H2020-EINFRA-2017

EINFRA-21-2017 - Platform-driven e-infrastructure innovation

DARE [777413] "Delivering Agile Research Excellence on European e-Infrastructures"

D3.5 - DARE API I

Project Reference No	777413 — DARE — H2020-EINFRA-2017 / EINFRA-21-2017
Deliverable	D-3.5 DARE API I M12
Work package	WP3: Large-scale Lineage and Process Management
Tasks involved	T3.3: User-driven Control enabling API
Type	DEM: Demonstrator, pilot, prototype
Dissemination Level	PU = Public
Due Date	31/12/2018
Submission Date	31/12/2018
Status	Draft
Editor(s)	Antonia Tsili (NCSRD), Iraklis Klampanos (NCSRD), Alessandro Spinuso (KNMI)
Contributor(s)	Federica Magnoni (INGV), Amy Krause (UEDIN), Rosa Filgueira (UEDIN), Xavier Pivan (CERFACS), Christian Pagé (CERFACS), Antonis Koukourikos (NCSRD)
Reviewer(s)	Malcolm Atkinson (UEDIN)

Document description	Implementation of the DARE software API, ready to be introduced in the integrated DARE software stack

Document Revision History

Version	Date	Modifications Introduced	
		Modification Reason	Modified by
1	21/11/2018	Initial version	Antonia Tsili (NCSRD)
2	23/11/2018	PE registry information and documentation annex	Iraklis Klampanos (NCSRD)
3	30/11/2018	Initial version of generic PEs	Amy Krause and Rosa Filgueira (UEDIN)
4	3/12/2018	ENES PEs	Xavier Pivan (CERFACS)
5	3/12/2018	EPOS PEs	Federica Magnoni (INGV)
6	6/12/2018	Components and Datasets registries specification	Antonia Tsili (NCSRD) and Antonis Koukourikos (NCSRD)
7	10/12/2018	Minor corrections on types and descriptions of PEs	Antonia Tsili (NCSRD)
8	28/12/2018	Internal review	Malcolm Atkinson (UEDIN)
9	31/12/2018	Finalisation	Iraklis Klampanos (NCSRD)

Executive Summary

This reports on the current version of DARE platform's software and services APIs, collectively referred to as the DARE API. Programmatic access to DARE services is offered via two discrete channels: a library of dispel4py processing elements (PEs) and via accessing RESTful and semantic catalogues. In accordance with the Architecture principles of D2.1, the DARE API progresses towards achieving a Common Conceptual Core Catalogue via federating over existing as well as new catalogues. The Datasets and PEs parts of the catalogue contain elements directly associated with the IS-ENES and EPOS use-cases, in line with WP7 and WP6 requirements.

Table of Contents

List of Terms and Abbreviations

Abbreviation	Definition
REST	Representational State Transfer
API	Application Program Interface
EPOS	European Plate Observing System
ESGF	Earth System Grid Federation
IS-ENES	Infrastructure for the European Network for Earth System
EOSC	European Open Science Cloud

1 Introduction

This reports on the current version of DARE platform's software and services APIs, collectively referred to as the DARE API. Programmatic access to DARE services is offered via two discrete channels: a library of dispel4py processing elements (PEs) and via accessing RESTful and semantic catalogues. These programmatic services cover many aspects of the platform. The dispel4py interface makes available the PEs catalogue, which will enable the development of domain-specific user-facing tools and scientific experiments. Other catalogues offer programmatic access to data provenance records, the software components DARE has access to, known datasets, etc.

1.1 Approach and Relationship with other Work Packages and Deliverables

In accordance to the Architecture principles of D2.1, the DARE API progresses towards achieving a Common Conceptual Core Catalogue via federating over existing as well as new catalogues. The API of the platform in progress introduces users to an interface through which they can access a *Thing* as expressed in D2.1 representing any component stored in the the overall logical catalogue. The constituent registries (or catalogues), namely the dispel4py PEs registry, the Software components catalogue and the Datasets catalogue are being integrated by WP4. The Datasets and PEs parts of the catalogue contain elements for the implementation of the IS-ENES[1] and EPOS[2] use-cases, in line with WP7 and WP6 requirements.

1.2 Methodology and Structure of the Deliverable

This Deliverable covers the two points of programmatic interfacing offered by DARE: the dispel4py interface and the programmatic access of the DARE catalogues. The dispel4py API includes generic as well as domain-specific PEs, aligned with the IS-ENES and EPOS requirements. Moreover, the connection of dispel4py interface and data provenance is explained. In addition to dispel4py, we provide the APIs to the rest of the DARE catalogues, as they are currently implemented. These catalogues include the PEs registry, the Software components catalogue as well as the Datasets catalogue. Programmatic access to provenance records is also provided the via ProvenenceType, which is also presented below.

