Towards Robotic Laboratory Automation Plug & Play: LAPP Reference Implementation with the TIAGo Mobile Manipulator

Panna Zsoldos[a,*], Ádám Wolf[b,c], Károly Széll[d], Péter Galambos[a]

[a] *Antal Bejczy Center for Intelligent Robotics, Óbuda University, Budapest, Hungary*
[b] *Baxalta Innovations GmbH, a Takeda company, Wien, Austria*
[c] *Doctoral School of Applied Informatics and Applied Mathematics, Óbuda University, Budapest, Hungary*
[d] *Alba Regia Technical Faculty, Óbuda University, Székesfehérvár, Hungary*

Abstract

Laboratory automation with mobile manipulators presents a promising avenue for enhancing efficiency and reducing the dependency on human activity in laboratory workflows. To achieve seamless integration, the Laboratory Automation Plag & Play concept has been introduced. This study presents an experimental implementation of the LAPP concept through the development of labware transfer functionality for a mobile manipulator employing the SiLA and ROS frameworks as widely accepted control layers. While validating the feasibility of the LAPP concept, this reference implementation accentuates the need for advancements in both hardware and software environments to fully capitalize on its benefits in contemporary laboratory settings.

Keywords: Laboratory automation, Mobile robotics, Autonomous manipulation, System integration, Plug and Play, Digital Twin

1. Introduction

Recent technological advancements and the growing demand for high-throughput research processes have spurred a significant evolution in laboratory automation across various industries. A pivotal industry in this regard is the pharmaceutical sector. [1, 2]. This evolution encompasses a diverse array of cutting-edge technologies, sophisticated processes, and advanced software solutions aimed at minimizing dependency on human intervention. This includes state-of-the-art automated equipment such as liquid handlers, centrifuges, robotic labware handlers, and conveyor belts, complemented by specialized software

*Corresponding author

Email addresses: panna.zsoldos@irob.uni-obuda.hu (Panna Zsoldos), adam.wolf@takeda.com (Ádám Wolf)

tailored for seamless workflow management, precise instrument control, and efficient data acquisition. The seamless integration of automation holds paramount importance, particularly in high throughput laboratories where the routine processing and screening of large sample volumes are fundamental tasks.

Diverse laboratory requirements have resulted in varied types and degrees of laboratory automation. Since there is no single definition for laboratory automation classification, we mention two viewpoints to categorize automation levels. First is a generally accepted classification method, exploring the complexity of instrument integration, scaling from no automation through partial laboratory automation to Total Laboratory Automation (TLA), as a spectrum [3]. In the *no automation* scenario, all tasks and processes are conducted manually, whereas in *TLA*, various analyzers are physically integrated into a modular system resembling an assembly line. *Partial automation* exists between these extremes, involving automated tasks, sub-tasks, or parts of sample handling processes. However, even in partially automated labs, manual transportation between processing stations still requires involvement from lab workers [4].

An alternative classification approach is drawn from the realm of autonomous vehicles, specifically the standard SAE J3016 [5] which has been created for straightforward classification of automation levels, commonly referred to as *Level of Autonomy (LOA)*. This six-level scale has been adopted by other fields, including surgical robotics [6]. Building on this trend, a comparable six-level scale has been proposed for biological laboratories [7].

One of the earliest laboratories using robots for automation was created in Japan in the 1980s in a medical school [8]. This laboratory utilized multiple tube-handling robots and a conveyor belt, which transported patient samples to various analytical workstations and automated pipetting robots. At some workstations, single-arm stationary robots performed pipetting and dosing steps to carry out more complex analytical tasks.

Today, automation has already been relatively widespread in biotechnology and pharmaceutical research, especially in high-throughput applications. Automatic pipetting robots and liquid handlers, are commonly used for this purpose and are considered as relatively mature technology [9]. At the same time, clinical laboratories are significantly slower in adopting robotic automation. Most medium-sized hospital laboratories, which typically handle no more than 2,500 samples a day, have found it difficult to justify the investment in multi-million dollar systems. On the contrary, pharmaceutical companies have made substantial investments in robotic automation to enhance their drug discovery rates, alongside investments in accelerating research in manufacturing technology. Introducing automated sample screening caused a three to five times increase in the number of samples screened per unit of time. As these companies are earning extra profit from new drugs, justifying the costs is easier [8].

When considering the dimensions of the Return on investment (ROI) model, laboratory automation presents advantages that align with key factors. Lower long-term costs contribute to the financial dimension of ROI, while increased efficiency and reduced turnaround time (TAT) impact the time dimension by enhancing productivity and yielding higher returns over time. Enabling overnight

and weekend operation enhance the utilization dimension and maximize operational efficiency. Improved traceability of samples and reduced risks associated with bio-hazardous materials address the quality and risk dimensions, respectively, ensuring reliability and compliance. Conversely, limitations such as high initial costs and increased maintenance expenses affect the cost dimension, underscoring the need for careful cost-benefit analysis. Moreover, potential dependence on automation during downtime raises concerns related to the risk dimension, emphasizing the importance of resilience and contingency planning. Evaluated within the ROI framework, laboratory automation initiatives enable organizations to assess financial viability while minimizing risks and optimizing returns.

Various tasks in laboratory environments require different types of instrumentation and spatial considerations, necessitating the utilization of specific types of robots tailored to each task. Solutions for these tasks often involve employing either *task-specific machines*, such as pipetting robots designed for precise liquid handling, or *general-purpose robots* like robotic manipulators capable of versatile manipulation. Common tasks in laboratories include handling liquid samples, transporting labware, and operating equipment such as pushing buttons or opening doors. While these tasks may have low added value, they are essential connective steps in the overall laboratory process [10]. The adoption of *general-purpose robots* in laboratory settings is becoming increasingly common, replacing the need for task-specific equipment. This trend offers numerous advantages, including the versatility of robots and the ability to program them to operate equipment originally designed for human use. Research advocating for Self-Driving Labs (SDL) illustrates the potential benefits of this approach [11] [12]. This way they can perform a broad range of different tasks without having to modify their surroundings. Also, if a laboratory changes its purpose drastically, the robot can be reprogrammed to fulfill new requirements seamlessly.

In selecting robots for laboratory applications, mobility is a crucial consideration, encompassing *fixed-base* and *mobile* categories. *Fixed-base robot arms* operate within a confined local area, with a myriad of options available on the market, necessitating careful selection tailored to specific applications. Variations are evident in structural design (e.g., Cartesian, cylindrical, spherical, SCARA, parallel), joint configuration, degrees of freedom, gripper compatibility, programming capabilities (offline/online), human collaboration potential, and cost. Leading manufacturers like FANUC, KUKA, ABB, Omron, and Universal Robots primarily target industrial or general tasks. However, there are specialized robots tailored for laboratory automation, such as PreciseFlex devices [13], optimized for benchtop applications with integrated safety features. The design of PreciseFlex robots facilitates stationary sample transportation, particularly beneficial due to the SCARA configuration's mechanical constraints on end-effector orientation along the roll and pitch axes, alongside simplified control and a robust mechanical structure. This design ensures objects are manipulated parallel to the ground, a necessity for tasks involving open liquid containers such as microplates.

For tasks requiring transportation over longer distances, such as moving samples between separate equipment stations, manual handling is the default approach. However, this can be addressed with *autonomous mobile robots* (AMRs). Typically equipped with manipulators, these robots, known as *mobile manipulators* (MoMas), offer versatile functionality while navigating through dynamic workspaces. Unlike stationary counterparts, MoMas operate in undefined environments, adapting to changing conditions and expanding workspaces. Choosing the right MoMa involves considering two main factors: mobility type, which determines the base structure and operational range, and the type of manipulation, which dictates the gripper used. Options include non-steerable, fixed, orientable, or omnidirectional wheels for mobility [14], and various gripper types based on the manipulation required, the most common is the parallel gripper. Multiple mobile manipulators are available on the market, such as Kevin by Fraunhofer [15], Biosero's mobile robots [16], Astech Projects' Robotic Lab Assistant [17], and United Robotics' uMobileLab [18]. The use of mobile robots in laboratories is still in its infancy, but their integration could lead to the full automation of life science laboratories [19].

The state-of-the-art laboratory MoMa solutions lack a standardized integration framework, which would enhance their interoperability. In our paper, we present a reference implementation for integrating a bi-manual mobile manipulator. For this purpose, we utilize the Laboratory Automation Plug & Play (LAPP) framework, the SiLA2 communication protocol, its unified labware transfer feature definitions, the Robot Operating System (ROS) and the SiLA-ROS bridge reference implementation.

The paper is organized as follows: Section 2 presents the technologies, standards and frameworks on which the solution focuses. In Section 3, our contribution and prior work on popularizing the adopted standard is specified. Section 4 details all steps of the implementation in both hardware and software environments as well as a higher level and generalized overview of the task at hand. Testing and results on the setup's functionality is revealed in Section 5. Results are discussed in Section 6, possibilities of further development are demonstrated in Section 7. Final thoughts and conclusions are summarised in Section 8.

2. Technological context

This section discusses the challenges and solutions in integrating liquid handling robots and other analytical instruments in laboratory automation. It introduces the SiLA Consortium's efforts to establish a standardized communication protocol (SiLA2) and the Laboratory Automation Plug & Play (LAPP) framework [1] for seamless integration of lab robots. The LAPP framework emphasizes a hierarchical system architecture and introduces the concept of digital twins for teaching-free integration.

