

# Asteroid Exploration System Design Document

**Author: Ann-Katrin Reuel**

## Introduction

This design document defines the design for the Asteroid Exploration System (AES), a software architecture developed for an artificial project for the International Space Agency (ISA) to manage their mining and exploration efforts throughout the universe. It is a software system designed to manage the missions that explore and mine asteroids, the asteroids discoveries as well as the related spacecraft.

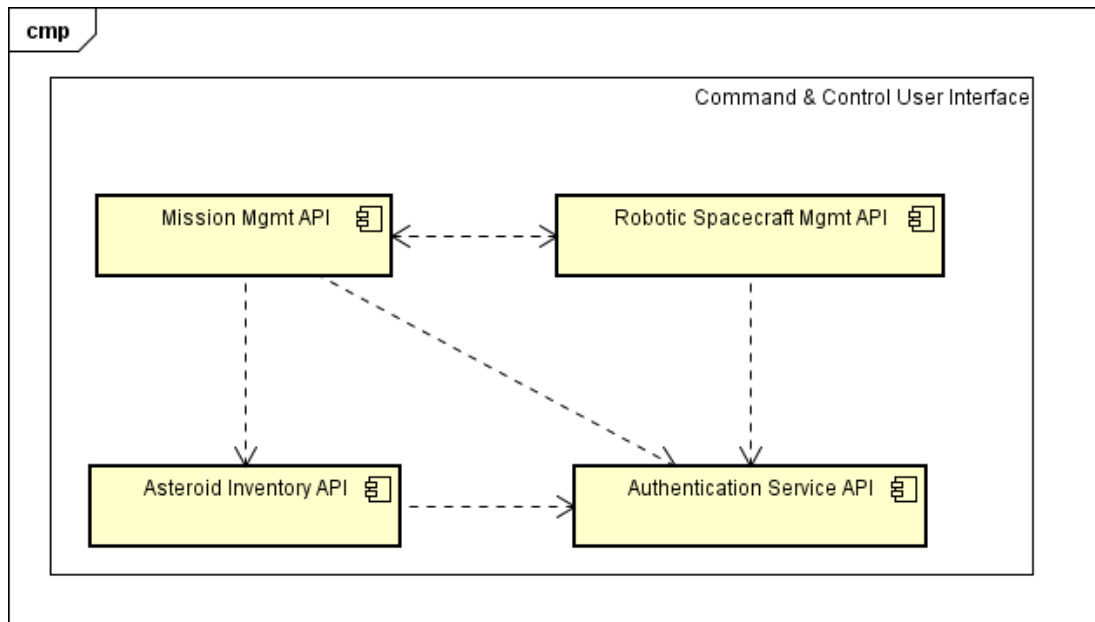
The document will be structured in the following way:

1. Overview: Identification of problem to be solved and how this is achieved on a high level. Also an overview of how the subsystems fit together (includes module component diagram + explanation of interdependencies)
2. Requirements: What is required for the holistic system as well as for the subsystems
3. Use Cases: In which scenarios will the architecture be employed and for what can it be used
4. Implementation: How is the design implemented and how are the requirements met given the class and interface structure
5. Class Diagram: Overview of the classes involved in each package. One subsystem is equivalent to one package
6. Class Dictionary: Listing of classes present in each package, with an overview of corresponding methods, properties and attributes
7. Implementation Details: Detailed explanation of how the subsystems interact with each other to fulfill requirements
8. Sequence Diagrams: Further presentation of how different subsystems interact with each other in three sample cases
9. Activity Diagram: Shows the process flow for a new mission being created
10. Command & Control User Interface: Sample Wireframes to design the user interface
11. Exception Handling: Identification of potential error sources and how the system would deal with them
12. Testing: Provision of sample test case scenarios
13. Risks: Recognition of potential additional error sources and parts of the system that need special attention

## Overview

The problem to be solved is to design a coherent system that integrates the five subsystems required by the AES: the Asteroid Inventory System (AIS), the Robotic Spacecraft Management System (RSMS), the Mission Management System (MMS) and the Authentication Service. Furthermore, an intuitive Command and Control User Interface will be designed to allow for easy access to all services to the users.

The following Module Component Diagram gives an overview of the system architecture and how the different subsystems are being integrated:



The Command & Control User Interface is a user-facing layer that makes use of the service APIs of the other subsystems. The Authentication Service API is used by all subsystems to enable access restriction and user identification within the various packages. The Asteroid Inventory API, on the other hand, is only used by the Mission Management API. The latter is entitled to update the Asteroid Inventory System with regards to discovered asteroids and their characteristics. At the same time, the Mission Management API has a bidirectional connection to the Robotic Spacecraft Management System: It accesses the Spacecraft Management to manage available spacecraft and to employ them to missions as appropriate while the system is able to receive and act on messages sent by the spacecraft.

By designing a fully integrated system, ISA is provided with a coherent architecture to optimally support the exploration and mining of resources on asteroids within the asteroid belt.

## Requirements

Overall, ISA requires the system to be fully functional, internally cohesive and loosely coupled. Furthermore, the system is required to follow a Service Oriented Structure, in accordance with level 5 of the Modularity Maturity Model, as well as to store the data persisted in a database. All subsystems are also required to enable restricted access via a central Authentication Service.

In addition to those requirements, the subsystems need to provide the functionalities listed below. For further details, please refer to the Asteroid Exploration System Requirements.

### Asteroid Inventory System

The AIS is required to maintain an inventory of all discovered asteroids and their characteristics. It must support the functionality to return a list of all known asteroids, to lookup an asteroid by ID, to create and update asteroids as well as to perform a text-based search query across the database.

### Robotic Spacecraft Management System

The RSMS provides a platform to access and manage all available spacecraft. It is supposed to support adding a new spacecraft if enough resources are available, listing of all existing spacecraft, looking up spacecraft by ID and saving updates to spacecraft.

### **Mission Management System**

The MMS is a subsystem that enables the user to control missions, with points of intersection with both the Asteroid Inventory System as well as the Robotic Spacecraft Management System. Within the MMS, the user should be able to create a new mission while at the same time it checks whether resources are available and reduces them accordingly or sends a message stating that the mission cannot be created because resources are missing. The system should also be in charge of the maintenance and surveillance of the status of land-based communication links and the automated control system. Additionally, the MMS will be able to receive messages from spacecraft and act on them, e.g. update of discoveries or the spacecraft status.

### **Authentication Service**

The employed Authentication Service will follow the design that was created in assignment 4. It will provide the other subsystems with an interface to validate users and control access to the methods within each software component. For a detailed design description, please refer to the Authentication Service Design Document.

### **Command and Control User Interface**

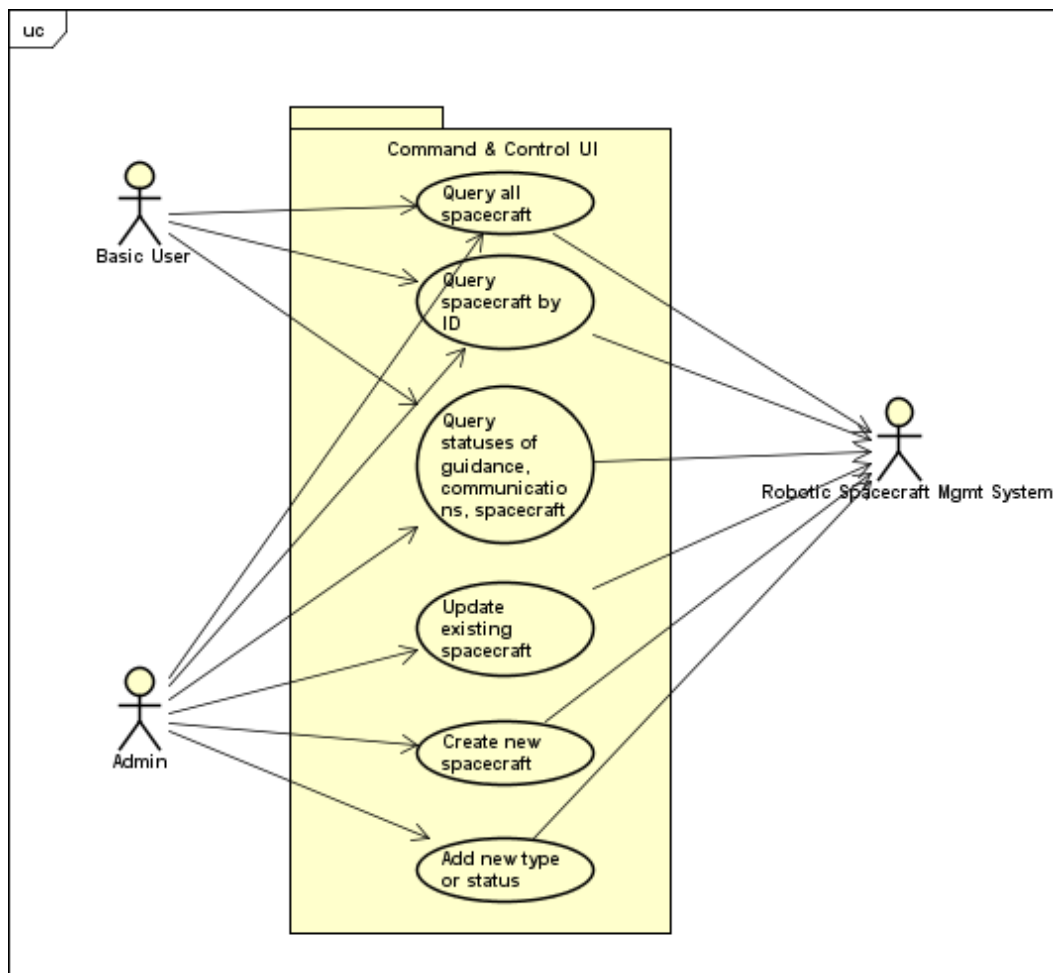
The CCUI will not be a subsystem itself but rather a platform that accesses the service APIs from the four other subsystems. The user will be able to login and logout via the CCUI, define missions, monitor and update missions, spacecraft and asteroids, monitor available resources, monitor the status of ground-based communication links and the automated control system as well as to display incoming messages from spacecraft.

