AOS - ADD Document

Shai Havivyan
Vladi Odesski
Dan Roizman
Shahar Nahman

Chapter 1 - Usage Scenarios

1.1 User Profiles – The Actors

In the AOS system, there are two main groups of actors:

1) regular users – the programmers who use the system. These programmers are professionals in developing and maintaining robots. They have a solid background in programming.

2) AOS developers – programmers who developed the current working system. Like the first group of actors, they too have a solid background in programming.

Once we build a fully functional system, the AOS developers will have the following responsibilities: maintaining the system and expanding the system with additional features. Such responsibilities are not expected from the first group of users, and that is the main difference between these groups in our system.

Since our actors are programmers, it is of utmost importance to develop a productive, convenient, useful interface.

1.2 Use-cases

Use Case 1: Activate/Deactivate AOS server

Description

User who wants to activate the AOS server, to perform actions relevant to the server itself, or a user that has ended its current use in the system. This action is accessible through a button in the interface. Also, the interface constantly provides information about the state of the server: activated/deactivated.

Actors

All actors stated in section 2.1

Pre-Conditions

None.

Post-Conditions

Activation/Deactivation ended successfully.

Main Success Scenario

I. The user requests from the system to activate/deactivate the AOS server.

II. The system invokes/shuts down the AOS server application

III. The system notifies the user that the server is up/down and provides information about the state of the server after the action. The system saves the new state of the server.

Use case 2: Create new project

Description

Each project contains a set of skills documentation. Also, each project contains an environment file. Each skill reflects an ability of the robot. (e.g., navigation, pick up objects, image processing). The robot uses these skills to achieve the project's goal. For each skill, the system requires AM, SD files. The documentation files for each project will be saved in a separate folder.

Actors

All actors stated in section 2.1

Pre-Conditions

None.

Post-Conditions

New project created successfully.

Main Success Scenario

I. The user requests the system to create a new project.

II. The system inquires the user for the project name, a list of global variables and their types.

III. The system generates a new template of an environment file.

IV. The system saves the new project, notifies the user for the successful creation.

V. The user can now add new skills to the project and edit the project.

Alternative Scenarios

I. The user provided an empty project name. The action cannot be completed until the user provides a valid project name. (alphanumeric characters). Appropriate error message will be presented to the user.

II. The user provided the project name, which already exists in his projects folder. The action cannot be completed until the user changes the project name. Appropriate error message will be presented to the user.

Use case 3: Add new Skill to project

Description

Each skill requires SD, AM files that describe the nature of the skill, how it affects the world, return values, instructions for functions activation, etc. When the user wants to add a new skill, he provides the name of the skill and skill's parameters.

Actors

All actors stated in section 2.1

Pre-Conditions

The requested project exists.

Post-Conditions

The new skill added successfully to the requested project.

Main Success Scenario

I. The user requests the system to add a new skill to a project, from a list of available projects which is provided by the system.

II. The system inquires the user for the skill name and skill's parameters.

III. The system generates new templates of SD, AM files.

IV. The system saves the files of the new skill and adds them to the project folder. The system notifies the user that the action ended successfully.

V. The user can now edit the skill's files.

Alternative Scenarios

I. The user provided an empty skill name. The action cannot be completed until there is a valid skill name. (Begins with a non-special character)

II. The user provided a skill name, which already exists in the project's skills. The action cannot be completed until the user changes the skill name.

Use Case 4: Edit skill/environment file in project

Description

The SD/ AM/ environment files are JSON files. The User can edit them based on their predefined template created by the system.

Actors

All actors stated in section 2.1

Pre-Conditions

The requested project\ project and skill exist.

Post-Conditions

The file was edited successfully.

Main Success Scenario

I. The user chooses a project and a file to edit (skill's files, environment file), from a list of available projects and their sub-files, which is provided by the system.
II. The user can select between two options: edit the file in an external editor, or edit the files in the interface, using its built-in editor. After selecting the desired editing mode, the user can edit the files in the interface, or the system opens the files in an external editor.

Use Case 5: Delete Skill from Project

Description

If the user wishes to delete a skill from a project, he should choose the skill to delete.

Actors

All actors stated in section 2.1

Pre-Conditions

The requested project and skill exist.

Post-Conditions

The skill was deleted successfully.

Main Success Scenario

I. The user chooses a project and a skill in the project to delete, from a list of available projects and their sub-files, which is provided by the system.
II. The system requests confirmation for the deletion from the user, and after receiving it, deletes the requested skill and notifies the user the action ended successfully.

Use Case 6: Documentation Check

Description

The user edits the documentation files (SD, AM). At any point, the user can validate the correctness of the written code using the documentation check. For example, documentation check, checks that the global variables defined exists in the documentation files. However, it does not support checking the correctness of programming language code inside the JSON fields, and therefore differs from a compilation check. Our goal in documentation check is to verify the correctness of the template, and basic logic.

Actors

All actors stated in section 2.1

Pre-Conditions

AOS server is up.

At least one project exists in the system to check.

Post-Conditions

The documentation check ended successfully, or with errors regarding the user's code.

Main Success Scenario

I. The user requests the system to perform a 'documentation check' on a project.
II. The system sends an integration request to the AOS server.
III. The system notifies the user that the documentation check ended successfully.
IV. The user can now proceed to the next use case: Integration request.

Alternative Scenarios

I. The result of the integration request from the server returned errors in the user's code. In this case, the system refers the user to the problematic parts in the code. For each part, the system includes the relevant error message that appeared in the result of the request.

Use Case 7: Integration Request

Description

The user can request from the AOS server to build the project, by performing integration to the project's skills and environment file. There are several modes of integration request: code generation only, inner simulation (without activating the robot), sequence of action to run, robot activation, robot activation\ inner simulation without rebuilding the solver engine. The user can choose the integration mode and add additional parameters relevant to it. (e.g., in robot activation mode, the user can choose time interval between robot actions)

Actors

All actors stated in section 2.1

Pre-Conditions

AOS server is up.

At least one project exists in the system.

Post-Conditions
Integration request ended successfully.

Main Success Scenario

I. The user requests from the system to perform an integration request
II. The system inquires the user for integration mode, and relevant parameters for the chosen mode.
III. The system sends the integration request to the AOS server. The AOS server performs the integration request and builds the solver engine code (unless the chosen mode is code generation/start without rebuilding) successfully. The system notifies the user the action ended successfully.

Alternative Scenarios

I. One of the parameters values provided by the users is illegal. (e.g., The parameter represents time interval and the value provided is negative) The request cannot be sent until the user provides a valid parameter value. Appropriate error message will be presented to the user.

II. The documentation files/environment file provided by the user contains errors. The use case fails at the stage of the integration request. The system notifies the user there are errors and detailed information about the errors. The user should fix the errors and try again.

III. If the integration request ended successfully, but the build of the solver engine code fails, the system presents an error list to the user. The user can open a specific error from the list in its location in the documentation files, in an external editor. (e.g. vs code)

Use Case 8: Request Robot's state

Description

See Robot state definition in Glossary. The user wishes to receive the updated state of the robot. The user provides a single parameter that represents the maximum number of states that the belief state should contain.

Actors

All actors stated in section 2.1

Pre-Conditions

Use case 7 – integration request ended successfully in activation robot/inner simulation/sequence of actions modes.

AOS server is up.

Post-Conditions

None.

Main Success Scenario

I. The user requests to receive the robot's current state, and provides the requested parameter.
II. The system sends an execution outcome request to the AOS server.
III. The system extracts from the results the current belief state of the robot and notifies the user the action ended successfully.
IV. The user receives the belief state of the robot according to a predefined template.

Alternative Scenario

I. The user provides invalid vector size value. the size must be bigger than 0. In addition, an empty vector is a meaningless result in our context. Means, the user must provide a value bigger than one or the action won't be completed.

Use Case 9: Request Robot's after a certain action

Description

The use case is identical to use case 8: request robot’s state, apart from one change: the user now wants to receive the robot’s state after the execution of some action. The user needs to provide an additional parameter that represents the action’s number.

Actors

All actors stated in section 2.1

Pre-Conditions

Use case 7 – integration request ended successfully in activation robot/inner simulation/sequence of actions modes.

AOS server is up.

Post-Conditions

None.

