



Building an Open Representation for Biological Protocols

BRYAN BARTLEY, JACOB BEAL, and MILES ROGERS, Raytheon BBN Technologies, USA

DANIEL BRYCE and ROBERT P. GOLDMAN, SIFT, LLC, USA

BENJAMIN KELLER, University of Washington, USA

PETER LEE, Ginkgo Bioworks, USA

VANESSA BIGGERS and JOSHUA NOWAK, Strateos, Inc., USA

MARK WESTON, Netrias, Inc., USA

28

Laboratory protocols are critical to biological research and development, yet difficult to communicate and reproduce across projects, investigators, and organizations. While many attempts have been made to address this challenge, there is currently no available protocol representation that is unambiguous enough for precise interpretation and automation, yet simultaneously “human friendly” and abstract enough to enable reuse and adaptation. The Laboratory Open Protocol language (LabOP) is a free and open protocol representation aiming to address this gap, building on a foundation of UML, Autoprotocol, Aquarium, SBOL RDF, and the Provenance Ontology. LabOP provides a linked-data representation both for protocols and for records of their execution and the resulting data, as well as a framework for exporting from LabOP for execution by either humans or laboratory automation. LabOP is currently implemented in the form of an RDF knowledge representation, specification document, and Python library, and supports execution as manual “paper protocols,” by Autoprotocol or by Opendrivers. From this initial implementation, LabOP is being further developed as an open community effort.

CCS Concepts: • **Applied computing** → **Life and medical sciences**; • **Computing methodologies** → **Knowledge representation and reasoning**;

Additional Key Words and Phrases: Protocol, biology, representation, UML, RDF, SBOL

ACM Reference format:

Bryan Bartley, Jacob Beal, Miles Rogers, Daniel Bryce, Robert P. Goldman, Benjamin Keller, Peter Lee, Vanessa Biggers, Joshua Nowak, and Mark Weston. 2023. Building an Open Representation for Biological Protocols. *ACM J. Emerg. Technol. Comput. Syst.* 19, 3, Article 28 (June 2023), 21 pages.

<https://doi.org/10.1145/3604568>

This work was supported by Air Force Research Laboratory (AFRL) and DARPA contracts FA8750-17-C-0184, FA8750-17-C-0231, and HR001117C0095.

Authors' addresses: B. Bartley, J. Beal, and M. Rogers, Raytheon BBN Technologies, 10 Moulton Street, Cambridge, MA 02138; emails: bryan.a.bartley@raytheon.com, jakebeal@ieee.org, miles.rogers@raytheon.com; D. Bryce and R. P. Goldman, SIFT, LLC, 319 N. 1st Avenue, Suite 400, Minneapolis, MN 55401; emails: dbryce@sift.net, rpgoldman@sift.net; B. Keller, University of Washington, 185 Stevens Way, Seattle, WA 98195; email: bjkeller@uw.edu; P. Lee, Ginkgo Bioworks, 27 Drydock Avenue, 8th Floor, Boston, MA 02210; email: plee@ginkgobioworks.com; V. Biggers and J. Nowak, Strateos, Inc., 3565 Haven Ave Suite 3, Menlo Park, CA 94025; emails: vanessa.biggers@strateos.com, josh.nowak@strateos.com; M. Weston, Netrias, Inc., One Broadway, 14th Floor, Cambridge, MA 02142; email: weston@netrias.com.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.

1550-4832/2023/06-ART28 \$15.00

<https://doi.org/10.1145/3604568>

1 INTRODUCTION

Laboratory protocols are critical to biological research and development. However, protocols are often difficult to communicate or reproduce, given the differences in context, skills, instruments, and other resources between different projects, investigators, and organizations. One of the preconditions for addressing these challenges is for there to be a commonly used data representation for describing laboratory protocols that is unambiguous enough for precise interpretation and automation, yet simultaneously “human friendly” and abstract enough to support reuse and adaptation.

While there has been much prior work on representations for protocols, prior approaches have generally been limited either by their dependence on natural language, which may be ambiguous or lacking crucial detail, or the expressiveness of their representation, which may be overly specific to an organization’s execution environment. Currently, many protocol representations focus on simplifying the capture and distribution of descriptions in natural language. Such representations include protocols.io [34] and the many commercial electronic laboratory notebook products. A similar approach is used for recording protocol execution information with community-defined minimum information standards such as MIAME [8], MIFlowCyt [20], and STRENDA [35]. Much of the information in such approaches is encoded using natural language, a “human friendly” option that is much easier to elicit from experimentalists, at the expense of machine interpretability. Consequently, protocols and protocol execution records captured with such representations cannot be automatically validated and are often ambiguous, incorrect, or lack key information. For the same reasons, such protocols cannot generally be executed by laboratory automation systems.

Other protocol representations have focused on automation-assisted execution. In many cases, these are highly bespoke solutions catering to particular hardware, which in turn is often proprietary and vendor-specific. Some representations have been made applicable to a broader set of automation systems, however, such as Autoprotocol [23] and Antha [31], or instead use laboratory technicians as their automation, as in the case of Aquarium [38]. All of these representations have been generally “low level” in their description of protocols, focusing on very specific details of each operation. This specificity, which, on the one hand, enables automated execution, on the other hand, imposes barriers to adoption and to generalization and reuse, since this level of detail often obscures understanding and is too tied to the specifics of a particular laboratory to readily transfer into different environments. Such representations are typically also difficult to translate into more “human-friendly” forms.

Finally, there are a number of workflow languages that solve similar problems in business logic or information processing, such as **Unified Modeling Language (UML)** [26], the Common Workflow Language [1], Taverna [39], and Cromwell [9], not to mention biology-specific workflow systems such as Toil [37] and Galaxy [16]. The execution models of such systems are in some cases general enough to be applied to the description and execution of laboratory protocols, but to the best of our knowledge no such application has been implemented. Further, the very generality of these approaches can make it more difficult for a biological domain expert to comprehend how to apply them to protocols, given the large gap between abstract task execution concepts and the specifics of particular tasks that must be performed in a laboratory.

Critically, none of this prior work simultaneously supports all three of these key requirements: (1) translation into “human-friendly” natural language descriptions of protocols, (2) transferring protocols across heterogeneous automation platforms, and (3) generality of workflows. These requirements became critically apparent to the authors during the DARPA **Synergistic Discovery and Design (SD2)** program, which aimed to develop data-driven discovery and design for laboratory-based disciplines such as synthetic biology. With more than 100 researchers across more than 20 organizations, several of which ran experimental facilities with different forms of

high-throughput automation, participants were forced to confront challenges around protocol sharing, experiment reproducibility, automation heterogeneity, and communication between domain experts and automation experts. The lessons learned in tackling these challenges led directly to identification of the three key requirements above and the development of the protocol representation presented here.

We thus present a unified approach to protocol representation that bridges prior approaches through the development of the **Laboratory Open Protocol language (LabOP)**, a free and open protocol representation building on a foundation of UML [26], Autoprotocol [23], Aquarium [38], the **Synthetic Biology Open Language (SBOL)** [2, 22], and the **Provenance Ontology (PROV-O)** [24]. In Section 2, we introduce the high-level requirements for a common protocol language that were established in stakeholder discussions. In Section 3, we introduce the technological foundations of LabOP as a linked-data standard, followed by a higher-level overview of how protocols are represented. Section 4 discusses the current prototype software implementations of LabOP, and Section 5 provides examples of how LabOP supports each of the design requirements. Finally, Section 6 discusses ongoing development by the LabOP community and plans for future development.

2 DESIGN REQUIREMENTS

Information about laboratory protocols is used for a wide range of purposes in research and development at many different stages of experiment design, execution, data analysis, interpretation, and communication and sharing with other groups. Through our bioprotocols working group discussions, including representatives from industrial labs, academic labs, and biofoundries, we identified the following set of design requirements necessary for a broadly applicable representation of biological protocols:

- **Execution by either humans or machines:** When available, laboratory automation can greatly improve the productivity of researchers, so protocols should be specified in sufficient detail to enable them to be mapped for machine execution. Many laboratories, however, do not have automation available. Moreover, even when some degree of automation is available, it is common for protocols to incorporate both automated and manual stages, so protocols also need to be able to be presented in a succinct and human-friendly form.
- **Mapping protocols from one laboratory environment to another:** Protocol replication and reuse requires the ability to map a protocol from one laboratory to another, despite their differences in equipment, inventory, and information systems.
- **Representation of diverse workflow patterns:** A protocol representation must be general enough to span the wide variety of activities and workflow patterns used in protocols. It must be possible to extend it to activities and workflow patterns not considered when the representation was designed.
- **Maintaining execution records and associated metadata markup:** When a protocol is executed, it is important to be able to record the specific time of execution, the laboratory and personnel that executed the protocol, equipment used, and so on. Information in the protocol can be used to automatically tag measurements with metadata, such as the entities being measured, their roles in the experimental design, units of measurement, and so on.
- **Recording modifications of protocols and the relationship between different versions:** Protocols are likely to be the subject of ongoing improvement and maintenance. For example, a protocol may be modified to make the protocol simpler to execute or more reliable, enable it to be executed at a lab with different equipment, to scale it up or down, and so on.

