ADVANCED MATERIALS INTERFACES
www.advmatinterfaces.de

Enabling Modular Autonomous Feedback-Loops in Materials Science through Hierarchical Experimental Laboratory Automation and Orchestration

Fuzhan Rahmanian, Jackson Flowers, Dan Guevarra, Matthias Richter, Maximilian Fichtner, Phillip Donnely, John M. Gregoire, and Helge S. Stein**

Materials acceleration platforms (MAPs) operate on the paradigm of integrating combinatorial synthesis, high-throughput characterization, automatic analysis, and machine learning. Within a MAP, one or multiple autonomous feedback loops may aim to optimize materials for certain functional properties or to generate new insights. The scope of a given experiment campaign is defined by the range of experiment and analysis actions that are integrated into the experiment framework. Herein, the authors present a method for integrating many actions within a hierarchical experimental laboratory automation and orchestration (HELAO) framework. They demonstrate the capability of orchestrating distributed research instruments that can incorporate data from experiments, simulations, and databases. HELAO interfaces laboratory hardware and software distributed across several computers and operating systems for executing experiments, data analysis, provenance tracking, and autonomous planning. Parallelization is an effective approach for accelerating knowledge generation provided that multiple instruments can be effectively coordinated, which the authors demonstrate with parallel electrochemistry experiments orchestrated by HELAO. Efficient implementation of autonomous research strategies requires device sharing, asynchronous multithreading, and full integration of data management in experimental orchestration, which to the best of the authors' knowledge, is demonstrated for the first time herein.

1. Introduction

Ever increasing performance demands necessitate the acceleration of materials science and chemistry.[1,2] Progress within the Materials Genome Initiative,[3] advances in high-throughput experimentation,[4] and proliferation of machine learning have enabled the emergent paradigm of conducting research in materials acceleration platforms (MAP)[5,6]. Within these MAPs different research tasks are accelerated and integrated to efficiently address the ever increasing complexity of materials optimization through targeted materials synthesis, processing, analysis, and insight generation.[7]

Demonstrations of autonomous workflows to date,[8–11] have been based on a single instrument in a single laboratory.[12] This limited purview of the autonomous experimentation is rooted in the laboratory middleware in which orchestration of the laboratory hardware occurs within a single computer-instrument pairing.[11] Some notable examples include ChemOS,[12] which in principle is capable of distributing work across different machines through the ROS[13] backend. This inarguably powerful software does however impose complex software dependencies that grow with increased purview of the experimental platform. While commercial software such as LabView by National Instruments can facilitate programming for instrument automation, it does not meet the needs of the MAP community due to its incompatibility with the open-source development of version-controlled software. In instances where there is no (official) application programming interface (API) for a device, or an instrument's software driver must continually evolve with

F. Rahmanian, J. Flowers, M. Fichtner, H. S. Stein
Helmholtz Institute Ulm (HIU)
Helmholtzstr. 11, 89081 Ulm, Germany
E-mail: helge.stein@kit.edu

The ORCID identification number(s) for the author(s) of this article can be found under https://doi.org/10.1002/admi.202101987.

[+]Present address: Carleton College, Northfield, MA, USA

DOI: 10.1002/admi.202101987

F. Rahmanian, J. Flowers, H. S. Stein
Karlsruhe Institute of Technology (KIT)
Institute of Physical Chemistry (IPC)
Fritz-Haber-Weg 2, 76131 Karlsruhe, Germany

D. Guevarra, M. Richter, P. Donnely,[+] J. M. Gregoire
Division of Engineering and Applied Science
and Liquid Sunlight Alliance (LiSA)
California Institute of Technology (Caltech)
Pasadena, CA 91125, USA
E-mail: gregoire@caltech.edu

M. Fichtner
Karlsruhe Institute of Technology (KIT)
Institute of Nanotechnology (INT)
P.O. Box 3640, D-76021 Karlsruhe, Germany

hardware advancements, ROS and LabView can incur substantial overhead in software management.

Modular software design facilitates community sharing of techniques across sub-fields. For example, organic chemistry uses tailored languages,[14] to express research tasks in a human and computer readable format.[15,16] Sharing of experimental control techniques across such domains requires a high level of modularity in conjunction with the data management-informed design of the experimental control framework.

In addition to the necessity of being able to orchestrate a multitude of laboratory instruments, there is a critical need to be able to trace back all undertaken steps that lead to the acquisition of data or synthesis of a material,[14,17] beyond FAIR[18] guidelines. Experiment provenance management is critical for enabling reproduction of an experiment.[19] Such reproductions or more general sharing of experiment protocols can be enabling of the computer automation of laboratory devices via drivers that provide an abstraction layer between the central software and hardware. If these criteria are met, autonomous inter-laboratory workflows[7] can be deployed and motivate the discretization of an experimental provenance into its elementary instrumental actions. We therefore view the levels of experiment abstraction to be hierarchical in nature.

The hierarchical laboratory automation and orchestration framework was built with the goals of being able to integrate any laboratory device for which a software driver is available or can be written, and to enable any configuration of the devices including serial and parallel experimentation, sharing of equipment across multiple instruments, and orchestration of multiple measurements in multiple laboratories. To facilitate continued adoption of active learning in experiment workflows, the framework is designed for facile switching between human and machine-based experiment selection. The framework adopts a data management wherein all gathered data and all instructions are stored in a "FAIR" way, giving the instruction data the same level of attention as the resulting measurement data. For these requirements to be met, we seek a software framework for communicating with devices hosted or operated on different computers (i.e., some instruments are mutually exclusive to be connected on a PC due to driver constraints). We seek to be platform independent and minimize additional requirements such as extensive software dependencies.

