A Supplementary information

Supplementary Information document for article *Towards a Modular Architecture for Science Factories* by Rafael Vescovi, Tobias Ginsburg, Kyle Hippe, Doga Ozgulbas, Casey Stone, Abraham Stroka, Rory Butler, Ben Blaiszik, Tom Brettin, Kyle Chard, Mark Hereld, Arvind Ramanathan, Rick Stevens, Aikaterini Vriza, Jie Xu, Qingteng Zhang, and Ian Foster.

A.1 An example workcell specification: RPL

We present below the YAML description of the workcell illustrated in Figure 8. This workcell is used by both the PCR workflow in Section A.2 and the color picker workflow in Section A.3.

In summary: A configuration block (lines 4–11) give identifiers for external services that may be used when running the workcell, and Line 12 gives the coordinates of the workcell in the laboratory. Lines 15–135 provide for each module in the workcell the following information. `Name`: A name for this module instance, unique within the workcell. `Model`: A name for the physical component name in the module. `Interface`: The adapter used by the module: e.g., `wei_ros_node` to indicate that actions will be sent over a ROS service. `Config`: Configuration information, e.g., a ROS channel name. `workcell_coordinates`: The physical location(s) of the component, expressed as an (X, Y, Z) position relative to the workcell origin, plus a four-element quarternion representing orientation. Finally, lines 137 forward present the locations of the various exchange locations in the workcell, most as seven-element `pf400` joint values.