Overall, the design should reduce the complexity of the system interaction and provide the user with an easy-to-use, functional GUI.

## Use Cases

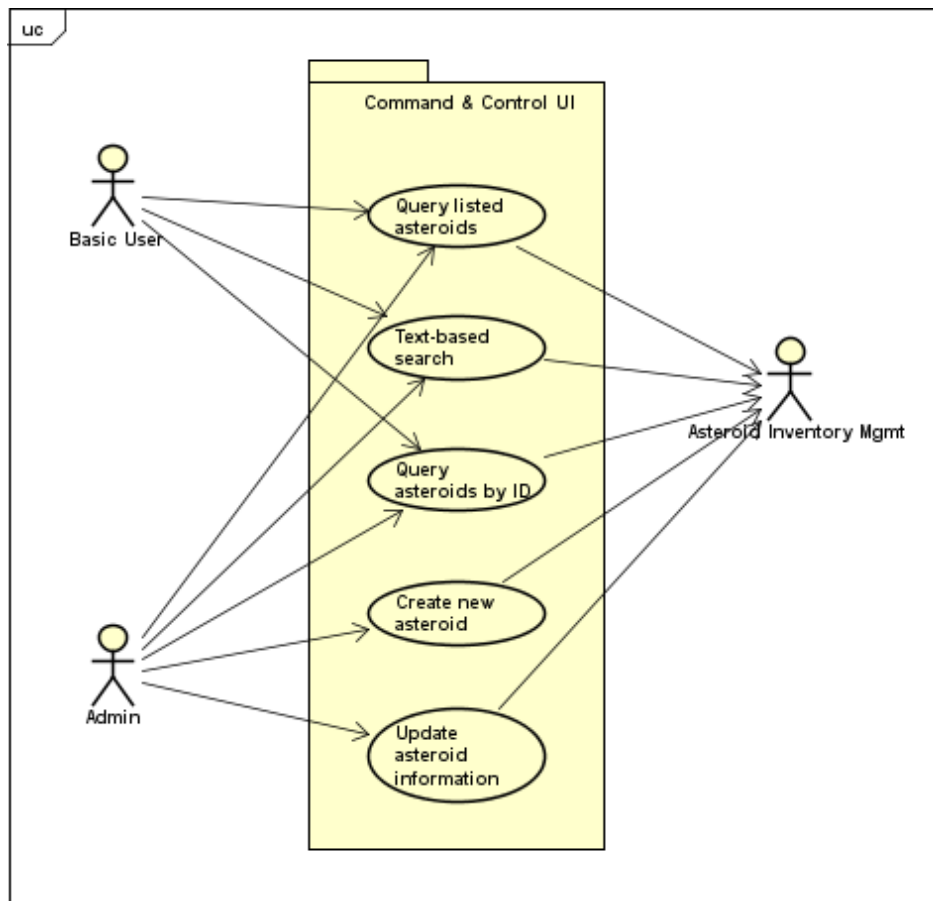
### A. Case 1: Robotic Spacecraft Management System

1. A basic user can
  - i. query all spacecraft via the CCUI to receive a list of all listed spacecraft from the RSMS
  - ii. query spacecraft by ID to get the properties of a specific spacecraft from the RSMS
  - iii. query the status of guidance, communications and spacecrafts from the RSMS
2. An admin user can
  - i. pursue all actions that a basic user is entitled to
  - ii. update the properties of existing spacecraft in the RSMS via the CCUI
  - iii. create a new spacecraft in the RSMS via the CCUI
  - iv. add a new type to spacecrafts in the RSMS via the CCUI
  - v. add a new status to spacecrafts, communications or guidance in the RSMS via the CCUI



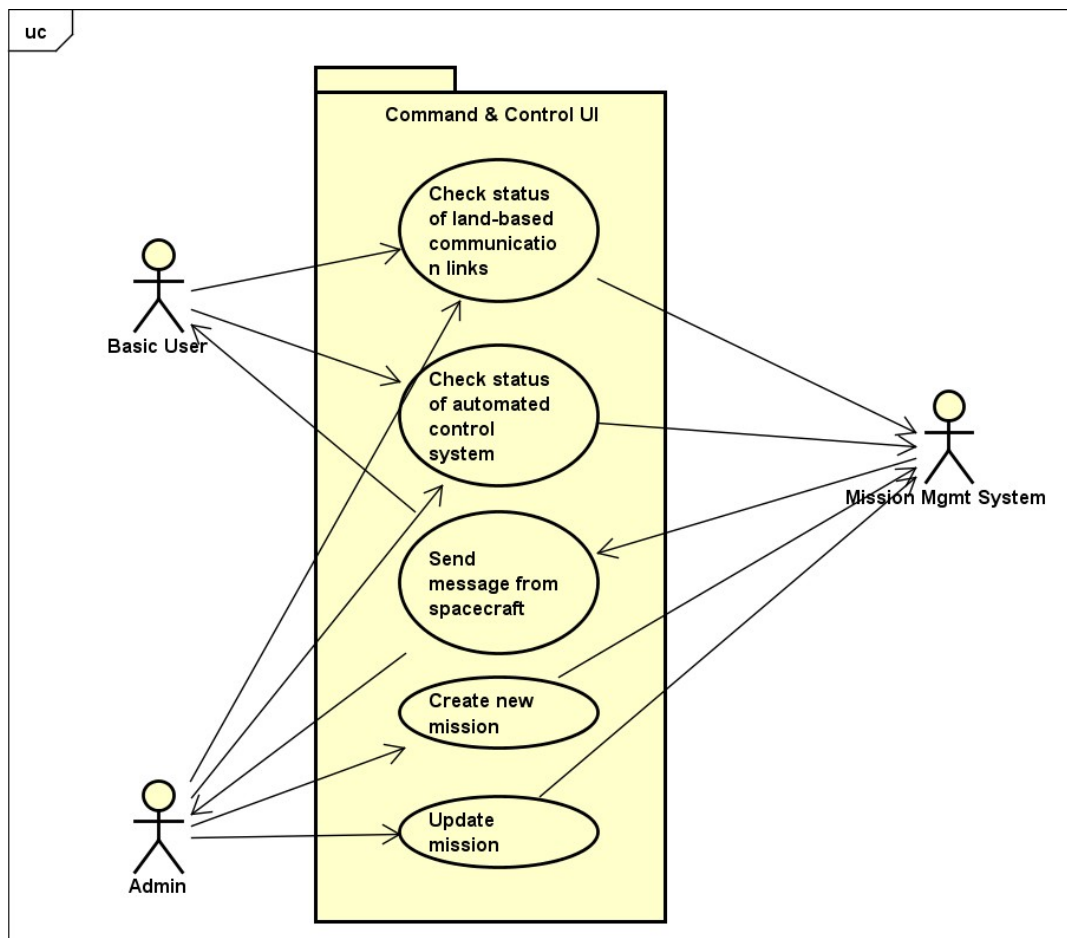
B. Case 2: Asteroid Inventory Management System

1. A basic user can
  - i. query for all listed asteroids to receive a list of all known asteroids from the AIMS via the CCUI
  - ii. pursue a free, text-based search to retrieve asteroids which match the search from the AIMS via the CCUI
  - iii. query asteroids by ID to retrieve information about a specific asteroid from the AIMS via the CCUI
2. An admin can
  - i. pursue all actions a basic user is entitled to
  - ii. create a new asteroid via the CCUI in the AIMS
  - iii. update information of an asteroid via the CCUI in the AIMS



