AIT Lablink Documentation

AIT Lablink Development Team

Aug 02, 2022

GETTING STARTED

The **AIT Lablink** aims at supporting an efficient, modular and flexible development and evaluation environment, which focuses on laboratory hardware and software equipment. Its main goal is to support and automate common tasks such as coupling components, logging results and automating test cases.

A common middleware that handles communication among distributed clients comprises the core of the AIT Lablink. This middleware is called **Lablink Core** and provides data routing and and encoding facilities. Lablink Core is designed as a distributed application, implemented in a dedicated library, whose interfaces expose the functionality of Lablink.

The Lablink Core infrastructure provides the basis to implement dedicated **Lablink clients**, enabling access to common components such as laboratory hardware or domain-specific simulation tools. In addition, utility tools for synchronizing clients or recording data can be implemented on top of the Lablink Core.

CHAPTER

ONE

GETTING STARTED

Lablink in a Nutshell gives a brief overview of the main concepts of the AIT Lablink.

To get started and run a few simple Lablink setups, you can go through the Lablink examples.

1.1 Lablink in a Nutshell

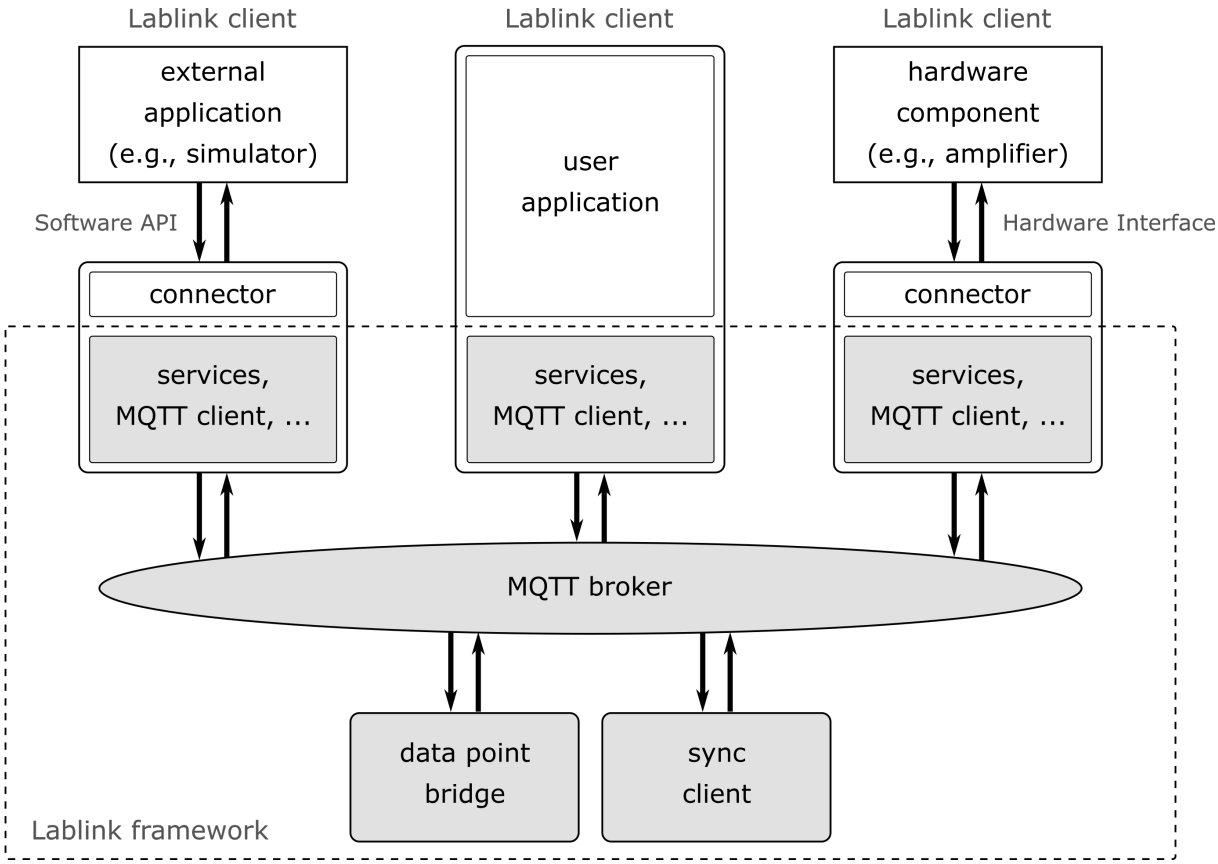
The Lablink infrastructure is designed to loosely couple various components in the laboratory environment. Basically it is a communication protocol especially designed for the requirements in a Co- or Hardware-in-the-loop simulation system in a modern laboratory.

