SoftwareX 17 (2022) 100991

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

The SiLA 2 Manager for rapid device integration and workflow automation

Lukas Bromig, David Leiter, Alexandru-Virgil Mardale, Nikolas von den Eichen, Emmeran Bieringer, Dirk Weuster-Botz *

Institute of Biochemical Engineering, Technical University of Munich, Boltzmannstr. 15, 85748 Garching, Germany

ARTICLE INFO

Article history:
Received 9 March 2021
Received in revised form 23 August 2021
Accepted 11 January 2022

Keywords:
Laboratory automation
Device integration
SiLA2
Data integrity
Biotechnology
Life sciences
Software framework
Web application
IoT

ABSTRACT

Rapid integration of laboratory devices into automated workflows remains an arduous and time-consuming process. A lack of interface standardization among hardware vendors leads to inflexible device setups and highly customized and expensive software solutions. Thus, practical integration capabilities of existing laboratory control systems (LCS) are insufficient when it comes to developing small, rapid and cost-efficient automation solutions. The increasing importance of data integrity as well as software system and workflow documentation, due to cGMP regulations, is calling for structured, reliable and transparent middleware solutions. The SiLA 2 Manager uses the emerging SiLA 2 standard and provides a lean and extendable framework for device discovery, management, and workflow design, thereby bridging the gap between the physical device and higher software levels. This Internet of Things (IoT) application enables immediate control of SiLA 2 devices and ensures user-friendly real-time data collection and storage.

Code metadata

Current code version	v1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-21-00051
Code Ocean compute capsule	*n/a*
Legal Code License	MIT
Code versioning system used	Git
Software code languages, tools, and services used	Python, Typescript, Shell, Bash, HTML, Angular, FastAPI, Docker, SiLA 2, PostgreSQL, Redis, InfluxDB, Nginx
Compilation requirements, operating environments & dependencies	Linux, Unix-like, Microsoft Windows, Python $\geq$ 3.7, node.js $\geq$ 12
If available Link to developer documentation/manual	https://sila2-manager.readthedocs.io/en/latest/
Support email for questions	lukas.bromig@tum.de

Software metadata

Current software version	v.1.0
Permanent link to executables of this version	https://gitlab.com/lukas.bromig/sila2_manager
Legal Software License	MIT
Computing platforms/Operating Systems	Linux, Unix-like, Microsoft Windows
Installation requirements & dependencies	See installation documentation and requirements.txt
If available, link to user manual	https://sila2-manager.readthedocs.io/en/latest/
Support email for questions	lukas.bromig@tum.de

* Corresponding author.
E-mail address: dirk.weuster-botz@tum.de (D. Weuster-Botz).

https://doi.org/10.1016/j.softx.2022.100991

1. Motivation and significance

Laboratory automation in the commercial and academic environment in the life and chemical sciences has been gaining momentum over the past two decades [1,2]. The trend of process parallelization and miniaturization in the chemical and biochemical laboratory, coupled with Design of Experiment (DoE) based software assisted experiment planning is calling for increasingly automated experimental setups and advanced capability for digital device interaction [3–6]. The main driving force is not only increased efficiency and throughput, but also automated documentation and a reduction in human error. Regulatory bodies have expanded their guidelines to keep up with technological advances in laboratory automation, encompassing not only audit trail documentation, but documentation and validation of software and computerized systems as well (cGMP, EudraLex). Increasingly detailed documentation requirements can only be fulfilled efficiently by digitized systems [7–9].

Background and problem statement

Computerized systems in a laboratory environment are hierarchical constructs of several subsystems. They range from the enterprise or laboratory-level information management systems (LIMS) to the laboratory or process control software down to the so-called middleware on the device level. In most cases, the process control software directly integrates the necessary devices without the use of a designated middleware. Prominent examples are liquid handling stations that operate with several periphery devices and require dedicated software for their operation. Access to the respective periphery devices is only possible through the vendor software interface. The integration of new devices is difficult and, due to the closed-source nature of the software, requires application specialists. Furthermore, such software solutions generally lack an appropriate application programming interface (API) for further integration with other process control software or into higher-level organizational systems such as LIMS.

A variety of vendor-specific, proprietary software and hardware interfaces exist for standalone laboratory equipment. Custom solutions that combine several devices to create automated workflows have long development times and require programming and software engineering expertise. This makes the integration of individual devices into an automated workflow an arduous task. The combination of various standalone devices with each other, like a digital scale with a robotic arm or the integration of an unsupported third-party device into a liquid handling station (LHS), enables a more efficient mode of operation. However, the lack of standardized device interfaces and of appropriate middleware still impedes the development of automated workflows [10,11].

Hence, larger automation systems grow in software and hardware complexity and require subject matter experts. Replacement or upgrading of individual devices is not straightforward and the most often used closed-source, non-modular software solutions lead to low code reusability and a high long-term maintenance cost [12,13]. Due to a lack in hardware interface standardization, most commercial automation solutions focus on workflow and audit trail documentation or data and user management, but not basic middleware for hardware integration.

Despite recent advances in high throughput system technology and a constantly broadening automation product range, laboratory automation is still most often confined to custom-built island solutions [2,7]. These solutions, whether implemented as closed or open systems, are highly inflexible and come at a high development cost. The resulting cost to automate a task in contrast to the cost of the respective manual labor, i.e. a tradeoff between the amount of repetitions versus the complexity of the task, still defines the grade of automation in the laboratory environment to this day. This tradeoff is an old paradigm which may hold true for high-throughput stations but falls short when considering the laboratory environment as a whole. There are many aspects to consider in laboratory automation, such as gained flexibility, replaceability, integration into existing systems, time savings and thus competitive advantage, minimization of human error, reduction of sample volume and data integrity [12,14,15]. Reducing development cost through the use of innovative software for device integration directly will make lab automation more affordable and result in the aforementioned benefits.

Laboratory digitization can be overwhelming and the amount of combinations of architectural solutions for distributed computing (REST, SOAP, JSON-/XML-RPC, gRPC), programming languages and database types is vast. Developing custom solutions that integrate all kinds of devices is possible and has been shown by Porr et al. (2020) [11]. However, such solutions are quickly increasing in complexity and maintenance effort making rapid integration difficult. Laboratory automation should not be the problem, but the solution. Reducing the aforementioned complexity, in accordance with the law of parsimony, should be the main priority of any automation solution. Converging on a minimal number of network protocols, programming languages, databases and overall software dependencies is a key aim of this software proposal.

Standardization in laboratory automation - SiLA 2

Device integration requires intricate knowledge about the used communication protocol. From a user perspective, this turns simple commands like “Start_pump” into a programming challenge of e.g. string parsing, checksum calculations, numeral system conversions and worries about thread safety or buffer queue size. Furthermore, the intuitive assumption that the integration solution of one device is transferable to another device with the same functionality, but from different vendors, fails immediately if the first glance at the documentation reveals that two entirely different communication protocols are used, effectively doubling the development effort. In an ideal world, the end-user should not have to worry about such integration complexities and a workflow script written for a specific task should only care about the device type and be independent of the device make. However, temporal and structural behavior of existing interfaces is vastly different and error messages and error handling capabilities are generally incompatible with third-party products. Introducing a standardized device interface as an additional level of abstraction resolves these problems and reduces the users effort to a simple “Start_pump”-command, resulting in rapid integration capability and device interchangeability [16].