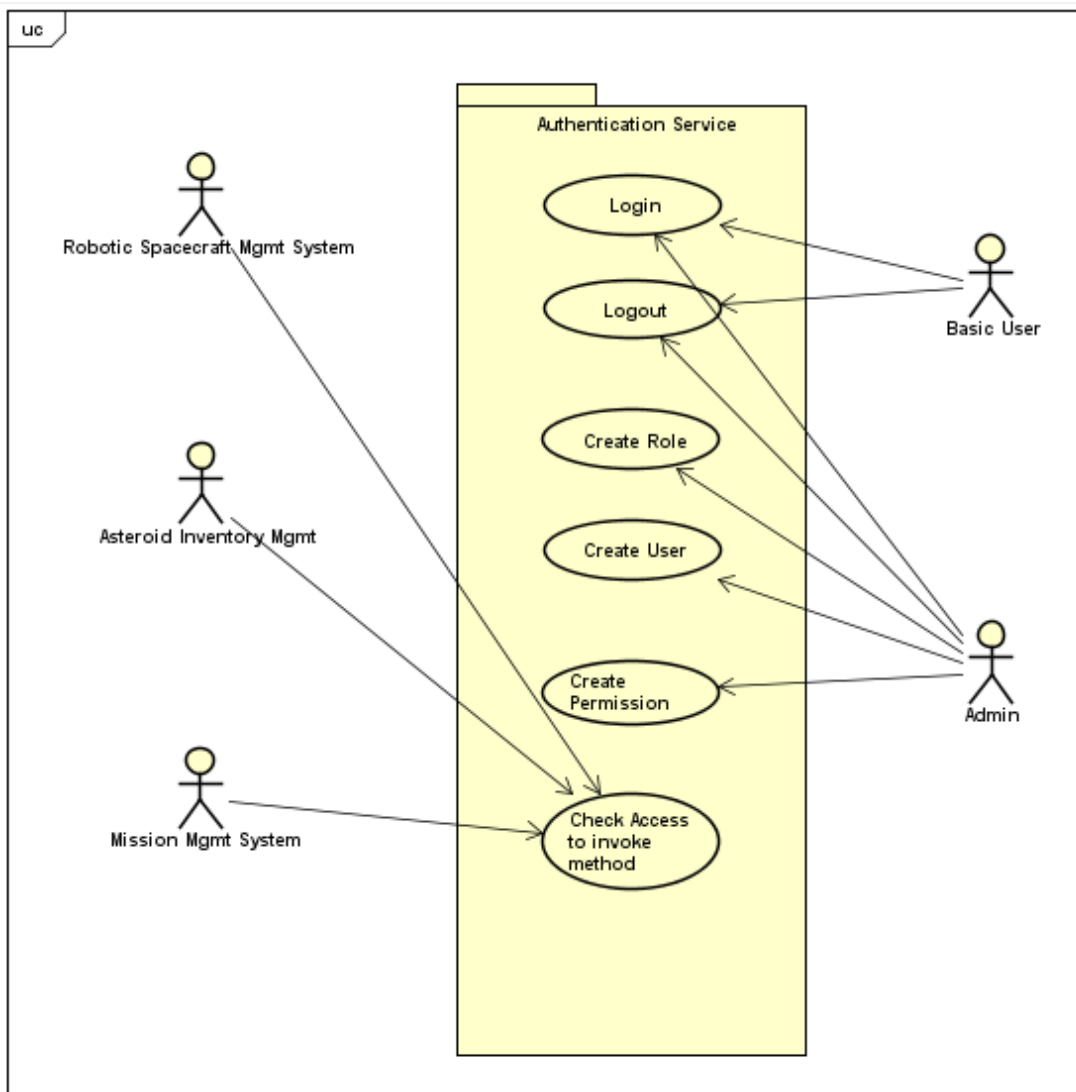
C. Case 3: Mission Management System

1. A basic user can
  - i. check the status of the land-based communication links from the MMS via the CCUI
  - ii. check the status of the automated control system from the MMS via the CCUI
  - iii. receive messages via the CCUI which have been sent from the RSMS to the MMS
2. An admin can
  - i. pursue all actions which a basic user is entitled to
  - ii. create a new mission via the CCUI in the MMS
  - iii. update the specifications of an existing mission via the CCUI in the MMS



D. Case 4: Authentication Service

1. A basic user can
  - i. log in the system
  - ii. log out of the system
2. An admin can
  - i. pursue all actions that a basic user is entitled to
  - ii. create a new role within the authentication service
  - iii. create a new user within the authentication service
  - iv. create and allocate new permissions to existing users
3. The RSMS, the AIMS and the MMS can
  - i. use the Authentication Service to check that the respective user has the access rights to invoke a method or query a database



## Implementation

Java JDK 1.8 is used to implement the AES. It is implemented as micro service within the same Java Virtual Machine to allow direct level method access to the service interfaces.

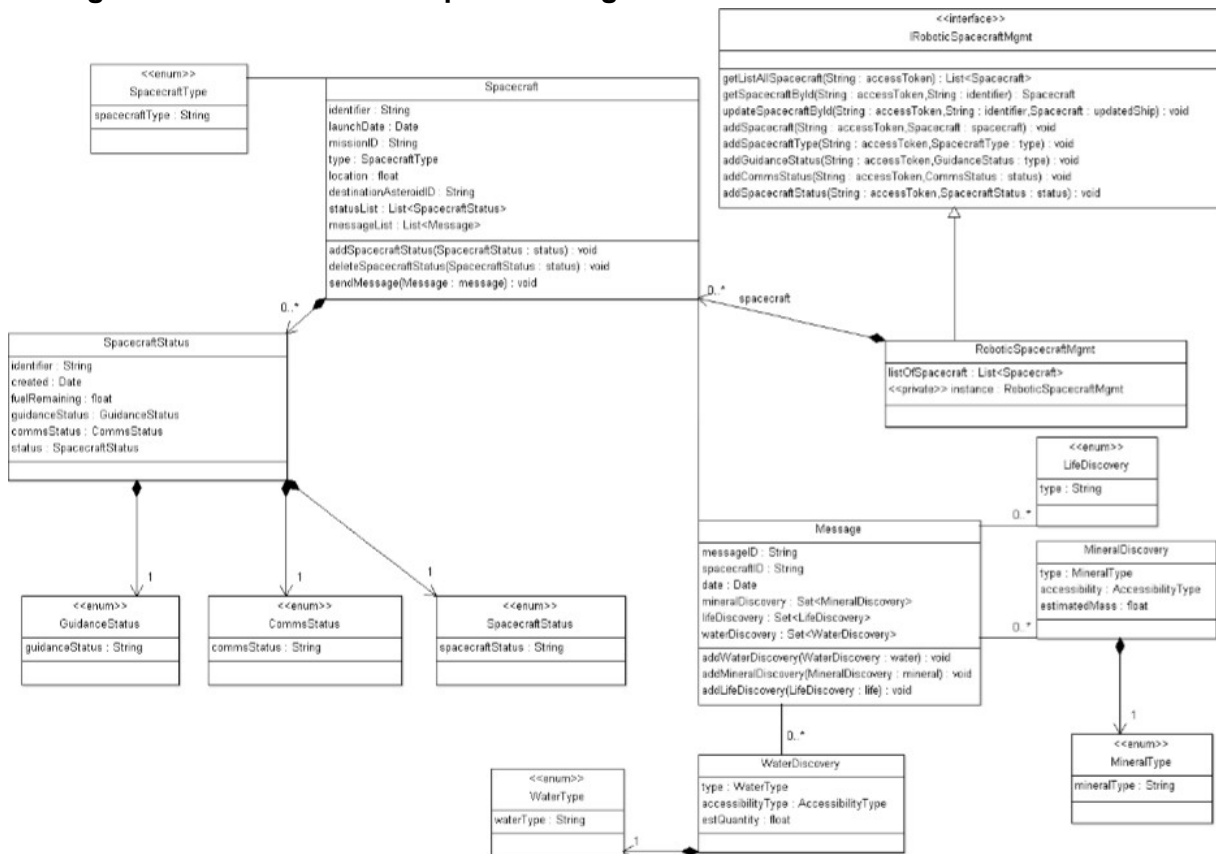
Furthermore, each subsystem will be specified in its own package. As discussed below, we will make use of the Façade and Factory Design patterns. The only public facing interfaces will be the interfaces of the Singleton instances. All other classes will be implemented as package private to ensure an internally cohesive and loosely coupled system employing a Service Oriented Structure.

To ensure consistency across entities and within the database, enumerations will be used to limit a user's ability to specify properties of missions, spacecraft, asteroids and similar entities. The functionalities defined within the requirements will be employed by corresponding classes within the package structure. Those classes will furthermore be equipped with the required features as properties as laid out in the Asteroid Exploration System Requirements.

To be able to better differentiate method access, the authentication service will support basic and admin users with different rights with regards to invoking methods.