- **Verification and validation of protocol completeness and coherence:** Authoring a protocol requires substantial care and effort, and the usefulness of the protocol is compromised if its specification is ill-formed, erroneous, or incomplete (e.g., the classic “inadequate methods section” issue in scientific publications).
- **Planning, scheduling, and allocation of laboratory resources:** Laboratory resources are valuable, and some organizations will want to be able to optimize their use. To do so, a protocol representation should support (at least) extraction of resource requirements and estimated durations from activities in the protocol.

Note that the first three requirements correspond to the key requirements of “human-friendliness,” transfer between heterogeneous environments, and generality, while the latter four requirements are more generic automation capabilities. In Section 5 below, we provide concrete examples that demonstrate how LabOP meets each of these requirements.

3 REPRESENTATIONAL FOUNDATIONS

To make LabOP effective as a broadly shared community standard, we have adopted a principle of building upon existing standards to maximize compatibility and interoperability, taking advantage of existing tooling, and making the implementation as lightweight as possible. The LabOP data model thus unites representational elements that have proven useful in other protocol efforts by our stakeholder community.

Specifically, LabOP is defined as an ontology for storing linked data and draws from representational elements found in UML [26], Autoprotocol [23], Aquarium [38], the **Synthetic Biology Open Language (SBOL)** [2, 22], and the **Provenance Ontology (PROV-O)** [24]. In this respect, the key virtue of LabOP is not so much any specific novelty, but rather its unifying approach, which integrates these elements to satisfy all of the requirements in Section 2.

In this section, we provide a high-level, conceptual overview of these representational elements. We do not attempt a detailed description of the LabOP data model here; for this, we refer readers to the current specification document [7] or to direct inspection of the data model using an ontology editor such as Protégé [25].

3.1 Ontology Encoding Provides a Common Linked Data Format

LabOP is fundamentally an ontology encoded in the **Web Ontology Language (OWL)** [6] and is based in Semantic Web practices and resources. This means that language uses **Uniform Resource Identifiers (URIs)** to unambiguously identify and define protocol elements, such as activities, samples, datasets, and so on, and links these objects together into a graph data structure. It is, to our knowledge, the first protocol language based on a linked-data representation. This implementation choice has multiple advantages for supporting the requirements described above.

First, as an ontology, LabOP supports machine reasoning about conceptual abstractions (i.e., classes, subclasses, superclasses, etc.), explicit instances of classes, and their relationships. The LabOP ontology thus defines a data model or “language” that organizes protocol-related concepts. This enables it to specify protocol details (e.g., labware) at different levels of abstraction, while using ontological inference to translate protocols into different laboratory environments, as highlighted later in Section 5. LabOP is also interoperable with all of the many other controlled vocabularies and bio-ontologies with the same linked-data interface, including PubChem [18], ChEBI [12], UniProt [11], and the widely used ontologies in the OBO Foundry [17] such as the Sequence Ontology [14] and the NCI Thesaurus [32]. These are used to annotate LabOP protocols and their execution records with standardized metadata that do not depend on vagaries of natural language and human interpretation (e.g., unambiguously identifying a reagent via a PubChem substance identifier).

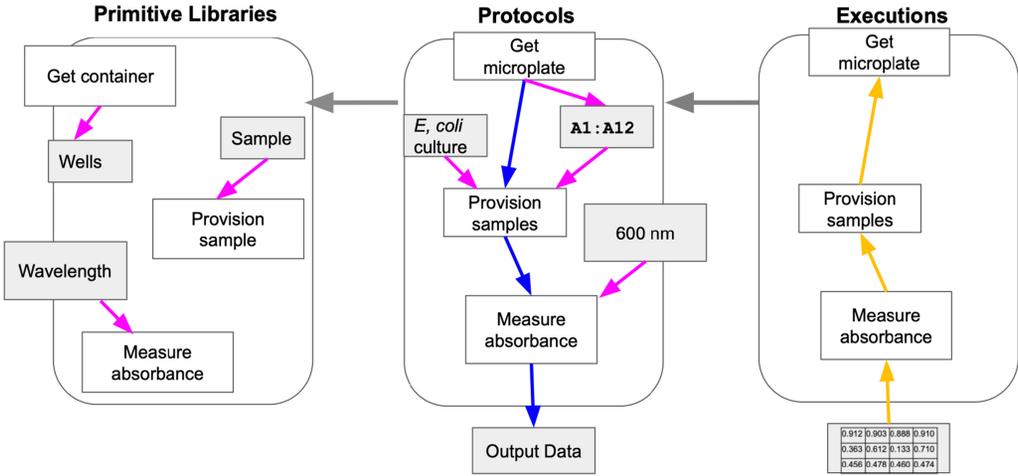


Fig. 1. The LabOP language specifies three layers of abstraction: operation libraries, protocols, and executions. Blue arrows denote the control flow of activities; magenta arrows, flows of input and output tokens; and yellow arrows, an execution trace.

Furthermore, because LabOP is encoded as linked data, the history and contextual details of a protocol execution can be reconstructed from recorded metadata. For example, given a protocol that collects flow cytometry data on samples of different microbial strains under varying growth conditions, the protocol specification supports automated linking of each FCS file produced by the flow cytometer with contextual information about the strain, growth conditions, time of data collection, calibration, and so on, of the sample from which the FCS file was produced.

3.2 UML Provides Flexible Semantics for Different Workflow Models

Because the core of a protocol is a workflow of activities to be carried out, a key design choice in LabOP was to adopt an established standard for workflow modeling that could provide a well-defined and general formal semantics, yet also be sufficiently abstract as to allow succinct expression and adaptation. We found such a model in **Unified Modeling Language (UML)** behavior representations (specifically, the current version 2.5.1 [26]), which forms the central abstraction layer for LabOP (“Protocols” in Figure 1). UML behaviors provide a general, domain-independent workflow model. We have translated this portion of the UML specification into OWL, which is, to the best of our knowledge, the first such encoding that has been developed.

UML’s execution model is universally expressive, including support for serial, parallel, non-deterministic, and distributed execution. The execution semantics are based on a notion of token flow similar to those of *Petri nets* [28]. A *Petri net* is a graph in which nodes have input edges and output edges. Roughly speaking, a Petri net node is enabled to “fire,” sending a token out all of its output edges when all of its input edges are filled with tokens; at the same time, those input tokens are removed. The control flow semantics can support ordered steps as well as steps in parallel or in arbitrary order. Execution patterns may also include loops and recursions. Furthermore, since tokens can potentially be communicated between different agents, the model also allows for execution to be distributed, e.g., between several pieces of automation equipment, as a mixture of human and automated execution, or even across a group of collaborating laboratories.

Another advantage of UML is that it was developed for use as a diagrammatic language as well as a data model. Thus, it also provides a set of graphical flow control and abstraction constructs for succinct and human-friendly communication about complex workflows. At the same time, the

LabOP Library/Operation	Autoprotocol Equivalent
Library: sample_arrays EmptyContainer PlateCoordinates Rows Columns DuplicateCollection ReplicateCollection	Undocumented “SUPPLY NEW CONTAINER” operation <i>n/a - Autoprotocol references only single locations</i> <i>n/a - Autoprotocol references only single locations</i> <i>n/a - Autoprotocol references only single locations</i> <i>n/a: operation makes an array with the same shape as an input</i> <i>n/a: like DuplicateCollection, but also adds a new replicate dimension</i>
Library: plate_handling Cover Incubate Seal AdhesiveSeal ThermalSeal Spin Uncover Unseal	cover incubate seal, flexible mode seal, mode=thermal seal, mode=adhesive spin uncover unseal
Library: liquid_handling Provision Dispense Transfer TransferInto PipetteMix	provision liquid_handle, mode=dispense liquid_handle up in one location, down elsewhere liquid_handle with a non-empty destination liquid_handle up and down repeatedly in the same location
Library: spectrophotometry MeasureAbsorbance MeasureFluorescence	spectrophotometry, mode=absorbance spectrophotometry, mode=fluorescence
<i>Currently unimplemented</i>	acoustic_transfer, flow_cytometry, measure_mass, measure_volume, spectrophotometry mode=luminescence/shake

Fig. 2. LabOP’s laboratory operations are organized into libraries based on required equipment types. The initial collection of libraries “bundled” into the LabOP prototype are based on Autoprotocol, currently implementing most functionality from that language, plus operations for defining and manipulating collections of samples.

formal execution model provides an unambiguous semantics for verification, validation, and other forms of machine reasoning.

3.3 Autoprotocol Provides Definitions of Laboratory Operations

As UML is abstract and domain-independent, it does not provide any definitions for laboratory operations. For these, LabOP begins by drawing definitions from Autoprotocol [23]. Autoprotocol describes biological protocols in terms of a sequence of instructions, and while this linear workflow is not expressive enough (hence UML), the instructions themselves provide useful examples of laboratory operations that can be readily mapped from one laboratory environment to another.

These instructions, such as `liquid_handle`, `incubate`, `provision`, and `spin`, have been designed and refined by the authors of Autoprotocol to be a basis set for expressing the activities of common biological protocols in a manner readily transported between different pieces of laboratory automation. The Autoprotocol instructions thus provide a reasonable starting point for building out LabOP’s collection of libraries of laboratory-independent operations.

Figure 2 shows how the current implementation of LabOP organizes operations adapted from Autoprotocol into four libraries. Some activities commonly expressed using Autoprotocol operations are at a lower level than desirable for a human experimenter, however, such as specifying pipette mixing as a sequence of repetitive liquid handling operations. Not only are such activities not human-friendly, supplying such fine details makes protocols harder to translate from one lab

to another. In cases such as these, sets of Autoprotocol instructions are replaced by more abstract alternatives that capture an Autoprotocol use pattern.