2.1. SiLA

Analytical instruments and liquid handling robots are commonly used in traditional pharmaceutical research laboratories, often as separate entities. The integration of instruments can result in improved efficiency and continuous operation. Although, several factors hinder the cost-effective and rapid integration of these instruments into fully automated systems. High costs, long implementation times, and inflexible systems are all barriers to entry for smaller laboratories. An automated laboratory may include instruments for sample preparation and analysis, such as incubators, pipettors, dispensers, and plate readers. These instruments are linked to a software system that streamlines the workflow from set-up, but without the use of a standardized communication protocol, they have a number of disadvantages. These disadvantages include:

- different interfaces for each device
- completely different command sets for devices with the same or similar functions
- significant differences in timing and structural behavior
- different status descriptions for each device
- huge differences in error messages and error handling capabilities of the devices

Many device manufacturers have developed their own internal proprietary standards and application programming interfaces (APIs) for a wide range of devices. Many system integrators would create drivers in the form of wrappers around these APIs, to enable the integration of the device with their scheduler. However, integrator-proprietary communication protocols limit the compatibility to clients (e.g., schedulers) of the specific integrator. In contrary, implementing a driver around the device API using standardized communication protocols ensure compatibility with a broader range of clients. The development of a standard interface could eliminate the need for repetitive work in this area.

A solution for this problem was developed by the Standardisation in Lab Automation (SiLA) Consortium [20] [21]. The organization consists of software suppliers, system integrators and users, such as pharmaceutical and biotech companies and academical institutes. SiLA has created an emerging standard by the name SiLA2, which aims to define the interface between laboratory instruments and the automation software system [22].

The ultimate and simplest solution would be, if device vendors used standardized protocols *instead of* proprietary APIs. Reducing the number of interfaces in the chain of connections maximizes simplicity and robustness.

Throughout the rest of this paper, we will use the abbreviation SiLA to refer to the communication protocol SiLA2, developed by the Consortium.

In the SiLA-ecosystem functionalities and capabilities of lab devices are represented as feature definitions in XML format. Although there are conventions for creating feature definitions, there are still a few possible ways of defining them for each device or laboratory. This results in differing definitions among different companies. Examples of this are the feature definitions for MoMas from the Fraunhofer Institute for the Kevin robot [23] or from Astech Projects [24].

Every feature definition can consist of commands and properties, for using basic or more high-level and complex tasks. Properties are usually responsible for returning read-only values through SiLA calls, while commands usually start/stop a process, set values, and other functions that require some modification.

The main limitation regarding these solutions from different vendors appears to be the lack of a harmonized integration blueprint. To make device integration easier, we proposed the Lab Automation Plug & Play (LAPP) concept [1, 2, 25].

2.2. LAPP

The Laboratory Automation Plug & Play (LAPP) framework outlines an overarching blueprint for the integration of supportive lab robots within the context of life science laboratory automation. The LAPP frameworks spans an agnostic conceptual Reference Architecture Model (RAM), covering the entire vertical of lab robot integration. By identifying well-established building blocks, such as the Robot Operating System, and the SiLA communication protocol, LAPP is also a collection of best practices [26]. Centered around the labware transfer capability of mobile manipulators, the LAPP concept maps out a hierarchical multi-layer system architecture, which reflects the decomposition of the workflows into Ⓣ Tasks, Ⓢ Subtasks and low-level robotic activity representations (RARs) [25] (this issue).

As a crucial part of the LAPP framework, we proposed the digital twin concept (DT) [27]. With the LAPP-DT pre-existing geometrical information of the laboratory devices enables the teaching-free integration of robots.

In this paper we present a prototype implementation of the LAPP Reference Architecture Model, utilizing a research platform MoMa. As such, the present work represents a reference implementation, which binds concrete technological building blocks to the concepts outlined in the previous papers.

3. Preliminary work in the SiLA Robotics Working Group

The SiLA Robotics Working Group (SRWG) [1] is a domain-specific group of subject-matter experts (SMEs). The SRWG's key work packages include the assessment of user needs in the labs, and identifying present and future robotic capabilities that can answer these needs. We published the results of this endeavor in [25] (this issue). We identified the labware transfer capability to be the highest requested item. It is also relatively well-covered already with

[1]The SRWG is lead by Ádám Wolf since March 2022 and as of the time of writing this present paper.

state-of-the-art fixed-base and mobile robotic solutions. We also condensed down the additional robotic capabilities - the ones which are not yet addressed by mature off-the-shelf robotic solutions. As a result, we created a categorized system of namespaces and the stubs for each activity [28].

The next step taken by the SRWG was to unify the feature definition for labware transportation. It has been made available on the SiLA GitLab site [29]. An important standpoint always kept in mind throughout development is the device agnostic approach, meaning that a feature, including its commands and properties, should be compatible with any robot or device with the same functionalities, regardless of the manufacturer.

The current version of the unified labware transfer features can be found on the SiLA GitLab repository [30]. The features are divided into two practices: one for devices which are used for manipulating object (called *active* devices); and one for *passive* lab equipment. Other design principles during definition creation were:

- to be vendor agnostic, so the usage does not depend on the manufacturer,
- to not be dependent on whether the robot is stationary or mobile,
- to be modular with distinct categories,
- to have a high-level and a low-level implementation possibility.

After the feature had been unified, the SRWG created reference implementations for stationary robots, as a part of the BioLAGO SiLA 2 AniML Serial Hackathon (BioSASH). This was a series of hackathons co-organized by BioLAGO [31] and the SiLA Consortium. These events gave an opportunity for people from different fields to work together in hopes of finding new solutions to solve common challenges in process automation.

During the third installment of this series, the goal of connecting the SiLA communication to the robots' driver software was set out. The result can be found on GitLab [32] as the SiLA-ROS bridge. This solution is further discussed in Section 4.7.

The fourth installment of BioSASH took place at the end of September 2021 in Konstanz, Germany. One of the five working groups set the goal to "SiLA-fy" the labware transportation process for stationary robots [33]. Also presented as part of the event were a general, example implementation of a labware transportation between two devices, a destination and a source equipment, e.g., liquid handlers and analytical instruments. As a basis, the group used the `LabwareTran⌋sferManipulatorController` and `LabwareTransferSite-Controller` feature definitions [34]. A working implementation was delivered for the Universal Robots UR3 and a PreciseFlex 3400 robotic arm, performing a pick-and-place task from one robot to another through a shared station between them. This demo helped with constructing the *unified feature definition* for labware transfer tasks. Implementation results can be reached as a GitLab project [35] and the final presentation, given at the end of the event, can be reached here [36].

These steps make it possible to create implementations of the LAPP concept for ROS-based MoMas using the feature definitions and the connecting bridge.

4. Implementing Labware Transfer Functionality for Mobile Manipulators

The goal of this preliminary study is to demonstrate the possible usage of the LAPP model on a MoMa in a real-world setting. We carried out the reference implementation on a PAL Robotics TIAGo++ unit (from now on, *TIAGo* or *robot*). The project was aimed at implementing a pick-and-place labware transfer functionality between two stations for microplates[2]. This is a typical task in a laboratory workflow that renders a feasible job for a robot. The complete operation consists of the following generalized steps.

- Undock from Docking Station
- Drive to point of interest (PoI) Station A
- Pick up the object from Station A object holder site
- Place the object to the onboard holder
- Drive to PoI Station B
- Place the object to Station B object holder site
- Drive to Docking Station PoI
- Dock, if charging is needed

In the rest of the section, a more detailed sequence will be described in the context of architectural aspects and implementation details.

4.1. TIAGo

The TIAGo++ robot was developed by the Barcelona-based PAL Robotics company, primarily focusing on providing robotic solutions for research purposes. PAL Robotics makes multiple variations of TIAGo robots with different hardware and software configurations as well as a humanoid robots and a quadruped [37].

The most notable parts of TIAGo are the double manipulator arms of 7 degrees of freedom (DoF) each, with parallel grippers. These features provide outstanding flexibility when moving objects. The lifting torso makes it possible to operate in an extended vertical range.

The robot's controller runs Ubuntu 18.04 OS with Robot Operating System (ROS) as the middleware for using robot-specific functionalities.

ROS is a meta-operating system incorporating a broad collection of functional robot software libraries. In research projects, ROS is the industry standard for

[2]Meets the Standards ANSI/SLAS 1-2004 through ANSI/SLAS 4-2004.

developing control software. Decisive factors for selecting TIAGo as the subject of this project were that ROS is open-source, well-documented, and driven by a dedicated community [38].

Although the immense documentation by PAL Robotics and their availability through the support website proved to be adequate, TIAGo exhibited shortcomings, emerging mostly during autonomous navigation. This seemed to be caused by the insufficient functionality of the suspension in the actuation mechanism and the four free wheels making contact with the uneven floor of our lab.

4.2. Hardware customizations

The robot's default end effector is a parallel gripper with two independently movable fingers. The whole end effector and fingers separately are interchangeable components. The original gripper fingers were considered too narrow to be able to grasp the microplates, especially along its longer side. To overcome this, we made a set of custom 3D printed fingers, which can be fixed to the end effectors as shown in Figure 1. The fingertips are covered with silicon sheet cutouts for a better grip.

Figure 1: Images of Station A (1) and Station B (2) and the custom-made gripper, both in open (3) and closed (4) positions

There is also an additional 3D-printed microplate holder on the front of the robot's base. This serves as a *hotel*, which is used for storing the object during the driving between the two stations, resulting in safer transportation.

To mimic the real-world labware transfer situation, two experimental stations (dummy sites) have been built using laser cutting and 3D printing (Figure 1). Each site includes a mechanical fixture that holds the microplates in place and a visual marker (ArUco) that serves as an anchor for visual navigation. The fixtures are made with suitably selected tolerances that are important both when picking and placing the objects.