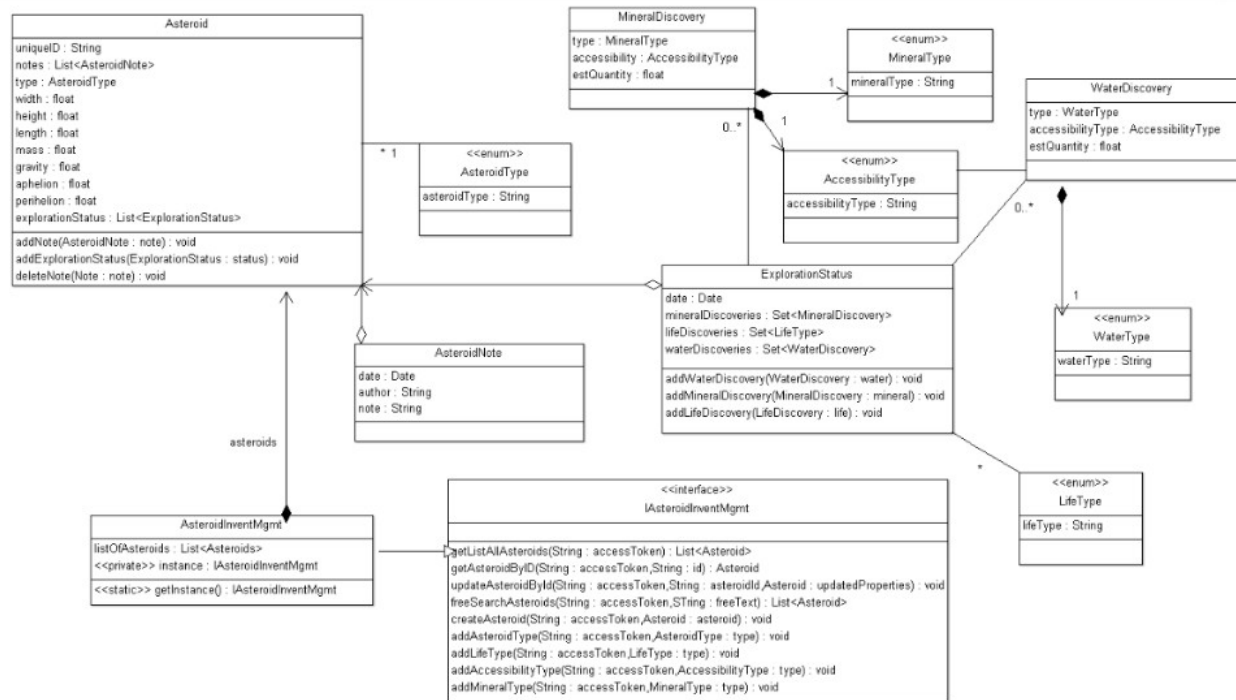
## Class Diagram

Package: cscie97.asn5.roboticspacecraftmgmt

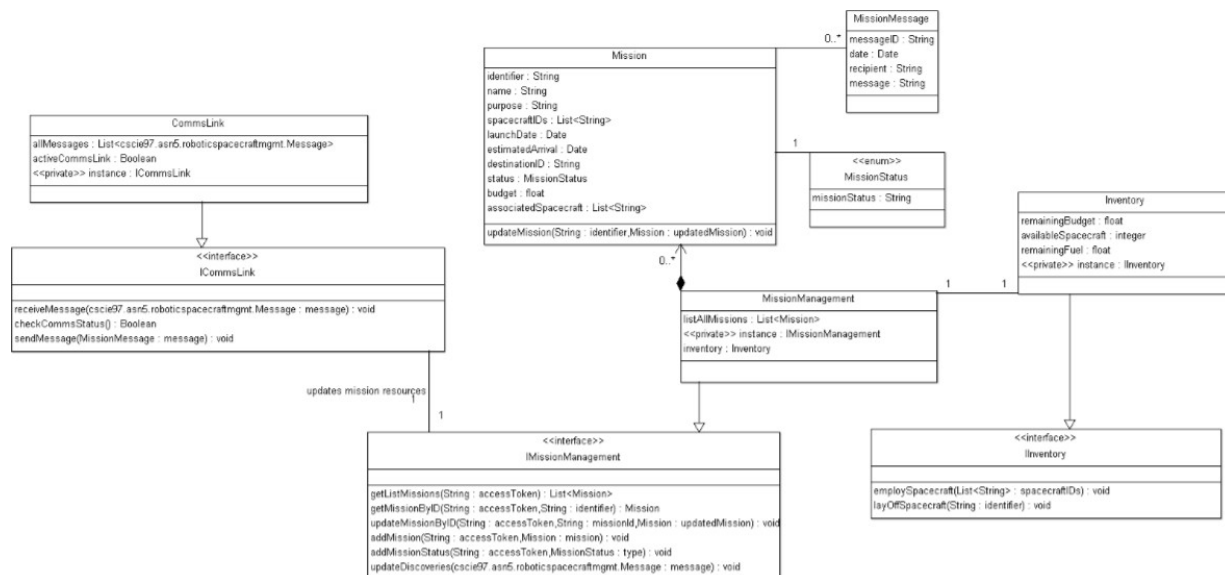




## Package: cscie97.asn5.asteroidinventorymgmt



## Package: cscie97.asn5.missionmgmt



## Class Dictionary

### Package: cscie97.asn5.authenticationservice

This component has been modelled previously and will not be specified again to avoid redundancies. Details regarding the Authentication Service can be found in the Authentication Service Design Document.

### Package: cscie97.asn5.roboticspacecraftmgmt

In addition to the classes and enumerations listed below, this package also has exact copies of several enumeration types and classes from the cscie97.asn5.asteroidinventorymgmt package, including MineralDiscovery, WaterDiscovery, MineralType, AccessibilityType, WaterType and LifeDiscovery.

### IRoboticSpacecraftMgmt <<interface>>

The IRoboticSpacecraftMgmt interface is the core of the RSMS. It contains the methods necessary for the user and systems relying on the RSMS to manage the spacecraft database. By employing an interface as well as the RoboticSpacecraftMgmt class which implements the interface, the façade design pattern is applied, as only public methods will be visible outside of the package.

#### Methods

Method Name	Signature	Description
getSpacecraftByID	(String: accessToken, String: identifier): Spacecraft	Returns a specific spacecraft and its properties based on its unique identifier. Requires an access token for authentication to invoke the method.
getListAllSpacecraft	(String: accessToken): List<Spacecraft>	Returns a list of all spacecraft registered in the system. Requires an access token for authentication to invoke the method.
addSpacecraft	(String: accessToken, Spacecraft : spacecraft): void	Registers a new spacecraft to the system. Requires an access token for authentication to invoke the method. Changes are persisted to database.
addCommsStatus	(String: accessToken, CommsStatus: type): void	Adds a communication status to the CommunicationStatus enumeration. Requires an access token for authentication to invoke the method. Changes are persisted to database.
addSpacecraftType	(String: accessToken, SpacecraftType: type): void	Adds a spacecraft type to the SpacecraftType enumeration. Requires an access token for authentication to invoke the method. Changes are persisted to database.

updateSpacecraftById	(String: accessToken, String: identifier, Spacecraft: updatedShip): void	Updates the properties of an existing spacecraft. Identifies the spacecraft to be updated by id. Requires an access token for authentication to invoke the method. Changes are persisted to database.
addGuidanceStatus	(String: accessToken, GuidanceStatus: type): void	Adds a guidance status to the GuidanceStatus enumeration. Requires an access token for authentication to invoke the method. Changes are persisted to database.

### RoboticSpacecraftMgmt, implements IRoboticSpacecraftMgmt

By implementing the IRoboticSpacecraftMgmt interface, both the Singleton pattern as well as the Façade pattern are implemented.

#### Properties

Property Name	Type	Description
listOfSpacecraft	List<Spacecraft>	List with all spacecraft registered in the system.
instance <<private>>	IRoboticSpacecraftMgmt	Singleton instance of IRoboticSpacecraftMgmt class.

#### Methods

Method Name	Signature	Description
getInstance <<synchronized, static>>	(): IRoboticSpacecraftMgmt	Returns the instance of the RoboticSpacecraftMgmt for further use. Requires an access token for authentication to invoke the method.

### Spacecraft

The Spacecraft class represents spacecraft used to mine and exploit asteroids.

#### Properties

Property Name	Type	Description
identifier	String	String that uniquely identifies a spacecraft in the system.
location	Float	Current location of the spacecraft.

destinationAsteroidID	String	Unique identifier of the destination asteroid.
statusList	List<SpacecraftStatus>	List of all statuses sent out by the spacecraft.
messageList	List<Messages>	List of all messages sent out by the spacecraft to the Mission Control Management.
launchDate	Date	Date the spacecraft was launched.
missionID	String	Identifier for the mission the spacecraft is employed in.
type	SpacecraftType	Type of the spacecraft.

### Methods

Method Name	Signature	Description
deleteSpacecraftStatus	(SpacecraftStatus: status): void	Deletes a spacecraft status from the SpacecraftStatus enumeration. Requires an access token for authentication to invoke the method. Changes are persisted to database.
addSpacecraftStatus	(SpacecraftStatus: status): void	Adds a spacecraft status to the SpacecraftStatus enumeration. Requires an access token for authentication to invoke the method. Changes are persisted to database.
sendMessage	(Message: message): void	Sends a message regarding status and/or discovery to Mission Control Management. Requires an access token for authentication to invoke the method. Changes are persisted to database.

### Message

The message class enables spacecraft to send messages about discoveries to the mission control management.

### Properties

Property Name	Type	Description
messageID	String	A unique String to identify each message.
waterDiscovery	Set<WaterDiscovery>	A set of all water discovered on the asteroid.
mineralDiscovery	Set<MineralDiscovery>	A set of all minerals discovered on the asteroid.
lifeDiscovery	Set<LifeDiscovery>	A set of all life discovered on the asteroid.

spacecraftID	String	A unique String to identify the spacecraft that sends the message.
date	Date	Date the discovery was made.

### Methods

Method Name	Signature	Description
addWaterDiscovery	(WaterDiscovery: water): void	Adds a water discovery to the exploration status. Requires an access token for authentication to invoke the method. Changes are persisted to database.
addMineralDiscovery	(MineralDiscovery: mineral): void	Adds a mineral discovery to the exploration status. Requires an access token for authentication to invoke the method. Changes are persisted to database.
addLifeDiscovery	(LifeDiscovery: life): void	Adds a life discovery to the exploration status. Requires an access token for authentication to invoke the method. Changes are persisted to database.

### SpacecraftStatus

A spacecraft status gives a picture of the current status of a spacecraft by showing its remaining fuel, its guidance status and its communication status.

### Properties

Property Name	Type	Description
identifier	String	Identifier that uniquely identifies a spacecraft.
commsStatus	CommsStatus	Current communication status of the spacecraft.
status	SpacecraftStatus	Overall status of the spacecraft.
fuelRemaining	Float	The fuel that is available to the spacecraft in percentage.
guidanceStatus	GuidanceStatus	Current guidance status of the spacecraft.

### **GuidanceStatus <<enumeration>>**

The GuidanceStatus enumeration gives an indication whether everything is working for the spacecraft (except communication link, see extra class below) or whether a problem arose.

#### **Properties**

<b>Property Name</b>	<b>Type</b>	<b>Description</b>
ALL_OK	String	Indicates that the guidance status of a spacecraft is ok.
PROBLEM	String	Indicates that the spacecraft faces a problem with regards to its guidance.

### CommsStatus <<enumeration>>

The CommStatus enumeration gives an indication whether the communication link is working for the spacecraft or whether a problem arose.

#### Properties

Property Name	Type	Description
ALL_OK	String	Indicates that the communication status of a spacecraft is ok.
PROBLEM	String	Indicates that the spacecraft faces a problem with regards to its communications.

### SpacecraftType <<enumeration>>

The SpacecraftType enumeration represents the spacecraft types available in the system and restricts the selection of a type in case of creating a new spacecraft to the types listed below.

#### Properties

Property Name	Type	Description
EXPLORER	String	Spacecraft that is used to explore an asteroid and search for resources.
MINER	String	Spacecraft that is used to mine discovered resources.

### SpacecraftStatus <<enumeration>>

This enumeration captures potential overall spacecraft statuses.

#### Properties

Property Name	Type	Description
AWAITING_LAUNCH	String	Spacecraft hasn't embarked yet to the mission.
EXPLORING	String	Spacecraft is currently exploring the asteroid.
MINING	String	Spacecraft is currently mining the asteroid.
MALFUNCTION	String	Spacecraft detected a malfunction in its system.
LOST	String	Lost contact to the spacecraft. Cannot be recovered.

CRASHED	String	Spacecraft crashed into obstacle. Needs to be recovered.
LANDED	String	Spacecraft has successfully landed on destination asteroid.
HOMEWARD_BOUND	String	Spacecraft is on the way back to the basis.

