H2020-EINFRA-2017

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

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

ID2.2-M25: DARE Architecture and Technology internal report

Project Reference No	777413 — DARE — H2020-EINFRA-2017 / EINFRA-21-2017
Internal Deliverable	ID2.2-M25: DARE Architecture and Technology internal report
Work package	WP2: Architecture Specification and Innovation
Tasks involved	T2.1: Architecture Specification, Tools and Components
Type	R: Internal Document, report
Dissemination Level	P = Public
Due Date	17/02/2020
Publication Date	25/02/2020
Status	Completed and approved for release
Editor(s)	Malcolm Atkinson (UEDIN) Iraklis Klampanos (NCSRD)

Contributors	Malcolm Atkinson (UEDIN) Rosa Filgueira (UEDIN) André Gemünd (FRAUNHOFER) Vangelis Karkaletsis (NCSRD) Iraklis Klampanos (NCSRD) Antonis Koukourikos (NCSRD) Amélie Levray (UEDIN) Mike Lindner (KIT) Federica Magnoni (INGV) Christian Pagé (CERFACS) Andreas Rietbrock (KIT) Alessandro Spinuso (KNMI) Sissy Themeli (NCSRD) Xenofon Tsilimparis (GRNET) Fabian Wolf (FRAUNHOFER)
Reviewers	Andreas Rietbrock (KIT) Vangelis Karkaletsis (NCSRD)
Document description	Development of the DARE architecture since D2.1 and innovation planning to address the requirements of DARE user communities and to enhance sustainability.

Document Revision History

Version	Date	Change made	Contributor
1	10/12/2019	Initial draft	Malcolm Atkinson (UEDIN)
2	13/12/2019	Start of Section 3	Alessandro Spinuso (KNMI)
3	16/12/2019	Section 3.3	Rosa Filgueira (UEDIN)

4	16/12/2019	Section 2	Malcolm Atkinson (UEDIN)
5	19/12/2019	Start of Section 3.1	Federica Magnoni (INGV)
6	20/12/2019	Start of Section 3.2	Christian Pagé (CERFACS)
7	20/12/2019	Section 3.1	Federica Magnoni (INGV)
8	23/12/2019	Section 3.2	Christian Pagé (CERFACS)
9	02/01/2020	First draft Section 4.1	Sissy Themeli (NCSRD)
10	06/01/2020	Editorial review of Section 4	Malcolm Atkinson (UEDIN)
11	07/01/2020	Additions to Section 4.1.4	Rosa Filgueira (UEDIN)
12	07/01/2020	Started Section 4.2	Malcolm Atkinson (UEDIN)
13	08/01/2020	Section 5.2	André Gemünd (Fraunhofer)
14	09/01/2020	Comments in Section 3.1	Mike Lindner (KIT)
15	14/01/2020	Included in 4.2.4 Data Semantics Catalogue from F. Wolf (FRAUNHOFER)	Malcolm Atkinson (UEDIN)
16	17/01/2020	Section 4.2.4 Registry	Amélie Levray (UEDIN)
17	17/01/2020	Comments in Section 4.2.4	Alessandro Spinuso (KNMI)
18	27/01 to 07/02/2020	Extensive revisions responding to reviewers and Toulouse discussions	Malcolm Atkinson (UEDIN)
19	28/01/2020	Section 4.3 - P4 Tools	Alessandro Spinuso (KNMI)
20	10-14/02/2020	Polishing and consistency checking for consideration by DARE plenary 17/02/2020	Malcolm Atkinson (UEDIN)
21	17-24/02/2020	Adopting suggested corrections.	Malcolm Atkinson (UEDIN)

Executive Summary

DARE is an ambitious project that aims to provide novel approaches for creating and using data-powered methods at the frontiers of today's research and innovation. DARE's central goal is to support research developers – domain-expert software developers – to transparently make use of European e-infrastructures, research infrastructures and other platforms and software in order to create data- and computationally-intensive applications for their domains. DARE aims to achieve these goals by providing much needed technology and methodology aligned with EOSC developments.

This internal deliverable presents the progress since D2.1 and the new understanding of what is required from the DARE architecture, driven by the interplay of user requirements and technological opportunities. These are constrained and enhanced to yield future utility, extensibility and sustainability. New requirements for users to directly create and control complex computational and data challenges are pushing DARE to extend its integration, automation and optimisation. The initial plans for these are presented. They lead to a strategy for sustainability.

Table of Contents

1. Introduction

The DARE architecture has to pursue two goals:

1. Shape the DARE platform and its future versions to meet emerging and anticipated requirements and thereby improve user communities' research work, and
2. Identify frameworks and strategies that will be extensively used to improve return on investment and sustainability.

This requires a practical balance between delivering the required capabilities to the two DARE user communities and a longer-term vision. The architecture described in D2.1 [Atkinson *et al.* 2018] shaped the first platform release [Klampanos *et al.* 2019]. The overall structure interlinking three major subsystems, as shown in Figure 1.1 has proved successful and has been retained.

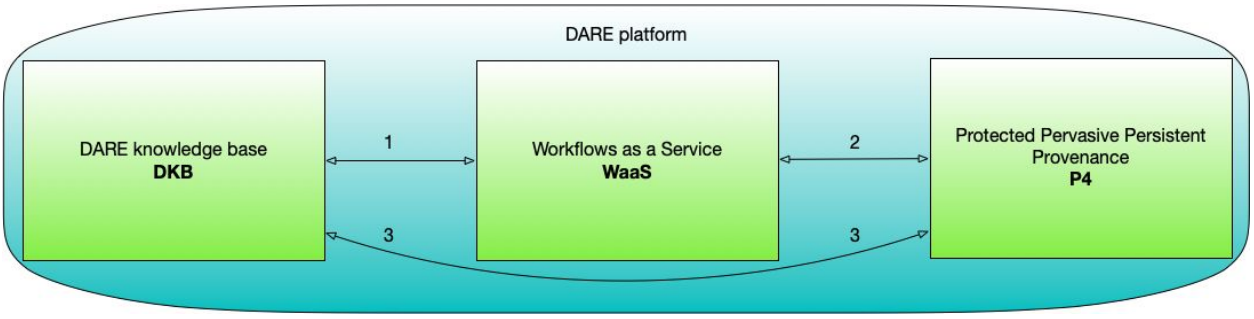

Figure 1.1: *The three principal subsystems forming the DARE platform.*

However, each technological pillar has been further developed, as outlined here and as reported in more detail in Section (**§**) 4.

1. A lightweight integration of the DARE Knowledge Base (**DKB**) [Atkinson & Levray 2019] makes it more easily used by the other subsystems and as a standalone service. It enables incremental introduction into established research environments and provides a foundation for increased abstraction, automation and stability co-existing with innovation. It will provide an API and a Python library to enable developers and methods to use it directly. It has two novel features: (a) research Contexts to manage scope within its information space, and (b) a Conceptual library to accelerate productive use and help with organising and interpreting the information. See §4.2 for more details.
2. The Workflows-as-a-Service (**WaaS**) has developed containerisation, orchestration and dynamic deployment of dispel4py data-streaming workflows to meet demanding user needs. It has include CWL[1] formalised workflows as part of that work, and to extend the range of methods facilitated. See §4.1 for details.
3. The provision and exploitation of provence via the Protected Pervasive Persistent Provenance (**P4**) subsystem has extended its scope, configurability and visualisation, as reported in [Spinuso *et al.* 2019]. It aims to deliver Reproducibility-as-a-Service (**RaaS**).

[1] W3C Common Workflow Language https://www.commonwl.org/

The adoption of provenance by scientists is being incentivised by more powerful provenance-driven tools. See §4.3 for details.

The combined and released platform is supporting challenging data-intensive and computationally demanding scientific methods and making them easy to deploy and use. For example, the seismic rapid assessment calculation and comparison of ground motion - see §3.1. The developments planned are motivated by the next stages of the seismological research and of computational integration for climate-impact modelling. These will then be generalised to accommodate additional communities.

The current implementation and plans for further developing it are presented in §4.4. The WaaS will be extended to handle more workflow/scripting languages and to employ optimisation when mapping dispel4py workflows onto production platforms, to improve scalability. The DKB functionality will be introduced to provide an extensible and flexible information sharing facility that should prove easy to use and thereby aid self sufficiency. The provenance handling provided by P4 will extend its collection capacity, support more complex queries against the stored usage history, and introduce additional provence-powered tools. This will incentivise increased adoption of provenance by application communities, a long-term step towards reproducibility and minable records of scientific procedures and progress. Offering convenient WaaS and good provenance-driven tools is a critical step in achieving reproducible science and accessible foundations for the evidence used in making major decisions.

Sustainability is now a key issue. The current progress and plans are presented in §5. Sustainability depends on establishing value and having that value recognised and on recruiting sufficient support to meet the costs of support. These should be minimised by careful engineering and by progressively empowering user communities to be self-sufficient by reducing the hurdles encountered and by simplifying, automating and eliminating tasks.

Section 6 draws together all these issues and proposes a way forward that continues to extend the DARE platform's capabilities, while improving self-sufficiency and sustainability. The sustainability strategy presented there (and in §5) is well aligned with the sustainability plans recently reported to the EC project officer.

A substantially revised version of this document will be published as a final deliverable in December 2020. Work related to this document will be ongoing throughout 2020 (see §6). Several publications developing aspects of its contents are anticipated. Consequently, we would very much appreciate criticisms, observations or advice pertinent to this work. Please email: Malcolm.Atkinson@ed.ac.uk or iaklampanos@iit.demokritos.gr and we will respond, take account of suggestions and acknowledge contributions.

2. Architecture overview

The DARE architecture should shape a framework that facilitates ambitious research undertaken by distributed, loosely federated multi-disciplinary communities typified by the solid-Earth and climate communities DARE works with. This imposes several requirements and constraints.

1. The work of developers and specialists should retain its value as digital technology evolves. This requires their work should be expressed precisely and abstractly so that it can be mapped (as far as possible automatically) to new digital infrastructures. This should accelerate advances as new power becomes available while minimising disruption and loss of methods and established practices. As reported in §3 and §4, DARE has already made significant progress towards this goal. Meeting it also facilitates deploying the DARE platform on a diversity of institutional computing services.
2. Multiple expert viewpoints co-exist, as illustrated in Figure 2.1. Their collaboration should be facilitated, e.g., between:
    a. Application *domain experts* who set goals, pioneer new research methods and organise teams, resources and campaigns.
    b. *Research developers* who draw on RSE products (see below), compose, package, steer and revise those elements to deliver tested contributions to their application's goals.
    c. *Research Software/Systems Engineers (RSEs)* who have specialist knowledge in some aspects of computer science, distributed systems engineering, simulation systems, data analytics, etc. They draw on theoretical and practical advances and develop subsystems, libraries, simulation codes, etc. for use by multiple application communities.
    d. *Resource providers* who establish and sustain computation, storage, information and other resources as services on which research communities depend.
3. The architecture has to be implementable, sustainable and affordable while meeting today's goals as rapidly as possible. At the same time, it has to deliver a good foundation on which to build support for future research goals exploiting emerging and specialised technologies. Keeping these immediate and longer-term considerations in balance is an architectural duty with a concomitant obligation to communicate with and gain buy-in from all of the stakeholders.

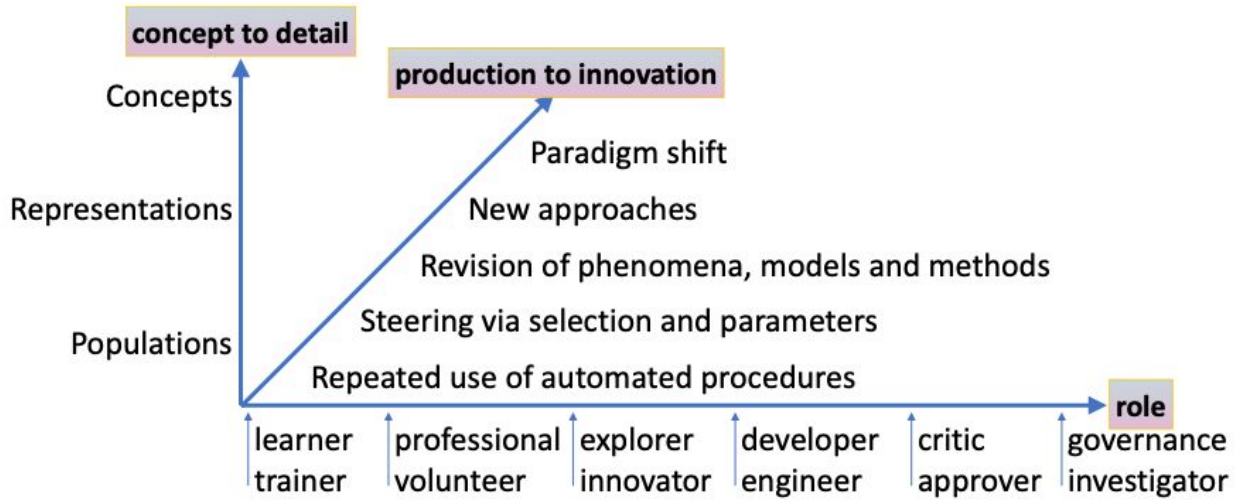

Figure 2.1: *Diverse roles are shown on the x axis. They may have inherently different viewpoints which change as their activities move in the other two dimensions. Research success depends on effective collaboration between these viewpoints while avoiding being slowed by attempting bring everything into a rigid consistent framework.*

The architecture therefore has to lead from the current implementation (§4) towards a composition of the following goals that matches the stakeholders' priorities.

1. *Delivering power and control to each user community*. Ideally, they should be able to immediately implement and use new methods exploiting all of the available computational power and the full richness of available data to address their most demanding and complex research challenges.
2. *Reducing the application communities' dependence on IT specialists*. A synergy between computational experts and domain experts will always be necessary to push some of the frontiers of research or to polish the optimisation of an intensively used method. However, most of the needed innovation will be achievable by the application communities themselves.
    a. Depending on others requires investment in explaining what is wanted and introduces delays and sometimes leads to divergence.
    b. It means that an application community takes longer to spot new opportunities.
3. *Achieving affordable long-term sustainability*. Ultimately, all software on which the application communities have to depend has to be maintained and supported[2]. As far as possible, the DARE platform should be built using standard components that are widely used and therefore their maintenance is amortised over an extensive community[3]. The remaining software which tailors and integrates existing software and services has to be

[2] Maintenance is approximately 90% of lifetime software costs. Open source doesn't remove this cost, it redistributes it.
[3] E.g., all environmental science R&D, a global alliance, all geo-spatial R&D, all data-intensive or computationally challenging R&D.

engineered with maintenance in mind[4]. This is equivalent to considering the operational and maintenance costs for a building. In the end, each application community has to meet its share of these costs, either by finding the resources, expert staff time or funds, or by persuading its funders to top-slice budgets to meet these common requirements.