4.3. System architecture

The proposed architecture is based on the integration of ROS and SiLA concepts. Figure 2 describes the connection between the ROS and SiLA ecosystems at the level of the main components. The contract for the communication between the SiLA server and SiLA client is the *feature definition*. The bridge represents the SiLA-ROS bridge package [39], which is a reference implementation published on the SiLA GitLab [32]. On the ROS side, communication is managed through the standard concepts: topics, services, and actions. Further descriptions of SiLA client, SiLA-ROS bridge, ROS action server, and feature implementation components are detailed in 4.7.

Two main ROS nodes have been implemented: one handles the ArUco marker-based arm coordination, while the other one is responsible for the execution of the transportation sequence. These two nodes correspond to an action server written in C++ and an action client which is a Python program. Both are embedded as ROS nodes into the robot's functionalities which are initiated at startup automatically. A detailed explanation will follow in Sections 4.6 and 4.5.

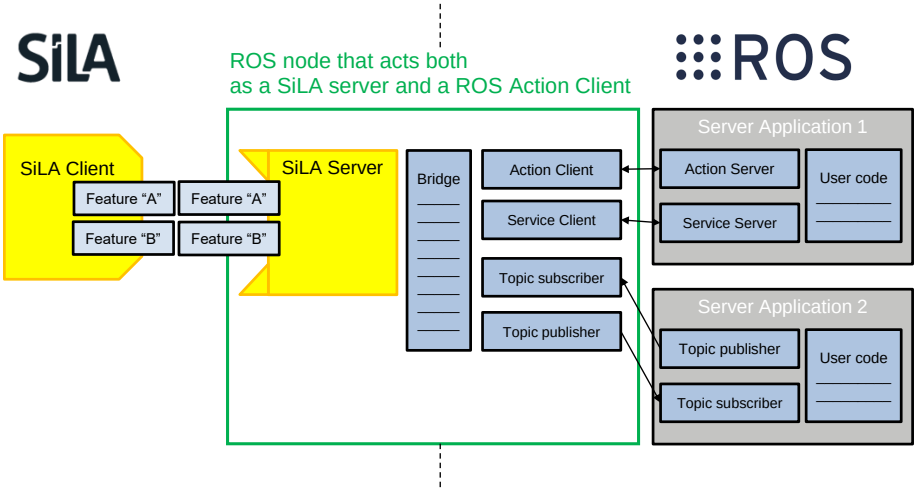

Figure 2: Conteptual software architecture of the SiLA-ROS hybrid system.

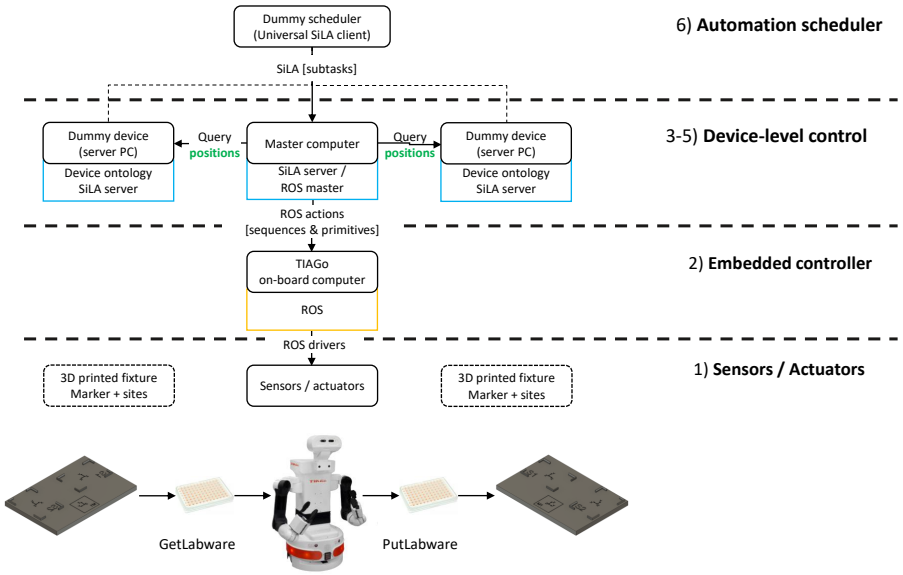

Figure 3: Architecture by layers, according to [25] (this issue): 1) Sensors/Actuators layer, 2) Embedded controller layer, 3-5) Device-level control layers, 6) Automation scheduler layer. The ROS component is outlined with yellow, and the SiLA components with blue.

In Figure 3 a layered view of the system architecture can be seen, using the numbering that we introduced in [25] (this issue). The layers span from the low-level actuator and sensor control through an embedded controller (ROS), a device-level controller (SiLA) to an automation scheduler (Universal SiLA client, USC). On the device-level control layer, the Digital Twin (DT) of a generalized (dummy) device is depicted. In detail we discuss the canonical layered architecture in [25] (this issue). The representation here is an example of its implementation, using specific technological building blocks.

4.4. Technology mapping

To map the relevant ROS and SiLA functionalities to the LAPP concept, we created a UML-like diagram, seen in Figure 4. We created this representation with different levels of RARs in relation to each other with color and letter coding. Texts in blue are depictions of SiLA features, commands, or attributes, while yellow text represents an implementation as ROS functions, all in correlation to the LAPP representation. The UML namespaces represent a categorization of RARs. The T Task- and S Subtask-level RARs represent outcome-oriented activities, which we categorize based on the function. In contrast, Q Motion sequence, P Motion primitive, and A Actuator primitive RARs are considered to be low-level activities, which are, in most cases, masked away from the scheduler. We categorized these based on the robot components. In [25] (this issue) presented a structure for the extended taxonomy of outcome-oriented RARs [28].

A detailed sequence diagram of the labware transfer process between a fridge and a pipettor device is presented in the Supplementary material (S1). The steps correspond to the RARs displayed in Figure 4, matching with the color coding. It can be seen that a (T) Task (red) is manifested as the entire labware transfer feature, while (S) Subtasks (orange) are distinct SiLA commands, which are, in a general case, issued by the scheduler. (Q) Motion sequences (yellow) are implemented on the robot's middleware (ROS) and are not exposed directly via SiLA. In the figure, these are represented as self-calls on the components' side.

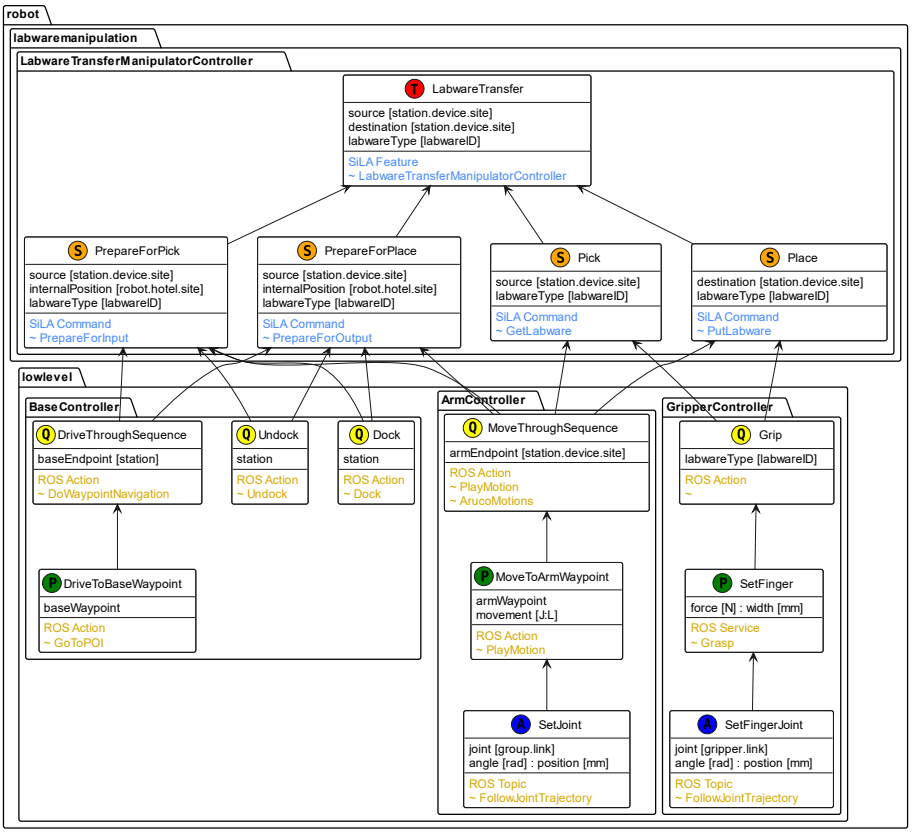

Figure 4: UML-like representation of the labware transfer RARs. Namespaces represent a way of categorizing the RARs. The arrows represent how a high-level RAR is composed (broken down into) low-level RARs. Task (T), Subtask (S), Motion sequence (Q), Motion primitive (P) and Actuator primitive (A) are represented as classes. Parameters are shown as class members, with the data type in square brackets []. Overloaded parameters (in the case of polymorphic RARs) are separated by a colon (:). Mapping the SiLA implementation is set in blue font, while the ROS implementation is in yellow. The names of the specific ROS functionalities are marked by a tilde (∼)

4.5. Action server implementation

As outlined in Section 4.4, we implemented the Arm Motion Sequence Q `MoveThroughSequence` in the form of a ROS action server, written in C++. The name of the node is `ArucoMotions`, referring to the functionality it accomplishes. This action server is started during the robot's startup process with the other default startup nodes.

