EasyVVUQ Bringing your UQ workflows to the exascale! VECMA

Vytautas Jancauskas (jancauskas@lrz.de) [1], Jalal Lakhlili (jalal.lakhlili@ipp.mpg.de) [2], Onnie Luk [2], Wouter Edeling [3], Robin A. Richardson [4], David W. Wright [5], Robert Sinclair [6], Bartosz Bosak [7], Piotr Kopta [7], Derek Groen [8]

[1] Leibniz-Rechenzentrum der Bayerischen Akademie der Wissenschaften, Germany [2] Max-Planck-Institut für Plasmaphysik, Germany [3] Centrum Wiskunde & Informatica, Netherlands [4] eScience Center, Netherlands [5] GTN Ltd, London, UK [6] University College London, UK [7] Poznań Supercomputing and Networking Center [8] Department of Computer Science, Brunel University London, UK.

What is EasyVVUQ?

https://github.com/UCL-CCS/EasyVVUQ

EasyVVUQ is a Python 3 library designed to facilitate verification, validation and uncertainty quantification (VVUQ) for a wide variety of simulations. It was conceived and developed within the EU funded VECMA (Verified Exascale Computing for Multiscale Applications) project.

We use the existing Python scientific computing infrastructure and extend it with tools for executing the jobs in HPC environments. We also automate away most of the tedious details of UQ usage scenarios.

Why You Need EasyVVUQ?

UQ typically requires a very large number of jobs to be run and analysed. Most existing UQ tools are not designed with large scale execution in mind. Therefore, we have designed EasyVVUQ to augment them with the capabilities to run on modern HPC machines. The framework is designed to be open-ended with regards to the execution of simulations. It is currently possible to execute jobs locally or on HPC clusters using QCG pilot job or Dask JobQueue.

Existing simulations can make use of EasyVVUQ by extending several of the classes that make up the framework. In most cases these would just be the `Encoder` and `Decoder` responsible for parsing the inputs and outputs of your simulation respectively.

The end goal of EasyVVUQ is to allow you to test your UQ procedures on your laptop and then take it directly to a supercomputer. It is designed to be flexible, fault tolerant and resumable. In the code example below, we outline a typical use scenario. It involves a toy simulation of a cooling coffee cup and goes through all the main stages of a UQ run with EasyVVUQ.

