

Asteroid Exploration System Design Document

CSCI E-97 Assignment 5

Asteroid Exploration System Design Document

Date: 12/15/2013

Author: Siddharth SINGH

Reviewer(s): Hemant BAJPAI

Introduction

This document describes the design of asteroid exploration system. The document gives overview of the problem and requirements and discusses use cases with help of use case diagram. The component diagram explains the individual components involved in implementing the asteroid exploration system.

The UML class diagram shows the classes, associations and methods for components or modules of the system. The document shows class diagrams for Asteroid inventory system, Spacecraft management system and Mission Management System. The document explains responsibilities these individual components as well as relationship between components.

Class dictionary sections details out methods, properties and associations of classes used in the implementation for the three components mentioned above. The class dictionary section gives detail about how classes are associated and implemented.

Sequence Diagram section shows and explains sequence of events and message flows that occur in the work flow for the following tasks

- Spacecraft discovers water on an asteroid
- Spacecraft has completed a mission
- Creating a new mission

Activity diagram section shows and explains activities to create a new mission.

This design document also explains the design details and design patterns used to meet the requirements in the implementation details section.

The document shows user interface and their workflow.

The document explains how the modules and components can be tested and the risks involved.

Overview

Asteroid exploration system manages fleet of spacecraft sent to explore asteroids in the space. Each spacecraft has an associated mission and has an asteroid as a target destination. Spacecraft explores the target asteroid and communicates with ground based communication links to notify about their findings on asteroid. Spacecraft tries to find life, mineral or water on asteroids and notifies the mission control center using ground based communication link. Spacecraft updates mission control center about its status and state. Mission control center manages mission, asteroid inventory and spacecraft. Upon receiving messages from spacecraft mission control takes relevant actions in response to the message. E.g. the status and state of space craft and mission are updated accordingly. If spacecraft makes any discovery on any asteroid then the asteroid is updated accordingly.

There is a command and a user interface which sends commands to mission management and mission management notifies user interface if any change has happened in the asteroid exploration system. Following summarizes interaction between command/user interface and mission management.

- Login/Logout.
- Create missions.
- Monitoring and updating status of asteroids, spacecraft and missions.
- Monitoring mission resources (fuel, budget etc.).
- Monitoring status of ground based communication links
- Monitoring messages from the spacecraft.

Use Cases

There are mainly two types of users: robotic space craft, technical operator and mission control manager.

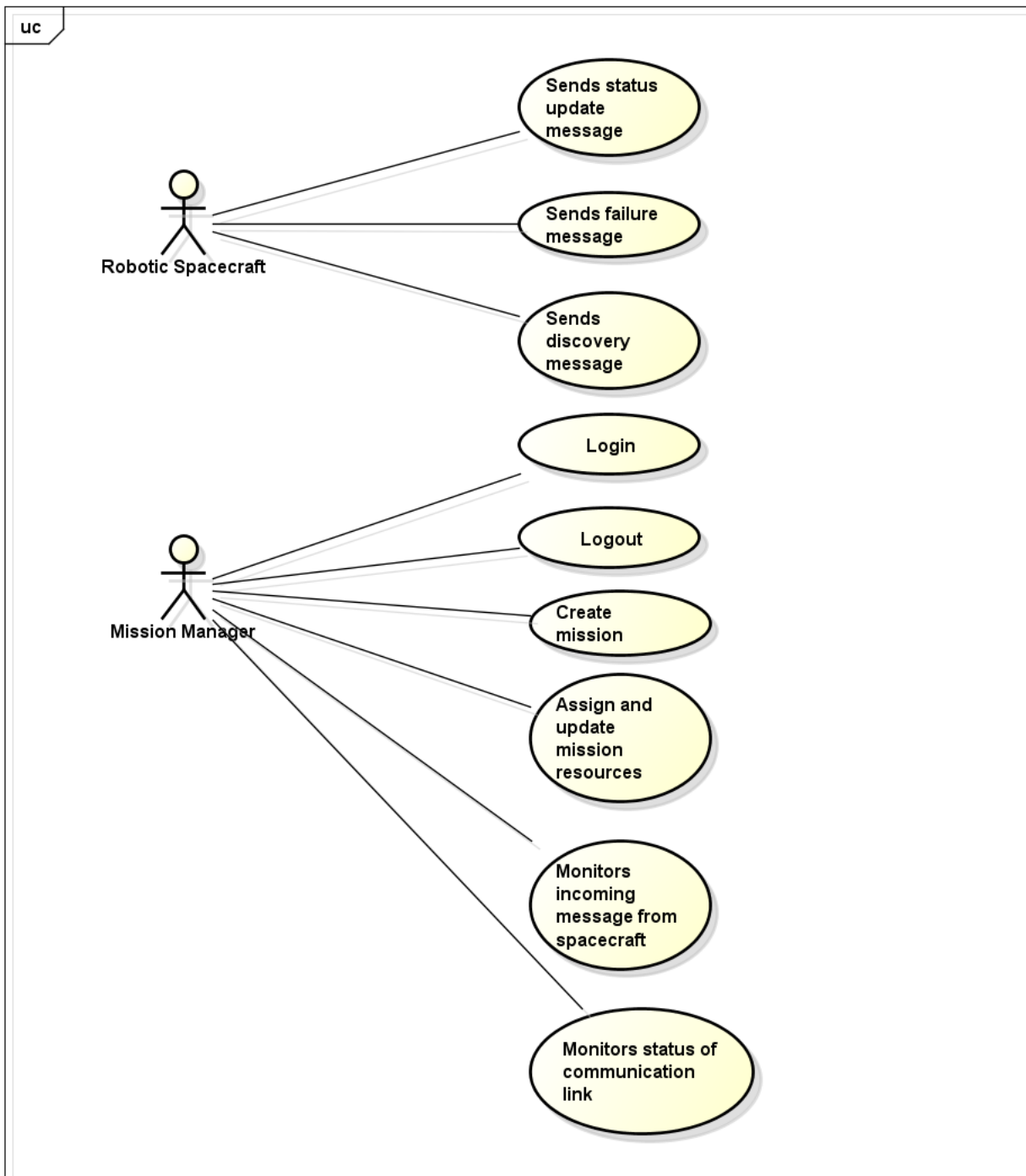
Robotic spacecraft manager sends messages to ground based communication link. Through messages spacecraft notifies mission control about following things:

- Status update: Spacecraft notifies about changes in fuel level, current state, location, guidance and target asteroid. The mission control center updates spacecraft's status accordingly.
- Spacecraft Failure: Spacecraft notifies about failure (e.g. crashed) and mission control center updates spacecraft and mission accordingly.
- Discovery: Spacecraft notifies about discovery of life, water or minerals on asteroid and mission control center updates asteroid accordingly.

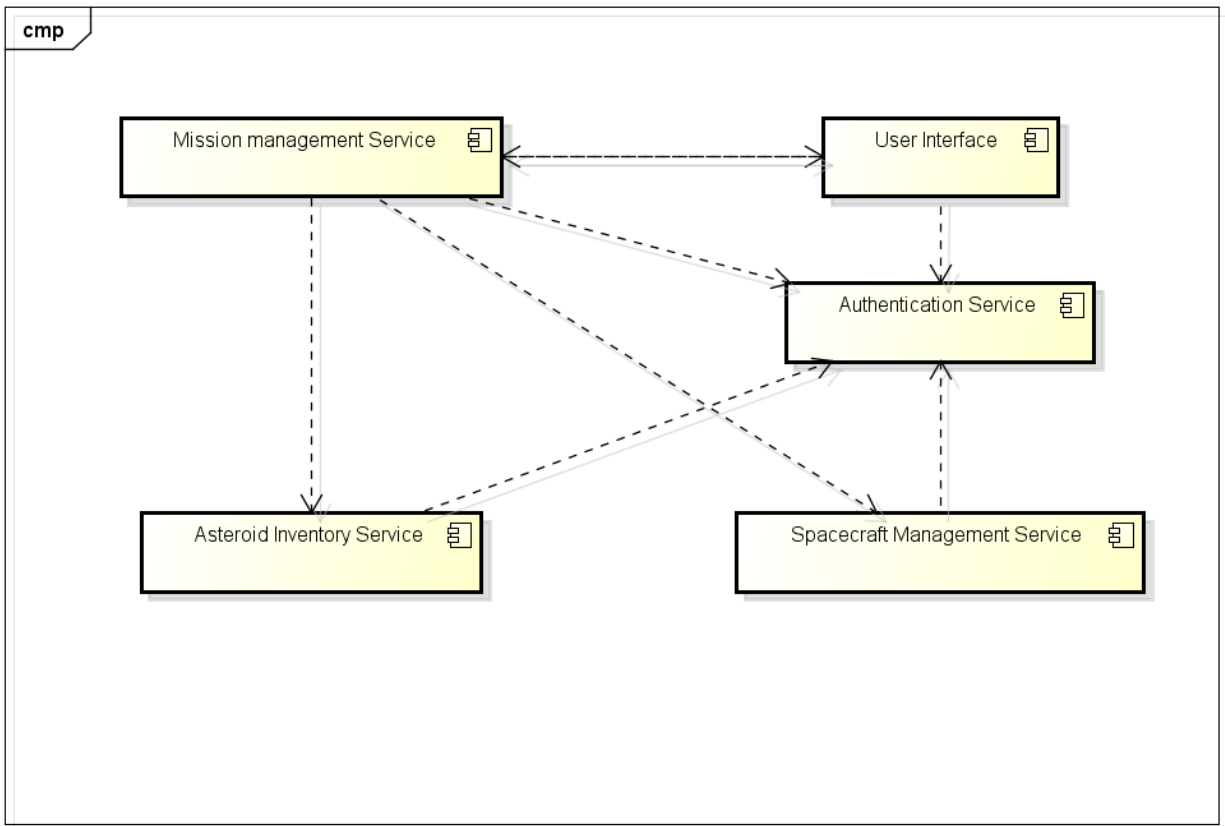
Mission managers perform following tasks:

- Create mission
- Updates spacecraft status (e.g. fuel, guidance, state, location, target asteroid).
- Monitors and assigns resources to mission and spacecraft.
- Monitors incoming message from spacecraft.

