Asteroid Exploration System Design Document

Date: Dec 12th, 2017
Author: Stephen Akaeze
Reviewers: Joseph Kintzel and David Hu

Introduction

This document defines the design of an Asteroid Exploration System (AES) used in exploring, discovering and, potentially, mining valuable minerals from respective asteroids. For easier comprehension, this document includes component, activity, class and sequence diagrams. It also comprises a class dictionary and user interface descriptions for interaction with the AES. The document will also discuss use cases, exception handling, risks and instability analysis.

Overview

The simplified challenge of the AES is to securely create and manage asteroid exploration missions. Several asteroids contain valuable minerals that are of great business value. The AES aims create, manage and simulate exploration missions, spacecraft and asteroids. It also provides quality communication between the earlier listed elements and a user interface. This way new resources can be tracked and stored.

The AES aims to solve this problem using a Service Oriented Architecture (SOA). An SOA implementation ensures that each subsystem maintains a service interface that is callable by other subsystems.

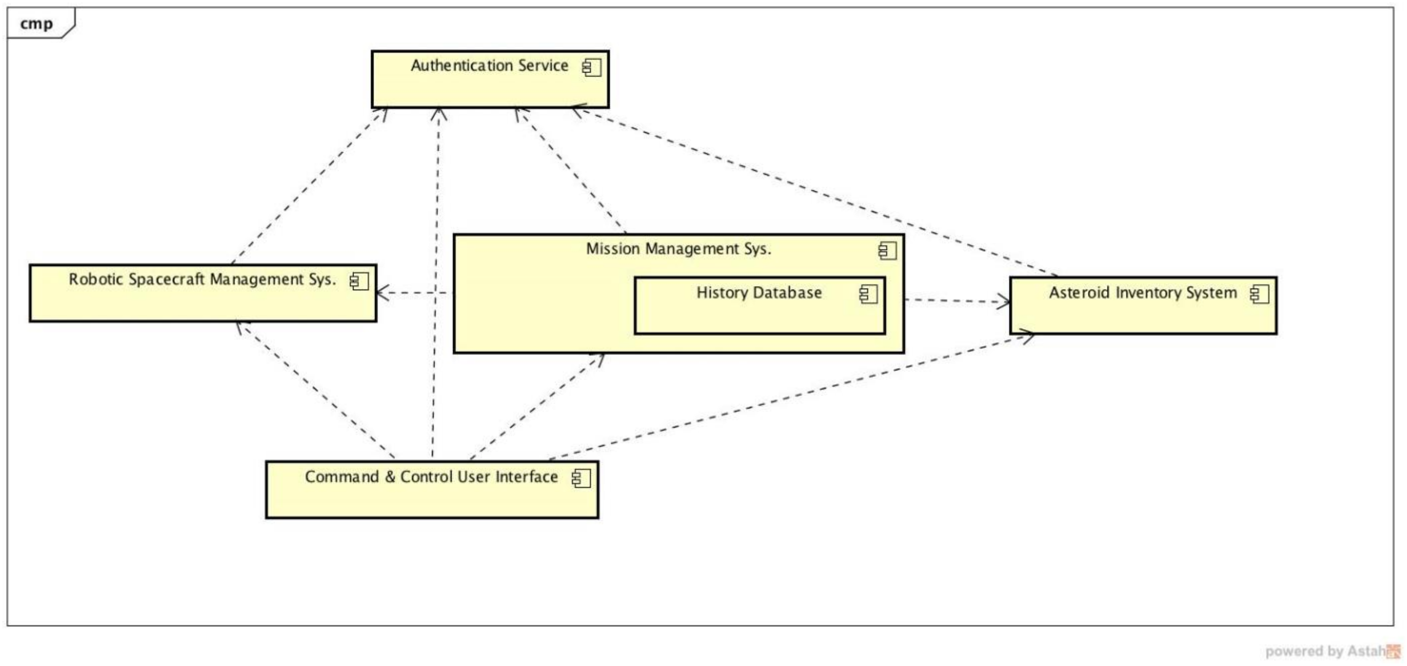
cmp
Authentication Service
Mission Management Sys.
History Database
Robotic Spacecraft Management Sys.
Asteroid Inventory System
Command & Control User Interface
powered by Astah

Fig 1. The proposed Asteroid Exploration System module component diagram

As shown in the component diagram above, AES comprises five submodules with individual service interfaces as follows

1. Robotic Spacecraft Management System (RSMS) is responsible for managing the fleet of robotic spacecraft.
2. Mission Management System (MMS) is responsible for creating and launching missions
3. Asteroid Inventory System (AIS) is responsible for maintaining an inventory of all known asteroids and their features
4. The Command and Control User Interface (CCUI) is responsible for providing a human friendly graphical user interface for managing the AES
5. Authentication Service is responsible for securing the CCUI as well as ensuring service interface integrity between all subsystems

Requirements

Below are the requirements for each AES subsystem

Robotic Spacecraft Management System

The system maintains the current status of each of the spacecraft.

Each spacecraft has the following set of attributes:
Identifier: unique spacecraft call sign
Launch Date: date of launch
Mission: mission identifier
Type (Explorer, Miner)
Status
Fuel (% remaining)
Systems
Guidance (OK, Not OK)
Communication Link (OK, Not OK)
State (waiting for launch, in route, lost, crashed, landed, exploring, mining, homeward bound, malfunction)
Location (AUs from Sun)
Destination (target Asteroid identifier)

As resources permit, new spacecraft can be added. Existing spacecraft can be listed, and also looked up by id. Updates to spacecraft can be saved. All spacecraft management methods have restricted access.

Mission Management System

The following mission attributes are required:
unique id for mission
name of mission (e.g. “sling shot”),

purpose of mission (e.g. search for water)
spacecraft id of fully provisioned spacecraft that will perform the mission
launch date (start time)
eta, estimated time of arrival
destination: the destination asteroid for the Mission
status: (waiting for launch, in progress, complete, aborted)

As missions are defined, the available resources should be updated. Resources of fuel, spacecraft, and operating budget should be reduced in accordance with the cost of each new mission.

Mission Management must maintain status of the land-based communication links, and also the automated control system. Each system must have a current status and be able to alert the Mission Control Center of any failures.

Mission Management is also responsible for receiving messages from the spacecraft. As messages are received, the message content should be parsed and understood. The status of the robots should be updated appropriately. Also, if a discovery is reported by a spacecraft, the associated asteroid should be updated. For example, if water is discovered on Ceres, then the Ceres asteroid should be updated to reflect that it contains water.

Messages must be defined in sufficient detail to identify the spacecraft sending the message, the type of message, with required details to update the spacecraft status, and possibly the target Asteroid in case of a discovery. Mission Status should be updated appropriately though the life cycle of the mission. For example, if a spacecraft accidently crashes into rather than lands on the target asteroid, the spacecraft status should be set to "crashed" and the mission status should be set to "aborted". All mission management method interfaces have restricted access.

Asteroid Inventory System

The Asteroid Inventory System manages the collection of known Asteroids and their associated features as follows:

Identifier: the given unique identifier for the asteroid (e.g. 1 Ceres, 4 Vesta, 433 Eros , etc.),
Notes: description and other notes about the asteroid. Notes can be added over time. Notes have a date, author and text.
Exploration Status:
Minerals found, estimated mass, accessibility (e.g. platinum, 20 metric tons, surface deposit).
Water found, quantity, state (e.g. yes, 20 million liters, ice)
Life (none, single cell organisms, multi cell, intelligent, friendly)
Asteroid Type (C-Type, M-Type, S-Type, Inner-belt comet)
Size: width, length, height
Mass: approximate mass
Surface Gravity: gravitational field at surface
Aphelion: furthest distance from Sun in AUs

Perihelion: closest distance from Sun in AUs

The Asteroid Inventory System must support the following interface functions:
List asteroids
Lookup Asteroid by Id
Create new and Update Existing Asteroids
Query for Asteroids: Text Search: Search for matching text in notes.
All Asteroid Inventory System methods have restricted access.

Command and control User interface

The CCUI should support the following functionalities

- logging in and out
- define missions
- monitoring and updating mission status
- monitoring and updating spacecraft status
- monitoring and updating asteroid status
- monitoring Mission Control resources (fuel, number of available spacecraft, operating budget)
- monitoring status of the ground based communication links and the automated control system.
- Monitoring incoming messages from the spacecraft

The Command and Control User Interfaces should leverage the service interfaces of the

- Asteroid Inventory System
- Robotic Spacecraft Management System
- Mission Management System
- Authentication Service

Authentication Service

The Authentication Service (AS) is the same used in assignment 4. Below are its primary functionalities used in the AES

- Logging in and out
- Validating a token to identify any AES subsystem.

Note: Please proceed to next page for more information

Use Cases

Command and Control User Interface

As shown below, a CCUI admin can successfully login to create and monitor several elements of the AES. All the listed functionalities are achieved via the service interfaces of the submodules. Please refer to AEs GUI section for more information