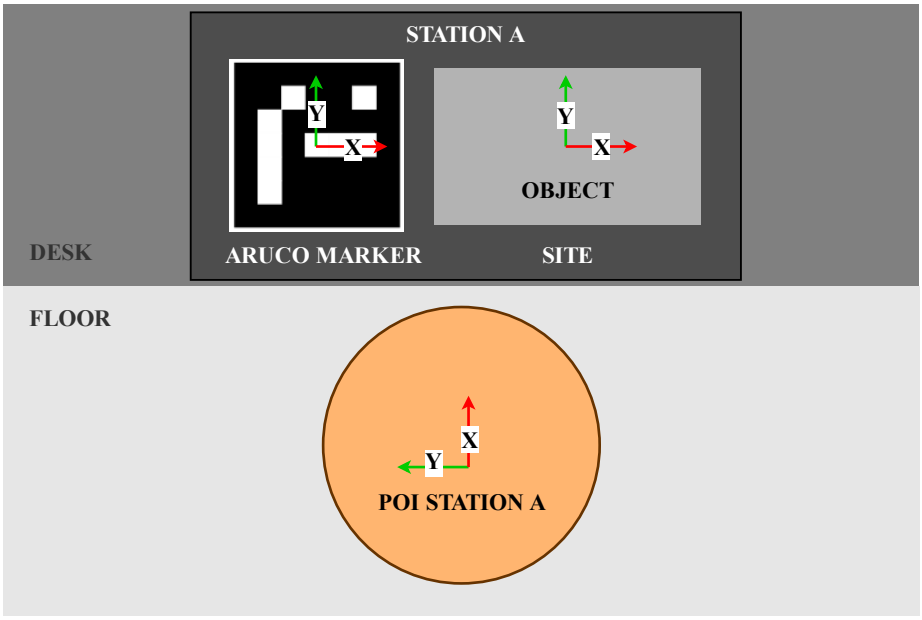

Figure 5: Close-up diagram of a station. The diagram represents a top-down view of a station, including a POI position on the floor (yellow circle), the dummy device itself on top of the desk (dark grey), including a marker and a site (light grey), where the manipulated object can be placed. The coordinate frame at the POI represents the base of the robot, in the correct direction, when it is doing the manipulation. The frame seen on the marker is placed in the direction as the robot's head camera sees it. The frame of the site is expressed as a translation in X and Y, relative to the marker's frame. All three frames have red axes for X, and green for Y.

The node handles the detection of the ArUco marker and the derivation of the site (s) and the site approach (ss) positions as transformations from the marker's frame. It also executes a MoveIt [40] based motion to move the tool center point (TCP) to those frames. The two dedicated frames are directly above the microplate holder slot at two different distances from the plate itself. (ss) is used for approaching, while (s) is for the final position which is used when grabbing the plate. Both are expressed relative to the marker, with translations along the x, y, and z axes.

When the upcoming step of the overall pick-and-place sequence is an ArUco-based motion, this node checks if there is a marker on the head camera's image. If not, it throws an error, if there is one, it calculates the transformed frames and

sends a goal position through the C++ MoveIt interface. While this executes, the rest of the main sequence (seen in Section 4.6) is blocked. Every time the main sequence calls this node and there is a marker in sight, the motion starts.

Using the naming conventions of [2], the coordinate frames, seen in 5 are defined as follows:

- POI STATION A (PoI) - base pose for the station
- ARUCO MARKER (m) - fiducial marker for device
- SITE (s) - hand-over site (plate nest) of device
- the site approach (ss) and device approach (sd) frames are the same in this instance since a device includes only one site

4.6. The Sequence

We implemented the sequence in the form of a ROS action client. The node `transportation`(`transportationScript.py`) handles the sequence steps as a Python script. In [2] we introduced a generalized sequence, for which a detailed implementation is presented in this paper.

In the supplementary table S2 we present how specific ROS-based functions implement each RAR in the canonical sequence. Moreover, we list the digital twin parameters for each function. In a LAPP-compliant system, these parameters are represented in the digital twin instance of the corresponding components.

The three main arm configurations are shown in Figure 6: *Home*, *Benchtop* and *Nest* from left to right. These presets make it possible to repeat motions between these configurations as desired. For declaring these motions within the framework, we used the ROS package `play_motion_builder`, which was built on MoveIt and was provided with the robot. The `play_motion` package is a tool for executing upper body motions using pre-defined keyframes with the help of MoveIt on-the-fly trajectory planning capabilities [41].

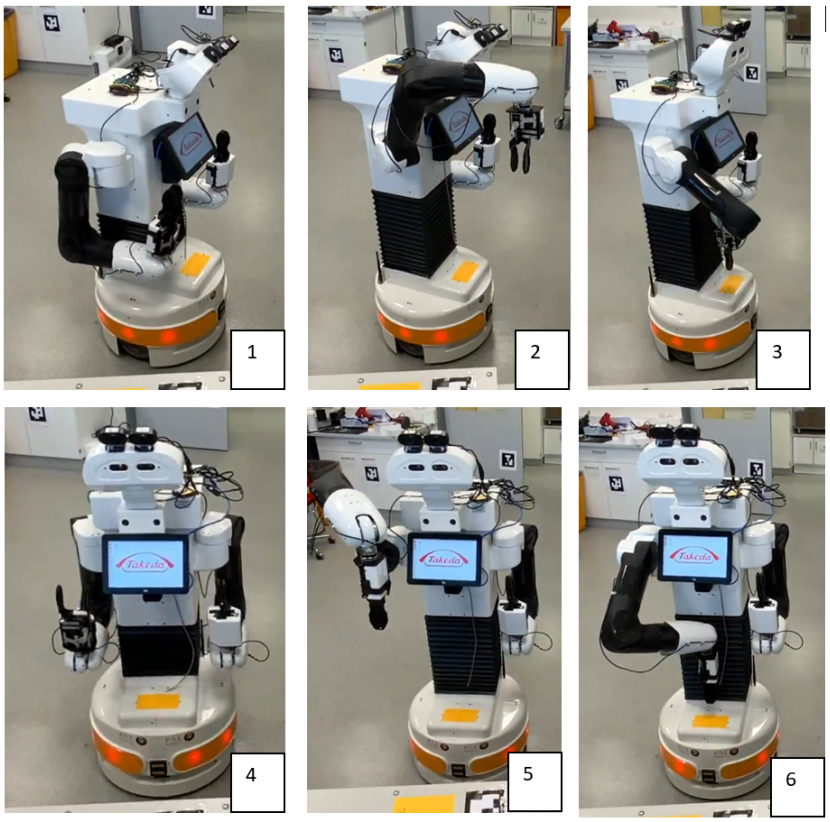

Figure 6: The three configurations between which the robot's movements are defined.. 1,4 - *Home*. 2,5 - *Benchtop*. 3,6 - *Nest*

We implemented the low-level RARs by the means of ROS functionalities:

- (Q) Motion sequences: `Dock()`, `Undock()`, `Motion(motion_goal_name)`, `ArucoMotion(pose_goal_name)`
- (P) Motion primitives: `GoToPOI(POI_goal_name)`, `Move`, `grasping`
- (A) Actuator primitives: `open_gripper_right`, `close_gripper_right`

Figure 4 presents how we implemented each RAR in the form of SiLA and ROS functionalities.

4.7. SiLA-ROS bridge

A crucial part of the implemented structure is the SiLA-ROS bridge. This allows for smooth communication between the ROS and the SiLA ecosystems, based on a client-server model. It helps to connect one or multiple ROS-based devices at once to the higher-level SiLA control layer. In the open-source software repository of the SILA Consortium exists a reference implementation for realizing the SILA-ROS integration. During the implementation we encountered problems with version compatibility of ROS, Python, OS and robot firmware, making it difficult to find a configuration where all components could communicate properly. Attempts to run the SILA-ROS bridge developed in Python 3 on the robot with ROS Melodic on Ubuntu 18.04 proved unfeasible, requiring partial reimplementation and posing challenges. However, such difficulties can be avoided in the future, as each subsystem is continuously maintained.

In Table 1 we mapped the corresponding concepts of the SiLA and ROS ecosystems against each other, based on their functions. We also show the best-practice usage of each Element.

Table 1: Corresponding concepts between SiLA and ROS. Elements on the two sides in the same row are considered functionally analogous [21, 42].

SiLA		**ROS**	
Element	**Function/ description**	**Element**	**Function/ description**
Data type	Describes the data type of any information exchanged between SiLA client and SiLA server	Message definition	Describes the data structure of any information exchanged between ROS nodes.

Observable property	Used for continuous status update from the server to the client, when the client should be notified about any change of the property value	Topic	Continuous data streams. Named buses over which nodes exchange *messages*.
Unobservable property	When the change of the property value is not very frequent, SiLA client has to poll for new values. When there is no need for regular updates of the property value.	Parameter	Parameter server: a shared, multi-variate dictionary that is accessible via network APIs. Nodes use this server to store and retrieve parameters at runtime. Best used for static parameters.
SiLA server / client	Server: a system (a software system, a laboratory instrument or device) that offers *features* to a SiLA client. Client: a system that is using *features* offered by a SiLA server.	Node	A process that performs computation. Nodes are combined together into a graph and communicate with one another using streaming *topics*, *remote procedure call (RPC) services* and the *parameter server*.
Observable command	Any command for which observing the progress or status of the command execution on the SiLA server is possible and makes sense, e.g., measuring a spectrum.	Action / topic	Longer activity with continuous feedback. Client-server communication. Clients send a request to a server in order to achieve some *goal* and will get a *result*. While being performed, the server sends progress *feedback* to the client.
Unobservable command	Any command for which observing the progress or status of the command execution on the SiLA server is not possible or does not make sense.	Service / topic	Short activity with a simple response. Request-reply-based communication is defined by a pair of messages.