3.4 Array-based Sample Operations Are Inspired by Aquarium

To support automated, high-throughput experimentation, LabOP's execution model handles operations on samples in a batched fashion in a manner inspired by Aquarium [38]. In contrast, in Autoprotocol it is only possible to address one location at a time, i.e., a single container or a single compartment within a container, such as a well on a plate. That limitation means that in Autoprotocol any operation on multiple locations must name every location as a fixed value in the definition of the protocol, which in turn means that protocols are often both extremely large (e.g., individually operating on every well of a 96-well plate) and rigid, since locations cannot be supplied as a parameter value or determined dynamically at runtime.

In common practice, protocols are often described in terms of operations on physical or logical collections of samples, such as "Wells A1 to D2 in a standard 96-well plate," "A 6 by 3 group of 10ml tubes: three replicates each for six conditions," or "All wells showing green fluorescence >500 MEFL." Being able to represent such descriptions directly allows representations to be more compact, more intelligible to humans for both authoring and execution, and also more reusable, since they can be communicated as parameters or determined dynamically.

3.5 SBOL Represents Experimental Materials Used in Protocols

The **Synthetic Biology Open Language (SBOL)** [2, 22], provides succinct representations for all of the materials that would be used by a typical biological protocol—strains, reagents, media, experimental sample designs, and so on—along with the ability to track and distinguish between specific physical aliquots and replicates. On the input side of a protocol, SBOL's combinatorial design specifications [30] offer the ability to compactly specify combinations of experimental conditions (e.g., "measure absorbance at 12 time points from samples of cells transformed by 10 different genetic constructs at 8 different levels of induction, 3 replicates per condition"). SBOL also incorporates the **Ontology of Units of Measure (OM)** [29] for specifying and recording measurements, as well as the **W3C Provenance Ontology (PROV-O)** [24] for linking specifications, samples, and data via traces of activity records.

3.6 PROV-O Captures Execution Histories

While UML models activity flows, it does not actually provide a representation to capture activity executions and associate traces with data. For this, we use the Provenance Ontology (PROV-O) [24]. This last is precisely complementary to our selection of UML behaviors for representing workflows, as PROV-O leaves the actual definition of activities to users. Recall that a key objective of LabOP is to aid in tracing the connections between datasets and the protocols that produced them—the issue of *dataset provenance*. PROV-O is a well-accepted representation for encoding this kind of information, so we use it to represent the data-producing relationship between protocols, executions of protocols, and the resulting datasets. PROV-O also specifies several annotation properties that we adopt to mark up protocol executions, such as timestamps for the beginning and end of an activity execution. Thus, we can use PROV-O as the basis for capturing execution traces, with the activities in the trace defined using the UML data model, built from laboratory primitives based on Autoprotocol, and the inputs, outputs, and data relations encoded using SBOL.

4 PROTOTYPE

Our software implementation and proof-of-concept include a Python code library (labop), a web-based tool for visualizing, editing, and executing LabOP protocols (LabOPed), and example

protocols demonstrating how LabOP meets the requirements listed in Section 2. These resources are available on Github under the Biological Protocols Working Group organization [7].

In this section and the next, we will use as a running example the iGEM LUDOX protocol for calibration of plate reader **optical density (OD)** by comparing the absorbance of water and suspended silica. This protocol was first introduced in the 2016 iGEM interlaboratory study [5] and refined thereafter [4]: Specifically, we use its 2018 version as our running example. The LUDOX calibration protocol is an extremely simple protocol consisting of three steps: Water is added to four wells in a 96-well plate, LUDOX silica suspension is added to another four wells, and then all eight samples are measured at some specified absorbance (600 nm in its original usage) to obtain a baseline measurement of OD, validate machine behavior, and allow path-length correction. In addition to being simple enough to be a succinct example, this protocol is an ideal case study, as quantitative biophysical characterization and reproducibility are key goals for synthetic biology and, to date, the protocol has been performed in hundreds of labs in the context of the iGEM interlaboratory reproducibility studies.

4.1 The Specification Document and Python API Are Machine-generated

As introduced in Section 3, the LabOP data model is encoded as machine-readable OWL files. These specification files are the primary source from which we automatically generate a human-readable specification document as well as a Python API through code generation (Figure 3). The specification document includes automatically generated class diagrams and descriptions in addition to some static introductory material introducing the motivation and context of the specification. This document is built nightly using automated Github actions, ensuring that it is continually updated with respect to upstream changes in the OWL specification, and is available for download as a workflow artifact file. Readers are referred there for a detailed technical description of the data model [7].

The Labop Python library provides an object-oriented API that enables construction of protocol data structures using the classes in the LabOP data model. To facilitate protocol sharing and re-use, the back-end provides serialization and parsing of protocols into a number of standard RDF file formats, e.g., RDF/XML, Turtle, and so on. To generate the API automatically, we leverage a tool called SBOLFactory [3] that dynamically translates an ontology file into Python class definitions and an object-oriented API.

In addition to using OWL to specify LabOP's class structure, we use the **Shapes Constraint Language (SHACL)** to support automated validation of user-created protocol files. SHACL is an RDF-based language that describes graph patterns to which valid RDF documents must conform [19]. For LabOP, we have developed a set of SHACL constraints for protocol data structures. The Labop software stack leverages the pySHACL validator tool [33] to check our set of constraints and ensure that individual LabOP protocols are complete, consistent, and valid data structures. By using OWL and SHACL, the LabOP data model is specified using non-ambiguous, machine-readable languages. The use of OWL and SHACL thus decouples the specification from its implementation in any one programming language and from its formulation(s) in natural language.

Our approach of automatic generation has many advantages and may serve as a useful example for other standards development efforts. Because the standard is still in a relatively early stage of development and is expected to evolve, it is particularly valuable that revisions to the proposed data model can be rapidly generated, tested, and released. Moreover, since the human-readable specification document and the software library are generated from a single source, we avoid the possibility of introducing discrepancies between these resources while simultaneously lowering maintenance costs. These factors have enabled rapid development and will continue to support

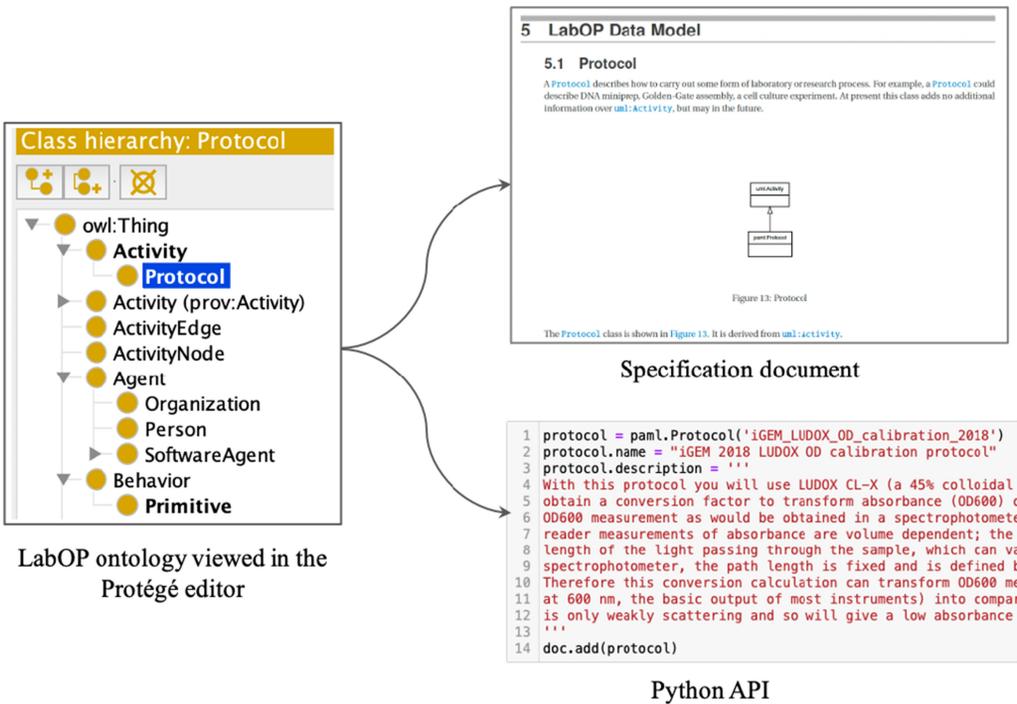


Fig. 3. The LabOP specification is fully machine-readable and encoded in OWL/SHACL from which documentation and software implementations are automatically generated.

responsive updates in our prototype implementations as new needs and use cases are identified by the community.

4.2 The labop Python API Enables Authoring Protocols and Developing Tools for Authoring Protocols

The labop Python API supports protocol editing operations by providing functions that build a protocol. Protocol generation scripts execute a sequence of API functions that construct the LabOP representation of a protocol.

For example, Figure 4 shows code implementing the iGEM LUDOX protocol Lines 5 to 7 define the protocol and add it to an SBOL3 document (representing the protocol as RDF). Line 10 defines an input parameter for the wavelength. Line 13 defines the SBOL3 object for double-distilled water, grounding it in a link to a PubChem identifier. Line 19 defines a microplate object that will hold the samples. Lines 22 to 25 identify the wells that will hold water and provision the water into those wells. Lines 30 to 33 identify which wells to measure and then measure the absorbance. Finally, lines 36 to 39 define the protocol output parameter for absorbance and link it to the output of the absorbance measurement activity.