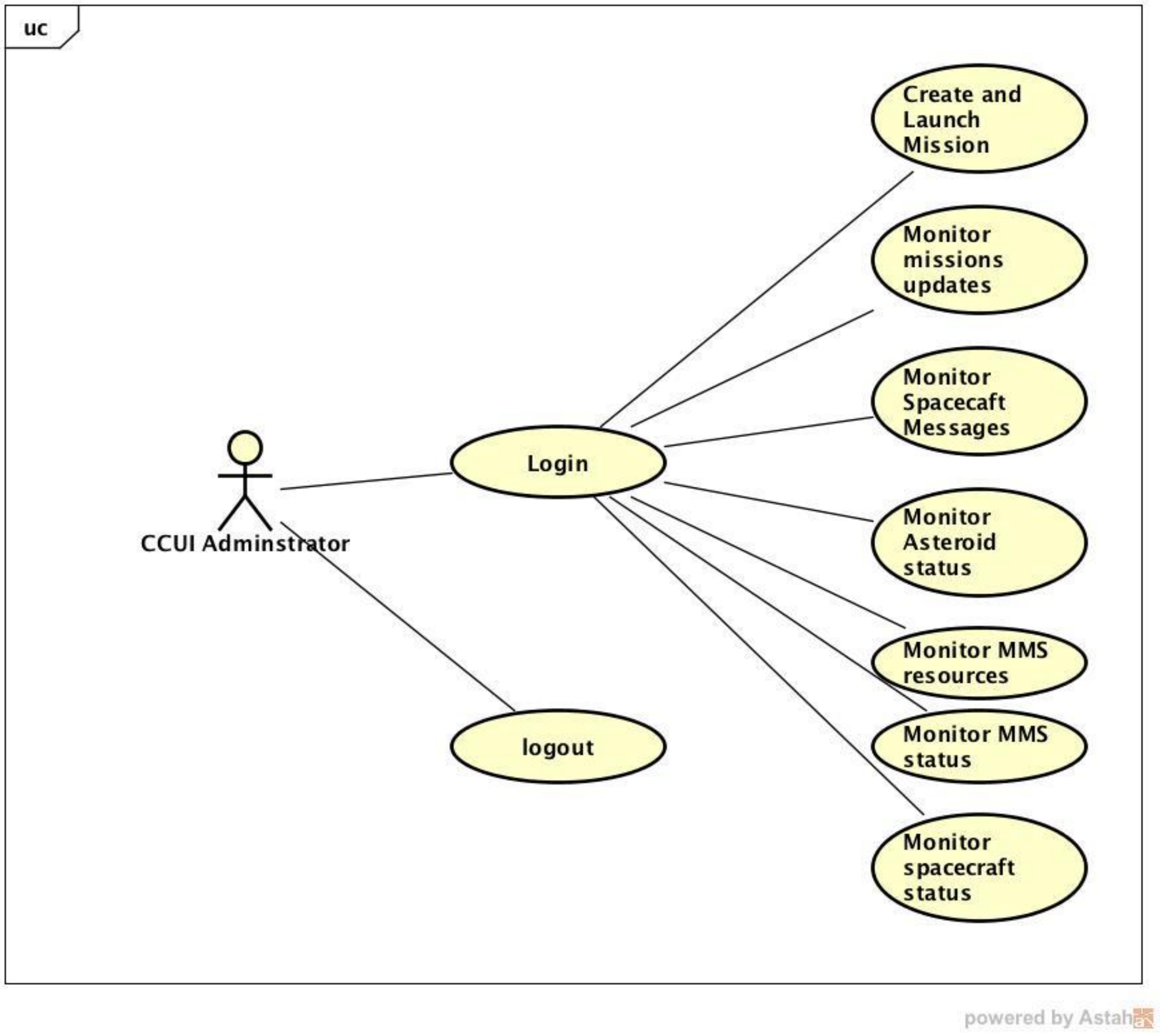

Fig 2. CCIU use cases

Mission Management Service

As shown below, the mission management is capable of performing the several functions essential to AES. These functionalities primarily cater to managing all missions

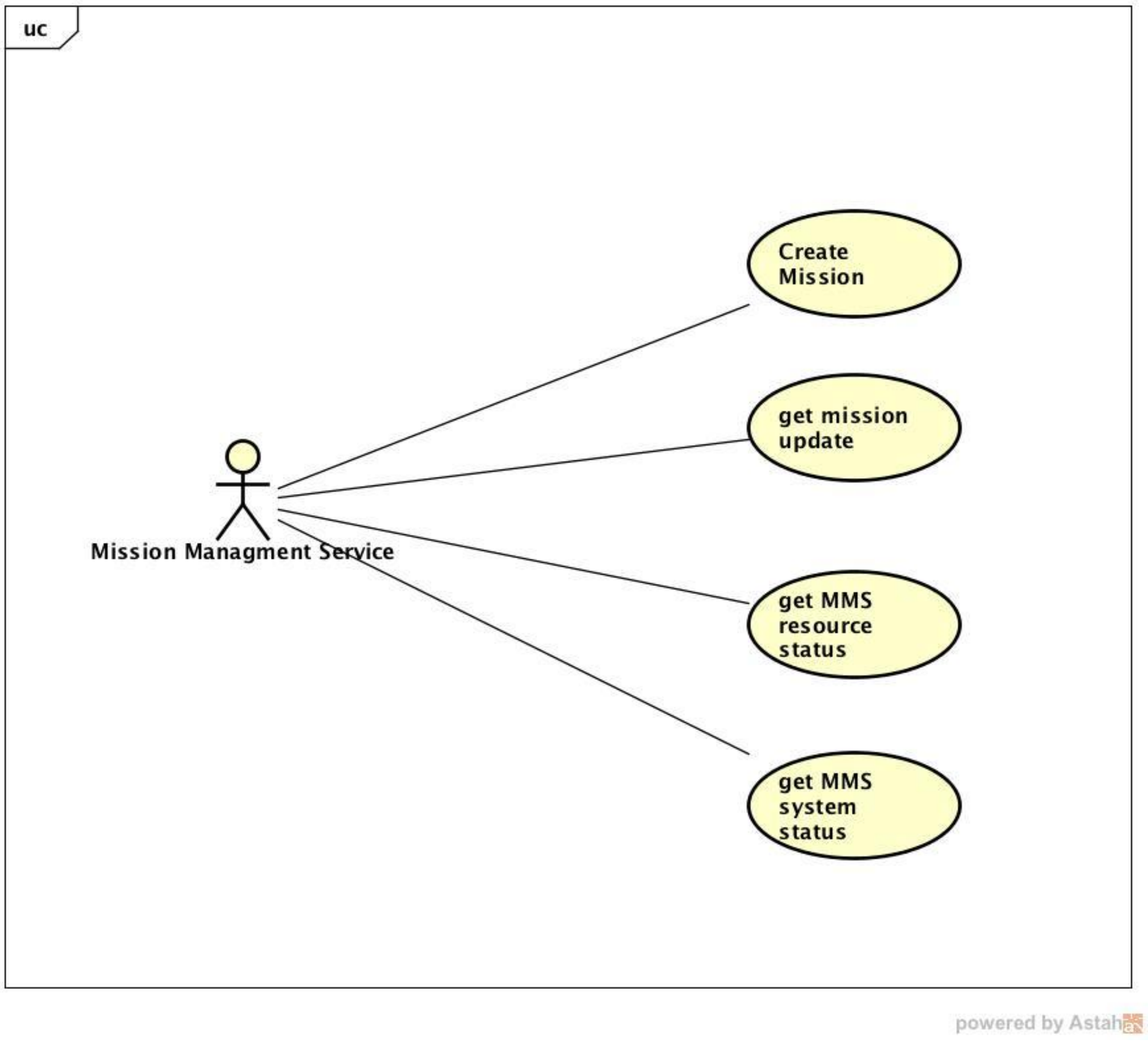

Fig 3. MMS use cases

Robotic Spacecraft Management Service

As shown below, the RSMS possess several useful functionalities for managing the spacecraft fleet.

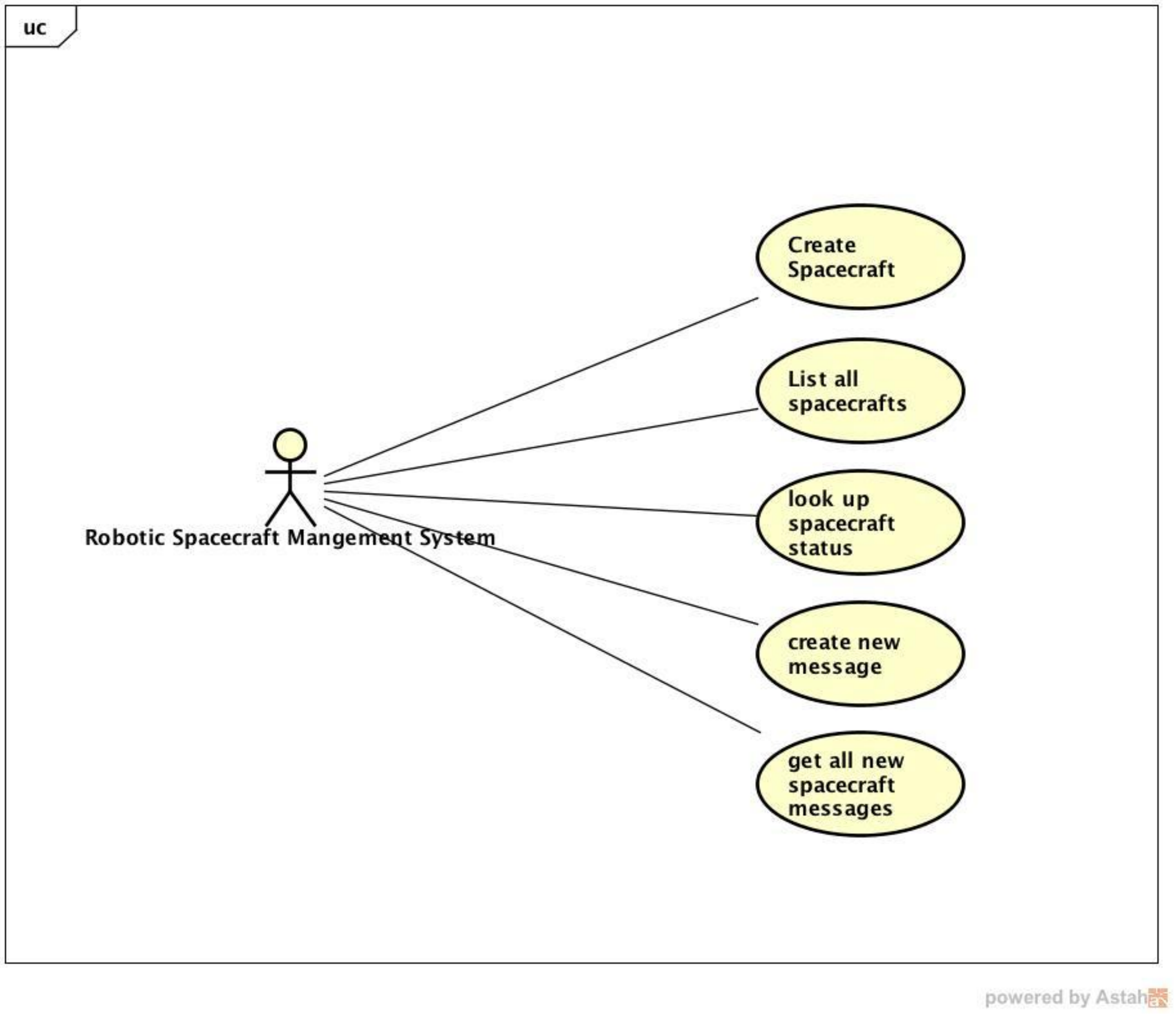
uc
Robotic Spacecraft Mangement System
Create Spacecraft
List all spacecrafts
look up spacecraft status
create new message
get all new spacecraft messages
powered by Astah

Fig 4. RSMS uses cases

Asteroid Inventory System

As shown below, the AIS is very helpful in creating, storing and updating all know asteroid information