2 dispel4py interface

The DARE platform aims to ease the orchestration of persistent computational services and the establishment of high-throughput data channels between them. This is achieved by creating stream-based workflows and establishing connection interfaces between the main operators through which data is either consumed or forwarded to output, called processing elements (PEs). In DARE, these PEs

[1] IS-ENES

[2] EPOS

are described in the PEs registry. Below we provide the PEs that form the programmatic interface to DARE's dispel4py, organised in generic ones and in domain-specific. The PEs represent high-level operations abstractly, so that multiple implementations and mappings are possible. The data flow between PEs consists of sequences of data units. This permits optimisation such as direct data transfer, pipelining and parallelisation over varying and heterogeneous e-Infrastructures.

In the API descriptions below "a" denotes an abstract type of input or output data as a single stream, while "[a]" is the same type of data distributed in a vector of streams.

2.1 Generic PEs

In the following tables we declare PEs used in generic context that are separated according to their functionality.

PEs that define various mechanisms for splitting the sequence of data units on an incoming stream into a sequence of data units distributed across a vector of output streams.

Package	Name	Description	Input Connections	Output Connections
generic	split	The incoming stream is distributed across the outputs in any manner the implementation of dispel4py chooses. This will normally be round robin or random but the encoded scientific method does not require a specific distribution.	a	[a]
generic	split_round_robin	The first arriving data unit is despatched to outputs[0] and the next to outputs[1], and so on until a data unit is despatched to outputs[n-1], where there are n output streams, and then the despatches sweep from outputs[0] onwards again.	a	[a]
generic	split_random	The implementation uses a random number generator to distribute data units over the n output streams, with an approximately	a	[a]

		equal number being despatched on each stream. A seed may be supplied to achieve repeatability during debugging.		
generic	split_by_group	The incoming data units are assigned to output streams according to the group in which they fall, c.f. the group by operation of relational algebra. An attribute of the data unit or a discriminating function operating on the data unit as a whole needs to be specified to identify the group. Normally there is one group per value that occurs in the attribute, or is yielded by the function. However, the range of permitted values may be specified. In which case the data units that yield a value not in the range are sent to one of two reject streams, aboveTop or belowBottom, to distingush two kinds of failure.	a	[a]
generic	split_by_bucket	This is as split by group, but a sequence of ranges for the value yielded from each data unit are also supplied in the order of the output streams. If the yielded value falls into the first range the data unit is despatched on outputs[0], for the second range on outputs[1] and so on. If the yielded value does not	a	[a]

		match any range the data unit is despatched on one of two reject streams, aboveTop or belowBottom, to distingush two kinds of failure.		
generic	split_by_demand	This attempts to balance the quantities on each output stream with the capacity of the processing capacity down stream. It starts like split_round_robin but if the output streams buffers are filling, then it despatches more data units to those that have fewer data units unpro-cessed in their output buffers. This requires a mechanism for a PE to interrogate the state of the output buffers.	a	[a]

PEs that define various mechanisms for merging the sequences of data units on incoming streams into a sequence of data units on one output stream.

Package	Name	Description	Input Connections	Output Connections
generic	merge	The incoming streams are combined to form one output stream in any manner the implementation of dispel4py chooses. This will normally be round robin or on arrival but the encoded scientific method does not require a specific order.	[a]	a

	merge_round_ robin	Take the first data unit from inputs[0], the next from inputs[1] and so on, wrapping around to inputs[0] after taking a data unit from inputs[n-1]. This will preserve order if used in conjunction with split_round_robin provided the intermediate processing has not removed or inserted data units or changed the stream order.	[a]	a
generic	merge_on_ arrival	Take the first data unit from the first stream that delivers one, and continue roughly in arrival order. This avoids delays awaiting the slowest path, but the order of data units has no logical basis.	[a]	a
generic	merge_sorted	This is provided with an extra parameter, e.g. a function or a conditional expression to map to a function, that takes two data units and yields true if the first argument should precede the second. It assumes that each input stream is already ordered according to that predicate. It then waits until every stream that will produce a data unit has produced one, and sends the first data unit according to that ordering to the output. That data unit will be replaced by its stream. The process continues until all	[a]	a

		of the input streams have signalled ’end_of_stream’. Note that this can only be used with continuous streams if the data items are intrinsically ordered, e.g. by timestamp, otherwise the stream needs to be divided into batches for sorting before merge_sorted.		
generic	merge_sorted_ distinct	This is logically equivalent to merge_sorted except that data units that have equal value under a supplied or default equality test are represented by their first arrival, the remainder are discarded.	[a]	a
generic	merge_ concatenated	This takes all of the data units on inputs[0] and sends them to output. When that stream yields an ’end_of_stream’ signal it then copies all of the data units in inputs[1] in their arrival order and so on. When it has copied inputs[n-1] to output it sends its ’end_of_stream’ signal. Note that this means that each subgraph feeding one of the inputs can be activated in succession and terminated when it has completed.	[a]	a

PEs that define mechanisms for chopping sequences of data units on an input stream into a sequence of batches of data units on an output stream and to do the inverse.