- Monitors communication link with ground based communication system and with spacecraft.



Components



powered by Astah

Asteroids exploration consists of five components.

- Asteroid inventory service.
- Spacecraft management service.
- Mission management service
- Authentication service
- User interface

Asteroid inventory service and Spacecraft management service are independent components and manages Asteroids and spacecraft. Asteroid inventory service creates updates and manages lifecycle of an asteroid. Similarly spacecraft management service manages spacecraft. Both these services have methods with restricted access and authentication service ensures that access to those methods is authorized. So both asteroid and spacecraft services depend only on authentication service.

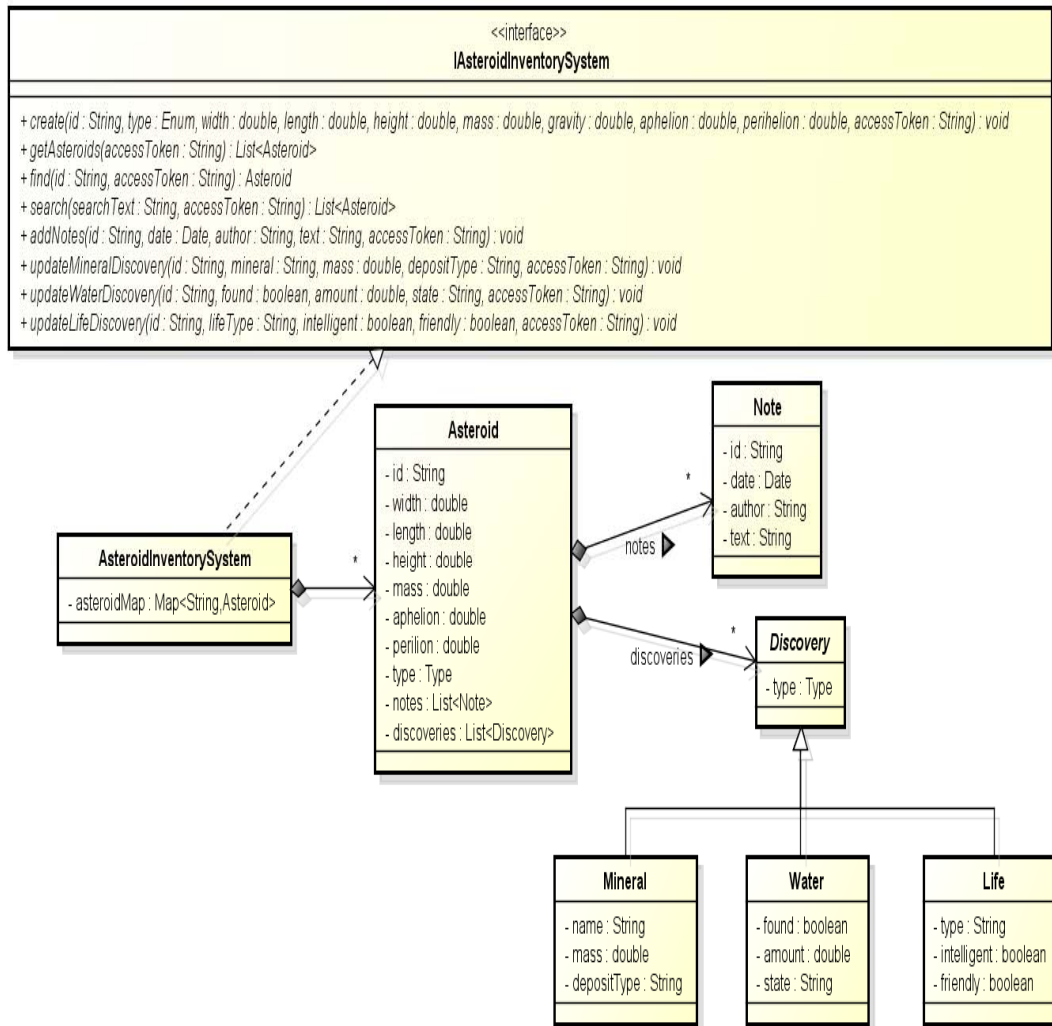
Mission management service acts as mediator between all the components in the system. It receives message from spacecraft and take relevant actions like updating missions, asteroids and spacecraft. It

manages missions and calls asteroids and spacecraft service APIs in order to make changes in them. Mission management notifies user interface about changes. It has restricted methods for which it uses authentication service to make sure that access is authorized. Mission management system depends on asteroid service, spacecraft management service, user interface and authentication service.

User interface observes changes in mission management system so it registers itself with mission management system to be its observer. User interface depends on mission management system.

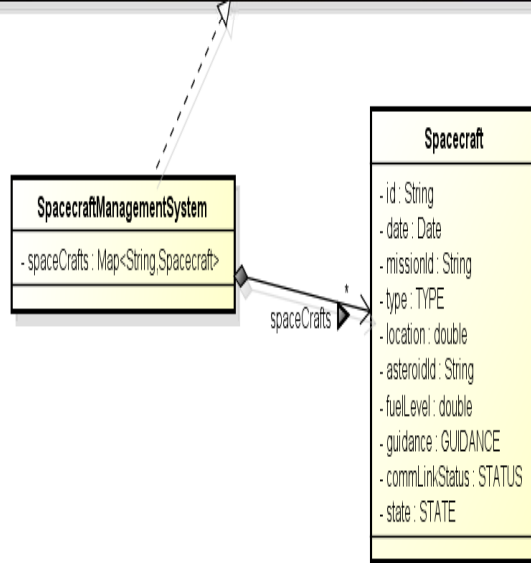
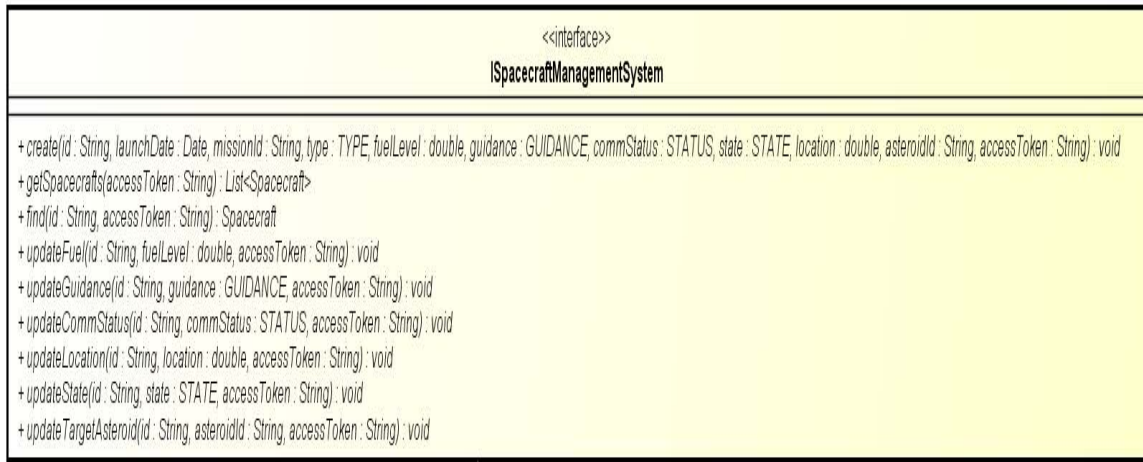
Class Diagram