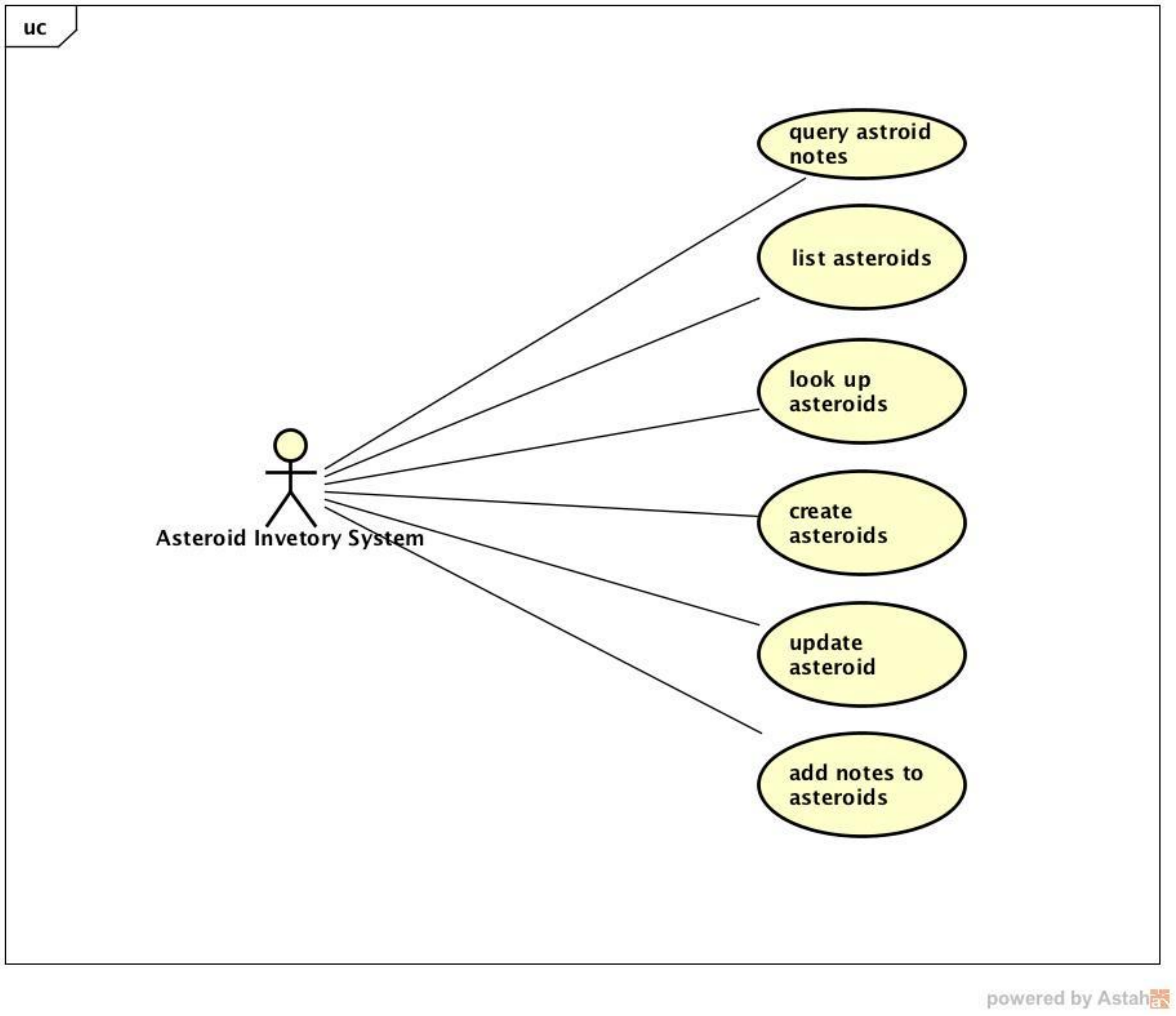

Fig 5. AIS use cases

Implementation

Mission Management Service Class Implementation

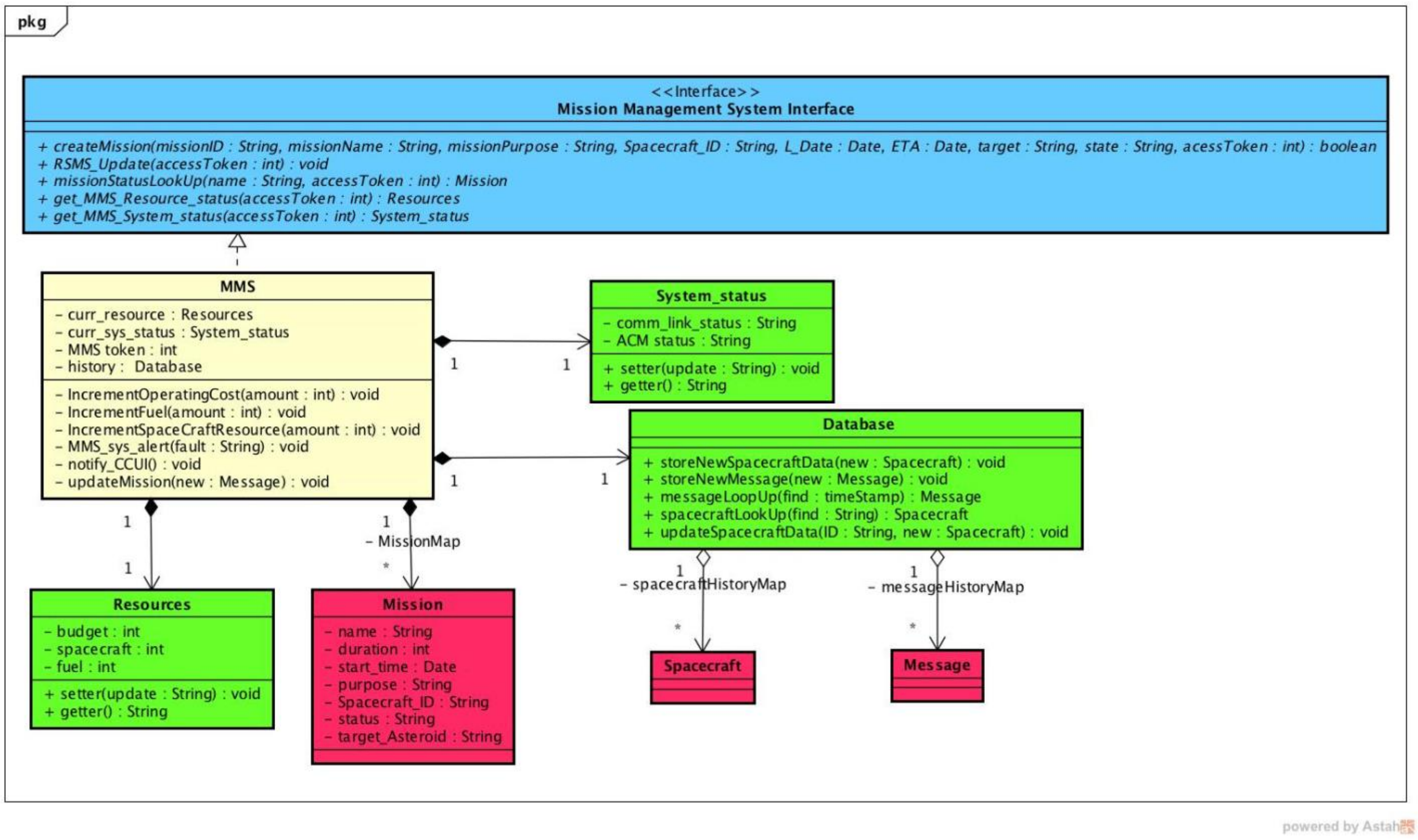
pkg
<<Interface>>
Mission Management System Interface
+ createMission(missionID : String, missionName : String, missionPurpose : String, Spacecraft_ID : String, L_Date : Date, ETA : Date, target : String, state : String, acessToken : int) : boolean
+ RSMS_Update(accessToken : int) : void
+ missionStatusLookUp(name : String, accessToken : int) : Mission
+ get_MMS_Resource_status(accessToken : int) : Resources
+ get_MMS_System_status(accessToken : int) : System_status
MMS
- curr_resource : Resources
- curr_sys_status : System_status
- MMS token : int
- history : Database
- IncrementOperatingCost(amount : int) : void
- IncrementFuel(amount : int) : void
- IncrementSpaceCraftResource(amount : int) : void
- MMS_sys_alert(fault : String) : void
- notify_CCUI() : void
- updateMission(new : Message) : void
System_status
- comm_link_status : String
- ACM status : String
+ setter(update : String) : void
+ getter() : String
Database
+ storeNewSpacecraftData(new : Spacecraft) : void
+ storeNewMessage(new : Message) : void
+ messageLoopUp(find : timeStamp) : Message
+ spacecraftLookUp(find : String) : Spacecraft
+ updateSpacecraftData(ID : String, new : Spacecraft) : void
Resources
- budget : int
- spacecraft : int
- fuel : int
+ setter(update : String) : void
+ getter() : String
Mission
- name : String
- duration : int
- start_time : Date
- purpose : String
- Spacecraft_ID : String
- status : String
- target_Asteroid : String
Spacecraft
Message
1
1
1
1
1
1
1
- MissionMap
*
1
- spacecraftHistoryMap
*
1
- messageHistoryMap
*
powered by Astah

Fig 6. Mission Management Service class diagram

Mission Management System Interface

This is the service interface of the MMS. It is an interface that other subsystems rely on to utilize the MMS functionalities. Each method requires an access token which will be validated by the Authentication service. All methods in the Mission Management Interface are equally implemented in the MMS class.

Methods

Method Name	Signature	Description
createMission	(missionID : String, missionName : String, missionPurpose : String, Spacecraft_ID : String, L_Date : Date, ETA : Date, target : String, state : String, acessToken : int) : void	This is a public method implemented in the MMS which is used to create and launch new missions

RSMS_Update	(accessToken : int) : void	This is a public method implemented in the MMS that enables the MMS to observe new message updates from the RSMS
missionStatusLookUp	(name : String, accessToken : int) : Mission	This is a public method implemented in the RSMS which returns the status of any specified mission as an a Mission instance
get_MMS_Resource_status	(accessToken : int) : Resources	This is a public method implemented in the MMS which returns the current status of the MMS resources as a Resource instance.
get_MMS_System_status	(accessToken : int) : MMS system status	This is a public method implemented in the MMS which returns the status of the MMS

MMS (Mission Management Service)

This is the Mission Management Service which is dependent on the Mission Management Service Interface to execute service requests from other subsystems. It is a singleton class that automatically implements all the Mission Management Service Interface Methods in addition to the following methods.

Methods

Method name	**Signature**	**Description**
IncrementOperatingCost	(amount:int) : void	This is a private method is used to update the operating budget cost resource whenever a new mission is created.
IncrementFuel	(amount : int) : void	This is a private method is used to update the operating budget cost resource whenever a new mission is created.
IncrementSpaceCraftResource	(amount : int) : void	This is a private method is used to update the spacecraft resource whenever a new mission is created.

MCM_sys_alert	(fault : String) : void	This is the private method called whenever the automated control system or communication links are down
notify_CCUI	() : void	This is the private method that updates the CCUI observer after new message(s) has been processed
updateMission	(new : Message) : void	This is the private method that updates any mission as specified by the received spacecraft message.

Properties

Property Name	**Type**	**Description**
curr_resource	Resources	This is the MMS Resources instance that stores that values of the MMS resources at any given time.
curr_sys_status	MMS system status	This is the MMS system_status instance that stores that state of the communication link and Automated control system at any given time.
MCM token	int	This stores the MMS attribute that stores value of the MMS token initially assigned by the authentication service.
history	Database	This stores all received messages and retains spaceraft updates. This ensures total system persistency in addition to the Asteroid Inventory Service.

Associations

Association Name	**Type**	**Description**
MissionMap	HashMap<String,Mission>	This is the MMS association which stores all known missions according to their respective names

Mission
This is the class which is designed to track the data and status of every created mission.