In the present work we describe the hierarchical experimental laboratory automation and orchestration (HELAO) framework to address the needs of next-generation experiments. At a high level, the modularity of HELAO is built upon a widely used web framework called fastAPI[20] as shown in **Figure 1**. The main design idea is to represent every device of an instrument as a (asynchronous) web server (Figure 1, right side). Basic functions of devices are exposed to and bundled by actions, which themselves are again web servers. Only these actions are called by an orchestrator executing experiments on one or multiple instruments (Figure 1, left side). For future proofness, HELAO was developed in python 3.8+ with type hinting and pydantic type validation. The modular design allows for the integration of arbitrary devices, including those operated through OPC-UA.[21]

The design guidelines and protocols necessary to orchestrate instrumentation in the laboratory are outlined in the following sections, together with a detailed description of the individual constituents. We demonstrate the orchestration of an active learning run on two instruments and deposited the resulting

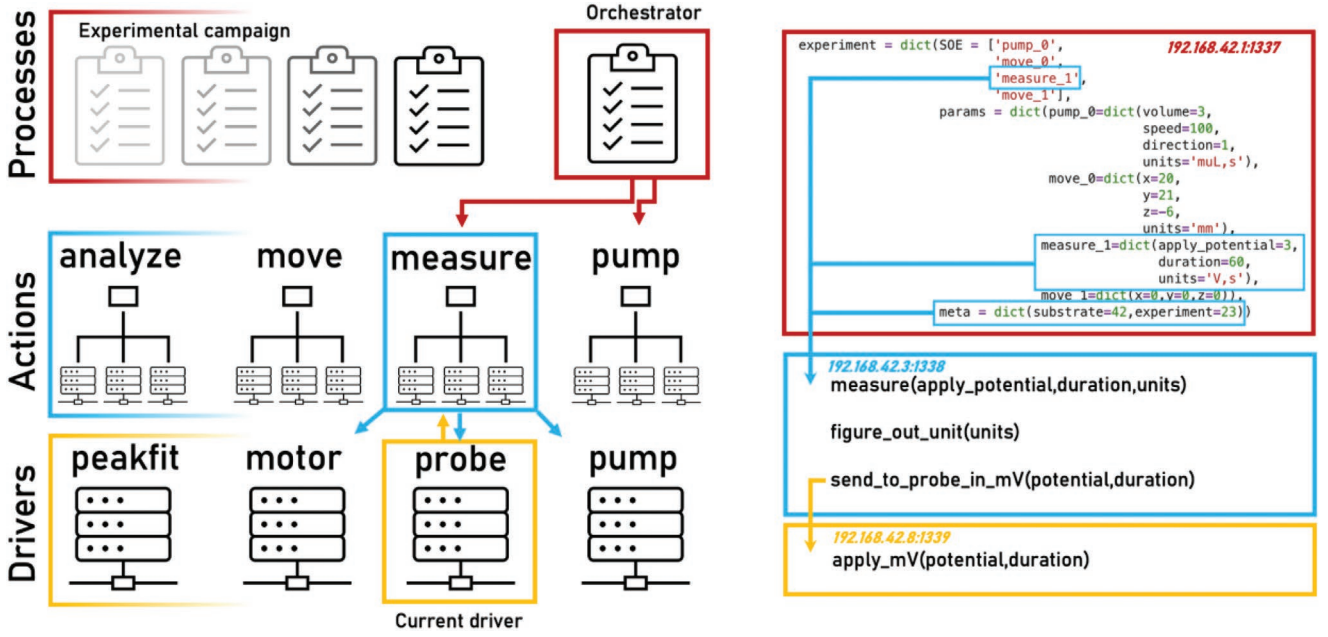

Figure 1. A schematic representation of HELAO where experiments are executed by sequentially calling actions which are high level wrappers for lower level driver instructions. Communication goes hierarchically down from the orchestrator level to actions, which may however communicate among each other, to the lowest level of drivers which can only communicate with actions. The orchestrator, actions, and drivers are all exposing python class functions through a web interface allowing for highly modular and distributed hosting of each item. Experiments are encoded as python dictionaries (a data type) containing a sequence of events (SOE) that outlines in which the actions are to be executed. Many experiments form a process. All actions require parameters and metadata that are all echoed back.

data including all instructions and the code at github.com/helgestein/helao-pub.

2. Results

2.1. Implementation of Hardware Drivers

The aim of HELAO is to be a universal laboratory automation framework, democratizing accelerated experimental research workflows. To this end, the two laboratories at the California Institute of Technology (Caltech) and Karlsruhe Institute of Technology (KIT) started to implement major hardware components amenable for automatization. In **Table 1**, all currently implemented hardware drivers are listed. During development of these drivers, it became apparent that there exist two major types of drivers based on whether their function calls are natively blocking or non-blocking. Those with non-blocking operations typically accept an instruction, execute it, and require the user to ask if the current operation is finished.

A special class of devices is auxiliary (aux) devices. These are broadly defined as software "devices" used, for example, for data analysis, regression, and prediction. These aux drivers could in principle be written for any python interfaceable software or hardware which is necessary for a special experiment, for example, background inference algorithms[22] or special machine learning models.[23]

With the devices available at the time of writing this manuscript, highly complex instruments have and are being built, whose detailed descriptions will be the subject of future work. As an initial example of the scope of the present HELAO implementations, the operated instruments are comprised of the devices shown in Table 1 that include four scanning droplet cells (SDCs) at Karlsruhe Institute of Technology (KIT) (each consisting of lang, autolab, pump, force, aux, kadi), one SDC at Caltech (galil, gamry), a coupled Raman and FTIR spectrometer (owis, ocean, arcoptix, aux, kadi), a battery cycler (arbin, aux), and a coin cell assembly system (mecademic, rail, arbin, arduino, aux).

2.2. Hardware-in-the-Loop Active Learning

A hardware-in-the-loop demonstration run of HELAO is shown in **Figure 2**. The instrument is copied two times where one setup was run in a fume hood and another one was run in a glove box. The two instruments share a common learner and optimizer, which are controlled along with both instruments by a single orchestrator. An example video of a parallel active learning run can be found in the Supporting Information. To demonstrate the operation and identification of a known global maximum, the potentiostat driver in each instrument was replaced by a synthetic data generator. This synthetic driver returns a scaled Schwefel function[24] depending on the position where the SDC touches down on a substrate, providing the source data with which the active learning server identifies the next target substrate position **3**.