Application of a standardized interface, as proposed by the not-for-profit membership organization SiLA (SiLA, Rapperswil-Jona, Switzerland), can drastically reduce the complexity and development times of automation solutions. SiLA 2 provides the tools for rapid integration and enables device-agnostic comprehensive middleware platforms for a new generation of laboratory devices with plug-and-play capability. According to the standard, each device is implemented as a SiLA server that exposes its service in the network as a set of features. Each feature comprises a set of functionally related commands and properties available for the client to call. A SiLA device is a special case of a SiLA server, as a SiLA server can implement any kind of service. Network communication is based on gRPC via HTTP/2. The client can be incorporated in a higher-level software, such as in a LIMS or a PCS, or as in the case of the proposed middleware software, a simple workflow script [14,17–19].

The number of embedded SiLA 2 compatible devices is increasing steadily, but adoption is still in its infancy as hardware vendors have not yet fully committed resources to in-house implementations. However, since SiLA is a software solution to a hardware interface problem, any legacy device can be converted into a SiLA 2 device in a fast and cost-effective manner as shown by Porr et al. (2020) [20]. This makes every piece of laboratory hardware a potential addition to an automation workflow. Moving forward, a SiLA server that implements a laboratory device will be referred to as a service. There are current efforts to specify a standard based on OPC-UA, which is widely used in the closely related process industry. The OPC-UA Laboratory Agnostic Device Standard (LADS) may compete with SiLA 2 in the future [21].

Expectations and comparison with existing tools

A modern process control software framework should be modular and based on industry standards. The used technology must be widely adopted to ensure long-term support and, as the software itself, be open-source to ensure transparency and reduce third-party company dependency. Industrial adoption of open-source software – often held back by unjustified quality concerns – is increasing, especially if value-added services are offered [22]. Prominent examples of successful open-source software are Linux or the databases MySQL and InfluxDB.

The dogma of limiting device access to software installed on a local computer is outdated. Devices should be accessible by the local network in a device-as-a-service fashion. Henceforth, the controlling software should be readily accessible from any eligible end user device. A further major advantage of this web-based concept is the resulting operating system or platform independence.

In the age of information, all data is important and data recording should start at the lowest level: the device level. Raw data and meta-data should be collected not just in regulated environments, but in academia and research as well [15]. Gathering comprehensive datasets is time-consuming and prone to human error. Thus, any laboratory software should be capable of linking certain streams of data to a selected database which can be connected to the laboratory or company LIMS.

The SiLA Browser (UniteLabs AG, Basel, Switzerland) is a free-to-use closed-source IoT application that enables discovery and direct control of SiLA 2 devices. However, it is unsuitable for workflow automation due to the lack of a scripting environment or workflow editor. Other notable closed-source software solutions are niceLab (EQUIcon Software GmbH, Jena, Germany) and zenLab® (Infoteam Software AG, Bubenreuth, Germany), the successor of the iLab [23], whereas the zenLab®, a scalable middleware automation framework, follows the similar core principles as the proposed software, but accompanied by the restrictions of proprietary vendor software. The SiLA 2 Manager provides a free, open-source alternative for researchers, engineers, educators, and small laboratories in general, to automate their workflows. In comparison to the SiLA Browser, the proposed software is a functional open-source alternative and vast expansion.

Workforce change management in the laboratory environment in regard to new software solutions is an often underestimated barrier due to a low level of computer and programming skills. This further raises the importance of standardized, plug-and-play like device interfaces and intuitive graphical user interfaces. Intuitive, browser-based solutions that offer easily accessible device-as-a-service functionality are needed to transform the laboratory environment into an automated workspace [24].

2. Software description

2.1. Software architecture

The SiLA 2 Manager is based on a client–server architecture, with a python backend and a typescript frontend. The user interacts with the software through the web-based frontend which communicates with the backend API by HTTP and WebSockets. The frontend is secured by using the open authorization protocol OAuth2. Authentication of authorized users is managed by issuing a JSON Web Token. The automatic token timeout and renewal policy can be configured to meet the required security level. Fig. 1 shows the relationship between the user and the involved software systems. A user interacts with the application by using the web-based graphical interface to discover and control available services or execute experimental scripts. The SiLA 2 Manager relies on multicast DNS service discovery (zeroconf) and is able to discover SiLA servers that multicast their service by default. A generic SiLA 2 python client uses the provided information to connect to any of the discovered services. Once connected, the user can explore the devices features, execute commands and integrate the devices in experimental workflow scripts.

Furthermore, SiLA 2 services can be linked to InfluxDB databases (v.1.7) [25] to automatically store selected process information for the duration of an experiment. This stored service data, e.g. from a device, can be accessed from within the scripting environment with the respective python database client. Stored information can also be accessed and visualized via the web-based database browser chronograf, which is part of the influxdata stack.

To allow further expansion and customization, the frontend and backend have been strictly separated. The python backend uses the FastApi web framework to transfer data via HTTP and WebSocket to and from the Angular frontend. Fig. 2 shows the different components of the system and the interactions between them. Each component is a container, e.g. a module, that can be executed independently.

To store persistent application data, e.g. service or booking information, a relational SQL database is used because of its high reliability, guaranteed data consistency and ease of use. In addition to this, the relational data model, with its ability to efficiently join rows from different tables, makes it a natural fit for the persistent application data of the SiLA 2 Manager. The open-source database PostgreSQL (v.13) was selected because it is fast and has excellent SQL conformance [26]. Most No-SQL database alternatives benefit from high horizontal scalability, i.e. they can distribute data over many servers and load balance the access to it. However, this does not provide any real benefit for the proposed use case since there are no large distributed datasets. The disadvantage of making it more difficult to ensure precise data consistency would outweigh the advantages provided by the horizontal scalability of NoSQL database types. Further, the ability to have very loosely defined non-structured data structures is of no benefit and thus not a relevant decision criterion for the presented application.

The application consists of several subprocesses as shown in Fig. 2. The in-memory database Redis (v.6.0.9) is used as a message broker to enable interprocess communication. An in-memory, No-SQL database was chosen over an SQL database because information transmission between 2 components of the system rather than persistent storage is required. Redis is a well-established and high-performance database [27]. A major advantage, as compared to other messaging libraries, is that Redis can be used for other complex operations, like fast in-memory storage or caching, as well.