Asteroid Inventory Service

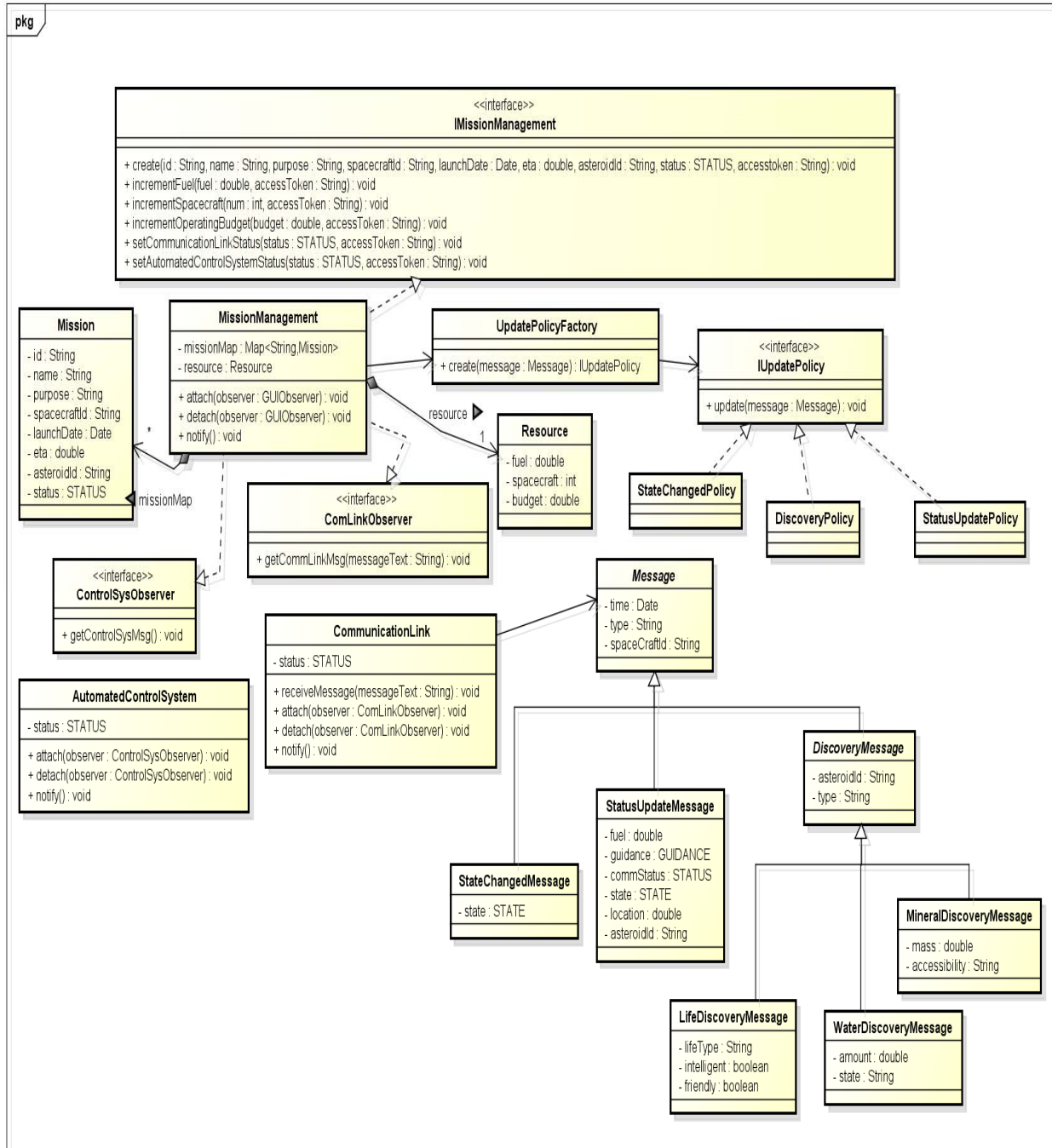


Spacecraft Management Service

pkg



Mission management Service



powered by Astah

Class Dictionary

Asteroid Inventory Service

IAsteroidInventorySystem<<interface>>

Methods

Method Name	Signature	Description
create	create(id : String, type : Enum, width : double, length : double, height : double, mass : double, gravity : double, aphelion : double, perihelion : double, accessToken : String) : void	Creates a new asteroid with given id and other asteroid parameters passed as arguments of the function.
getAsteroids	getAsteroids(accessToken : String) : List<Asteroid>	Returns all the asteroids in the asteroid inventory.
find	find(id : String, accessToken : String) : Asteroid	Returns Asteroid with given id. Search is case insensitive.
search	search(searchText : String, accessToken : String) : List<Asteroid>	Searches asteroids in asteroid inventory with searchText appearing in notes.
addNotes	addNotes(id : String, date : Date, author : String, text : String, accessToken : String) : void	Add notes the asteroid with given id.
updateMineralDiscovery	updateMineralDiscovery(id : String, mineral : String, mass : double, depositType : String, accessToken : String) : void	Adds mineral discovery to asteroid with given id and mineral parameters.
updateWaterDiscovery	updateWaterDiscovery(id : String, found : boolean, amount : double, state : String, accessToken : String) : void	Adds water discovery to asteroid with given id and water parameters.
updateLifeDiscovery	updateLifeDiscovery(id : String, lifeType : String, intelligent : boolean, friendly : boolean, accessToken : String) : void	Adds water discovery to asteroid with given id and mineral parameters.

AsteroidInventorySystem

This class implements IAsteroidInventoryService interface. It keeps map of asteroid id and asteroid objects. Key of the map is stored in upper case to make search case insensitive.

Associations

Association Name	Type	Description
asteroidMap	Map<String,Asteroid>	map of asteroid id and asteroid objects in the inventory.

Asteroid

This class represents asteroid objects. The data member of this class are the asteroid parameters.

The objects of this class is persistent and id is the key for persistence.

Associations

Association Name	Type	Description
notes	List<Note>	List of all the notes associated with an asteroid. Persistent : Yes
discoveries	List<Discovery>	List of all the discoveries on an asteroid. A discovery can be a life, water or mineral. Persistent : Yes

Properties

Property Name	Type	Description
id	String	Access token of admin user. Persistent : Yes
width	double	Asteroid width. Persistent : Yes
length	double	Asteroid length. Persistent : Yes
height	double	Asteroid height. Persistent : Yes
mass	double	Asteroid mass. Persistent : Yes
aphelion	double	Asteroid closest distance from sun. Persistent : Yes
perilion	double	Asteroid longest distance from sun. Persistent : Yes
type	Enum type	Type of Asteroid (C-Type,M-Type,S-Type, Innerbelt Comet) Persistent : Yes

Note

This class represents note association of an asteroid.

The objects of this class is persistent and id is the key for persistence.

Properties

Property Name	Type	Description
id	String	Note id. Persistent : Yes
date	Date	Note date. Persistent : Yes
author	String	Note author. Persistent : Yes
text	String	Text of a note Persistent : Yes

Discovery<<abstract>>

The abstract class represents discovery on an asteroid. It is sub classed by concrete classes of life, water and mineral.

The objects of this class is persistent and id is the key for persistence.

Properties

Property Name	Type	Description
id	String	Discovery identifier. Persistent : Yes
type	TYPE	Type of discovery (Life, Water or Mineral.) Persistent : Yes

Mineral

Inherits discovery and represents mineral discovery on asteroid.

Properties

Property Name	Type	Description
name	String	Name of mineral.
mass	double	Mineral property.
depositType	String	Mineral property.

Water

Inherits discovery and represents water discovery on asteroid.

Properties

Property Name	Type	Description
found	Boolean	Water found or not.
amount	double	Amount of water found.
state	String	State (e.g. ice)

Life

Inherits discovery and represents life discovery on asteroid.

Properties

Property Name	Type	Description
type	TYPE	(None, single cell, multi cell etc)
intelligent	Boolean	Intelligent or not.
friendly	Boolean	Friendly or not.

Spacecraft Management Service

ISpacecraftManagementSystem<<interface>>

Methods

Method Name	Signature	Description
create	create(id : String, launchDate : Date, missionId : String, type : TYPE, fuelLevel : double, guidance : GUIDANCE, commStatus : STATUS, state : STATE, location : double, asteroidId : String, accessToken : String) : void	Creates a new spacecraft object with given id and other spacecraft parameters passed as arguments of the function.
getSpacecrafts	getSpacecrafts(accessToken : String) : List<Spacecraft>	Get all spacecraft in spacecraft management system.
find	find(id : String, accessToken : String) : Spacecraft	Find spacecraft with given id, returns Null if no spacecraft is found with given id. Search is case insensitive.
updateFuel	updateFuel(id : String, fuelLevel : double, accessToken : String) : void	Updates spacecraft's fuel level and sets it to fuelLevel whose id is given.
updateGuidance	updateGuidance(id : String, guidance : GUIDANCE, accessToken : String) : void	Updates spacecraft's guidance state and sets it to guidance whose id is given.
updateLocation	updateLocation(id : String, location : double, accessToken : String) : void	Updates spacecraft's location and sets it to location whose id is given.

	double, accessToken : String) : void	to location whose id is given.
updateState	updateState(id : String, state : STATE, accessToken : String) : void	Updates spacecraft's state whose id is given.
updateTargetAsteroid	updateTargetAsteroid(id : String, asteroidId : String, accessToken : String) : void	Updates target asteroid to the asteroid whose id is asteroidId of an asteroid whose id is given.
updateCommStatus	updateCommStatus(id : String, commStatus : STATUS, accessToken : String) : void	Updates communication status of spacecraft with ground based communication link to commStatus.

SpacecraftManagementSystem

This class implements ISpacecraftManagementSystem interface. It keeps map of spacecraft id and spacecraft objects. Key of the map is stored in upper case to make search case insensitive.

Associations

Association Name	Type	Description
spaceCrafts	Map<String,Spacecraft>	map of spacecraft id and spacecraft objects in the spacecraft management system.

Spacecraft

This class represents proxy of spacecraft objects. The data member of this class are the spacecraft properties.

The objects of this class is persistent and id is the key for persistence.