The active learning run is stopped once a threshold value (top percentile) is found. Actions in this run consist of, for example, "move to waste", "remove the droplet", "move to sample offset", "move to the defined point", "move down to substrate", "get output value", "predict the next best position using active learning algorithm." The hdf5 file generated during this run was recorded on 05.10.2021 and has been uploaded to KaDI4Mat upon completion of the session under the records 20287 and 20280. Public release of the dataset[25] with the https://doi.org/10.6084/m9.figshare.16798177.v1 had been triggered on 09.10.2021. The hdf5 file for this run may also be found in the Supporting Information.

One experiment takes ≈108 s. Depending on the number of datapoints the learning step requires more time. During the

Table 1. Currently implemented devices in the laboratories at KIT and Caltech. Instruments built from this include scanning droplet cells, high-throughput spectrometers, and a battery assembly robot. The extreme modularity allows us to mix and match any of these devices by simply defining a sequence of events, that is, to build an integrated SDC and spectrometer or a sample exchange robot without code changes to HELAO. For each device we note the communication protocol and the physical quantity being controlled and/or measurement. We also note whether the instrument is "natively blocking" meaning that the device is unable to process new commands until the currently running command is finished.

Device name	Type	Communication	Measures/Controls	Manufacturer	Natively blocking
lang	Motion	.net API	Position	Lang GmbH	No
galil	Motion, IO	TCP/IP	Position	Galil Motion Control Inc.	No
owis	Motion	Serial commands	Position	Owis GmbH	No
mecademic	Motion	Python TCP/IP API	Position, state	Mecademic Ltd.	no
rail	Motion	TCP/IP	Position	Jenny Science AG	No
autolab	Potentiostat	.net API	Electrochemistry	Methrohm Autolab B.V.	Yes
gamry	Potentiostat	.dll for serial communication	Electrochemistry	Gamry Instruments Inc.	Yes
arbin	Potentiostat	autohotkey	Electrochemistry	Arbin Inc.	No
pump	Pumping	Serial commands	n.a.	CAT engineering GmbH	No
arcoptix	Spectroscopy	.dll api	IR spectra	arcoptix S.A.	Yes
ocean	Spectroscopy Raman	Python package	Raman spectra	ocean insights GmbH	Yes
force	force sensing	Serial commands	Force	ME Meßsysteme GmbH	n/a
arduino	Relays, I/O	Python package	n.a.	arduino	No
kadi	Data management	Python package	n.a.	KIT	Yes
aux	Machine learning and analysis	Python package	n.a.	n.a.	Yes

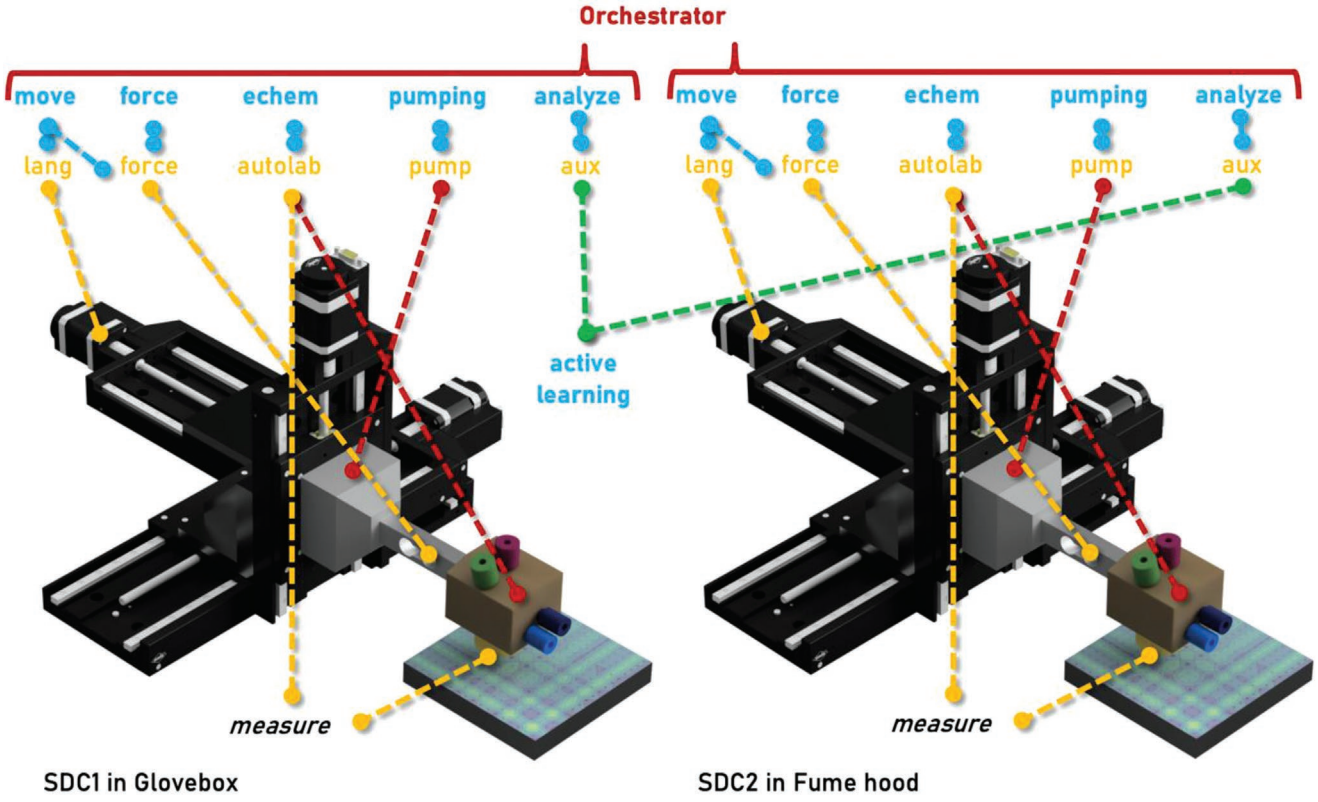

Figure 2. Schematic drawing of the HELAO hardware-in-the-loop active learning run with two instruments running parallel and the corresponding actions, drivers, and orchestrator. The red dashed lines illustrate drivers that were removed from HELAO for the demonstration presented herein, where the pumps were not operated and the potentiostat, which would typically provide the primary measurement data, was replaced by a synthetic data driver that returns a function value from the Schwefel function depending on the visited substrate position. The active learning action and driver are shared among the instruments/threads.

