Fundamentals of Adaptive Software - SUAVE
Exemplar Interface

Fundamentals of Adaptive Software 2023

Group: 3_2
Member Names: [redacted]
Emails: [redacted]
VUnetIDs: [redacted]
Master Program and Track: [redacted]

November 12, 2023

Contents

1 Introduction **2**
1.1 UPISAS . . . 2
1.2 SUAVE . . . 2

2 Methodology **3**
2.1 Design of API Endpoints . . . 3
2.1.1 Monitor . . . 3
2.1.2 Execute . . . 3
2.1.3 Adaptation Options . . . 4

3 Implementation **5**
3.1 UPISAS . . . 5
3.2 SUAVE . . . 5
3.2.1 Technical Difficulties . . . 6

4 Testing **7**
4.1 Manual Testing . . . 7
4.2 Automated Testing . . . 7

5 Discussion **8**
5.1 Feasibility of the Project in Real-world . . . 8
5.2 Future Work . . . 8

A Division of Work **9**

1 Introduction

The Self-adaptive Underwater Autonomous Vehicle Exemplar (SUAVE) is a self-adaptive service-based system exemplar that performs inspections on pipelines that are underwater. In this assignment, we propose a way to externally control the system.

All links to the related code repository can be found at the beginning of section 3.

1.1 UPISAS

UPISAS (Unified Python Interface for Self-adaptive Systems) is a framework for self-adaptive software. It works by collecting information or intelligence from the self-adaptive system through certain API interface, judging with its knowledge-base, then make adaptation requests through the API interface.

More specifically, the interface used in this project is HTTP interface.

1.2 SUAVE

SUAVE is a research project for Self-Adaptive Underwater Vehicles performing pipeline inspection. This project uses ROS2 (Robot Operating System 2) as it's main controlling framework.

In this project, an extension over the system is created by implementing an HTTP server which exposes an API.

2 Methodology

2.1 Design of API Endpoints

The three tables below briefly describes the design of three HTTP APIs for reporting information or requesting adaptations. It is worth noting that there are other three APIs which are not listed in the table below:

- `/monitor_schema (GET)`
- `/execute_schema (GET)`
- `/adaptation_options_schema (GET)`

These APIs are mainly for the requesting composing and response validation purposes. Therefore, they will not be detailed in the following tables.

2.1.1 Monitor

Field	Description
Description	This endpoint is used to monitor the diagnostic information for this exemplar. Specifically, the monitorable information exposed by the SUAVE system are: • Status of the six thrusters (true or false) • Visibility of the water (a float value)
Endpoint	`/monitor`
Allowed HTTP Method	`GET`
Sample Response Body	`{ "thrusters": { "c_thruster_1": true, "c_thruster_2": true, "c_thruster_3": true, "c_thruster_4": true, "c_thruster_5": true, "c_thruster_6": true }, "water_visibility": 1.4887287570312693 }`
Sample Return Status	`200 OK`

2.1.2 Execute

Field	Description
Description	This endpoint is used to execute various adaptations in the system. Meanwhile, the corresponding options and type of adaptation are specified in the request body.
Endpoint	`/execute`
Allowed HTTP Method	`PUT`
Sample Request Body	`{"adaptation": "/task/cancel", "option": "search_pipeline"}`
Sample Return Status	`200 OK`

2.1.3 Adaptation Options

Field	Description
Description	This endpoint is used to retrieve all the possible adaptations of the system. The adaptations of the system can be categorized into the following: • Task control (`/task/cancel` and `/task/request`) • Mode of motion control (`/f_maintain_motion/change_mode`) • Mode of searching path (`/f_generate_search_path/change_mode`) • Mode of pipeline following (`/f_follow_pipeline/change_mode`)
Endpoint	`/adaptation_options`
Allowed HTTP Method	`GET`
Sample Response Body	`{ "tasks": { "adaptation_options": [ "search_pipeline", "inspect_pipeline" ], "operations": [ "/task/cancel", "/task/request" ] }, ... }`
Sample Return Status	`200 OK`

3 Implementation

The following implementation analysis corresponds to the following source code:

- UPISAS
  - Repository: [redacted]
  - Branch: `assignment-1`
  - Commit: `latest`
- SUAVE
  - Repository: [redacted]
  - Branch: `assignment-1`
  - Commit: `latest`

3.1 UPISAS

For UPISAS, the main implementation focuses on the initialization of SUAVE system (i.e. spawning working Docker container).