1.1.1 Lablink Core

The core component of Lablink provides a communication system based on MQTT as a transport protocol, with asynchronous messaging and syncronous remote procedure calls. On top of this, the Lablink core implements services for clients (see below).

The Lablink core is (currently) implemented as a Java library. The source code is available online in the Lablink Core code repository, the corresponding packages are available as JAR files via the dedicated remote Maven repository.

1.1.2 Lablink System Architecture



1.1.3 Lablink Clients

A Lablink client is a self executing process that will have a **single** connection to the Lablink system. A client can have different purposes. It can be a connector to an external simulator that allows the data exchange between other Lablink clients and the simulator (e.g., Digsilent PowerFactory). A Lablink client can also be connector to a hardware device within the laboratory using defined communication standards (e.g., Modbus TCP or a serial interface). The third possibility is a Lablink client that implements itself the logic and the Lablink connection in one software process.

1.1.4 MQTT Broker

Lablink is based on MQTT as low level communication. Therefore, for using Lablink an MQTT broker is required that will distribute the messages between different Lablink clients.

The usage of MQTT provides some common functionalities that can be used with the Lablink. An example of such a functionality is the usage of secured connections. MQTT provides a configurable SSL secured connection between a client and the broker. Another example can be the load distribution of the MQTT broker. In the open-source world there are MQTT brokers available that implement this feature.

1.1.5 Lablink Connection

The Lablink connection is part of the core library. The connection is an application interface that provides the access to the Lablink system. Every Lablink client uses a single Lablink connection. The Lablink connection provides an interface and allows the sending of messages and remote procedure calls through MQTT to other Lablink clients.

1.1.6 Address scheme

Each Lablink connection must have an unique identifier that will be used within the Lablink system. With this unique identifier the Lablink client can be addressed by other clients. The identifier consists of three parts represented as strings:

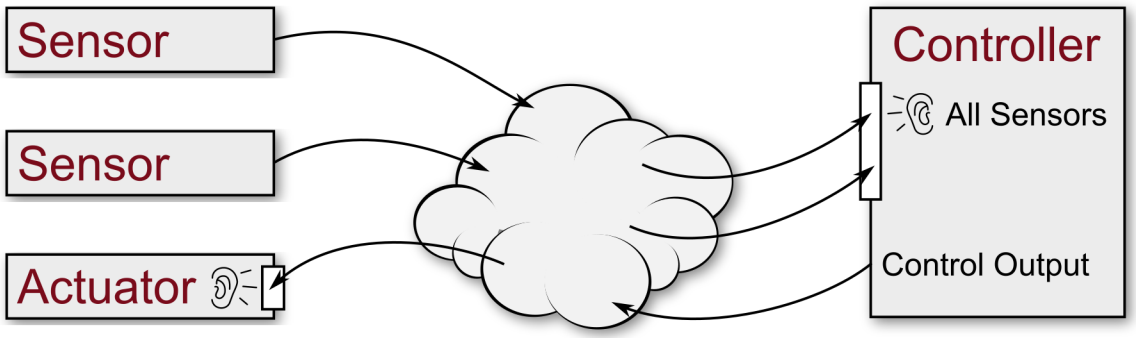
- **Application Identifier**: The application identifier will be unique within a Lablink application. Only clients with the same application identifier can talk to each other. This identifier is necessary to allow different applications to run in parallel on a single broker. A common problem during the implementation of such an application is that to clients have different application identifiers and they won't see each other.
- **Client Group**: Every client is member of a group. A group can be addressed at once within the Lablink system. So it is possible to receive a message from all clients within a defined group.
- **Client Identifier**: Every client has its on identifier. The combination of the group and client identifiers makes the addressing of the client unique. A single client can be addressed by using the group and client identifier.