measurement, all data is constantly logged from all devices and subsequently uploaded to the data management repository (KaDI4Mat). The overall time required for the entire run was a little less than 3 h and allows for a fine-grained analysis of what action consumes the most experimental time as shown in **Figure 3**. From this analysis, it is for instance evident that motor movement between measurements consumes a substantial fraction of the experimental time, motivating efforts to enable faster movement. With 3856 s for the sequential run there is a significant speed up when the experiment is run in parallel where the instrument 1 (in the fume hood) takes 2041 s and the instrument 2 requires 2424 s to complete. As is evident from these numbers the speedup is a little less than 2× for a parallel active learning run as asynchronous locks and the machine learning consume some of the time. To the best of our knowledge this is the first demonstration of an active learning run involving two spatially distributed instruments involving more than one operational PC.

In this demonstration, the two instruments perform the same type of measurement in the same search space with the same active learning acquisition function, which is a simplification of our vision of enabling the active learning to incorporate multiple types of data and to make distinct decisions' policies for each instrument.0 For example, a property-measuring instrument and a structure-measuring instrument could be effectively combined for an accelerated structure–property mapping. This concept requires an active learning framework that chooses different targets for property and for structure measurements while unifying the distinct data sources. HELAO is designed to deploy such advanced modes of experimentation as the field of autonomous materials research evolves.

3. Discussion

Herein we present a versatile, stable, and modular approach to laboratory automatization that offers capabilities to deploy autonomous experimentation in materials science. The framework was built using modern asynchronous programming and operates in a safe hierarchical layout. State of the art server-based communication between laboratory devices is used to ensure maximum modularity and reusability of devices across instruments and laboratories. Higher level sequences requiring the interaction within one or among several devices are wrapped in actions that are exposed as web servers. This design allows for a distributed operation across computers and locations, in addition to being resilient against single machine crashes.

Through utilization of a facile underlying web framework like fastAPI and pydantic type annotation, documentation to most functions is autogenerated. This design allows users also to quickly adopt new devices and actions without the need of installing clients or servers, as drivers and actions can be called through python's built in "request" package or even through the auto generated web documentation. Moreover, this allows any HELAO action-driver pair to be called by virtually any other software as users develop

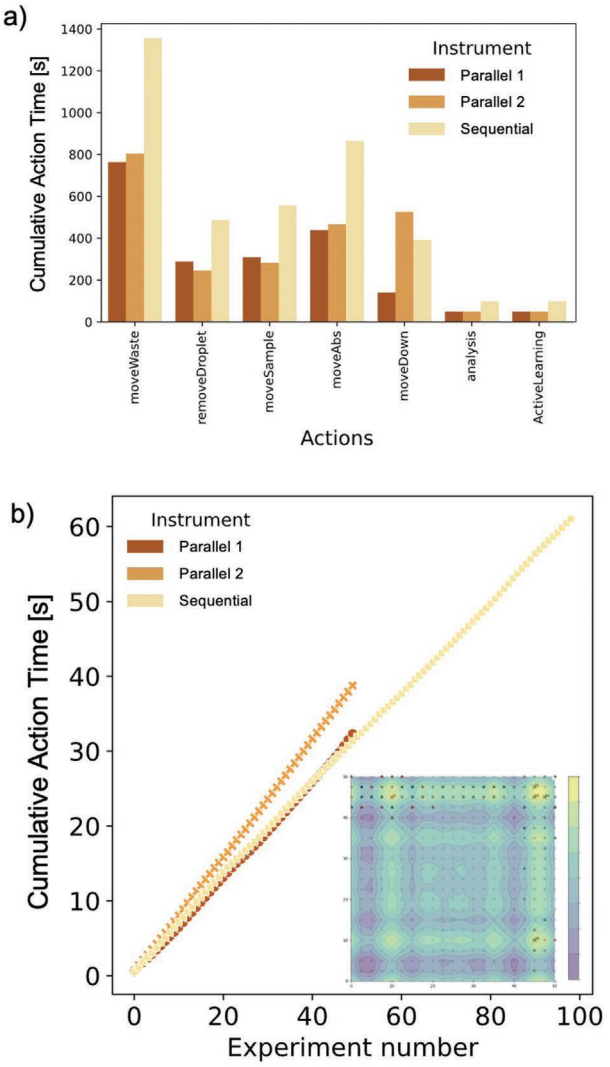

Figure 3. a) Time spent at each action for a sequential and a parallel run with two instruments; b) total time spent per run. The time spent does not form a perfectly straight line as some actions need different time (i.e., movements are shorter or longer). The inset shows the parallel run and highlights visited points in black and red depending on whether they were visited by instrument 1 or 2. The sequence of events for each measurement is typically the order shown in the horizontal axis of (a).

orchestrators to employ complementary modes of research. If users wish to deploy active learning to a device that does not accept standard web requests like OPC-UA (often encountered in industrial settings) fastAPI compatible wrappers can be built.