Note that while programming at the level of the LabOP API is not feasible for many experimentalists, programmers can use these LabOP API functions to build visual editors that support authoring and inspection of protocols by experimentalists. LabOPed is an example of such a visual editor.

4.3 Protocols May Be Visualized and Understood as Linked Graphs

Visualization helps to specify and understand a protocol, as well as helping understand more generally how to think about workflows using the UML Activity model upon which LabOP is based.

```

1 import sbol3
2 import labop
3
4 # declare a protocol and add it to an SBOL3 document doc (doc initialization omitted).
5 protocol = labop.Protocol('iGEM_LUDOX_OD_calibration_2018')
6 protocol.name = "iGEM 2018 LUDOX OD calibration protocol"
7 doc.add(protocol)
8
9 # add an optional parameter for specifying the wavelength
10 wavelength_param = protocol.input_value('wavelength', sbol3.OM_MEASURE, optional=True, default_value=sbol3.Measure(600,
    tyto.OM.nanometer))
11
12 # create the materials to be provisioned (Ludox omitted for brevity)
13 ddh2o = sbol3.Component('ddh2o', 'https://identifiers.org/pubchem.substance:24901740')
14 ddh2o.name = 'Water'
15 doc.add(ddh2o)
16
17 # get a plate (spec omitted for brevity)
18 spec = labop.ContainerSpec('plateRequirement', queryString='cont:Plate96Well')
19 plate = protocol.primitive_step('EmptyContainer', specification=spec)
20
21 # identify wells to use
22 c_ddh2o = protocol.primitive_step('PlateCoordinates', source=plate.output_pin('samples'), coordinates='A1:D1')
23
24 # put water in selected wells
25 provision_ddh2o = protocol.primitive_step('Provision', resource=ddh2o, destination=c_ddh2o.output_pin('samples'), amount=
    sbol3.Measure(100, tyto.OM.microliter))
26
27 # similar Ludox PlateCoordinates and Provision steps omitted
28
29 # identify wells to use
30 c_measure = protocol.primitive_step('PlateCoordinates', source=plate.output_pin('samples'), coordinates='A1:D2')
31
32 # measure the absorbance
33 measure = protocol.primitive_step('MeasureAbsorbance', samples=c_measure.output_pin('samples'))
34
35 # link input parameter to measure primitive input
36 protocol.use_value(wavelength_param, measure.input_pin('wavelength'))
37
38 # link measurement output to protocol output
39 output = protocol.designate_output('absorbance', sbol3.OM_MEASURE, measure.output_pin('measurements'))

```

Fig. 4. LabOP Python script to construct a portion of the iGEM LUDOX calibration protocol. An interactive Jupyter notebook is available at <https://colab.research.google.com/drive/1WPvQ0REjHMEsginxXMj1ewqfFHZqSyM8?usp=sharing>.

A LabOP protocol consists of a set of activities and controls (nodes) that are linked by data and control flows (edges), all of which may be naturally visualized as a graph.

Figure 5 illustrates the rendering of the iGEM 2018 LUDOX calibration protocol via GraphViz [15]. It includes an initial control node (filled black circle) that is followed by the [acquire an] EmptyContainer activity node for allocating the microplate. The “followed by” relationship is denoted by a blue control edge. Activity nodes include “pins” for inputs and outputs. For example, the EmptyContainer node has an input pin named specification and an output pin named samples. Data flow edges, drawn as black arrows, link the activities’ pins. For example, the EmptyContainer samples output pin links to the source input pin for PlateCoordinates, indicating that the coordinates will reference samples within the empty container that has just been allocated. Data flow edges link either directly or through control nodes such as a fork node, illustrated by a black bar. Data flow edges also link protocol input—e.g., wavelength—and output—e.g., absorbance—parameters. These parameters are drawn as rectangles with doubled outlines.

While Figure 5 illustrates the protocol as a directed graph, in principle, LabOP can be presented in any number of formats. For example, the protocol illustrated by Figure 5 may also be described as a list of activities because the activities can be sorted into a total order. LabOP protocols that are partially ordered or include decision nodes can be represented by other paradigms such as block-based programs [21] or visual scripts [13, 36].

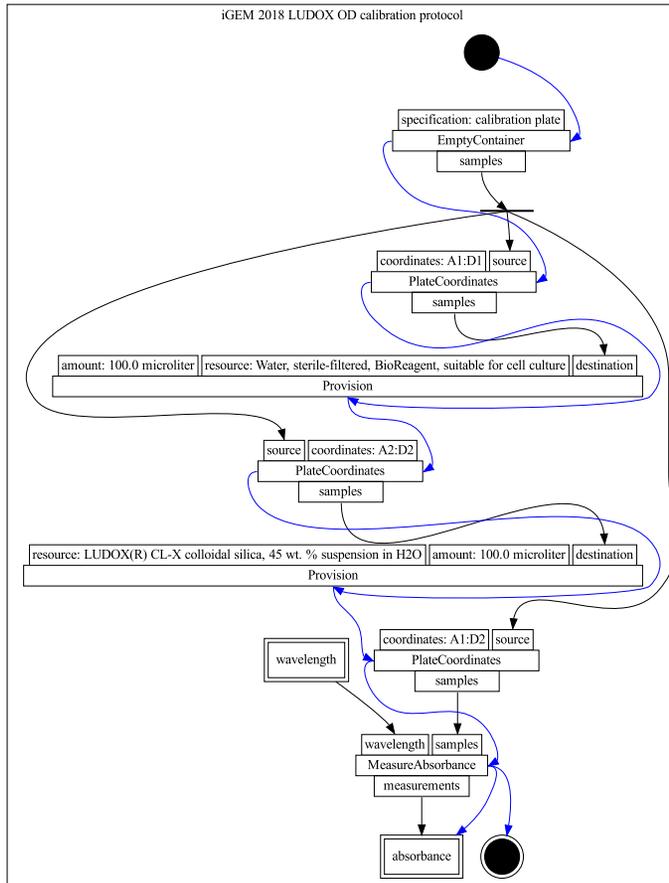


Fig. 5. LabOP protocol for the iGEM 2018 LUDOX calibration protocol, automatically rendered by LabOP visualizer. The graph includes protocol activities that follow a control flow denoted by blue edges and data flow denoted by black edges. Per UML diagram conventions, the initial node is shown as a black circle, the fork node is shown as a black bar, and the final node is shown as a double circle. The graph also illustrates protocol input (e.g., wavelength) and output (e.g., absorbance) parameters with double boxes.

4.4 Execution Can Be Specialized for Cross-laboratory Execution

To support execution, we have developed an execution engine with an extensible framework for third-party specializations that translate protocols into various target execution environments. This engine translates a LabOP protocol into another representation as a *side-effect* of simulating the execution of the protocol. In this way, LabOP serves as an intermediate representation for protocols enabling translation from one laboratory execution environment to another, employing different instrument platforms and execution languages. LabOP also supports conversion to natural language and various automation formats, including other protocol languages. Note that conversion from LabOP to other automation formats is not generally reversible, because these other formats are less expressive. The overall procedure for protocol execution is described below.

The execution engine uses a token-based execution semantics that implements the UML activity model based upon Petri-nets [28]. Execution (simulation) involves tracking a set of

tokens generated by each activity or control node in the protocol graph. Nodes may consume input tokens and generate output tokens, and tokens can either hold data representing objects created by activities or can denote control. When simulating an activity node for the purpose of translation, the engine will emit a translated description of the activity, such as one or more linked Autoprotocol objects, thus incrementally building the translation as a side effect of the simulation process.

After executing an activity, the execution engine non-deterministically selects one of the next enabled nodes in the control flow. Finally, the engine invokes specialized interpreters to generate instructions specific to the target execution environment that actuate equipment (or provide instructions to lab technicians) and that capture acquired measurement data and metadata in an execution trace. When targeting automation, such as Autoprotocol or Opentrons, such information can often be reported directly via the API for the automation system. When the execution is manual or no API integration is available, however, the protocol can be run interactively and data can be supplied back into the engine by a human operator.

Currently, our prototypes implement execution specializations that convert LabOP into Markdown,¹ Autoprotocol [23], and Opentrons [27] formats. In principle, however, specializations might also be configured to dynamically execute LabOP protocols by directly interfacing with instrument-specific APIs, rather than performing a translation, further enabling cross-platform laboratory execution. The specialization framework also includes methods for hooking into laboratory information management systems, as illustrated in Section 5.2.

Given the generality of LabOP workflows, the execution engine supports hierarchical nesting, re-use, and repetition of sub-protocols as activities within a protocol. As a consequence, serializing LabOP protocols to more restricted target languages may require compiling away some of the control structure. For example, control flow in Autoprotocol is limited to a strict linear step order with no branching or sub-protocols, so a LabOP protocol must be linearized with all loops “unrolled” and all sub-protocol invocations expanded inline to produce a linearized version of the protocol suitable for mapping into Autoprotocol. The Autoprotocol execution engine thus unrolls the protocol into a series of distinct executions, in which activities in sub-protocols may appear in the execution history multiple times as they are repeated. Furthermore, a single LabOP protocol may need to be translated into multiple Autoprotocol sequences. In such cases, a single Autoprotocol sub-sequence will be run to completion, and then there will have to be human intervention to choose the appropriate next sub-sequence to run, depending on the outcomes of the previous one.

