 Max: 4294967295	The size of the target profile to follow in bytes
Target profile	Buffer	N/A	Target profile used for detection
Duration	Integer	Min: 0 Max: 4294967295	Time duration for which to search for the profiled target in seconds
Search pattern	Enum	0 – single point 1 – spiral outward 2 – switchback	The search pattern used to locate target
Neutralization mode	Enum	0 – report only 1 – fire weapon	Indicates what to do when the target is located. Not applicable if Neutralization mode == 0
Neutralization retries	Unsigned integer	Min: 0 Max: 4294967295 -1 = until weapon runs out	Indicates how many times to retry if the first neutralization attempt failed
Weapon Type	Char[50]	N/a	Type of weapon to employ. Not applicable if Neutralization mode == 0

Message 106 – Mission Transfer Status

This message is used as the handshaking message between agents who are exchanging mission specifications.

Field Name	Type	Valid Values	Description
Addressed mission ID	Char[50]	Null-terminated string	Indicates the mission ID whose specification is requested, transferred or verified.
Request status	Enum	0 – Request Mission to be transferred 1 – Reject mission request 2 – Request authorized 3 – Transfer in progress 4 – Request mission to be evaluated 5 – Mission to be	N/A

		uploaded 6 – Assign mission	
Progress	Unsigned integer	Min: 0 Max: 100	If Request status == 3, indicates the percentage of transfer completion. Not applicable if status != 3.

Message 107 – Mission Summary

Field Name	Type	Valid Values	Description
Number of waypoints	Unsigned integer	Min: 0 Max: 4294967295	The number of waypoints in the mission
Duration	Double	Positive real number	The total mission duration
Cost	Double	Positive real number	The total cost of mission
Potency	Double	Min: 0 Max: 100	Level of confidence that the mission will be completed successfully in percentage

Service Profile Messages

The 200-series messages are used to convey the service profiles of UAV systems. The Mission Editor agent requests service profiles from external UAVs and subsequently store them to the Database agent using the same set of messages.

Message 201 – Flight Parameters Message

This message details a UAV's flying capability. One message 201 is sent per UAV system to indicate its flight parameters.

Field Name	Type	Valid Values	Description
Addressed UAV ID	Unsigned integer	Min: 0 Max: 4294967295	The ID of the UAV agent
Max cruise speed	Double	Positive real number	Meters per second
Max cruise altitude	Double	Positive real number	Meters
Max climb rate	Double	Positive real number	Meters per second
Terrain avoidance/detection	Enum	0 – not available 1 – available	Indicates whether the UAV is furnished with the terrain avoidance capability
Endurance	Double	Positive real number	Duration for which the UAV can stay in the air in seconds
Cost per mile	Double	Positive real number	The cost of flying this UAV per mile
Deployment cost	Double	Positive real number	The overhead deployment cost that is independent of the distance flown by the UAV. (i.e. – take off, landing and risk of loss)

Message 202 – Camera Services Message

This message details the camera services provided by the UAV. One instance of message 202 is sent for each camera onboard the UAV.

Field Name	Type	Valid Values	Description
-------------------	-------------	---------------------	--------------------

Addressed UAV ID	Unsigned integer	Min: 0 Max: 4294967295	The ID of the UAV agent
Camera type	Enum	0 – EO/IR 1 – SAR 2 – Other	A bit vector specifying the list of cameras onboard this UAV. More than 1 bit can be turned on to indicate the availability of multiple cameras
Min Image FOV	Double	Min: 0 Max: PI	Minimum field of view in radians
Max Image FOV	Double	Min: 0 Max: PI	Maximum field of view in radians
Target detection	Enum	0 – Not available 1 – Available	Indicates whether the ability to detect a profiled target is supported
Target tracking	Enum	0 – Not available 1 – Available	Indicates whether the ability to track an identified target is supported

Message 203 – Weapon Services Message

This message details the weapons that a UAV carries. One instance of message 203 is sent for each weapon payload onboard the UAV.