Significant progress towards the first goal has been made in DARE, see §3 and [Atkinson *et al.* 2019, Klampanos *et al.* 2019, Pagé *et al.* 2019a, Spinuso *et al.* 2019 & Magnoni *et al.* 2019a].

Figure 2.2 shows the ideal state when this goal is reached. Application teams incrementally build and test complex methods. When these are judged ready by the application experts themselves, they can move their work to production. There it can be repeatedly used, on specified targets, with steering of diagnostics, provenance collection, data handling and parameter revision when the practitioners wish. Otherwise these default to community agreed standards. When improvements are identified the application team can implement them and deploy the improved version. Work is still required to generically support this goal and to fully automate and optimise production. This includes tools to make this easier for application teams and to reduce the need to master technical detail. Both of which will help with system mobility and address the users' requirement to be able to instal and run DARE on their local, community or national computational services.

[4] The Software Sustainability Institute, https://www.software.ac.uk/, develops and promotes the required standards.

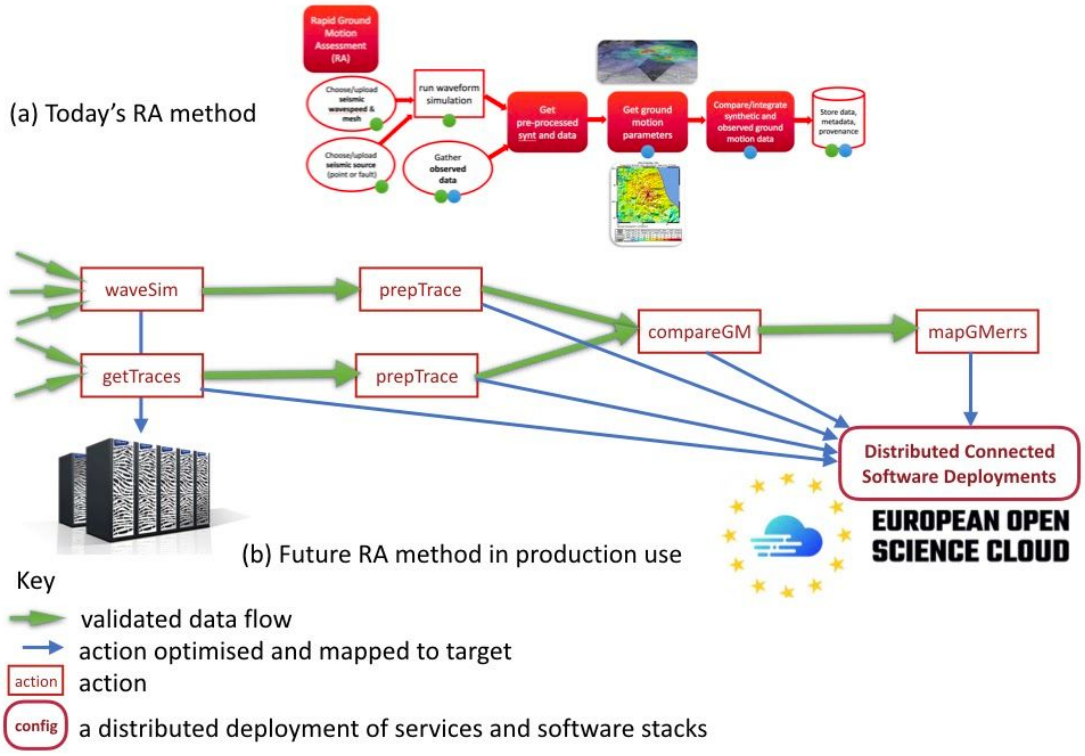

Figure 2.2: *Fluent path from an application group's development of a method to production (see 3.3 for details). The method (in this case for computational seismologists performing rapid assessment of ground motion) is developed and repeatedly refined and tested by the application experts. When they judge it ready it is moved to production automatically on the specified or automatically chosen target computing, data and network services.*

The current intense interaction with specialist computer scientists, distributed systems engineers, data scientists and data architects is highly beneficial and leading to rapid progress. However, it depends on research-project funding[5]. That level of research funding designed to stimulate innovation cannot be indefinitely sustained or spread to much wider communities undertaking application domain long-running campaigns and application-led R&D. For these reasons, and for the reasons given above, it is essential to arrange that the *majority* of this work can proceed effectively and efficiently without sustained specialist input. The need for specialist input at places where significant innovation is needed will always reappear.

Achieving application-domain control and self-sufficiency depends on three interrelated goals:

[5] A succession of eInfrastructure and big-data projects that have built the capabilities and skills we draw on; they were invaluable and much appreciated.

1. *Improving automation* so that there are fewer administrative tasks and less need for users to provide information that could be supplied. This reduces the rate of failures during development, learning and production.
2. *Raising the level of abstraction* so that their work is mainly achieved using stable and implementation-independent concepts, terms and data - less to learn and fewer requirements to re-learn.
3. *Intellectual ramps* facilitating the acquisition of new skills incrementally, without having to climb over substantial thresholds before you can start to benefit from a new skill.

These improvements need to be delivered for each role in a community's team; but, in DARE, we focus on those for *application specialists* and *application developers*. However, if we meet the requirement to deploy instances of the DARE platform on institutional facilities, then the *systems administrators* who install and support those instances will also need consideration. These aspects of the DARE architecture increase in importance as the basic functionality is delivered. Progress and plans may be found in §3 and §4 respectively.

Sustainability is critical for two reasons.

1. Without it we are *behaving unethically*, by leading application communities to depend on a research environment that may disappear, leaving them a difficult recovery path finding replacements and reformulating their methods and working practices.
2. Without it the *return on investment* is lost; the funds put in by our funders, ultimately European tax payers, and the effort put in by many researchers, developers and engineers will yield very little.

Sustainability is hard to assess. It can only be measured in retrospect. It depends on the balance between the cost of sustaining facilities (maintenance and support) and the available resources. The later depends on two factors:

1. The importance they attach to it, which depends on the quality and power of the system, and
2. The breadth and scale of the user community.

DARE seeks to minimise costs by building on widely used software components. The Python and notebook technologies used are very widely adopted and supported[6], for example. And by adopting professional software and systems engineering practices; e.g., those recommended by SSI (§4). The development of take up is covered in §5.

[6] Notebooks are particularly useful in combining documentation with functionality. They still need to be used carefully, i.e., avoiding distracting detail that reduces learning success, mobility and longevity.

3. Use of the Architecture

Architectural components such as WaaS and aspects of P4 have been integrated within the platform, further extend and used by the communities' workflows. However, aspects of the DARE API exposing the WaaS to research developers require to be smoothed out, in order to improve usability, stability/portability and performance. These concern the management of input and output files, authentication and adoption within more interactive environments, such as Notebooks.

For what concerns the provenance information, the communities represented by WP6 and WP7 need to take ownership of the provenance traces, by deciding the granularity and the metadata to be recorded during the execution of their experiments. The framework allows for configuration and detailed extraction of customised information. This should be better exploited to gain effective benefits in terms of results management, discovery of relevant past runs and reproducibility of the experiments.

DARE supports the execution of different workflow technologies (dispel4py and CWL). While dispel4py can be explicitly implemented and configured by the user, CWL is used as a backend engineering solution to organise and execute macro tasks. Thus, research-developers are not directly exposed to CWL. For instance CWL is used to organise and execute SPECFEM seismic simulation workflows by a dedicated method exposed via the DARE API. The two systems, however, present different approaches to provenance generation and description. We are working on homogenising the representation and management of the provenance data, which will require substantial effort in the interpretation and conversion of the formats, trying to achieve better interoperability and usability of the provenance traces produced. To be noted that while S-PROV extends the ProvONE ontology[7], CWL follows a different choice, by adopting a different workflow ontology WFPROV[8]. We have pursued an initial technical mapping that would facilitate the re-use of the storage and query API offered by s-ProvFlow.

Other aspects which are relevant to an effective use of the DARE conceptual design and architecture is the management of user's identities and how these are handled across all of the DARE components and microservices.

Both communities of seismologists and climate scientists benefit from the adoption of remote development environments based on executable notebooks (Jupyter, Jupyter Lab)[9] [Rule, Adam, *et al.* 2018] . Notebooks became familiar among computational scientists because they facilitate the generation and sharing of documentation of methods, source-code and results in a single de-facto standard format. The successful support of such tools requires us to introduce in

[7] ProvONE Data Model. https://purl.dataone.org/provone-v1-dev
[8] Wfprov Ontology, https://wf4ever.github.io/ro/2016-01-28/wfprov/
[9] https://jupyter.org/

the DARE API a set of utilities that allows users to develop, execute and evaluate the results of a workflow within the same notebook page.

The adoption of advanced and interactive development environments, such as notebooks, in contexts where reproducibility is a priority, opens challenges concerning the correct use of such tools and the realisation of mechanisms that facilitate consistent provenance acquisition and interpretation. This has to capture changes to computational environments, such as software stacks, configurations and resources. The latter including new algorithms, as well as data. Concepts concerning system and application domain that are described in the DKB need to be combined to represent setup and exploitation of the computational spaces in a way that guarantees the consistent interpretation of the various entities involved in the long term. Although enactment technologies might change over time, the provenance records should guarantee that researchers are able to locate and understand failures during attempts to reproduce a certain result or re-apply a method. DARE in cooperation with the ENVRIFair project[10] is addressing these challenges. We will discuss these efforts further in section 4.3.

3.1 Use by seismologists

Collaborative work of domain specific scientists, data architects and developers produced significant advances in the design and implementation of the EPOS seismological use case with its foreseen test cases [Rietbrock *et al.* 2018]. Following the requirements detailed in Deliverable D2.1 §7.1 [Atkinson *et al.* 2018] and exploiting the main components described there, we started focusing on the ground motion Rapid Assessment (RA) test case. The aim was to structure a workflow that could help researchers to ease and speed-up the calculation of seismic ground motion parameters (such as the peak ground acceleration (pga), peak ground velocity (pgv) or peak ground displacement (pgd)), especially after large earthquakes, generating specified outputs useful both scientifically and for communication with public and emergency authorities. We used the RA test case as a typical example of our working methods to steer the DARE platform development and build an easy-to-use, customisable framework made of reusable, abstract and flexible components that can serve multiple purposes and extend beyond the immediate EPOS seismological community.

The RA workflow has been designed with modular high-level steps that are represented in Figure 3.1 and described in Deliverables D6.1 [Rietbrock *et al.* 2018] and D6.3 [Magnoni *et al.* 2019b].

[10] ENVRIfair EU project https://envri.eu/

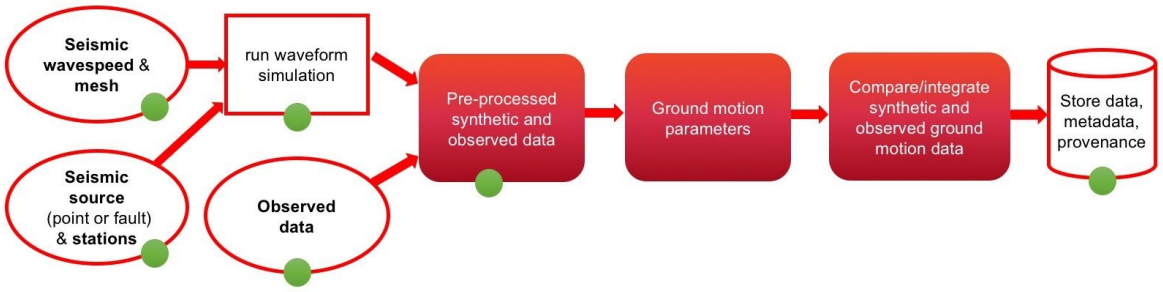

Figure 3.1: *The Rapid Assessment (RA) workflow*. Green dots are the steps in common and reusable with MT3D test case (Fig. 3.2).

The implementation and execution of RA has been made possible by the development of the DARE API components forming the first and second releases of the DARE platform (Milestones MS6 & MS21). In particular, fundamental components meeting this aim are (see §4):

- The *Execution API* to enable distributed and scalable execution of numerical codes and dispel4py workflows;
- The *dispel4py Registry* (or Processing Elements Library) to provide a workspace structure for registering workflow entities (as processing elements) supporting reusability and sharing;
- The *Provenance components* sProv and sProv-viewer to record metadata and provenance and offer visualisation functionalities.

Exploitation of DARE API components to execute workflows is realised through the development environment of a Jupyter Notebook as detailed in §3.3. Specifically, for the RA test case the notebook allows users to access the API functionalities to:

- Register new dispel4py workflows;
- Upload required input files (as user customised input models, see Fig. 3.1) or download useful output;
- Launch numerical simulations, specifically with the code SPECFEM3D_Cartesian (Fig. 3.1), with a simple API call that executes on the DARE cluster a dockerized version of the code containing all the required dependencies:
  F.submit_specfem(n_nodes,data_url=[zip_input_file], token=F.auth(), creds=creds) ;
- Execute dispel4py workflows, as those describing the other steps of RA (Fig. 3.1), through other specific API calls that allow users to specify particular requirements (see §3.3 for more details):
  F.submit_d4p(impl_id=impl_id,pckg=[wp_name],workspace_id=workspace_id,pe_name=[pe_name],token=F.auth(),creds=creds,n_nodes,n_processes,iterations,reqs=[requirements],target=[d4p_model], inputdata=[input_files])

A refinement phase followed the initial implementation of the RA test case in order to remove obsolete intermediate, fine-grain steps that produced input files for main steps or that post-processed outputs from previous main steps. This results in an even more modular

workflow constituted by proper, self-contained dispel4py sub-workflows that perform specific tasks and can be executed, parallelised at scale, by themselves (if the required input files are already available) or in a pipeline. Thus, they can be easily reused for other workflows or can be customised or updated in the future without the need of modifying the whole procedure.