5 DEMONSTRATION

In this section, we present examples demonstrating how the LabOP language and our current prototype implementation fulfill each of the design requirements laid out in Section 2. These demonstrations use the same running example as in the previous section: the iGEM LUDOX protocol for calibration of plate reader **optical density (OD)** [4, 5].

5.1 Execution by Either Humans or Machines

While some labs may perform the LUDOX calibration protocol using robotic liquid handlers, more resource-limited labs perform it using manual pipetting. LabOP is converted to either Autoprotocol or Markdown using specialized interpreters that are invoked by the execution engine. For example, the iGEM LUDOX calibration protocol is shown converted and rendered into Markdown and

¹Markdown is a rich text format designed to be readable both in its source form and to be translatable into other widely used formats such as HTML, PDF, and so on.

1 iGEM 2018 LUDOX OD calibration protocol

1.1 Description:

With this protocol you will use LUDOX CL-X (a 45% colloidal silica suspension) as a single point reference to obtain a conversion factor to transform absorbance (OD600) data from your plate reader into a comparable OD600 measurement as would be obtained in a spectrophotometer. This conversion is necessary because plate reader measurements of absorbance are volume dependent: the depth of the fluid in the well defines the path length of the light passing through the sample, which can vary slightly from well to well. In a standard spectrophotometer, the path length is fixed and is defined by the width of the cuvette, which is constant. Therefore this conversion calculation can transform OD600 measurements from a plate reader (i.e. absorbance at 600 nm, the basic output of most instruments) into comparable OD600 measurements. The LUDOX solution is only weakly scattering and so will give a low absorbance value.

1.2 Protocol Materials:

- Water, sterile-filtered, BioReagent, suitable for cell culture
- LUDOX(R) CL-X colloidal silica, 45 wt. % suspension in H2O

1.3 Protocol Inputs:

- wavelength = 600.0

1.4 Protocol Outputs:

- absorbance

1.5 Steps

1. Provision a container named `samples` meeting specification: `cont:ClearPlate` and `cont:SLAS-4-2004` and `(cont:wellVolume some ((on:hasUnit value om:microlitre) and (on:hasNumericalValue only xsd:decimal|[:~200~]:xsd:decimal|)))`.
2. Pipette 100.0 microliter of Water, sterile-filtered, BioReagent, suitable for cell culture into `samples(A1:D1)`
3. Pipette 100.0 microliter of LUDOX(R) CL-X colloidal silica, 45 wt. % suspension in H2O into `samples(A2:D2)`.
4. Make absorbance measurements (named `measurements`) of `samples(A1:D2)` at 600.0 nanometer.
5. Report values for absorbance from `measurements`.

(a) Generated Markdown

```

1  {"instructions": [
2  { "op": "provision",
3    "resource_id": "rs1c7pg8qs22dt",
4    "measurement_mode": "volume",
5    "to": [
6      {
7        "well": "samples/0",
8        "volume": "100:microliter"
9      },
10     <Others omitted>
11   ]},
12  { "op": "provision",
13    "resource_id": "rs1b6z2vgatkq7",
14    "measurement_mode": "volume",
15    "to": [
16      {
17        "well": "samples/1",
18        "volume": "100:microliter"
19      },
20     <Others omitted>
21   ]},
22  { "op": "spectrophotometry",
23    "dataref": "measurements",
24    "object": "samples",
25    "groups": [
26      {
27        "mode": "absorbance",
28        "mode_params": {
29          "wells": {
30            "samples/0",
31            <Others omitted>
32          },
33          "wavelength": [
34            "600:nanometer"
35          ]}}]}],
36  "refs": {
37    "samples": {
38      "id": "ct1g9qsg4wx6gcj",
39      "discard": true
40    }}

```

(b) Generated Autoprotocol

Fig. 6. (a) Markdown “paper protocol” generated from LabOP for iGEM 2018 LUDOX calibration protocol (the version presented here has been post-processed to translate the Markdown into PDF); (b) Autoprotocol for the iGEM LUDOX calibration protocol as generated by the Autoprotocol execution specialization.

Autoprotocol in Figure 6. The respective execution specializations collect the translated sequence of protocol activities and also help to resolve the objects appearing in the protocol. In addition to interpreting the steps, the specializations format the syntax needed for each target language:

The Markdown specialization interprets the protocol to construct a human-readable string that describes each step. The specialization also makes use of Markdown syntax to embed hyperlinks to online definitions, generated from the usages of materials or containers in the protocol. This hyperlinking is enabled by LabOP’s linked data nature and its leveraging of existing ontologies. For example, in Figure 6(a), each of the two materials is blue, because the Markdown includes a link to the material’s NCBI PubChem substance definition, which in turn provides purity and supplier information for these required reagents.

Similarly, the Autoprotocol specialization formats protocols as a list of instructions in JSON. For example, the Autoprotocol specialization interprets the `EmptyContainer` activity to generate code that will identify an available container in the Strateos **laboratory information management system (LIMS)** that satisfies the specification in the protocol. In Figure 6(b), this is implemented via lines 37–39, which declare a collection of wells named “samples” that are the targets of the provision and spectrophotometry steps.

5.2 Mapping Protocols from One Laboratory Environment to Another

Laboratories often differ in terms of the specific equipment, inventory, labware, and reagents that are available, even if they happen to be using the same execution platform, thus requiring substitutions and adaptations of a protocol at runtime. For this reason, the labop library is distributed with an ontology of labware and equipment. The purpose of this ontology is to maintain a catalog of specific instances of labware that may be commonly used in laboratories,

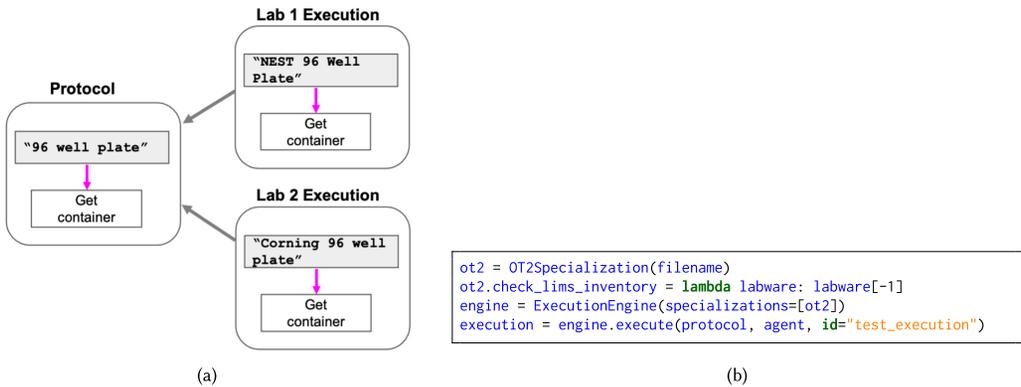


Fig. 7. (a) LabOP supports inferential reasoning to map protocols into laboratories with different equipment and inventory; (b) Invoking a protocol execution using the LabOP API and Opentrons specialization. The `check_lims_inventory` method provides a hook to interface with laboratory inventory control.

as well as their properties, such as geometric characteristics and from which vendor they may be procured. These labware instances belong to more general categories such as “plate,” “microfuge tube,” and “pipette tip rack.” Like the LabOP data model, this ontology is also specified using OWL and is queryable through the LabOP API. The labop execution specialization framework leverages this feature to select labware items that are available in a laboratory’s inventory and which most closely match a queried set of constraints.

For example, the LUDOX calibration protocol specifies that samples should be loaded into a “96 well plate.” A human technician ordinarily has little trouble interpreting these instructions and obtaining a suitable plate given the context of the protocol. However, execution on an Opentrons robot requires that the brand of 96 well plate is explicitly specified, as the robot must take into account vendor-specific differences in labware geometry. Therefore, when the LUDOX protocol is run using the Opentrons execution specialization, an inference is made to determine which instances of a 96 well plate are available in the laboratory’s inventory (Figure 7(a)). An example invocation of an Opentrons execution is shown in Figure 7(b). By overriding the `check_lims_inventory` method, the user hooks the specialization into their inventory management system to select a specific instance from a list of available labware. While this implementation currently simply chooses an arbitrary item from a list of available labware that satisfied the provided specification, a more nuanced user-defined `check_lims_inventory` could further refine its selection process based on multiple criteria, such as cost, availability, or preference for one brand versus another.

5.3 Representation of Diverse Workflow Patterns

Even a protocol as simple as the LUDOX protocol demonstrates how the universal workflow model that LabOP takes from UML can support and mix workflow patterns. The workflow diagram in Figure 5 shows sequential ordering, in which the plate must be allocated before materials are added to the wells, which in turn must happen before absorbance is measured. The same workflow illustrates nondeterministic ordering as well, in which the water and LUDOX can be provisioned in any order—or, in principle, even simultaneously with two sets of pipettes.

LabOP also permits conditional (branching) execution, where the output of a condition evaluation primitive flows into a decision node and the matching output of the decision node is executed.