Main Success Scenario

I. The user requests to receive the robot’s current state after a certain action, and provides the requested parameters.
II. The system sends an execution outcome request to the AOS server.
III. The system extracts from the result the belief state of the robot after the requested action performed and notifies the user the action ended successfully.
IV. The user receives the belief state of the robot according to a predefined template.

Alternative Scenario

I. The user provides invalid vector size value. the size must be bigger than 0. In addition, an empty vector is a meaningless result in our context. Means, the user must provide a value bigger than one or the action won't be completed. Appropriate error message will be presented to the user.

II. The user provides invalid action value. meaning, the user requests the state of the robot after action 'x', and the robot performed only 'k' actions so far, such that x < k. The user must try again. Appropriate error message will be presented to the user.

Use case 10: Request History of robot's actions

Description

The user can request a history of a robot's actions, as part of specific current activation. Means, if the robot is currently activated, the user can request the history of the actions performed by the robot until now. As part of this use-case, users can also query the system for the robot’s belief state after certain action, which triggers use-case 9

Actors

All actors stated in section 2.1

Pre-Conditions

Use case 7 – integration request ended successfully in activation robot/inner simulation/sequence of actions modes.

AOS server is up.

Post-Conditions

None.

Main Success Scenario

I. The user requests to receive the history of the robot's actions.
II. The system sends an execution outcome request to the AOS server.
III. The system extracts the history of the robot’s actions and their observations from the result and notifies the user the action ended successfully.

Use case 11: Request to stop the robot

Description

The user can request to stop the activation of a currently activated robot/stop a running simulation in a project.

Actors

All actors stated in section 2.1

Pre-Conditions

Use case 7 – integration request ended successfully in activation robot/inner simulation/sequence of actions modes. The AOS server is up.

Post-Conditions

None.

Main Success Scenario

I. The user requests to stop a robot in a certain project.
II. The system sends a request to the AOS server.
III. The system notifies the user the action ended successfully.

Use Case 12: Request graphical representation of robot state

Description

The result of the robot's state in use case 8, is not user-friendly. Therefore, the user can request a graphical representation of the data. State variables are linked to graphical objects, such as pictures or icons for the representation.

Actors

All actors stated in section 2.1

Pre-Conditions

Use case 7 – integration request ended successfully in activation robot/inner simulation/sequence of actions modes.

AOS server is up.

Post-Conditions

None.

Main Success Scenario

I. The user requests a robot state, according to Use-Case 8.
II. The user requests a graphical representation of the returned state.
III. The system will process the returned state and represent it to the user.

Use case 13: Request the generated code

Description

After the successful completion of use case 7: integration request, the AOS server generates the code of the solver engine (= decision making engine) and middleware layer (= functions as a mediator between the solver and the robot). The user can request to view/edit the generated files for debug purposes.

Actors

All actors stated in section 2.1

Pre-Conditions

Use case 7 – integration request ended successfully. The AOS server is up.

Post-Conditions

None.

Main Success Scenario

I. The user can request to open the generated files from various locations in the documentation files in the UI interface. For each location chosen, the system can link it to a location in the generated code and open it there.
II. The system will open an external editor, already configured with the project of the solver engine.
III. The user can run the project in the external editor for debug purposes.

Chapter 2 – System Architecture

Our system architecture is Layered Architecture. We have the following layers:

- Presentation Layer – implemented in MVC architecture.
- Service Layer – includes all functionality exposes the API on the system to the presentation layer and therefore to the user.
- Domain Layer – includes the main logic of the system, and all the main components which we will elaborate later.

The system uses a service of external sever – the AOS server.

Different software components in our system

JavaFx

Client application platform for desktop, mobile and embedded systems built on Java. We will use JavaFx for the implementation of the User Interface (Presentation Layer).

Therefore, the Service layer and the Domain Layer will be implemented in java.

Vs Code

One of the requirements is to enable the launch of vs code in the location of an error. (see use case 7 – alternative scenario 3). Therefore, we need to integrate with vs code in our system.

OkHttp

To communicate with the external AOS server, we need a client to sent http requests to the AOS server, which exposes a REST API. We will use this component in the domain layer.

Testing

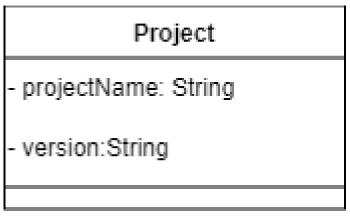
Our testing environment will be implemented using JUNIT5.

Chapter 3 – Data Modeling

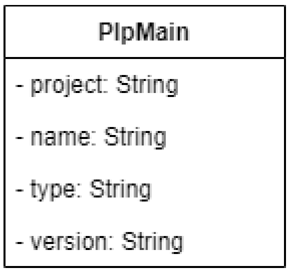
3.1 description of data objects

In the following section, we will introduce the main data objects in the system. Most of those objects are sections from the documentation files (JSON sections). The reason we include them in our system is to perform documentation file correction checks and other logical checks.
Therefore, the meaning of their fields relates to the inner logic of the AOS server. We provide a short explanation of each field for completeness of the explanation, though it is not needed.

- Project – the main entity in our system. Our system is designed to create and manage projects, their skills and environment file. Project entity contains the project name, the current version of the AOS server, an environment object, and a collection of skill objects.

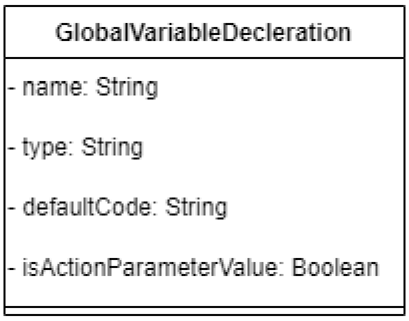


- PlpMain – The "PlpMain" is the header section that each documentation file contains. It contains the project name, documentation file name (for environment file – the name is constant – "environment", for skills – each SD, AM describing the same skill will have the same name), documentation file type (environment, plp- for SD, glue- for AM), and version of the documentation (currently 1.2).

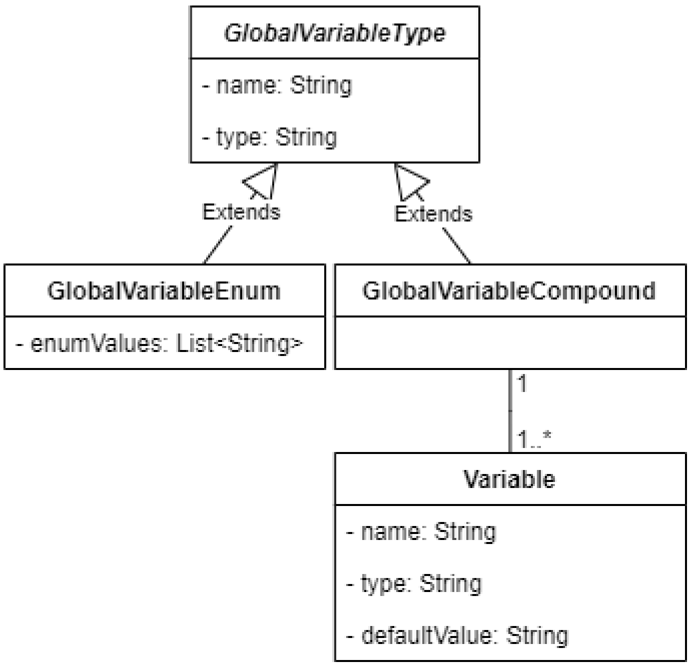


- Env – represents an environment file. Should contain a collection of global variables, their name, and types (e.g., Enum or complex type or primitive C++ type). The env object also contains the initial belief state of the robot and special states code. Env contains only complex data models, therefore it has an empty representation in the UML.

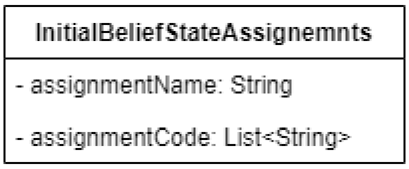
- GlobalVariable - represents the template for global variable in the environment file – contains a name, type, defaultCode and isActionParameterValue field. All these values are for internal use of the AOS server, this is the expected structure of global variables.



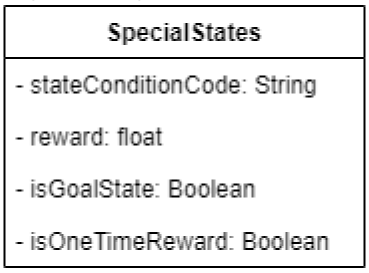
- GlobalType- if a global variable has non primitive C++ type, it's type should be declared by its name and either it Enum values as String or the types it composed of.



