AOS - ARD Document

Chapter 1 - Introduction

Introduction

A growing body of code for diverse robotis skills is available to roboticists. The AOS (Autonomous operating system) is based on POMDPs, generative models and probabilistic programming for integrating such code into a working autonomous robot controller that can perform diverse tasks by appropriately scheduling skill-code execution and responding to new observations.
The AOS system offers a practical contribution in the form of a true Plug and Play experience: any skill that the robot can execute, properly documented, can be used by the controller without any additional programming effort.

1.1 The Problem Domain

Nowadays the AOS users and developers need to use several different tools in order to activate the robot, documenting the robot's skills, debugging and monitoring the robot's behavior. They need to send HTTP requests to the AOS server with a 3rd party program (such as Postman), document the robot's skills manually using some text editor, and debug locally using an IDE, which makes the use of the system more cumbersome than necessary.

1.2 Context

The AOS is an already existing system, however to make a truly plug n' play solution, we have to minimize programmers' effort.
The current operating system requires handling several different sub-systems and tools, each one of them has a different API. Overall, The system is not user friendly.
Due to the disadvantage mentioned above, the system cannot be used for commercial purposes.

1.3 Vision

Our goal is to reduce the learning curve of the system, in order for the system to be widely used by the autonomous robots developers community. To do so, we will develop a one stop command and control application, which will provide all the functionalities required to use

the AOS system, and will be highly focused on usability. The system is a stand-alone system, which will be communicating with the AOS server.

1.4 Stakeholders

The product is relevant to the autonomous robots developers community, the main goal is to maximize the current system usability. Another stakeholder is the lab we are working with, the product needs to be modular, easy to maintain and flexible for future development.

Chapter 2 - Usage Scenarios

2.1 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.

2.2 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.

Alternatives 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.

2.3 Special Usage Considerations

- Since we are communicating with an external server, many problems can arise: server failure, communication to server failure, slow communication, server not responding, etc.
- Since part of the actors in our system are the AOS developers, there are several use cases that might affect (and possibly change) the implementation of our interface, such as editing the template of documentation files. There is a wide range of use cases in that aspect so it is important to consider it and implement the system properly.
- The system is of high probability of being expanded in the future (by developers on behalf of the client). Therefore, as part of our system development, we'll provide an easy, friendly way of adding the following:
    - Adding support for new integration requests to the AOS server.
    - Modifying existing integration requests (i.e adding new params to the request, etc).

Chapter 3 - Functional Requirements

1. General

	Requirement	**Priority**	**Risk**
1.1	The system must be able to activate/deactivate the AOS server	MH	Low
1.2	The system must show the status (on/off) of the AOS server	MH	Low
1.3	The system must present all the existing available projects	MH	Low
1.4	The system must support opening an existing project	MH	Low

2. Project and skills

	Requirement	**Priority**	**Risk**
2.1	The system must support creating a new project	MH	Low
2.1.1	The system will support suggesting templates for the documentation files	MH	Low
2.2	A project contains an environment file	MH	Low
2.3	A project contains a set of skills	MH	Low
2.4	The system must support adding/deleting a skill to/from an existing project	MH	Low
2.5	The system must show the available skills in a project	MH	Low
2.6	The system must support persistency of new created projects	MH	Low

3. Documentation files

	Requirement	Priority	Risk
3.1.1	The system must support editing the documentation files via third party IDE	MH	High
3.1.2	The system must support editing the documentation files via the application itself	MH	High
3.2	The system must support checking the correctness of documentation files	MH	Low
3.3	The system must present the exact place in which the syntax/compilation error occurred in the specific documentation file	MH	Mid

4. Robot actions and states

	Requirement	Priority	Risk
4.1	The system must be able to present the robot's belief state after each action	MH	High
4.2	The system should present the robot's state in a graphical manner	MH	High
4.3	The system must present the history of the robot's actions throughout the execution process	MH	Low
4.4	The system must support requests to stop an inner simulation	MH	Low
4.5	The system must support requests to stop the robot	MH	Low

5. Integration with the AOS Server

	Requirement	**Priority**	**Risk**
5.1	The system must support an integration request to activate the robot - generate the code and build.	MH	Low
5.2	The system must support an integration request to only generate the code	MH	Low
5.3.1	The system must support an integration request to activate the robot without rebuilding the project's files.	MH	Low
5.3.2	The system must support an integration request to activate an inner simulation of the robot	MH	Low
5.4	The system must support a project's documentation files correction check request which is supported by the AOS server API	MH	Low
5.5	The system must support opening the output generated file by clicking on the documentation file it was generated from	MH	Mid
5.6.1	The system must support presenting compilation errors in the decision engine generated code	MH	Mid
5.6.2	The system must support opening the generated file by clicking on an error in the specific place where the error occurred.	MH	Mid

Chapter 4 - Non-Functional Requirements

	Requirement	**Priority**	**Risk**
1	The system must introduce an easy, usable graphic user interface	MH	High
2	The system must run on Ubuntu 20.04 OS and later updates	MH	Mid
3	The system should create a friendly graphical mapping of a robot's belief state	NTH	High
4	The system must support running on the localhost	MH	Low
5	The system should introduce a generic template for users to define the structure of their documentation files	NTH	Mid
6	System's GUI must be easy for maintenance as the project will be definitely expanded on behalf of the costumer	MH	Mid
7	The system must support asynchronous requests	MH	Low
8	The system will persist each new project in a unique folder	MH	Low

Chapter 5 - Risk assessment & Plan for the proof of concept

The risk assessments on parts of the project that will be challenging to develop:

- Communicating with local processes such as VSCode in order to open files in specific lines (opening generated file).
- Choosing the right framework for the GUI implementation which would allow us to best implement the needed controllers in a usable, friendly manner. Among are:
    - Editing and creating documentation files
        - Requires us also to create a general template which users can define for creating documentation files for each project.
    - Presenting robot's state
    - Presenting robot's action history
- Integrate all new technologies that will be used in the system
- Building a module that allows users to easily define a mapping from robot's state to graphical representation.

The focus in our prototype would be on the first 3 bullets above, trying to reduce the risk of project's failure.

Prototype description

- The prototype would be a desktop application, running on an Ubuntu OS together with the AOS server
- Firstly, the client would be a dummy, with minimal functionality, mainly demonstrating the following abilities:
    - All relevant controllers for GUI development - mainly for creating/editing documentation files
    - The ability to open a file in a specific line in the local file system in VSCode
- Secondly, the application will mainly demonstrate basic functionality like:
    - Documentation files checker
    - Documentation files generator
    - Communication with the AOS server
- Lastly, we'll provide a graphical representation of the robot's belief state.

- By providing the above prototype, we'll reduce dramatically the risk of project's failure.
    - First, this is demonstrating that we've chosen the right technologies for the project which would save us a lot of development time later on.
    - Second, these are some of the main core functionalities that our customer has declared are most important.

Glossary

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.