The user is provided with an Angular client for easy access to the functionalities of the system, which are exposed over an HTTP

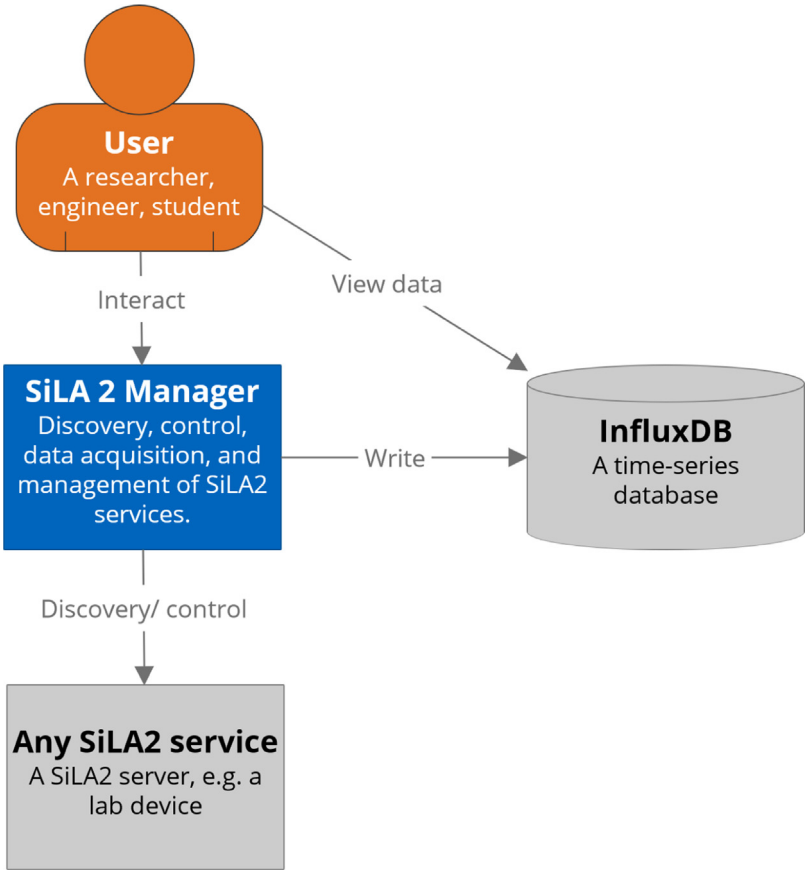

Fig. 1. The context diagram showing the scope of the software and its relation to the user and other software systems. The user interacts with the software through the web-frontend and may view process data through web-frontend of the database. The SiLA 2 Manager discovers new services in the network via the mDNS multicast server registrations of the SiLA servers and connects to the services via gRPC using a generic SiLA client. Recorded process data is forwarded from the SiLA 2 Manager to a selected InfluxDB database.

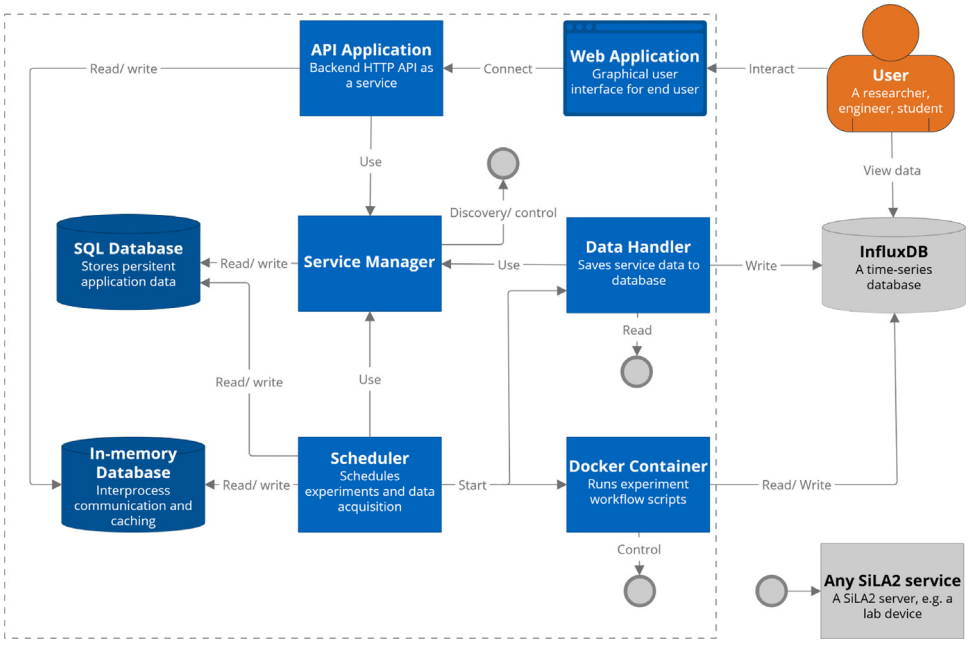

Fig. 2. The container diagram of the SiLA 2 Manager is a detailed description of the software system introduced in Fig. 1. The web-frontend makes API calls to the backend API. The API forwards these calls to the main sub-system, the service manager. A PostgreSQL database is used to store application related data. Temporary and quickly changing data storage, such as device status and log data, is handled by the in-memory database Redis by using publish and subscribe (PUBSUB) channels. The scheduler is an independent service that controls experiments in a docker container and initiates data transfer via the data handler.

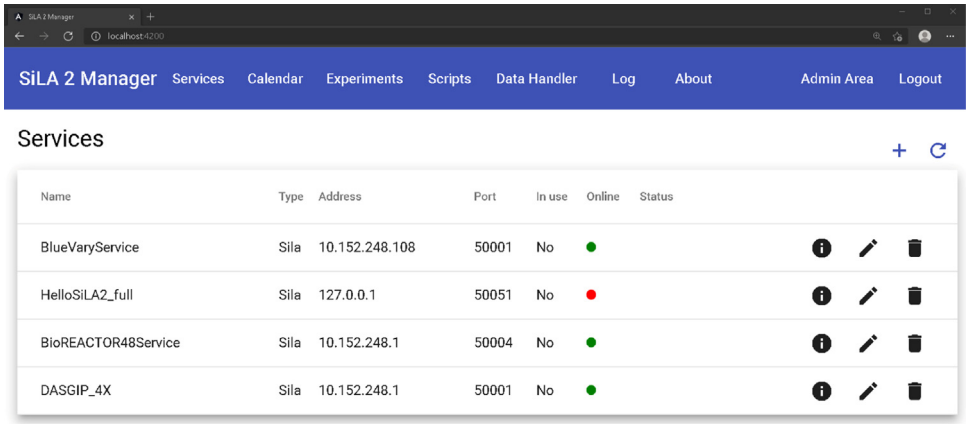

Fig. 3. The homepage shows registered services and the top layer of the service tree that contains more detailed information on the service's features, commands and properties.

Rest-like API through the FastAPI Python framework. The service manager handles detection of services on the network, communication with services, and storage and retrieval of persistent information (e.g. service, experiment, and time-series database connection details) to and from the PostgreSQL database. The angular frontend is served by nginx [28], which is also used as a reverse proxy for the FastApi backend. Angular was chosen because it is a proven technology to create modern single page web applications.