Command parameter	A parameter of a command which MUST be submitted with the command execution request.	Action goal	The actions use ROS topics to send goal messages from a client to the server.
Command return value	A command response contains a result of a command execution.	Action result	When the *goal* is achieved, the server returns a result message and status.
Estimated remaining time; progress	Progress info: is the estimated progress of a command execution, in percent. Estimated remaining time: is the estimated remaining execution time of a command.	Action feedback	After receiving a *goal*, the server processes it and can give information back to the client, like the *feedback*. The *feedback* callback function processes information about this *goal*'s execution periodically whenever a new *feedback* message is received.

In our implementation four important Python programs handle the task, based on the SiLA-ROS bridge reference implementation:

- A SiLA server implementation (`sila_labware/sila_labware/server.py`)
- A SiLA feature implementation for `LabwareTransferManipulatorController`(`sila_labware/feature_implementations/labwaretransfermanipulatorcontroller_impl.py`)
- The actual ROS bridge, which is both an action client for the ROS side and a SiLA server (`sila_ros_bridge/src/_bridge_node.py`)
- A ROS action server (`ros_labware/src/getlabware_action_node.py`)

A sequence diagram of how the bridge works is shown in Figure 7. The command call originates from the SiLA client in the direction of the SiLA server. A command represents a RAR (presented in Figure 4). Then, the server calls the RAR function on the feature implementation, which registers callbacks towards the bridge. The bridge breaks down the callbacks into ROS functionalities by calling the respective ROS actions on the ROS action server. When the action ends, the result is sent back all the way to the SiLA client through the respective interfaces. From the first call and the succeeded state, continuous feedback is given among all interfaces.

We demonstrate the detailed mechanisms of the bridge process with the `LabwareTransferManipulatorController` feature's `GetLabware` command, broken down into the previously introduced interfaces. The SiLA server acts as a dictionary for storing and managing the callbacks. The skeleton for this is normally generated by the code generator, based on the feature definition. Storing is done in the variable, `_registered_callbacks`, where other callback functions can be added if needed.

The feature implementation is an example of the `LabwareTransferManipulatorController` feature's `GetLabware` command. This program acts as a server for the callbacks as they are defined here. If the client calls the `GetLabware` command it will execute here. In the case of an observable command/property, a feedback function has to be defined for the corresponding feature. Feedback comes from the ROS action server. If a feature is called, a callback function has to be passed to the ROS side's action client.

The actual SiLA-ROS bridge program is both an action client on the ROS side and a SiLA server. Callbacks from the server side e.g., `GetLabware`, have to be registered here, as `sila_server.register_callback ("getlabware", action_callback_getlabware)`. Launching the SiLA server, e.g., the server generated with the SiLA code generator is done in the bridge. The `fn_feedback` parameter, given to the function `action_callback_getlabware` is the feedback given by the SiLA side's `LabwareTransferManipulatorController` feature implementation. The action `/getlabware_action_node` is used in this program by creating an action client with `GetLabwareAction` type. By using

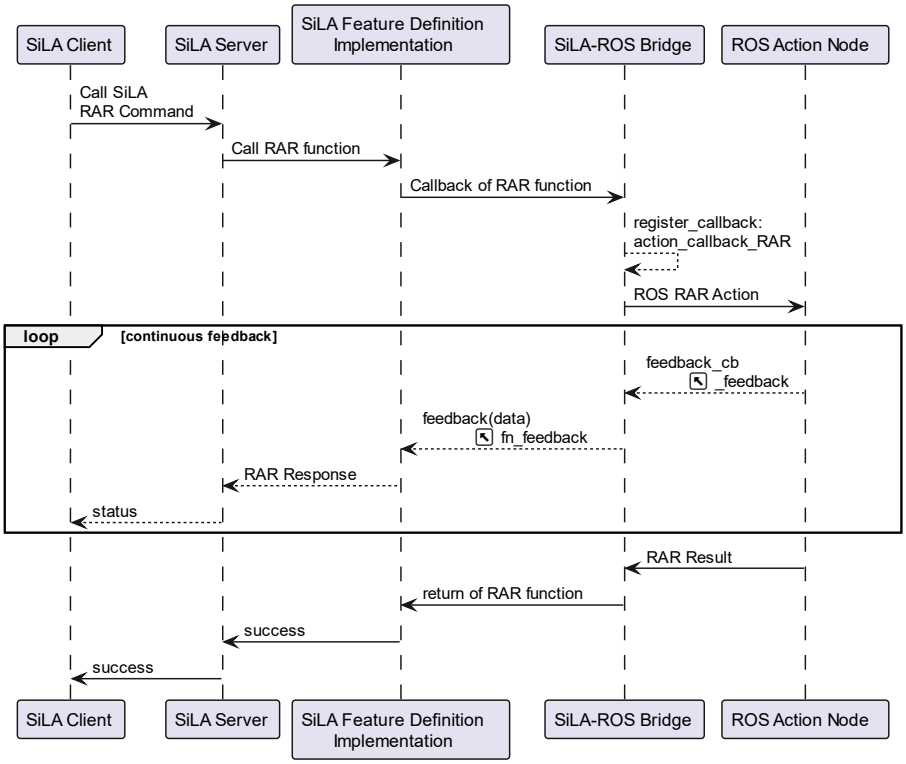

Figure 7: Sequence diagram of the SiLA-ROS bridge. Components are the programs taking part in the bridging process, callbacks are arrows moving to the right, feedback and results to the left. Feedback between the programs is implemented in a continuous feedback loop.

this action, we can send a *goal*, where the `fn_feedback` given as a parameter is the SiLA side feedback, this is handed over to the `feedback_cb` property as the callback. The `topic_callback_status` function implements an update in case a ROS topic is used for observing a variable, so the callback is called from the ROS side every time e.g., if a robot's position needs to be shown on a SiLA interface, (we register the robot's odometry) with the rate matching with the rate of the ROS topic. This corresponds with a SiLA observable property as seen in Table 1. Implementing a service call is also possible, by calling similarly to the `action_callback_getlabware()` function. Steps should be calling the service, waiting for the service to start, executing, updating the SiLA server values if needed, and waiting for the service to end in a looping fashion because there is no feedback in a ROS service. This is analogous to a SiLA unobservable command. Another way would be to convert the service to an action.

Another possibility is to use ROS topics instead of actions or services, all of which can be broadcast through the SiLA action server. Subscribing to a topic is performed by `rospy.Subscriber("status", Status, topic_callback_status, queue_size=1)`.

The lowest layer of the SiLA-ROS bridge is the ROS action server, which handles the `GetLabware` action on the ROS side. This gives feedback on the execution of the actions for the bridge. This action client publishes messages like the `HandoverPosition` as a parameter. If a service type is needed to be used - i.e., an activity without continuous feedback - a mock-up can be implemented in the `execute_callback()`. First, we have to call the service, then propagate the state of waiting for the service in a loop, and then give feedback when the service is finished. Encapsulating the service makes it possible to use it as a SiLA observable command in the bridge. This way, ROS and SiLA functionalities that do not match (as outlined in Table 1) can also be linked.

4.8. SiLA test client

Our approach for demonstrating the *scheduler functionality*, as described in [25] (this issue) is to use the Universal SiLA client [43] and the more recent SiLA Suite, developed by UniteLabs, a member of SiLA Consortium [44]. For this experimental setup, such a test client proved to be suitable. In a real-world laboratory setup, this role would be filled by a full-blown scheduler, such as Green Button Go by Biosero, NiceLab by EquiCon or LabExpert®by AstechProjects.

5. Testing

The efforts connected to the LAPP initiative span across the *SiLA Robotics Working Group* and the present work. The different milestones of this initiative can be assigned to the Technology Readiness Level (TRL) scale [45], these levels can help when describing the testing status.

Testing took place in two phases, at two different locations. The first phase was at the University's Robotics Laboratory (IROB), while the second was at a pharmaceutical company's robotics laboratory. As for the first phase, we created a TRL level 4 test scenario for the pick-and-place testing, while the second phase concluded a TRL level 5 scenario, in a relevant environment, in this case a pharmaceutical laboratory. Both phases verified the same pick-and-place scenario with as similar setups as possible in order to create consistent and repeatable environments.

5.1. Test runs

We tested different Ⓢ Subtask-level sequences multiple times, namely the navigation, the picking and placing separately and the Ⓣ Task-level sequence as a whole. These activities showed different levels of robustness during testing.

At first, we focused on testing the `ArucoMotion` node, while standing in one place in front of a station, starting from the *Benchtop* configuration, looking down at the station. We ran the following steps in a loop:

- Open gripper
- Move to the site approach position, based on marker transformations
- Move down to the site's level
- Grasping
- Move up to site approach position
- Move back down to the site's level
- Open gripper
- Move up to site approach position

This sub-sequence was running for around ten minutes, when one of the gripper's motors overheated, generating an error. This caused one of the fingers to get stuck in one position. This way, testing could not be continued, only after cooling the motor back to a usable temperature. This was caused by a bug in the grasping functionality, which uses force feedback. Apart from this problem, the accuracy of the gripper placement and arm movements proved to be adequate. The repeatability and reliability proved to be insufficient, mostly during the placing of the microplate on the site. Around 10% of the placements were not accurate in that the plate was misplaced by a few millimeters. This caused the plate to not fit correctly in the socket. The most obvious solution would be to redesign the site sockets to be more forgiving and have a greater zone of success. The inaccuracy is an accumulation of mechanical play in the kinematic chain and the error of the vision system. Addressing these limitations lies outside the scope of the present paper.

Next, we tested the navigation with the simple steps of sending the robot to the three POIs repeatedly. Out of the three subtasks, we observed the biggest inaccuracy in this step. Imperfections of the floors in both locations were causing troubles for the robot due to its six-wheel setup. It got stuck easily, caused by situations where one of the driven wheels hovered above the floor. This misalignment ultimately disoriented the odometry. In instances when the laser localization cannot correct for the mechanical deviations of the movements, the robot's perceived location and orientation can shift.