The next EPOS test case taken into account during the second DARE phase focuses on the analysis of the parameters that characterise the earthquake source and uncertainties of these parameters to be able to calculate numerically partial derivatives of the model parameters. For the present application, the seismic source is approximated as a point source and the studied parameters are the earthquake location, magnitude and rupture mechanism represented by the moment tensor, hence up to 9 free parameters (see D2.1 [Atkinson *et al.* 2018] and [Aki & Richards 1980]). The final goal is to improve an initial model of the earthquake source by calculating the perturbations to its parameters that minimise the misfit between simulated and recorded waveforms, a typical inverse problem, and estimating the uncertainties attributed to the new solution. The workflow structure is represented in Fig. 3.2 and the main steps are:

- Choose an initial model of the earthquake source and a three-dimensional (3D) model to represent the Earth structure;
- Simulate the synthetic waveforms for the chosen models using a starting model at initial three-dimensional source location;
- Perturb the source parameters of the initial model and simulate the related synthetic waveforms (called ‘derivative synthetics’);
- Download the available recorded waveforms (e.g. from EIDA archive) for the chosen event;
- Pre-process all the synthetics (initial and derivative) and the data in the same way;
- Compare data and initial synthetics on time windows suitable for the inversion;
- Calculate improved source parameters and uncertainties based on a source inversion procedure.

Since this application specifically considers a 3D model to represent the Earth structure and invert for moment tensor solutions, we named it *Moment Tensor in 3D* (MT3D).

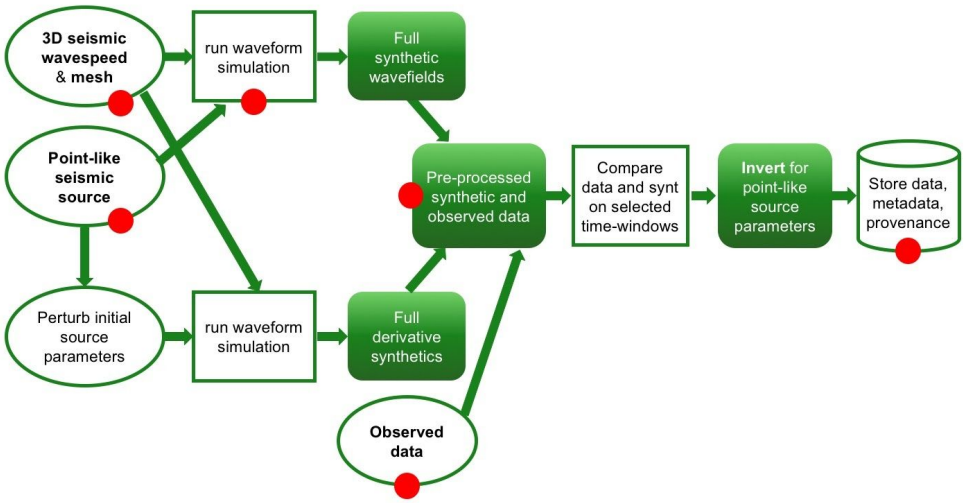

Figure 3.2: *The Moment Tensor in 3D (MT3D) workflow*. It calculates improved seismic source parameters by minimising the misfit between recorded and simulated waveforms. Red dots are steps in common with RA test case (Fig. 3.1)

To implement and execute the MT3D workflow in the DARE platform we benefit from the common steps with RA (see the red dots in Fig. 3.2) already implemented as described above and easily reusable. A new important step is then the simulation of the synthetics for perturbed source parameters. The basic simulation with SPECFEM3D is the same as RA but now multiple simulations with different input files should be managed. Moreover, related metadata and provenance information should be carefully handled in order to combine these simulated derivative synthetics in the following steps of the procedure. This will also enable reusability for future seismic events with a similar starting solution thereby progressively and significantly reducing computation time and costs as the system will have stored more and more well-described fundamental solutions over time. Finally, our plan is to perform the other two new steps by using well-established Python codes already cited in D2.1 [Atkinson *et al.* 2018] as needed seismological components: pyflex[11] for the selection of the time windows suitable for waveform comparison and inversion, and pycmt3d[12] for seismic source inversion in a three-dimensional Earth structure.

Regarding these new steps, DARE architectural and technical task forces will decide:

- whether multiple SPECFEM3D simulations should be managed by launching multiple dockers or a single docker running multiple simulations; and
- whether the new python codes (pyflex and pycmt3d) should be implemented as dockers or as dispel4py workflows. Future update/substitution of these codes will be necessary as the science advances. It is essential that such upgrades should be straightforward.

[11] Pyflex, L. Krisher; http://krischer.github.io/pyflex

[12] pycmt3d, https://github.com/wjlei1990/pycmt3d

The simplicity of future upgrades must be taken into account when making implementation decisions[13].

The development and implementation of the MT3D test case is facilitated by the recent creation of a playground framework, as described in §3.3. It allows users to directly test and debug their workflows by launching 'debug API calls' in the Notebook getting inline output and logs. Further exploitation of the Jupyter Notebook framework could become fundamental if used as an environment where processing elements can be directly developed, registered and executed.

In general, from the point of view of both research developers and domain experts we can highlight some significant advantages of exploiting the DARE platform for the EPOS use case but also for more general scientific applications:

- Exploiting the Cloud for execution, this means elasticity in acquiring and using resources and also possible on-demand computing and storage resources.
- Transparent set up and execution of runs without the need to deal with environment specificity and details of code/scripts execution. Here a single call is used to do all the required steps to prepare the environment and run a SPECFEM3D simulation.
- Exploiting Research Infrastructure (RI) services by including them in the whole workflow procedure, so taking care of the required input, query parameters and gathered output. Here a simple call allows users to query FDSN web services of European archives to download recorded waveforms.
- Rapid and transparent data analyses and transfer between co-working environments.
- Automatic description and storage of complete lineage and multiple metadata that allow us to track runs and data through the whole workflow, to easily search and reuse them and to also combine numerous outputs from multiple workflows that are in widespread use in many scientific applications.
- High-level description of workflow steps that are as abstract as possible to increase the flexibility in reusing them to assemble different workflows.
- Existence of managed knowledge-bases (e.g. the PE registry) that allows users to easily exchange information about what they deployed and executed.
- Workflow structure and provenance information that can be customised.

The incremental advances in the platform components and structure will favour the development of more complex test cases that could interest other communities, beyond the EPOS seismologists, who are looking for a powerful and easy-to-use framework to develop their applications. An example is the proposed Ensemble Simulation analyses [Rietbrock *et al.* 2018] that statistically characterises ground motion parameters and their uncertainties. To do this it analyses ensembles of models representing the variability of the input parameters. Thus, it combines multiple executions of a seismic source analysis test case using e.g. MT3D (requiring multiple numerical simulations). The results are used to explore the variability and uncertainty of

[13] This simplicity requirement, "*it remains easy to install and use new versions of application-domain software and services now and after the project ends*", applies in virtually every aspect of applications development and platform use. **It is a critical aspect of sustainability.** WP6 are exposing it first.

the ground motion parameters, requiring in turn multiple executions of the RA test case. Keeping track of the runs and output to reuse and combine them in ensemble statistics and to spot procedure errors become even more essential.

At the end of the first reporting period a training event has been organised in order to present the execution of the RA workflow running on the first release of the DARE platform (D6.3, [Magnoni *et al.* 2019b], and D8.4, [Casarotti *et al.* 2019]). The main advantages of using the platform have been successfully caught by the trainees, while useful suggestions have been gathered on lacking or perfectible aspects of the pilot implementation and platform development.

Based on this and on the planned work for the remaining of the project, main requirements from the seismological pilot point of view are:

- Ease the handling of a large number of simulations or analyses with slightly different input by counting on smart, abstract and customisable management of metadata and lineage;
- Improve control on workflow execution, error tracking, input and output management;
- Minimise the need of expert developers' help by simplifying and documenting the creation of new dispel4py workflows to favour work independence of research scientists;
- Allow for easy deployment of the DARE platform into external (e.g. institutional) resources to facilitate user access to large computational facilities that will help scaling up the procedures and foster operative seismological applications.
- Provide documentation and training that accelerates the uptake of DARE enabled methods.

3.2 Use by climate-impact modellers

Exciting developments have taken place towards the implementation of a generic workflow tool to access and process climate data, aimed at the climate change impact modelling research communities' users. The tool supports climate platform developers to provide on-demand data processing for users using heterogeneous computational platforms, through the deployment of the DARE Platform. One of the major objectives is to provide transparent access to on-demand data processing using easy to use front-end for end users, through interaction with tailored front-ends, such as that provided by climate4impact.eu (Figure 3.3)

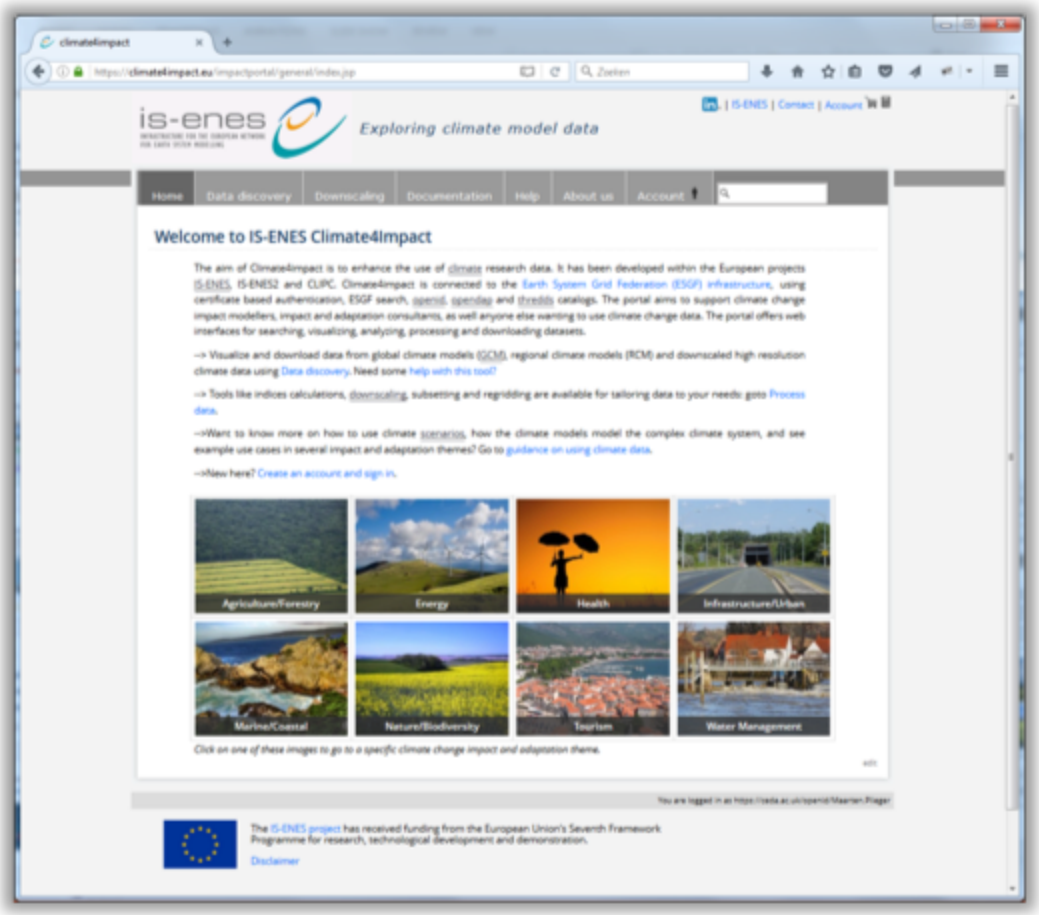

Figure 3.3: climate4impact.eu Platform front-end, providing guidance and on-demand data processing.

The design of the first and second prototypes are taking into account user requirements that have been gathered. Those are described in details in Deliverable 7.1. To complement this approach, User Stories (described in Deliverable 3.1) have been used to provide information on how to properly design components' developments and related architecture. The second prototype of this generic climate data analysis workflow tool, which has been implemented in October 2019, is shown in Figure 3.4.

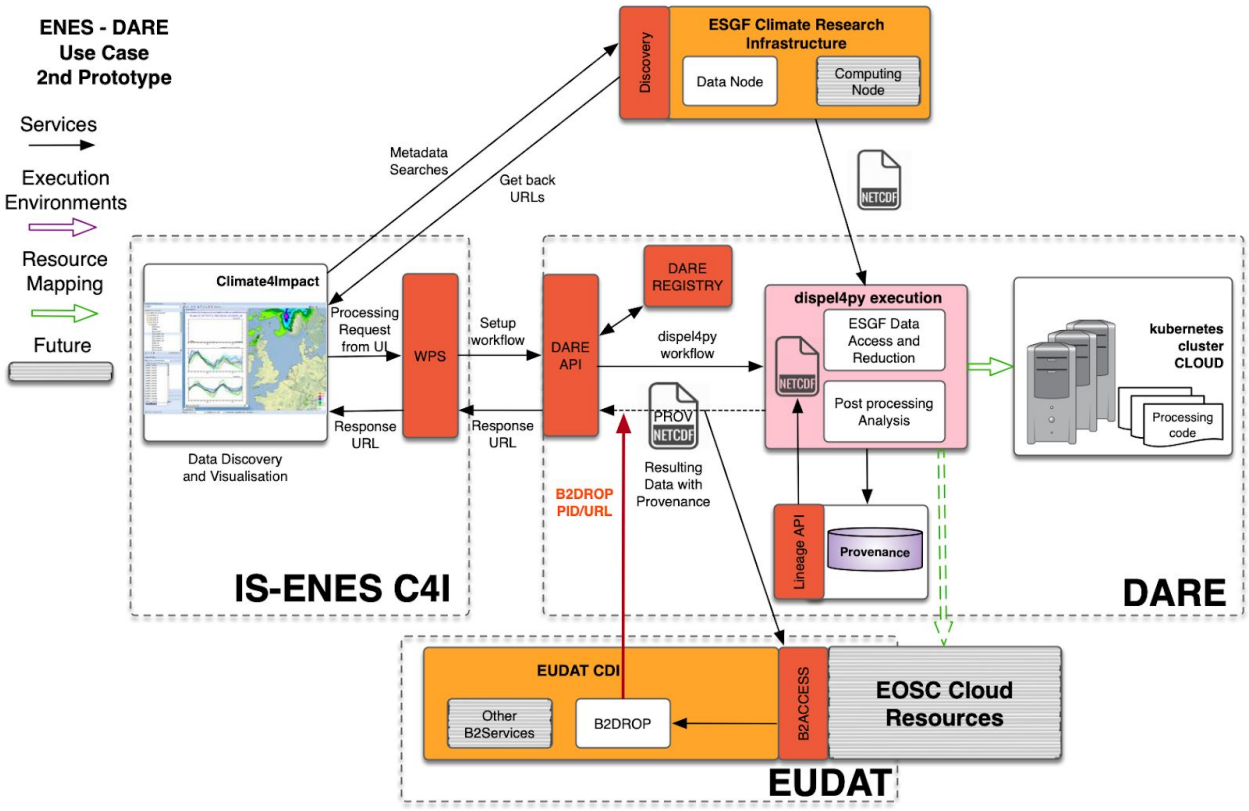

Figure 3.4: Generic climate data analysis workflow second prototype. Implemented in October 2019.