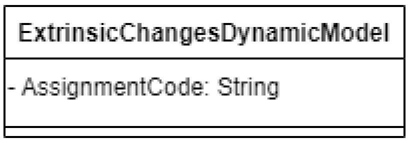
- InitialBelifeStateAssignment – represent every belief state that the robot can be in, it contains the belief state name, and list of the string contain code to determinate if the robot is in this state.



- SpecialStates – this section in the SD file allows the user to define desired or undesired states. Desired states have larger rewards and vice versa. To define such state, the user should supply stateConditionCode (string, code in c# for example), reward value(decimal), is it one time reward, and is it a goal state

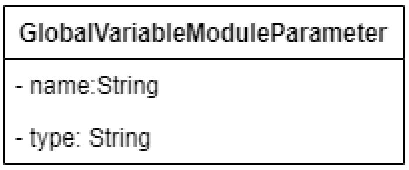


- ExtrinsicChangesDynamicModel- this class represents changes in the environment that did not happen due to the robot's actions and can affect the robot next course of actions.

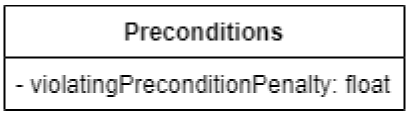


- SD- represent the SD file it contains the list of *GlobalVariableModuleParameter*, the global parameter that this skill use. precondition- a list of the Precondition that need to be held before the skill will be execute. *DynamicModels*- list of *DynamicModel* represent how the skill change the state of the robot.

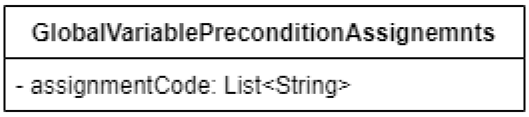
- GlobalVariableModuleParameter- class that define the global variable used in this skill, it contains *name* the name of the variable as it declared in the env class, *Type* the type of the variable as declared in the env class (if it composed type only its name).



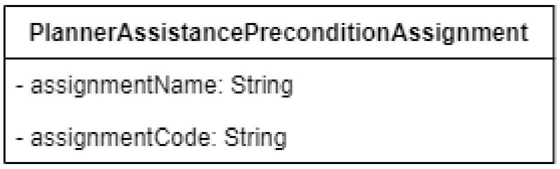
- Preconditions – class that represent the condition need to be held before the skill execution. It contains list of *GlobalVariablePreconditionAssignment*, and ViolatingPreconditionPenalty – a float that the solver uses to "punish" if the skill precondition is held but did not execute.



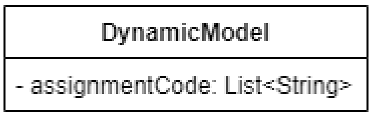
- GlobalVariablePreconditionAssignments – by default, the skill meets the precondition. The user can assign a value to a parameter "__meet condition", to determine if the precondition is met or not. The assignment block is within the field "assignment code" as string. (, e.g. "AssignmentCode": "__meetPrecondition = oDestination != state.robotLocation;")


GlobalVariablePreconditionAssignemnts
- assignmentCode: List<String>

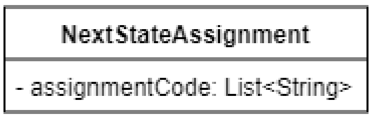
- PlannerAssistancePreconditionAssignment – the user can assist the solver engine with decision making (choosing next skill to activate), by assigning a value to param "__heuristicValue". The assignment is made in this section, and composed of assignmentName (String), assignmentCode (String).



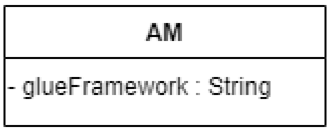
- DynamicModel – Class that represents how the skill changes the state of the robot, it contains list of *NextStateAssignment.*



- NextStateAssignment- class that represents the next state of the robot. It contains *AssignmentCode* list of string, each string is a code in C++ that assign the values of the next state of the robot.



- AM – class that represents the AM file – more detailed for the skill. Its conations *GlueFramework* string that represent the robot framework, *ModuleResponse* – object to define how the outcome of the execution of the skill translate to variables values, *ModuleActivation* Object that describe the activation of the skill's code, *LocalVariablesInitializations* list of LocalVariablesInitialization. The local variables used in this skill and their initialization.



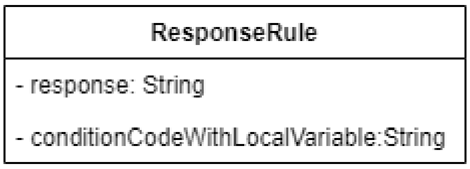
- LocalVariablesInitialization – Object that represents the parameters used in this skill, there are three types of this object, *SDSource* parameters from SD file, SkillSource variable that returned from the execution of previous skill, and variables that derive from the robot's framework (e.g., ROS).

- SkillSource- object derives from *LocalVariablesInitialization.* Its attributes are *LocalVariableName* a string the name of the variable, *VariableType* the type of the variable, *FromROSServiceResponse,* Boolean value, AssignmentCode string of code in python for assigning the value of this variable, importCode List of *ImportCode* the imports modules needed to receive the value of the variable.

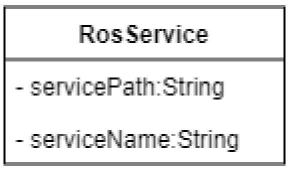
- SDSource – object derive from *LocalVariablesInitialization*, its attributes are InputLocalVariable as string of the name of the local variable, FromGlobalVariable, string of the global variable we want to copy the value of.

- ModuleResponse – the ModuleResponse JSON section defines the translation between an actual execution outcome of a skill, to observations the AOS planning engine (AKA solver) can reason about. This section contains one or more response rules, which defines the mapping.

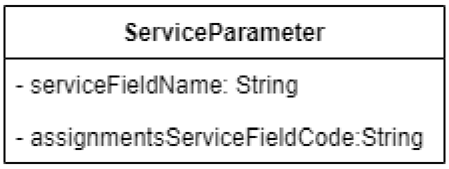
- ResponseRules – each response rule contains "response" field, defines the observation name (e.g., "success") and "ConditionCodeWithLocalVariables", string field that uses the user to define when the skill returns the current response, may depend on local variables values.



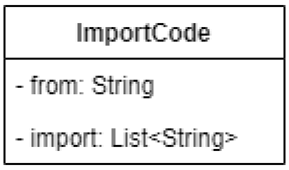
- ModuleActivation - The ModuleActivation section describes how to activate the skill code in the AOS server. Contains "RosService" section since we currently support ROS (meta-operating system for robots).
- RosService – if the skill code is based on ROS framework, we define how to activate a cmd tool "ROS1 service", which the AOS server uses to activate the skills. Include fields: service path (String), service name (String), importCode, serviceParameters.