Properties

Property Name	Type	Description
id	String	Spacecraft id Persistent : Yes
date	Date	Launch date of spacecraft. Persistent : Yes
missionId	String	Id of mission with which it is associated. Persistent : Yes
type	TYPE	Enum TYPE (explorer, miner). Persistent : Yes
location	double	AUs from sun. Persistent : Yes
asteroidId	String	Id of target asteroid. Persistent : Yes
fuelLevel	double	Fuel(% remaining). Persistent : Yes

commLinkStatus	STATUS	OK or NOT OK Persistent : Yes
state	STATE	Enum STATE (waiting for launch, int route, lost, crashed etc). Persistent : Yes
guidance	GUIDANCE	OK or NOT OK Persistent : Yes

Mission Management Service

IMissionManagement<<interface>>

Methods

Method Name	Signature	Description
create	create(id : String, type : Enum, width : double, length : double, height : double, mass : double, gravity : double, aphelion : double, perihelion : double, accessToken : String) : void	Create mission with given id and mission parameters.
incrementFuel	incrementFuel(fuel : double, accessToken : String) : void	Updates fuel level. The argument fuel can be negative in which case fuel available with mission service gets decremented.
incrementSpacecraft	incrementSpacecraft(num : int, accessToken : String) : void	Updates number of spacecraft available to mission service. The argument num can be negative in which case numbers of spacecraft will decrement.
incrementOperatingBudget	incrementOperatingBudget(budget : double, accessToken : String) : void	Update budget of mission, budget can be negative in which case total budget will go down.
setCommunicationLinkStatus	setCommunicationLinkStatus(status : STATUS, accessToken : String) : void	Sets status of ground based communication links.
setAutomatedControlSystemStatus	setAutomatedControlSystemStatus(status : STATUS, accessToken : String) : void	Sets status of automated control system.

MissionManagement

This class implements IMissionManagement interface. It keeps map of mission ids and mission objects. Key of the map is stored in upper case to make search case insensitive. It also composes resource available to mission management. This class acts as mediator between different componets of asteroid exploration system. Receives messages from communication links and updates mission, asteroid and spacecraft depending upon the message contents. It manages missions itself and to update asteroid it calls asteroid inventory service APIs and to update spacecraft it calls spacecraft management APIs.

It acts as subject to GUI in context of observer pattern and notifies GUI about any changes that occur in the system.

Methods

Method Name	Signature	Description
attach	attach(observer : GUIObserver) : void	Registers GUI observer in order to notify it about any changes.
detach	detach(observer : GUIObserver) : void	Deregisters in case observer does not want to observe.
notify	notify() : void	Notify all observers about the changes occurred in the system.

Associations

Association Name	Type	Description
missionMap	Map<String,Mission>	map of mission id and mission objects in the mission management system.
resource	Resource	Resources available to mission management (fuel, budget etc).

Mission

This class represents mission objects in the system. It defines all the properties of a mission.

The objects of this class is persistent and id is the key for persistence.

Properties

Property Name	Type	Description
id	String	Mission id. Persistent : Yes

name	String	Mission name. Persistent : Yes
purpose	String	Purpose mission. Persistent : Yes
spacecraftId	String	Associated spacecraft. Persistent : Yes
lauchDate	Date	Launch date of spacecraft. Persistent : Yes
eta	double	Estimated time of arrival. Persistent : Yes
asteroidId	String	Associated asteroid. Persistent : Yes
status	STATUS	(waiting for launch, in progress, complete, aborted) Persistent : Yes

Resource

This class represents resource objects available to mission management. Mission management has certain resources to allocate to every mission. This class defines all those resources.

Properties

Property Name	Type	Description
fuel	double	% remaining.
spacecraft	int	Number of sapcecraft.
budget	double	Budget available.

CommunicationLink

This class acts as proxy for ground based communication link. It acts as subject in context of observer pattern and notifies mission management when it receives message from spacecraft or status of link changes.

Methods

Method Name	Signature	Description
attach	attach(observer : ComLinkObserver) : void	Registers observer in order to notify it about any changes.
detach	detach(observer : ComLinkObserver) : void	Deregisters in case any observer does not want to get notified.
notify	notify() : void	Notify all observers when it receives a message or status of link changes .

Properties

Property Name	Type	Description
status	STATUS	OK or NOT OK.

AutomatedControlSystem

This class acts as proxy for ground based automated control system. It acts as subject in context of observer pattern and notifies mission management when it's status changes.

Methods

Method Name	Signature	Description
attach	attach(observer : ControlSysObserver) : void	Registers observer in order to notify it about any changes.
detach	detach(observer : ControlSysObserver) : void	Deregisters in case any observer does not want to get notified.
notify	notify() : void	Notify all observers when it's status changes.

Properties

Property Name	Type	Description
status	STATUS	OK or NOT OK.

ComLinkObserver<<interface>>

Interface which declares a method for observing communication link. Mission management implements this function.

Methods

Method Name	Signature	Description
getCommLinkMsg	getCommLinkMsg(messageText : String) : void	Communication link calls this function in notify() to notify all the observers.

ControlSysObserver<<interface>>

Interface which declares a method for observing automated control system. Mission management implements this function.

Methods

Method Name	Signature	Description
-------------	-----------	-------------

getControlSysMsg	getControlSysMsg() : void	Automated control system calls this function in notify() to notify all the observers.
------------------	---------------------------	---

Message<<abstract>>

Abstract class to represent message between spacecraft and communication link.

Properties

Property Name	Type	Description
time	Date	Time at which message sent.
type	String	Message type.
spaceCraftId	String	Spacecraft id which sends message.

StateChangedMessage

Concrete message that notifies spacecraft about change in state.

Properties

Property Name	Type	Description
state	STATE	Current state of spacecraft.

StatusUpdateMessage

Concrete message that notifies spacecraft about change in status.

Properties

Property Name	Type	Description
location	double	AUs from sun.
asteroidId	String	Id of target asteroid.
fuelLevel	double	Fuel(% remaining).
commLinkStatus	STATUS	OK or NOT OK
state	STATE	Enum STATE (waiting for launch, int route, lost, crashed etc).
guidance	GUIDANCE	OK or NOT OK

DiscoveryMessage<<abstract>>

Abstract class to represent message if spacecraft has made a discovery.

Properties

Property Name	Type	Description
asteroidId	String	Id of target asteroid.
type	String	Message type.

DiscoveryMessage

Concrete discovery message class which represents discovery of life on an asteroid.

Properties

Property Name	Type	Description
type	TYPE	(None, single cell, multi cell etc)
intelligent	Boolean	Intelligent or not.
friendly	Boolean	Friendly or not.

WaterDiscoveryMessage

This concrete discovery message class represents discovery of water on an asteroid.

Properties

Property Name	Type	Description
amount	double	Amount of water found.
state	String	State (e.g. ice)

MineralDiscoveryMessage

This concrete discovery message class represents discovery of mineral on an asteroid.

Properties

Property Name	Type	Description
accessibility	String	e.g. surface deposit.
mass	double	Mass of mineral found.

UpdatePolicyFactory

The update of mission, asteroid and spacecraft is implemented using strategy pattern or policy based design. Depending on the message type an object of corresponding policy is created. And we have three kinds of update policies: Discovery, state changed and status update. This class as a factory and creates corresponding policy object based on the message passed to create() function.

Methods

Method Name	Signature	Description
create	create(message : Message) : IUpdatePolicy	Based on the message it creates and returns corresponding policy object.

IUpdatePolicy<<interface>>

This Interface which represents update policy class. Each concrete class will implement this interface and apply the update rules in definition of the function. E.g. Discovery policy will update the asteroid and state change policy will update mission and spacecraft about change in state.

Methods

Method Name	Signature	Description
update	update(message : Message) : void	Apply update policy depending on message type and content.

StateChangedPolicy

Implements IUpdatePolicy and implements business logic when state of space craft changes. E.g. It will update spacecraft and mission.

DiscoveryPolicy

Implements IUpdatePolicy and implements business logic when spacecraft makes a discovery on an asteroid. It will update the asteroid.

StatusUpdatePolicy

Implements IUpdatePolicy and implements business logic when status of spacecraft changes. It will update the spacecraft.