This second prototype has been evaluated during the first training that took place on June 21st, 2019 in Utrecht, Netherlands. The developers who participated in this evaluation have shown a very significant interest in this generic workflow approach, notably by requesting access to the DARE Platform API as soon as possible to test it and to begin developing user services using it. The results of this evaluation is detailed in Deliverable 7.3.

The approach here for this generic workflow is to provide software developers using a platform such as C4I a faster way to develop and provide on-demand data processing for users. The goal with those platforms are not to provide an operational climate services like the one provided by Copernicus C3S. It is rather aimed at researchers in other scientific domains than climate, as well as at researchers doing impact modelling. Those users, with such a generic workflow approach, will be able to provide their own processing functions (as Python functions added to the icclim[14] open source software), as well as to be able to use their own OpenDAP-accessible datasets or use non-standard MIPs from CMIP5 and CMIP6 experiments.

This approach requires an extensive metadata data description, as well as extensive lineage information. This is because when data processing is automated and delegated, metadata standards are necessary to be able to correctly process datasets. Furthermore, it is important

[14] https://github.com/cerfacs-globc/icclim

for users to know how to reproduce the calculations as well as to know exactly which methods and software were used.

During the mid-term review of the DARE project the idea of designing a new climate-related use case more in line with the seismological use cases emerged. This should attract users as it would offer significantly different tools from those offered in other platforms such as the Copernicus C3S Service[15]. We considered the possibility of running climate impact models, but these proved to be too specific and too complex to achieve given the effort available within WP7. We therefore decided to develop a new use case centered on a method/tool that can track extra-tropical as well as tropical cyclones[16] in climate simulations. This use case is called the cyclone-tracking climate use case. Currently, no front-end or online platform offers researchers the opportunity to run on-demand extra-tropical cyclone tracking on climate simulations selected by users. The idea is to provide researchers and users of climate data the possibility to have access to this advanced tool and to provide them with the possibility to generate end-products on-the-fly, such as tracks' density plots, such as shown in Figure 3.5. Given the framework of the DARE Platform the workflow will be developed by composing the following steps:

1. Selection of input file(s) by the user
2. Start of the execution initiated by the user using the C4I front-end
3. The WPS will prepare input parameters using interface user input
4. WPS will upload of the tracking algorithm configuration file using the DARE API
5. WPS will initiate the execution of the algorithm using the DARE API
6. The pre-processing of the input file(s) will take place on the DARE Platform
7. Download of the input file(s) by the DARE Platform
8. Execution of the tracking algorithm (binary executable) on the input file(s)
9. Post-processing of the output files
10. Generation of end-products (tracking density map)
11. Upload of end-products and raw output files to B2DROP
12. Notify user and retrieve files onto the C4I front-end user space

The crucial benefit of DARE is that this can then be easily transferred to a sufficient range of target eInfrastructures, that the resources for the users using this new facility can be sustained.

[15] Copernicus Climate Change Service https://climate.copernicus.eu/
[16] https://github.com/cerfacs-globc/cyclone_tracking

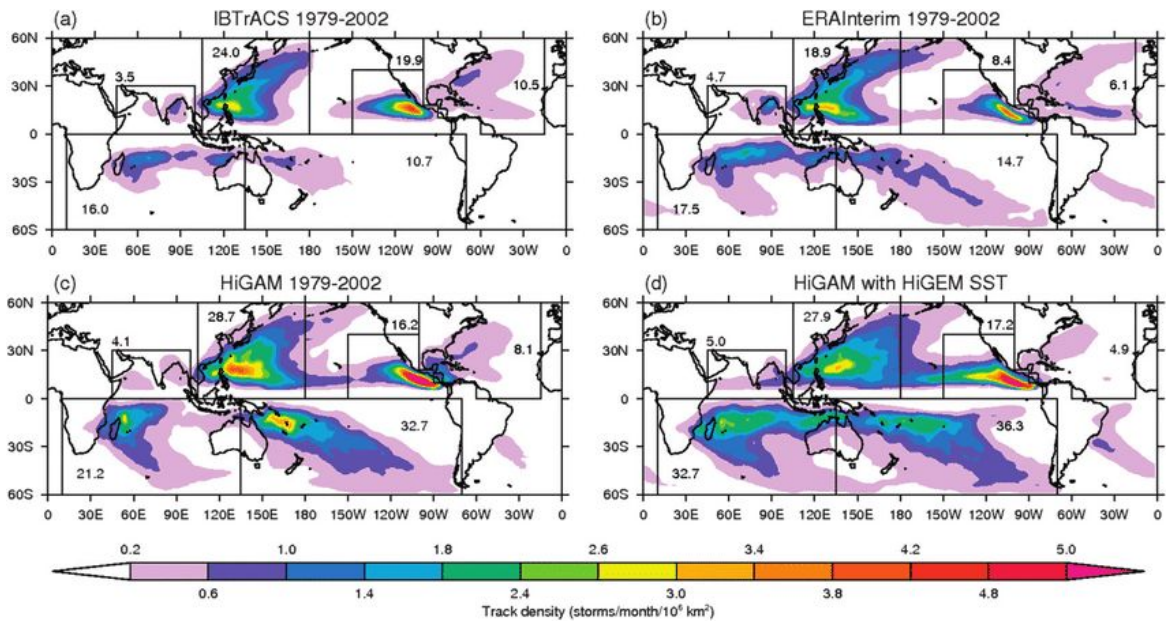

Figure 3.5: Tropical cyclone track density plots. From [Bell *et al.* 2013].

3.3 Use for development

There is a continuity between users and developers, in the sense that some application specialists use DARE workflows developed by others, but still choose them, parameterise them and want to make adjustments to them. Others focus on developing new research methods, materialising them as software libraries, simulation and analysis tools and integrated workflows. There is no boundary between them, rather a continuum. Indeed, individuals move in this continuum as their research requires. Consequently, that continuum needs consistent support.

The new interfaces that we are building on DARE provide a fluent path from prototyping to production. Applications are not locked to platforms but can be moved to suitable new platforms without human intervention and with the encoded method's semantics unchanged.

The complex development and debugging requirements encountered in the latest steps of co-design and co-development with the seismologists have provoked refinement of this development playground and will lead to requirements for self-sufficiency in conducting further refinements and production use. The quality of this support is being tested by a member of the KIT team using it for an unanticipated use case. He will develop, exploiting DARE's framework, a method for using a computational model of mass transport via ash and other ejecta from a volcanic eruption. The target is to have this production ready to be used by staff and MSc students in the summer of 2020 during a field course on Stromboli with its recent volcanic activity.

The DARE platform acts as an intermediary between users' applications and the underlying computing resources, making use of technologies including:

- Container Orchestration -- Kubernetes
- Distributed Engineering -- MPI cluster
- Workflows' technologies -- dispel4py, CWL, Registry, S-Prov

The DARe API (see Figure 3.6), allows users and developers to register dispel4py workflows (applications) to the registry, such as the RA seismology use case from WP6. Once a workflow has been registered, it can be submitted for an execution, and the DARE API will automatically deploy all the necessary environment on demand. Furthermore, it also allows for monitoring the execution status of a workflow in the platform.

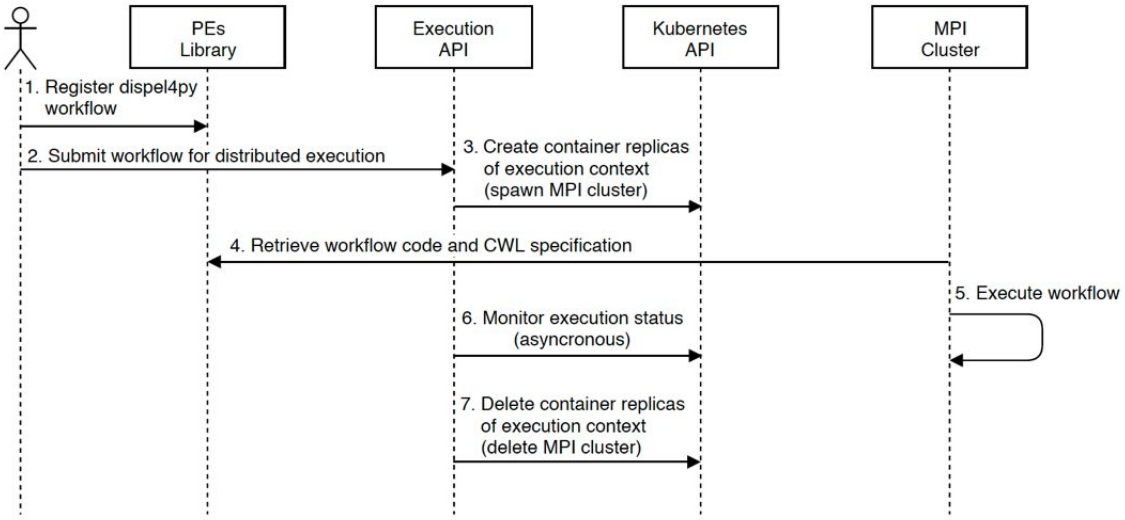

Figure 3.6: *The executions of a dispel4py workflow using the DARE API. This figure shows all the underlying steps as well as the entities that are involved*.

We have selected Jupyter Notebooks to act as the interface between users and the DARE working environment.

In order to facilitate the development and testing of dispel4py workflows, a docker container has been made available to users, to allow them to develop workflows locally (on their laptops or local hosts), which mimics the configuration of the DARE platform. The idea is that users and developers can have a "DARE environment" locally, which has the same libraries, python versions, and so on that the DARE platform has. In the case that a library is missing, or another version for a particular library is required for implementing a use case, these could be installed locally in the docker container, and later they need to be specified to the DARE API at the time of submitting the workflow by using a requirement file, as shows the following API call:

```
F.submit_d4p(impl_id=impl_id, pckg="mysplitmerge_pckg", workspace_id=workspace_id,
pe_name="mySplitMerge",
reqs='https://gitlab.com/project-dare/dare-api/raw/master/examples/jupyter/requirements.txt',
token=F.auth(), creds=creds, n_nodes=6, no_processes=6, iterations=1)
```

Furthermore, a playground endpoint has been recently provided to users with more facilities for debugging their applications and workflows inside the DARE platform. The playground simulates a terminal allowing users to provide a command and see immediately the output results, giving more direct control to users. Below, we show the previous example submitted in a "debug" mode:

```
F.debug_d4p(impl_id=impl_id, pckg="mysplitmerge_pckg", workspace_id=workspace_id,
pe_name="mySplitMerge", token=F.auth(), creds=creds, no_processes=6, iterations=1,
reqs='https://gitlab.com/project-dare/dare-api/raw/master/examples/jupyter/requirements.txt')
```

The current methodology for users to develop new workflows is to use the local docker container, and then later test them on the DARE platform using the "playground mode". When they are satisfied with their validation they submit them to the platform in the "normal" mode.

It is worth noting that the DARE API allow users not only to register and submit dispel4py workflows (with or without additional requirements), but also to monitor and download the results, files and log files associated with a workflow execution.

An example of how the DARE API can be used by users and developers can be found at the following notebook: https://gitlab.com/project-dare/dare-api/blob/master/examples/mySplitMerge/mySplitMerge%20workflow.ipynb, in which a "mySplitMerge" dispel4py workflow is registered, submitted, executed (with and without additional requirements) debugged, and monitored using the DARE API.

Finally, we can find here the list of functions that can be used from the DARE API by a user or developer:

Table 3.1: Functions currently provided via the DARE API.

DARE platform API function	Description
`login(username, password, hostname)`	Get dispel4py registry credentials by logging in
`create_folders(hostname, token)`	Create the working environment
`get_auth_header(token)`	Return the authentication header
`get_workspace(name, creds)`	Get a workspace URL by name

create_workspace(clone, name, desc, creds)	Create a workspace using dispel4py registry API
create_pe(desc, name, conn, pckg, workspace, clone, peimpls, creds)	Create ProcessingElement / dispel4py workflow using d4p registry API
create_peimpl(desc, code, parent_sig, pckg, name, workspace, clone, creds)	Create ProcessingElement/ Workflow Implementation using d4p registry API
auth(length=10)	Generate user "access token" / Simulate user login
submit_d4p(impl_id, pckg, workspace_id, pe_name, n_nodes, token, creds, reqs=None, **kw)	Spawn MPI cluster and run dispel4py workflow
debug_d4p(impl_id, pckg,workspace_id, pe_name, token, creds, reqs=None, output_filename="output.txt", **kw)	Debug a dispel4py workflow in "playground mode"
exec_command(hostname, token, command, output_filename="output.txt")	Allows for running a command in "playground mode".
upload(token, path, local_path, creds)	Upload data into a working environment
myfiles(token, creds) and files_pretty_print(_json)	List the uploaded files.....
download(path, creds, local_path)	Downloads a file using exec-api filesystem reference.
delete_workspace(name, creds)	Deleted a workspace
submit_specfem(n_nodes, data_url, token, creds)	Spawn an MPI cluster and run a specfem workflow
my_pods(token, creds)	Returns user created pod properties (name and status)
send2drop(token, path, creds)	Uploads a file from the exec-api shared filesystem to the project-dare b2drop account in order to get a shareable link for a single file
pod_pretty_print(_json)	Monitoring container status
monitor(creds): Monitor	Monitor a dispel4py workflow run

3.4 Summary and conclusions

The current progress with both user communities has been significant, providing evidence of the potential value of the DARE platform and approach. The seismic methods are developing in complexity and computational demands with a prospect of wider use throughout EPOS and beyond. The climate-impact modelling is beginning to take a similar path. However, issues are emerging that need attention in the coming year.

1. The application developers retain too great dependence on expert help from DARE specialists that will not be available after the end of DARE, e.g., in formulating their work in dispel4py, in configuring the provenance collection and in making methods ready for production use. The following should be considered:
    a. Improved documentation, self-guided tutorials and training
    b. Improved tools and automation
    c. Direct use of method descriptions they already use, such as Python and Jupyter.
2. The computational and data management resources they have access to, e.g., provided by institutional and regional services, cannot easily set up and run DARE platform instances.
3. The PlayGround developments to support advanced developers and innovators are proving their value already. They need further development, improving flexibility and usability for those not already embedded in the DARE platform-development team.