1.1.7 Communication paradigms

The messaging components specially focus on the aspect of data exchange. Transferred data is encapsulated into transmission units of finite length which are called messages. Two different transmission schemes are supported. The uni-directional transmission scheme is called **Message-scheme** and the bi-directional scheme is called **RPC-scheme**.

Message-Scheme

The Message-scheme is used to publish data without specifying a particular destination. The data source labels the published data with a statically defined identifier. A subscribing data sink allows the user to configure the received data by specifying a set of identifiers. Hence, the data source does not need to specify the destination and may be implemented efficiently. Following the Message-scheme, only a uni-directional communication link is established.



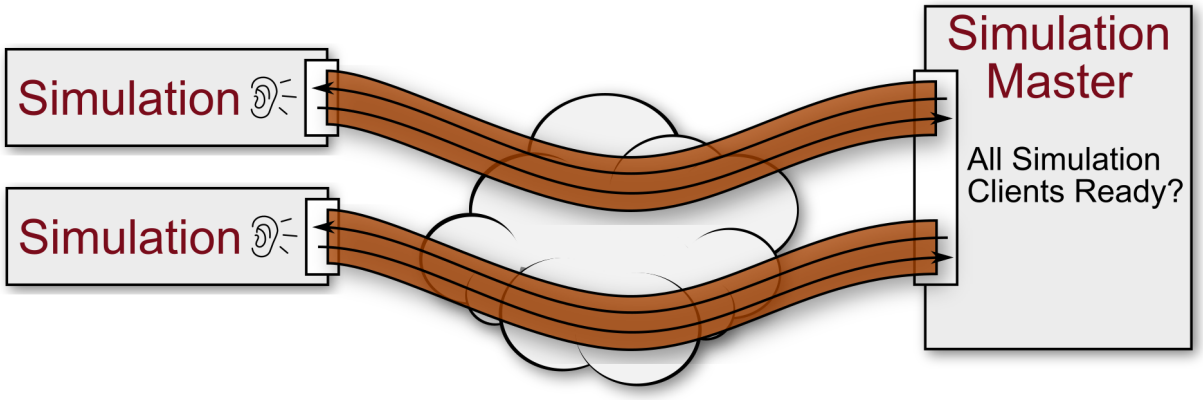
For instance, a sensor may act as a data source and publishes data as soon as new readings are available. The name of the published data point is statically defined. A controller is configured as a data sink which subscribes to a particular set of sensors. As soon as new data readings are available, the controller processes its input and publishes the result. An actuator may be configured as a data sink and may listen to any outputs from a particular controller. In the above

example only uni-directional communication is required. Hence, loose coupling can be handled efficiently by the Message-scheme.

RPC-Scheme

The RPC-Scheme is used to establish a bidirectional communication on top of a publish/subscribe-based transmission. It allows a single client to address a set of servers, which will be able to respond to that client. The first message from the clients to the servers is called request, the set of return messages replies. The RPC-scheme allows to correlate received reply messages with their requests.

In contrast to the Message-scheme, the destination of a request will be configured at the sender. The RPC server statically registers an end point and the client reads a user defined configuration to determine the destination addresses. The request sink may be used to provide functionality (e.g., a remote procedure) without defining the clients which use it. In order to receive a reply, the requester may dynamically register the return channel before sending the request. After all requests are received, the channel is unregistered. Alternatively, it may statically register the return channel to avoid frequent registration overhead.



A simulation master may utilize the transition scheme by acting as a RPC requester. Each simulation client registers a RPC request end point. The simulation master may request the current status of each client by sending an RPC request. The clients reply a status message. The simulation master continues as soon as all response messages have arrived. Since each response is correlated to the initial request, a bi-directional channel is established.

Subject

Every message and RPC call has a subject. This subject will be used to identify the meaning of the message or call.

Message Subscription

A client can register a subscription to a subject. If the messaging is used than the subscription can use wildcards for filtering only specific parts of a message subject.

RPC Destination

The RPC Destination defines the receivers of a remote procedure call. There are three possibilities to define the receivers.