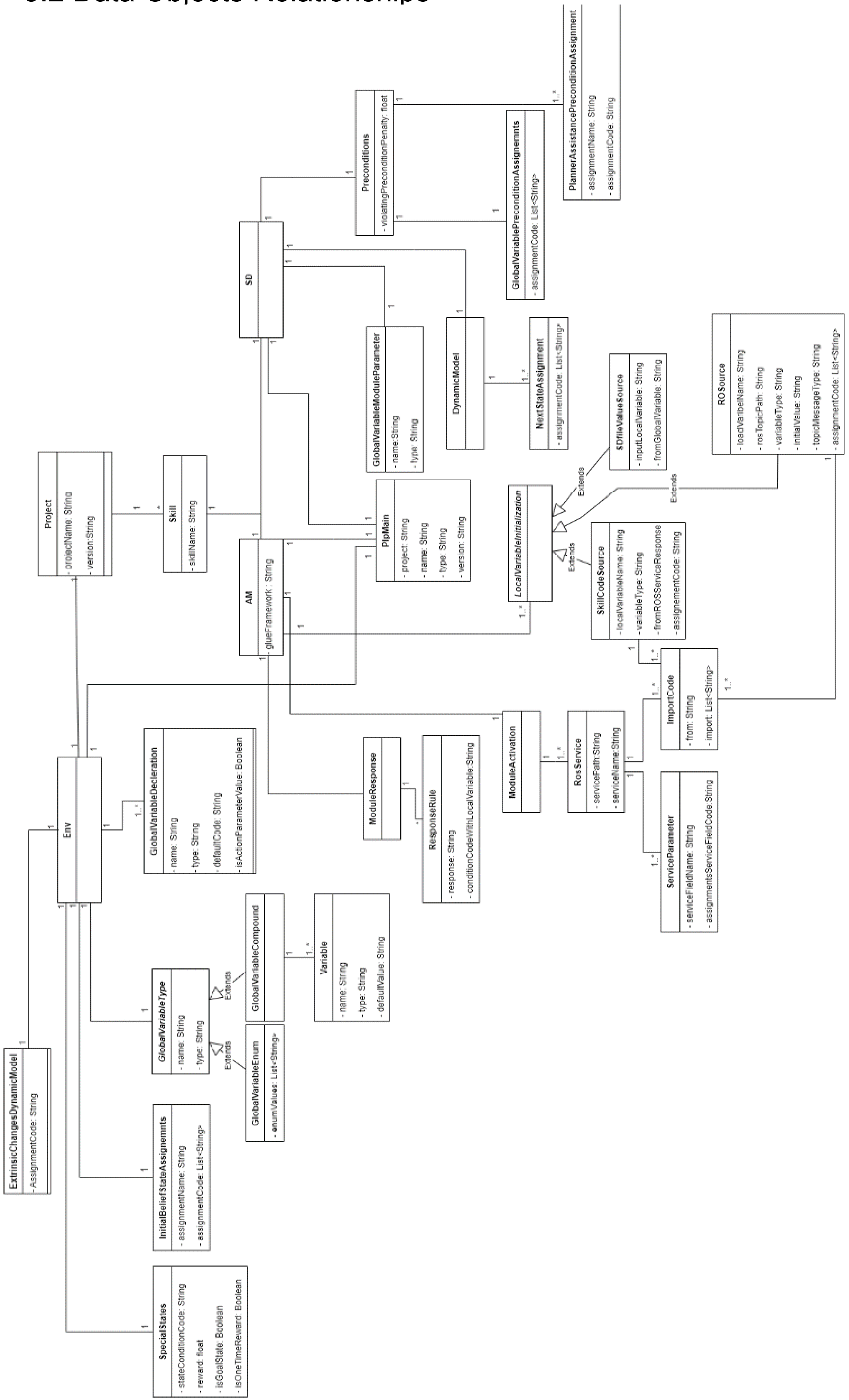
- ServiceParameters – parameters we sent to the service – serviceFieldName (String), assignmentsServiceFieldsCode (String)


ServiceParameter
- serviceFieldName: String
- assignmentsServiceFieldCode:String

- importCode - represents imported modules used when calling the service. Contains "to" (String) and "import" (List<String>) fields. (e.g., from numpy import transpose)