Field Name	Type	Valid Values	Description
Addressed UAV ID	Unsigned integer	Min: 0 Max: 4294967295	The ID of the UAV agent
Weapon Type	Char[50]	Null-terminated string	String identifying the weapon type carried by the UAV
Range	Double	Positive real number	The weapon's maximum range
Collateral damage radius	Double	Positive real number	The radius of damage around the point of contact by the weapon in meters
Guided weapon	Enum	0 – No 1 – Yes	Indicates whether the weapon is guided to achieve high accuracy
Accuracy	Double	Positive read number	The hit-point error in meters at maximum range
Cost	Double	Positive read number	Cost per deployment in dollars

Message 204 –Service Profile Transfer Message

This is the handshaking message used to initiate, stop and validate transfer of service profiles.

Field Name	Type	Valid Values	Description
Addressed UAV ID	Unsigned integer	Min: 0 Max: 4294967295	The ID of the UAV agent whose service profile is being requested/reported/verified
Request status	Enum	0 – Request service profile 1 – Reject service profile request 2 – Request authorized 3 – In progress 4 – Request storage 5 – Reject storage 6 – Storage succeeded 7 – Storage failed	N/A

Progress	Unsigned integer	Min: 0 Max: 100	If Request status == 3, indicates the percentage of transfer completion. Not applicable if status != 3.
----------	------------------	--------------------	---

Intelligence Exchange Messages

The 300-series messages are used by the Database agent to receive and convey intelligence with other agents.

Message 301 – Terrain Information Message

This message is used to convey terrain information between agents.

Field Name	Type	Valid Values	Description
Top left longitude	Double	Min: -PI Max: PI	Longitude of the top left corner of the addressed terrain.
Top left latitude	Double	Min: -PI Max: PI	Latitude of the top left corner of the addressed terrain.
Bottom right longitude	Double	Min: -PI Max: PI	Longitude of the bottom right corner of the addressed terrain.
Bottom right latitude	Double	Min: -PI Max: PI	Latitude of the bottom right corner of the addressed terrain.
Terrain profile size	Unsigned integer	Min: 0 Max: 4294967295	Size of the terrain profile data block
Terrain profile data	Buffer	N/A	Terrain profile data

Message 302 – Weather Condition Message

This message may optionally accompany message 301 to provide weather information.

Field Name	Type	Valid Values	Description
Precipitation	Double	Positive real number	Amount of precipitation in mili-meters in the past 24 hrs.
Temperate	Double	Real number	Current temperate in Kelvin
Air Pressure	Double	Positive real number	Current ground level air-pressure in psi
Weather alert condition level	Enum	0 – No alert 1 – level 1 2 – level 2 3 – level 3 4 – level 4 5 – level 5	Level of adverse weather condition alert
Weather alert type	Enum	0 – Strong wind 1 – Excess precipitation 2 – Sand storm 3 – Hurricane/typhoon	Indicates the type of weather condition giving rise to the alert condition above.

Message 303 – Intelligence Transfer Message

This message is used as the handshaking message to initiate, stop and validate transfer of environment information between agents.

Field Name	Type	Valid Values	Description
Intelligence type	Enum	0 – Environment 1 – Mission execution report 2 – Performance history	Indicates the type of intelligence being transferred/requested.
Request status	Enum	0 – Request intelligence 1 – Reject intelligence request 2 – Request authorized 3 – In progress 4 – Request verification 5 – Reject verification 6 – Verification succeeded 7 – Verification failed	N/A
Progress	Unsigned integer	Min: 0 Max: 100	If Request status == 3, indicates the percentage of transfer completion. Not applicable if status != 3.

Message 304 – Mission Execution Progress Report Message

This message is used by UAV system agents to report the progress of mission execution.