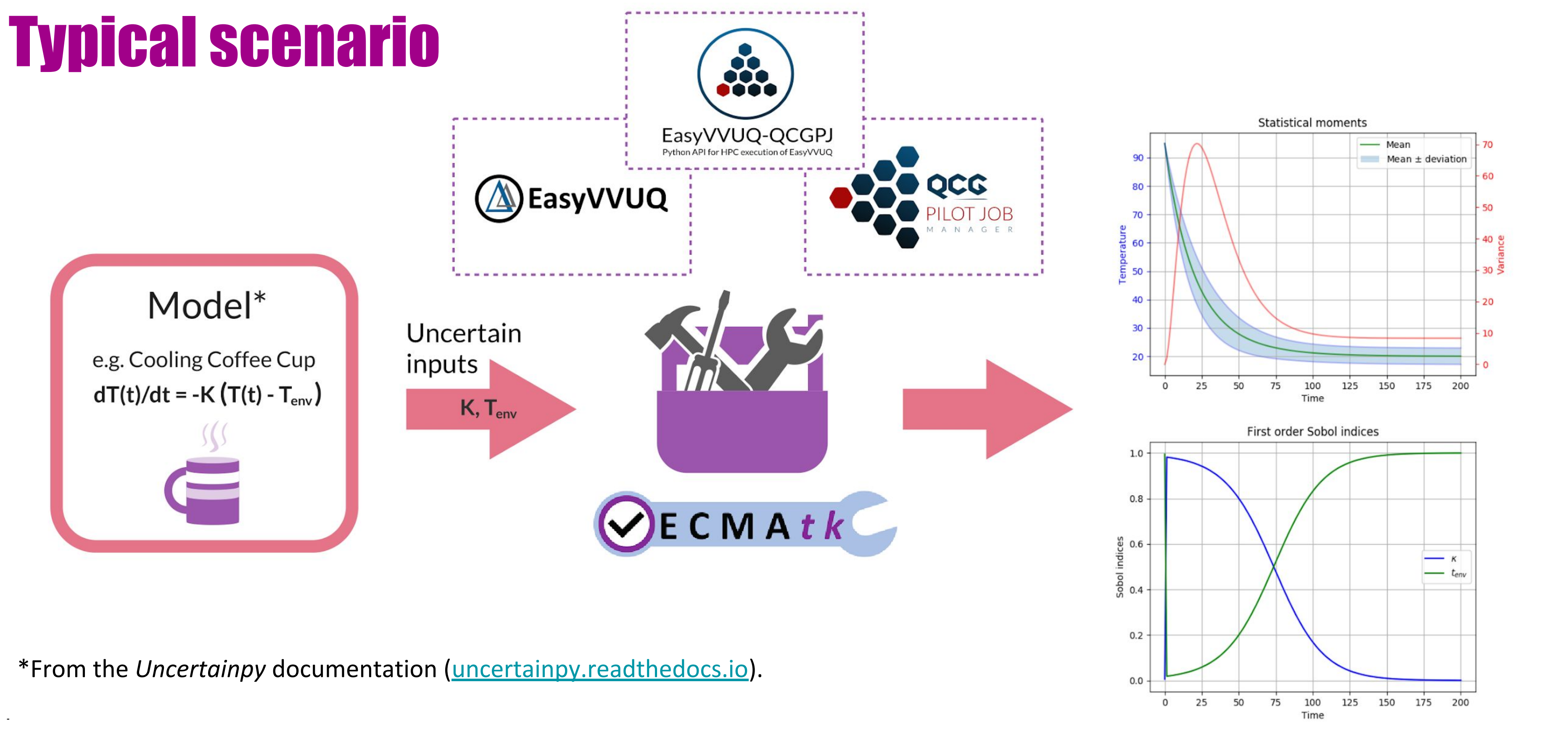
Large scale execution

Direct submission of a large group of jobs to a scheduling system can result in long aggregated time to finish as each single job is scheduled independently and waits in the queue. To ensure efficient allocation, we create large container jobs within which we can then allocate and schedule the individual simulation runs. To do this, we can use either the QCG PilotJob[3] component in the VECMA Toolkit or Dask JobQueue[5].

QCG PilotJob Manager is designed to schedule and execute many jobs inside a single allocation. It provides a container for sub-jobs, for which necessary resources are then assigned efficiently. In order to simplify the use of QCG PilotJob from EasyVVUQ workflows, a lightweight integrating code EASYVVUQ-QCGPJ[4] is provided (e.g. for Fusion and UrbainAir applications[7]). Users can also choose to use Dask JobQueue, which is useful for smaller runs.

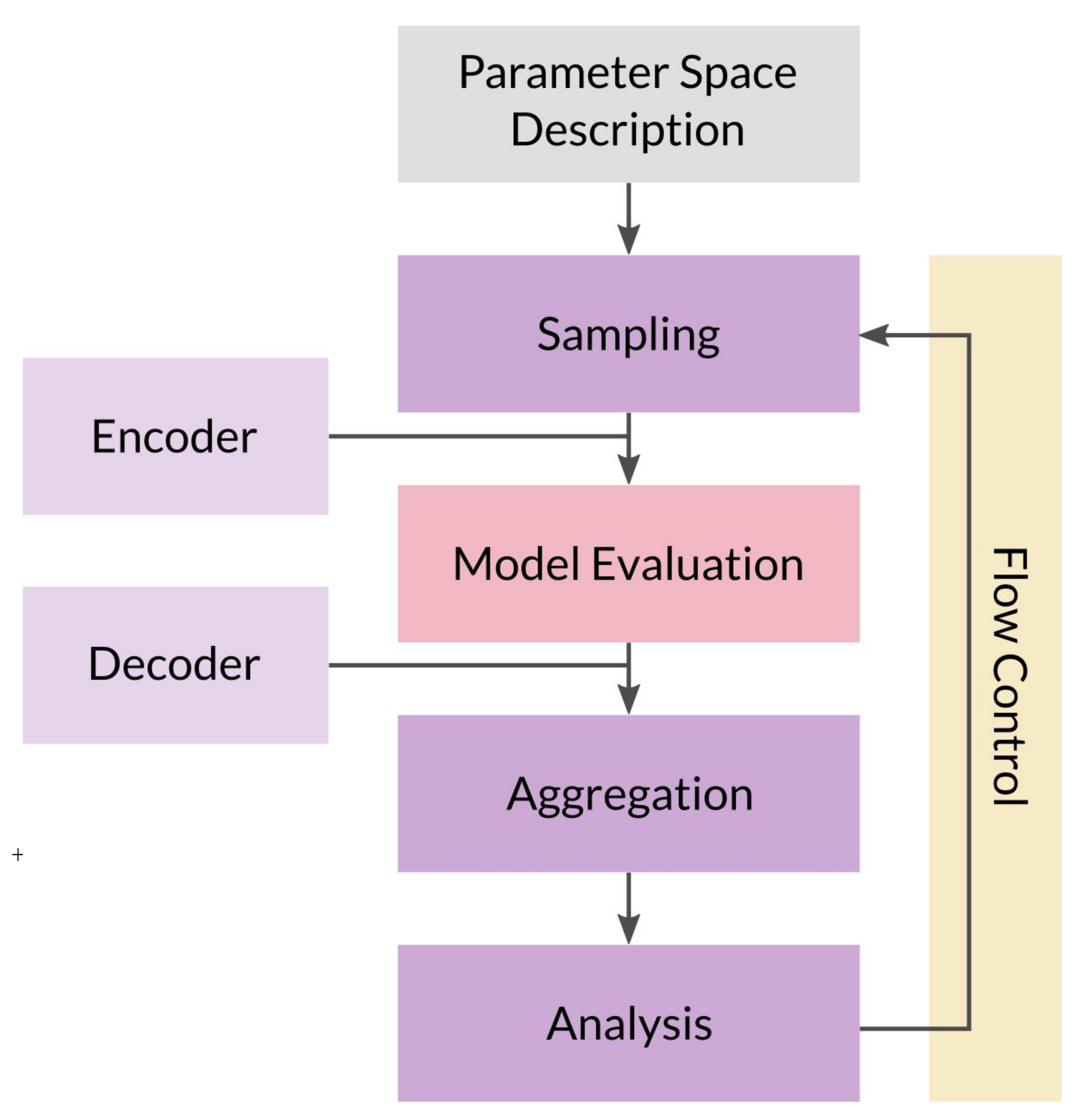
For two applications (migration and CFD[7]) we combined complex workflows, using automation, remote access and curation features from FabSim3[6], with VVUQ offered by EasyVVUQ. This integration can be made either by using the FabSim3 API in a script, or by incorporating EasyVVUQ in a FabSim3 plugin.