Finally, we tested the whole sequence at once. Since the duration of the overall sequence was more than six minutes, and, due to the nature of the disorientation error, it required constant supervision, the number of such complete test runs was limited. Even though there were some successful rounds also recorded on video, most runs failed at some point. See the video of the final test sequence in four times acceleration in the supplementary materials S3. As a solution, the possibility of using a different robot can be helpful to build a more robust system. Another solution is to place ArUco markers or use other localization tools e.g., on the floor where the robot can easily detect them and set fixed POIs this way.

Figure 8 presents the testing or the marker-based arm motions, while Figure 9 shows a snippet from the recording of the full sequence testing.

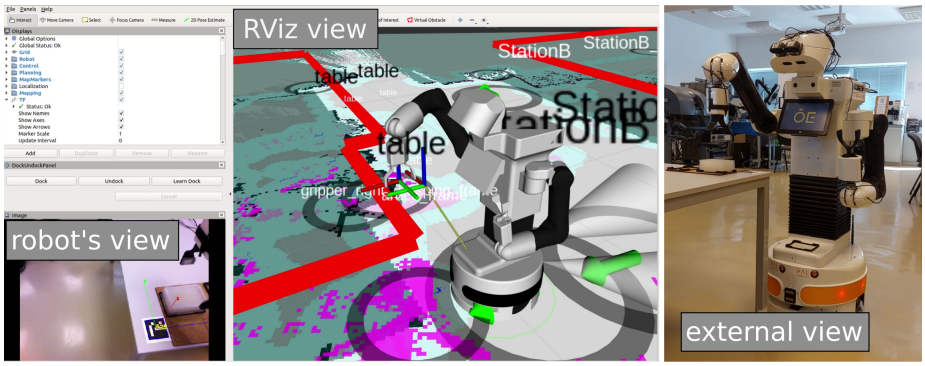

Figure 8: Testing of the ArUco-based arm motions and the microplate picking, including the robot's head camera view (left) the RViz view of the simulated robot state (middle), and an external view of the robot (right). In the RViz view, dark grey areas framed by red lines are restricted areas, light grey areas show where the robot is allowed to navigate, the green arrows are the POIs placed on the map, magenta dots are the real-time obstacle data detected by the LIDAR, grey circles mark the virtual objects on the map which can be modified by dragging and moving them

5.2. Results

In summary, our testing showed that the bottleneck lies in navigation accuracy, as malfunctions occur when the robot traverses on an uneven surface. The 6-wheel design of the AMR proved to be a sub-optimal setup for this application.

Other problems arose especially in the IROB laboratory with the network bandwidth, probably caused by the high demand image broadcasting through ROS, used for the ArUco marker detection. This constant stream of images seems to be excessive on a shared network where the only connection to any mobile robot is through WiFi or wireless connection. These problems almost always caused the whole sequence to fail at some point.

During this project, the supported software environment consisted of Ubuntu 18.04, ROS Melodic any Python 2, which was already outdated. This resulted in compatibility problems concerning the SiLA implementation, since the SiLA environment requires Python 3. According to the PAL Robotics roadmap, TIAGo robots will have a major software update with ROS 2 and Python 3 support.

Despite the difficulties with the hardware, the setup proved to be a sufficient platform for a prototype implementation to demonstrate the LAPP concept.

6. Discussion

In the scope (supportive lab robotics, labware transportation with mobile manipulators), we implement the LAPP Reference Architecture Model and the SiLA communication protocol on an open research platform, utilizing ROS.

The high-level functionality and the interfaces are, in general, robot agnostic. Proceeding toward the lower layers, the implementation gets progressively more

Figure 9: Testing of the full sequence. See full video in the supplementary material S3. View from the robot's head camera (top left), map view seen through RViz (top right), and two different recorded external views (bottom) displaying the physical robot. Markings on the map view are the same as on Figure 8.

robot-specific. The benefit of ROS, however, is that most of the functionality and the corresponding interfaces (navigation, MoveIt, etc.) are robot-independent.

Setting this concept up for a mobile manipulator for a laboratory environment, namely for tasks about labware handling, gives a stable infrastructure and a reliable reference for most tasks in similar situations. Regarding maturity, the here-discussed reference implementation can be considered as a test and demonstration environment. For production use further testing and possibly the inclusion of an industrial-grade robot are necessary.

7. Future work

This feasibility study is considered as the link between the concept proposal (LAPP) and the real-life implementation (TRL 5-6). Multiple robot vendors are motivated to enter the lab automation segment and are eagerly looking for guidance from the users, including the desired capabilities, the concept, and a blueprint. LAPP and SiLA serve this purpose, and this work takes the concepts a step closer to real-life implementation.

8. Conclusion

In this study, we have demonstrated a minimum viable prototype of the Laboratory Automation Plug & Play (LAPP) concept, focusing on its Reference Architecture Model (RAM) and Robotic Activity Representations (RAR) through the implementation of the LabwareTransfer feature on a ROS-based mobile

manipulator. Utilizing a combination of pre-existing building blocks, including the SiLA standards and ROS stacks, we established a foundational technical implementation of the proposed concepts.

The experimental implementation presented herein underscores the feasibility of integrating ROS-based mobile manipulators into realistic laboratory workflows by adhering to the SiLA standard. Furthermore, our work has illuminated the importance of software compatibility in the integration process. The challenges encountered underscore the necessity for meticulous management of software integration from the project's inception to ensure seamless functionality and interoperability within the laboratory environment.

Most importantly, the LAPP concept has proven its capability to harmonize state-of-the-art software and hardware frameworks, aligning them with the specific needs of laboratory automation within the pharmaceutical industry. This successful integration also opens avenues for further innovation and optimization.

Despite the challenges faced, including software compatibility issues and the quest for improved mechanical accuracy, the LAPP concept's implementation via the LabwareTransfer feature represents a significant step forward in the pursuit of fully automated laboratory systems. Looking ahead, further research should aim at refining these integrations, enhancing the accuracy and reliability of robotic manipulations, and expanding the scope of automation to encompass a broader range of laboratory functions. By continuing to build on the foundation laid by this study, the future of laboratory automation appears promising, with the potential to revolutionize the efficiency and effectiveness of research and development in the pharmaceutical industry and beyond.

Conflict of interest statement

Ádám Wolf is employed with "Baxalta Innovations GmbH", a wholly owned subsidiary of "Takeda Pharmaceutical Company Limited", and owns stock of Takeda and/or its subsidaries/affiliates.

Acknowledgements

The authors kindly express their gratitude towards the members of the SiLA Robotics Working Group, and of the BioSASH hackathon series, including, but not limited to: Johannes Waidner and Stefan Maak, who put together the reference implementation of the SiLA-ROS bridge; Stefan Koch, for creating the `LabwareTransferController` feature definition for benchtop robots; and finally, Mark Dörr, Lukas Bromig and Georg Hinkel, who provided consultancy on the feature definition.

This work was funded by Baxalta Innovations GmbH, a Takeda company.

This work was supported by the Doctoral School of Applied Informatics and Applied Mathematics, Óbuda University.

Péter Galambos and Károly Széll thankfully acknowledge the financial support of this work by the project no. 2019-1.3.1-KK-2019-00007 implemented with the

support provided from the National Research, Development and Innovation Fund of Hungary, financed under the 2019-1.3.1-KK funding scheme. Péter Galambos is a Bolyai Fellow of the Hungarian Academy of Sciences. Péter Galambos is supported by the UNKP-23-5 (Bolyai+) New National Excellence Program of the Ministry for Innovation and Technology from the source of the National Research, Development, and Innovation Fund.

References

[1] A. Wolf, D. Wolton, J. Trapl, J. Janda, S. Romeder-Finger, T. Gatternig, J.-B. Farcet, P. Galambos, K. Széll, Towards robotic laboratory automation plug & play: The "LAPP" framework, SLAS Technology 27 (1) (2022) 18–25. doi:10.1016/j.slast.2021.11.003.
URL https://www.sciencedirect.com/science/article/pii/S2472630321000224

[2] A. Wolf, S. Romeder-Finger, K. Széll, P. Galambos, Towards robotic laboratory automation plug & play: Teaching-free robot integration with the LAPP digital twin, number: arXiv:2205.08210 (2022). arXiv:2205.08210[cs].
URL http://arxiv.org/abs/2205.08210

[3] C. D. Hawker, Laboratory automation: Total and subtotal, Clinics in Laboratory Medicine 27 (4) (2007) 749–770. doi:10.1016/j.cll.2007.07.010.
URL https://www.sciencedirect.com/science/article/pii/S0272271207000856

[4] G. Lippi, G. D. Rin, Advantages and limitations of total laboratory automation: a personal overview, Clinical Chemistry and Laboratory Medicine (CCLM) 57 (6) (2019) 802–811, publisher: De Gruyter. doi:10.1515/cclm-2018-1323.
URL https://www.degruyter.com/document/doi/10.1515/cclm-2018-1323/html?lang=en

[5] J3016_202104: Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles - SAE international, [accessed : 2023-08-28] (2021).
URL https://www.sae.org/standards/content/j3016_202104/

[6] T. D. Nagy, T. Haidegger, Performance and capability assessment in surgical subtask automation, Sensors 22 (7) (2022) 2501, number: 7, Publisher: Multidisciplinary Digital Publishing Institute. doi:10.3390/s22072501.
URL https://www.mdpi.com/1424-8220/22/7/2501

[7] J. Beal, M. Rogers, Levels of autonomy in synthetic biology engineering, Molecular Systems Biology 16 (12) (2020) e10019. doi:10.15252/msb.202010019.
URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7744957/

[8] J. Boyd, Robotic laboratory automation, Science 295 (5554) (2002) 517–518, publisher: American Association for the Advancement of Science. doi:10.1126/science.295.5554.517.
URL https://www.science.org/doi/full/10.1126/science.295.5554.517

[9] Á. Wolf, K. Széll, A review on robotics in life science automation, in: 14th International Symposium on Applied Informatics and Related Areas organized in the frame of Hungarian Science Festival 2019 by Óbuda University, 2019, pp. 106–111.