Listing 1. Example workcell specification, rpl_modular_workcell.yaml

```
1  name: RPL_Modular_workcell
2
3  #Info about data processing and location of the workcell
4  config:
5    globus_compute_local: "299edea0-db9a-4693-84ba-babfa655b1be"
6    globus_transfer_local: ""
7    ##
8    globus_search_index: "aefcecc6-e554-4f8c-a25b-147f23091944"
9    globus_portal_ep: "bb8d048a-2cad-4029-a9c7-671ec5d1f84d"
10   ##
11   globus_group: "dda56f31-53d1-11ed-bd8b-0db7472df7d6"
12   workcell_origin_coordinates: [9.4307, -10.4176, 0, 1, 0, 0, 0]
13
14 #List of all components accessible in this workcell
15 modules:
16
17   #Human-legible name specific to an individual component
18   - name: sealer
19
20     #Type of device being connected to, might be multiple components of same
21     #model with different names in a workcell
22     model: A4S_sealer
23
24     #Method of communication with the device
25     interface: wei_ros_node
26
27     #Relevant communication configuration
28     config:
29
30       #In this case, the ROS service used to send actions to the component
31       ros_node_address: "/std_ns/SealerNode"
32
33     # X, Y, Z, Q0, Q1, Q2, Q3 relative to workcell_origin_coordinates
34     workcell_coordinates: [0.8209, 0.7264, 1.0514, 0.7071, 0, 0, 0.7071]
35
36   - name: pf400
37     model: pf400
```

```
    interface: wei_ros_node
    config:
      ros_node_address: "/std_ns/pf400Node"
    workcell_coordinates: [0, -0.0321, 0.8420, 0.7071, 0, 0, 0.7071]

  - name: sciclops
    model: sciclops
    interface: wei_ros_node
    config:
      ros_node_address: "/std_ns/SciclopsNode"
    workcell_coordinates: [0.5713, 1.0934, 1.0514, 0.9831, 0, 0, 0.1826]

  - name: peeler
    model: brooks_xpeel
    interface: wei_ros_node
    config:
      ros_node_address: "/std_ns/PeelerNode"
    workcell_coordinates: [0.7464, 0.3862, 1.0514, 0.7071, 0, 0, 0.7071]

  - name: ot2_pcr_alpha
    model: ot2
    interface: wei_ros_node
    config:
      ros_node_address: "/std_ns/ot2_pcr_alpha"
    workcell_coordinates: [0.7400, -0.2678, 1.0514, 0.7071, 0, 0, 0.7071]

  - name: ot2_growth_beta
    model: ot2
    interface: wei_ros_node
    config:
      ros_node_address: "/std_ns/ot2_growth_beta"
    workcell_coordinates: [0.7071, 0, 0, 0.7071, 0.7071, 0, 0, 0.7071]

  - name: ot2_cp_gamma
    model: ot2
    interface: wei_ros_node
    config:
      ros_node_address: "/std_ns/ot2_cp_gamma"
    workcell_coordinates: [-0.7315, -1.0018, 1.0514, -0.7071, 0, 0, 0.7071]

  - name: biometra
    model: biometra (96well)
    interface: wei_ros_node
    config:
      ros_node_address: "/std_ns/biometra96"
    workcell_coordinates: [-0.6283, 0.6415, 1.0514, 0.7071, 0, 0, -0.7071]

  - name: biometra_192
    model: biometra (192well)
    interface: wei_ros_node
    config:
      ros_node_address: "/std_ns/biometra_192"
    workcell_coordinates: [-0.6283, 0.3481, 1.0514, 0.7071, 0, 0, -0.7071]

  - name: camera_module
    model: camera (logitech)
    interface: wei_ros_camera
    config:
      ros_node_address: "/std_ns/camera_module"
    workcell_coordinates: [0,0,0,1,0,0,0]

  - name: hidex
    model: Hidex
    interface: wei_tcp_node
    workcell_coordinates: [0,0,0,1,0,0,0]
    config:
      tcp_node_address: "146.137.240.22"
      tcp_node_port: 2000

  - name: barty
    model: RPL BARTY
    interface: wei_rest_node
    workcell_coordinates: [0,0,0,1,0,0,0]
    config:
      rest_node_address: "http://barty.cels.anl.gov:8000"
      rest_node_auth: ""

  - name: MiR_base
    model: MiR250
    interface: wei_rest_node
    workcell_coordinates: [0,0,0,1,0,0,0]
    config:
      rest_node_address: "http://mirbase1.cels.anl.gov/api/v2.0.0/"
      rest_node_auth: "/home/rpl/Documents/mirauth.txt"

  - name: ur5
    model: ur5
    interface: wei_ros_node
    workcell_coordinates: [0,0,0,1,0,0,0]
    config:
      ros_node_address: "/ur5_client/UR5_Client_Node"

locations:
  pf400: #Joint angles for the PF400 Plate Handler
    sciclops.exchange: [222.0, -38.068, 335.876, 325.434, 79.923, 995.062]
    sealer.default: [205.128, -2.814, 264.373, 365.863, 79.144, 411.553]
    peeler.default: [225.521, -24.846, 244.836, 406.623, 80.967, 398.778]
    ot2_pcr_alpha.deck1_cooler: [247.999, -30.702, 275.835, 381.513, 124.830, -585.403]
    ot2_growth_beta.deck2: [163.230, -59.032, 270.965, 415.013, 129.982, -951.510]
    ot2_cp_gamma.deck2: [156, 66.112, 83.90, 656.404, 119.405, -946.818]
    biometra.default: [247.0, 40.698, 38.294, 728.332, 123.077, 301.082]
    camera_module.plate_station: [90.597,26.416, 66.422, 714.811, 81.916, 995.074]
    wc.trash: [218.457, -2.408, 38.829, 683.518, 89.109, 995.074]
  sciclops: #Joint angles for the Sciclops Plate Crane
    sciclops.exchange: [0,0,0,0]
  workcell: #Coordinates relative to the workcell origin
    sciclops.exchange: [0.7400, -0.2678, 1.0514, 0.7071, 0, 0, 0.7071]
```

A.2 An example workflow specification: PCR

We show here the YAML description of the PCR workflow described in Section 4.2. Lines 2-5 provide metadata; lines 9–15 list the modules to be used; and lines 19 forward describe the workflow steps.