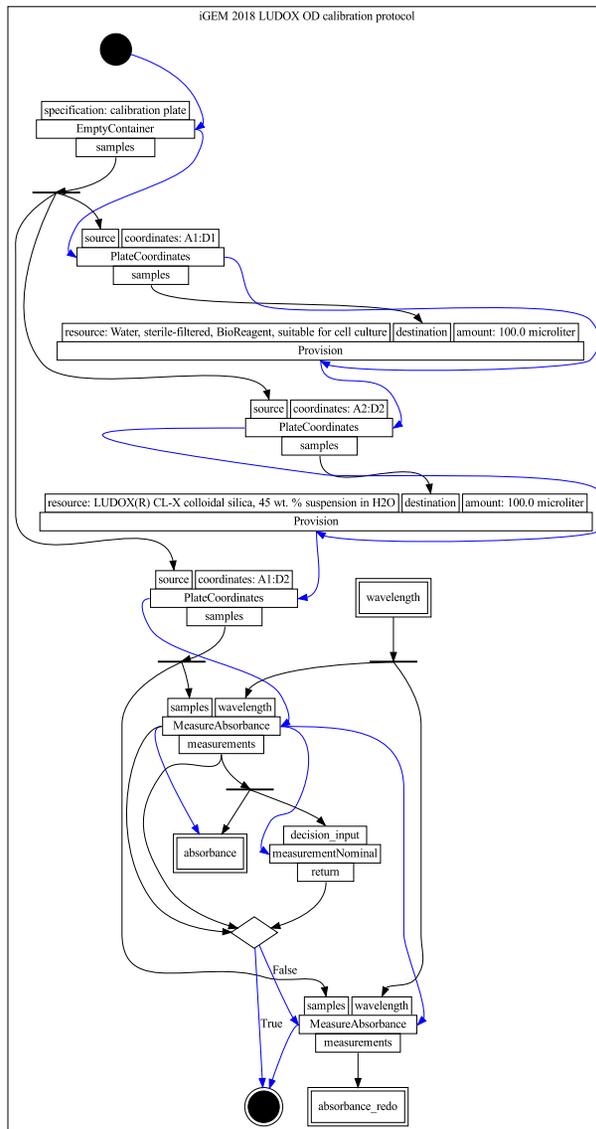


Fig. 8. LabOP uses decision nodes (diamonds) to enable runtime branching during protocol execution. The decision node branches execution on the return value of the measurementNominal primitive and executes a second MeasureAbsorbance primitive when measurementNominal returns False.

Figure 8 illustrates the addition of a condition evaluation primitive measurementNominal to assess whether the MeasureAbsorbance primitive measurements are within tolerance. If not nominal, then the protocol executes a second MeasureAbsorbance primitive. The condition evaluation primitive can be automated or handled by a human (providing a quality control checkpoint).

Finally, this workflow also illustrates extensibility. Of the 11 activities in the protocol workflow, only 4 are built into the language specification: the initial node, the fork, and the input and output variables. The other 7 activities are all taken from libraries, and while these specific libraries happen to be bundled with the LabOP distribution, the libraries are defined and implemented

using the same LabOP mechanism for extensibility that anyone can use to define new libraries or custom operations.

5.4 Maintaining Execution Records and Associated Metadata Markup

As described above, the LabOP execution engine includes support for creating persistent data structures that record a specific protocol execution and for linking such execution records back to protocol specifications. Figure 9 shows an example visualizing an execution trace for the LUDOX protocol, including execution times for every library activity and records of every piece of data that flowed along an edge in the protocol graph. Note that this trace also provides all of the links necessary for metadata tagging of the data collected in the course of protocol execution.

5.5 Recording Modifications of Protocols and the Relationship between Different Versions

In laboratory practice, protocols are frequently updated and revised. For example, different environmental parameters may be controlled or monitored, additional steps may be added to optimize yields, or new reagents may be introduced. Version control for laboratory protocols may help practitioners explore experimental design space more efficiently, improve scientific reproducibility, and reduce uncertainty in interpreting experimental results.

In accordance with our principle of leveraging existing tools as much as possible, LabOP serializations may be managed like any other software using conventional revision control systems. RDF-based serialization formats, such as RDF/XML or Turtle, are problematic for conventional software versioning systems, because these formats do not specify a deterministic order in which data structures are serialized. The N-triples RDF format, however, is deterministic when sorted, so LabOP uses this format as a stable serialization that can be readily differenced and inspected with standard, text-based version control tools. Figure 10(a) illustrates an example of version-control differencing between versions of a LabOP protocol in sorted N-triples RDF.

The LabOP data model does not intrinsically manage versioning of individual protocols, but rather relies on revision control systems. This has an added benefit of simplifying dependency management between related protocols. Groups of related protocols may be version-controlled collectively in a repository, thus avoiding the difficulty of maintaining explicit version requirements (i.e., “pins” or “ranges”) between related protocols. However, version tags are inserted into generated Markdown protocols to keep track of protocol hard-copies at the laboratory bench (Figure 10(b)).

Thus, LabOP allows protocols to be maintained by distributed communities of contributors using standard software development version control such as git, as well as the larger ecosystem of associated tooling for project management and community-driven development.

5.6 Verification and Validation of Protocol Completeness and Coherence

Supporting protocol authors in achieving correctness is an important goal for a protocol representation. While the implementation will depend on specific tooling, the representation specification must provide guidance as to what it means for a protocol to be complete, consistent, and so on. This is especially important for automatically executed protocols, since the control system cannot be counted on to repair flaws in protocols on the fly, and in the worst case, an incorrect specification could even cause damage to equipment or endanger lab personnel.

To aid the user in construction of valid protocols, the LabOP API provides a method for checking these validation rules. Figure 11 shows an example of this API being invoked on a malformed protocol, in this case for a protocol where the nodes representing its start condition, stop condition, and parameters have not been connected up to form a coherent workflow. Deeper levels of

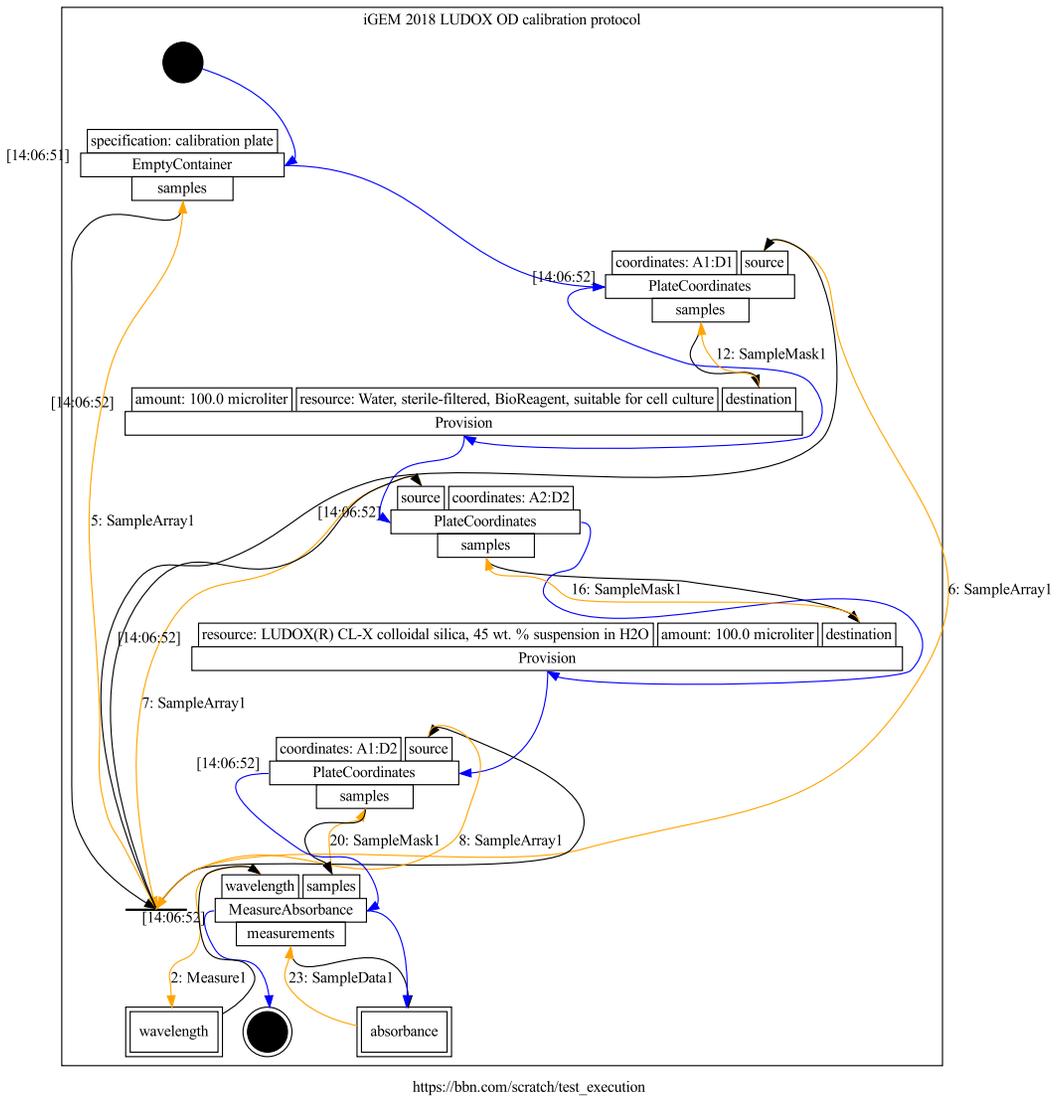


Fig. 9. LabOP execution trace for the iGEM 2018 LUDOX calibration protocol, layered on the protocol visualization shown in Figure 5. Yellow edges denote data flow with placeholder and computed values for an offline execution of the protocol.

semantic validation, such as checking whether a protocol might add more liquid to a container than it can hold, can be implemented with the aid of simulations in the execution engine.