All aspects of these emerging requirements influence the architecture implementation reported in the next section, §4. A broader issue is their link with sustainability considered in §5. Solutions must use minimal new software and build on widely used software and standards - see §5.2. However, significant new software technology is developed by DARE. It will need to be supported beyond DARE as part of DARE's sustainability plan. That support will include software maintenance and expert advice. Addressing the extension of user requirements as summarised above will increase the potential user community thereby expanding the number of organisations contributing to that support. For example, if more of the workflow systems used by EPOS are accommodated, if their model of catalogues interworks with the DKB and if provenance is collected and delivered to P4 tools (§4.3) from all relevant software, the investment in support will come from the whole EPOS community.

4. Architecture Implementation

In this section we review the current DARE platform's architecture, note the progress since D2.1 [Atkinson *et al.* 2018] and propose development paths to address the architectural goals in §2 and to meet the users' requirements in §3. The current implementation is reported in [Klampanos *et al.* 2019] and [Spinuso *et al.* 2019]. The key points are summarised here; readers are referred to those papers for more detail.

As introduced with Figure 1.1 the DARE platform has three technological pillars:

1. *Workflows-as-a-Service* (**WaaS**) help communities develop and use formalisations of their methods. For the supported scripting notations and workflow languages it enables authoring, debugging, validation and optimised productive use of methods. It selects appropriate targets for enactment, prepares them, e.g., by installing the required container, updating its configuration and initiating processing on a network of interconnected distributed processes. This is significantly more integrated and therefore easier to use for all stages of method development than the system reported in D2.1 and the accommodated formalisations now include CWL and Jupyter notebooks. Its technical details and planned development are presented in §4.1.
2. *DARE Knowledge Base* (**DKB**), has three roles:
    a. *Human communication*, a place where practitioners in any role can leave any information they wish for themselves or others.
    b. *Software communication*, a place where software can leave information for its own or other systems' future use.
    c. *Human-system communication*, a place where humans leave information for software to use and where software leaves information for humans; this should improve human-system relationships, improve understanding and enable responsible control of quality of methods and results.

    This was based on two independent catalogues: the Processing-Element (PE) registry [Klampanos *et al.* 2015] for the components workflows are built from and the Data Catalogue, as D2.1 reported. These have been developed further and used more extensively in the current release of the platform. The current development brings these into an integrating, flexible, extensible and incrementally adopted common framework - see §4.2.
3. *Protected Pervasive Persistent Provenance* (**P4**) and the tools and interactions it supports. This is intimately connected with the DKB, as it is a major repository of and source of information about user and system behaviour. However, it warrants separate identification because of the crucial role it has underpinning the quality of science and evidence. By delivering *reproducibility* it has a stand-alone role that may be utilised by some research communities. By supporting provenance-driven tools it significantly improves understanding, addressing the second architectural goal in §2. It is possible to mine information from the growing persistence repository, for example:
    a. The costs and resources used for running processes and complete methods.

b. The frequency with which each category of data is used.
c. The bottle-necks and common pitfalls encountered when performing established or required procedures.
d. The parts of data collections or model-parameter spaces that have been explored; perhaps alerting researchers to critical omissions.
e. The errors users exploiting or authoring methods are making repeatedly; implying that changes should be made to help those users.

These uses of the growing wealth of provenance data have great potential to improve the science and the methods used pursuing that science. This potential is just becoming available, but there are also more mundane steps needed. See §4.3 for details and future developments.

These subsystems are considered in more detail below. They need to maintain consistency with each other as detailed in D2.1 §8.4 [Atkinson *et al.* 2018]. Their current implementation is shown in Figure 4.1.

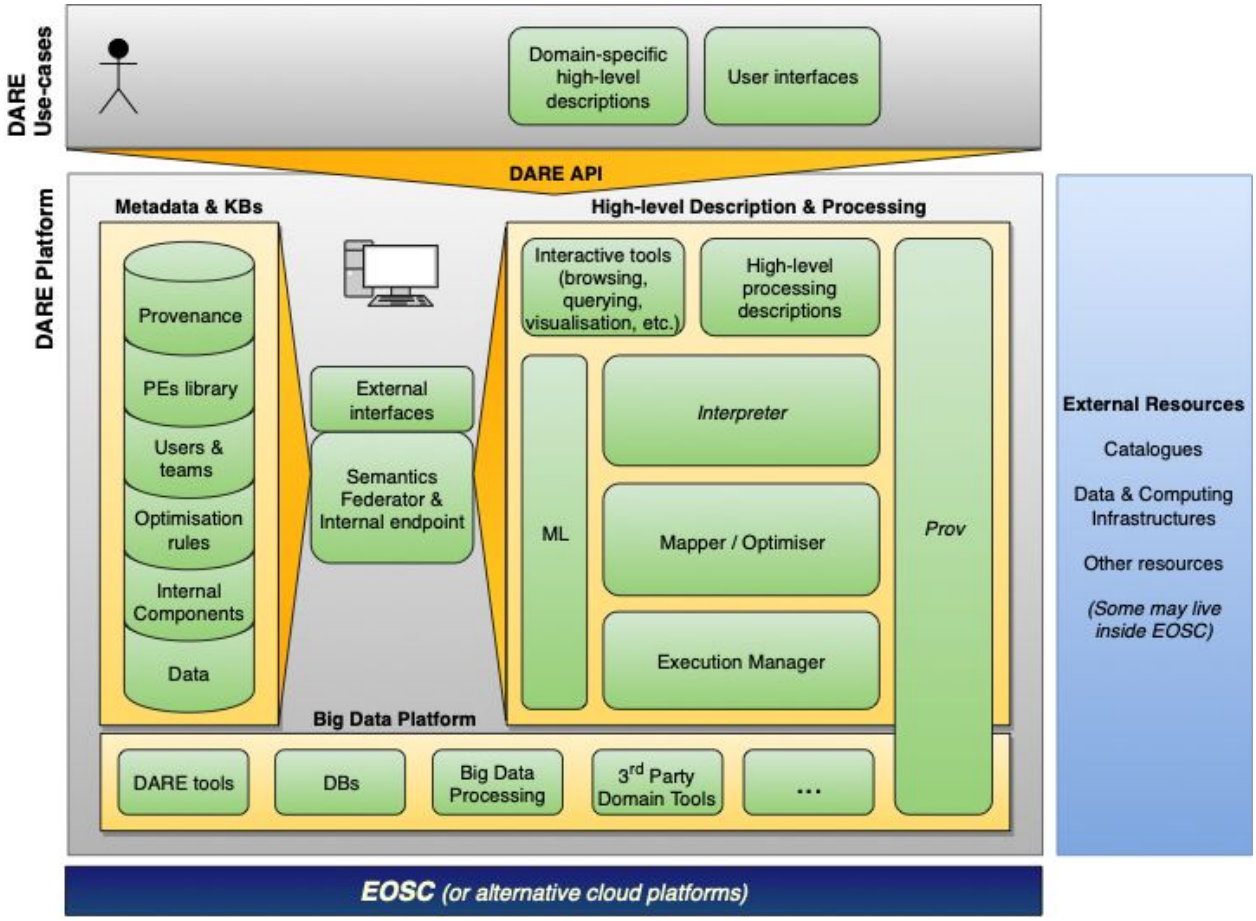

Figure 4.1: *The structure and main components of the current DARE platform from [Klampanos et al. 2019] §A.*

4.1 Workflows as a Service (WaaS)

Workflows-as-a-Service extend the functionality of typical Workflow Management Systems (WMS) to automate set up and management of the computing facilities the WMS needs for the requested tasks. A WMS will support all of the phases of workflow development and the repeated enactment of the resulting workflows. A WaaS identifies the needs of a requested enactment and locates e-Infrastructure resources to meet those needs. It then prepares them using software deployment, configuration, interconnection and orchestration mechanisms, monitors the enactment, organises data movement and storage, and delivers results and enactment records to the users [Filgueira *et al.* 2016, Rodriuez & Buyya 2018]. The DARE WaaS fully supports the dispel4py WMS and is being extended to include other WMS.

4.1.1 Concepts

The DARE platform provides all the necessary tools to research developers for them to execute various workflow development, deployment and production-use tasks. Research developers write code in Python, utilising a workflow language library (dispel4py[17]), which allows them to define fine-grained workflows of arbitrary complexity. Conceptually, a workflow is a graph that connects well-defined units of processing functionality - processing elements (**PE**s). More information on dispel4py can be found in [Filgueira *et al.* 2017]. Each PE defines a process Python method that should contain the code to be executed. An experiment is logically divided into multiple PEs, connected by directional arcs in the aforementioned graph. Data units flow along these arcs, from outputs on a source PE to inputs on one or more destination PEs; order is preserved. This enables dispel4py to represent abstractly a number of parallelisation patterns.

The main concepts of dispel4py are managed via a dispel4py Information Registry (**Registry**)[18], which is part of the DARE platform's knowledge base (§4.2). The Registry is used in order to efficiently store and retrieve workflows and enable workflow reusability. Users can create their own workspaces and register the Processing Elements (PEs) that they intend to execute or share. The Registry provides an API that enables creating, updating and deleting workspaces and PEs. Before a workflow can be executed, it needs to be registered in the Registry.

4.1.2 User Instructions

In order to execute a workflow, users need first to create or reuse a workspace and inside it register the necessary PEs in the Registry. PEs that are stored in the Registry can be reused in future experiments/executions by providing the PE name. More information on how to use the Registry API can be found in the respective GitLab repository[19].

[17] https://gitlab.com/project-dare/dispel4py
[18] https://zenodo.org/record/3361395#.Xg22gy2Q0Wo
[19] https://gitlab.com/project-dare/d4py-registry

The DARE platform provides a test environment, as mentioned in §3.3, in order to execute workflows with immediate diagnostic information and direct control with the DARE platform's computational environment accurately emulated. This accelerates development and substantially improves research developers' powers to investigate issues. The relevant component ("*playground*") provides two functionalities. Firstly, a user can simulate a workflow execution and immediately check the logs and outputs of the execution. The second functionality provides the environment to execute any command, simulating a terminal. More details are available in the corresponding GitLab[20] repository.

When the workflow is ready for execution, the user can execute it via the official API endpoint[21] of the DARE platform. While a workflow is being executed, the user can monitor the containers that execute the workflow, through the API endpoint[22] provided for that purpose. Users have their own directory where the files are organised per execution (test and production executions use different directories). The DARE platform through its API provides functions to list the folders and files in those directories as well as to download any produced files (see Table 3.1).

4.1.3 Workflow Execution

The DARE platform provides workflow execution as a service via an easy to use RESTful API. In order to use the provided services, the first step is to register the workflows in the Registry. Subsequently, users can execute the workflows using the registered name. Users can configure the execution parameters, for example the number of nodes required for the execution. Based on the requested number of nodes, the DARE platform generates an appropriate number of MPI containers to execute the requested workflow.

The DARE platform contains a Shared File System that the MPI containers can access to store or read files. Each run is stored in a different directory. When a workflow requires additional Python libraries, a virtual environment is generated inside the respective directory. After the execution, all the output files are also collected in the run directory.

Through the platform, a user can obtain provenance information in order to track what experiments have been executed, as well as to obtain the input and output data. The DARE platform provides a user interface on top of the provenance API and storage, where the user can view the executions and the data produced in the platform (§4.3).

4.1.4 Future Work - Optimisations

In the preceding sections, we have described the current state of the DARE platform. In the next phase of the DARE project we will improve the use of the shared file system by separating the executions of a user based on the respective experiments (see §4.2). Users should not need to

[20] https://gitlab.com/project-dare/playground
[21] https://testbed.project-dare.eu/exec-api/run-d4p
[22] https://testbed.project-dare.eu/exec-api/my-pods

specify platform- or implementation-specific details, such as the number of processes to be utilised. These matters will be investigated as part of the workflow-optimisation effort. They are important for method portability and durability. They will also reduce the distractions of underlying detail, initially for application-domain users and eventually for research developers when they trust the automation. Additionally, we will investigate the possibility of exposing an API endpoint for CWL workflow execution as well as plain Python or bash scripts.

Another aspect of our work on optimisations is to enable dynamic deployment of dispel4py workflows. Currently, during enactment and prior to execution each PE is translated into one or more PE instances (an executable copy of a PE with the input and output ports running in a process as a node in the data-streaming graph), depending on the number of nodes to be utilised, and once assigned a PE instance to a process, it can not be changed during the execution. The main inconvenience of this static deployment, is that if a PE during its execution needs to be mapped to more processes (e.g. the data-rate consumed/produced by a PE has increased more than expected) or to fewer (e.g. a PE is just executed in few occasions) processes, we can not do anything about it apart from manually intervening to stop the current execution and re-assign the process to PEs either manually or by applying an assignment algorithm based on previous executions. This is clearly costly in human effort and computational resources compared with preemptive adaptation.

By enabling dynamic deployment and enactment of dispel4py workflows, PE instances will not be locked to specific processes, scheduling PE instances on-the-fly, meaning that if a PE needs more or less “resources”, it will dynamically up-scale or down-scale, rebalancing automatically the graph, without stopping the workflow execution. To do so, we are planning to implement the *work-stealing scheduling strategy*[23] [Frigo *et al.* 1998, Mattheis *et al.* 2012]. A mechanism to provide load balancing in case of dynamic workloads, which offers several benefits (e.g. data locality, scalability) in terms of efficiency and usability. It has been employed in a number of frameworks for parallel programming, e.g. as Intel Threading Building Blocks (Intel TBB[24]) or GrPPI [Dolz *et al.* 2018], and has found a variety of applications, from simple divide-and-conquer algorithms to more complex stream processing applications [Anselemi & Gaujal 2009, Navarro *et al.* 2009]. This work will create a new dispel4py enactment mapping, based on ZeroMQ message queue[25], which will implement a runtime work-stealing scheduler to execute the different PEs respecting dependencies and balancing the parallel workload. The concept of *affinity* will be exploited in this new mapping to ensure locality-aware scheduling.

4.2 The DARE Knowledge Base (DKB)

The DARE Knowledge Base (**DKB**) is an integration and packaging of the information repositories that developers and application domain experts use. Its design, which is still

[23] https://en.wikipedia.org/wiki/Work_stealing
[24] Intel TBB
https://software.intel.com/en-us/blogs/2018/08/16/the-work-isolation-functionality-in-intel-threading-building-blocks-intel-tbb
[25] ZeroMQ https://zeromq.org/

evolving, is described in [Atkinson & Levray 2020]. There will be one instance of the DKB per DARE platform deployment. The current developments can be found in the GitLab repository for the DKB[26] and in those for the other information sharing facilities, described in §4.2.4. The current functionalities are summarised in Appendix 2.