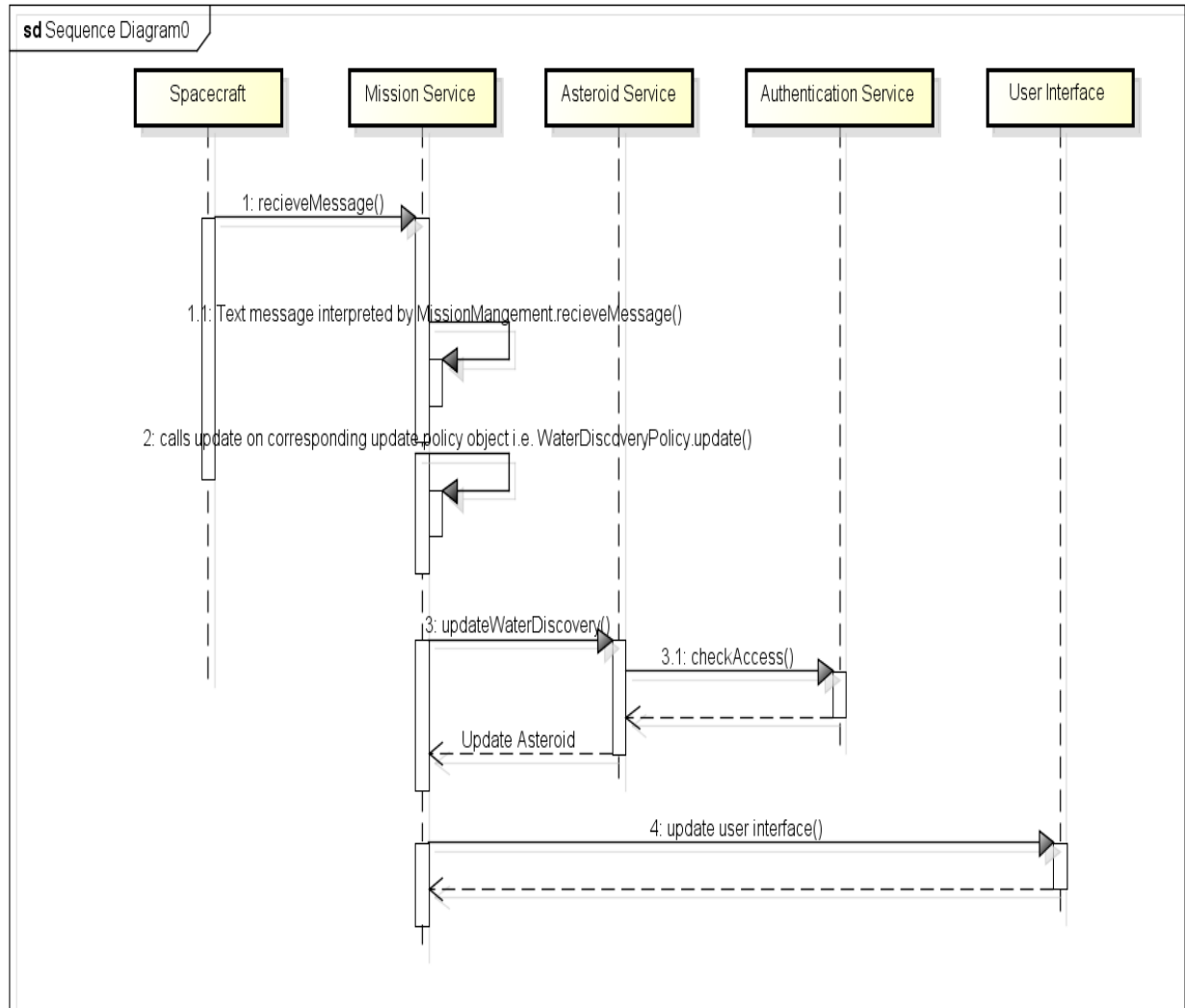
Sequence Diagrams

Discovery of Water:

Spacecraft sends message to communication link. Communication links notifies mission control and passes the message to mission control service. Mission control service interprets the message and creates water discovery message type. Mission management service sends message to update policy factory which returns object of discovery policy class. The update() method of discovery policy

implements the business logic of water discovery, i.e. it updates the asteroid in asteroid inventory. Asteroid inventory checks access with authentication service and if authorized updates the asteroid.

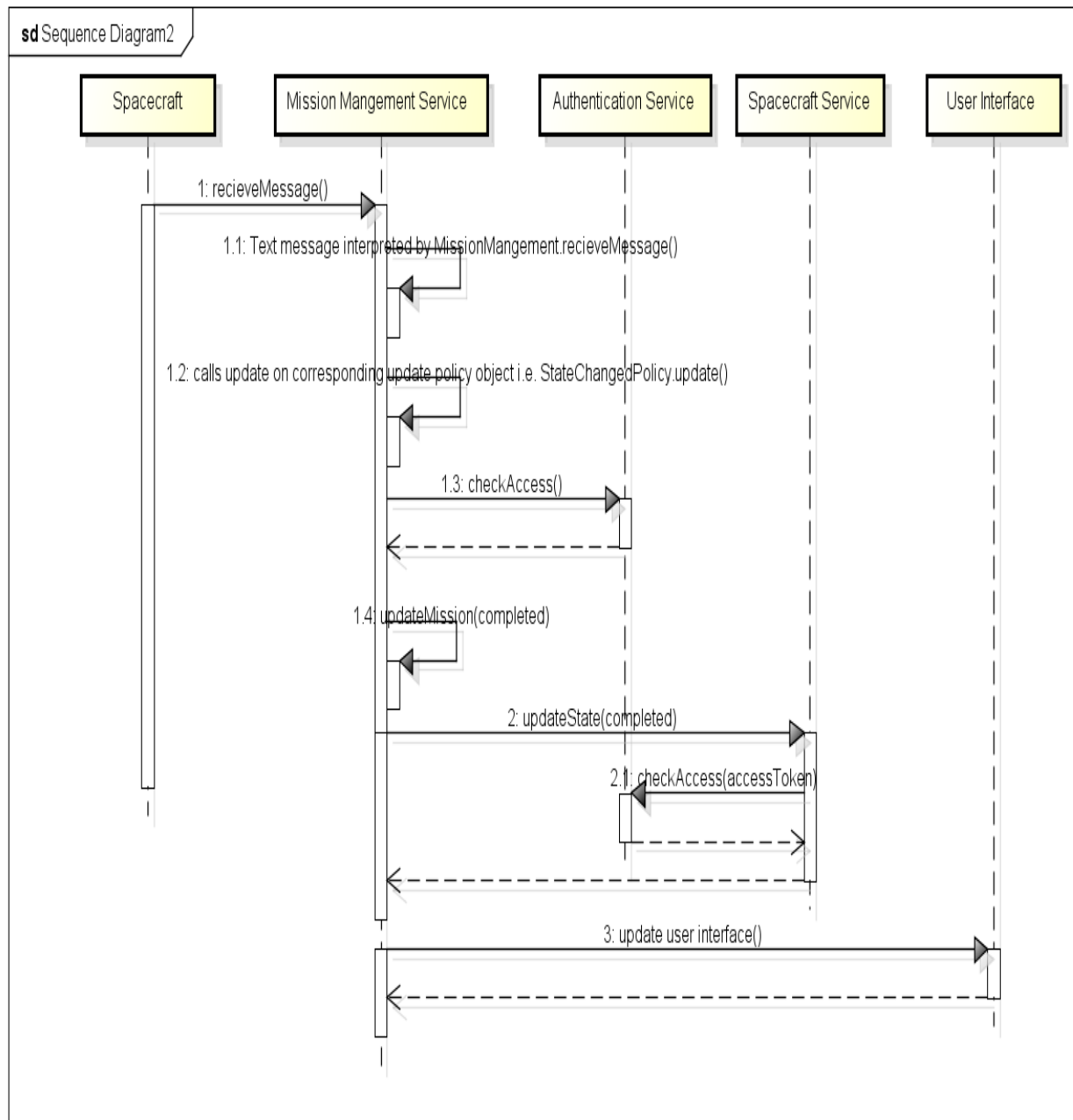
Mission management also notifies user interface about the changes happened.



powered by Astah

Mission completion by spacecraft:

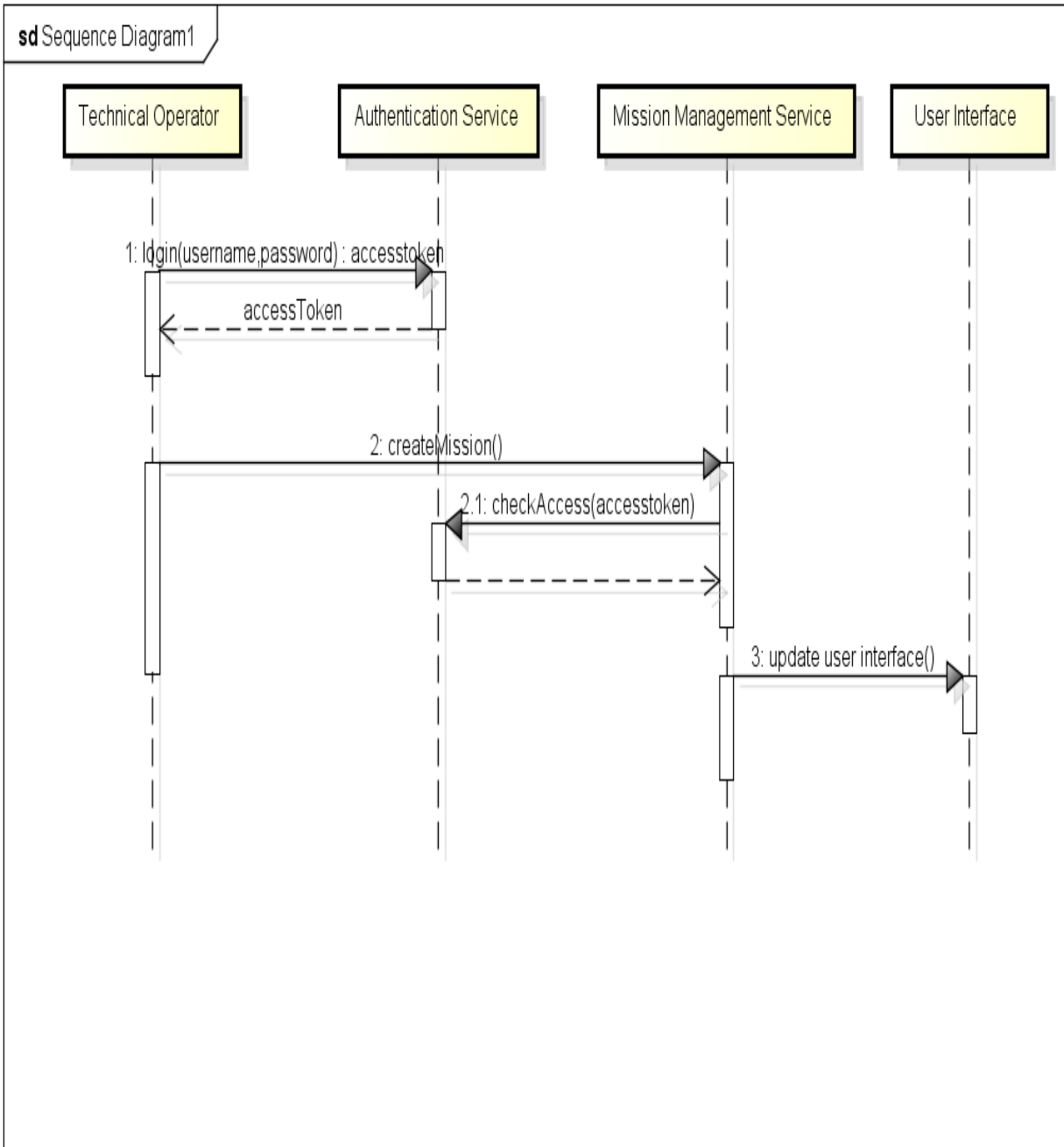
The workflow is similar to what described above for water discovery except that status change policy implementation updates spacecraft in spacecraft management system and mission management updates the mission's state.