Typical scenario



*From the *Uncertainpy* documentation (uncertainpy.readthedocs.io).

Conceptual Model



Overall workflow

```
Campaign
my_campaign = uq.Campaign(name='cooling_cup')
Parameter Space Description
vary = {
    "kappa": cp.Uniform(0.025, 0.075),
    "t_env": cp.Uniform(15, 25)
}

Encoder
encoder = uq.encoders.GenericEncoder(
    template_fname='tutorial_files/cooling.template',
    delimiter='$',
    target_filename='cooling_in.json')

Decoder
decoder = uq.decoders.SimpleCSV(target_filename="output.csv",
                                output_columns=["te"],
                                header=0)
Sampling
my_sampler = uq.sampling.PCESampler(vary=vary, polynomial_order=3)
my_campaign.draw_samples()

Model evaluation
my_campaign.apply_for_each_run_dir(uq.actions.ExecuteLocal(app))

Aggregation
my_campaign.collate()

Analysis
analysis = uq.analysis.PCEAnalysis(sampler=my_sampler, qoi_cols=output_columns)

Descriptive Statistics
results = my_campaign.get_last_analysis()
```

```
Dask JobQueue

my_campaign = uq.CampaignDask(name='coffee_pce')
cluster = SLURMCluster(job_extra=['--cluster=mycluster'],
                       queue='myqueue',
                       cores=48, memory='64 GB')
cluster.scale(96)
client = Client(cluster)
```

OR

```
QCG-PJ
qcgpjexec = Executor()
qcgpjexec.create_manager(dir=my_campaign.campaign_dir)
qcgpjexec.add_task(Task(
        TaskType.EXECUTION,
        TaskRequirements(cores=Resources(exact=1)),
        application=app
))
qcgpjexec.run(
        campaign=my_campaign,
        submit_order=SubmitOrder.EXEC_ONLY)
qcgpjexec.terminate_manager()
```

Partners

References & Acknowledgements

[1] Derek Groen et al. "Introducing VECMAtk-Verification, Validation and Uncertainty Quantification for Multiscale and HPC Simulations". International Conference on Computational Science. Springer, Cham, 2019.
[2] Tomasz Piontek et al. "Development of science gateways using qcg — lessons learned from the deployment on large scale distributed and hpc infrastructures". Journal of Grid Computing 14(4) (Dec 2016) 559–573.
[3] QCG PilotJob https://github.com/vecma-project/QCG-PilotJob
[4] EasyVVUQ-QCGPJ https://github.com/vecma-project/EasyVVUQ-QCGPJ
[5] Dask JobQueue https://github.com/dask/dask-jobqueue
[6] FabSim3 https://github.com/djgroen/FabSim3
[7] VECMAtk application tutorials https://www.vecma-toolkit.eu/tutorials/

We acknowledge funding support from the European Union's Horizon 2020 research and innovation programme under grant agreement 800925 (VECMA project, www.vecma.eu).