Package	Name	Description	Input Connections	Output Connections
generic	batch	The input stream is partitioned into a sequence of batches of length m where m is supplied as a parameter. The batches will be emitted on output with batch start and end markers so that it is not necessary to hold a whole batch in memory. The last batch will normally hold fewer than m items.	a	a
generic	unbatch	This removes the outer level of batch markers, effectively concatenating the batches on input to one sequence of data units on output.	a	a

2.2 Provenance Type

A workflow is a program that combines atomic and independent processing elements (PEs) via a specification language and a library of components. More advanced systems adopt abstractions to facilitate re-use of workflows across users' contexts and application domains. While methods can be multidisciplinary, provenance should be meaningful to the domain adopting them. Therefore, dispel4py offers portable specification of the mechanisms allowing the contextualisation of the provenance produced. For instance, users may want to extract domain-metadata from a component or groups of components adopting vocabularies that match their domain and current research, or tune the granularity and precision of recording input to output dependencies. To allow this in dispel4py, we introduce the concept of **ProvenanceType**. Types may be developed as **Pattern Type** and **Contextual Type** to represent complex computational patterns and i/o dependencies respectively, and to capture specific metadata contextualisations associated with the produced output data. Detailed documentation of the **ProvenanceType** class and its methods is provided in the DARE project's gitLab[3].

[3] https://gitlab.com/project-dare/dispel4py/blob/master/provenance_documentation.md

The same page also reports the description of a preliminary set of already implemented patterns types that capture the lineage of processing elements, which may present different kinds of stateless and stateful behaviours. The rationale for these choices is described with more detail in the DARE Architecture and technical positioning deliverable (D 2.1), where we also introduce the need to model additional provenance patterns. These aim at capturing interactive behaviours of the research-developers when they are developing, classifying and invoking their methods during research sessions targeting a specific task. The provenance patterns will be shaped by the forthcoming refinement and implementation of the DARE API, in cooperation with WP2 and WP4.

2.3 Provenance Configuration

To enable the user of a data-intensive application to configure the attribution of provenance types, domain's semantics and provenance-driven operations, we introduce in the framework the concept of *Provenance Configuration*. With the configuration users can specify several properties, such as user's attribution, provenance types, provenance clusters, provenance sensors (experimental setups that allow PEs to read and process provenance data within the workflow itself to trigger steering actions), selectivity and transfer rules. The configuration is used at the time of the initialisation of the workflow to prepare its provenance-aware execution. We consider that a chosen configuration may be influenced by personal and community preferences, as well as by rules introduced by institutional policies. For instance, a Research Infrastructure (RI) may indicate best practices to reproduce and describe the operations performed by the users exploiting its facilities, or even impose requirements which may turn into quality assessment metrics. This could require users to choose among a set of contextualisation types, in order to adhere to the infrastructure's metadata portfolio. Thus, a provenance configuration profile plays in favour of more generality, encouraging the implementation and the re-use of fundamental methods across disciplines, taking into account the relevance of the provenance information to the domain and to the stage method development has reached. Aspects concerning the contextualisation of the lineage are developed to reflect architectural principles (D 2.1). We foresee that many properties of the configuration will be inferred from the DARE catalogue by analysing the user's profile and the characteristics of the PE. This would foster the automatic setup of the provenance profile or the provision of targeted recommendations, leaving a user or research-developer in overall control.
The API method supporting configuration is reported as *configure_prov_run* in the project gitLab[4].