An RPC call can be send to: * all clients within the application (same application identifier) * to all clients of a specific group (same group identifier) * to a specific client (group and client identifier)

1.1.8 Services

A service within the Lablink system is a container for higher-level functionalities. A service combines different messages and remote procedure calls between different clients to provide a higher-level function. This service hides the complexity of the Lablink communication and provides a simple interface to the client for a specific task.

Currently the Lablink core implements two services:

- **DataPoint Service**: The datapoint service allows the exchange of simple values between Lablink clients. The datapoint server provides the exchange of these values using simple get and set methods. It also provides state information to the client, like the detection of an established connection between the clients or the notification of changed or received new values from a remote client. The datapoint bridge is a standalone Lablink client that implements and uses this service.
- **Sync Service**: The sync service provides the time synchronization between the Lablink clients. This service will be used if the Lablink works as a Co- or HIL-Simulation framework. The synchronization host is a standalone Lablink client that uses and implements this service.

1.1.9 Lablink Application Interface Concepts

The Lablink core is implemented in a multi-threaded way. Every incoming message will use its own thread for execution. This allows the parallel execution of the receiving task.

A client can register a callback method to react on an incoming message. This callback method will be called from the Lablink core in different threads. Therefore a synchronization of the callback methods can be necessary.

All methods of the Lablink core that a client calls will be executed synchronously. This means that the method can be blocked by the core.

1.2 Lablink examples

To get started with using Lablink, you may start with the following examples:

- OPC UA client example: A simple Lablink setup for connecting an OPC UA server with a data source and a plotter, a good starting point for undestanding how to use the available Lablink resources for your own purposes.
- Lablink examples: Introducing the basic concepts of Lablink, a good starting point if you want to create your own Labling clients.
- Universal API client example: This example demonstrates the use of the Lablink client that implements the [ERIGrid Universal Data Exchange API](https://erigrid2.github.io/JRA-3.1-api/universal-api.html).

CHAPTER

TWO

AVAILABILITY

The AIT Lablink is an open-source software, publicly available on Github and the Maven Central Repository. For more information, please take a look *here*.

2.1 Software Repository

The AIT Lablink is an open-source software, the source code is publicly available on Github.

2.2 Maven Central Repository

The compiled AIT Lablink packages are available on the Maven Central Repository.

2.3 Lablink License

Lablink's core and the majority of all Lablink clients are licensed under a 3-clause BSD license.

Note: This license allows unlimited redistribution for any purpose (including commercial use) as long as the copyright notices and the license's disclaimers of warranty are maintained. This license is also compatible with the GNU GPL.

```
Copyright (c) 2020 AIT Austrian Institute of Technology GmbH
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this
   list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice,
   this list of conditions and the following disclaimer in the documentation
   and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
   contributors may be used to endorse or promote products derived from
   this software without specific prior written permission.
```

(continues on next page)

(continued from previous page)

```
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

CHAPTER

THREE

THE LABLINK FRAMEWORK

The AIT Lablink is a modular framework that comes with a collection of readily available clients. For more information, please take a look *here*.

Lablink Development Repository

For a complete overview of available Lablink resources you can also take a look a the public Lablink repository on Github!

3.1 Lablink Core

The Lablink Core provides a common middleware that handles communication among distributed clients

3.2 Datapoint Bridge

Datapoint Bridge provides the connection between the datapoints (also called services, inputs, outputs etc.) of Lablink clients.

3.3 Configuration server

The Configuration server is a simple server for Lablink setups

3.4 Synchronization host

The Synchronization host is a standalone Lablink client that provides the functionality to synchronize clients during a simulation run.

3.5 Redis client

The Redis client provides an adapter for interfacing with a Redis database.

3.6 OPC UA client

The OPC UA client provides an adapter to communicate via OPC UA with external resources.

3.7 FMU Simulator

The FMU Simulator provides clients that can import and execute Functional Mock-up Units (FMUs).

3.8 Universal Data Exchange API

The Universal API client implements the ERIGrid Universal Data Exchange API. This allows to interact with other Lablink clients through a simple REST interface.

3.9 CSV client

The CSV client provides a data source using CSV data.

3.10 Plotter

The Plotter is a Lablink client for visualizing data as plotted graphs in a separate window.