This high degree of modularity and interoperability is only possible through a very lightweight design that puts relatively few restrictions on the user compared to middleware like ROS or ChemOS. Other competing softwares and frameworks are ARES OS that is currently only demonstrated to operate on a single computer instrument pairing. Another mature and great alternative to the lightweight implementation of HELAO is the bluesky project.[26] Bluesky works with similar hardware abstraction ideas like HELAO, but puts significantly more constraints on a user and is, in our view, more built around streamlining the research process as a whole. However, orchestrating multiple instruments in parallel has not been demonstrated by any other laboratory automation framework. These parallelization efforts will be increasingly impactful with development of optimizers that incorporate uncertainty and multiple optimization strategies.[27]

The framework is built with the goal of being fully FAIR compliant and allows users to rerun an experiment without much or any overhead. We view this degree of data management to be FAIR+. By logging every possible parameter along the entire research process, it is possible to extract utilization figures, find bugs, and determine bottlenecks in high-throughput experimentation. Direct interfacing with data management software has been demonstrated, to the best of our knowledge, for the first time in an autonomous research environment. All data gathered during the active learning sessions has been automatically uploaded upon the completion of the session and is publicly available at figshare[25] and from the Supporting Information of this manuscript. Within the university network all recorded data is made publicly available by default without an embargo period as a statement to encourage more data sharing. HELAO is demonstrated to be stable and versatile and is published under the LGPL license at https://github.com/helgestein/helao-dev. Stand-alone example configurations with reference driver implementations and a how-to guide of writing custom drivers are available alongside documentation thereof as part of the public code repository.

The parallel active learning run with hardware-in-the loop of HELAO demonstrates for the first time that two (and technically unlimited more) spatially separated instruments in a materials science laboratory are capable of collaboratively optimizing together for faster discovery. Contributions and collaborations with and by the community to expand the hardware support for HELAO is therefore warmly welcome. Future efforts will aim to bridge HELAO with methods from theoretical materials' science to build modular physics-informed instrumentation and autonomous feedback loops connecting laboratories.

4. Experimental Section

Design Guidelines and Protocols: From the bottom-up hardware perspective, a research instrument is an assembly of devices. A device is a piece of laboratory equipment, defined as the largest "thing" that has a dedicated communication stream, that is, a multi-channel potentiostat, or a motor control board.

Devices are typically shipped with a driver that enables access to some or all its functions, that is, measuring a current. From the top-down perspective, a user or operator wishes to perform a series of experiments, which are each a list of actionable events defined as "actions" in HELAO. The actions are to be executed in a particular order with predefined or variable parameters and/or designed on-the-fly via a decision policy. In this latter case, evaluation of the decision policy can be viewed as a particular type of action whose execution impacts future actions. The instructions for an experimental campaign are given to an orchestrator, which governs their sequential execution from a queue. Each action is materialized by the drivers, thereby completing the top-to-bottom instrument framework. Everything that happens to or on that instrument originates within the orchestrator.

In order for the ensemble of devices to operate in concert as a single instrument, it is convenient to assemble the various elements listed above into a single software framework. Hierarchically from bottom to top, each device driver (internally communicating through, for example, serial, TCP/IP commands, or a dynamic link library) is exposed through

a uvicorn[28] web server through fastAPI. Construction of actionable functions ("actions") are constructed from the API calls exposed by the drivers, where each action may involve multiple API calls.

An example of an action to pump a mixture of three fluids would therefore be "initiated" by the orchestrator calling the respective action. This action then calls the pump driver server. Internally the driver server is sequentially called by the mixture action as the hardware requires us to first initialize each pump channel, prime the pumps, and only then turn the pumps on. The orchestrator will receive a nested reply from the action that entails all information exchanged, down to the lowest level, that is, initialization, priming, execution.

A rigorous commitment to data management is foundational to this framework's implementation. Requests to the driver and action servers track all functions called, as well as all (echoed) input parameters and outputs of those functions. The orchestrator tracks additional metadata, such as the time at which an action was performed or the point on a substrate at which an experiment was conducted, in addition to accepting arbitrary custom metadata. All of these are then automatically saved (redundantly), in the native file format (if applicable), and in an hdf5 file together with the parameters and metadata. Methods for depositing the hdf5 file into institutional repositories like KaDI4Mat[29] or MEAD[19] repositories are automatically executed after each session. From KaDI4Mat,[29] experimental data can be accessed internally but also be shared with the community through materials cloud or to inform simulations through AIIDA.[30,31]

Due to each element of the authors' software framework being a server, a very high degree of modularity is achieved. This allows, for instance, a single instance of a device to fulfill requests from multiple action servers sequentially. This type of resource sharing requires the actions to be programmed to ingest calls from multiple orchestrators, for example by notifying the orchestrators of the un/availability of each action. A more straightforward implementation of resource sharing is for a single orchestrator to govern multiple instruments with a shared piece of equipment, which they demonstrate herein. More generally, this design allows for distributed hosting of devices on different machines, potentially dispersed around the globe.

Drivers and Driver Servers: Any (autonomous) experimental workflow consists of smaller organizational units, that is, a SDC[32] instrument consists of several hardware devices such as motors, a force sensor, pumps, and a potentiostat. In addition, there are software devices such as data analysis and experimental design. All of these devices need to be able to receive commands, perform the instruction, and reply with measurement data or a status and echo back the input parameters. With respect to the orchestrator, interaction with an analysis or active learning module is equivalent to that of a hardware device, motivating virtual and physical devices to be implemented in the same manner.

Driver Server Design: Drivers provide the lowest-level interaction with devices based on the elementary communication commands for the respective device, for example, connect, disconnect, query the device status, or read data. Some drivers are therefore more complex than others, as some devices offer direct python APIs whilst others require development of python wrappers or source code. Positively notable examples are for instance python drivers offered by Mecademic or Palmsens offering well-documented software development kits (SDKs). Each device is paired to a dedicated driver server. Calling driver functions can only be done through the web-based API by sending web requests, enabling software modularity that mimics hardware modularity wherein devices can be reconfigured into new instruments.