2.4 EPOS PEs

Package	Name	Description	Input Connections	Output Connections
EposPilot	code_sel	select the simulation code	list of available codes [<script>]	selected code <script>

[4] https://gitlab.com/project-dare/dispel4py/blob/master/provenance_documentation.md

EposPilot.spec	code_par	set the parameter file of the code	python dictionary for the values of the parameters {<parameter>:<value>}	json and ascii file for the code parameters (file.json,file.txt)
EposPilot.spec	mesh_sel	select a mesh for the simulation	list of available meshes [<mesh>]	ascii files for the selected mesh (file.txt)
EposPilot.spec	model_sel	select a seismic wave speed model	list of available wave speed models for the selected mesh [<wsm>]	ascii file for the selected wave speed model (file.txt)
EposPilot.spec	eqk_sel	set the input file of the seismic source based on the selected earthquake	python dictionary for the values of the source parameters {<parameter>:<value>}	ascii file for the selected earthquake source (file.txt)
EposPilot.spec	stat_sel	set the input file of the seismic stations based on the selected receivers	python dictionary for the values of the station parameters {<parameter>:<value>}	ascii file for the selected seismic stations (file.txt)
EposPilot	launch	launch the simulation		
EposPilot.spec	decompose	launch the routine of the code that decomposes the mesh into chunks for each computing processor; then collect the output	ascii files of the mesh and of the code parameters (file.txt)	binary files of the mesh partitioning (binary_file)
EposPilot.spec	generate	launch the routine of the code that generates the specfem databases; then collect the output	binary files of the mesh partitioning and ascii file of the code parameters (binary_file)	binary files describing the wave speed model interpolated at the mesh points

				(binary_file)
EposPilot .spec	solver	launch the routine of the code that runs the solver; then collect the output	binary files of the wave speed model at the mesh points, ascii files of the code parameters, of the earthquake source and of the seismic stations (file.txt,binary_file)	ascii files of the seismogram; binary files of the wave propagation movies (file.txt,binary_file)
EposPilot	transf_sy	transform the ascii seismograms into seed files and png images	ascii files of the seismogram (file.txt)	seed and png files for the seismogram of each component at each seismic station (seed_file,file.png)
EposPilot .spec	gen_mov	generate wave propagation movies	binary files of the wave propagation movies (binary_file)	file of the wave propagation movie (so far in .avi format) (file.avi)
EposPilot .spec	gen_kmz	generate kmz files to show simulation results	png files for the seismograms; values of the seismic source parameters (file.png)	kmz file showing the output seismograms on a map (file.kmz)
EposPilot .spec	store_simul	store simulation output and visualization with related metadata	ascii, seed and png files of the seismograms; binary files of the movie; kmz file; related metadata (file.txt,seed_file,file.png)	
EposPilot	dwl_ob	download observed data in a given time range	python dictionary for the values of the earthquake origin	seed files of the observed seismograms

			time {<eot>:<value>}	(seed_file)
EposPilot	transf_ob	transform the seed seismograms into png images	seed files of the observed seismograms	png files for the seismogram of each component at each seismic station (file.png)
EposPilot	store_ob	store observed data with related metadata	seed and png files of the observed seismograms and related metadata (seed_file,file.png)	
EposPilot	process_data	pre-process observed and synthetic seismograms and produce corresponding pictures	seed files of raw observed and synthetic seismograms (seed_file)	seed and png files of processed observed and synthetic seismograms (seed_file,file.png)
EposPilot.ra	pgm_par	calculate ground motion parameters from both observed and synthetic seismograms	seed files of observed and synthetic seismograms (raw or processed) (seed_file)	json files for ground motion parameters (file.json)
EposPilot.ra	pgm_comp	compare observed and synthetic ground motion parameters	json files with ground motion parameters both observed and synthetic (file.json)	json file for results of misfit procedures (file.json)
EposPilot.ra	pgm_plot	plot maps of peak ground motion from synthetics or observed data (or a combination of them)	json files with ground motion parameters both observed and synthetic (file.json)	png files of ground motion maps (file.png)

EposPilot.ra	store_pgm	store observed and synthetic ground motion parameters, visualizations, results of comparisons, related metadata	json files with ground motion parameters; json files with results of ground motion misfit procedures; png files of ground motion maps (file.json)	

2.5 ENES PEs

Package	Name	Description	Input Connections	Output Connections
ENES usecase	NetCDFProcessing	wrapper for icclim package. Perform climate indice calculation	python dictionary with the requested parameters containing all the necessary input for icclim.indice function {<parameter>:<value>}	parameters from python dictionary, name and location of netcdf produced and the indice name {[<parameter>],<name>,<location>,<index>}
ENES usecase	StreamProducer	process the json files to generate new input such as the name of intermediate netcdf	json files from user requests from climate4impact Portal (file.json)	python dictionary formatted to fit the PE in ENES usecase package {<parameter>:<value>}
ENES usecase	NetCDF2xarray	return an xarray from a netcdf file	python dictionary with file location or URL of netcdf file and the variable name {<parameter>:<value>}	xarray of the netcdf file {xarray}
ENES usecase	ReadNetCDF	read a netcdf file or URL and return	python dictionary with file or URL	return the time vector and the

		a numpy array for the time and the variable from netcdf file	location of netcdf file {<parameter>:<value>}	variable matrix {[<time>],[<variable>]}
ENES usecase	StandardDeviation	perform standard deviation on a 3D array	3D array streamed from previous PE	return a time serie of the spatial standard deviation from input array
ENES usecase	AverageData	calculate the spatial average on 3D array	3D array streamed from previous PE	return a time serie of the spatial average from input array
ENES usecase	CombineData	Combine streamed data together to perform a calculation overall these data together	Several data streamed from previous PE	stream the data for one PE
ENES usecsase	ReturnPlot	Create a figure for time serie vector	time series of climate variable streamed	png files with the time series of climate variable (file.png)