4.2.1 DKB requirements

The DKB should deliver the following benefits compared with developing and using individual information-sharing subsystems directly:

1. *Easier for application-focused users to extend and use* - they should be able to directly create, change and use their shared information to help them organise their production, collaboration and innovation work.
2. *Provide incremental support for adoption* so that current developed practices, catalogues and information-sharing frameworks can sustainably co-exist with DKB use[27]. Users and developers will decide when they use existing information services directly and when they or their software works via the DKB. In the latter case, they can make local changes in one place to accommodate changes in the used service or to add functionality.
3. *Progressively raising abstraction* improves understanding, portability and durability. It protects those who use it from unwanted external changes and distracting technical detail. This also enables DKB implementers to re-engineer their mapping to services to accommodate and exploit external changes and to address issues identified by users, with less disruption to ongoing production and development work.
4. *Boundary crossing*, experts from different disciplines need to collaborate to develop innovations and advances in complex and challenging campaigns. The DKB should deliver the support for CSCW *boundary objects* where their work overlaps, while delivering as much freedom as possible to experts where it doesn't.
5. *Commonalities* between and within communities can be discovered and exploited.

The DKB has to comply with the following constraints:

1. It must be *open ended* because the paths researchers, developers and communities will take are unpredictable - this mirrors Linked Open Data (LOD) represented by RDF[28]. However, within the DARE KB we underpin this freedom with a consistent foundation and accelerate productive use with a conceptual library, intending to get the best of both worlds.
2. It must be *directly controllable and manageable* by application communities as this makes them more self sufficient, with agility (speed of response to needs and opportunities) and less dependent on technical experts. This means that the formal underpinnings and implementations must be well hidden.

[26] https://gitlab.com/project-dare/dare_kb.
[27] In DARE these include the Registry, the Data Catalogue and the Kubernetes catalogues. The relationship with P4 is complex and under development. Each user domain also has its established archival practices and data-exchange standards.
[28] Resource Description Framework (RDF) https://www.w3.org/RDF/

3. It must, nevertheless, facilitate *sustained productive collaboration* between application and technical experts, as identified by Trani for the EPOS RI [Trani *et al.* 2018]. This means that the formal underpinnings and implementations must be understood by experts in their use and in the aspects of technology they are used to describe.
4. It *cannot require a 'green field' site*; it must operate in conjunction with existing information stores, operational software and established professional practices.
5. Constraints 2 and 4 imply that it cannot take *full* responsibility for correctness and consistency. It should do this for information entirely assembled via its functions. It may also support methods for verifying consistency that developers and others may employ.
6. It must *persistently retain information* entrusted to it, while respecting the structure and dynamics of the organisations that contribute to and use the DARE platforms. This means reflecting the recursive pattern of commonalities and releases while retaining all local work accomplished in the context of earlier releases. This requires methods for preserving the work so far, installing the new releases and adapting to any changes as it restores local work. Discrepancies encountered during these procedures must be referred to relevant users.
7. It must be *sufficiently fast* to meet production and concurrency requirements.
8. It must be *sufficiently protected* to prevent tampering, leakage of private or confidential information, and loss by accidents and user errors.

Within these goals and constraints, the DKB must be co-designed and co-developed with DARE platform and application developers to establish its immediate relevance and path to adoption.

4.2.2 DKB roles

The term 'DKB' in this section refers to the composition of the *logically* centralised integrator and the co-existing other information services. Four *low-level roles* are supported:

1. Any user[29] may *enter* information into the DKB for their own or other users' future use.
2. Any user may *enter* information into the DKB for software's future use.
3. Software may *enter* information into the DKB so that it informs users - with appropriate adaptations for the target recipients if at all possible.
4. Software may *enter* information into the DKB for its own or other software's future use.

We therefore introduce the generic term "***Entry***" for any item of information considered as a unit by the agent that enters it. As for other data, we then need to manage the lifetime of each Entry as shown in Table A2.1. Some of the transitions envisaged during such a lifetime are shown in Figure 4.2.

[29] Acting directly, e.g., using a Python function in an interactive Jupyter session, or interactively via intermediate software.

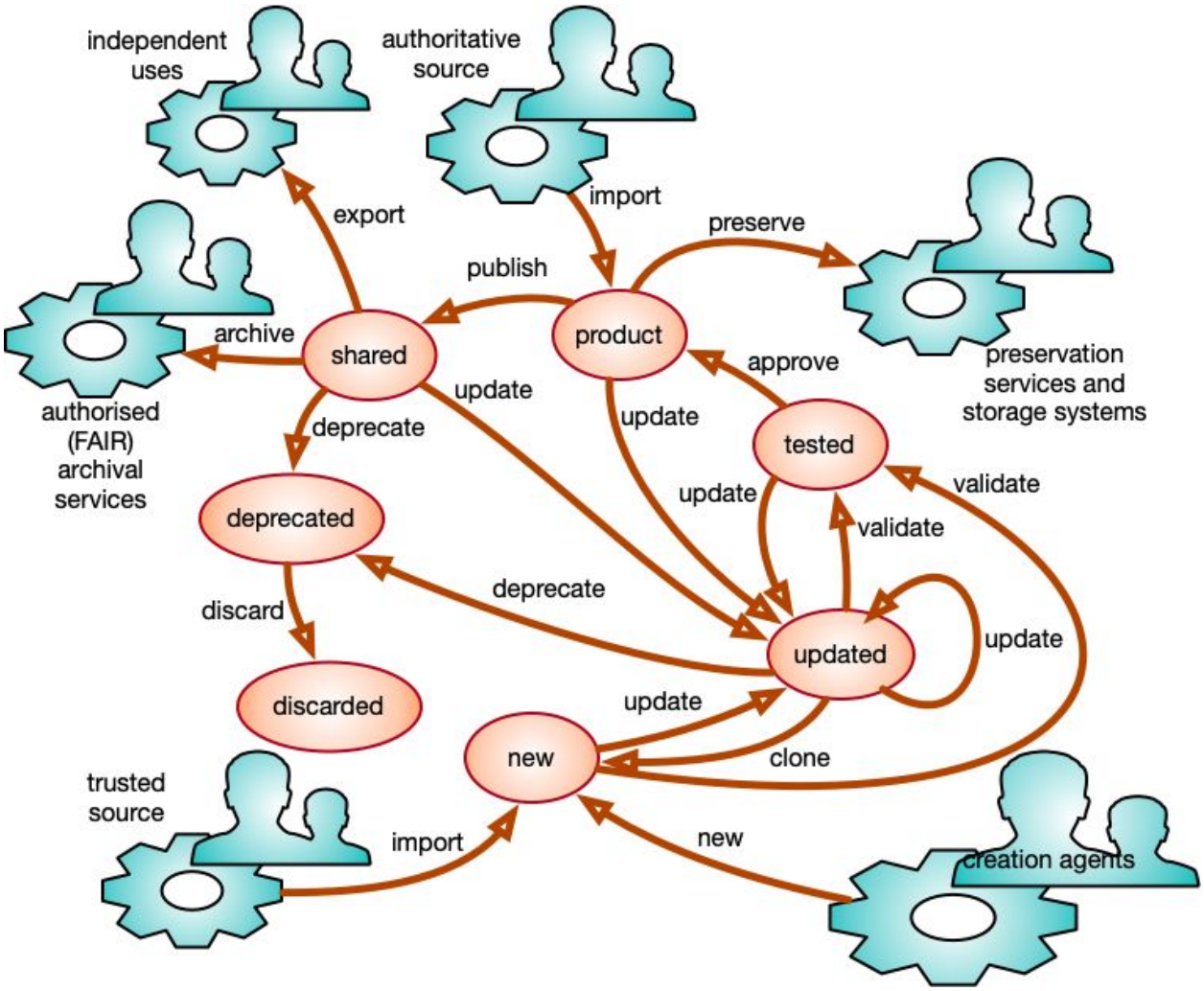

Figure 4.2: State transitions during the life of a DKB Entry. Not all are shown. The external initiation of an action causing a transition is only shown for the initial creation, but transitions are normally initiated and controlled by external agents. Research developers perform much of their work in Git-managed spaces, so many of the imports will be from such spaces.

The implementation of these functions is an appropriate combination of the following mechanisms.

1. *Delegation*, the direct presentation of a function of some information service or a wrapped request for that service, e.g., to handle simple format changes.
2. *Local action*, the representation of the relevant information entirely within the DKB service with changes to that information required to implement the action. Normally, actions implemented within the DKB will be *atomic*, i.e., the state change is complete or there is no change.
3. *Harvesting*, the acquisition of required information from an external source that is then held locally. This will be a snapshot. The external service may later change the source

data[30] making the snapshot out of date. This may be semantically significant, only a user can decide which value is now the required one.

4. *Querying*, sending queries to one or multiple sources and then combining and transforming the results returned.
5. *Caching*, conducting queries, as in harvesting and querying with a policy for discards from the cache, to limit local resource use or to reduce the risk using out-of-date results.
6. *Proxying*, the DKB represents aspects of an information service as if they are local based on agreements with that service, ideally supported by digital or human protocols. It then presents the selected service locally, potentially introducing adaptations when details of that service change, but otherwise representing it accurately.

In some cases the implementation may combine these using a workflow mechanism. Consequently, the relationship with the provenance system, P4, needs careful design to avoid unwanted behaviour. This is revisited in contemporaries (§4.2.4) and in DKB R&D planning, (§4.2.5).

When fully exploited the methods created by research developers will use the DKB frequently. The interpretation of user actions, developer actions and the coded methods will all interrogate the DKB to translate into finer-grained actions, to map to evolving infrastructure and to optimise based on accumulated information about prior work, about users, about services, about software and about data. This is illustrated in Figure 4.3.

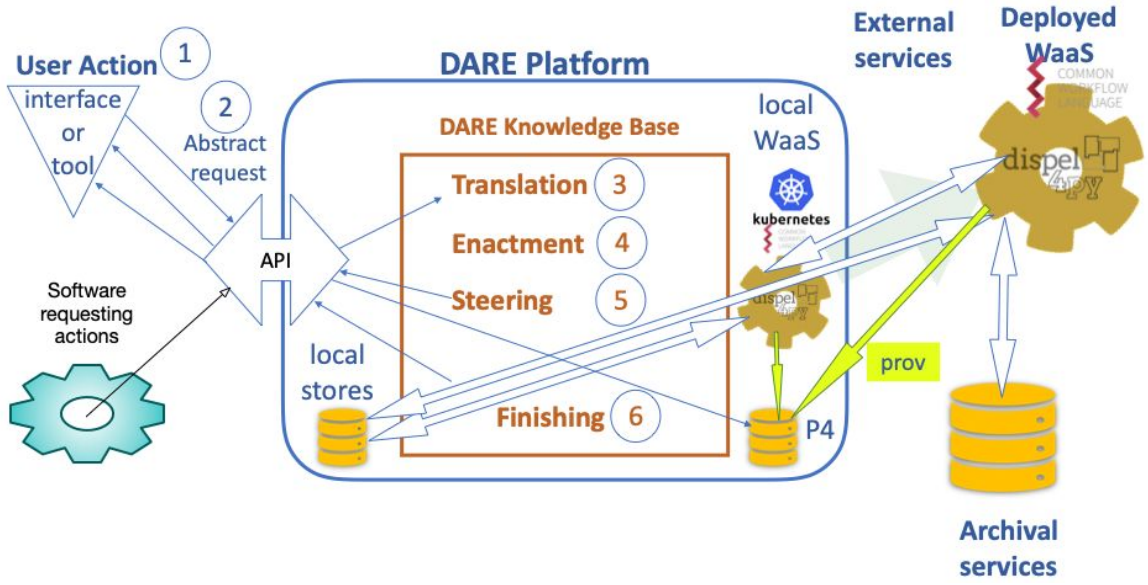

Figure 4.3: The DKB acting as an intermediary throughout the initiation and enactment of a sophisticated method steered by and reported to a DARE platform user. The numbered stages of this process are described below.

[30] Protocols may be introduced to detect and warn of this divergence, but they are not standardised or commonly available - they should exist for DKB contemporaries (§4.2.4).

Figure 4.3 depicts a future DARE platform where the DKB is used intensively.

1. A user, using a tool such as a Jupyter notebook requests an action in a form suitable for them. Similarly, an action may be requested by some external or internal software.
2. The request with parameters, etc., arrives in an agreed form at the DARE platform API. Entries in the DKB describing methods and built-in functions automatically populate this API.
3. The DKB retrieves information about the identified method, parameters, and referenced data. It passes this to the WaaS, which may use this information, may request more and may write records for subsequent parts of this enactment or future similar enactments.
4. The WaaS receives the request and may request further information from the DKB in order to optimise, map to a target technology and to deploy and configure the required virtual infrastructure[31]. It may consult the DKB for descriptions of potential targets the requestor is authorised to use. It will ask the DKB about past costs to estimate expected costs. Harvesting-processes may scan provenance data to summarise past costs.
5. During the conduct of the workflow, information gathered by P4 will be transformed using DKB data about the requestor's preferences. Similarly, incoming steering actions will be translated using DKB information. Engaging the DKB in the information flows from the DARE platform and running software is a crucial innovation. It is during such flows towards users, particularly when failures are reported, that systems expose technical detail that was abstracted away in input flows. Mapping these to forms understood by the interacting user is essential[32]. Otherwise, users have to learn to understand them. They may then exploit them in their future work, locking their methods into a particular technical context. At the very least it is a distraction. However, developers may want them. Hence the tailoring to the current user.
6. The final records written in the DKB will link up the run with the provenance records and with results and if they were specified in the run's profile, intermediate data sets normally discarded. If a user chooses to consult the results and diagnostic data at any time in the future, the DKB will 'know' where they can be found, how they can be retrieved and transformed for that user.

Each deployed instance of the DARE platform will have its own DKB instance as shown in Figure 4.4.

[31] This may require integration with the orchestration technology, currently Kubernetes (see §4.1 & §5.2).
[32] By working via P4, the mapping is from one standardised, PROV-O representation that avoids higher-level tools and platform elements having to work with the vast diversity of system monitoring data.