Listing 2. Example workflow specification, pcr_demo_wf.yaml

```yaml
name: PCR - Workflow

metadata:
  author: RPL Team
  info: Example workflow for WEI
  version: 0.1

#This is a list of modules used in the workflow
modules:
  - name: ot2_pcr_alpha
  - name: pf400
  - name: peeler
  - name: sealer
  - name: biometra
  - name: sciclops
  - name: camera_module

#This is a list of steps in the workflow
#each step represents an action on a single module
flowdef:

  #This is a human legible name for the step
  - name: Sciclops gets plate from stacks

  #This defines which module the action will run on
    module: sciclops

  #This tells the module which action in its library to run
    action: get_plate

  #These arguments specify the parameters for the action above
    args:
      loc: "tower2"

  #This is a place for additional notes
    comment: Stage pcr plates

  - name: pf400 moves plate from sciclops to ot2
    module: pf400
    action: transfer
    args:
      source: sciclops.exchange
      target: ot2_pcr_alpha.deck1_cooler
      source_plate_rotation: narrow
      target_plate_rotation: wide

  - name: ot2 runs the "Mix␣reactions" protocol
    module: ot2_pcr_alpha
    action: run_protocol
    args:
      config_path: PCR_prep_full_plate_multi_noresource.yaml
      use_existing_resources: False

  - name: pf400 moves plate from ot2 to sealer
    module: pf400
    action: transfer
    args:
      source: ot2_pcr_alpha.deck1_cooler
      target: sealer.default
      source_plate_rotation: wide
      target_plate_rotation: narrow

  - name: Seal plate in sealer
    module: sealer
    action: seal
    args:
      time: payload:seal.time
      temperature: 175

  - name: pf400 moves plate from sealer to biometra
    module: pf400
    action: transfer
    args:
      source: sealer.default
      target: biometra.default
      source_plate_rotation: narrow
      target_plate_rotation: wide

  - name: Close lid of biometra
    module: biometra
    action: close_lid

  - name: Run biometra program
    module: biometra
    action: run_program
    args:
        program_n: 3

  - name: Open lid of biometra
    module: biometra
    action: open_lid

  - name: pf400 moves plate from biometra to peeler
    module: pf400
    action: transfer
    args:
      source: biometra.default
      target: peeler.default
      source_plate_rotation: wide
      target_plate_rotation: narrow

  - name: Peel plate
    module: peeler
    action: peel

  - name: pf400 moves plate from peeler to camera
    module: pf400
    action: transfer
    args:
        source: peeler.default
        target: camera_module.plate_station
        source_plate_rotation: narrow
        target_plate_rotation: narrow

  - name: camera takes picture of plate
    module: camera_module
    action: take_picture
    args:
      file_name: "final_image.jpg"

  - name: pf400 moves plate to final location
    module: pf400
    action: transfer
    args:
      source: camera_module.plate_station
      target: wc.trash
      source_plate_rotation: narrow
      target_plate_rotation: narrow
```

A.3 A second example workflow specification: Color mixing

We show here the YAML description of the second workflow, `cp_wf_mixcolor.yaml`, used by the color picker application described in Section 4.1 and shown in Section A.4.

Listing 3. Example workflow specification, cp_wf_mixcolor.yaml

```
name: Color Picker - Mix Colors - Workflow

metadata:
  author: Tobias Ginsburg, Rafael Vescovi
  info: Main workflow for the RPL Color Picker
  version: 0.1

modules:
  - name: ot2_cp_gamma
  - name: pf400
  - name: camera_module

flowdef:
  - name: Move from Camera Module to OT2
    module: pf400
    command: transfer
    args:
      source: camera_module.positions.plate_station
      target: ot2_cp_gamma.positions.deck2
      source_plate_rotation: narrow
      target_plate_rotation: wide
    comment: Place plate in ot2

  - name: Mix all colors
    module: ot2_cp_gamma
    command: run_protocol
    args:
      config_path: combined_protocol.yaml
      color_A_volumes: payload.color_A_volumes
      color_B_volumes: payload.color_B_volumes
      color_C_volumes: payload.color_C_volumes
      destination_wells: payload.destination_wells
      use_existing_resources: payload.use_existing_resources
    comment: Mix colors A, B and C portions according to input data

  - name: Move to Picture
    module: pf400
    command: transfer
    args:
      source: ot2_cp_gamma.positions.deck2
      target: camera_module.positions.plate_station
      source_plate_rotation: wide
      target_plate_rotation: narrow

  - name: Take Picture
    module: camera_module
    command: take_picture
    args:
      file_name: "final_image.jpg"
```

A.4 An example application: Color picker

We present below a somewhat simplified version of the color-picker application described in Section 4.1, which runs three workflows (including the `cp_wf_mixcolor.yaml` of Section A.3) on a workcell with a `pf400`, `ot2`, and `camera`. The Python program line runs a first workflow to get a new plate for subsequent use for color mixing (line 58). The program then runs a loop until the experiment budget is exhausted (line 63). In each iteration, it calculates what colors to mix (line 65), runs a second workflow to mix and image the colors (line 74), calls Globus Compute to run an image analysis program (line 79), and runs a Globus flow to publish relevant information (lines 116). Finally, it runs a third workflow to discard the used plate (line 123). Throughout, various logging commands publish events regarding significant occurrences, for consumption by external monitoring programs. This is done explicitly with exp.events, as in lines 62, 118, and 124, and also implicitly whenever a workflow is run.