Properties

Property Name	Type	Description
name	String	This is the property that stores the name of the mission
duration	int	This is the property that stores the expected duration of the mission
start_time	date	This is the property that stores the start time of the mission
purpose	String	This is the property that stores the purpose of the mission
Spacecraft_ID	String	This is the property that stores the ID of the spacecraft assigned to the mission
status	String	This is the property that stores the current status of the mission
target_Asteroid	String	This is the property that stores the target ID of the asteroid for the mission

Resources
This is the class that is designed to track the status of the MMS resources. These resources are Fuel percentage, operating budget and spacecraft remaining

Methods

Method Name	Signature	Description
setter	(update : String) : void	This is the method used to set the new resource values using the update argument. For example update = “fuel”+”spacecraft”+”budget”
getter	() : String	This is the method used to retrieve the values of the resources at any time.

Properties

Property Name	Type	Description
budget	int	This stores the value of the operating budget
fuel	int	This stores the value of the fuel percentage at anytime
spacecaft	int	This stores the number of remaining spacecraft in the MMS.

System_status

This is the class designed to store the MMS system status at any time. These system elements include communication links and Automated Control System status(ACM).

Methods

Method Name	Signature	Description
setter	(update : String) : void	This is the method that sets the value of comm link and Automated Control Status. For example, update = “comm_link” +”ACM”.
getter	():String	This is the method that returns the values of the comm link and ACM

Properties

Property Name	Type	Description
comm_link_status	String	This is the attribute that stores the status of the communication link
ACM_status	String	This is the attribute that stores the status of the Automated Control System.

Database

This is the class that maintains the persistence of the entire system. While the Asteroid Inventory System stores all asteroid data, the database stores all received messages and spacecraft status. This enables persistence throughout the system.

Methods

Method Name	Signature	Description
storeNewSpacecraftData	(new : Spacecraft) : void	This is method that stores all new spacecraft updates extracted from new messages
storeNewMessage	(new : Message) : void	This is the method that stores all messages.
messageLoopUp	(find : timeStamp) : Message	This is the method that returns any message based on time stamp
spacecraftLookUp	(find : String) : Spacecraft	This is the method that returns a requested spacecraft data

updateSpacecraftData	(ID : String, new : Spacecraft) : void	This method returns the data of a requested spacecraft

Associations

Association Name	**Type**	**Description**
spacecraftHistoryMap	HashMap<String,Spacecraft>	This is the association that stores all know spacecraft and their updates. Check the Spacecraft class under the Robotic Spacecraft Management System.
messageHistoryMap	HashMap<int,Message>	This is the association that stores all received messages. Check the Message class under the Robotic Spacecraft Management System.

Robotic Spacecraft Management System Class Implementation

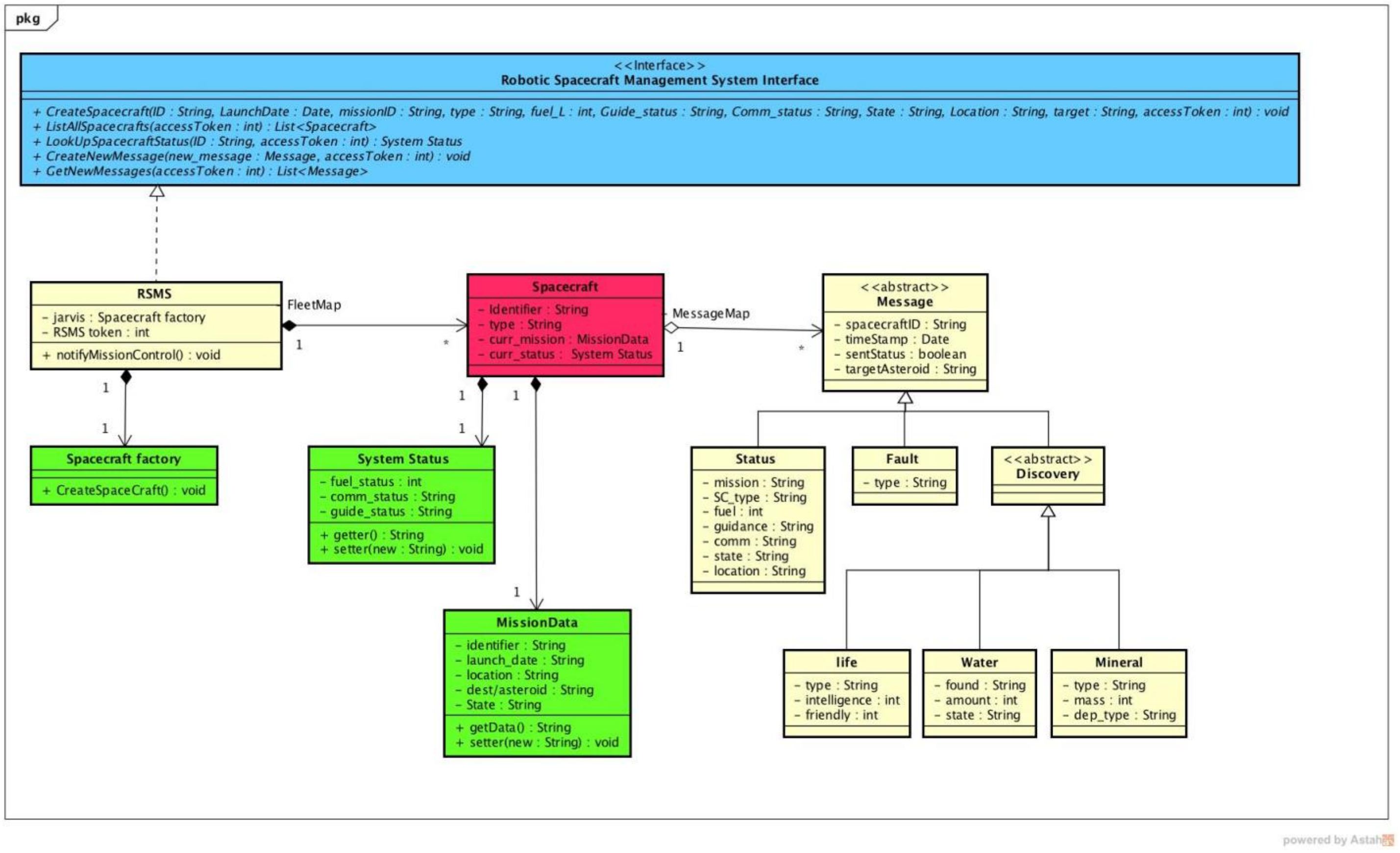

Fig. 7 Robotic Spacecraft Management System class diagram

Robotic Spacecraft Management System Interface

This is the service interface of the RSMS. Each method requires an access token which will be validated by the Authentication service. All methods in the Mission Management Interface are equally implemented in the RSMS class.

Methods

Method Name	Signature	Description
CreateSpacecraft	(ID : String, LaunchDate : Date, missionID : String, type : String, fuel_L : int, Guide_status : String, Comm_status : String, State : String, Location : String, target : String, accessToken : int) : void	This is the public method that creates a new spacecraft.

ListAllSpacecrafts	(accessToken : int) : List<Spacecraft>	This is the public method that lists all current spacecrafts
LookUpSpacecraftStatus	(ID : String, accessToken : int) : System Status	This is the method that enables a spacecraft status look up
CreateNewMessage	(new_message : Message, accessToken : int) : void	This is the method that allows a message to be created by a simulator
GetNewMessages	(accessToken : int) : List<Message>	This is the method that allows other units to query the RSMS for new messages

RSMS (Robotic Spacecraft Management Service)

This is the Robotic Spacecraft Management Service which is dependent on the Robotic Spacecraft Management Interface to execute service requests from other subsystems. It is a singleton class that automatically implements all the Robotic Spacecraft Management Interface Methods in addition to the following methods:

Methods

Method name	**Signature**	**Description**
notifyMissionControl	() : void	This is the method that notifies the Mission Control Mangement observer

Properties

Property Name	**Type**	**Description**
jarvis	Spacecraft factory	This is the object instance that creates new spacecrafts
RSMS token	int	This stores the value of the RSMS token

Associations

Association Name	**Type**	**Description**
FleetMap	HashMap<String, Spacecraft>	This is an association that stores spacecraft according to their IDs

Spacecraft factory
This is the factory class that creates new spacecraft

Method

Method Name	Signature	Description
CreateSpacecraft	():void	This is the method that creates a new spacecraft instance and stores it in the FleetMap

Spacecraft
This is the class that represents a spacecraft.

Properties

Property name	Type	Description
Identifier	String	This attribute stores the spacecraft identifier.
type	String	This attribute stores the spacecraft type
curr_mission	MissionData	This stores the data of the current mission of each spacecraft.
curr_status	System status	This stores the system status such as fuel, communication link status and guide status.

Associations

Association name	Type	Description
MessageMap	HashMap<String, Message>	This stores all the messages from all the spacecraft.

System Status
This is class that stores and tracks the status of every spacecraft such as the fuel status, guidance and communication link status such as the "curr_status" instance in the Spacecraft class

Methods

Method Names	Signature	Description
getter	() : String	This is the method that is used returning the value of the system status elements
setter	(new : String) : void	This is method that is used for setting the system status values

Properties

Property Name	Type	description
fuel_status	int	This stores and tracks the fuel percentage status
comm_status	String	This stores and tracks the communication status
guide_status	String	This stores and tracks the guidance status

MissionData

This is the class that stores and updates the current mission data of each spacecraft such as the “curr_mission” instance in the Spacecraft class.

Methods

Method Name	Signature	Description
getData	() : String	This is the method that returns the mission data values as a concateneated String
setter	(new : String) : void	This is the method that updates the mission data elements

Properties

Property Name	Type	Description
identifier	String	This stores the mission ID
launch_date	String	This stores the mission launch date
location	String	This stores the current mission location
dest/asteroid	String	This stores the target asteroid ID
state	String	This stores the current state of the mission

Message

This is the class that describes the messaging objects of the AES.

Properties

Property Name	Type	Description
spacecaftID	String	This stores the ID of the spacecraft that creates the message
timeStamp	date	This stores the time when the message was created
sentStatus	boolean	This stores the sent status of the message from
targetAsteroid	String	This stores the target asteroid in the respective messages

Status
This is the message subclass which describes messages about the status of each respective spacecraft.

Properties

Property Name	Type	Description
mission	String	This stores the status of mission ID
SC_type	String	This stores the spacecraft type
fuel	int	This stores the spacecraft fuel percentage level
guidance	String	This stores the spacecraft guidance system status
comm	String	This stores the spacecraft comm status
state	String	This stores the spacecraft state
location	String	This stores the current location of the spacecraft

Fault
This is the message subclass which describes fault messages from respective spacecraft.

Properties

Property Name	Type	Description
type	String	This describes the type of fault message

Discovery
This is the message subclass which describes the discovery messages from respective spacecraft

Life
This is the discovery message subclass which describes information about life on an asteroid by a respective spacecraft.

Properties

Property Name	Type	Description
type	String	This stores the type of life discovered
intelligence	int	This describes the intelligence level of the discovered life
friendly	int	This describes the friendliness of the discovered life

Water

This is the discovery message subclass which describes information about water discovered on an asteroid by a respective spacecraft.

Properties

Property Name	Type	Description
found	String	This shows if water was discovered
amount	int	This stores the amount of water discovered
state	String	This describes the state of the discovered water

Mineral

This is the discovery message subclass which describes the mineral discovered on an asteroid by a respective spacecraft.