The scheduler is an independent subprocess that is responsible for the execution of experiments and the initialization of the data handler and docker containers. The scheduler periodically checks for new experiment booking entries in the PostgreSQL database and queues it in its in-memory scheduling list. With the provided buttons to start and stop experiments, the user can interact with the scheduler directly over a WebSocket connection to the backend, which itself communicates with the standalone scheduling process over the in-memory database Redis. When an experiment is ready to be scheduled, the scheduler loads the corresponding data and script from the PostgreSQL database and launches a docker container and an associated data handling process. The user-provided script contains a predefined entry point, which gets called with a dictionary of instantiated device objects and their name as parameter.

A docker container is used to execute a Python script. SiLA services can be incorporated into this user script to control the laboratory devices. Experiment associated script and service data are loaded from the SQL-database and copied into the docker container as a tar archive file at the beginning of an experiment. Docker allows the execution of python scripts in isolation; therefore, the scripts cannot interfere with each other or compromise the host system. Containers can also be run independently of the host operating system. This is crucial to guarantee server security and stability. The default python docker image (Python 3.8.3-alpine) is used as it is an appropriate base image for most use cases and is based on the lean Alpine Linux distribution. At the same time, the data handler is started and periodically saves user-specified data from the services into an InfluxDB time-series database. Polling intervals and the selection of data to be stored are configured in the data-handler.

A detailed documentation of the software architecture, the installation process and several application examples are available in the software documentation in the Git-repository or online on readthedocs.io (https://sila2-manager.readthedocs.io/en/latest/).

2.2. Software functionalities

The homepage for the SiLA 2 Manager software is shown in Fig. 3 and can be accessed, upon login, by entering the IP of the host computer followed by the default port in the URL-address bar of any modern browser, e.g. 127.0.0.1:4200 for a local host. There are six main tabs, each dedicated to one of the core functionalities as well as an about page with instructions and additional information. The core components are: Services, Calendar, Experiments, Scripts, Data Handler, and Log. User management and user authentication is handled in the tabs Admin Area and Login/Logout. The main page is the Services tab in which all registered services and respective important information is shown. New services are discovered and added in this view and the expanded card of the services will open the more detailed service tree view and the interactive command execution environment.

Service manager and browser - Services

The core functionality of the proposed software is service discovery, control, and management. All of these are available on the main page. SiLA services can be added either by using the discovery mode or by manual specification of the server address and IP. Previously added SiLA services are listed as shown in Fig. 3. When adding a new service, the service discovery initiates a scan of the local network and returns the registered SiLA servers within. SiLA services are registered on the unicast domain name system (DNS) server and can be identified by their hostname which contains '*sila.local*' as terminal identifier. Services are assigned the server name but can be renamed upon addition. Furthermore, the service is assigned an internal universally unique identifier (UUID) and the service details are stored in the PostreSQL database. A dynamic SiLA 2 python client is used to establish a connection with the service and access information regarding implemented features and functions. According to the SiLA standard, the features of a SiLA service are defined in the feature definition language (FDL), which is based on the extended markup language XML. Several FDL-files describe the full functionality of the service. During the initial connection to the SiLA 2 server, these files are queried by calling standard functions of the SiLAService feature, a feature that is always implemented by default. All further communication with the SiLA 2 server is realized by the dynamic client which is part of the SiLA 2 python library. Other implementations, such as C# and the C++, also offer a generic (dynamic) client as part of the distribution.

Fig. 4. The bottom layer of the service tree allows direct execution of command and property calls. If no parameter is supplied for command calls, the default parameter will be used. in case of device services, accidental execution may have serious unwanted consequences; thus, execution must be confirmed.

The new device will appear in the service list and, if a connection is established successfully, will be shown as online. Clicking on the service name or the information icon, the service detail tree is expanded as shown in Fig. 4. The service tree shows all implemented features of the service and the functions they comprise. According to the SiLA specifications, a SiLA service must implement features containing functionally related commands and properties, which can be observable or unobservable. Standard features are implemented by default, whereas custom features are device specific. Each feature can be further expanded to show command and property call information. While service properties are static (e.g. a device serial number) or dynamic values (e.g. device time or status), commands require a parameter to be supplied with the command call. Equipment that requires special security considerations can be further secured by mandatory incorporation of the SiLA Standard Features "AuthenticationFeature" and "LockFeature" in the SiLA Server of that device, making command execution only possible with valid access credentials on unlocked devices.

Scripting environment - Scripts

The scripting environment enables the user to write automation scripts that incorporate the registered services as shown in Fig. 5. In contrast to device specific vendor software, the scripting environment's functionality is non-restrictive. Scripts can be uploaded, created, and edited with the embedded python editor. Each user only has access to their own scripts to avoid unwanted access or script execution by unqualified users. The editor is based on the open-source monaco code editor and supports type hinting, auto-completion, and syntax control for python-based scripts. Devices are automatically instantiated, and commands and properties can be called using the SiLA python syntax of the generic client. For ease of use, the respective usage syntax is explicitly stated in the service detail view of each call. External data-sources and non-standard python packages can be imported but must be specified in the dockerfile.

Workflow creation - Experiments

Processes can be executed by scheduling experiments. An experiment consists of an experiment name, start and end date, a workflow script, and a selection of services. New experiments can be added which will open the experiment definition window (Fig. 6). In this window, the desired script to be executed, as well as the required services are selected. The start and expected end-time of the process are specified. The used services will be reserved and blocked for the requested time slot. At the time of execution, the specified script is run in a docker container and the services are marked as "currently unavailable" in the service list overview. Automated experiment workflows require access to device services, a script that orchestrates device operation, and a database to store relevant information. The experiments view enables the user to setup such workflows and schedule their execution. In a commercial setting, each part of the workflow, i.e. the SiLA 2 servers of the devices and the workflow script, would need to be validated and access to these workflows be restricted by using the inbuilt user management and authentication system.

The main view of the experiments tab shows a list of scheduled experiments, accompanied by the most important information such as the start and end time of the experiment, booked services, the user script and the current status of the experiment. Scheduled experiments automatically create a booking entry in the calendar view. It is possible to start experiments before their scheduled starting time by pressing the play icon. A running experiment can be aborted prematurely. An embedded terminal displays the output of the docker container to observe the current status of the script execution.

Bookings and calendar overview - Calendar

If a service is assigned to a specific experiment, a booking is automatically created for the entire timeframe of the experiment. A service can only be assigned to an experiment if it is available throughout the experiments start and end time. The calendar page visualizes these bookings and provides the user with additional information on the booking, such as the respective experiment, script and the user who created it. Furthermore, it is possible to create bookings manually. In a future release, shared or part-time service access will be available for bookings as well. It is possible to delete bookings manually. Even automatically created bookings can be deleted. However, this would circumvent the in-built security mechanism and may lead to services being accessed by multiple experimental scripts at the same time, as the script does not communicate with the booking system anymore once started. This may be desirable if the SiLA service implements a virtual device or some other software solution, such as a DoE or data analysis software.