Listing 4. Example application program, color_picker_app.py

```
 1 # For extracting colors from each plate
 2 from tools.plate_color_analysis import get_colors_from_file
 3
 4 from solvers.evolutionary_solver import EvolutionaryColorSolver
 5 from globus_compute_sdk  import Client
 6 gcc = Client()
 7
 8 # For publishing to RPL Portal
 9 from tools.publish_v2 import publish_iter
10
11 # For creating a payload that the OT2 will accept from the solver output
12 from tools.color_utils import convert_volumes_to_payload
13
14
15
16 # For running workflows, submits the job and waits for it to complete.
17 from tools.run_flow import run_flow
18
19 #For managing the overall experiment, generates a unique ID, handles logging and
       running flows. Is passed to the ruN_flow function to facillitate communication.
20 from rpl_wei.exp_app import Experiment
21
```

```
MAX_PLATE_SIZE = 96

def run(target_color, exp_budget, pop_size): #
    # workflows used
    wf_dir = Path("/home/rpl/workspace/rpl_workcell/color_picker_app/workflows")
    init_workflow = wf_dir / "cp_wf_newplate.yaml"
    loop_workflow = wf_dir / "cp_wf_mixcolor.yaml"
    final_workflow = wf_dir / "cp_wf_trashplate.yaml"

    # Constants
    solver_out_dim = (pop_size, 3)

    # Resource Tracking:
    plate_n = 1  # total number of plates
    current_iter = 0  # total number of itertions
    num_exps = 0  # total number of wells used
    curr_wells_used = []  # list of all wells used

    # Information Tracking:
    current_plate = None
    cur_best_color = None
    cur_best_diff = float("inf")
    diffs = []  # List of all diffs from all runs of the experiment
    payload = {}  # Payload to be sent to the workflow runs

    #Globus Compute Setup:
    gc_endpoint = "299edea0-db9a-4693-84ba-babfa655b1be"

    # Initialize the Experiment: start logging on the server and create log files.
    # The events object it used to log important occurences in application execution.
    # Logging is also integrated in the run_flow and publish_iter functions
    exp = Experiment("127.0.0.1", "8000", "Color_Picker")
    exp.register_exp()

    # Step 1: Run first workflow, to get a new plate
    steps_run = []
    steps_run, _ = run_flow(init_workflow, payload, steps_run, exp)
    curr_wells_used = []

    # Step 2: Repeatedly mix and image colors
    exp.events.log_loop_start("Main Loop")
    while num_exps + pop_size <= exp_budget:
        # Step 2a: Calculate volumes and current wells for use in the OT2 protocol
        plate_volumes = EvolutionaryColorSolver.run_iteration(
            target_color,
            current_plate,
            pop_size=pop_size,
            out_dim=(pop_size, 3),
            return_volumes=True,
        )

        # Step 2b: Run second workflow, to mix colors
        steps_run, run_info = run_flow(loop_workflow, payload, steps_run, exp)

        # Step 2c: Analyze image: output should be list [pop_size, 3]
        exp.events.log_globus_compute("get_color_from_file")
        img_path = run_info["hist"]["Take Picture"]["action_msg"]
        gc_result = gcc.run(get_colors_from_file,  gc_endpoint, img_path)
        plate_colors_ratios = gc_result.result()[1]

        # Swap BGR to RGB
        plate_colors_ratios = {a: b[::-1] for a, b in plate_colors_ratios.items()}

        # Find the colors to be processed by the solver
        current_plate = [],
        wells_used = []
        for well in payload["destination_wells"]:
            color = plate_colors_ratios[well]
            wells_used.append(well)
            current_plate.append(color)

        # save those and the initial colors, etc
        exp.events.log_local_compute("find_best_color")
        plate_best_color_ind, plate_diffs = EvolutionaryColorSolver._find_best_color(
            current_plate, target_color, cur_best_color
        )
        plate_best_color = current_plate[plate_best_color_ind]
        plate_best_diff =
            EvolutionaryColorSolver._color_diff(plate_best_color, target_color)
        diffs.append(plate_diffs)

        # Find best colors
        if plate_best_diff < cur_best_diff:
            cur_best_diff = plate_best_diff
            cur_best_color = plate_best_color

        # Step 2d: Record best colors
        report = {
            "best_color": cur_best_color,
            "best_diff": cur_best_diff
        }

        with open(folder_path / "report.txt") as f:
            json.dump(report, f)
        publish_iter(folder_path, dest_path, exp)

        exp.events.log_loop_check(
            "Sufficient wells in experiment budget", num_exps + pop_size <=
   exp_budget
        )

    # Step 3: Run third workflow
    steps_run, _ = run_flow(final_workflow, payload, steps_run)
    exp.events.end_experiment()
# Run the application. Arguments are:
#   Target color: RGB representation of target color
#   Experiment budget: Number of colors to mix and image in total
#   Population size: Number of colors to mix and image at a time
run( [120, 120, 120], 100, 10 )
```