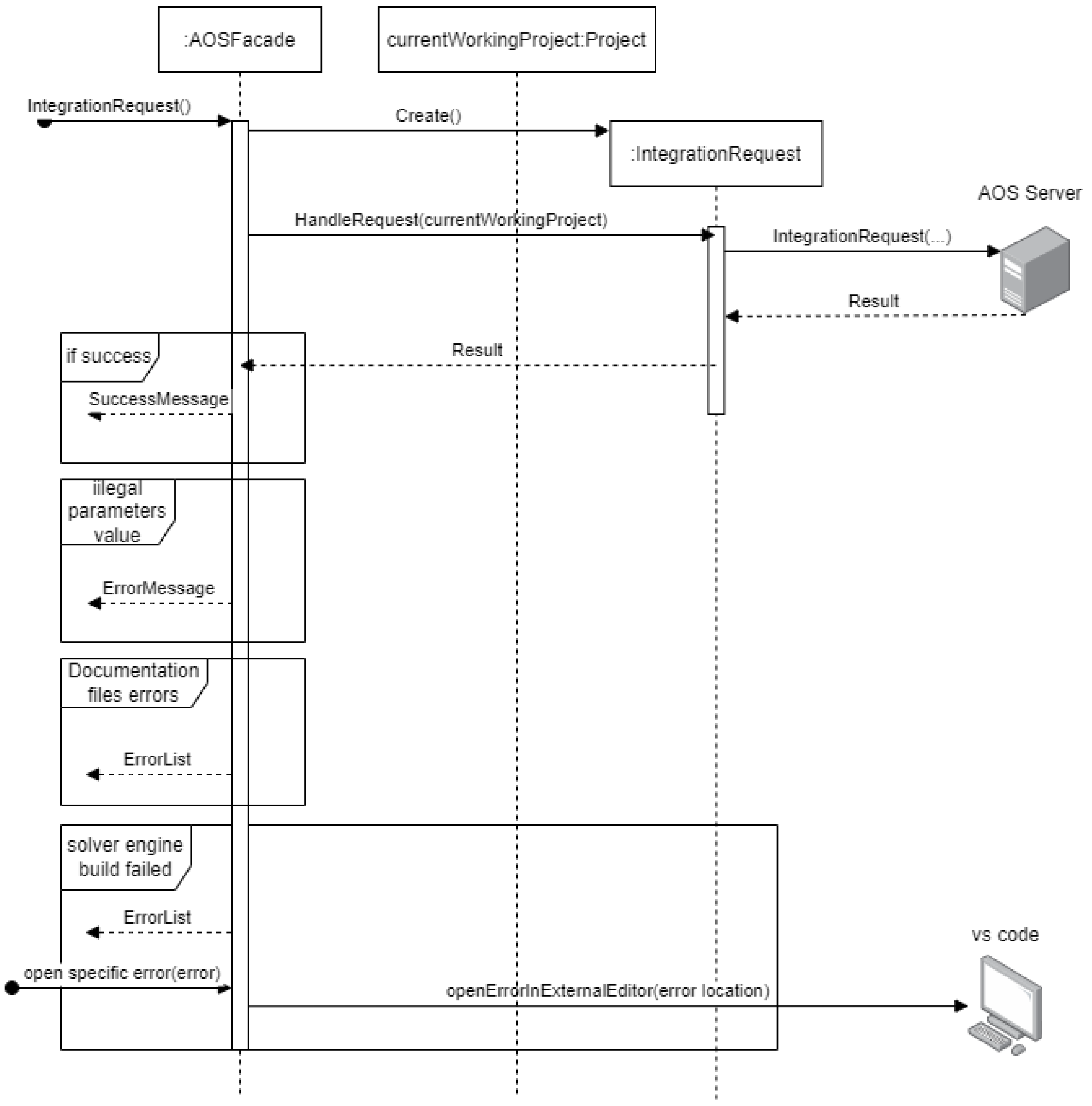
3.2 Data Objects Relationships



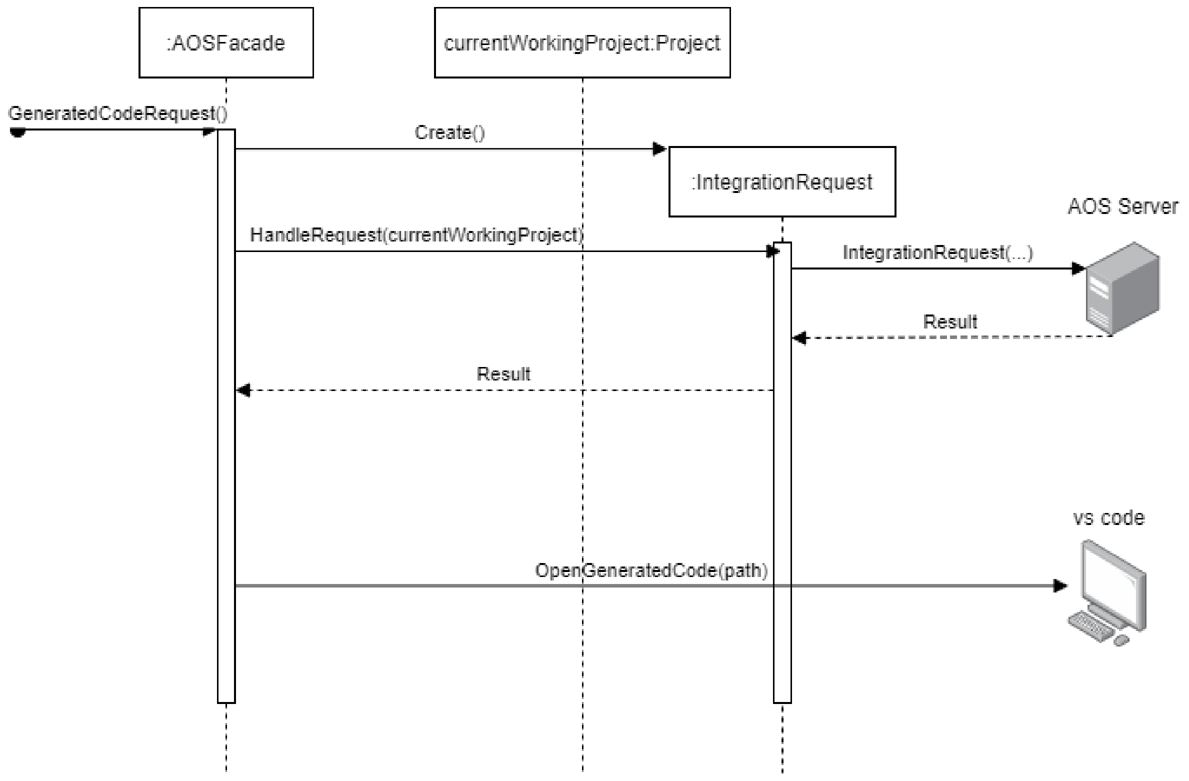
Chapter 4 - Behavioral Analysis

4.1 Sequence Diagrams

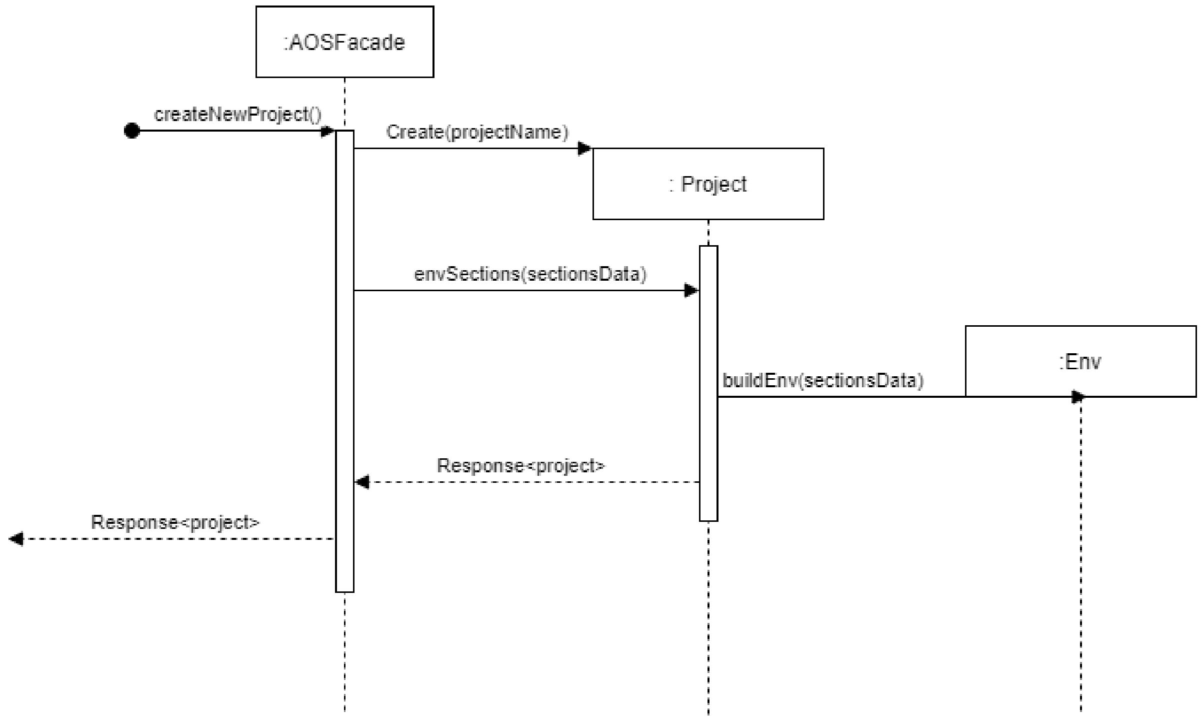
Integration Request



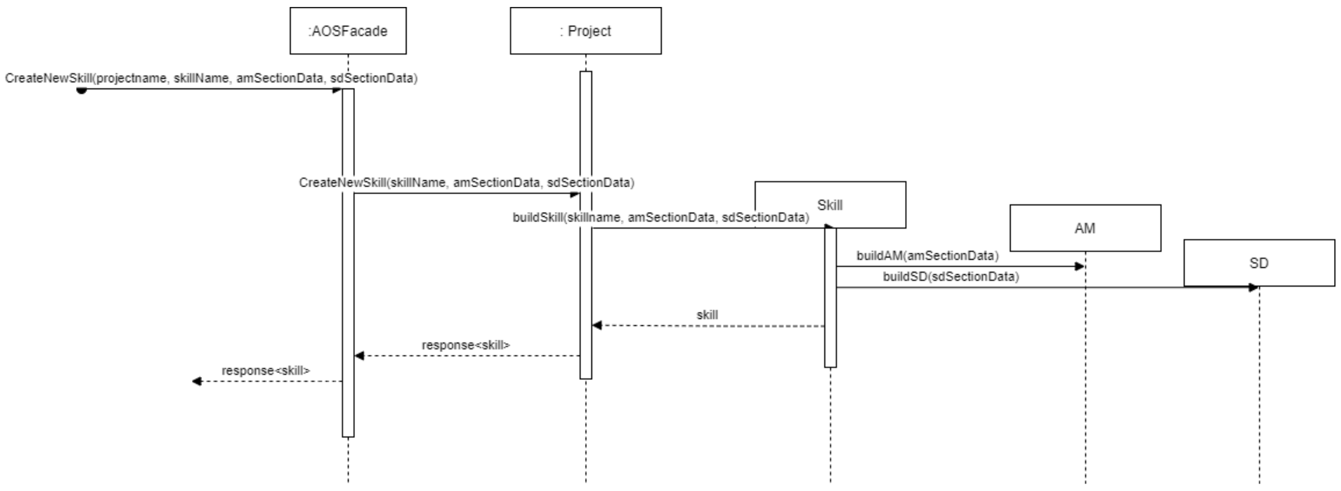
Request Generated Code



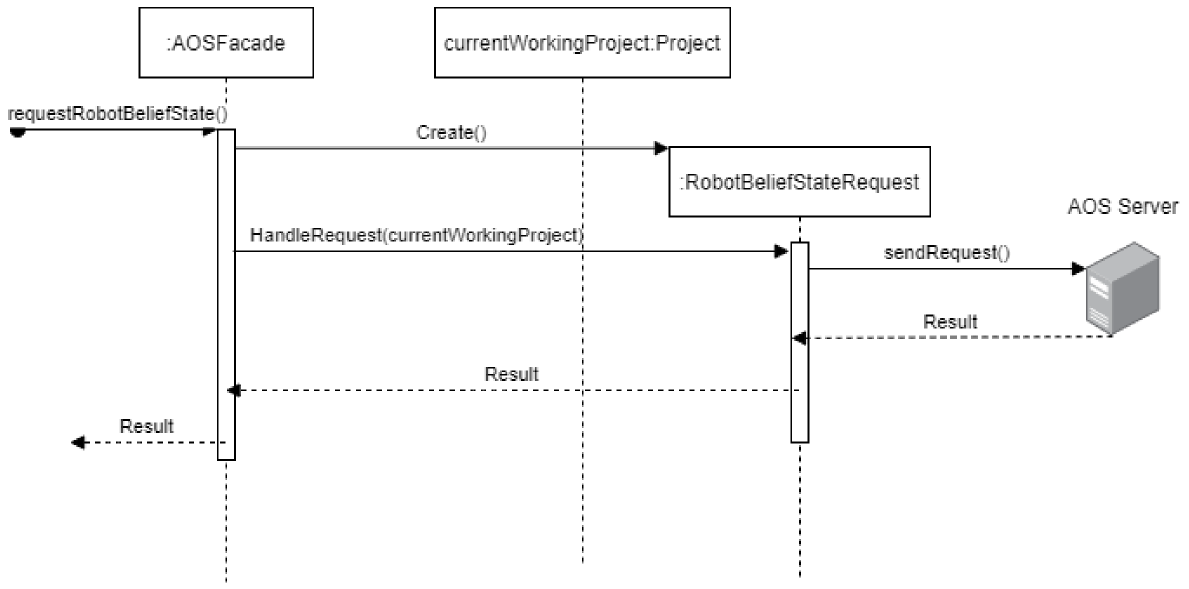
Create New Project



Add New Skill

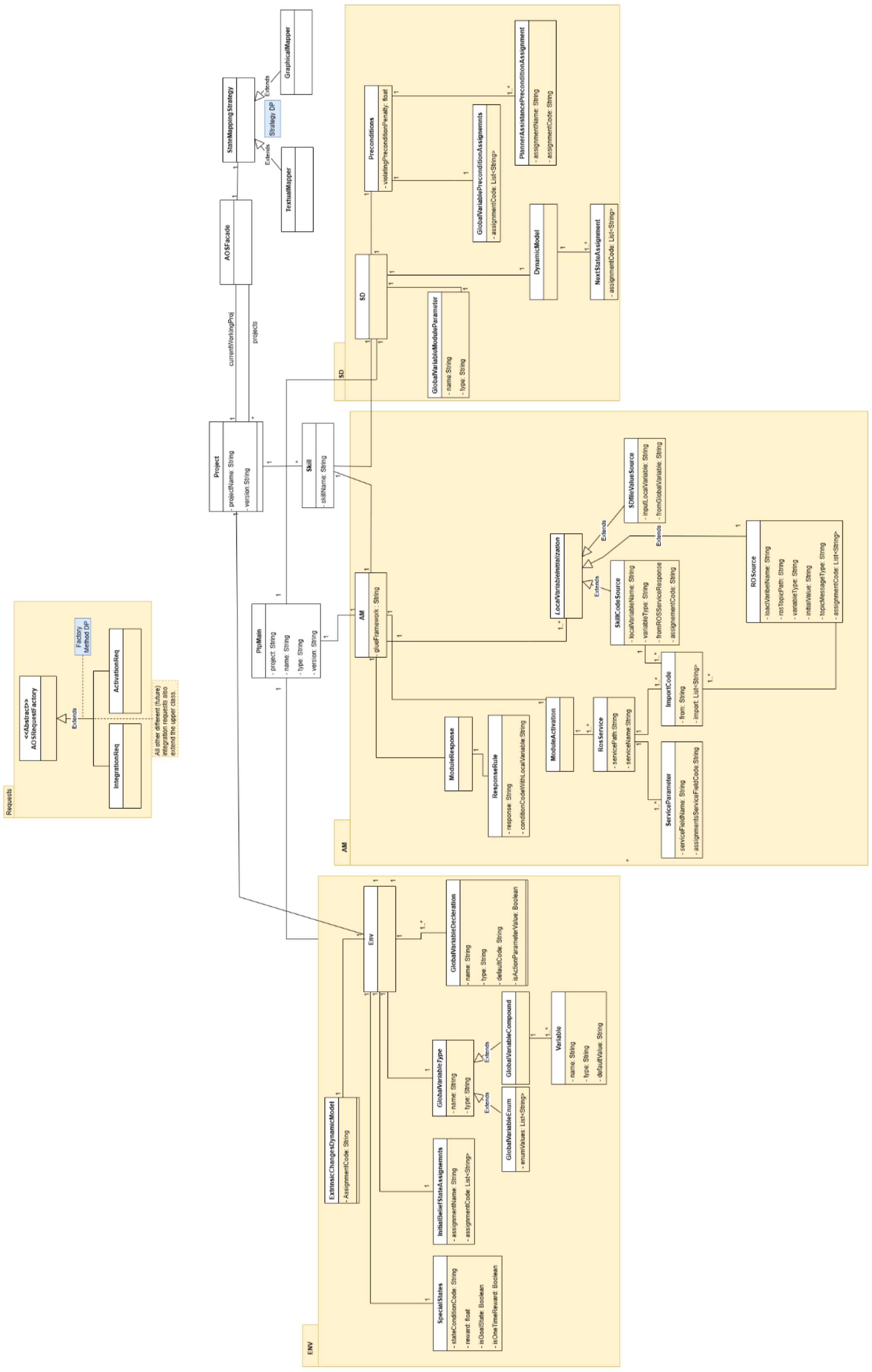

:AOSFacade
: Project
CreateNewSkill(projectname, skillName, amSectionData, sdSectionData)
CreateNewSkill(skillName, amSectionData, sdSectionData)
Skill
buildSkill(skillname, amSectionData, sdSectionData)
AM
buildAM(amSectionData)
SD
buildSD(sdSectionData)
skill
response<skill>
response<skill>

Request Robot Belief State


:AOSFacade
currentWorkingProject:Project
requestRobotBeliefState()
Create()
:RobotBeliefStateRequest
AOS Server
HandleRequest(currentWorkingProject)
sendRequest()
Result
Result
Result

Chapter 5 – Object Oriented Analysis

5.1 Class Diagram



5.2 Class Description

- **AOSFacade:**
  - Represents The facade to the domain layer.
  - Exposes the API to the domain layer.
  - Contains a collection of Project entity
  - Contains a reference to the current Working Project.
  - Contains the current strategy of the belief state representation. (Will be explained later)

RequestFactory: Implements the factory method DP
  - Represent a request to the AOS server.
  - An abstract class, which has an abstract handleRequest() method, responsible for handling requests for the AOS server.