Database connection - Data handler

Data collection is enabled by default, although the user should specify which data is collected. This configuration is done on the data handler page shown in Fig. 7. InfluxDB databases can

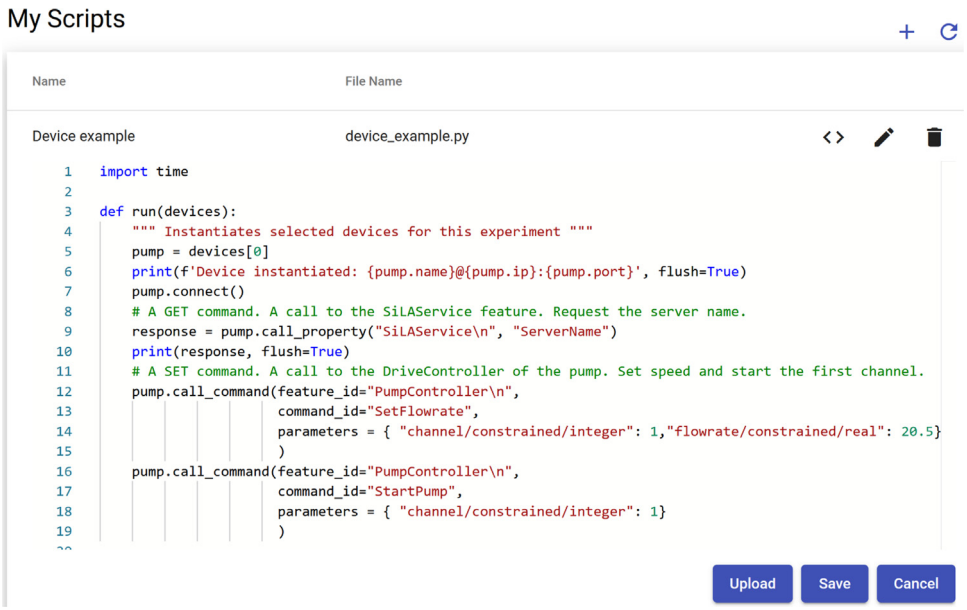

Fig. 5. An interactive code editor is used to view, create and edit workflow scripts. Scripts can be uploaded and saved. The device services are automatically instantiated and passed to the *run()* function as attribute. They are accessed either by key index in the order of assignment (see experiment overview for order) or by key name, which is identical to the name the device services are listed as in the service list overview. In this example code snippet, a peristaltic pump is imported. A get and a set command is executed to demonstrate the interface syntax. The command syntax is shown in the detailed service list view (See Fig. 4) for each command to further simplify the process.

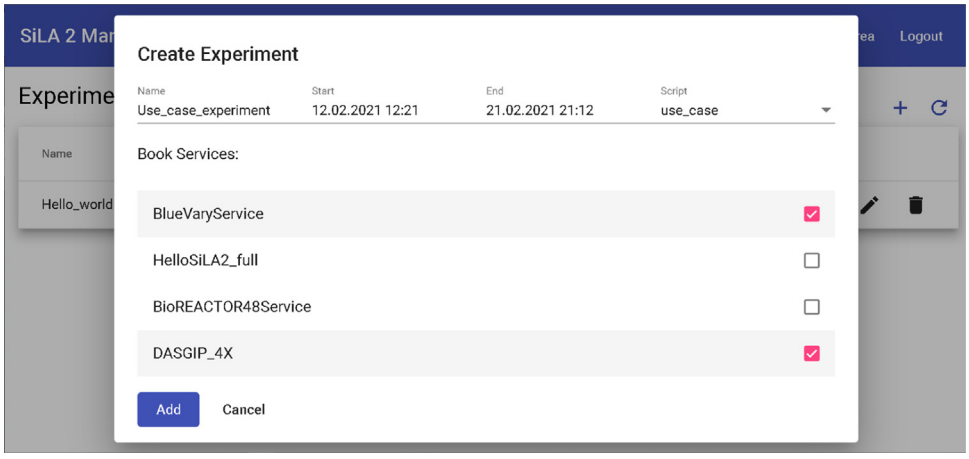

Fig. 6. An example experiment is defined by selecting a workflow script, the used services, and by specifying the start time of execution and the anticipated end time. The logging output of the script is forwarded to the frontend and can be viewed in a terminal when the experiment is expanded.

be registered and linked to services. InfluxDB is a time-series database that is well suited for experimental time-series data [29, 30]. To use this feature, an InfluxDB server must be running within the network. Providing the connection details to the SiLA 2 Manager is sufficient. A username and password can be added optionally for additional security. A registered database can be linked to a service to setup automated data transfer by clicking the link symbol. Data transfer is started when the booking of a service commences, i.e. the experiment the service is used in, is started.

The data handler will repeatedly execute the SiLA calls in the user-specified polling intervals and store the responses in the linked database with experiment name, service name, and user-name as tags. Several options are available to configure the data calls, the data type, and the polling interval. The SiLA 2 Manager is service agnostic and the functional response of individual calls is unknown. Therefore, the user must specify the calls that should be executed periodically and must deactivate unwanted calls, such as certain set-commands. If set-commands are to be called, a parameter can be supplied.

Most types of data can be classified as either meta-data or measurement data. Typically, meta-data does not need to be queried frequently. In most cases, requesting meta data (device ID, calibration data, etc.) hourly or just once at the beginning of an experiment is sufficient. Measurement data (Temperature, pressure, etc.) on the contrary is usually queried on a more frequent basis. The data handler distinguishes between the two data types. Since there is no way to distinguish the type of data queried by a call automatically in a reliable fashion, the user can specify the type for each command using the meta-checkbox. Depending on the selection, a default value is implemented (1 h for meta-data, 30 s for measurement data). Different users have different needs regarding polling intervals. Therefore, the defaults can be overwritten to transfer data according to a custom polling interval.