[10] D. M. F. Olaniyan, LABORATORY INSTRUMENTATION AND TECHNIQUES, CreateSpace Independent Publishing Platform, 2017.
URL https://www.amazon.com/Laboratory-Instrumentation-Techniques/dp/1547012226

[11] N. Yoshikawa, M. Skreta, K. Darvish, S. Arellano-Rubach, Z. Ji, L. Bjørn Kristensen, A. Z. Li, Y. Zhao, H. Xu, A. Kuramshin, A. Aspuru-Guzik, F. Shkurti, A. Garg, Large language models for chemistry robotics, Autonomous Robots[accessed : 2023-11-20] (2023). doi:10.1007/s10514-023-10136-2.
URL https://doi.org/10.1007/s10514-023-10136-2

[12] M. Seifrid, R. Pollice, A. Aguilar-Granda, Z. Morgan Chan, K. Hotta, C. T. Ser, J. Vestfrid, T. C. Wu, A. Aspuru-Guzik, Autonomous chemical experiments: Challenges and perspectives on establishing a self-driving lab, Accounts of Chemical Research 55 (17) (2022) 2454–2466, publisher: American Chemical Society. doi:10.1021/acs.accounts.2c00220.
URL https://doi.org/10.1021/acs.accounts.2c00220

[13] Precise automation - PreciseFlex 400 sample handler, [accessed : 2023-08-29].
URL https://preciseautomation.com/SampleHandler.html

[14] J. Javier Moreno, E. Clotet, R. Lupiañez, M. Tresanchez, D. Martinez, T. Pallejà, J. Casanovas, J. Palacín, Design, implementation and validation of the three-wheel holonomic motion system of the assistant personal robot (APR), Sensors 16 (2016) 1658. doi:10.3390/s16101658.

[15] Kevin: mobile robot for laboratory automation, [accessed : 2023-08-29] (2022).
URL https://www.kevinrobot.com/

[16] Mobile robots | biosero, [accessed : 2023-08-29] (2022).
URL https://biosero.com/integrations/mobile-robots/

[17] Robotic lab assistant, [accessed : 2023-08-29].
URL https://astechprojects.co.uk/laboratory/robotic-lab-assistant/

[18] uMobileLab, an automation solution for laboratories, [accessed : 2023-08-30].
URL https://unitedrobotics.group/en/robots/umobilelab

[19] K. Thurow, System concepts for robots in life science applications, Applied Sciences 12 (7) (2022) 3257, number: 7 Publisher: Multidisciplinary Digital Publishing Institute. doi:10.3390/app12073257.
URL https://www.mdpi.com/2076-3417/12/7/3257

[20] SiLA rapid integration, [accessed : 2022-05-18] (2018).
URL https://sila-standard.com/

[21] SiLA documentation, [accessed : 2024-02-13] (2021).
URL https://sila2.gitlab.io/sila_base/

[22] H. Bär, R. Hochstrasser, B. Papenfuß, SiLA: Basic standards for rapid integration in laboratory automation, Journal of Laboratory Automation 17 (2) (2012) 86–95, publisher: SAGE Publications Inc. doi:10.1177/2211068211424550.
URL https://doi.org/10.1177/2211068211424550

[23] S. Kleine-Wechelmann, K. Bastiaanse, M. Freundel, C. Becker-Asano, Designing the mobile robot Kevin for a life science laboratory, in: RO-MAN 2022 - 31st IEEE International Conference on Robot and Human Interactive Communication: Social, Asocial, and Antisocial Robots, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 870–875. doi:10.1109/RO-MAN53752.2022.9900786.

[24] Lab automation solutions | 25+ years | astech projects, [accessed : 2023-11-14].
URL https://astechprojects.co.uk/expertise/laboratory-automation

[25] Á. Wolf, P. Zsoldos, K. Széll, P. Galambos, Towards Robotic Laboratory Automation Plug & Play: Reference Architecture Model for Robot Integration, preprint, submitted to this issue (2024).

[26] A. Wolf, D. Wolton, J. Trapl, J. Janda, S. Romeder-Finger, T. Gatternig, J.-B. Farcet, P. Galambos, K. Széll, Towards robotic laboratory automation plug & play: The "LAPP" framework, SLAS Technology 27 (1) (2022) 18–25. doi:10.1016/j.slast.2021.11.003.
URL https://www.sciencedirect.com/science/article/pii/S2472630321000224

[27] A. Wolf, Teaching-free robot integration with LAPP digital twin, future Labs Live (2023).
URL https://www.terrapinn.com/conference/future-labs-live/index.stm

[28] feature_definitions/hu/uniobuda/irob · unified_robot_features · Ádám Wolf / sila_base_robotics · GitLab, [accessed : 2023-11-03] (2022).
URL https://gitlab.com/adam.wolf1/sila-base-robotics/-/tree/unified_robot_features/feature_definitions/hu/uniobuda/irob

[29] feature_definitions/org/silastandard/instruments/labware/manipulation · master · SiLA2 / sila_base · GitLab, [accessed : 2023-09-04] (2023).
URL https://gitlab.com/SiLA2/sila_base/-/tree/master/feature_definitions/org/silastandard/instruments/labware/manipulation

[30] Files · master · SiLA2 / sila_base · GitLab, [accessed : 2023-09-06] (2023).
URL https://gitlab.com/SiLA2/sila_base/-/tree/master?ref_type=heads

[31] Biolago.
URL https://www.biolago.org/en/

[32] SiLA2 / sila_robotics / sila_ros · GitLab, [accessed : 2023-05-13] (2022).
URL https://gitlab.com/SiLA2/sila_robotics/sila_ros

[33] A. Wolf, BioSASH-4 intro, [accessed : 2023-09-18] (2022).
URL https://docs.google.com/presentation/d/18KRi_EXG3U2Xs1hK5KAfGyVQQf6D9hRW

[34] S. Koch, SiLA2 labware transfer controller features, [accessed : 2023-09-19] (2023).
URL https://docs.google.com/presentation/d/1yxWksPMuKrcYjP_ZYytU_k0v7LIVXyQh

[35] SiLA2 / sila_robotics / sila_pick_and_place_example · GitLab, [accessed : 2023-09-19] (2022).
URL https://gitlab.com/SiLA2/sila_robotics/sila_pick_and_place_example

[36] A. Wolf, BioSASH-4 results, [accessed : 2023-09-19] (2022).
URL https://docs.google.com/presentation/d/1GKa13sUpJGoAOyef5O5UbaAh5CQGgFOd

[37] TIAGo - the mobile manipulator robot platform for your research, [accessed : 2022-05-13].
URL https://pal-robotics.com/robots/tiago/

[38] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, et al., Ros: an open-source robot operating system, in: ICRA workshop on open source software, no. 3.2 in 3, Kobe, Japan, 2009, p. 5.

[39] sila_ros_bridge · master · SiLA2 / sila_robotics / sila_ros · GitLab, [accessed : 2023-09-18] (2022).
URL https://gitlab.com/SiLA2/sila_robotics/sila_ros/-/tree/master/sila_ros_bridge

[40] Robots/TIAGo/tutorials/motions/play_motion - ROS wiki, [accessed : 2023-10-26] (2023).
URL http://wiki.ros.org/Robots/TIAGo/Tutorials/motions/play_motion

[41] MoveIt motion planning framework, [accessed : 2023-10-26].
URL https://moveit.ros.org/

[42] ROS/concepts - ROS wiki, [accessed : 2023-11-03] (2022).
URL http://wiki.ros.org/ROS/Concepts

[43] SiLA2 / universal_sila_client / sila_universal_client · GitLab, [accessed : 2023-05-08] (2023).
URL https://gitlab.com/SiLA2/universal-sila-client/sila_universal_client

[44] UniteLabs / integrations / sila2 / sila-browser · GitLab, [accessed : 2023-10-25] (2023).
URL https://gitlab.com/unitelabs/integrations/sila2/sila-browser

[45] S. R. Sadin, F. P. Povinelli, R. Rosen, The NASA technology push towards future space mission systems, in: Acta Astronautica, Vol. 20, 1988, pp. 73–77, NTRS Author Affiliations: NASA Headquarters, NASA Office of Aeronautics and Space Technology NTRS Report/Patent Number: IAF PAPER 88-033 NTRS Document ID: 19890030268 NTRS Research Center: Legacy CDMS (CDMS).
URL https://ntrs.nasa.gov/citations/19890030268

Supplementary Data for: Towards Robotic Laboratory Automation Plug & Play: Reference Implementation with the TIAGo Mobile Manipulator

Panna Zsoldos[a], Ádám Wolf [b,c], Károly Széll [d], Péter Galambos [a]

[a] *Antal Bejczy Center for Intelligent Robotics, Óbuda University, Budapest, Hungary*
[b] *Baxalta Innovations GmbH, a Takeda company, Wien, Austria*
[c] *Doctoral School of Applied Informatics and Applied Mathematics, Óbuda University, Budapest, Hungary*
[d] *Alba Regia Technical Faculty, Óbuda University, Székesfehérvár, Hungary*

Contents

S1. Sequence diagrams

*Corresponding author
Email addresses: panna.zsoldos@irob.uni-obuda.hu (Panna Zsoldos), adam.wolf@takeda.com (Ádám Wolf)