3.2 SUAVE

For SUAVE, the main implementation focuses on the HTTP API. To better integrate into the SUAVE system, the HTTP server and request handling is encapsulated within a ROS node.

Internally, the node subscribes to the topic `/diagnostics` and filter the corresponding sources for the real-time diagnostic and monitoring data.

```
        self.diagnostics_sub = self.create_subscription(
            DiagnosticArray,
            '/diagnostics',
            self.diagnostics_cb,
            10
        )
```

For all the adaptive options, ROS service clients are created for sending requests to corresponding services.

```
        self.task_request_client = self.create_client(Task, 'task/request')
        self.task_cancel_client = self.create_client(Task, 'task/cancel')
        self.task_req = Task.Request()

        self.f_maintain_motion_client = self.create_client(ChangeMode,
        ↪  'f_maintain_motion/change_mode')
        self.f_generate_search_path_client = self.create_client(ChangeMode,
        ↪  'f_generate_search_path/change_mode')
        self.f_follow_pipeline_client = self.create_client(ChangeMode,
        ↪  'f_follow_pipeline/change_mode')
        self.change_mode_req = ChangeMode.Request()
```

3.2.1 Technical Difficulties

During the implementation of the SUAVE part, the major difficulty is threading and synchronization. ROS in Python uses `rclpy`, which is essentially a Python binding of the C++ implementation of ROS ecosystem. This means the threading model of ROS in Python in essentially the same as the one that ROS used in C++, which is native threading.

However, in the design of our implementation, the Python-native HTTP library `Flask` is used, which uses Python's native threading model. Inevitably, these two components caused threading compatibility issue.

Our solution to the problem is separating the two components, resulting in adding locks for both components in the critical part of the code.

```
                    if value.key == "water_visibility":
                        with self.lock:
                            self.buffer[str(value.key)] = float(value.value)
```

This might result in performance trade-off during high-concurrent scenario. However, it is necessary safety measure for the stability of the system.

4 Testing

4.1 Manual Testing

Manual testing was done by inspecting the real-time logging of the SUAVE system to ensure that all received external monitoring or control commands are executed correctly.

4.2 Automated Testing

The automated testing is carried out by the updated unit test script within UPISAS. The test script mainly runs through all the endpoints and verifies:

- The reachability of all the endpoints
- The validity of the responses from these endpoints

Eventually, the unit test script showed all tests were passed.

5 Discussion

5.1 Feasibility of the Project in Real-world

In this assignment, it is required to implement an HTTP API to expose part of the SUAVE's system control to a higher-level adaptive controller (UPISAS). This could be feasible in the context of other self-adaptive systems, such as Simulator of Web Infrastructure and Management (SWIM); However, in the context of robotics systems, where real-time responsiveness (ideally, within microseconds) is a strict requirement, an extra HTTP overhead would be far less than ideal. Much latency and overhead could be involved in such design:

- Latency of the TCP protocol itself (three-way hand-shaking, management of stateful connections, etc.)
- Latency and overhead introduced by the HTTP server (in our case, by using `Flask` library)

Therefore, the feasibility of the project in reality remains questionable. Is is recommended to specialize and integrate UPISAS as an external node of the ROS system to achieve the best possible performance and accuracy of real-time data.

5.2 Future Work

Though the major part of the implementation is finished, there is still room for improvements:

- The handling process of HTTP API requests can be further simplified to improve code readability, especially for `adaptation_options`.
- The launch process of the HTTP server can be integrated into the unified ROS launch specification.
- The implementation of asynchronous ROS service client can be better improved.

A Division of Work

Name	Division of Work
[redacted]	Project analysis, Implementation of SUAVE interface, Writing of report.
[redacted]	Project analysis, Implementation of SUAVE interface, Writing of report.
[redacted]	Project analysis, Implementation of SUAVE interface, Writing of report.
[redacted]	Project analysis, Implementation of SUAVE interface, Implementation of UP-ISAS, Writing of report.