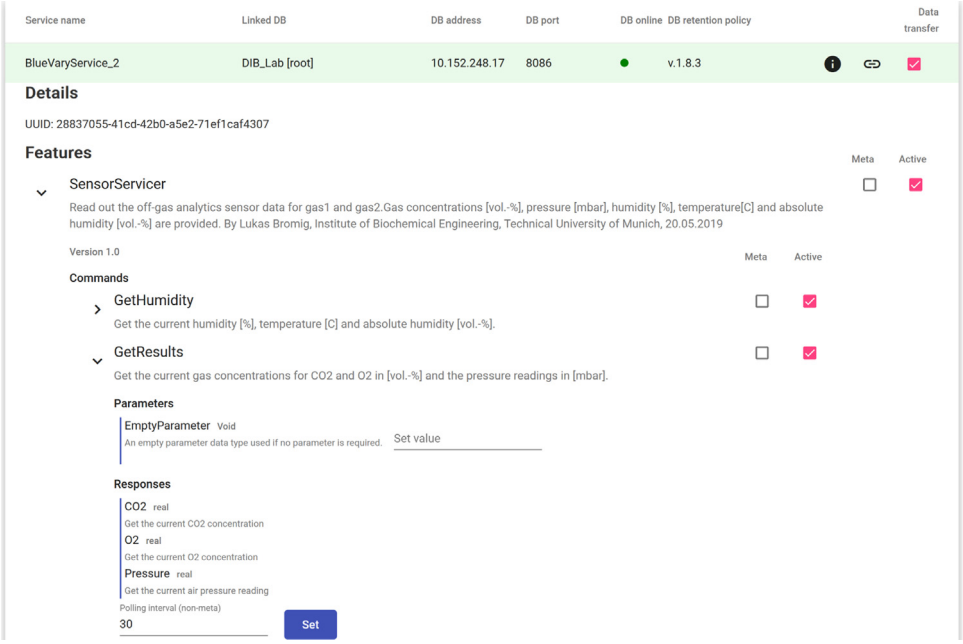

Fig. 7. The service tree for the data handler configuration. Data transfer can be de-/activated for each property and command. The user assigns a meta-data or measurement data flag which determines the polling interval used. If no custom polling interval is supplied, the default polling intervals are used. Some command calls, like most set commands, may require additional parameters. The example above shows a configured off-gas analysis device that is linked to an InfluxDB. Sensor results related to gas concentrations, pressure, and humidity are polled in the default non-meta data interval of 30 s.

The data handler configuration is stored in the PostgreSQL database. The lower level of the data handler service tree is shown in Fig. 7. A customized configuration is crucial to disable the undesired execution of set commands or the storage of unnecessary data.

The data handler simplifies data-acquisition and encourages collection of all data and meta-data for improved data integrity. The separation of the data acquisition from the user script used in the experiment has several advantages:

1. The query calls are not part of the user-script, improving readability and making the script shorter.
2. Reduces the amount of code that needs to be written by the operator.
3. Data-acquisition is out-sourced to a separate process. This way data-acquisition is guaranteed to continue in case an experiment crashes.
4. The data can be easily accessed from within the user-script by a respective python client library. An example script is provided in the "Scripts"-section of the application.

2.3. Example and application use cases

Example use cases

Use cases can stretch from very simple routine tasks to highly complex, event-driven process control workflows. A simple use case is scheduled device calibration (e.g. a digital scales) or self-maintenance/cleaning tasks, or the collection of basic sensory laboratory data such as room temperature and pressure. Setting up the latter would be the simplest use case. Add the sensor device to the SiLA 2 Manager, customize the data you want to collect, link the device to a database, schedule an experiment with an empty script and start the experiment. The data-handler will now collect the data and meta-data of your device and store it in your database.

Application use case

The impact of intermittent substrate feeding on microbial fermentation processes is a key factor for successful scale-up in bioprocess development and remains an active area of research [31,32]. Frequently changing nutrient availability leads to rapidly changing metabolic states and may result in population heterogeneity [33]. The effect of intermittent substrate feeding on microbial process performance can be studied in laboratory-scale parallel reactor systems to mimic the quasi-intermittent feeding in large scale reactors due to nutrient gradients caused by non-ideal mixing.

Using a parallelized and miniaturized bioreactor system with 48 single-use stirred-tank reactors on a mL-scale (BioREACTOR48, 2mag AG, Munich, Germany) automated by a LHS (Hamilton STARLet, Hamilton Bonaduz AG, Bonaduz, Switzerland) enables a fast analysis of the process parameter space and thus the quantification of the effect that the substrate feeding strategy has on product formation and population heterogeneity. In this application example, a scale up experiment is performed for a selection of substrate feeding intervals, to ensure that the observed influence on microbial protein expression is the effect of the varying feeding interval length and that this effect is independent of reactor size. To achieve this, the substrate feeding strategy must be emulated as closely as possible on the L-scale. Substrate addition on the mL-scale is realized by pulse addition with the pipetting robot. A dynamic scheduling algorithm coordinates this feeding event with other tasks, such as sampling, at-line measurements or pH-control, resulting in small variations of the feed interval duration. The challenge at hand is the direct translation of these irregular events into pump action on the L-scale.

The experimental setup consists of a parallel stirred tank bioreactor system with four bioreactors (DASGIP Parallel Bioreactor System, Eppendorf, Hamburg, Germany) that is connected to a proprietary control computer and an off-gas analysis device. The vendor software DASware® control (DASware® control, Eppendorf, Hamburg, Germany), with its integrated OPC-UA interface, serves as access point for the SiLA server of the DASGIP bioreactor

system. Peristaltic pumps used for substrate addition, pH-control, and addition of antifoam are periphery devices of the reactor system. A gateway module, based on a BeagleBone Green microcomputer and provided by Porr et al. (2020) [20], is connected to the off-gas analysis device by RS-232. The corresponding SiLA server is hosted by the microcomputer and uses the serial interface for device control. Both SiLA servers are based on the *sila2lib* release 0.2.5 of the open-source SiLA python repository and the code was generated using the python *sila2codegenerator* version 0.2.0.

The vendor software of the L-scale reactor system offers only limited scripting capability. Scheduling of non-trivial substrate addition events, especially if based on external files, is not possible. Furthermore, the used off-gas analytics are from a different vendor (BlueVary, BlueSens, Herten, Germany). Although the off-gas concentrations are not used actively for process control in this particular use case, it is desirable to store and monitor all generated data in and from a single source, i.e. the laboratory database (InfluxDB, Influxdata, San Francisco, USA). In this setup the SiLA Manager is used for data and meta-data acquisition of both devices as well as the advanced orchestration of the feed pump events, whereas the simple tasks of pH-, DO-, and temperature control are carried out by DASware® control. Establishing full control with the SiLA 2 Manager is possible but was not realized to keep the application example as simple as possible.

The resulting workflow is written in form of a python script and is uploaded to the SiLA 2 Manager. DASware® control is started and device connection established and verified. At the scheduled experiment start time, the docker container is created and the user script is executed automatically. Substrate additions performed by the liquid handling station during the 48 preceding experiments on a mL-scale are stored in the same database that is used for this experiment. Thus, the timestamps and the added substrate volumes of the discrete feeding events can easily be extracted from the database using the *influxdb-client* within the user script. For reasons of simplicity and reproducibility, a different approach was chosen to avail the required data within the docker environment. Experimental data regarding the substrate feeding events and volumes during the experiment in the BioREACTOR48 were extracted from the database in a csv-file format and manually moved to the data folder of the docker environment directory. The csv-file is provided as part of the use-case example in the Git repository. Files in this directory are automatically moved into the docker container. This feature enables the user access to any kind of file from within the scripting environment.

Any logic necessary to control substrate addition events on the L-scale is based on the provided data and is defined in the python user script a priori. Data acquisition for both devices, the reactor system and the off-gas analysis device, is started automatically according to the selected criteria in the data handler settings. Hence, no additional commands related to data acquisition need to be specified in the workflow script. Workflow progress is monitored through the embedded terminal on the experiments page and measurement data can be visualized through the browser-based database interface chronograf. All process related data is stored with a respective experiment and user tag to enable quick and fast data selection. Further workflow script examples dealing with device and database integration as well as simplified version of the use case described above can be found in the example folder of the project Git repository.