Properties

Property Name	Type	Description
type	String	This indicates the type of mineral
mass	int	This stores the amount of mineral discovered
dep_type	String	This describes the type of mineral deposit

Asteroid Inventory System Class Implementation

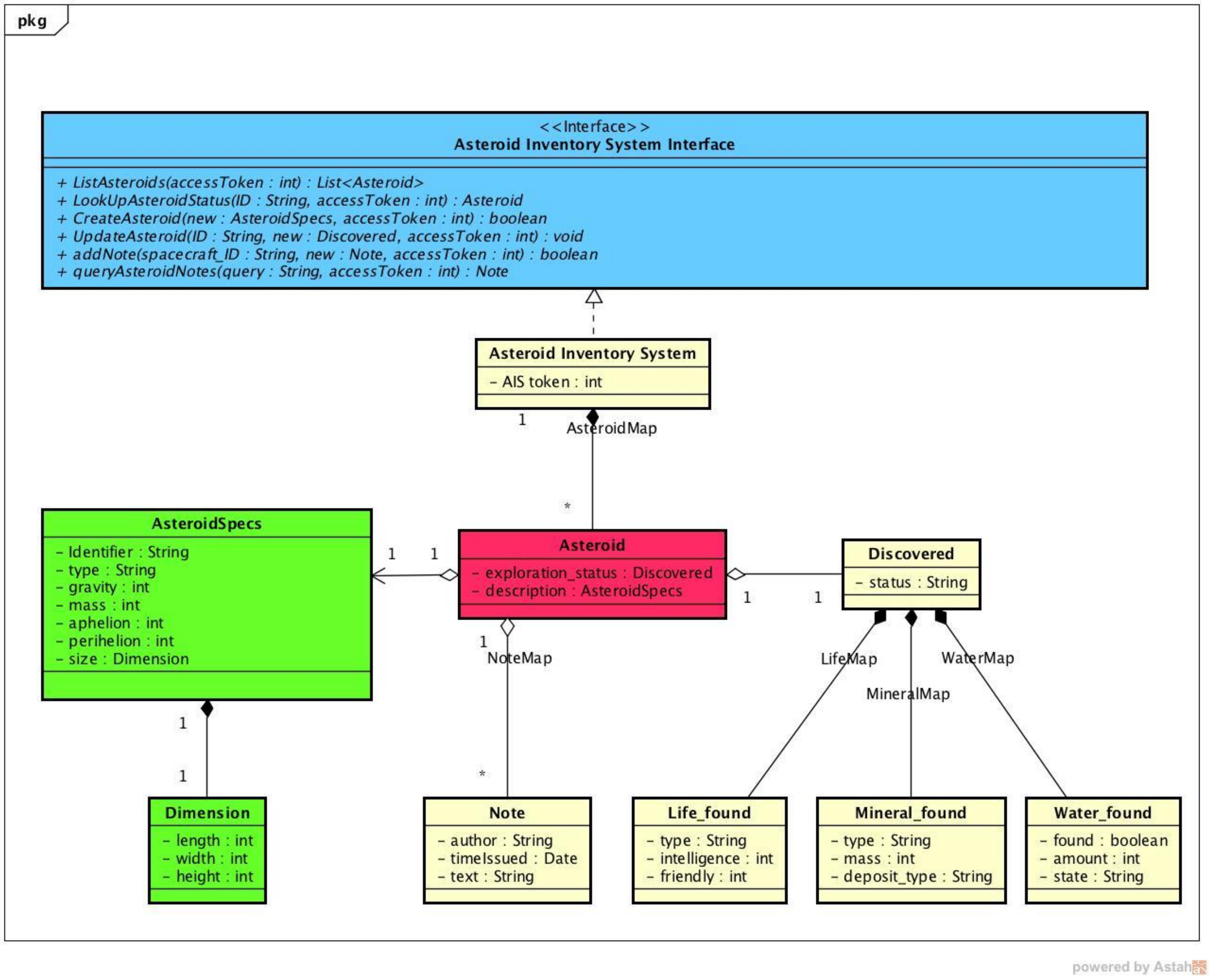

Fig 8. Asteroid Inventory System class diagram

Asteroid Inventory System Interface

This is the service interface of the AIS (Asteroid Inventory System). Each method requires an access token which will be validated by the Authentication service. All methods in the Mission Management Interface are equally implemented in the AIS class.

Methods

Method name	Signature	Description
ListAsteroids	(accessToken : int) : List<Asteroid>	Public method that returns a list of known asteroids

LookUpAsteroidStatus	(ID : String, accessToken : int) : Asteroid	Public method that returns an asteroid by its ID
CreateAsteroid	(new : AsteroidSpecs, accessToken : int) : boolean	Public method that creates new asteroids
UpdateAsteroid	(ID : String, new : Discovered, accessToken : int) : void	Public method that updates an asteroid with a new discovery object
addNote	(spacecraft_ID : String, new : Note, accessToken : int):boolean	Public method that adds a note to a specified asteroid
queryAsteroidNotes	(query : String, accessToken : int) : Note	Public method that returns a queried note based on a String text

AIS (Asteroid Inventory System)

This is the Asteroid Inventory System which is dependent on the Asteroid Inventory System Interface to execute service requests from other subsystems. It is a singleton class that automatically implements all the Asteroid Inventory System Interface Methods.

Properties

Property Name	**Type**	**Description**
AIS token	int	Stores the AIS token

Associations

Association Name	**Type**	**Description**
AsteroidMap	HashMap<string,Asteroid>	stores all asteroid according to their names/IDs

Asteroid

This is the class which stores all information and features of all asteroids discovered or know by the AES.

Properties

Property Name	**Type**	**Description**
exploration status	Discovered	Stores and tracks information on all life, minerals and water found on the asteroid
description	Asteroid Specs	Stores all known features of the asteroid such as ID, type, mass, gravity, dimensions etc.

Associations

Association Name	**Type**	**Description**
NoteMap	Time	Stores all notes associated with each asteroid

Asteroid Specs

This class contains elements for describing any asteroid such as ID, type, mass, gravity, dimensions etc. An example is the "description" instance in Asteroid class

Properties

Property Name	**Type**	**Description**
identifier	String	Stores the asteroid's name or identification
type	String	Stores the asteroid type
gravity	Int	Stores the asteroid's gravity
mass	int	stores the asteroid's mass
aphelion	int	stores the asteroid's furthest distance from the sun
perihelion	int	stores the asteroid's closest distance from the sun.
size	Dimension	stores the physical dimensions of the asteroid

Dimension

This class contains the physical specification of any spacecraft such as the "size" instance in asteroid specs.

Properties

Property Name	**Type**	**Description**
length	int	stores the length of each respective asteroid
width	int	stores the width of each respective asteroid
height	int	stores the height of each respective asteroid

Notes

This class described the format used to store notes connected to each asteroid and are all saved in the Asteroid Class NoteMap association

Properties

Property Name	**Type**	**Description**
author	String	stores the note's author name
timeIssued	time	stores the note's time of creation
text	String	stores the note's details/information

Discovered

This is the class which stores all discovered information for each asteroid such as the "exploration_status" instance in Asteroid class.

Properties

Property Name	**Type**	**Description**
status	String	This describes the current discovery status of the asteroid

Associations

Association Name	**Type**	**Description**
LifeMap	HashMap<String,Life_found>	stores all discovered life by their types
MineralMap	HashMap<String,Mineral_found>	stores all discovered minerals by their types
WaterMap	HashMap<String,Water_found>	stores all discovered water deposits by exactly where they were discovered

Life_found

This is the Discovered subclass that describes the types of life found on the asteroid

Properties

Property Name	**Type**	**Description**
type	String	describes the type of life found
intelligence	int	describes the intelligence level of the discovered lifeform
friendly	int	describes the friendliness level of the lifeform

Mineral_found
This is the Discovered subclass that describes the types of minerals found on the asteroid

Properties

Property Name	**Type**	**Description**
type	String	describes the type of mineral found
mass	int	describes the mass of the discovered mineral
deposit_type	int	describes the deposit type of the discovered mineral

Water_found
This is the Discovered subclass that describes the types of minerals found on the asteroid

Properties

Property Name	**Type**	**Description**
Found	boolean	describes if water was found
amount	int	describes the estimated volume of water found
state	String	describes the state of the water found

SEQUENCE DIAGRAMS

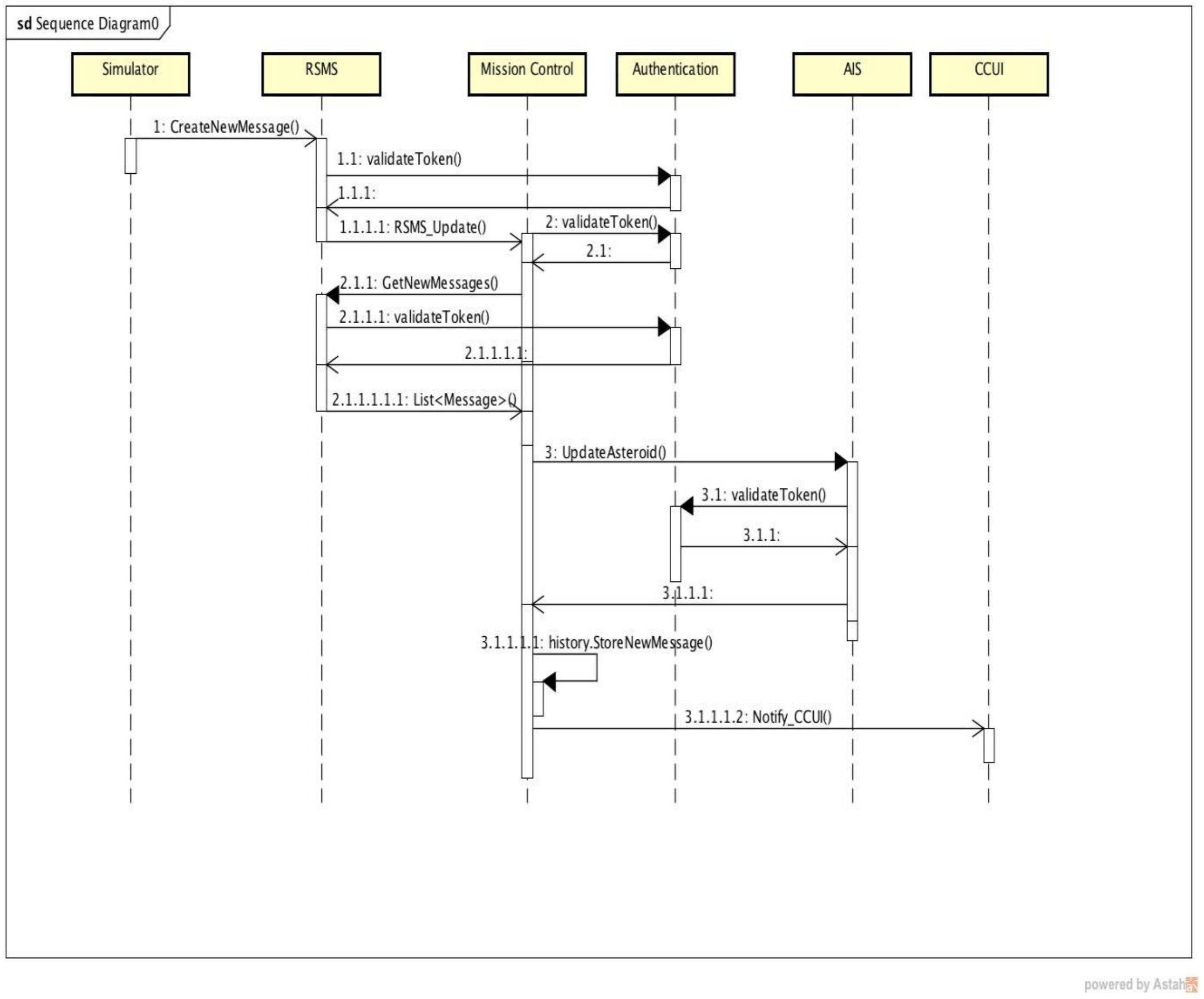

Fig 9. Sequence Diagram for the flow of message when a spacecraft discovers water on target asteroid