## Package: cscie97.asn5.asteroidinventorymgmt

### IAsteroidInventMgmt <<interface>>

The IAsteroidInventMgmt interface is the core of the AIMS. It contains the methods necessary for the user and systems relying on the AIMS to manage the asteroids database. By employing an interface as well as the AsteroidInventMgmt class which implements the interface, the façade design pattern is applied, as only public methods will be visible outside of the package.

### Methods

Method Name	Signature	Description
updateAsteroidByID	(String: accessToken, String: identifier, Asteroid: updatedProperties): void	Updates the properties related to the asteroid with the given unique identifier. Requires an access token for authentication to invoke the method. Changes are persisted to database.
freeSearchAsteroids	(String: accessToken, String: freeText): List<Asteroid>	Enables the user to search all fields within the asteroid database using a free text-based search. Returns a list of all matching asteroids. Requires an access token for authentication to invoke the method.
createAsteroid	(String: accessToken, Asteroid : asteroid): void	Creates a new asteroid in the database. Requires an access token for authentication to invoke the method. Changes are persisted to database.
getListAllAsteroids	(String: accessToken): List<Asteroid>	Returns a list of all asteroids listed within the database. Requires an access token for authentication to invoke the method.
getAsteroidByID	(String: accessToken, String: id): Asteroid	Returns a specific asteroid identified via its unique ID. Requires an access token for authentication to invoke the method.
addAsteroidType	(String: accessToken, AsteroidType: type): void	Adds an additional type to the AsteroidType enumeration. Requires an access token for authentication to invoke the method. Changes are persisted to database.
addLifeDiscovery	(String: accessToken, LifeDiscovery: type): void	Adds an additional type to the LifeDiscovery enumeration. Requires an



		access token for authentication to invoke the method. Changes are persisted to database.
addAccessibilityType	(String: accessToken, AccessibilityType: type): void	Adds an additional type to the AccessibilityType enumeration. Requires an access token for authentication to invoke the method. Changes are persisted to database.
addMineralType	(String: accessToken, MineralType: type): void	Adds an additional type to the MineralType enumeration. Requires an access token for authentication to invoke the method. Changes are persisted to database.

### AsteroidInventMgmt implements IAsteroidInventMgmt

By implementing the IAsteroidInventMgmt interface, both the Singleton pattern as well as the Façade pattern are implemented.

#### Properties

Property Name	Type	Description
listOfAsteroids	List<Asteroid>	List with all asteroids registered in the system.
instance <<private>>	IAsteroidInventMgmt	Singleton instance of IInventory.

#### Methods

Method Name	Signature	Description
getInstance <<synchronized, static>>	(): IAsteroidInventMgmt	Returns the instance of the AsteroidInventMgmt for further use by (outside) clients.

### Asteroid

This class describes the properties and methods associated with asteroids.

#### Properties

Property Name	Type	Description
uniqueID	String	Unique identifier to differentiate between asteroids.

explorationStatus	ExplorationStatus	Current status of the exploration, giving details about mineral, life and water discoveries on the asteroid.
gravity	Float	Gravity measured at the surface of the asteroid.
width	Float	Width of the asteroid.
height	Float	Height of the asteroid.
length	Float	Length of the asteroid.
mass	Float	Mass of the asteroid.
aphelion	Float	Point where the celestial body is farthest from the Sun
perihelion	Float	Point where the celestial body is closest from the Sun
asteroidNotes	List<AsteroidNote>	List of all notes logged for an asteroid.
asteroidType	AsteroidType	Type of the asteroid.

## Methods

Method Name	Signature	Description
addNote	(AsteroidNote: note): void	Adds a note to the asteroid. Requires an access token for authentication to invoke the method. Changes are persisted to database.
deleteNote	(AsteroidNote: note): void	Deletes a specific note from an asteroid. Requires an access token for authentication to invoke the method. Changes are persisted to database.
addExplorationStatus	(ExplorationStatus: status): void	Adds an exploration status to an asteroid. Requires an access token for authentication to invoke the method. Changes are persisted to database.

## ExplorationStatus

The ExplorationStatus class captures new discoveries made on an asteroid (minerals, water of life). They are stored in a separate database.

## Properties

Property Name	Type	Description
date	Date	Date at which the discovery took place.

waterDiscoveries	Set<WaterDiscovery>	Set of all ever-found water on an asteroids.
mineralDiscoveries	Set<MineralDiscovery>	Set of all ever-found minerals on an asteroids.
lifeDiscoveries	Set<LifeDiscovery>	Set of all ever-found life on an asteroids.

## Methods

Method Name	Signature	Description
addWaterDiscovery	(WaterDiscovery: water): void	Adds a water discovery to the exploration status. Requires an access token for authentication to invoke the method. Changes are persisted to database.
addMineralDiscovery	(MineralDiscovery: mineral): void	Adds a mineral discovery to the exploration status. Requires an access token for authentication to invoke the method. Changes are persisted to database.
addLifeDiscovery	(LifeDiscovery: life): void	Adds a life discovery to the exploration status. Requires an access token for authentication to invoke the method. Changes are persisted to database.

## AsteroidNote

An asteroid note captures free-text notes describing observations regarding an asteroid.

## Properties

Property Name	Type	Description
date	Date	Date the most recent note was written to the system.
author	String	Name of the service or person which wrote the note.
note	String	List of notes describing observations made with regards to an asteroid.

## WaterDiscovery

This class captures water discovered on an asteroid and corresponds to a database for each discovery.

## Properties

Property Name	Type	Description
type	WaterType	A mineral type that was found on the asteroid and that is listed in the WaterType enumeration class.
estQuantity	Float	Estimated quantity of water found.

accessibility	AccessibilityType	An accessibility type indicating how the minerals found can be accessed. Must take on one of the values listed in the AccessibilityType enumeration.
---------------	-------------------	--

## MineralDiscovery

This class captures minerals discovered on an asteroid and corresponds to a database for each discovery.

### Properties

Property Name	Type	Description
type	MineralType	A mineral type that was found on the asteroid and that is listed in the MineralType enumeration class.
estQuantity	Float	Estimated quantity of the minerals found in kg.
accessibility	AccessibilityType	An accessibility type indicating how the minerals found can be accessed. Must take on one of the values listed in the AccessibilityType enumeration.

## WaterType <<enumeration>>

This enumeration class captures all states in which water can be found.

### Properties

Property Name	Type	Description
ICE	String	Indicates that ice was found.
LIQUID	String	Indicates that water in liquid form was found.
GAS	String	Indicates that water in gas form was found.

## AsteroidType <<enumeration>>

This enumeration class captures all so far known asteroid types.

### Properties

Property Name	Type	Description
CTYPE	String	Indicates that asteroid is of type C.
INNERBELTTYPE	String	Indicates that asteroid is of type INNERBELT.
MTYPE	String	Indicates that asteroid is of type M.

STYPE	String	Indicates that asteroid is of type S.
-------	--------	---------------------------------------

### AccessibilityType <<enumeration>>

This enumeration class captures potential locations where resources (minerals and water) were found.

#### Properties

Property Name	Type	Description
SURFACE	String	Resources were found on the surface.
INCORE	String	Resources were found within the asteroid.
INAIR	String	Resources were found in the air.

### LifeDiscovery <<enumeration>>

This enumeration class captures life discoveries made on an asteroid.

#### Properties

Property Name	Type	Description
NONE	String	Indicating that no life was found on an asteroid.
SINGLECELL	String	Indicating that single cell organisms were found.
HOSTILE	String	Indicating that intelligent, hostile life was found.
FRIENDLY	String	Indicating that intelligent, friendly life was found.

### MineralType <<enumeration>>

This enumeration class captures potential locations where resources (minerals and water) were found.

#### Properties

Property Name	Type	Description
GOLD	String	Type for minerals identified as gold.
SILVER	String	Type for minerals identified as silver.
IRON	String	Type for minerals identified as iron.
UNKNOWN	String	Type for minerals which have not been identified so far.

**Package: cscie97.asn5.missioncontrolmgmt**

**IMissionManagement <<interface>>**

The IMissionManagement interface contains the methods necessary for the clients to monitor and manage the exploration and mining missions. By employing an interface as well as the Inventory class which implements the interface, the façade design pattern is applied, as only public methods will be visible outside of the package.

**Methods**

Method Name	Signature	Description
getListMissions	(String: accessToken): List<Mission>	Returns a list with all missions in the system, regardless of current status. Requires an access token for authentication to invoke the method.
getMissionById	(String: accessToken, String: identifier): Mission	Returns a specific mission identified by its ID. Requires an access token for authentication to invoke the method.
updateMissionById	(String: accessToken, String: identifier, Mission: update): void	Updates a mission (properties, status, and the like) based on the new parameters given. Requires an access token for authentication to invoke the method. Changes are persisted to database.
addMission	(String: accessToken, Mission: mission): void	Adds a mission to the system. Requires an access token for authentication to invoke the method. Changes are persisted to database.
addMissionStatus	(String: accessToken, MissionStatus: type): void	Adds a new status to the MissionStatus

		enumeration. Requires an access token for authentication to invoke the method. Changes are persisted to database.
updateDiscoveries	(cscie97.asn5.roboticspacecraftmgmt.Message: message): void	Updates the discoveries made on an asteroid based on a message sent by one of the exploring spacecraft. Requires an access token for authentication to invoke the method. Changes are persisted to database.

### **MissionManagement, implements IMissionManagement**

By implementing the IMissionManagement interface, both the Singleton pattern as well as the Façade pattern are implemented.

#### **Properties**

Property Name	Type	Description
listAllMissions	List<Mission>	Lists all concrete missions associated with Mission Management
inventory	Inventory	Instance of the inventory associated with Mission Control Management.
instance <<private>>	IMissionManagement	Instance of MissionManagement which ensures Singleton pattern.

#### **Methods**

Method Name	Signature	Description
getInstance <<static, synchronized>>	(): IMissionManagement	Returns the instance of the Mission Management for further use. Requires an access token for authentication to invoke the method.