powered by Astah

Create new mission:

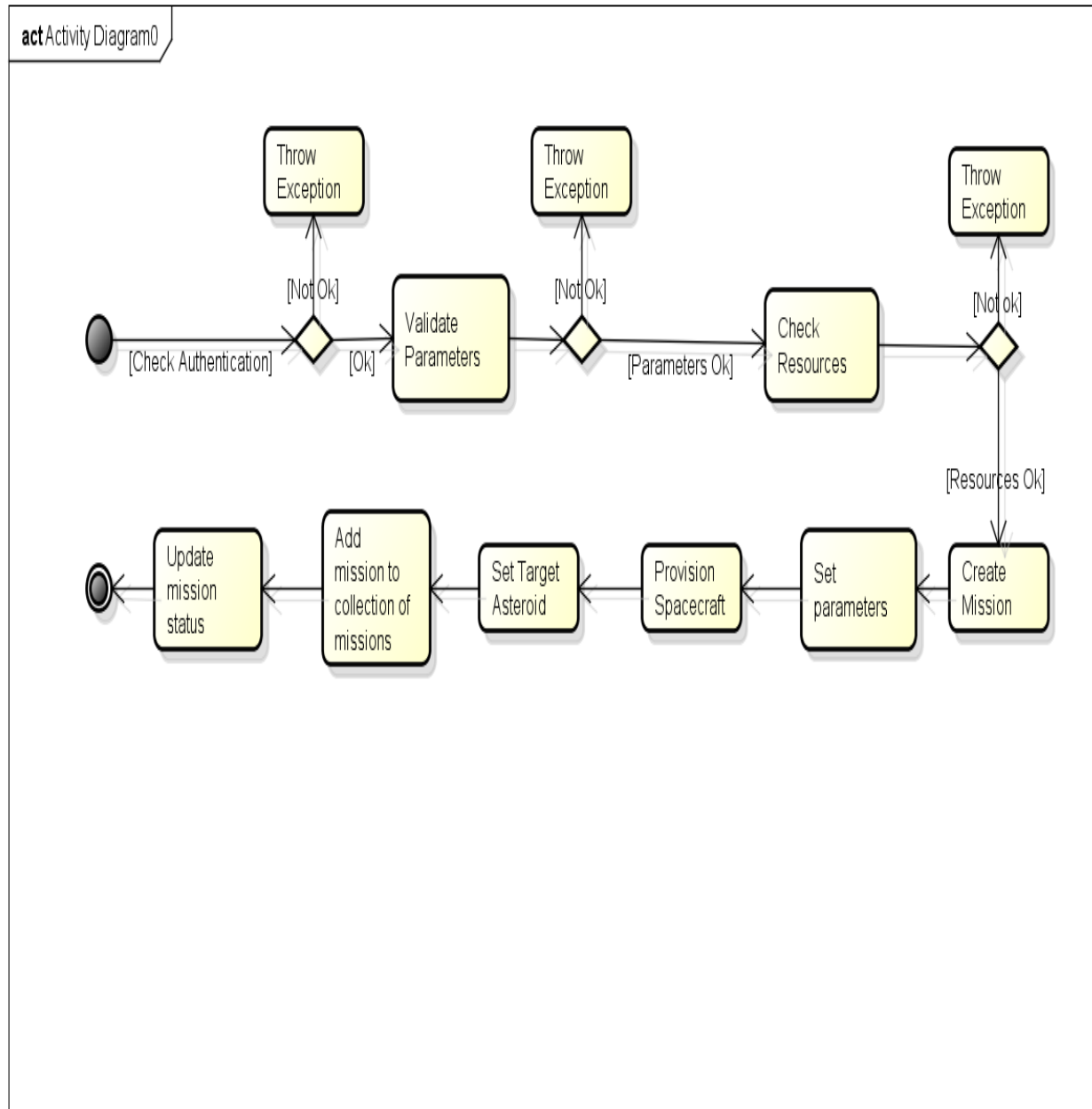
Technical operator logs into the system. Authentication service authorizes by verifying username/password. User calls mission management through command interface to create a new mission and passing the mission parameters. Mission management checks access with authentication service and on success it creates a new mission in mission management system.



powered by Astah

Activity Diagram for creating a new mission

Please see sequence diagram section about the workflow of creating a new mission. The following figure shows the activity diagram for creating a new mission.



Implementation Details

The Asteroid exploration system is divided into three components.

Each system exposes interface which allows abstraction in such a way that system can be accessed only through interface and implementation details are hidden. Each system exposes APIs which other components can use and interface defines the contract between the component and rest of the world. The simplified interfaces follow the **facade** pattern.

The three systems are:

1. Asteroid inventory system.
2. Spacecraft management system.
3. Mission management system.

The first two systems are independent and do not depend on any other component. Mission management system is in between and delegates messages between components. Mission management follows the **mediator** pattern.

Asteroid inventory system manages the collection of asteroids. It keeps map of asteroid ids and asteroid objects and provides interface to create and update individual asteroids. This component is stand alone and can be reused in any other component. All the APIs in this system have restricted access. It uses authentication service to verify access. So the only dependency for this component is Authentication service.

Spacecraft management system is very similar to asteroid management system. It provides APIs to create and update spacecraft and its parameters. The APIs have restricted access so similar to asteroid system it also uses authentication service to verify access. Since this component is independent (except dependency on authentication service) It can be reused in any other system.

Mission management system is heart of the implementation. It receives messages and communicates with different components of the system. It acts as **mediator** between other modules in the system. Mission management receives message from communication link in text format. Communication link receives message first. Mission management is registered with communication link as one of its observers. When communication link receives a message it notifies mission management and passes the text message. This is an implementation of **observer** pattern. The Text message has been converted from binary data received by communication radio center. The protocol is **XML**. So communication link receives message in xml format and then notifies mission management. Mission management parses the message and interprets it. Mission management categorizes message into three types : status update, state change and discovery. The abstract class Message is derived by subclasses and each represent one of the three message types described above. Mission management interprets the message and creates a corresponding message object. Each message corresponds to different type of action that's needs to be taken. Each message will bring different kind of changes in the system. E.g. discovery message will update asteroid and status change will update both spacecraft and mission. In order to implement business logic for each message type I have defined policy classes. These policy classes are derived from IUpdatePolicy interface. The interface has update() function which is implemented by policy subclasses. Each subclass knows what business logic they need to implement.

This follows **policy based** design or **strategy** pattern. Mission management needs to create corresponding update policy object at runtime depending on the message type and content. The policy object is created by policy **factory** class and create() function of the **factory** takes message as the argument and then depending upon the message it can create corresponding update policy object. Mission management then calls update() of the policy object to make necessary changes in the system.

Similar to communication link, mission management has registered itself with automated control system which notifies mission management about status changes.

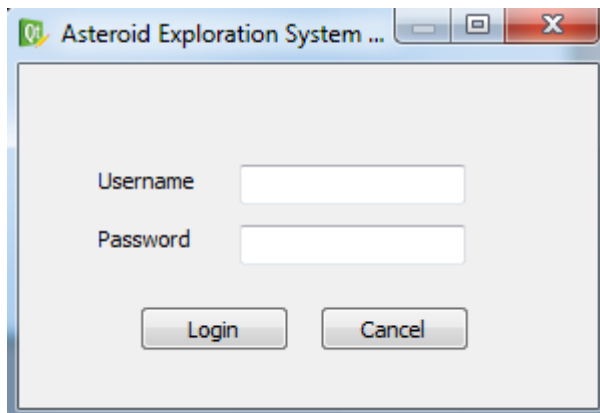
User interface observes mission management. Mission management notifies user interface about any changes occurred in the system. This is also implemented using **observer** pattern.