Field Name	Type	Valid Values	Description
Progress	Double	Min: 0 Max: 100	Percentage of completion
Previous waypoint number	Unsigned integer	Min: 0 Max: 4294967295	The waypoint number that the UAV has previously executed
Current waypoint number	Unsigned integer	Min: 0 Max: 4294967295	The waypoint number that the UAV is currently executing
Next waypoint number	Unsigned integer	Min: 0 Max: 4294967295	The waypoint number that the UAV will execute next
Previous action ID	Unsigned integer	Min: 0 Max: 4294967295	The waypoint action that has just been executed
Current action ID	Unsigned integer	Min: 0 Max: 4294967295	The waypoint action that the UAV is currently executing
Next action ID	Unsigned integer	Min: 0 Max: 4294967295	The waypoint action that the UAV will execute next
Mission status	Enum	0 – In progress 1 – Aborted 2 – Failed 3 – Succeeded	The overall status of the mission
Error report	Buffer	N/A	In case of mission failure/abortion, this field provides a detail report the of the cause of failure/abortion.

Mission 305 – UAV Performance History

The Database agent keeps track of mission execution status as reported in message 304, and the result of each mission execution is stored. Upon requests from other agents (with

message 303), the Database agent will use this message to report a UAV's performance history.

Field Name	Type	Valid Values	Description
Mission success rate	Double	Min: 0 Max: 100	Percentage of success
Mission abortion rate	Double	Min: 0 Max: 100	Percentage of intentional mission abortion
Mission failure rate	Double	Min: 0 Max: 100	Percentage of failure
Environment induced failure rate	Double	Min: 0 Max: 100	Percentage of failures caused by environment conditions
System malfunction failure rate	Double	Min: 0 Max: 100	Percentage of failures caused by system malfunction

Miscellaneous Messages

Message 401 – UAV Discovery Message

This message is used by the U2-MPS system to discover UAV systems on the network. This message is sent to the network's broadcast address (all hosts will receive this message). It is agreed upon that all UAV systems on the network that intends to offer service must listen to this message 401 on port 77. Upon receiving this message, the UAV agent must send a message 401 in response.

Field Name	Type	Valid Values	Description
Host	Char[50]	Null-terminated string	The host name of the sender
Port number	Unsigned integer	Min: 0 Max: 4294967295	The port number on which the sender is listening for incoming messages
Agent type	Enum	0 – UAV agent 1 – Mission planner	Identifies the sender agent's type

Message 402 – Generic parameter list message

This generic message is used by agents to pass a list of parameters. This message is used heavily between agents within the U2-MPS system as they pass information amongst themselves for computations and analysis.

Field Name	Type	Valid Values	Description
Parameter type	Enum	0 – UAV IDs 1 – Cost evaluation 2 – Potency evaluation 3 – Mission ID	The host name of the sender
Number of parameters	Unsigned integer	Min: 1 Max: 4294967295	The size of parameter array
Parameters	List of parameters,	N/A	List of parameters

	all of which are of type specified in "Parameter type" field.		
--	---	--	--

Message 403 – Generic parameter request message

This generic message is used by agents within the U2-MPS system to request parameters be sent to them.

Field Name	Type	Valid Values	Description
Parameter type	Enum	0 – UAV IDs 1 – Cost evaluation 2 – Potency evaluation 3 – Mission ID 4 – Waypoint ID	The host name of the sender

Use Cases

Three use cases are provided below which combines to illustrate how the five U2-MPS agents and external UAV System agents collaborates to help the user cast a mission specified using the Mission Editor agent. The first use case illustrates how the U2-MPS system is able to discover and store service profiles of external UAV systems. The second use case shows the sequence of events that takes place when the user requests the system to compute an optimized cast for the mission. The third use case presents the agent interactions that take place when the user selects a cast and assigns the mission to the UAV systems involved.

Use case #1: Gather and store UAV service profiles

Precondition: Mission Editor agent periodically sends message 401 to discover UAV systems on the network

1. New UAV System agent comes online
2. UAV System agent receives message 401
3. UAV System agent decides that it wants to provide service and reply with message 401
4. Mission Editor agent receives message 401
5. Mission Editor agent request service profile with message 204
6. UAV System reports its service profile with message 201 and optionally message 202 and 203 depending on the UAV system’s capability. While the UAV system reports its service profile, it also send message 204 to keep Mission Editor agent abreast of the current progress of profile reporting.
7. UAV System finishes reporting its service profile, and sends message 204 indicating so.
8. Mission Editor receives message 204 indicating service profile is complete

9. Mission Editor sends message 204 to the Database agent requesting permission to store the newly received UAV service.
10. Database agent replies with message 204 permitting storage of service profile.
11. Mission Editor sends the service profile to the Database agent via message 201, 202 and 203.
12. Mission Editor finishes sending profile and send message 204 indicating so.
13. Database agent stores the service profile information

Post-condition: UAV service profile is discovered and stored by the Database agent.