  - All different AOS integration requests extend this abstract class.
  - In this way, we can easily add future requests to the system

IntegrationRequest:
  - The object that is built by the above builder. Example to a request type in the system.

- **Project**
  - Project represent a defined goal for a robot. Therefore, includes two types of files: env, skill (also, two types of skill files: SD, AM), as explained in ARD.
  - Aggregates all the project files, provides a convenient access to all the inner files.

- **StateMappingStrategy**
  - represents the mapping strategy of the returned belief state of the robot. One of the requirements is the allow presenting the results in a graphical representation. The default is textual representation. Therefore, there are two classes inheriting StateMappingStrategy: TextualMapper, GraphicalMapper
  - implemented in Strategy Design Pattern.

- **TextualMapper**
  - Explained in StateMappingStrategy

- **GraphicalMapper**
  - Explained in StateMappingStrategy

- **Env**
  - As Explained in section 3.1, the Env class represent environment file in our system. It contains all the relevant data sections of the file and manage them.

- **Skill**
  - As Explained in section 3.1, the Skill class contains SD attribute and AM attribute.
  - Also, the class contains the skill name.
  - Represents a single skill in a project in our system.

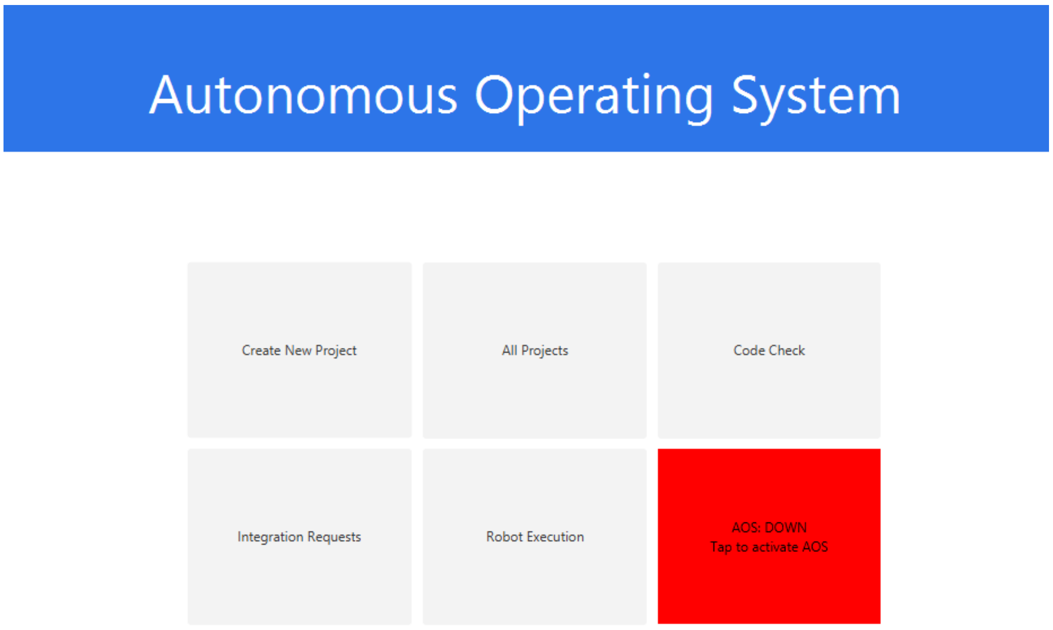
5.3 Packages

Our Domain Layer (Which is in the package 'Domain Layer') consists of four inner packages:

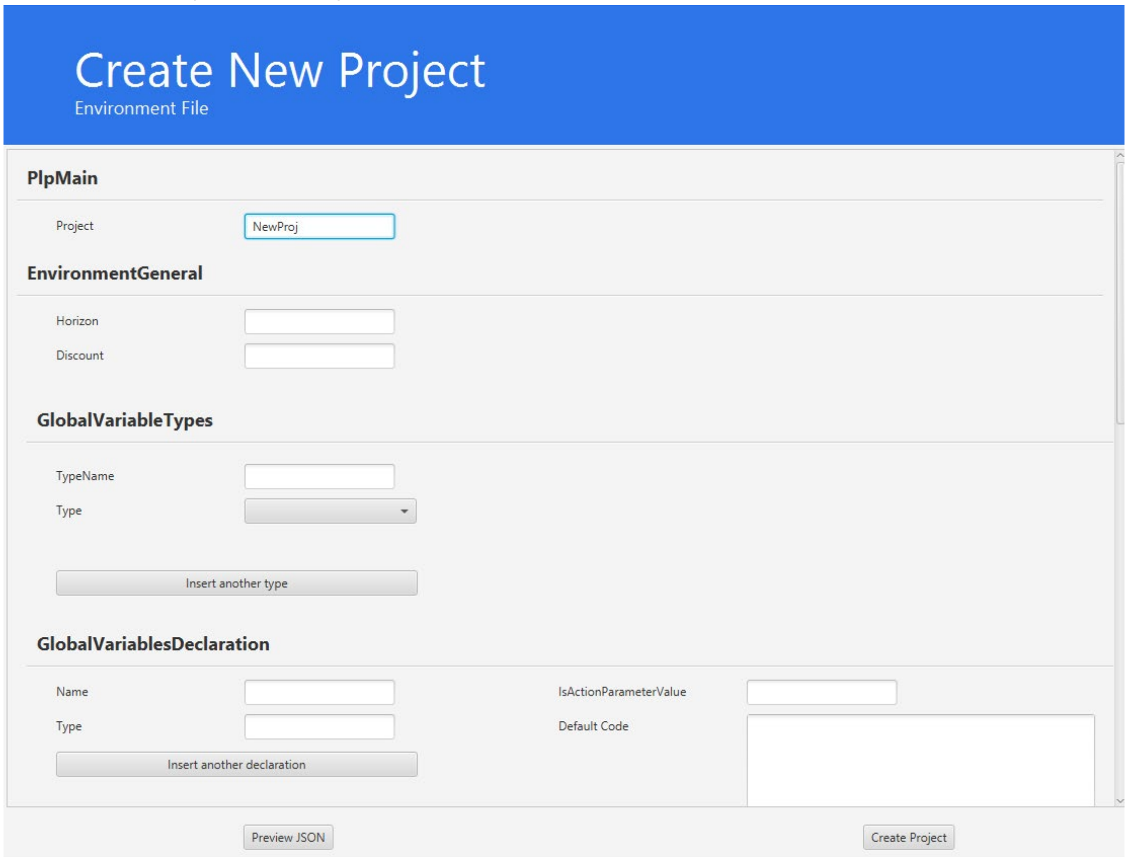
- Env – contains Env class, and all related classes that represents the attributes of env class.
- SD - contains SD class, and all related classes that represents the attributes of SD class.
- AM - contains AM class, and all related classes that represents the attributes of AM class.
- Requests – contains RequestsFactory, and all types of requests implementing it. This package contains all request related behavior.

Chapter 6 – User Interface

Main Screen of the system. Contains relevant functionality in basic design. Presents the state of the AOS server.

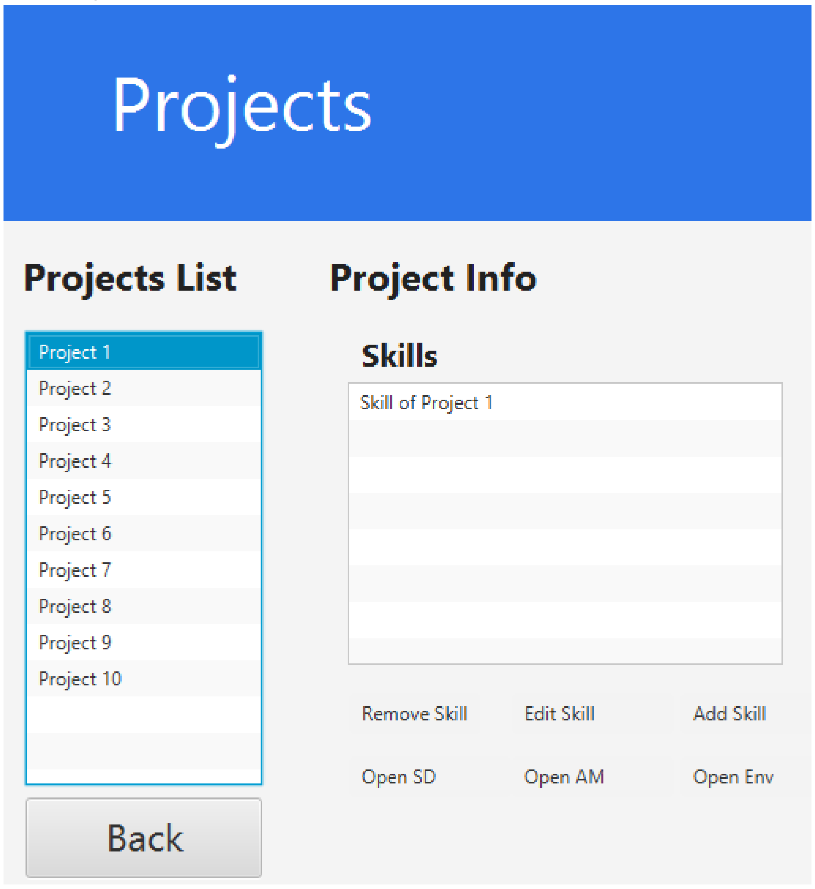


Create New Project Screen. Accessible from 'Create New Project' button.
The user can type the project name, and other variables related to the environment file



Project: NewProj

List of Projects screen. Accessible from 'All Projects' button.
The user can select a single project and add/remove/edit a skill and open all types of files in the project.


Projects

Projects List

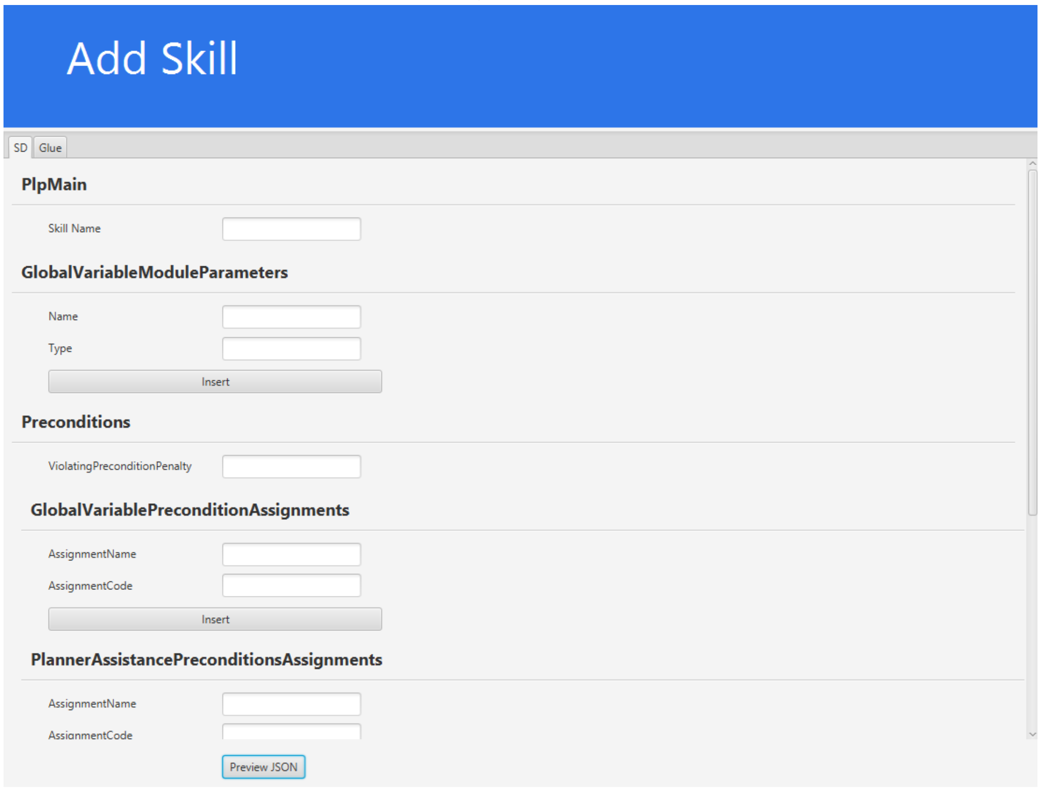
- Project 1
- Project 2
- Project 3
- Project 4
- Project 5
- Project 6
- Project 7
- Project 8
- Project 9
- Project 10