SEQUENCE DIAGRAMS

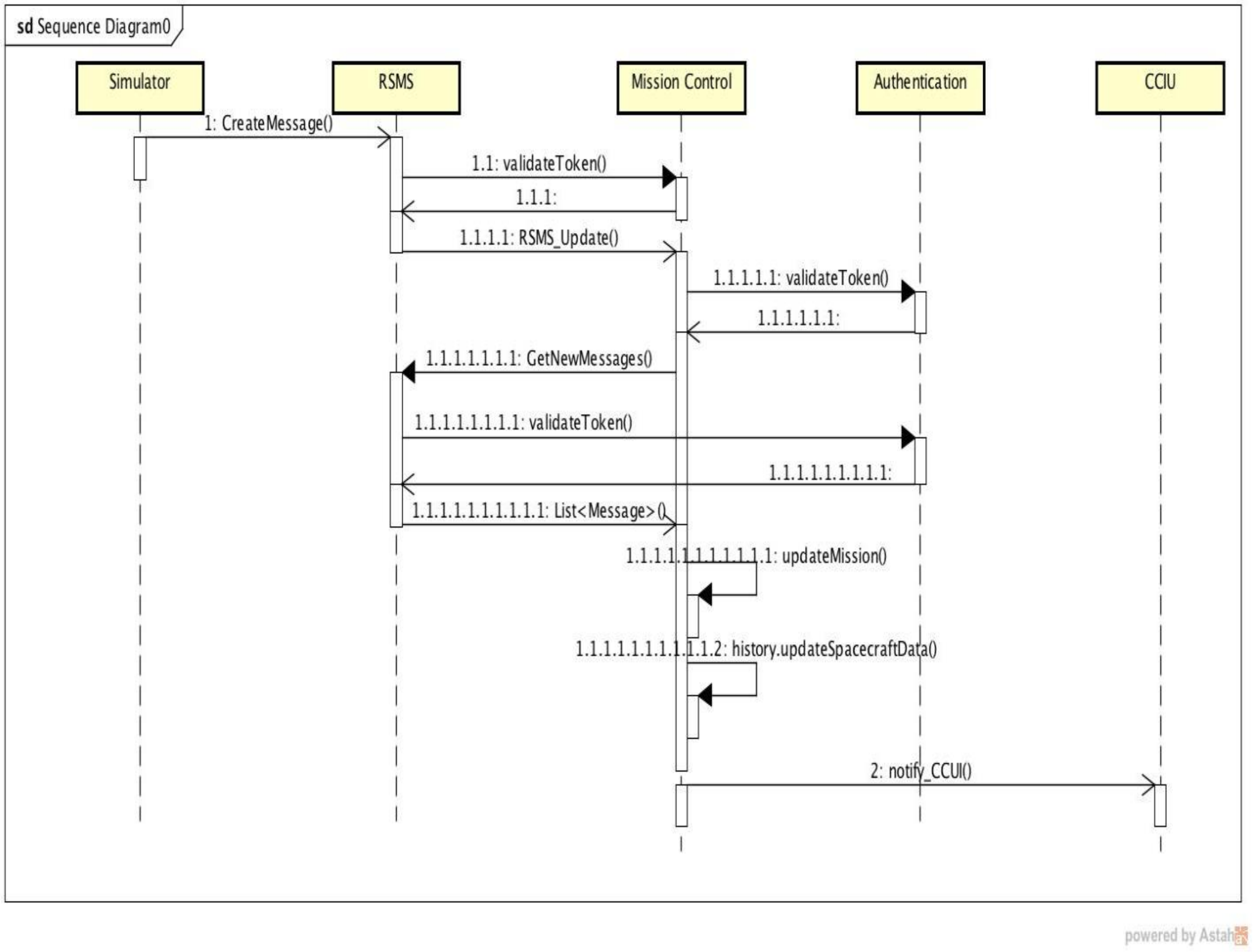
sd Sequence Diagram0
Simulator
RSMS
Mission Control
Authentication
CCIU
1: CreateMessage()
1.1: validateToken()
1.1.1:
1.1.1.1: RSMS_Update()
1.1.1.1.1: validateToken()
1.1.1.1.1.1:
1.1.1.1.1.1.1: GetNewMessages()
1.1.1.1.1.1.1.1: validateToken()
1.1.1.1.1.1.1.1.1:
1.1.1.1.1.1.1.1.1.1: List<Message>()
1.1.1.1.1.1.1.1.1.1.1: updateMission()
1.1.1.1.1.1.1.1.1.1.2: history.updateSpacecraftData()
2: notify_CCUI()
powered by Astah

Fig 10. Sequence diagram for the flow of events of status message when a spacecraft completes its mission

SEQUENCE DIAGRAMS

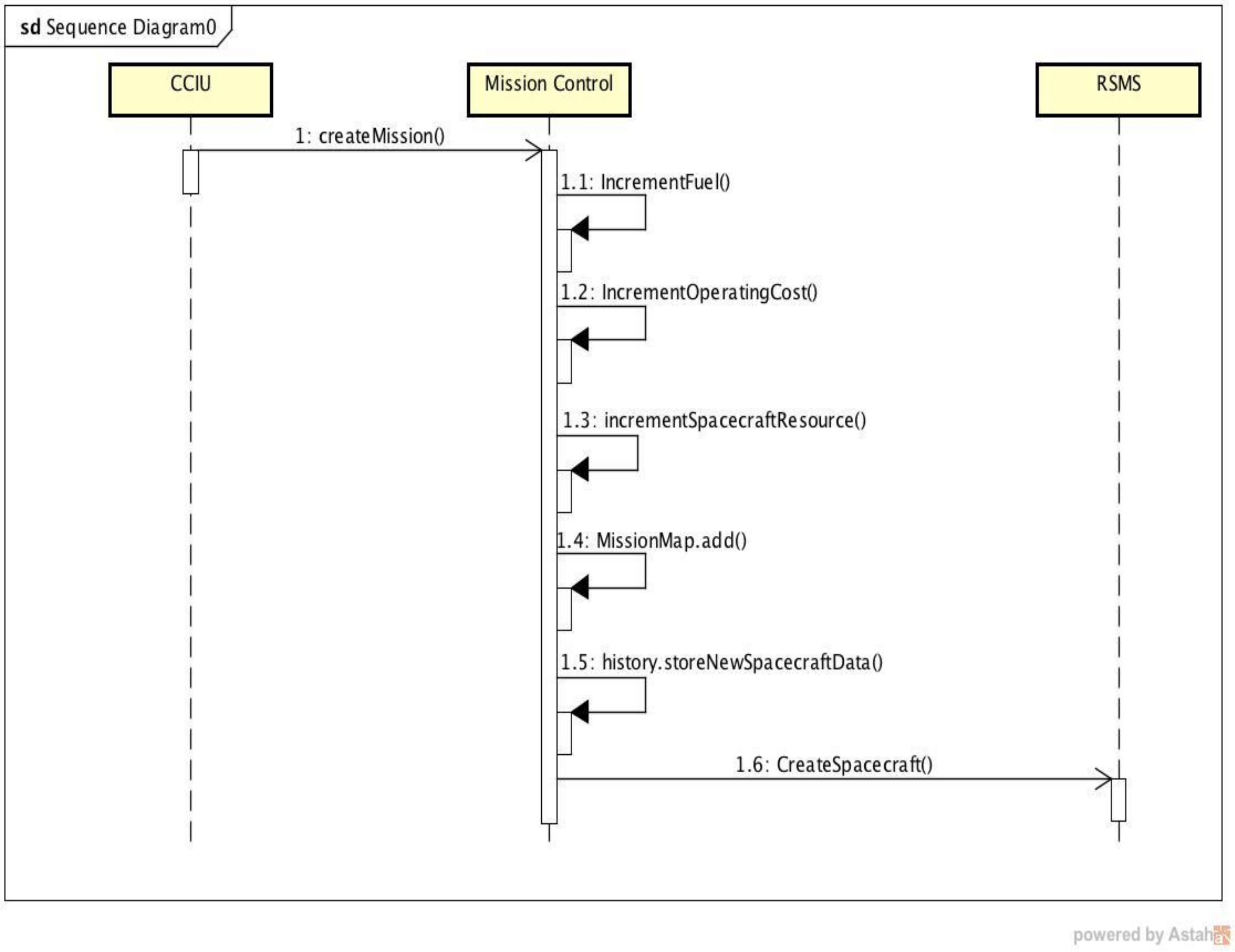

Fig 11. Sequence diagram for the flow of events during the provisioning of a new mission