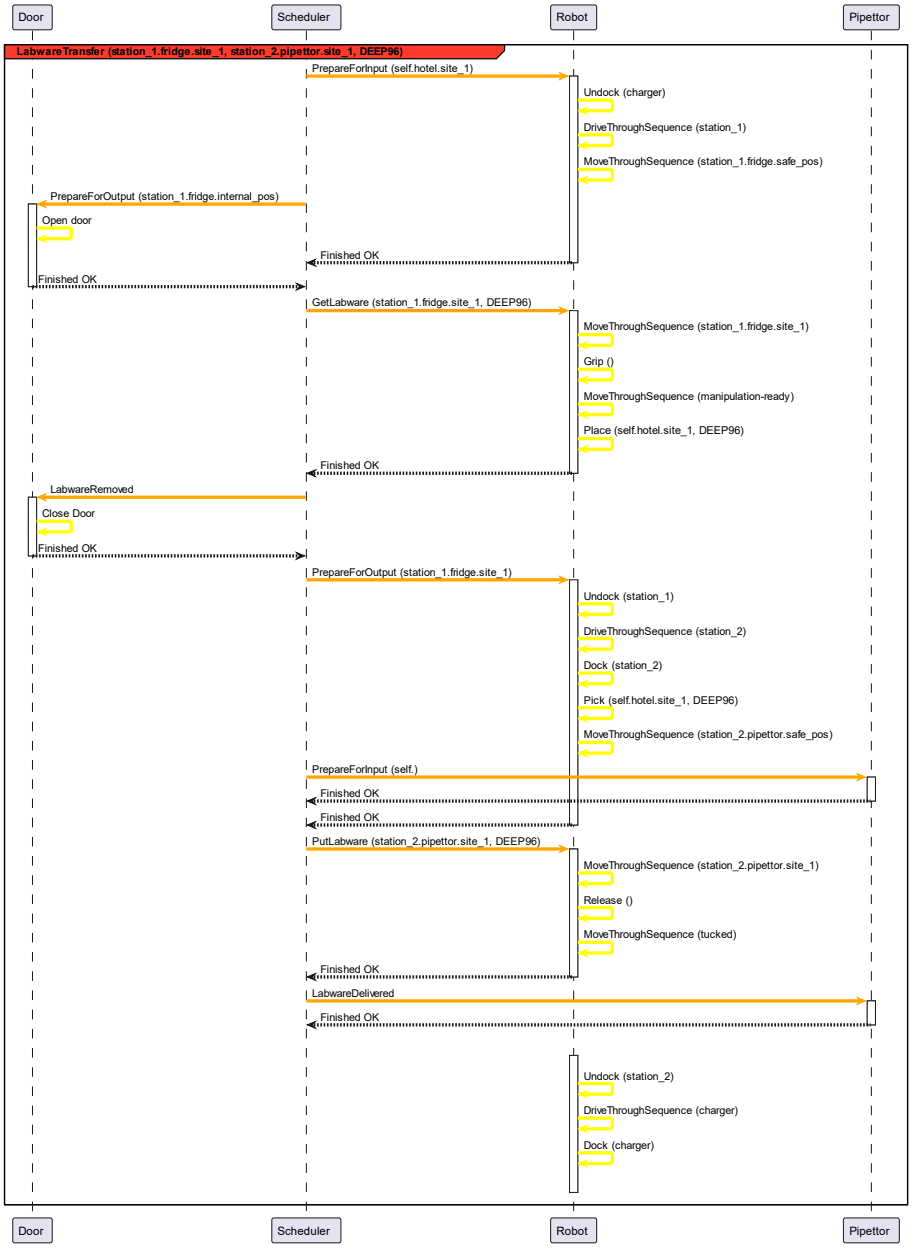

Figure S1: Sequence diagram of a labware transfer task with a robot, between a fridge and a pipettor equipment - Part 1

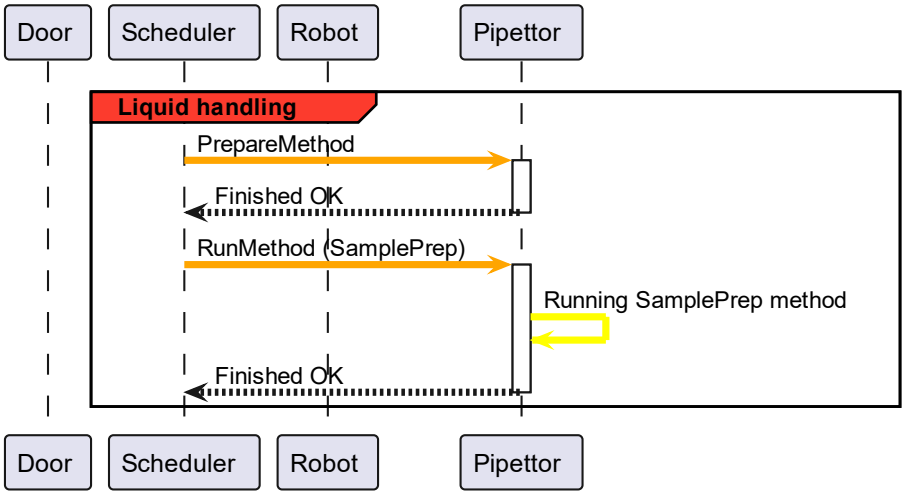

Figure S2: Sequence diagram of a labware transfer task with a robot, between a fridge and a pipettor equipment - Part 2

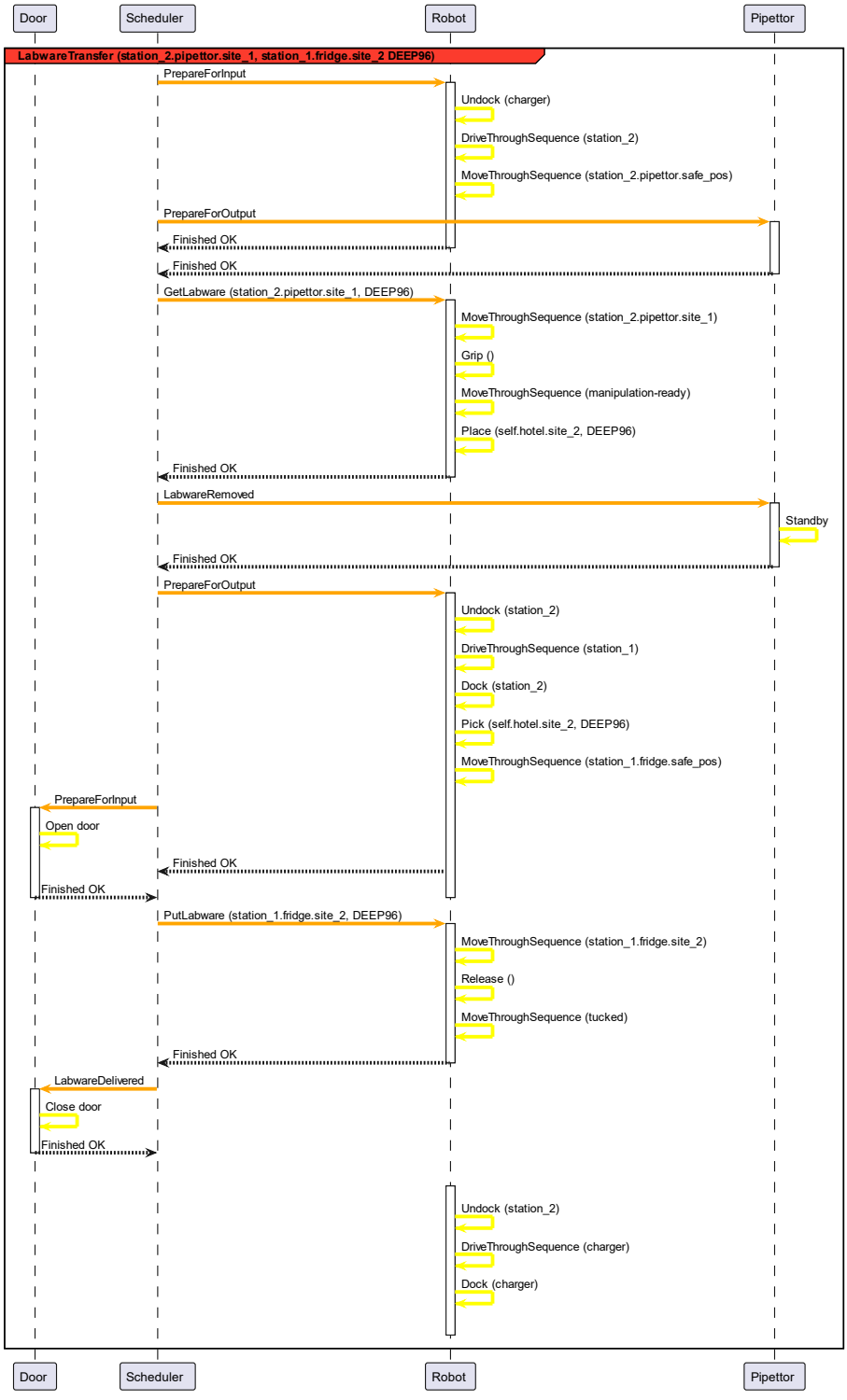

Figure S3: Sequence diagram of a labware transfer task with a robot, between a fridge and a pipettor equipment - Part 3

S2. RAR mapping

Mapping our implementation to the Robotic Activity Representations (RARs) of the Laboratory Automation Plug & Play (LAPP) concept.
`TIAGo_RAR_sequence.xlsx`

S3. Test sequence video

Testing of the full sequence. View from the robot's head camera (top left), map view seen through RViz (top right), and two different recorded external views (bottom) displaying the physical robot. Markings on the map view are the same as on Figure 8.
`test_sequence_4x.mp4`