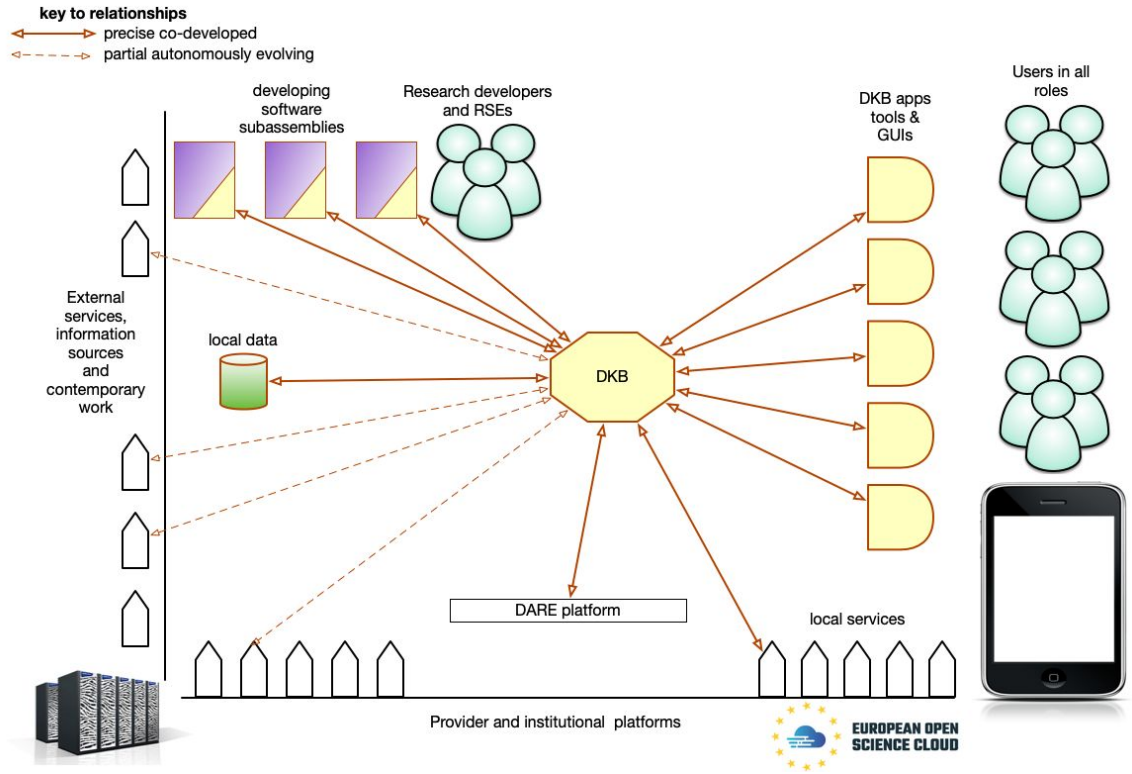

Figure 4.4: A typical deployment of a DKB maintains relationships in two forms. New applications and tools developed using the DKB maintain consistent information in it - solid arrows. Legacy systems and external systems controlled by others will not have complete information in the DKB. It will hold only the relevant aspects when last used - dashed arrows.

This incomplete information is an inevitable consequence of supporting research and innovation that may lead anywhere[33]. The quality and reliability of DKB information therefore depends on the care and precision of its developers and users. To mitigate the risk without constraining users with an excessively rigid regime we partition the DKB information space into *research contexts*, *Contexts* for short, that are the analogue of file-system directories and workspaces.

4.2.3 DKB contents, structure and functions

Users[34] are free to record any information in the DKB that they choose. However, that freedom has three problems:

1. It takes far too much effort before the benefits are available to most users.
2. It is hard for anyone else to understand what has been done.
3. Optimisation is limited to the regularity which software can uncover.

[33] It is also necessary because the DKB has to be introduced into operational contexts and it will not 'know' about everything that is going on.
[34] This includes any software that writes to the DKB, as it is implementing decisions to use the DKB made by its developer.

These are addressed by the following mechanisms:

1. Built-in regularity in the properties and management of every Entry.
2. Contexts in which users have total control.
3. A specifiable inheritance of information from other Contexts.
4. Local naming within Contexts.
5. Global persistent identification using automatically generated PIDs.
6. A conceptually organised library provided with releases of the DARE platform.

Entry specifications

The Entry as the unit of recording in the DKB was introduced at the start of §4.2.2 and the operations envisaged for an Entry[35] are in Table A2.1[36]. It will have a set of built-in attributes several of which are set automatically. These are illustrated in Listing 4.1.

```
concept Entry:
                "Common attributes of every Entry made in a DKB"
                name: String,           # name unique in its Context
                prefix: String,         # its prefix is unique in this DKB instance
                pid: String,            # persistent exportable identifier
                instanceOf: Concept,    # every Entry is an instance of something
                timestamp: Instant.     # when the Entry was created or changed
                state: String           # where it is in it's life, one of a DKB determined set of states
                …                       # beyond here inserted automatically during an update
                previous: Entry,        # if an Entry updated, PID of previous state
                subsequent: Entry,      # PID of the Entry superseding this Entry
```

Listing 4.1: The built-in properties of every Entry. Their usage is described further below.

Notes on these attributes (properties represented by a literal and relationships represented by a reference to another Entry) follow.

name	a user-chosen identifier, c.f., identifiers in programming languages (PLs). Extracted from a user's notation. Context in KBs performing role of scope in PLs.
prefix	the Context active when the name was defined - like a scope in programming or a directory in file systems. Tracked automatically.
pid	a unique identifier, PID, that is unlikely to be accidentally duplicated and that users and systems,e.g., P4, can store and use in the future. Set automatically[37].
instanceOf	each Entry is an instance of a Concept (see 'Conceptual Library' below). When a user doesn't specify a Concept it defaults to Thing ('<DKBinstance>:kb:0:Thing').
timestamp	A UCT instant of sufficient precision to let investigators and software time travel in the DKB to any instant and to arbitrate concurrent updates' precedence.

[35] We start Context identifiers with an upper-case letter, hence E begins Entry and C begins Context.
[36] Appendix 2 holds further details about the DKB.
[37] This is available as a URI by defining prefixes. The storage system may use an additional one and Concepts may have established conventional PIDs for their instances as well.

state	A String value from a small set of options, yet to be defined.
previous	The PID of the Entry that was updated to make this Entry.
subsequent	The PID of the Entry formed by updating this Entry[38].

Users may add any other data in an Entry, e.g., via functions that take a JSON file, or a Python dictionary. This direct use is deprecated because it takes too long to achieve results and because it often leads to poor structure. However, it is permitted and supported in case the library of supplied structures and functions isn't sufficiently extensible.

Instead, the conceptually organised library is provided to start users on a path that meets common requirements. It may be extended by communities and user groups to help with their additional recurrent needs or to specify how their information should be organised[39]. To encourage this we establish Conceptual modelling through the conceptual library[40].

The chaining of versions when an Entry is updated, is intended to allow stand alone uses of the DKB to work without a separate provenance capture system. The DKB will provide simple functions, called Actions, for which this record keeping is efficient. They will normally be atomic and affect only local state. They can then be used in methods whether or not those methods are using the DARE provenance system (§4.3). The required relationships will need to be formalised.

Persistent Identifiers (PIDs)

The PIDs are manufactured as shown in Figure 4.5.

Figure 4.5: The structure of the PIDs manufactured for each Entry

Precise and persistent identification of anything in a user's or a software system's world is essential to ensure unvarying interpretation of those entities when required, e.g., to ensure that an established practice is conducted consistently or to achieve reproducibility. The DKB takes on this responsibility by forging, preserving and interpreting PIDs. As users and software may copy references to Entries in the DKB, this interpretation cannot depend on local addresses or current storage arrangements. The first part of the PID identifies the particular instance of the DARE platform where this instance of the DKB is employed. If this identity, when expanded, meets PID guidelines, e.g., those espoused by the PID forum[41], then the Entry identity will also meet those, i.e., both will be URIs. But each user community will choose how formal to make

[38] If present the Entry cannot be updated again, so a linear chain of updates is formed, avoiding races.
[39] For example, seismology groups might establish a Context holding the FDSN features they use and another holding the OGC features they use. Groups and individuals would add these to their Context's search paths.
[40] Of course, the conceptual library provider may build this by using the Entry functions.
[41] https://www.pidforum.org/

their PIDs and how much to invest in ensuring the longevity of their interpretation. The context prefix is a string guaranteed to be unique in this DKB instance[42]. The uniqueness counter ensures successive updates to an Entry have a different PID[43]. The user's identifier is the name given by a user, e.g., as an identifier in their Python script. The DKB can act as a proxy for an external information source or a contemporary service and forge a local Entry with a PID containing a reference to that external service. It may hold timestamp and signatures of the referenced entity to detect autonomous mutation. We now have the machinery in place to build and use research Contexts.

Context specifications

The primary role of research Contexts is to gather a set of Entries to provide a work context well adapted to a particular user working on a particular task. They also represent the common requirements of multiple users, e.g., members of a group working on a common problem, or of multiple similar activities, e.g., repeated performances of a standard procedure. This is achieved by nested Contexts with specified inheritance from outer Contexts, as shown in Figure 4.6, where we see an additional Context for each user community and an upper-level, dare, common to all DARE applications, which in turn builds on a universal conceptual library, kb, and pre-imported bundles from standard sources.

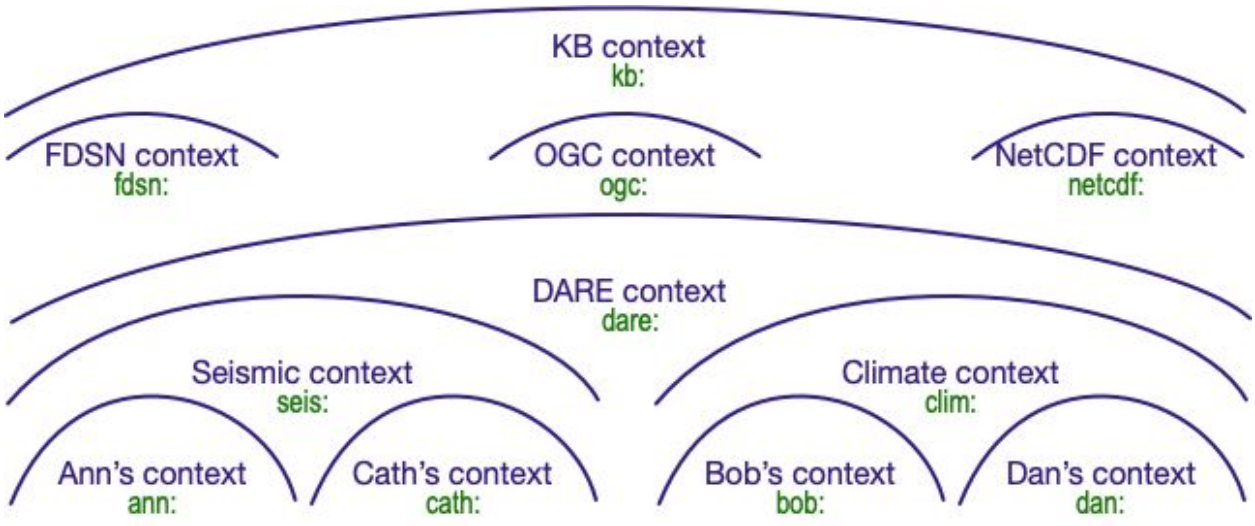

Figure 4.6: The nesting of research contexts, progressively forming work environments that are highly tuned to an activity, an individual or both.

A Context has a prefix that is unique for the instance of the DKB supporting it. It may have an initial population of Entries. These may be updated, added to or discarded as the Context is used. For example, a new Context may be formed by *cloning* an existing Context, e.g., to make a copy of a production Context in order to experiment with a new way of conducting its work, or by making a new empty one. Each Context has a specified search path of other Contexts, e.g.,

[42] It may also have a defined expansion for URIs in ontological representations - behind the scenes.
[43] An index from PID to Entry would accelerate DKB operations.

['seis', 'dare'] for Context 'ann', to arrange inheritance, and to specify overlapping interests. The algorithm for scanning search paths is given in A2.2. If users wish to inherit from a Context C, with prefix 'c', adopting all of C's inherited Entries, they simply specify ['c'] as the search path. This is the default, if they are in Context C when they make a new Context.

The uses of Contexts include:

1. *Importing a bundle of terms* and related entities into a KB, as illustrated by three Contexts in the second row of Figure 4.6. These may then be used in any search path and be separately maintained, e.g., to reflect changes issued by the authoritative source.
2. *Denoting a set of shared terms* and resources, as illustrated in the next two rows of Figure 4.6. Communities may govern how these are maintained. Explicit references using a full PID delay the impact of changes in such Contexts.
3. *Providing a work Context* for an individual, a group or a procedure, that is progressively tailored as it is used to better support that work. Illustrated as the bottom two rows of Figure 4.6.
4. *Providing a method enactment Context* (not shown in Figure 4.6). Methods are repeatedly run with the same or different parameters. The method needs a new Context for each run, so that it can use the same set of names each time, differentiated by the automatically varied prefix.
5. *Acting as a boundary for access controls* and authorisation. The Context can have a consistent aspect of sharing, privacy, confidentiality, etc., and an owner or governance body can set that.
6. *Supporting a user-controlled transaction*. The updates to all Entries within a Context from a defined time (denoted by the uniqueness maker's counter) to a chosen instant can be considered together, e.g., to be 'pushed' to persistence or to be retracted[44].

An Entry in an explicit Context, not necessarily in the search path may be specified as <prefix>:<name>, e.g., 'cath:eqEvent20190723' from Ann's context to refer to the latest update of the earthquake event Cath is working on[45]. A platform may provide functions, e.g., publish, to reveal such Entries and others to control visibility and mutability. Within such constraints, a user may specify an explicit version using its PID, e.g., to freeze a method in an authorised form, while others may be improving it[46].

When a user interacts with a DARE platform they use a specific Context[47]. They may be allocated a clone of a group's Context when they first start work. Their work will then modify that Context so that all new things they create and revisions they make appear *in their Context*. As searches are by name in the local Context first, and then in each Context along the search path,

[44] Uses 5 and 6 will not be attempted in the initial prototype.
[45] Future platforms will include DKBs with visability and mutability controls.
[46] An index from <prefix>:<name> to the latest Entry will be useful. A cache of names currently only found further along the search path with their prefix would be an accelerator.
[47] Normally the one they were using last time, which involves their identity being obtained via AAI and this being mapped to their Person PID in the DKB. This needs inclusion in the platform's login API.

they start in a rich and productive Context. They can proceed uninhibited to use names and create Entries as these are local to their Context. *Innovation is uninhibited*, since they can redefine things named along the search path and thereby hide them and experiment with new forms[48].

However, this does not support collaboration. That has to be done, by publishing new things to 'friends' and then they explicitly name them using your prefix, e.g., to confirm that what you have done is valid and useful in their Context, or to conduct the next authoring steps. When authorised, validated and valuable Entries will be promoted to a higher shared context others in the group can then use. Authorised users will explicitly visit a Context, e.g., to work directly on a shared context, to move between production and innovation, to maintain a shared Context or to help someone solve a problem[49].