Each service implementation is a **singleton** object which can be accessed by factory class for the interface.

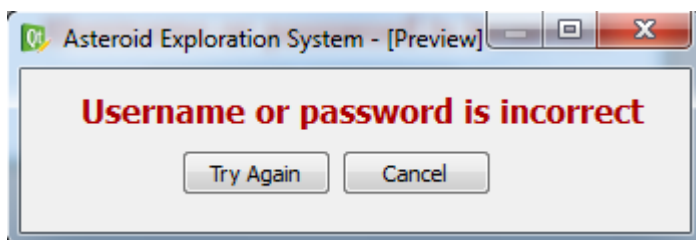
User interface

The following diagrams shows user interface for asteroid exploration System.

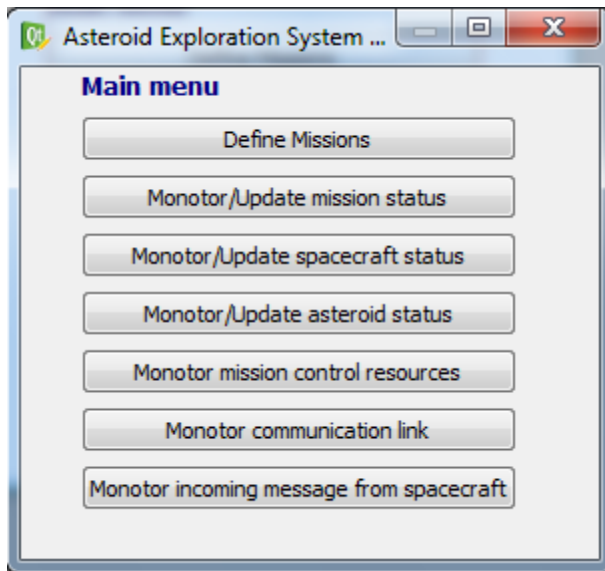
Login Screen: It calls authentication service login() method.



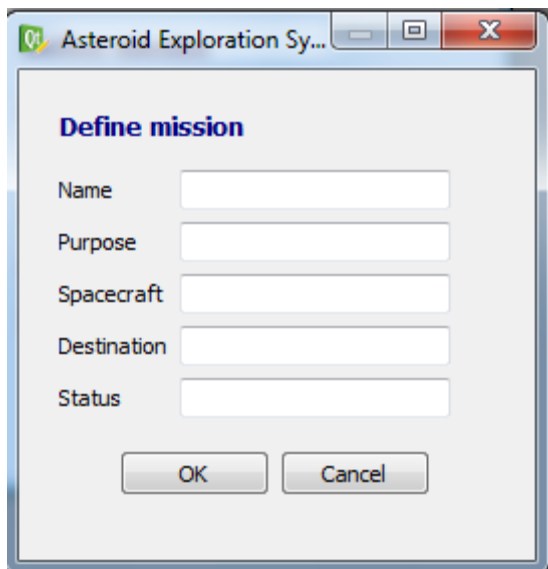
If Login fails:



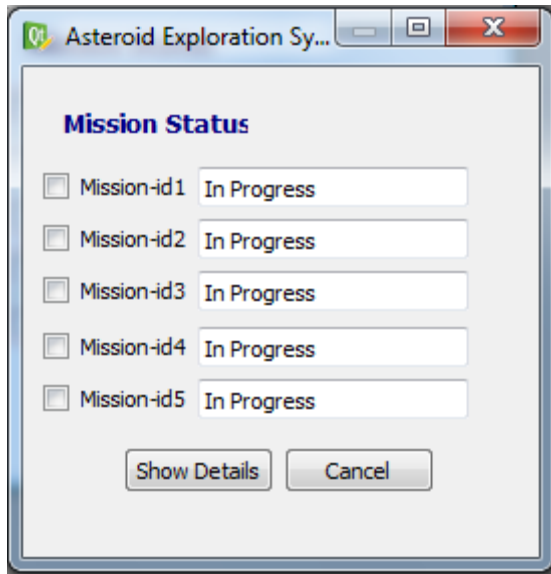
On login successful main menu is shown:



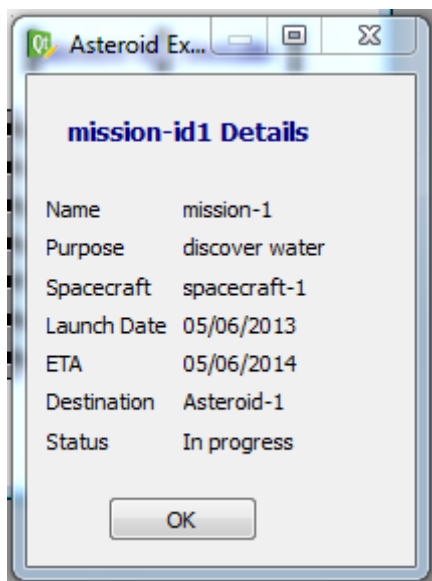
Define missions dialog: It calls IMissionManagement.create() to create mission



Monitor/update mission status: Mission management notifies user interface about status changes. User interface calls update functions (incrementFuel, incrementSpacecraft, incrementOperatingBudget) of IMissionManagement to change the status.



Mission Details:



Monitor/update spacecraft status: Mission management notifies user interface about spacecraft status changes. User interface calls update functions (updateFuel, updateGuidance, updateCommStatus, updateLocation, updateState, updateTargetAsteroid) of ISpacecraftManagementSystem to change the status.

Asteroid Exploration Sys...

Spacecraft Status

☐ spacecraft - 1

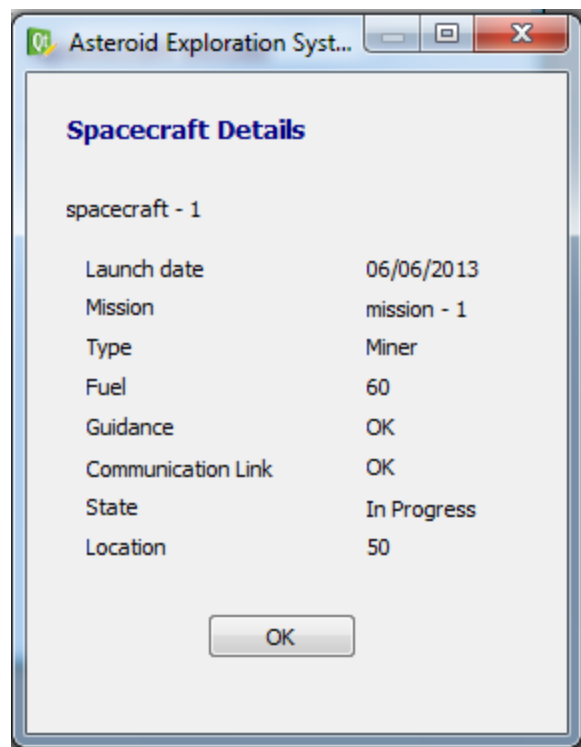
Fuel	65
Guidance	OK
Communication Link	OK
State	In Progress

☐ spacecraft - 2

Fuel	55
Guidance	OK
Communication Link	OK
State	In Progress

Show Details Cancel

Spacecraft Details:

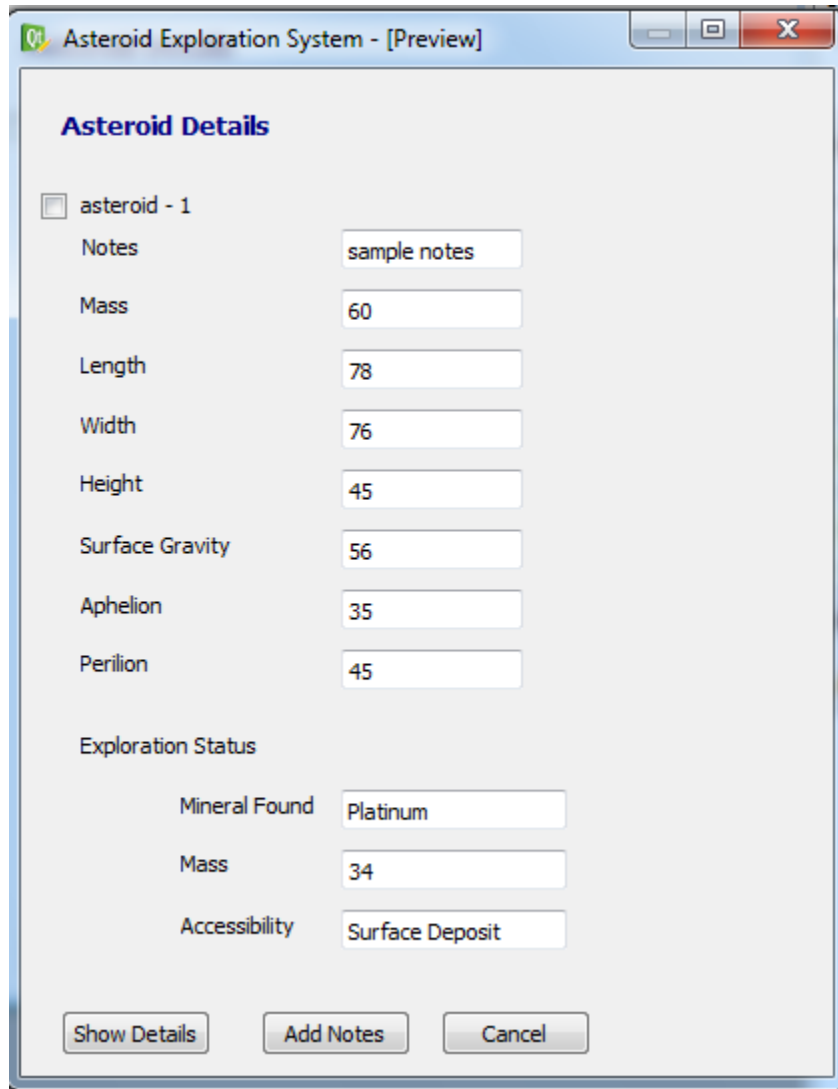


Monitor/update asteroid status: Mission management notifies user interface about asteroid's status changes. User interface calls update functions (updateMineralDiscovery, updateWaterDiscovery, updateLifeDiscovery) of IAsteroidInventorySystem to change the status.

The screenshot shows a window titled "Asteroid Exploration System - [Preview]". Inside, under the heading "Asteroid Status", there are two sections for different asteroids. Each section has a checkbox, a label "asteroid - 1" or "asteroid - 2", and a sub-label "Exploration Status". Below each sub-label are three text input fields. For "asteroid - 1", the fields are "Mineral Found" (Platinum), "Mass" (34), and "Accessibility" (Surface Deposit). For "asteroid - 2", the fields are "Water Found" (Yes), "Quantity" (20), and "State" (ice). At the bottom of the window are three buttons: "Show Details", "Add Notes", and "Cancel".

Asteroid	Mineral Found	Mass	Accessibility	Water Found	Quantity	State
asteroid - 1	Platinum	34	Surface Deposit			
asteroid - 2				Yes	20	ice

Asteroid Details:

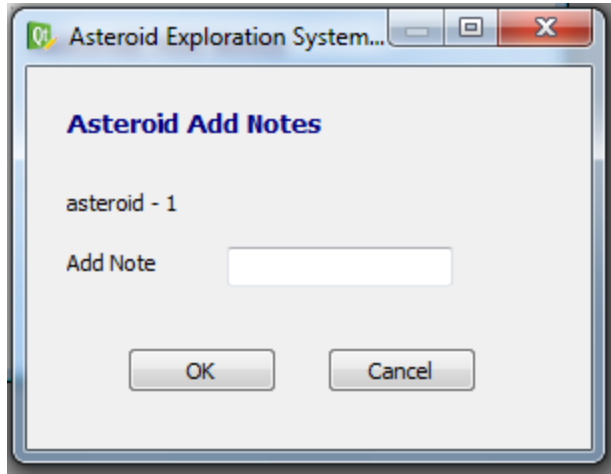


The screenshot shows a software window titled "Asteroid Exploration System - [Preview]". Inside, there's a section titled "Asteroid Details". A checkbox labeled "asteroid - 1" is checked. Below it, several fields are populated: Notes (sample notes), Mass (60), Length (78), Width (76), Height (45), Surface Gravity (56), Aphelion (35), and Perilion (45). Under the "Exploration Status" heading, there are three more fields: Mineral Found (Platinum), Mass (34), and Accessibility (Surface Deposit). At the bottom, there are three buttons: "Show Details", "Add Notes", and "Cancel".

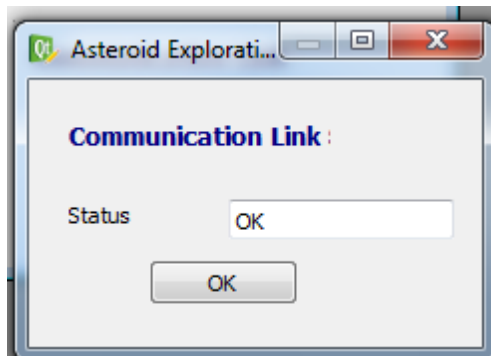
Asteroid Details	
<input checked="" type="checkbox"/> asteroid - 1	
Notes	sample notes
Mass	60
Length	78
Width	76
Height	45
Surface Gravity	56
Aphelion	35
Perilion	45
Exploration Status	
Mineral Found	Platinum
Mass	34
Accessibility	Surface Deposit

Buttons: Show Details, Add Notes, Cancel

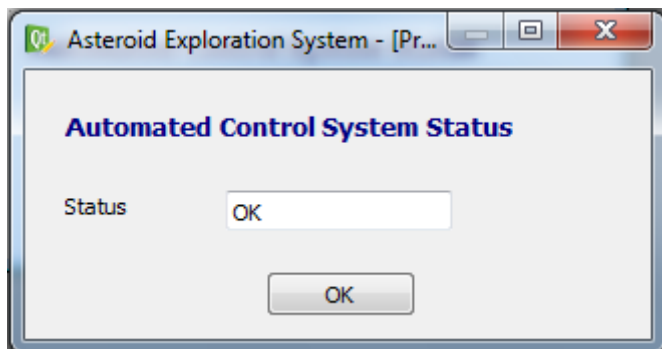
Asteroid Add notes: User interface calls `IAsteroidInventorySystem.addNotes()` to add notes



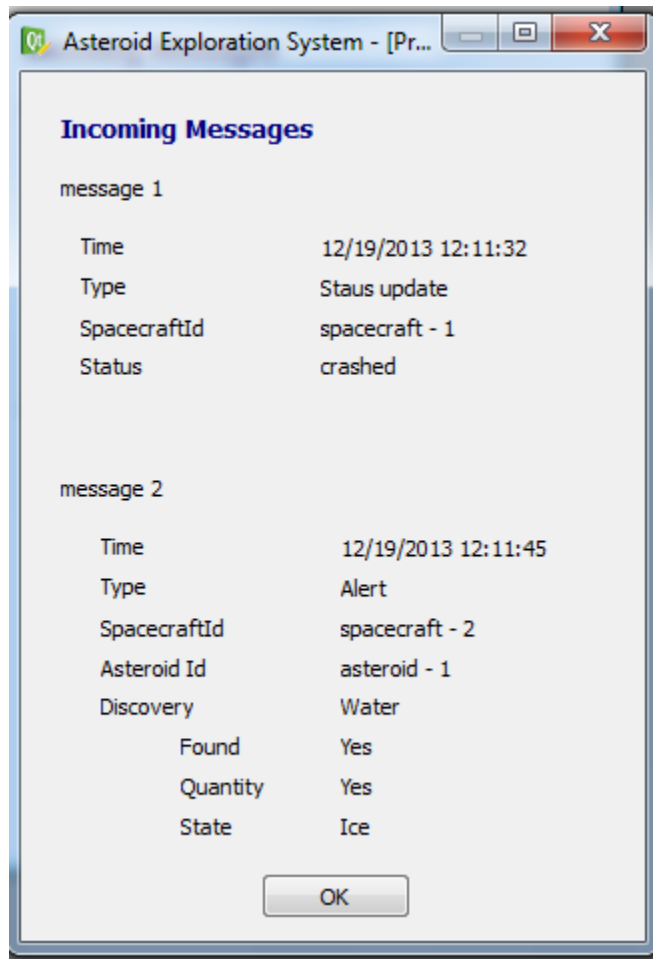
Communication Link Status: User interface calls `IMissionManagement.setCommunicationLinkStatus()` to set communication link status



Automated Control System Status: User interface calls `IMissionManagement.setAutomatedControlSystemStatus()` to set automated control system status.



Monitor incoming messages: Mission management notifies user interface when message is received.



Testing

From a test driver program we call APIs from services of asteroid management system to simulate the production environment. I perform these steps in order to test the system from test driver.

- Call authentication service to login.
- Create asteroid by calling create() of asteroid inventory service.
- Create spacecraft by calling create() of spacecraft management service.
- Assign asteroid to the spacecraft created above.
- Create mission by calling create() of mission management service.
- Assign spacecraft to mission created above.
- Create XML message to simulate status update.
- Call receiveMessage() on communication link object and pass the message.
- Check if spacecraft is updated by calling find() of spacecraft system.

- Create XML message to simulate state change.
- Call `receiveMessage()` on communication link object and pass the message.
- Check if mission and spacecraft is updated by calling `find()` of corresponding interfaces.
- Create XML message to simulate discovery.
- Call `receiveMessage()` on communication link object and pass the message.
- Check if asteroid is updated by calling `find()` of asteroid system.
- Call update functions of mission management and check if mission management is updates accordingly.

Risks

- There are some manual steps involved. E.g. Mission is created through user interface. If user enters data incorrectly then the application can go into inconsistent state.
- If messages from spacecraft are lost then also system can go in to unpredictable or inconsistent state.
- Exception safety is the key in this application. If system does not recover from exception then it can exhibit undefined behavior.
- Incomplete testing may leave some defects in the application.