3 Interfaces to DARE Registries

3.1 dispel4py PE Registry

The dispel4py PEs registry is exposed via a RESTful API adhering to the OpenAPI Specification and Swagger tools, which offer a user-friendly and self-documenting interface. Through that interface, one can find a set of resources, which include connections, function implementations, function parameters, groups, literals, PE implementations, user groups, users and workspaces, and a list of possible actions to apply on the resources. It is documented and all available methods are provided with a simple and useful description. A top-level view of provided methods is provided as Appendix A.

3.2 Components Registry

The Components Registry interface connects DARE's software parts with their semantic interpretation, which allows for facilitated indexing and searching. Each software component is documented according

to a custom schema describing classes and relations that includes any detail essential to software deployment in a containerised environment.

Container Technology

One aspect of the platform's action regarding implementation techniques is the usage of container technology. DARE is meant to include a diversity of software and physical resources that need to report on operational details in order to make software distribution possible. Container deployment offers this kind of potential providing the platform with uniformly shaped reports on operation.

Schema

We formed a schema describing the main characteristics of each software component, which serves the purpose of developing that part of a software components' catalogue and potentially facilitating the platform's contribution to processes' and experiments' optimisation potential regarding data load and infrastructure. The idea behind such is the formation of an OWL-ontology based on information gained through the inspection of the component's operation so that each software component can be stored and retrieved via a Virtuoso instance.

A view of the schema concerning the software components' part f the catalogue can be found in Appendix B.

3.3 Data Registry

Data included on DARE platform are currently intended to cover climate and seismic phenomena provided by the two collaborating pilots[5] of the project. We aim to provide support for datasets in various forms and locations that are also protected by different licences and on which each person or group has different rights. In order to store and organise such data loads, another custom schema was created with an OWL-ontology describing each dataset in a high but also more low-level manner. That way, the data part of the catalogue offers the ability to keep track of the available datasets and any changes performed on them.

Schema

The schema describing the dataset entries is formed with reference to DCAT[6] Vocabulary, which proved extremely helpful in including all the information needed for keeping track of a dataset's versions and states, as well as ease of access and collection. Each instance is associated with a software component in order to render a user or a service capable of acquiring the needed data. The aforementioned schema follows the concept that was adopted for the Software Components' one.

A view of the schema concerning the software components' part f the catalogue can be found in Appendix B.