### **IIventory <<interface>>**

The IInventory interface is another integral part of the MMS. It contains the methods necessary for the user and systems relying on the MMS to monitor and manage the overall inventory of the base. By employing an interface as well as the Inventory class which implements the interface, the façade design pattern is applied, as only public methods will be visible outside of the package.

#### Methods

Method Name	Signature	Description
employSpacecraft	(List<String>: spacecraftIDs): void	Enables MMS to employ spacecraft to a mission. It decreases the inventory by the given spacecraft IDs. Requires an access token for authentication to invoke the method. Changes are persisted to database.
layOffSpacecraft	(String: identifier): void	Method used by MMS to lay off a spacecraft from a mission and return it to the inventory. Requires an access token for authentication to invoke the method. Changes are persisted to database.

#### Inventory, implements IInventory

By implementing the IInventory interface, both the Singleton pattern as well as the Façade pattern are implemented.

#### Properties

Property Name	Type	Description
remainingBudget	Float	Remaining total budget available to mission control for new missions, etc.
availableSpacecraft	Integer	Total number of spacecraft which have not been deployed and are ready in the inventory
remainingFuel	Float	Total amount of fuel left in the inventory
instance <<private>>	IInventory	Singleton instance of IInventory.

#### Methods

Method Name	Signature	Description
getInstance <<static, synchronized>>	(): IInventory	Returns the instance of the Inventory for further use. Requires an access token for authentication to invoke the method.

#### ICommsLink <<interface>>



The ICommsLink interface is the core of the MMS. It contains the methods necessary for the user and systems relying on the MMS to monitor the communication link to Earth as well as to receive and send messages. By employing an interface as well as the CommsLink class which implements the interface, the façade design pattern is applied, as only public methods will be visible outside of the package.

#### Methods

Method Name	Signature	Description
sendMessage	(MissionMessage: message): void	Enables the Mission Management to send a message via the communication link. Requires an access token for authentication to invoke the method.
receiveMessage	(cscie97.asn5.roboticspacecraftmgmt.Message: message): void	Allows Mission Management to receive messages from spacecraft. Requires an access token for authentication to invoke the method.
checkCommsStatus	(): Boolean	Check whether the communication link is working. Requires an access token for authentication to invoke the method.

#### CommsLink, implements ICommsLink

By implementing the ICommsLink interface, both the Singleton pattern as well as the Façade pattern are implemented.

#### Properties

Property Name	Type	Description
activeCommsLink	Boolean	Returns true if current link is active, otherwise returns false.
allMessages	List<cscie97.asn5.roboticspacecraftmgmt.Message>	Lists all messages received by MMS.

Instance <<private>>	ICommsLink	Singleton instance of Inventory.
----------------------	------------	----------------------------------

## Methods

Method Name	Signature	Description
getInstance <<synchronized, static>>	(): ICommsLink	Returns the instance of the CommsLink for further use. Requires an access token for authentication to invoke the method.

## Mission

The mission class describes mining and exploration expeditions.

## Properties

Property Name	Type	Description
identifier	String	Uniquely identifies a mission.
name	String	Name of a mission.
launchDate	Date	Date the mission was launched.
estimatedArrival	Date	Date the arrival on the destination asteroid is estimated.
destinationID	String	Unique identifier of the destination asteroid.
status	MissionStatus	Status of the mission, selected from the MissionStatus enumeration.
purpose	String	Purpose of a mission (exploration, mining, ...).
associatedSpacecraft	List<String>	List of spacecraft identifiers which are associated with a mission.
budget	Float	Total budget needed for the mission.

## Methods

Method Name	Signature	Description
updateMission	(String: identifier, Mission: updatedMission): void	Updates a mission based on the new parameters given. Requires an access token for authentication to invoke the method. Changes are persisted to database.

## MissionMessage

The message class enables mission management to send messages to other bases, including Earth.

### Properties

Property Name	Type	Description
messageID	String	A unique String to identify each message.
date	Date	Date the discovery was made.
message	String	Free-text send to within the message body.
recipient	String	Recipient the message is addressed to.

### Methods

Method Name	Signature	Description
createNewMessage	(Date: date, messageID: identifier, String: message): void	Creates a new message with the specified parameters and sends it to specified recipient. Requires an access token for authentication to invoke the method. Changes are persisted to database.

### MissionStatus <<enumeration>>

This enumeration class is comprised of values that the mission status can attain.

### Properties

Property Name	Type	Description
INPIPELINE	String	Mission has not been started yet.
INPROGRESS	String	Mission is currently being carried out.
CANCELLED	String	Mission was cancelled.
COMPLETED	String	Mission has been completed.

## Implementation Details

In addition to the packages and classes described above, the system will employ a MySQL database to keep track of resources, missions and spacecraft and to implement the requirement for a persisted database. Any time an entity is added, deleted or otherwise updated, the changes will be written to the database immediately within that method.

The following components will be modelled in separate database tables:

- Enumerations
- Asteroids (primary key: identifier)
- AsteroidNotes
- ExplorationStatus
- MineralDeposit

- WaterDeposit
- Spacecraft
- SpacecraftStatus
- Message
- Mission
- MissionNotes
- MissionMessage

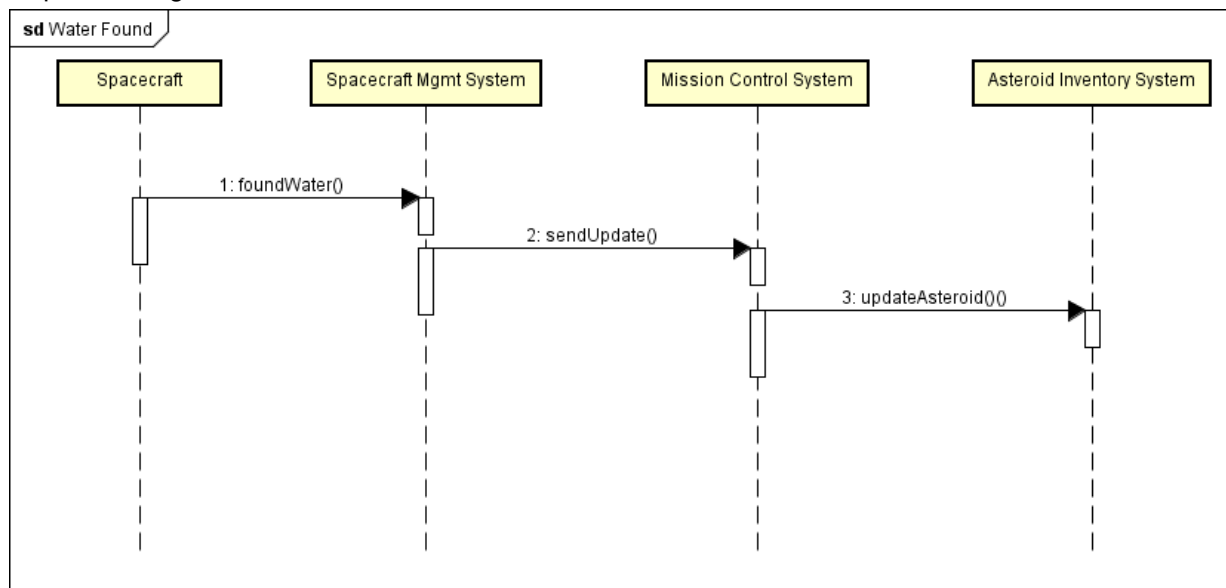
The unique identifiers for all entities will be UUIDs.

As mentioned above, the two design patterns used were the Singleton and the Façade patterns. For the first one, IAsteroidInventory, IRoboticSpacecraftMgmt, IInventory and IMissionMgmt have been modelled and implemented using AsteroidInventory, RoboticSpacecraftMgmt, Inventory and MissionMgmt to adhere to the Façade pattern. By using this structure, only the interfaces of the Singletons are being displayed to outside clients. Both patterns ensure that Service APIs are being provided, while it ensures a loosely coupled, inherently consistent architecture. The latter requirement is furthermore supported by implementing duplicate classes across packages – for example, the miningDiscoveries or waterDiscoveries classes, which are present in both the Robotic Spacecraft Management System package and the Asteroid Inventory Management package.

## Sequence Diagrams

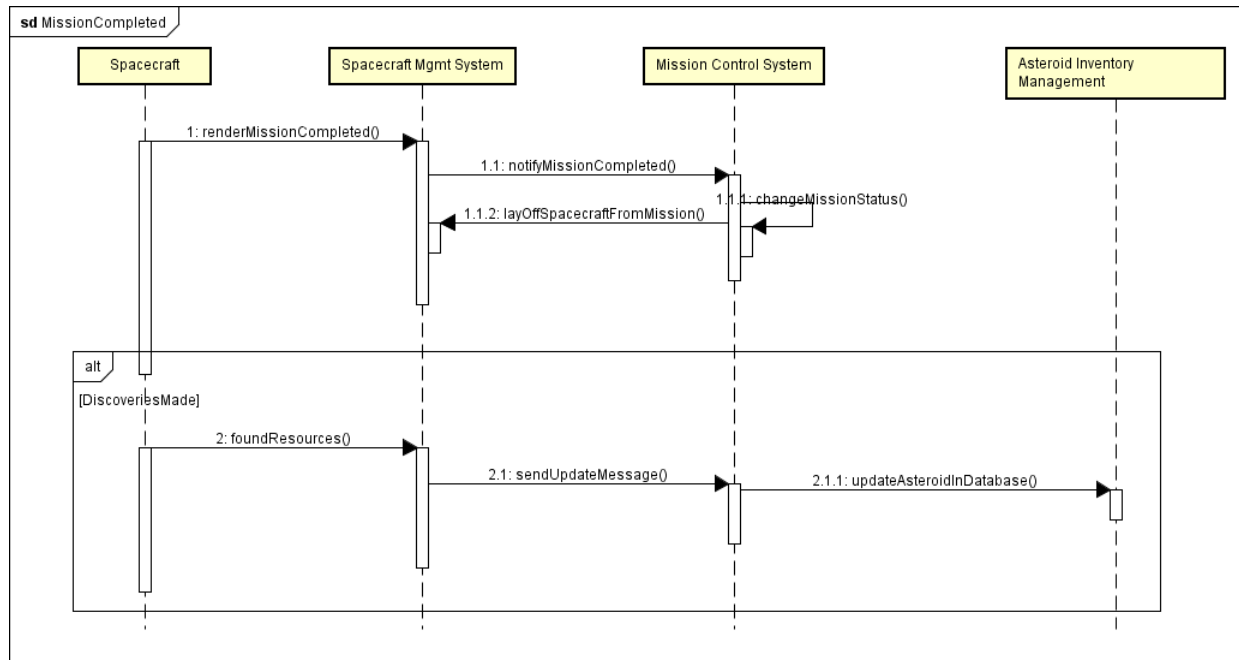
The following Sequence Diagrams model the logical process of water being found by an asteroid, a mission being completed and a new mission being started. They should help to understand the underlying processes and the interactions across the subsystems.