ACTIVITY DIAGRAM

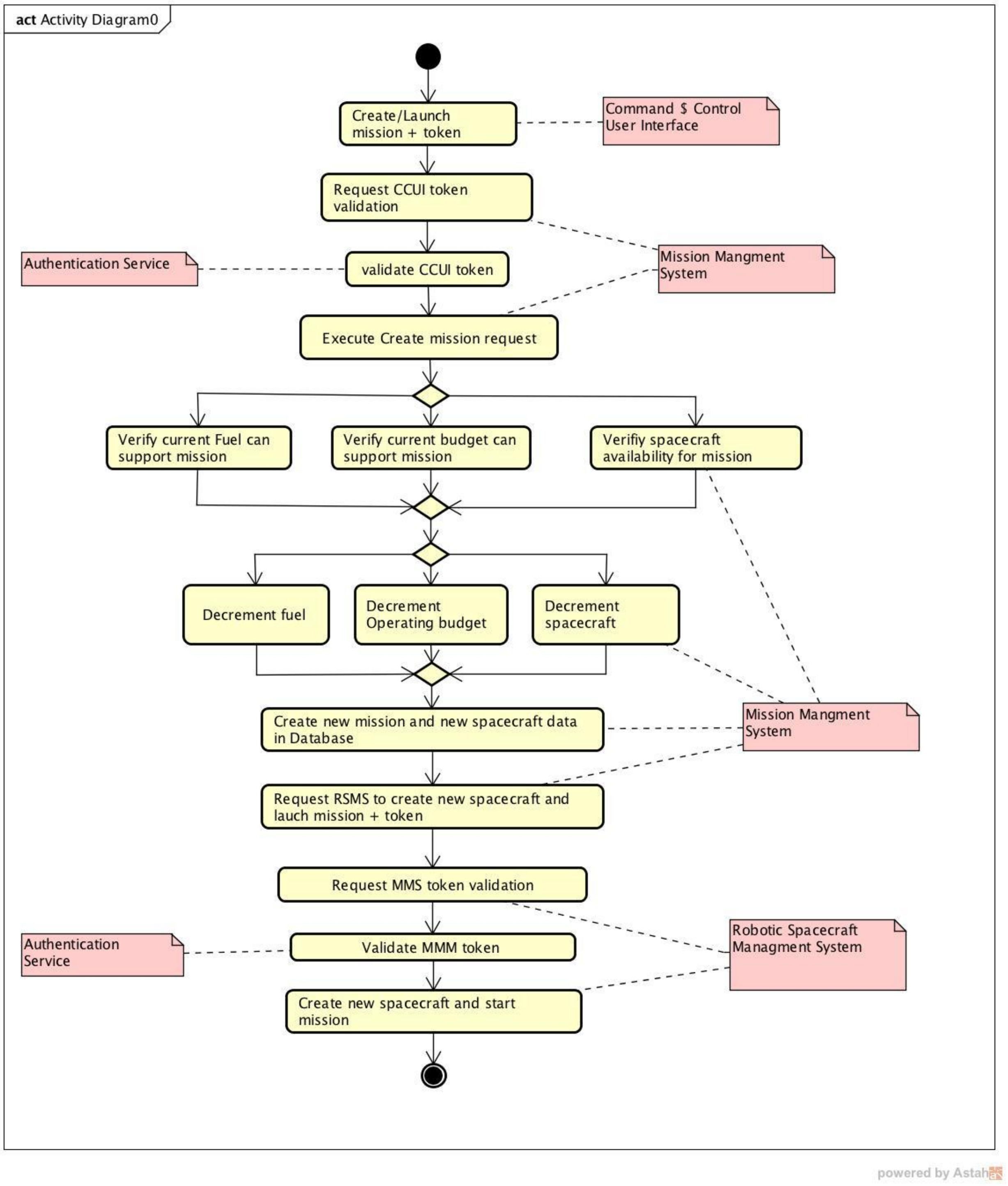
act Activity Diagram0
Create/Launch mission + token
Command $ Control User Interface
Request CCUI token validation
Authentication Service
validate CCUI token
Mission Mangment System
Execute Create mission request
Verify current Fuel can support mission
Verify current budget can support mission
Verifiy spacecraft availability for mission
Decrement fuel
Decrement Operating budget
Decrement spacecraft
Create new mission and new spacecraft data in Database
Mission Mangment System
Request RSMS to create new spacecraft and lauch mission + token
Request MMS token validation
Authentication Service
Validate MMM token
Robotic Spacecraft Managment System
Create new spacecraft and start mission
powered by Astah

Fig 12. Activity diagram documenting the provisioning of a new mission

Command and Control user Interface description

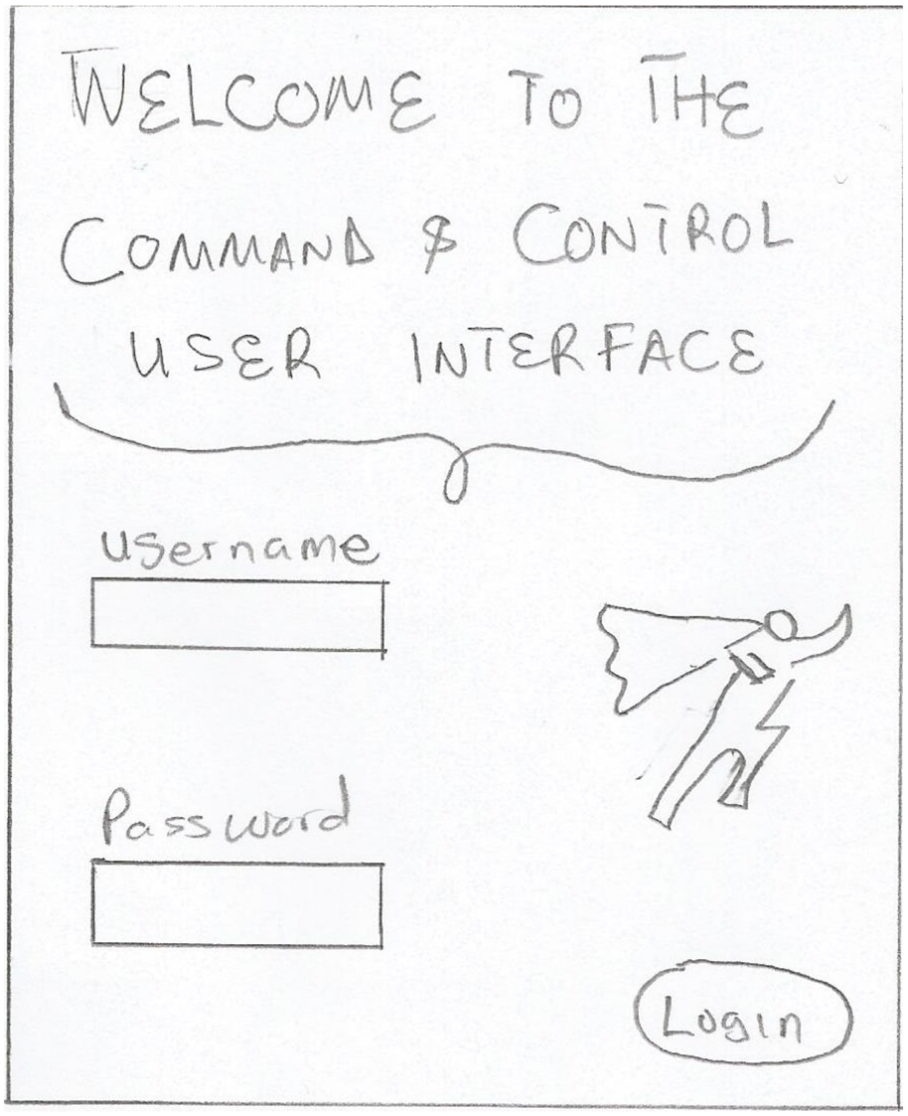

Fig 13. The CCUI Login/logout Graphical User Interface

Dependency

- This login and logout services depends on the login() and logout() methods of the authentication service for secure ujser access

Command and Control user Interface description

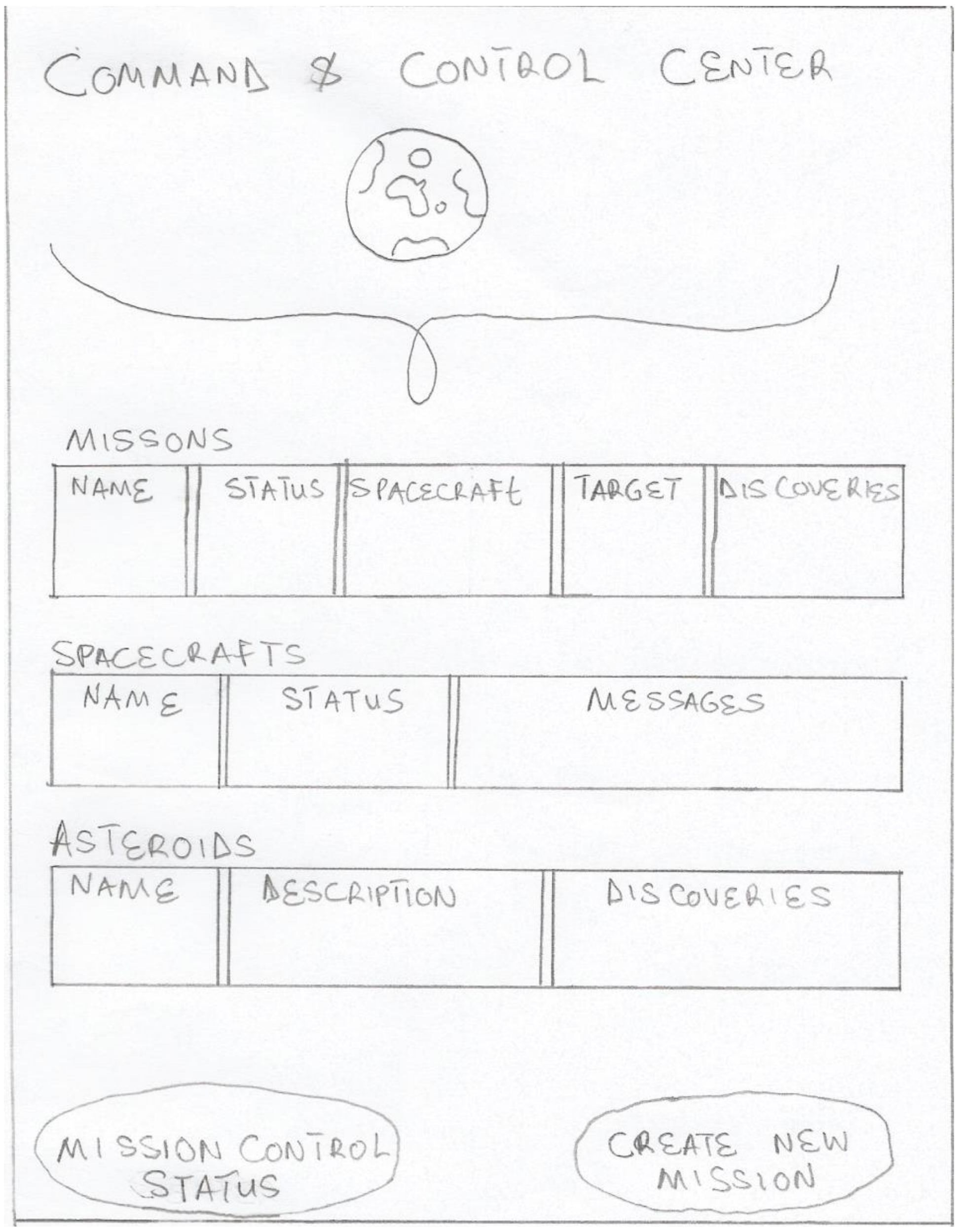

Fig 14. The CCUI AES status Graphical User Interface

Dependencies

- All Mission data are obtained via the **MMS** missionStatusLookUp(), get_MMS_Resource_status() and get_MMS_System_status() methods
- All Spacecraft data and messages are obtained via the **RSMS** ListAllSpacecrafts(), lookUpSpacecraft() and GetNewMessages() methods.
- All Asteroid data are obtained via the **AIS** ListAsteroid(), LookUpAsteroid() and queryAsteroid() methods.