Fig. 8 shows the resulting dissolved oxygen concentration in the fermentation with *E. coli* during a short window of the cultivation and compares the immediate catabolic response to a glucose feeding event on the mL- and the L-scale. It can be seen that the irregular glucose feeding events could be synchronized with each other which enables a direct comparison of the metabolic

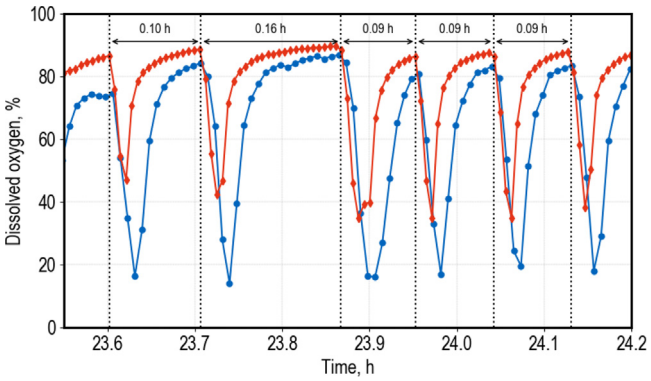

Fig. 8. The effect of intermittent glucose feed events on the dissolved oxygen level is shown for cultivations of *E. coli* in the mL- and L-scale. Feed events (▪▪▪) are unevenly distributed on the mL-scale (●) due to the dynamic nature of the task scheduling algorithm of the pipetting robot. The feeding pattern is mimicked as closely as possible with a peristaltic pump on the L-scale (♦) to investigate scale-up limitations/criteria. The resulting DO off-set is caused by different oxygen transfer rates. Stirrer speeds of the mL- and L-scale were set at 3000 rpm and 1100 rpm respectively.

response, i.e. the dissolved oxygen concentration. The remaining offset (minimum DO and maximum DO) can be attributed to different oxygen transfer rates of the stirred-tank reactors at mL- and L-scale and the different DO sensors. Dry cell mass concentrations were similar on both scales within the estimation error (14.3 ± 0.4 gL^{-1} on the mL- and 15.8 ± 0.9 gL^{-1} on the L-scale at 24.3 h cultivation time).

The presented use case only involves two devices. A more sophisticated example, including more devices, event-based actions, and active use of the influx data for process control would have been beyond the scope of this article. However, the presented example clearly demonstrates the advantages of simple integration and automated, centralized data acquisition.

3. Impact

The proposed software enables automation and data acquisition of common tasks and complex scientific experiments. Based on the SiLA 2 standard, it reduces integration effort of laboratory devices. Its intuitive GUI design and expandable software framework reduces the necessary experience to automate experimental setups in the long run. Device integration is decoupled from developing automation workflows. Furthermore, it is a beneficial development tool for testing of self-written SiLA servers. This software is published under the MIT license to allow for continuous application in science and industry. Addressing a common laboratory challenge results in a broad application spectrum beyond disciplines in biotechnology, the life and chemical sciences.

4. Conclusions and future development

The proposed software is mainly, but not exclusively, aimed at laboratories of the life sciences, biochemical sciences and chemical sciences working in research and development, that are planning to digitize their laboratory infrastructure and create automated, SiLA-based, workflows including devices from multiple vendors. Programming-agnostic users can use the basic SiLA-browser functionality, with its service discovery and direct control capability, while the workflow creation requires basic scripting experience in the python programming language. The powerful scripting environment allows the rapid integration of laboratory devices and databases into automated workflows and

empowers the user with the full spectrum of python packages. The data handler component enables the simple setup of database connections and encourages enhanced acquisition of measurement and meta-data according to the FAIR data principles: Findability, accessibility, interoperability, and reusability [34].

User management, device and script-specific access rights are critical when dealing with potentially dangerous or complicated equipment to avoid erroneous or malicious use. The proposed software incorporates these basic features that will be extended in future efforts to improve the overall operational safety of this software. Whereas device specific safety features, such as emergency shutdown procedures or preconditions for device start-up, should be handled by the respective SiLA Server of that device, the orchestration of multiple devices in a complex workflow requires intricate knowledge of the used resources.

The proposed software offers an implementation framework – a model – that bridges the gap between standardized device interfaces, such as SiLA 2, and higher-level control software. By clearly separating the challenges of device integration from customized workflow automation solutions, a holistic approach is chosen. By providing integrated devices as a network service, this software can be broadly applied and reduces development times.

It is a helpful tool for developers and researchers alike and can also be used as basis for a more comprehensive laboratory control software. Future work on this project will include an expansion of the experiment planning component, to enable advanced interaction capabilities with the running docker container, and event-based chaining of several individual experiments. A multi-level structure of main and sub-routines is planned for the scripting environment to increase code reusability. It is planned to include a scheduling algorithm to allow the sharing of resources during the runtime of multiple, simultaneously active workflows. SiLA 2 implementations are being actively developed and are quickly progressing towards meeting the full standard specifications, hence this software will be maintained to ensure long-term compatibility.

CRediT authorship contribution statement

Lukas Bromig: Conceptualization, Software, Validation, Methodology, Data curation, Writing – original draft, Visualization, Supervision, Project administration. **David Leiter:** Software, Validation, Methodology, Data curation, Writing – review & editing. **Alexandru-Virgil Mardale:** Software, Validation, Methodology, Data curation, Writing – review & editing. **Nikolas von den Eichen:** Investigation, Formal analysis, Data curation, Writing – review & editing, Visualization. **Emmeran Bieringer:** Investigation. **Dirk Weuster-Botz:** Resources, Writing – review & editing, Supervision, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors thank the German Ministry of Education and Research (BMBF) for funding within the national joint research project Digitalization in Industrial Biotechnology (DigInBio). Grant number: FKZ 031B0463B.

The authors thank Dr. M. Porr, Institute of Technical Chemistry, University of Hannover, for providing the gateway modules used for the integration of legacy devices.

Lukas Bromig and Nikolas von den Eichen thank for the support provided by the TUM Graduate School (Technical University of Munich, Germany).

References

[1] Neubauer P, Glauche F, Cruz-Bournazou MN. Editorial: Bioprocess development in the era of digitalization. Eng Life Sci 2017;17(11):1140–1. http://dx.doi.org/10.1002/elsc.201770113.

[2] Chapman T. Lab automation and robotics: Automation on the move. Nature 2003;421(6923):661–3. http://dx.doi.org/10.1038/421661a, Number: 6923 Publisher: Nature Publishing Group.

[3] Haby B, Hans S, Anane E, Sawatzki A, Krausch N, Neubauer P, et al. Integrated robotic mini bioreactor platform for automated, parallel microbial cultivation with online data handling and process control. SLAS TECHNOL: Transl Life Sci Innov 2019;24(6):569–82. http://dx.doi.org/10.1177/2472630319860775, Publisher: SAGE Publications Inc.

[4] Selekman JA, Qiu J, Tran K, Stevens J, Rosso V, Simmons E, et al. High-throughput automation in chemical process development. Annu Rev Chem Biomol Eng 2017;8:525–47. http://dx.doi.org/10.1146/annurev-chembioeng-060816-101411, Publisher: Annual Reviews.