A.5 Pointers to code

The applications, workflows, and protocols discussed in this paper, other than for the Thin Film application, are available publicly on GitHub. For each of the repositories discussed, a tagged release under "Digital Discovery Paper" can be found with the version of the codes used in this work. See `https://ad-sdl.github.io/wei2023` for summary information and pointers. Below, we provide pointers to relevant individual code files. In addition, the main core of the workflow executor is here: `https://github.com/AD-SDL/wei`.

A.5.1 Color Picker

Application: `https://github.com/AD-SDL/rpl_workcell/blob/main/color_picker_app/color_picker_application.py`

Workflows:

- `https://github.com/AD-SDL/rpl_workcell/blob/main/color_picker_app/workflows/cp_wf_newplate.yaml`
- `https://github.com/AD-SDL/rpl_workcell/blob/main/color_picker_app/workflows/cp_wf_mixcolor.yaml`
- `https://github.com/AD-SDL/rpl_workcell/blob/main/color_picker_app/workflows/cp_wf_trashplate.yaml`

OT2 protocol: `https://github.com/AD-SDL/rpl_workcell/blob/main/color_picker_app/protocol_files/combined_protocol.yaml`

Solver: `https://github.com/AD-SDL/rpl_workcell/blob/main/color_picker_app/solvers/evolutionary_solver.py`

A.5.2 PCR

Application: `https://github.com/AD-SDL/rpl_workcell/blob/main/pcr_app/pcr_full_application.py`

Workflow: `https://github.com/AD-SDL/rpl_workcell/blob/main/pcr_app/workflows/pcr_demo_wf.yaml`

OT2 protocol: `https://github.com/AD-SDL/rpl_workcell/blob/main/pcr_app/protocol_files/PCR_prep_full_plate_multi_noresource.yaml`

A.5.3 Growth curve

Application: `https://github.com/AD-SDL/BIO_workcell/blob/main/growth_app/growth_curve_app.py`

Workflows:

- `https://github.com/AD-SDL/BIO_workcell/blob/main/growth_app/workflows/create_plate_T0.yaml`
- `https://github.com/AD-SDL/BIO_workcell/blob/main/growth_app/workflows/read_plate_T12.yaml`

The code used to generate the `solo` protocol is at `https://github.com/AD-SDL/BIO_workcell/blob/main/growth_app/tools/hudson_solo_auxillary/hso_functions.py`

A.5.4 Pendant drop

Application: `https://github.com/AD-SDL/8IDI_workcell/blob/main/droplet_app/droplet_app.py`

Workflow: `https://github.com/AD-SDL/8IDI_workcell/blob/main/droplet_app/workflows/demo.yaml`

A.5.5 Electrochromic Polymers

Application: `https://github.com/AD-SDL/rpl_workcell/blob/main/polybot_app/demo_app.py`

Workflow: `https://github.com/AD-SDL/rpl_workcell/blob/main/polybot_app/workflows/demo.yaml`