Use case #2: Request cast recommendations

Pre-condition: The Mission Commander human actor has specified a mission with the user interface provided by the Mission Editor agent. This mission has been saved.

1. Mission Commander presses the “Cast” button on Mission Editor
2. Mission Editor sends message 403 to the Optimizer agent requesting for a list of UAV Systems to fill the roles of the current mission
3. The Optimizer agent receives message 403 and decides that it is available to help cast a mission
4. The Optimizer agent sends message 403 to the Mission Editor agent requesting for the mission ID for which a cast is to be recommended
5. The Mission Editor sends message 402 providing the mission ID.
6. The Optimizer receives the mission ID from Mission Editor and sends a message 106 to the Database agent requesting the detail mission specification.
7. The Database agent sends mission specifications to Optimizer via message 101 – 105. When it finishes, it sends message 106 indicating transfer is complete.
8. Optimizer receives message 106, and stores the mission in local memory.
9. For each waypoint in the mission, the Optimizer sends message 403 to the Mission Manager agent requesting for a list of UAV IDs capable of meeting the associated waypoint action requirements.
10. For each message 403 that the Mission Manager agent receives, it looks through its repository of service profiles and reports the list of UAV systems capable of performing the action via message 402 back to Optimizer.
11. Optimizer now has a number of candidates for each waypoint. Optimizer now figures out all the possible combinations of candidates, and sends each combination to the Analyst agent via message 101 – 106.
12. Analyst reports the results of evaluation (cost and potency) back to Optimizer via message 402.
13. Optimizer receives the cost/potency estimates from Analyst, and further requests intelligence from the Database via message 303.
14. The Database agent returns intelligence via message 301, 302, and 305.
15. The Optimizer scales the cost/potency estimates taking into account terrain information, weather condition and UAV performance history.

16. Optimizer stores the combination that returns the lowest overall cost and the combination that returns the highest overall potency.
17. Optimizer returns the two combinations with the lowest cost and highest potency to Mission Editor via message 402.

Post-condition: Mission Editor has two cast recommendations to present to the Mission Commander, one for optimized cost and one for optimized potency.

Use case #3: Upload Mission

Pre-condition: Use case #1 and #2 have been executed

1. Mission Commander selects a recommendation presented to him/her by the Mission Editor agent
2. Mission Commander presses the “Upload” button
3. Mission Editor stores the current mission (with UAVs assigned to each waypoint) to the Database by sending a sequence of message 101 to 105 describing the mission.
4. Mission Editor agent sends message 106 to the Mission Manager to request the mission identified by the mission ID field to be uploaded.
5. Mission Manager queries the mission specification to be uploaded from the Database agent with message 106.
6. Database sends the mission specifications to Mission Manager via message 101 to 105.
7. Mission Manager assigns the mission to UAVs stated in the mission specification via message 106.
8. UAV System agents receives message 106 with the mission ID to which it is assigned.
9. UAV System agent queries the Mission Editor agent (known host and port through initial handshaking done in use case #1) for the mission specification by sending message 106.
10. Mission Editor displays progress of mission transfer to the UAV System agents, and indicates transfer complete upon completion.

Post-condition: Mission assigned to UAV systems

REFERENCE

[1] T. Do, M. Kolp, S. Faulkner, “Structure-in-5 as an Agent Architectural Pattern”, <http://www.cs.toronto.edu/~mkolp/aoissin5.pdf>

[2] F. Giunchiglia, J. Mylopoulos, and A. Perini, “The Tropos Software Development Methodology: Processes, Models and Diagrams”, Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems, 2002.

[3] B. Henderson-Sellers, P. Giorgini, “Agent-Oriented Methodologies”, Idea Group Publishing, 2005, pp. 20 – 42.