Sequence Diagram 1: Water Found



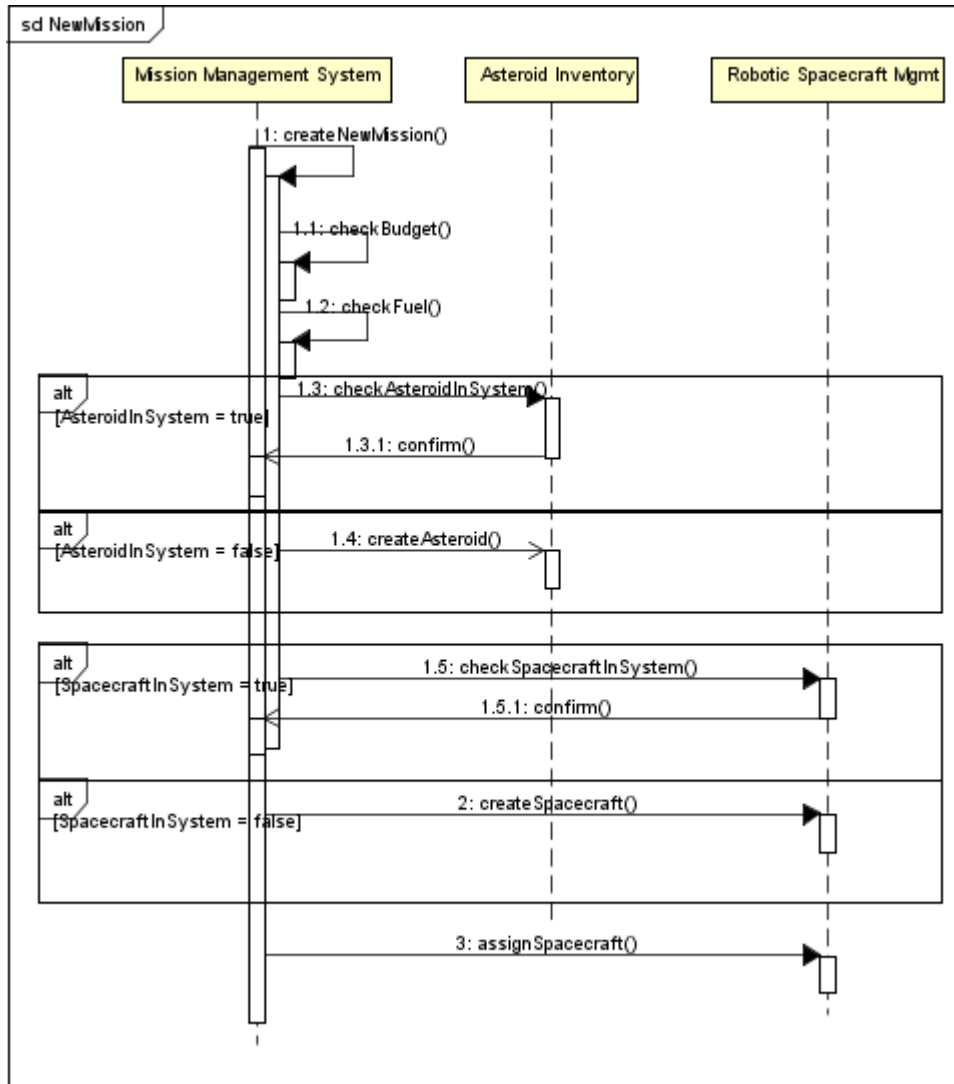
A spacecraft finds water on an asteroid. This information is passed on to the Spacecraft Management System, which in turn sends an update to the Mission Control System. The Mission Control System receives and parses the message and updates the Asteroid Inventory System accordingly.

Sequence Diagram 2: Mission Completed



This Sequence Diagram shows what happens if a mission is completed. First, the spacecraft logs that it completed its mission and returned to the basis. In case the spacecraft has made discoveries which have not yet been reported, it reports them and the process described in Sequence Diagram 1 is started. The Spacecraft Management System then send a mission completed notification to the Mission Control System. The Mission Control System first lays off the spacecraft(s) which were associated with that mission and makes them available for further missions. It also changes the status of the mission in its own system to “COMPLETED”.

Sequence Diagram 3: New Mission



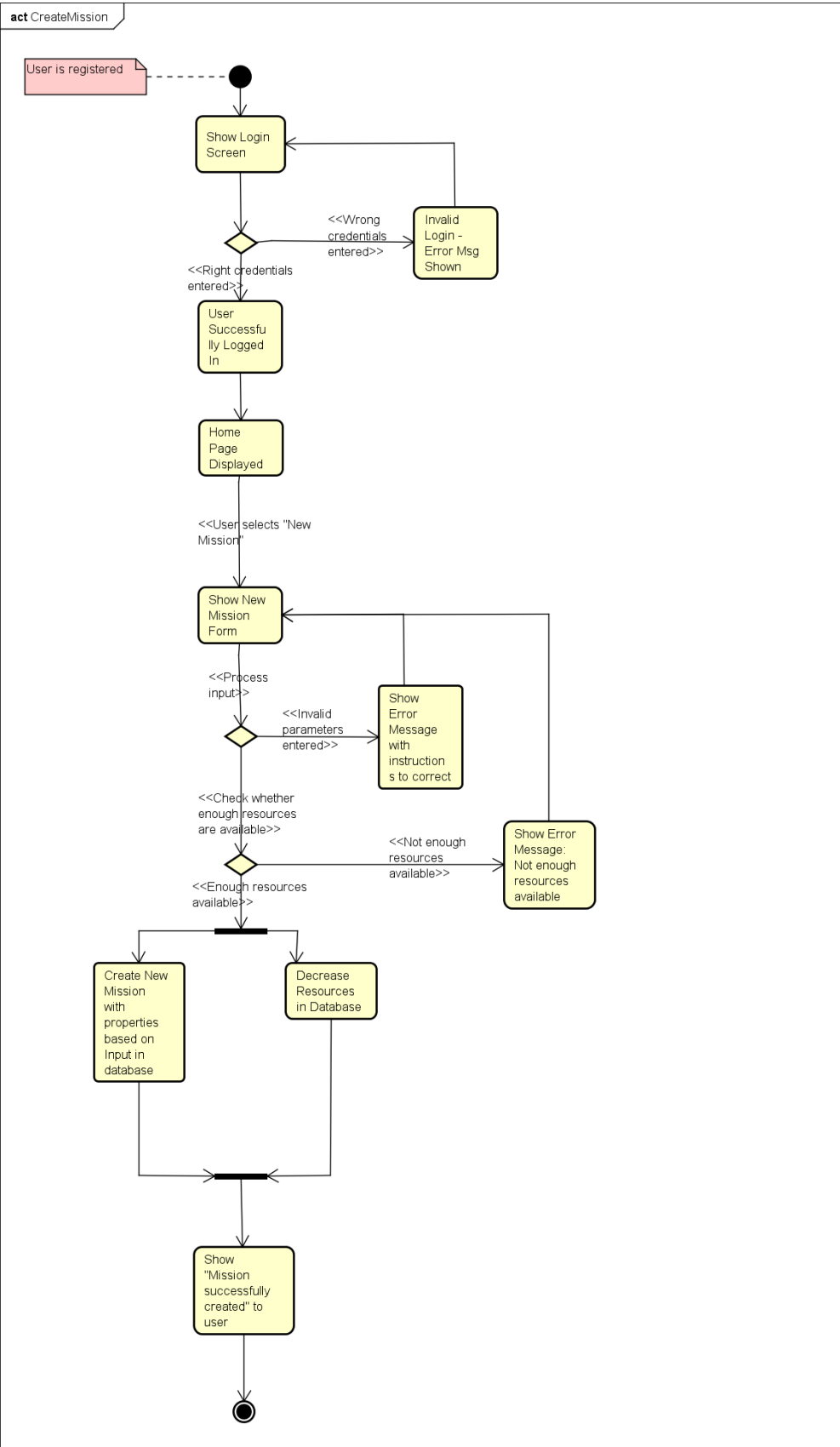
This sequence diagram describes which processes are being invoked when a new mission is started. Once the user filled in the form using the GUI to create a new mission, the Mission Management checks first whether enough budget and fuel are available. If enough resources are available, it checks whether the destination asteroid is in the Asteroid Inventory Management. If it is not, a new entry is created. The Mission Management then checks whether the spacecraft are in the system and available. If they haven't been registered yet, Mission Management updates the Robotic Spacecraft Management. It then employs the spacecraft on the mission and updates the status in the Robotic Spacecraft Management.

## Activity Diagram

The following Activity Diagram documents the provisioning of a new mission. The process is conditional of being invoked by a registered user.