[5] McMullen JP, Jensen KF. Integrated microreactors for reaction automation: New approaches to reaction development. Annu Rev Anal Chem 2010;3(1):19–42. http://dx.doi.org/10.1146/annurev.anchem.111808.073718.

[6] Puskeiler R, Kusterer A, John GT, Weuster-Botz D. Miniature bioreactors for automated high-throughput bioprocess design (HTBD): reproducibility of parallel fed-batch cultivations with escherichia coli. Biotechnol Appl Biochem 2005;42(3):227–35. http://dx.doi.org/10.1042/BA20040197.

[7] Bliesner DM. Laboratory control system operations in A GMP environment. John Wiley & Sons; 2020.

[8] USFDA. Part 211 - Current good manufacturing practice for finished pharmaceuticals (21CFR211). In: Code of federal regulations. 2019, URL https://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfcfr/CFRSearch.cfm?CFRPart=211.

[9] EudraLex. Annex 11: Computerised systems. In: Good manufacturing practice - medicinal products for human and veterinary use - the rules governing medicinal products in The European. Union Brussels, SANCO/C8/AM/SI/Ares(2010)1064599. 2011, URL https://ec.europa.eu/health/documents/eudralex/vol-4_en.

[10] Hawker CD. Laboratory automation: total and subtotal. Clin Lab Med 2007;27(4):749–70. http://dx.doi.org/10.1016/j.cll.2007.07.010, Publisher: Elsevier.

[11] Porr M, Lange F, Marquard D, Niemeyer L, Lindner P, Scheper T, et al. Implementing a digital infrastructure for the lab using a central laboratory server and the SiLA2 communication standard. Eng Life Sci 2020;n/a(n/a). http://dx.doi.org/10.1002/elsc.202000053.

[12] Fleischer H, Thurow K. Automation solutions for analytical measurements: Concepts and applications. John Wiley & Sons; 2017.

[13] Ampatzoglou A, Kritikos A, Kakarontzas G, Stamelos I. An empirical investigation on the reusability of design patterns and software packages. J Syst Softw 2011;84(12):2265–83. http://dx.doi.org/10.1016/j.jss.2011.06.047.

[14] Wolf A, Galambos P, Széll K. Device integration concepts in laboratory automation. In: 2020 IEEE 24th International Conference On Intelligent Engineering Systems (INES). 2020, p. 171–8. http://dx.doi.org/10.1109/INES49302.2020.9147171.

[15] Gauglitz G. Lab 4.0: Sila or OPC UA. Anal Bioanal Chem 2018;410(21):5093–4. http://dx.doi.org/10.1007/s00216-018-1192-6.

[16] Bär H, Hochstrasser R, Papenfuß B. SiLA: Basic standards for rapid integration in laboratory automation. J Clin Lab Autom 2012;17(2):86–95. http://dx.doi.org/10.1177/2211068211424550.

[17] Sila 2 working group, SiLA2 base repository. GitLab 2018. last visited: 08.03.2021. URL https://gitlab.com/SiLA2/sila_base.

[18] Sila 2 part (a) - overview, concepts and core specification. last visited: 08.03.2021. URL https://sila2.gitlab.io/sila_base/.

[19] SiLA. SiLA standard | SiLA rapid integration. SiLA-Standard 2018. last visited: 08.03.2021. URL https://sila-standard.com.

[20] Porr M, Schwarz S, Lange F, Niemeyer L, Hentrop T, Marquard D, et al. Bringing IoT to the lab: SiLA2 and open-source-powered gateway module for integrating legacy devices into the digital laboratory. HardwareX 2020;8:e00118. http://dx.doi.org/10.1016/j.ohx.2020.e00118.

[21] Ladwig B. Geräteintegration entlang des labor-workflows: Ein neuer standard für das smarte labor. 2020, p. 9.

[22] Nagy D, Yassin AM, Bhattacherjee A. Organizational adoption of open source software: barriers and remedies. Commun. ACM 2010;53(3):148–51. http://dx.doi.org/10.1145/1666420.1666457.

[23] Schmid I, Aschoff J. A scalable software framework for data integration in bioprocess development. Eng Life Sci 2017;17(11):1159–65. http://dx.doi.org/10.1002/elsc.201600008.

[24] Ng IC, Wakenshaw SY. The Internet-of-Things: Review and research directions. Int J Res Mark 2017;34(1):3–21. http://dx.doi.org/10.1016/j.ijresmar.2016.11.003.

[25] Shahid J. InfluxDB documentation. Release; 2019, URL https://buildmedia.readthedocs.org/media/pdf/influxdb-python/latest/influxdb-python.pdf.

[26] Makris A, Tserpes K, Spiliopoulos G, Zissis D, Anagnostopoulos D. Mongodb vs postgresql: A comparative study on performance aspects. GeoInformatica 2020. http://dx.doi.org/10.1007/s10707-020-00407-w.
[27] Jing H, Haihong E, Guan L, Jian D. Survey on nosql database. In: 2011 6th international conference on pervasive computing and applications. 2011, p. 363–6. http://dx.doi.org/10.1109/ICPCA.2011.6106531.
[28] Reese W. Nginx: The high-performance web server and reverse proxy. Linux J 2008;2008(173):2:2, URL https://dl.acm.org/doi/abs/10.5555/1412202.1412204.
[29] Nasar M, Kausar MA. Suitability of influxdb database for iot applications. Int J Innov Technol Explor Eng 2019;8(10):1850–7. http://dx.doi.org/10.35940/ijitee.J9225.0881019.
[30] Giacobbe M, Chaouch C, Scarpa M, Puliafito A. An implementation of influxdb for monitoring and analytics in distributed IoT environments. In: International conference on the sciences of electronics, technologies of information and telecommunications. Springer International Publishing; 2018, p. 155–62. http://dx.doi.org/10.1007/978-3-030-21005-2_15.
[31] Heins A-L, Reyelt J, Schmidt M, Kranz H, Weuster-Botz D. Development and characterization of escherichia coli triple reporter strains for investigation of population heterogeneity in bioprocesses. Microb Cell Factories 2020;19(1):14. http://dx.doi.org/10.1186/s12934-020-1283-x.
[32] Heins A-L, Weuster-Botz D. Population heterogeneity in microbial bioprocesses: origin, analysis, mechanisms, and future perspectives. Bioprocess Biosyst Eng 2018;41(7):889–916. http://dx.doi.org/10.1007/s00449-018-1922-3.
[33] Vasilakou E, van Loosdrecht MCM, Wahl SA. Escherichia coli metabolism under short-term repetitive substrate dynamics: adaptation and trade-offs. Microb Cell Factories 2020;19(1):116. http://dx.doi.org/10.1186/s12934-020-01379-0.
[34] Wilkinson MD, Dumontier M, Aalbersberg IJ, Appleton G, Axton M, Baak A, et al. (2016) The FAIR guiding principles for scientific data management and stewardship. sci data, 3, 160018. 2016, http://dx.doi.org/10.1038/sdata.2016.18.