[5] EPOS, ENES

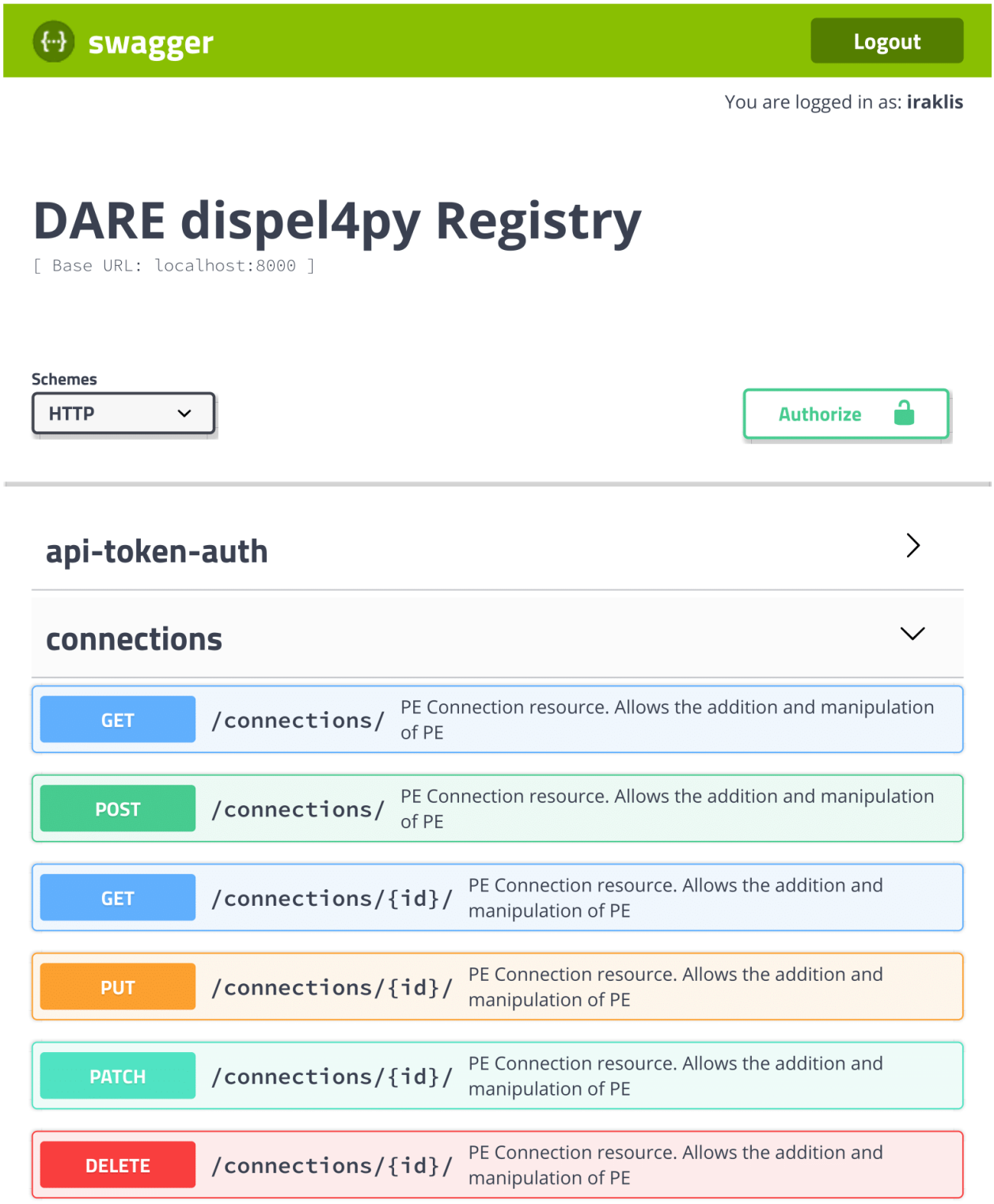
[6] https://gitlab.com/project-dare/dispel4py/blob/master/provenance_documentation.md

4 Conclusions

DARE platform aims to provide user-friendly tools to scientists whose fields and interests do not include programming but also need to manipulate large distributed data and remote systems. The platform's API's role is to instantiate the aforementioned user-friendly interface between user and a catalogue that encapsulates a diversity of data, including experimental data, custom scripts and available software tools. At this moment, our work is focused on the fields that are covered by WP2 and WP4 and in collaboration with EPOS and IS-ENES projects. The current deliverable describes the first step to create this type of catalogue with its parts discriminated. We intend to minimise exposed differences of access to data from different sources in future iterations.

APPENDIX

A Top level of the PE Registry API documentation


swagger
Logout
You are logged in as: iraklis
DARE dispel4py Registry
[Base URL: localhost:8000]
Schemes
HTTP
Authorize
api-token-auth
connections
GET /connections/ PE Connection resource. Allows the addition and manipulation of PE
POST /connections/ PE Connection resource. Allows the addition and manipulation of PE
GET /connections/{id}/ PE Connection resource. Allows the addition and manipulation of PE
PUT /connections/{id}/ PE Connection resource. Allows the addition and manipulation of PE
PATCH /connections/{id}/ PE Connection resource. Allows the addition and manipulation of PE
DELETE /connections/{id}/ PE Connection resource. Allows the addition and manipulation of PE

pes

GET	/pes/	PE resource. Allows addition and manipulation of dispel4py processing
POST	/pes/	PE resource. Allows addition and manipulation of dispel4py processing
GET	/pes/{id}/	Retrieves an item by id.
PUT	/pes/{id}/	PE resource. Allows addition and manipulation of dispel4py processing
PATCH	/pes/{id}/	PE resource. Allows addition and manipulation of dispel4py processing
DELETE	/pes/{id}/	PE resource. Allows addition and manipulation of dispel4py processing

registryusergroups

GET	/registryusergroups/	Registry user group resource.
POST	/registryusergroups/	Registry user group resource.
GET	/registryusergroups/{id}/	Registry user group resource.
PUT	/registryusergroups/{id}/	Registry user group resource.
PATCH	/registryusergroups/{id}/	Registry user group resource.

literals

GET	/literals/	Literal entities resource.
POST	/literals/	Literal entities resource.
GET	/literals/{id}/	Retrieves an item by id.
PUT	/literals/{id}/	Literal entities resource.
PATCH	/literals/{id}/	Literal entities resource.
DELETE	/literals/{id}/	Literal entities resource.