5.7 Planning, Scheduling, and Allocation of Laboratory Resources

Protocol authors must ensure that protocols can be executed in the laboratory. To this end, LabOP models constraints on the activities in the form of temporal constraints and container specifications. Temporal constraints impose bounds on the temporal distance between timepoints, such as the start and end of activities, and can be used prior and during execution to schedule activities. For example, Figure 12 illustrates a schedule for the LUDOX calibration protocol as an

```
diff --git a/version_control_example-before.nt b/
    version_control_example-after.nt
index c08d99d..e4f4d69 100644
--- a/version_control_example-before.nt
+++ b/version_control_example-after.nt
@@ -370,370 @@
--https://bbn.com/scratch/iGEM_LUDOX_OD_calibration_2018/
    CallBehaviorAction9/ValuePin3/LiteralIdentified1/Measure1 > <
    http://www.ontology-of-units-of-measure.org/resource/om-2/
    hasNumericalValue > "15.0"^^< http://www.w3.org/2001/XMLSchema#
    double >
+<https://bbn.com/scratch/iGEM_LUDOX_OD_calibration_2018/
    CallBehaviorAction9/ValuePin3/LiteralIdentified1/Measure1 > <
    http://www.ontology-of-units-of-measure.org/resource/om-2/
    hasNumericalValue > "20.0"^^< http://www.w3.org/2001/XMLSchema#
    double >
@@ -385,385 @@
--<https://bbn.com/scratch/iGEM_LUDOX_OD_calibration_2018/
    CallBehaviorAction9/ValuePin4/LiteralInteger1 > <http://
    bioprotocols.org/uml#IntegerValue > "5000"^^< http://www.w3.org
    /2001/XMLSchema#integer > .
+<https://bbn.com/scratch/iGEM_LUDOX_OD_calibration_2018/
    CallBehaviorAction9/ValuePin4/LiteralInteger1 > <http://
    bioprotocols.org/uml#IntegerValue > "1000"^^< http://www.w3.org
    /2001/XMLSchema#integer > .
```

(a)

```
diff --git a/version_control_example-before.md b/
    version_control_example-after.md
index 8c047ea..473f08b 100644
--- a/version_control_example-before.md
+++ b/version_control_example-after.md
@@ -21.0,22 @@ is only weakly scattering and so will give a low
    absorbance value.
++ `fluorescence`
@@ -38,39.2 @@ is only weakly scattering and so will give a low
    absorbance value.
-5. Import data for `absorbance measurements of calibration plate`
    into provided Excel file.
+5. Measure fluorescence of `calibration plate` with excitation
    wavelength of 488.0 nanometer and emission filter of 507.0
    nanometer and 20.0 nanometer bandpass.
+6. Import data for `absorbance measurements of calibration plate`,
    `fluorescence measurements of calibration plate` into
    provided Excel file.
@@ -40.2,42.2 @@ is only weakly scattering and so will give a low
    absorbance value.
-Timestamp: 2022-12-19 13:36:34.364128
-Protocol version: v1.0a2-3-g5f38c11
+Timestamp: 2022-12-19 12:19:10.900737
+Protocol version: v1.0a2-1-g9f27697
```

(b)

Fig. 10. Use text-based formats can support effective version control of both LabOP protocols and protocols generated from LabOP for execution. For example, (a) shows a diff between versions of the LUDOX calibration protocol in LabOP, as sorted in sorted N-triples, a deterministic RDF serialization, and (b) shows a diff between generated Markdown for the LUDOX calibration protocol.

```
>>> report = invalid_protocol.validate()
>>> for error in report:
...     print(error)
https://bbn.com/scratch/broken/ActivityParameterNode1: Too few values for property parameter. Expected 1, found 0
https://bbn.com/scratch/broken/InitialNode1: InitialNode must have no incoming edges, but has 1
https://bbn.com/scratch/broken/FinalNode1: Node has no incoming edges, so cannot be executed
```

Fig. 11. The LabOP API provides validation checking for protocols. In this example, an invalid protocol contains improperly linked control flow between activity nodes.

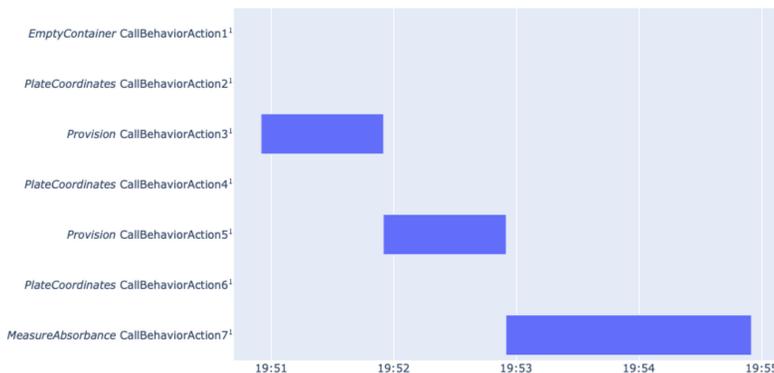


Fig. 12. Adding temporal constraints to the LUDOX calibration protocol allows schedulers to construct a schedule, shown here as an automatically generated Gantt chart.

automatically generated Gantt chart, where constraints between the start and end time of each activity enforce the duration and ordering of the activities. Container specifications constrain the types of containers that are compatible with the protocol, and, as previously described, facilitate allocation of laboratory resources.

The specifics of resource requirements and duration estimates will likely be a function of both the protocol and the available equipment in the laboratory in which it is to be executed (i.e., both the protocol and laboratory impose constraints). Which resources are limited and must be

considered in a planner or scheduler, versus those that can be effectively treated as unlimited, will also vary by laboratory, as will management styles and applicable policies.

6 FUTURE DIRECTIONS

The development efforts on LabOP described above have produced a representation that is simultaneously expressive enough and compact enough to satisfy all of the key goals that we have identified for a broadly applicable community standard. Our prototype implementation realizes this representation in the form of an ontology, specification, and Python library, which in turn have been used to implement test protocols and tools for visual editing and for execution, either by hand via export to a “paper protocol” or with laboratory robotics via export to Autoprotocol or OpenTrons.

The next critical stage in developing LabOP into an effective community standard for protocols is to refine the representation and expand the set of tools through involvement of interested stakeholders from the broader community. To that end, we organized an open community meeting at the COMBINE 2021 standards meeting in October 2021, during the course of which participants validated community interest in this initiative, prioritized next steps for LabOP, and began organization of an open pre-competitive community for its continued development. This community has continued to meet regularly and organize development, and more information about its activities may be found at <http://bioprotocols.org/>.

The key near-term goals for the development of LabOP, as prioritized by this community, are:

- putting LabOP to use in ongoing inter-laboratory collaborations within the stakeholder community,
- implementation of additional key execution environments, such as Emerald Cloud Lab, py-LabRobot [10], and protocols.io [34],
- implementing reasoning about the contents of samples, and
- improved user interfaces for protocol design, editing, and inspection.

If this nascent community is able to achieve these goals, particularly using LabOP to reduce protocol-related challenges faced by existing inter-laboratory collaborations, then it will form the basis for further development and utilization and, ultimately, may be able to establish an effective open standard representation for biological protocols, accelerating research and development across a broad range of fields and applications.

ACKNOWLEDGMENTS

This document does not contain technology or technical data controlled under either U.S. International Traffic in Arms Regulation or U.S. Export Administration Regulations. Views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. Approved for Public Release, Distribution Unlimited.

REFERENCES

- [1] Peter Amstutz, Michael R. Crusoe, Nebojša Tijanić, Brad Chapman, John Chilton, Michael Heuer, Andrey Kartashov, Dan Leehr, Hervé Ménager, Maya Nedeljkovich et al. 2016. Common workflow language, v1. 0. (2016). <https://doi.org/10.6084/m9.figshare.3115156.v2>
- [2] Hasan Baig, Pedro Fontanarrosa, Vishwesh Kulkarni, James Alastair McLaughlin, Prashant Vaidyanathan, Chris Myers, Bryan Bartley, Jacob Beal, Matthew Crowther, Thomas E. Gorochowski, Raik Grunberg, Goksel Misirli, Thomas Mitchell, Ernst Oberortner, James Scott-Brown, and Anil Wipat. 2021. Synthetic biology open language (SBOL) version 3.0.1. Retrieved from <https://github.com/SynBioDex/SBOL-specification/releases/tag/v3.0.1>.
- [3] Bryan Bartley. 2021. SBOLFactory: Ontology-driven code generation. In *HARMONY 2021*. https://github.com/SynBioDex/sbol_factory.