Command and Control user Interface description

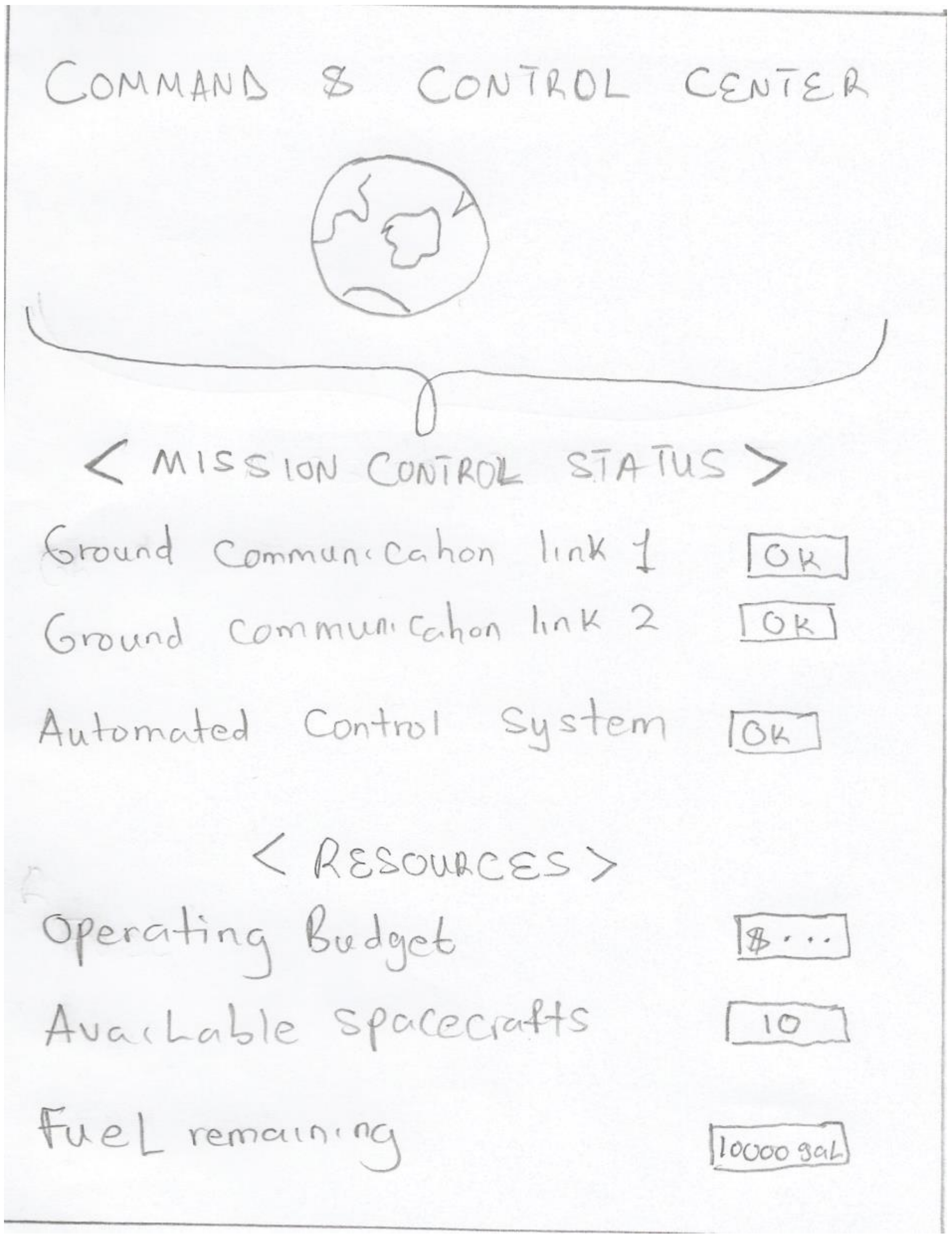

Fig 15. The CCUI Mission Control status Interface

Dependences

- All Mission data are obtained via the MMS get_MMS_Resource_status() and get_MMS_System_status() methods.

Command and Control user Interface description

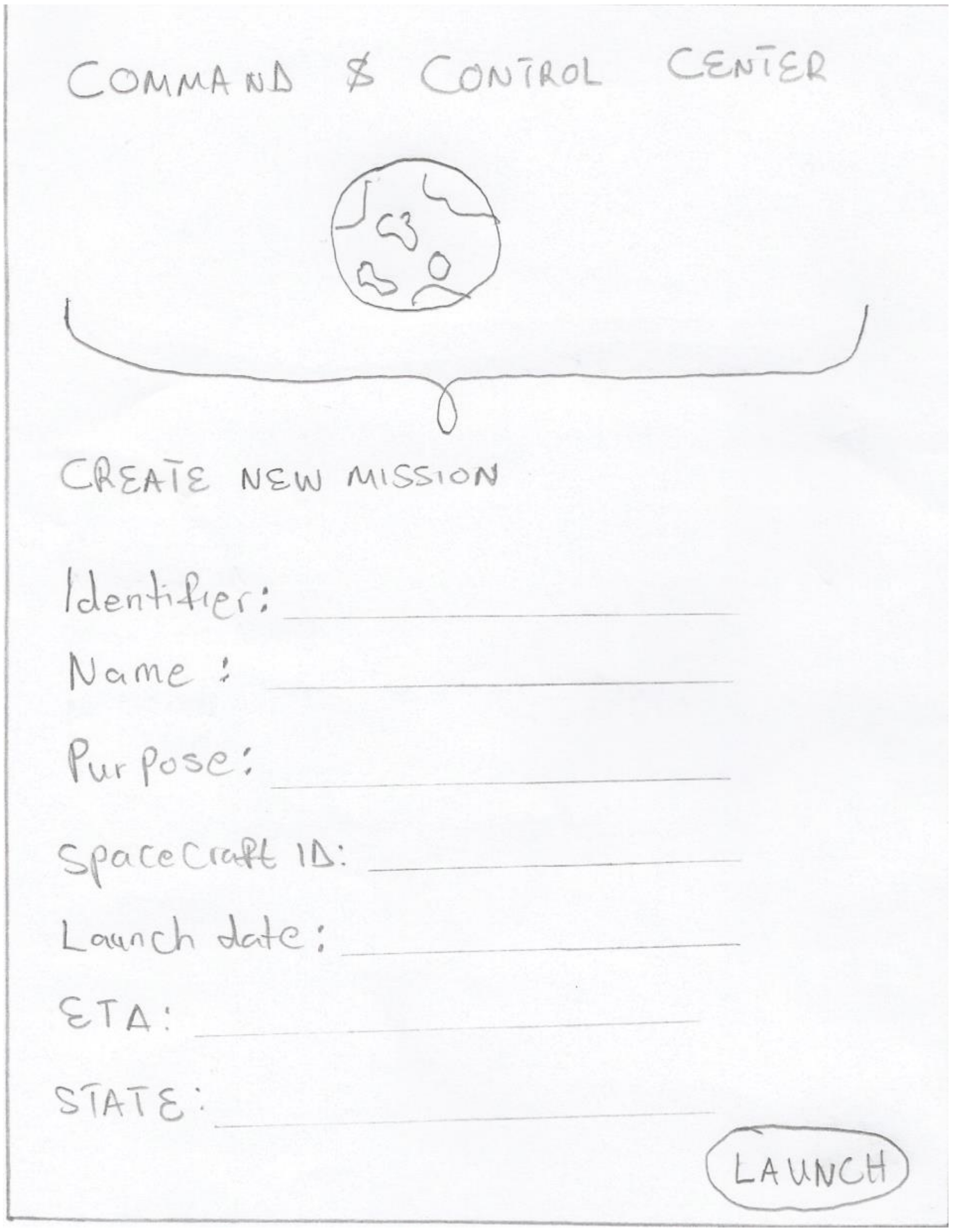

Fig 16. The CCUI Mission creation page interface.

Dependencies

- The mission creation page relies on the **MMS** createMission() method.

Abstractness and Instability Metrics

Abstractness = Ce/(Ce+Ca)
Where Ce are the efferent classes and Ca are the afferent classes in the package

Instability metric, A = Na/Nc
Where Na is the number of non-instantiable classes and interfaces and Nc be total number of classes and interfaces in the package

MMS:
Ce = AIS + RSMS = 2
Ca = CCUI = 1
Therefore,
Abstractness = 2/ (2+1) = 0.67
Instability metric = 1/7 = 0.143

RSMS:
Ce = MMS = 1
Ca = MMS + CCUI = 2
Therefore,
Abstractness = 1/ (1+2) = 0.33
Instability metric = 3/13 = 0.23

AIS:
Ce = 0
Ca = MMS + CCUI = 2
Therefore
Abstractness = 0/2 = 0
Instability metric = 1/10 = 0.1

How the requirements are met by the AES Implementation

Robotic Spacecraft Management System

- It implements singleton design pattern, observer pattern relationship with the MMS and factory pattern for creating new spacecraft
- All required attributes are implemented
- All required methods are implemented
- It implements a Service Interface and maintains a modular architecture
- It implements token validation to ensure request integrity

Mission Management System

- It implements singleton design pattern and observer pattern relationship with the RSMS.
- All required attributes are implemented
- All required methods are implemented
- It implements a Service Interface and maintains a modular architecture
- It implements token validation to ensure request integrity
- It also implements **persistence** by storing all messages and spacecraft data involved in the AES in **the "history" Database.** The AIS is already ensuring persistence on all Asteroid Data and the MMS MissionMap is already storing all mission data; therefore the Database only stores all new spacecraft and messages which will complete the persistency of the whole system.

Asteroid Inventory System

- It implements singleton design pattern.
- All required attributes are implemented
- All required methods are implemented
- It implements a Service Interface and maintains a modular architecture
- It implements token validation to ensure request integrity

User Interface (Command and Control user Interface)

- The GUI is described to enable full monitoring and control of the Asteroid Exploration System.
- It includes all necessary commands and their respective subsystem service interface dependencies

Authentication Service

- It ensures CCUI admin login integrity
- It also ensures subsystem request integrity using the accessToken elements

Exception handling

The following exceptions will be implemented in the various subsystems

- **InsufficientResourceException**: This is thrown by the MMS when there are insufficient resources to create or launch a mission.
- **UnknownMissionException**: This is thrown by the MMS when an unknown mission ID is requested by the CCUI.
- **UnknownSpacecraftexception**: This is thrown by the RSMS when an unknown spacecraft ID is used in a lookup request.
- **InvalidTokenException**: This is thrown by the subsystems when a request is accompanied by an invalid token.
- **SpacecraftyAlreadyExistsexception**: This is thrown by the RSMS when asked to create a spacecraft which already exists.
- **MissionAlreadyExistsexception**: This is thrown by the MMS when asked to create a mission which already exists.
- **UnknownAsteroidException**: This is thrown by the AIS when asked to lookup an unidentified asteroid ID
- **AsteroidAlreadyExistsException**: This is thrown by the AIS when asked to create an asteroid that already exists.

Testing

- **Functional testing:** This is accomplished by simulating several missions and messages from spacecraft to ensure that the entire AES is functional and accurate information was exchanged.
- **Performance testing:** This involves creating thousands of missions and messages from spacecraft to ensure that the entire AES is functional and accurate information was exchanged even when the mission is stressed. It also involved checking the process time to ensure that the system speed is not greatly affected
- **User testing:** This involved ensuring that the User interfaces are simple to understand and elegant enough for a business venture.

Risks

- Each subsystem’s tokens should be updated periodically to prevent all future security breach
- The MMS should always monitor its Communication links and Automated Control System to prevent message loss or entire system failure
- All information should always be persisted with timestamps to ensure effective future system debugs.

Thanks for an eventful semester!!! I enjoyed the classes and projects!!!