peimpls

GET	/peimpls/	PE Implementation resource. Allows the creation and manipulation of
POST	/peimpls/	PE Implementation resource. Allows the creation and manipulation of
GET	/peimpls/{id}/	PE Implementation resource. Allows the creation and manipulation of
PUT	/peimpls/{id}/	PE Implementation resource. Allows the creation and manipulation of
PATCH	/peimpls/{id}/	PE Implementation resource. Allows the creation and manipulation of
DELETE	/peimpls/{id}/	PE Implementation resource. Allows the creation and manipulation of

functions

GET	/functions/	Function resource. Allows addition and manipulation of dispel4py
POST	/functions/	Function resource. Allows addition and manipulation of dispel4py
GET	/functions/{id}/	Retrieves an item by id.
PUT	/functions/{id}/	Function resource. Allows addition and manipulation of dispel4py
PATCH	/functions/{id}/	Function resource. Allows addition and manipulation of dispel4py
DELETE	/functions/{id}/	Function resource. Allows addition and manipulation of dispel4py

groups

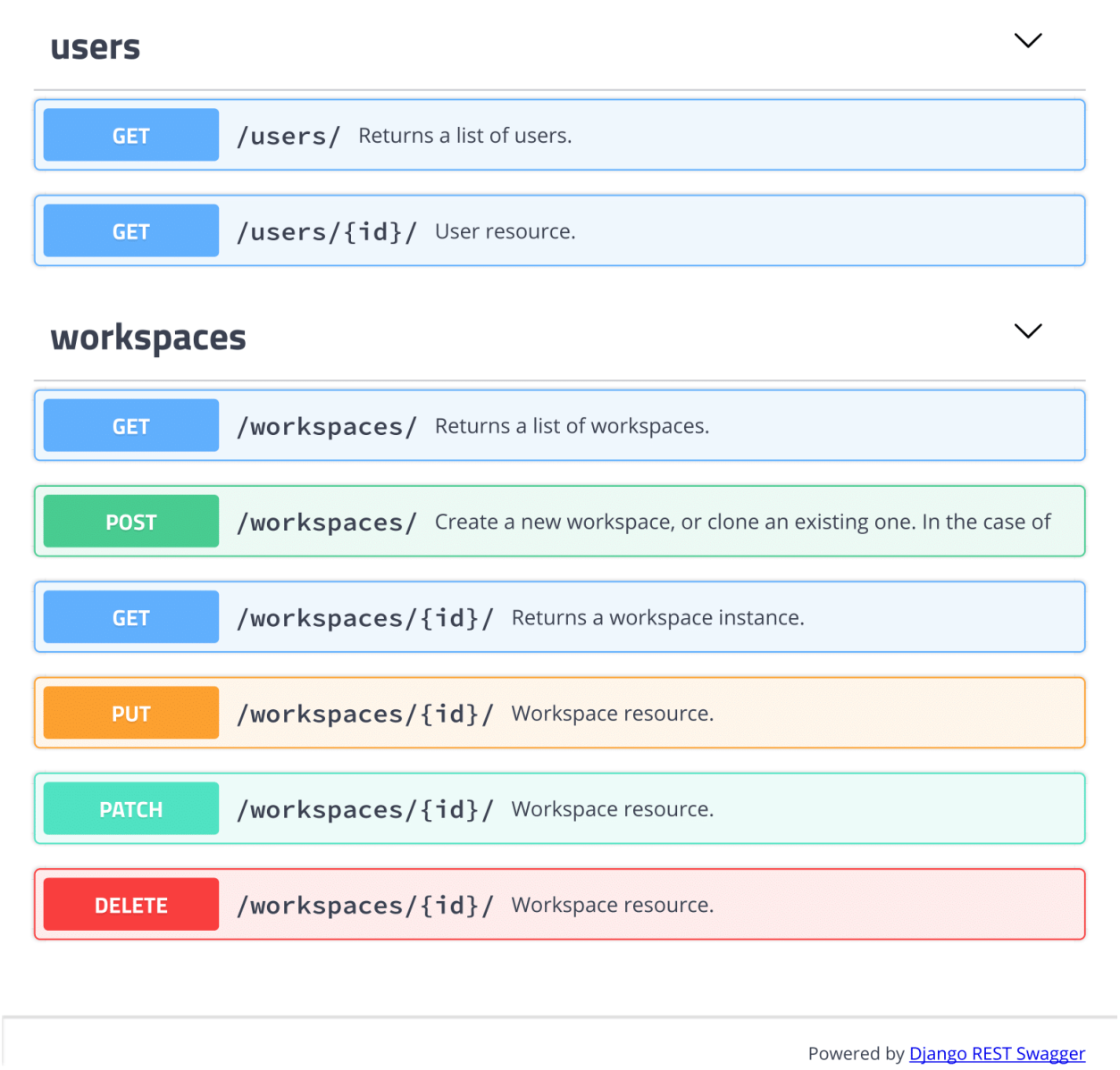
GET	/groups/	Basic user group resource.
POST	/groups/	Basic user group resource.
GET	/groups/{id}/	Basic user group resource.
PUT	/groups/{id}/	Basic user group resource.
PATCH	/groups/{id}/	Basic user group resource.
DELETE	/groups/{id}/	Basic user group resource.