- [4] Jacob Beal, Geoff S. Baldwin, Natalie G. Farny, Markus Gershater, Traci Haddock-Angelli, Russell Buckley-Taylor, Ari Dwijayanti, Daisuke Kiga, Meagan Lizarazo, John Marken et al. 2021. Comparative analysis of three studies measuring fluorescence from engineered bacterial genetic constructs. *PLoS One* 16, 6 (2021), e0252263.
- [5] Jacob Beal, Traci Haddock-Angelli, Geoff Baldwin, Markus Gershater, Ari Dwijayanti, Marko Storch, Kim De Mora, Meagan Lizarazo, Randy Rettberg, and iGEM Interlab Study Contributors. 2018. Quantification of bacterial fluorescence using independent calibrants. *PLoS One* 13, 6 (2018), e0199432.
- [6] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. 2004. OWL Web Ontology Language Reference. Retrieved from <https://www.w3.org/TR/owl-ref/>.
- [7] Bioprotocols Working Group. 2023. LabOP. Retrieved from <https://github.com/bioprotocols>.
- [8] Alvis Brazma, Pascal Hingamp, John Quackenbush, Gavin Sherlock, Paul Spellman, Chris Stoeckert, John Aach, Wilhelm Ansorge, Catherine A. Ball, Helen C. Causton et al. 2001. Minimum information about a microarray experiment (MIAME): Toward standards for microarray data. *Nat. Genet.* 29, 4 (2001), 365–371.
- [9] Broad Institute. 2019. The Workflow Description Language and Cromwell. Retrieved from <https://software.broadinstitute.org/wdl>.
- [10] Emma J. Chory, Dana W. Gretton, Erika A. DeBenedictis, and Kevin M. Esvelt. 2021. Enabling high-throughput biology with flexible open-source automation. *Molec. Syst. Biol.* 17, 3 (2021), e9942.
- [11] UniProt Consortium. 2019. UniProt: A worldwide hub of protein knowledge. *Nucleic Acids Res.* 47, D1 (2019), D506–D515.
- [12] Kirill Degtyarenko, Paula de Matos, Marcus Ennis, Janna Hastings, Martin Zbinden, Alan McNaught, Rafael Alcántara, Michael Darsow, Mickaël Guedj, and Michael Ashburner. 2008. ChEBI: A database and ontology for chemical entities of biological interest. *Nucleic Acids Res.* 36 (2008), D344–D350. Retrieved from http://nar.oxfordjournals.org/content/36/suppl_1/D344.short.
- [13] Zdena Dobesova. 2011. Visual programming language in geographic information systems. In *Proceedings of the 2nd International Conference on Applied Informatics and Computing Theory*. World Scientific and Engineering Academy and Society (WSEAS), 276–280.
- [14] Karen Eilbeck, Suzanna E. Lewis, Christopher J. Mungall, Mark Yandell, Lincoln Stein, Richard Durbin, and Michael Ashburner. 2005. The sequence ontology: A tool for the unification of genome annotations. *Genome Biol.* 6 (2005), R44. Retrieved from <http://genomebiology.com/content/6/5/R44>.
- [15] John Ellson, Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, and Gordon Woodhull. 2003. Graphviz and dynagraph: Static and dynamic graph drawing tools. In *Graph Drawing Software*. Springer-Verlag, 127–148.
- [16] Jeremy Goecks, Anton Nekrutenko, and James Taylor. 2010. Galaxy: A comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biol.* 11, 8 (2010), R86.
- [17] Rebecca Jackson, Nicolas Matentzoglou, James A. Overton, Randi Vita, James P. Balhoff, Pier Luigi Buttigieg, Seth Carbon, Melanie Courtot, Alexander D. Diehl, Damion M. Dooley et al. 2021. OBO foundry in 2021: Operationalizing open data principles to evaluate ontologies. *Database* 2021 (Oct. 2021).
- [18] Sunghwan Kim, Paul A. Thiessen, Evan E. Bolton, Jie Chen, Gang Fu, Asta Gindulyte, Lianyi Han, Jane He, Siqian He, Benjamin A. Shoemaker et al. 2016. PubChem substance and compound databases. *Nucleic Acids Res.* 44, D1 (2016), D1202–D1213.
- [19] Holger Knublauch and Dimitris Kontokostas. 2017. Shapes Constraint Language (SHACL). Retrieved from <https://www.w3.org/TR/shacl/>.
- [20] Jamie A. Lee, Josef Spidlen, Keith Boyce, Jennifer Cai, Nicholas Crosbie, Mark Dalphin, Jeff Furlong, Maura Gasparetto, Michael Goldberg, Elizabeth M. Goralczyk et al. 2008. MIFlowCyt: The minimum information about a flow cytometry experiment. *Cytomet. Part A: J. Int. Societ. Analyt. Cytol.* 73, 10 (2008), 926–930.
- [21] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. 2010. The Scratch programming language and environment. *ACM Trans. Comput. Educ.* 10, 4 (2010), 1–15.
- [22] James Alastair McLaughlin, Jacob Beal, Göksel Mısırlı, Raik Grünberg, Bryan A. Bartley, James Scott-Brown, Prashant Vaidyanathan, Pedro Fontanarrosa, Ernst Oberortner, Anil Wipat et al. 2020. The Synthetic Biology Open Language (SBOL) version 3: Simplified data exchange for bioengineering. *Front. Bioeng. Biotechnol.* 8 (2020), 1009.
- [23] Ben Miles and Peter L. Lee. 2018. Achieving reproducibility and closed-loop automation in biological experimentation with an IoT-enabled lab of the future. *SLAS Technol.* 23, 5 (2018), 432–439.
- [24] Paolo Missier, Khalid Belhajjame, and James Cheney. 2013. The W3C PROV family of specifications for modelling provenance metadata. In *Proceedings of the 16th International Conference on Extending Database Technology*. ACM, 773–776.
- [25] Mark A. Musen. 2015. The protégé project: A look back and a look forward. *AI Matters* 1, 4 (2015), 4–12.
- [26] Object Management Group. 2017. OMG Unified Modeling Language (OMG UML) Version 2.5.1. Retrieved from <https://www.omg.org/spec/UML/>.

- [27] OpenTrons. 2020. OT-2 Python Protocol API Version 2. Retrieved from <https://docs.opentrons.com/v2/>.
- [28] Carl Adam Petri. 1966. *Communication with Automata*. Ph.D. Dissertation. Universitat Hamburg.
- [29] Hajo Rijgersberg, Don Willems, Xin-Ying Ren, Mari Wigham, and Jan Top. 2021. Ontology of units of Measure (OM), version 2.0.31. Retrieved from <http://www.ontology-of-units-of-measure.org/resource/om-2>.
- [30] Nicholas Roehner, Bryan Bartley, Jacob Beal, James McLaughlin, Matthew Pocock, Michael Zhang, Zach Zundel, and Chris J. Myers. 2019. Specifying combinatorial designs with the Synthetic Biology Open Language (SBOL). *ACS Synth. Biol.* 8, 7 (2019), 1519–1523.
- [31] Michael I. Sadowski, Chris Grant, and Tim S. Fell. 2016. Harnessing QbD, programming languages, and automation for reproducible biology. *Trends Biotechnol.* 34, 3 (2016), 214–227.
- [32] Nicholas Sioutos, Sherri de Coronado, Margaret W. Haber, Frank W. Hartel, Wen-Ling Shaiu, and Lawrence W. Wright. 2007. NCI Thesaurus: A semantic model integrating cancer-related clinical and molecular information. *J. Biomed. Inform.* 40, 1 (2007), 30–43.
- [33] Ashley Sommer and Nicholas Car. 2021. pySHACL. DOI: <https://doi.org/10.5281/zenodo.4750840>
- [34] Leonid Teytelman, Alexei Stoliartchouk, Lori Kindler, and Bonnie L. Hurwitz. 2016. Protocols.io: Virtual communities for protocol development and discussion. *PLoS Biol.* 14, 8 (2016), e1002538.
- [35] Keith F. Tipton, Richard N. Armstrong, Barbara M. Bakker, Amos Bairoch, Athel Cornish-Bowden, Peter J. Halling, Jan-Hendrik Hofmeyr, Thomas S. Leyh, Carsten Kettner, Frank M. Raushel et al. 2014. Standards for reporting enzyme data: The STRENDA consortium: What it aims to do and why it should be helpful. *Perspect. Sci.* 1, 1-6 (2014), 131–137.
- [36] Unity. 2023. Unity Visual Scripting. Retrieved from <https://unity.com/products/unity-visual-scripting>.
- [37] John Vivian, Arjun Arkal Rao, Frank Austin Nothhaft, Christopher Ketchum, Joel Armstrong, Adam Novak, Jacob Pfeil, Jake Narkizian, Alden D. Deran, Audrey Musselman-Brown et al. 2017. Toil enables reproducible, open source, big biomedical data analyses. *Nat. Biotechnol.* 35, 4 (2017), 314.
- [38] Justin Vrana, Orlando de Lange, Yaoyu Yang, Garrett Newman, Ayesha Saleem, Abraham Miller, Cameron Cordray, Samer Halabiya, Michelle Parks, Eriberto Lopez et al. 2021. Aquarium: Open-source laboratory software for design, execution and data management. *Synth. Biol.* 6, 1 (2021), ysab006.
- [39] Katherine Wolstencroft, Robert Haines, Donal Fellows, Alan Williams, David Withers, Stuart Owen, Stian Soiland-Reyes, Ian Dunlop, Aleksandra Nenadic, Paul Fisher et al. 2013. The Taverna workflow suite: Designing and executing workflows of web services on the desktop, web or in the cloud. *Nucleic Acids Res.* 41, W1 (2013), W557–W561.

Received 17 December 2021; revised 20 January 2023; accepted 26 May 2023