First, the user is supposed to login to the system. Upon starting the CCUI, the login screen is shown and the user is prompted to enter his/her credentials. If the entered credentials are incorrect, an error

message is shown and the user is asked to enter valid credentials. If the credential entered are correct, the user is logged in to the system and is directed to the landing page. There, he/she can select "Create New Mission". This takes the user to an interactive form where all mission details (name, budget, etc.) can be entered. Once the user sends off the form, the system checks whether the entered data is valid, i.e. whether a mission with that name does exist already, whether the entered parameters are in a valid range, etc. If errors are detected, the user is shown an error message specifying the errors. If all information is correct, the system then proceeds to check with the inventory whether enough resources are available to start the mission. If this is not the case, the user is shown an error message indicating that not enough resources are available. If enough resources are available, the Mission Control Management System will create a new mission with the specified parameters in the mission database and at the same time reduce the available resources in its inventory by the amount needed for the mission. Eventually, the user is shown a notification stating that the mission has been created successfully.





## Command and Control User Interface

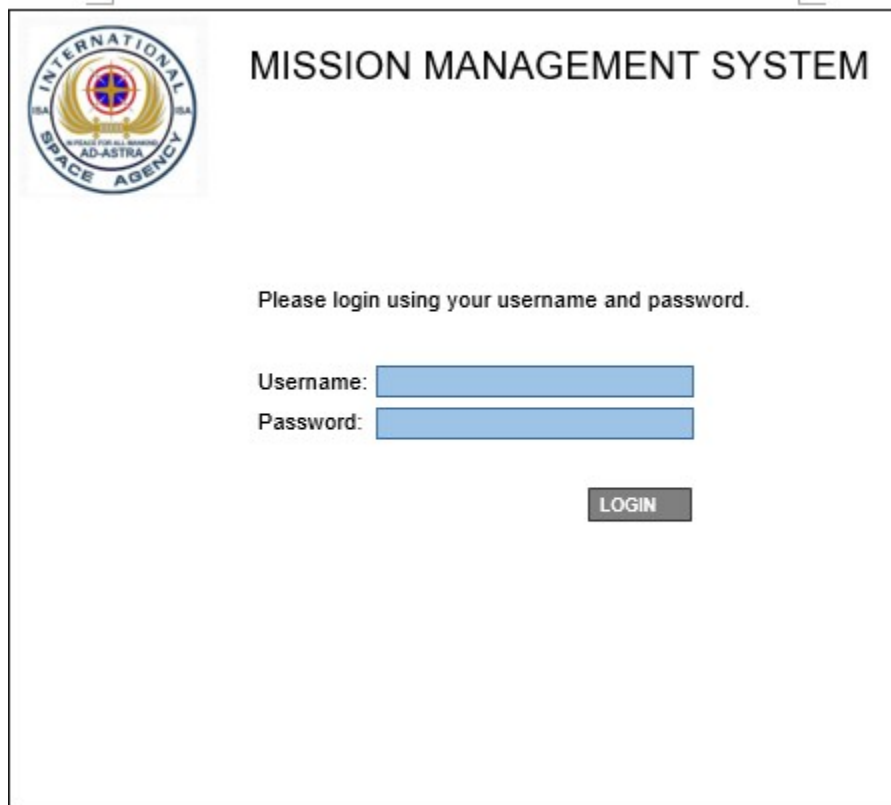
The Command and Control User Interface is directed towards the user and enables an intuitive use of the Service APIs provided by the subsystems used within the AES. The following screens are wireframes that should be used as template to model the user interface. Changes may be made at the discretion of the programmer.

The functionalities supported by the CCUI should encompass:

- 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 CCUI should furthermore support use on mobile devices.

CCUI Template 1: Login Screen



The wireframe shows a login screen for the 'MISSION MANAGEMENT SYSTEM'. In the top left corner is the logo of the 'INTERNATIONAL SPACE AGENCY' (ISA), which features a globe with a blue and red crosshair and the text 'ISA' and 'AD-ASTRA'. The title 'MISSION MANAGEMENT SYSTEM' is displayed in a large, bold, sans-serif font at the top center. Below the title, a prompt reads 'Please login using your username and password.' followed by two input fields: 'Username:' and 'Password:'. Each label is followed by a light blue rectangular input box. At the bottom center, there is a dark grey rectangular button with the word 'LOGIN' in white, uppercase letters.

CCUI Template 2: Landing Page



## MISSION MANAGEMENT SYSTEM

### MENU

RESOURCES

ASTEROIDS

SPACECRAFT

MISSIONS

COMMUNICATION

AUTHENTICATION

YOUR PROFILE


LOGOUT

Success!

You are logged in as **Ann-Katrin Reuel**. You are an **administrator**.

### Frequently Used Tasks:

- 1) [Create a New Mission](#)
- 2) [Query Resources](#)
- 3) [Access Authentication Service](#)
- 4) [View Communications](#)



# MISSION MANAGEMENT SYSTEM

MENU

RESOURCES

ASTEROIDS

SPACECRAFT

MISSIONS

COMMUNICATION

AUTHENTICATION

YOUR PROFILE

LOGOUT

## ASTEROIDS DATABASE

Please select one of the following options:

List all known asteroids

Search Asteroid by ID:

SEARCH

Search Asteroids by Text:

SEARCH

CCUID Template 4: Add New Mission



# MISSION MANAGEMENT SYSTEM

## MENU

RESOURCES

**ASTERIODS**

SPACECRAFT

MISSIONS

COMMUNICATION

AUTHENTICATION

YOUR PROFILE

LOGOUT

## Create a new Mission

Please enter the following details to create a new mission:

~~MissionID:~~

Name:

Purpose:

Spacecraft:

Launch Date:

Estimated Time of Arrival:

Destination:

Status:

CREATE

## Exception Handling

The following exceptions could show up during the use of the Asteroid Exploration System. Each one will be implemented as exception class within the package `cscie97.asn5.exceptions`:

- **Authentication Exception:** User tries to login with wrong password
- **Resource Exception:** User tries to create a new mission without enough resources being available
- **Not Found Exception:** User tries to update an entity (spacecraft, mission, asteroid, ...) that is not registered in the system
- **Duplicate Exception:** Enter new entity (mission, asteroid, spacecraft, ...) with ID that is already in the system

## Testing

The test strategy will be comprised of a functional, a performance and exception handling test. The functional test will cover the main functionalities of the system architecture. The following cases will be tested:

- Create new mission
- Simulate incoming message from a spacecraft which found life on an asteroid and ensure that database is being updated
- Login as both basic and admin user
- Searching the asteroid database for a specific asteroid
- Returning a list of all spacecraft available

Performance testing will take place by accessing the system with many users at the same time, as well as simulating many incoming messages with updates from the spacecraft fleet.

Exception handling tests will include the following scenarios:

- Trying to login with wrong password
- Trying to query for a non-existent asteroid
- Trying to update a non-existent mission
- Trying to create a mission without having enough resources at hand

Testing will be carried out using the following simulated command structures:

### Asteroid Commands

```
# create_asteroid, <asteroid_identifer>, <asteroid_type>, <width>,
<length>, <height>,
<mass>, <gravity>, <aphelion>, <perihelion>
# add_asteroid_note, <asteroid_identifer>, <date>, <author>, <text>
# add_asteroid_mineral_discovery, <asteroid_identifer>, <mineral>,
<estimated_mass>, <deposit_type>

# add_asteroid_water_discovery, <asteroid_identifer>, <water_found>,
<amount>,
```

```
<state>
# add_asteroid_life_discovery, <asteroid_identifer>, <life_type>,
<intelligent>, <friendly>
```

## Spacecraft Commands

```
# create_spacecraft, <spacecraft_identifier>, <launch_date>,
<mission_identifier>,
<spacecraft_type>, <fuel_level>, <guidance_status>,
<communication_status>, <state>,
<location>, <target_asteroid_identifier>
```

## Mission Management Commands

```
# create_mission, <mission_identifier>, <mission_name>,
<mission_purpose>,
<spacecraft_identifier>, <launch_date>, <eta>,
<target_asteroid_identifier>,
<mission_state>
# increment_fuel_resource <amount>
# increment_spacecraft_resource <amount>
# increment_operating_budge <amount>
# set_communicaiton_link_status <status>
# set_automated_control_system_status <status>
```

## Simulating Spacecraft Messages Commands

```
# create_spacecraft_status_message, <spacecraft_identifier>,
<timestamp>,
<mission_identifier>, <spacecraft_type>, <fuel_level>,
<guidance_status>,
<communication_status>, <state>, <location>,
<target_asteroid_identifier>
# create_spacecraft_fault_alert_message, <spacecraft_identifier>,
<timestamp>,
<failure_type>
# create_spacecraft_mineral_discovery_message, <spacecraft_identifier>,
<timestamp>, <asteroid_identifer>, <mineral>, <estimated_mass>,
<deposit_type>
# create_spacecraft_water_discovery_message, <spacecraft_identifier>,
<timestamp>,
<asteroid_identifer>, <water_found>, <amount>, <state>
# create_spacecraft_life_discovery_message, <spacecraft_identifier>,
<timestamp>,
<asteroid_identifer>, <life_type>, <intelligent>, <friendly>
```

## Risks

A potential risk is that the Authentication Service turns out to be a bottleneck in the operation of the entire system since it needs to be accessed every time a method is invoked to ensure that the respective authentication is valid. If multiple services and users access it at the same time, this might become an issue. To ensure that the systems are working even when the put-through is high, performance testing under high access loads is carried out.

Another risk is that databases might be accessed by two systems or users at the same time, which might lead to inconsistencies. To ensure that all changes are being recorded, incremental saves are done as well as the ACID principles for databases are being applied.

Generally, due to the high complexity that comes with multiple subsystems working together, regular sanity checks with regards to database consistency and general functionality should be carried out to catch exceptions or errors not listed in this document as early as possible.