Driver Parameters: Drivers accept parameters, which are validated through pydantic data types that inherit from the pydantic's BaseModel. This automatically annotated and type-hinted validation scheme allows users to assess how a request should be formatted in order to receive a desired device behavior. Additionally, the pedantic validation scheme ensures proper data handling, easing data management downstream. Traceability and ease in debugging are ensured by each driver server echoing all provided input parameters alongside the output data. For this purpose, the return object from any server including drivers is a python dictionary (an unsorted data type containing key value pairs) containing two keys for the input parameters and output data. The parameter key is described by its name, the value(s), and optionally a physical unit. The data key contains data, which contains the data acquired or derived from the device. These python dictionaries play a signal role between different organizational units, for example calling the pump requires specification of volume, speed, and direction. The response from the pump (a device acting but not measuring) is the entire serial string communication response (potentially containing valuable error messages) from the pump as output. The units returned for pumping are for instance speed in microliters per minute, total volume in microliters, and a binary flag for forward versus backward pumping direction.

Actions and Action Servers Design: Hierarchically above drivers, actions wrap one or many driver functions such that the action function has a name and parameters that are meaningful for the deployment of the device(s) in a particular type of experiment. This provides an abstraction layer where two action functions can be programmed in different labs using different devices/drivers, enabling shared higher-level code that calls the action functions. Similar to driver servers, actions also expose their functionalities as servers and again are not limited to a single instrument. Communication with multiple device drivers is intended for when knowing what multiple devices are needed to realize a single physical action, such as motor actuation with feedback from a force sensor. To manage shared driver/device-level resources, direct communication between drivers is forbidden, requiring any such message passing to occur via the action server.

Action Parameters: Similar to driver servers, the return statement of an action server is a python dictionary containing the parameters and data. The output from an action can be customized for the specific use case, but is generally the aggregate return statements received from all driver server calls downstream. After execution of the relevant action function, the return statements of the called actions will be received by the orchestrator as the highest level of this hierarchy.

A major advantage of driver/action distinction is the possibility of multiple operating computers sharing one device. Any failures on a higher level (i.e., computer crash and/or program failure of the deployed visualizer or orchestrator) do not affect the operation of an instrument.

This design also facilitates the resolution of hardware conflicts and smart instrument communication, since some simultaneously-executing actions could logically cause a contradiction. Therefore, a driver blocks further execution until the current request has been fulfilled. For instance, when the force sensor is measuring the amount of applied force as an action, this action server will block execution of subsequent actions until the current action is finished, which is implanted by awaiting (an asynchronous function call) the response from the force sensor driver. After the awaited response is received, the next action will be called. This locked execution of sequential instructions allows for a safe operation without the need of a state machine. An alternative implementation would require a state machine on the highest level, thus violating the design principle that dictates little to no changes upon addition of new hardware.

Orchestrator/Local and External Database: The highest level in the framework is the orchestrator, which sends out instructions to actions from a list of experiments to be performed, where the orchestrator also holds the sequence of actions and the respective parameters needed to perform each experiment. The orchestrator server accepts experiments through an API function called addExperiment, which adds an experiment to a list that is executed in the first-in-first-out order. Upon exhaustion of experiments from the process, the orchestrator remains online and awaits the next experiment(s). In total, initializing a HELAO session involves launching $n + m + 1$ servers, where n is the number of devices in the system and m is the number of action servers. In the standard configuration, each driver is controlled by its own action server ($m = n$). Alternatively, an action server may govern multiple devices or a driver may be directly incorporated into an action server ($m < n$). Based on an instrument-specific configuration file, a launch script governs the initialization of each server. If two copies of an instrument exist, they require unique configuration files with unique IP addresses, such that deploying HELAO for a cloned instrument can be achieved by updating the IP addresses in a copy of the configuration file.

Defining a Process: The main purpose of the orchestrator is the execution of a list of (dynamically editable) experiments from the process as well as data management. For defining a sequential experiment involving multiple devices, experiments need to be specified by a sequence of what actions are to be executed in a particular order with all necessary parameters. An instrument is factually defined by the devices called in a sequence of events. An experiment is defined by a python dictionary containing two dictionary keys: the Sequence of Events (SOE) key, which contains an ordered list outlining the exact order in which actions are to be executed and the "params" key containing all necessary parameters for any of the actions outlined in the SOE. As actions may be called multiple times they are numbered sequentially in the SOE. Calling an action through the orchestrator requires four parts: the name of the action, the desired function, a number which indicates the *n*-th time that we call that specific action within a SOE and the thread number (e.g., "motor/moveAbs-3:1" for calling the third absolute movement of a motor belonging to thread 1). Thread numbers are important for deploying active learning across multiple instruments. Two or more instruments/threads run independent of each other until some action requires input from all threads or a SOE is finished.

The parameters for actions are stored under the "MoveAbs-3" key in the parameter dictionary and contains the specific values for running that particular action (i.e., dx = 2 mm, dy = 3 mm, dz = 0 mm).

Defining a Session: To start and end a session lasting for one or more experiments the first and last actions to be called are the "start" and "finish" actions, natively implemented within the orchestrator (i.e., not as servers), hence being called native actions. The data acquired within a session is locally stored in a single hdf5 file that is then uploaded to KaDI4Mat upon calling the "finish" action. Storing data locally and uploading it at the end of a session has been shown to be significantly faster and avoids reliance on the speed or availability of the master database, which may not be directly controlled by the lab running the experiments. An illustrative example for this design choice is when we measured a series of Raman spectra and performed the upload after taking each spectrum. Whilst each spectrum only took a second to measure, the upload time was comparable, forcing instrument down time that was remedied by asynchronous data uploading.

Data Analysis and Machine Learning Servers: A goal of HELAO is to enable active learning accelerated experiments across a wide range of laboratory instruments. Active learning does however require automatic data analysis and machine learning based suggestion of the next best subsequent experiment.