Project Info

Skills

- Skill of Project 1

Add new skill screen. The user can type in all relevant variables to create the SD, AM(Glue).


Add Skill

PlpMain

Skill Name

GlobalVariableModuleParameters

Name

Type

Preconditions

ViolatingPreconditionPenalty

GlobalVariablePreconditionAssignments

AssignmentName

AssignmentCode

PlannerAssistancePreconditionsAssignments

AssignmentName

AssignmentCode

Term	**Definition**
Environment File	Each project must contain an environment file. This file contains four parts: ● Global variables defining system state ● Initial belief state ● Extrinsic changes that affect the robot's actions (e.g what happens when it's raining?) ● An objective function - which basically determines the goal of the task.
SD File	SD (Skill description) is a file for each skill in the project. This is a high level description of the next state once the skill is executed. It also describes how executing this skill affects the environment variables.
AM File	The SD file specifies an abstraction that (ideally) corresponds to our concept of what the skill does. The reality, however, is expressed in code. The AM file specifies the relation between these two abstraction levels, as well as providing how to execute the skill in a low level manner.
Robot's state	A state of the robot contains a collection of assignments to state variables. The state of the robot is a non-deterministic value, which means that the robot can only estimate its current state. When a robot is asked to return its state, it returns a belief state – vector of states as defined above. The vector may contain multiple instances of the same state, which allows us to calculate the state's probability.

Action's observation	The observation of an action, is a conclusion the model can include after the execution of the action. This is useful for the solver engine (decision making engine). The conclusion is based on the values of state variables, and the rules that define it appear in the documentation files of the skill that represents the action.
Execution outcome	Execution outcome is a type of request that can be requested from the AOS server. the result of the request contains the following attributes: current belief state, and then a series of tuples in the form of <current belief state, action performed, observation, next belief state>. for example: current belief state, <initial state, action 0, observation, belief state after 0>, <belief state before 1, action 1, observation, belief state after 1> etc.
AOS server	A server that exposes a restful API, which practically allows us to generate the decision algorithm according to the project's documentation files. Eventually this code executes the robot to perform the desired goal of the task.
AOS Server API	The API of the AOS server, it supports the following requests: integration request to activate robot, query for robot's state, etc
Project	A project mainly describes a task to be performed by a certain robot. It contains an environment file and skills (SD and AM files for each skill) that a robot can execute to achieve the goal of the task.

Skill	A basic skill that a robot can execute. Execution of a sequence of skills is often used to achieve a certain goal (=task). Skill is described by SD and AM files. (E.g - navigation, picking up objects, etc).
Documentation file	One of the following files: ● Environment File ● SD File ● AM File
Generated file	An output .cpp file that is generated out of the documentation file that is provided to the AOS server as an input.
Inner simulation	Simulating the decision engine of the AOS server in a certain project, without physically running the robot.

Chapter 7 – Testing

The following table is a description of the E2E (A.K.A Acceptance Tests) we will be conducting in order to test our system.
Each record contains the name of the test, a short description describing the test process and finally, the expected result of the test.

Note: *There are also integration tests and unit tests that we will be implementing throughout our development process.*

Test Name	**Description**	**Expected Result**
	Functional Requirements	
AOS Activation	1. Ask the system to activate AOS server. 2. Check that the AOS server is up (e.g. ping it).	AOS server is up and running, responds to tcp ping.
AOS Deactivation	*Note: The test assumes that the AOS server is running.* 1. Ask the system to turn down the AOS server. 2. Check that the AOS server is down (e.g. by checking the process table).	AOS server is down and doesn't show in the process table.
AOS Status	Repeat the following for each state (on/off) of the server: 1. Ask the system for the status of the AOS server 2. Check that the response matches the actual state of the server (up/down)	System's response matches the actual state of the AOS server.
Show Projects	1. Inject some projects into the system 2. Ask the system for all the projects in the system. 3. Check that all the projects injected in step 1 exist in the response list.	System returns all injected projects and nothing but them (assuming that there were no other projects prior to test execution).
Open Project	1. Ask the system to open a specific project.	System loads the project without exceptions, all

	2. Check that all project's properties are present (project name, json properties, etc).	properties match to the properties presented in the json file.
Create Project	1. Ask the system to create a new project and fill relevant input details. 2. Check that there is a new folder with the given name in the File System, with a matching env.json file in the folder.	The project is created successfully, a folder was opened and there's env.json file matching the input given to the system at step 1 of the test.
Add Skill	1. Ask the system to add a new skill to a given project and fill relevant input details. 2. Check that there are 2 new files (glue.json and sd.json) matching the skill name and the relevant input details inside the matching project directory in the File System.	The skill was added successfully to the project, 2 files were added to the project's folder in the file system.
Delete Skill	1. Ask the system to delete a specific skill 2. Check that there are no documentation files (am and sd) matching the skill's name inside the project's folder in the file system.	The skill was successfully deleted, no matching documentation files are existing with the skill's name inside the project's folder.
Show All Skills of Project	1. Ask the system to view all skills of a given project. 2. Check that all actual skills are present (and nothing else) in the response list.	A list of all matching skills is returned.
Editing Documentation File	1. Ask the system to edit a documentation file 2. Make some changes in the documentation file 3. Check that the system is now synced with the newly changes done manually in the documentation file.	Relevant documentation file is updated and saved in the file system.

Documentation File Correctness	Repeat the following for each of the doc file types (env, am and sd), for each type, repeat with a valid file and with an invalid file. 1. Ask the system to check the correctness of the documentation file 2. Check that the system prompts with the matching errors in the documentation file (if any).	System either succeeds in finding errors if any in the documentation file, or prompts with an “OK” response that the file is correct.
Robot’s Belief State	Manually tested from the UI: 1. Ask the system for the current belief state 2. Check that the belief state’s matching what we’ve expected to receive.	Belief state matches what we’ve expected to receive according to the graphic/textual mapping (state → presentation).
Robot’s history of actions	1. Ask the system for the robot’s history of actions 2. Check that the history is matching to the actions actually executed by the robot.	System successfully returns a list of robot’s previous actions. All actions in the list are ordered synchronously by the time of execution by the robot.
Integration Requests	For each HTTP request repeat the following: 1. Ask the system to integrate with the AOS server and provide relevant input data (regarding the integration request). 2. Check that the packet was successfully received by the AOS server without any changes 3. Check that the response is received correctly without any lose of information.	Connection is established successfully with the AOS server, all packets are received by the server upon http request. All response packets are received successfully by the system.
Non-Functional Requirements		
UI	Manual tests: 1. Iterate over the UI components	Non found, client is happy with the UX. UI

	2. Look for bugs in each component. 3. Look for any non-intuitive actions that the user might take. 4. Look for any bad UX.	is intuitive to use and fast to learn.
Async requests	Manual tests: 1. Create a mock requests that sends the current process to sleep for x seconds. 2. See that we still can operate the UI without any interruptions.	System is responsive even though there’s another task running in the background.
Supporting next Ubuntu versions.	The development is done over Java’s JVM – therefore, we can be sure that this is going to work on next Ubuntu’s versions as Java’s developers will do it for us.	None.