fnimpls

GET	/fnimpls/	Function implementation resource. Allows the creation and manipulation
POST	/fnimpls/	Function implementation resource. Allows the creation and manipulation
GET	/fnimpls/{id}/	Function implementation resource. Allows the creation and manipulation
PUT	/fnimpls/{id}/	Function implementation resource. Allows the creation and manipulation
PATCH	/fnimpls/{id}/	Function implementation resource. Allows the creation and manipulation
DELETE	/fnimpls/{id}/	Function implementation resource. Allows the creation and manipulation

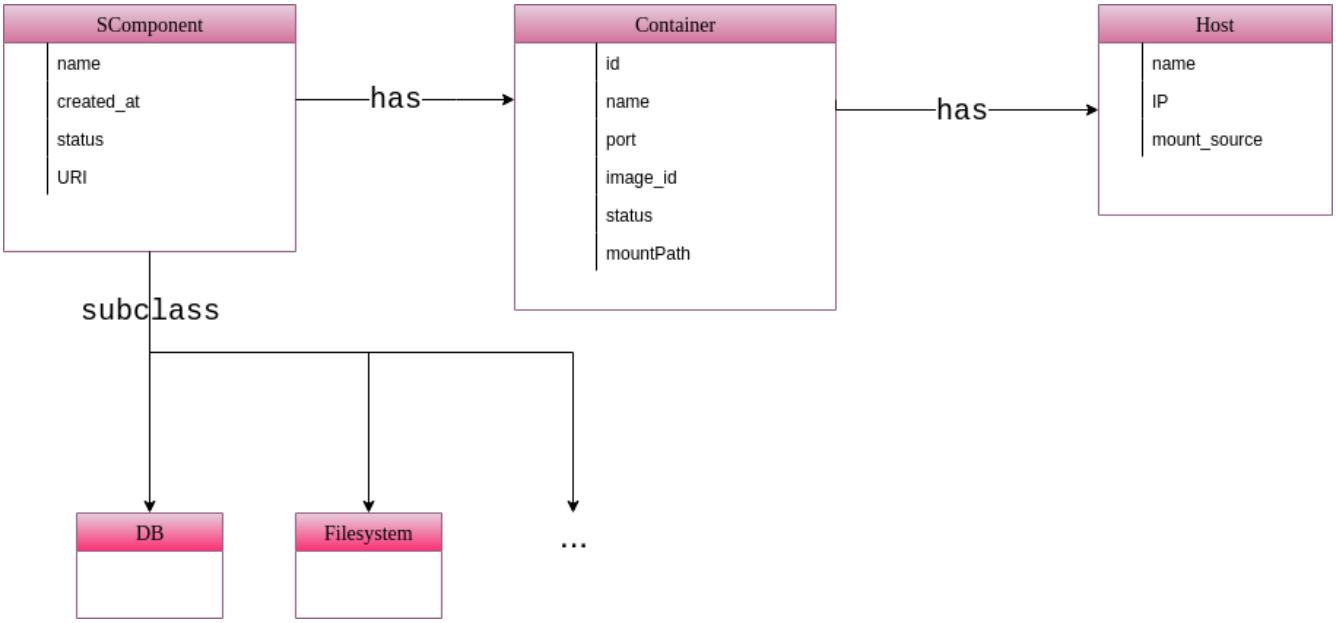
fnparams

GET	/fnparams/	Function parameter resource. Allows the addition and manipulation
POST	/fnparams/	Function parameter resource. Allows the addition and manipulation
GET	/fnparams/{id}/	Function parameter resource. Allows the addition and manipulation
PUT	/fnparams/{id}/	Function parameter resource. Allows the addition and manipulation
PATCH	/fnparams/{id}/	Function parameter resource. Allows the addition and manipulation
DELETE	/fnparams/{id}/	Function parameter resource. Allows the addition and manipulation



Powered by Django REST Swagger

B Schema of Software Components Catalogue



C Schema of Datasets Catalogue