These two functionalities are implemented as servers in HELAO. On a high level, active learning within the HELAO framework is simply the alteration of parameters of an action by some suggestion of an algorithm. The parameter to be changed in a subsequent experiment is referred to as the "target". The algorithm needs to have access to all (analyzed) data to suggest the target. This "source" data needs to be well posed for the machine learning algorithm, which typically requires analysis of the raw data. The automated data analysis in HELAO is again a server-action. A unique aspect of an analysis server is its required access to raw data, which is implemented by using pointers to the location in the orchestrator memory of where relevant data (the source data) is stored. Likewise, the server dedicated to machine learning for active learning needs to be pointed to the input and output values of the analyzed data. Inside the active learning action, the datasets are aggregated on-the-fly from the orchestrator temporary storage (what is later the hdf5 file being uploaded). A target can be specified from a list of candidates or be freely decided by the algorithm, depending on the chosen optimizer, and upon receiving the target the orchestrator updates and runs the pending measurement action.

These functionalities allow for autonomous operation where the user only has to define the budget of active learning runs, pointers to the input and output values, and the choice of optimizer and the estimator.

The active learning server can be equipped with a broad range of optimizers and regression algorithms. Also, several acquisition functions have been implemented including expected improvement (EI) and probability of improvement (POI). We envision the possibility for incorporating different fidelity sources by adaption of optimizers that can handle different fidelities, which is an active area of machine learning research where advancements can be readily incorporated into HELAO.

As some ML algorithms require significant computational resources within a thread and some actions are data-transfer intensive, servers may become unresponsive. To solve this issue, the most computationally expensive tasks like machine learning can be wrapped inside a celery[33] server. Celery is a server-based framework capable of distributing high workloads across compute clusters. We empirically observed this necessity for long running active learning runs with a high degree of freedom.

Visualizer: On the same hierarchical level of the orchestrator is the visualizer, which can be viewed as a "read only" orchestrator that has global access and does not store data. This server can display the live data of, for example, electrochemical test measurements or Raman spectroscopy to assess data quality during a run.

Supporting Information

Supporting Information is available from the Wiley Online Library or from the author.

Acknowledgements

This work contributes to the research performed at CELEST (Center for Electrochemical Energy Storage Ulm-Karlsruhe) and was partly funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany´s Excellence Strategy – EXC 2154 – Project number 390874152. This project has also received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 957189. Design of software architecture at Caltech was supported by the Liquid Sunlight Alliance, which is supported by the U.S. Department of Energy (DOE), Office of Science, Office of Basic Energy Sciences (BES), Fuels from Sunlight Hub under Award Number DE-SC0021266. The authors would like to thank Ephraim Schoof for developing the KaDI4Mat API and helping in the initial interfacing with data management. They would like to thank KIT IAM for hosting KaDI4Mat.

Open access funding enabled and organized by Projekt DEAL.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

H.S.S. and J.M.G conceived the idea and designed the first software layout. H.S.S. developed the first drivers and server-based communication protocols. F.R., J.F., D.G., M.R., and P.D. implemented drivers, wrote actions, and conducted the experiments. F.R. implemented drivers pertaining to SDC and deployed machine learning algorithms to HELAO. J.F. integrated these contributions into the orchestrator. All authors reviewed the manuscript.

Data Availability Statement

The data that support the findings of this study are openly available in [figshare] at [https://doi.org/10.6084/m9.figshare.16798177.v1], reference number [16798177].

Keywords

laboratory automation, materials acceleration, high-throughput experimentation, data management

Received: October 12, 2021
Revised: November 21, 2021
Published online:

[1] K. Alberi, M. B. Nardelli, A. Zakutayev, L. Mitas, S. Curtarolo, A. Jain, M. Fornari, N. Marzari, I. Takeuchi, M. L. Green, M. Kanatzidis, M. F. Toney, S. Butenko, B. Meredig, S. Lany, U. Kattner, A. Davydov, E. S. Toberer, V. Stevanovic, A. Walsh, N.-G. Park, A. Aspuru-Guzik, D. P. Tabor, J. Nelson, J. Murphy, A. Setlur, J. Gregoire, H. Li, R. Xiao, A. Ludwig, et al., *J. Phys. Appl. Phys.* **2019**, *52*, 013001.

[2] J.-P. Correa-Baena, K. Hippalgaonkar, J. van Duren, S. Jaffer, V. R. Chandrasekhar, V. Stevanovic, C. Wadia, S. Guha, T. Buonassisi, *Joule* **2018**, *2*, 1410.

[3] M. L. Green, C. L. Choi, J. R. Hattrick-Simpers, A. M. Joshi, I. Takeuchi, S. C. Barron, E. Campo, T. Chiang, S. Empedocles, J. M. Gregoire, A. G. Kusne, J. Martin, A. Mehta, K. Persson, Z. Trautt, J. Van Duren, A. Zakutayev, *Appl. Phys. Rev.* **2017**, *4*, 011105.

[4] E. J. Amis, X. D. Xiang, J. C. Zhao, *MRS Bull.* **2002**, *27*, 295.

[5] Materials Acceleration Platform—Accelerating Advanced Energy Materials Discovery by Integrating High-Throughput Methods with Artificial Intelligence. *109*, http://mission-innovation.net/wp-content/uploads/2018/01/Mission-Innovation-IC6-Report-Materials-Acceleration-Platform-Jan-2018.pdf.

[6] M. Aykol, J. S. Hummelshøj, A. Anapolsky, K. Aoyagi, M. Z. Bazant, T. Bligaard, R. D. Braatz, S. Broderick, D. Cogswell, J. Dagdelen, W. Drisdell, E. Garcia, K. Garikipati, V. Gavini, W. E. Gent, L. Giordano, C. P. Gomes, R. Gomez-Bombarelli, C. B. Gopal, J. M. Gregoire, J. C. Grossman, P. Herring, L. Hung, T. F. Jaramillo, L. King, H.-K. Kwon, R. Maekawa, A. M. Minor, J. H. Montoya, T. Mueller, et al., *Matter* **2019**, *1*, 1433.

[7] H. S. Stein, J. M. Gregoire, *Chem. Sci.* **2019**, *10*, 9640.

[8] K. F. Jensen, C. W. Coley, N. S. Eyke, *Angew. Chem., Int. Ed.* **2020**, *59*, 22858.

[9] C. W. Coley, N. S. Eyke, K. F. Jensen, *Angew. Chem., Int. Ed.* **2020**, *59*, 23414.

[10] T. Dimitrov, C. Kreisbeck, J. S. Becker, A. Aspuru-Guzik, S. K. Saikin, *ACS Appl. Mater. Interfaces* **2019**, *11*, 24825.

[11] P. Nikolaev, D. Hooper, F. Webber, R. Rao, K. Decker, M. Krein, J. Poleski, R. Barto, B. Maruyama, *npj Comput. Mater.* **2016**, *2*, 16031.

[12] L. M. Roch, F. Häse, C. Kreisbeck, T. Tamayo-Mendoza, L. P. E. Yunker, J. E. Hein, A. Aspuru-Guzik, *Sci. Rob.* **2018**, *3*, eaat5559.

[13] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, A. Ng, *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Workshop on Open Source Robotics, Kobe, Japan, May **2009**.

[14] I. M. Pendleton, G. Cattabriga, Z. Li, M. A. Najeeb, S. A. Friedler, A. J. Norquist, E. M. Chan, J. Schrier, *MRS Commun.* **2019**, *9*, 846.

[15] P. S. Gromski, J. M. Granda, L. Cronin, *Trends Chem.* **2020**, *2*, 4.

[16] P. S. Gromski, A. B. Henson, J. M. Granda, L. Cronin, *Nat. Rev. Chem.* **2019**, *1*, 119.

[17] I. E. Castelli, D. J. Arismendi-Arrieta, A. Bhowmik, I. Cekic-Laskovic, S. Clark, R. Dominko, E. Flores, J. Flowers, K. U. Frederiksen, J. Friis, A. Grimaud, K. V. Hansen, L. J. Hardwick, K. Hermansson, L. Königer, H. Lauritzen, F. L. Cras, H. Li, S. Lyonnard, H. Lorrmann, N. Marzari, L. Niedzicki, G. Pizzi, F. Rahmanian, H. Stein, M. Uhrin, W. Wenzel, M. Winter, C. Wölke, T. Vegge, *arXiv:2106.01616 [cond-mat.mtrl-sci]* **2021**.

[18] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne, J. Bouwman, A. J. Brookes, T. Clark, M. Crosas, I. Dillo, O. Dumon, S. Edmunds, C. T. Evelo, R. Finkers, A. Gonzalez-Beltran, A. J. G. Gray, P. Groth, C. Goble, J. S. Grethe, J. Heringa, P. A. C. 't Hoen, R. Hooft, T. Kuhn, R. Kok, J. Kok, et al., *Sci. Data* **2016**, *3*, 160018.

[19] E. Soedarmadji, H. S. Stein, S. K. Suram, D. Guevarra, J. M. Gregoire, *npj Comput. Mater.* **2019**, *5*, 79.

[20] FastAPI, https://fastapi.tiangolo.com/#license (accessed: June 2021).

[21] Home Page, https://opcfoundation.org/ (accessed: June 2021).

[22] S. E. Ament, H. S. Stein, D. Guevarra, L. Zhou, J. A. Haber, D. A. Boyd, M. Umehara, J. M. Gregoire, C. P. Gomes, *npj Comput. Mater.* **2019**, *5*, 77.

[23] B. Rohr, H. S. Stein, D. Guevarra, Y. Wang, J. A. Haber, M. Aykol, S. K. Suram, J. M. Gregoire, *Chem. Sci.* **2020**, *11*, 2696.

[24] F. Häse, M. Aldeghi, R. J. Hickman, L. M. Roch, M. Christensen, E. Liles, J. E. Hein, A. Aspuru-Guzik, *arXiv:2010.04153 [stat.ML]* **2020**.

[25] HDF5 Files, https://figshare.com/s/1578223bbf5ddde605af

[26] D. Allan, T. Caswell, S. Campbell, M. Rakitin, *Synchrotron Radiat. News* **2019**, *32*, 19.

[27] M. Aldeghi, F. Häse, R. J. Hickman, I. Tamblyn, A. Aspuru-Guzik, *arXiv:2103.03716 [math.OC]* **2021**.

[28] The Uvicorn Project, https://www.uvicorn.org/

[29] N. Brandt, L. Griem, C. Herrmann, E. Schoof, G. Tosato, Y. Zhao, P. Zschumme, M. Selzer, *Data Sci. J.* **2021**, *20*, 8.

[30] L. Talirz, S. Kumbhar, E. Passaro, A. V. Yakutovich, V. Granata, F. Gargiulo, M. Borelli, M. Uhrin, S. P. Huber, S. Zoupanos, C. S. Adorf, C. W. Andersen, O. Schütt, C. A. Pignedoli, D. Passerone, J. VandeVondele, T. C. Schulthess, B. Smit, G. Pizzi, N. M. Marzari, *Sci. Data* **2020**, *7*, 299.

[31] G. Pizzi, A. Cepellotti, R. Sabatini, N. Marzari, B. Kozinsky, *Comput. Mater. Sci.* **2016**, *111*, 218.

[32] The Potential of Scanning Electrochemical Probe Microscopy and Scanning Droplet Cells In Battery Research - Daboss - Electrochemical Science Advances - Wiley Online Library https://chemistry-europe.onlinelibrary.wiley.com/doi/full/10.1002/elsa.202100122 (accessed: October 2021).

[33] The Celery Project, https://celeryproject.org/ (accessed: October 2021).