The methods for conducting routine repeated processes start by cloning a Context containing the Entries that differ for each repetition, so that they are grouped under the new Context's prefix identifying the repetition with the same local names for every repetition. A user responsible for a number of these running concurrently will move between them. A summary of the operations on Contexts may be found in Table A2.2. The Python calls implementing these will be made available as a standard Python library and described in a separate document [Levray 2020].

Contexts are *not* static snapshots. They continue to change. Users and methods may exploit this. For example, they group a bundle of changes during an experiment or while a method runs. If these are to be discarded, the Context holds them. Similarly, if they are to be retained, the Context may be retained or promoted. There is an opportunity to run Git-like operations to incorporate changes and to detect conflicts[50].

Conceptual library specifications

Whilst users and software have the option of using the DKB in completely novel ways, there are compelling reasons for providing an initial library of Entries all in the Context 'kb', as they span potential KB platform uses. The commonalities expected in the variety of DARE applications will then build a Context 'dare' that exploits 'kb', see Figure 4.6. These introduce several groups of Concepts. We focus on *Concepts*, e.g., Energy, Temperature or Country, as they underpin thinking and communication in humans. We expect a body of well-established and widely adopted Concepts to underpin every research community; developed and sharply defined in their minds by their education and training. This enables them to communicate and think effectively, as they use words with those precise meanings. In the KB we intend that they should

[48] There may be a small subset of names that are marked as "may not be hidden", to enforce consistent use of those names.
[49] This is a common requirement for support staff. It may require permission from the Context's owner.
[50] Oscar Corcho reported doing this for ontologies at the INGV DARE plenary [Corcho 2019].

use those same terms as easily and precisely; building on Trani's work with EPOS [Trani 2019, Trani *et al.* 2018].
Contexts enable different groups to use different terms or the same terms differently. In due course, to enable boundary crossing and novel interworking between disciplines, the DKB will need an underpinning metadata translation and cross-referencing framework, to enable collaboration and innovation to co-exist productively. The requirements and a viable approach are well illustrated for Europe's museums collections of natural science specimens [Lannon et al. 2020]. The EU VRE4EIC project demonstrated the automation of metadata translation to deliver an integrated view of catalogues [Martin *et al.* 2019]. We need to build the foundations of the DKB and establish its initial use before these issues can be explored. Current thinking looks at immediate needs. The need to have variety and consistency, stability and innovation co-existing and flourishing together will emerge as soon as sustained use for a research campaign is attempted [Ramakrishnan 2018] (see page 263).

As research progresses the revision and refinement of the Concepts is inevitable. The DKB supports and stimulates those Conceptual refinements by enabling application communities and individuals to directly shape their active repertoire of Concepts.

We provide a Concept library prepopulated with Concepts and a structure relevant to data-intensive and computationally intensive research campaigns. Building on these, a research campaign or community will develop a sophisticated and highly tuned set necessary for their research. We expect this to become a significant intellectual and practical asset. Groups, organisations and specialists will also build on the initial foundation, on existing and contemporary developments and on imported bundles of relevant knowledge. The common starter includes:

1. An initial set of Concepts, some of which have instances, to provide users and application domains with common information and organisational structure that they will need.
2. Concepts that are examples to help those developing the use of the DARE platform, for themselves or for sub-communities.
3. Support for consistent structures, particularly for handling Collections, to provide optimisation opportunities and to steer users to well supported or efficient methods.

The conceptual library will also present an API with Python methods for *basic* Actions to change the state of the DKB. These include:

1. Actions on Concepts to define them and their relationships, evolving them and managing their lifetimes.
2. Actions on instances of Concepts, creating them, finding them, interrogating them and updating them.

As the definitions of refined Concepts and sophisticated Contexts develop new Actions specific to particular Concepts and methods composing basic Actions will be introduced. It is intended that user communities will become self sufficient, initially in using the basic Actions and

eventually in refining them and in maintaining sophisticated Contexts and home grown Python encoded Methods. To enable this, once *published*, new Concepts, new Actions and new Methods automatically become available in the API. Eventually their use may be controlled by authorisation mechanisms, but initially we will depend on communities collaborating with careful consideration for their colleagues. This is only feasible while the user community is small.

It will be possible to build more complex and longer running Methods out of these basic Actions and other Methods, including those in any workflow system that DARE supports. Some of these may be used to manage aspects of the DKB, e.g.,

1. Promote a set of revised Concepts and instances from an innovation Context to a shared Context.
2. Import a bundle of information from an authoritative source, such as a curated ontology, creating a Context to represent it.
3. Build a Collection representing the current Python loaded libraries.
4. Compare two such Collections and report on their differences.
5. Visualise the graph of specialisations of a particular Concept.

Such Methods will be implemented when needed. When they are not simple Actions[51], they will need to record progress in a provenance service, such as P4.

It is intended that DKB users will be able to do everything that they need to do in this way, i.e., by working in Contexts, by building on provided Concepts, and by composing basic Actions and Methods built ultimately from basic Actions. Direct use of Entries will be deprecated because it reduces the manageability of the DKB. However, it will be retained to permit work arounds if unintended limits are encountered in the prefered usage patterns. At the same time, the DKB should let experts helping the application communities or within those communities develop applications and workflows in whatever way they choose and then import them into the active Contexts, so that they may be used straightforwardly. Data architects, data scientists, systems specialists and software engineers may also work on Entries once they have stabilised and proved useful, to improve their performance, reduce their costs and improve their provenance tracking. This depends on supporting multiple views and uses of the Contexts. A future development that will only be demonstrated as feasible in the DARE project.

Concepts are introduced as they reflect established ways of thinking and communicating in application and technical fields. They often have agreed names, developed in the application's culture and global consortia and corresponding to an authorised terminology[52]. For each viewpoint/use of a Concept they have properties used by practitioners. The DKB supports

[51] For example, they may fail after changing some of the persistent state, and provenance records will be needed to support attempts to complete the work or to undo what cannot be completed.
[52] These precise terminologies may draw on and implement LOD ontologies, but it must not be a requirement to understand ontologies and OWL to use the DKB. However, experts in that approach should be able to help users using their knowledge. For example, they may develop and use methods to import authorised ontologies directly into the DKB and ready for production use.

communities and individuals agreeing and using their Concepts as shown in Figure 4.7. Note that an application community will have familiar Concepts, with well understood names, that are understood by them, but not by the systems team that enables their computational and data-driven work. Similarly, those system experts have their own vocabulary, Concepts and properties that they understand. The points where these worlds overlap, called "*boundary objects"*, are vital for effective collaboration, and have to be well supported by the DKB.

In some senses, Concepts are similar to classes in object-oriented programming. Indeed users may be helped by being able to use Python classes in the programs corresponding to the Concepts they are using, with instances of each class corresponding to instances of Concepts. *The extent to which the DKB system will automatically support this has yet to be decided*.

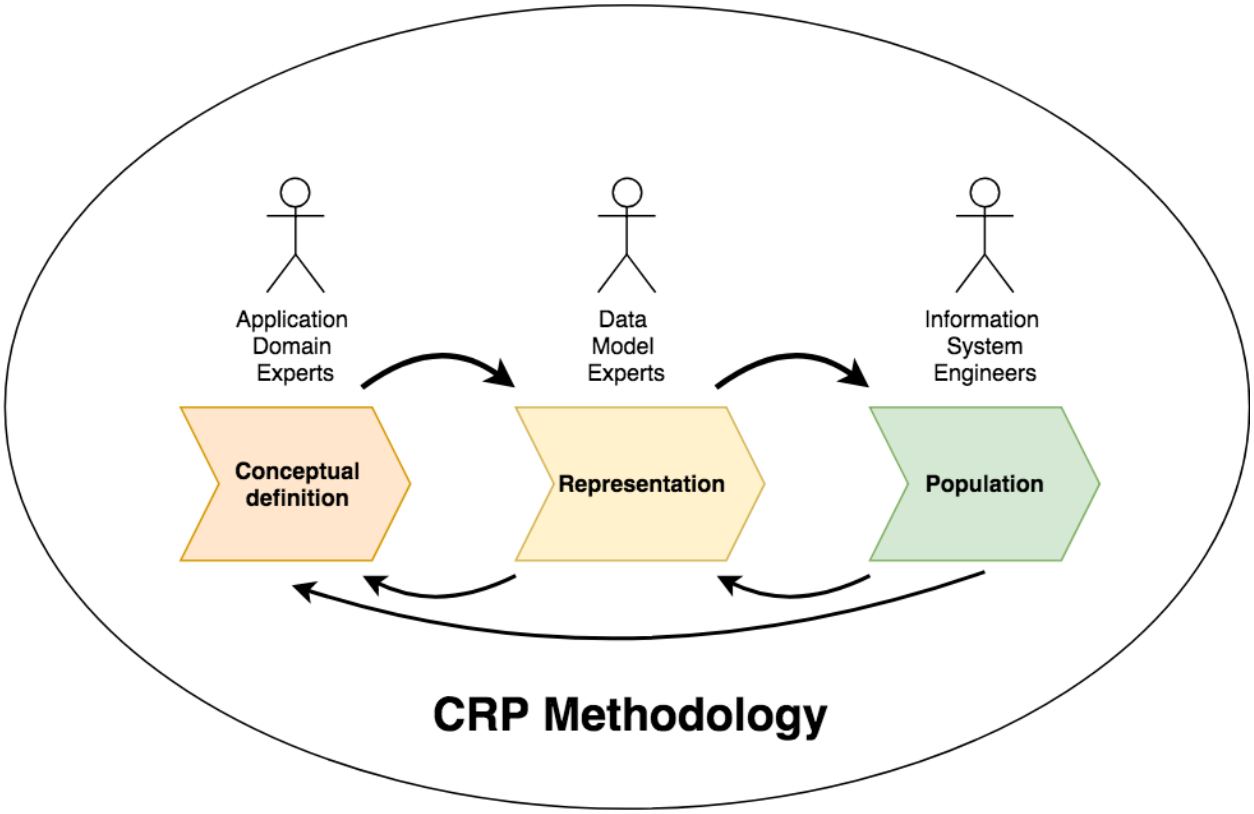

Figure 4.7: Showing the CRP methodology to develop and manage concepts shared by a collaborating research community (taken from Fig. 5.2 [Trani 2019]).

Having decided what Concepts they need, users choose the properties that they consider important, and how they would like them to be represented consistently. We envisage three categories of attributes:

1. *Mandatory*, ones that must appear in every instance as well as those already there because of the common Entry attributes. This has the effect of specifying the name[53] and

[53] The naming conventions are similar to those for Python, i.e., Concept identifiers begin with a capital letter and attribute identifiers begin with a lower-case letter.

the form of each attribute. It will trigger an error if an instance is made which does not have the attributes in the correct form.
2. *Recommended*, are properties that should be present in every instance, this has the effect of specifying the name and form of the attribute and may have the effect of prompting users to supply values for these attributes in each instance.
3. *Optional*, is a category of attributes that may be included. This specifies the name and form to be used whenever an attribute appears in this Context with the given name.

Given our philosophy of trusting users and their software, the instances may still include additional attributes in the same way as Entries do. This permits users to impose required structure while still facilitating experiments leading to innovations. Each attribute is either a property denoted by a suitably formatted literal value or a relationship referring to another Entry denoted by its PID. Listing A2.1 introduces a notation for defining a Concept in terms of its inheritance and attributes reflecting the style of Python class definitions. Concepts, e.g., SpecialWidget may inherit from the definition of another Concept, e.g., Widget. In that case, instances of SpecialWidget may appear whenever or wherever instances of Widget are required.

Listing A2.2 illustrates the introduction of two Concepts using this notation. A possible representation in the DKB is illustrated in Figures A2.1 and A2.2. Listing A2.3 then creates instances of those Concepts to produce a structure in the DKB shown in Figure A2.3. Available operations on Concepts additional to or revising those for Entry are given in Table A2.3, which is partitioned into 6 parts for the 6 Concept categories and each of those is slit into the functions currently being implemented and those that may be needed later.

Building all the required Concepts from scratch is laborious. Therefore, DARE offers a rich library of predefined Concepts shown in Table A2.4.i, partitioned to illustrate six categories:
1. *Concept definitions*, which may be extended and refined as users wish.
2. *Method definitions*, that include the available built-in methods Table 2.4.ii. Table A2.5.ii tabulates the currently anticipated actions on Method instances.
3. *Data definitions*, that include the Data Catalogue and its semantic extensions Table A2.4.iii. Table A2.5.iii shows the currently available on Data instances.
4. *Collection definitions*, that are critical for supporting a common user and software activity of building and using collections of things, Table A2.4.iv. Table A2.5.iv outlines anticipated actions on Collection instances. Virtual and lazy implementations of Collections are essential to handle the growing size of data and Collections many application communities now deal with. These increases will continue so these forms of Collection will become essential. DARE has special facilities for handling these, e.g., transmitting and processing them incrementally in data streams. Nevertheless, application experts and research developers will need to directly control their formation, content and use. Synergy between data engineering and distributed workflow/computational systems engineering will be needed. By making Collections

first-class citizens the DKB will facilitate that synergy. That is, they are a primary example of boundary objects.
5. *Sundry definitions*, that are needed in conjunction with the above, with Contexts and with organising people, processing and systems Table A2.4.v.
6. *Built in types*, The categories of data, such as String, Real and Time that are imported from underlying systems, such as Python Table 2.4.vi.

There are substantial conceptual, organisational and practical advantages from delivering in the library a harmonised bundle of Concept, Method, Data and Collection Entries. We anticipate each platform release will include advances in this bundle. Other categories of Concept may emerge. Please consult Appendix 2 for details and examples and [Atkinson & Levray 2020] for developments after the publication of this document.

4.2.4 DKB contemporaries

These include the data catalogue and the Registry. They also include some aspects of the P4 provenance handling framework described in §4.3. These provide two valuable aspects to the design and development of the DKB:

1. As well-established and populated subsystems they provide a significant part of the required functionality.
2. They also ensure that the DKB does not require a ‘green field’ operational context, as cooperating with complex contemporaries is essential for long-term adoption.

Data Catalogue

The DARE data catalogue has been operational throughout the platform’s development and provides basic services for recording information about the files in use. This is currently being enhanced and extended as the DARE Semantic Data Discovery Service.

The Semantic Data Discovery Service builds up on the Data catalogue component of the DARE platform. The Data catalogue stores metadata about datasets in DARE described with the Resource Description Framework (RDF) model, which conforms to the Data Catalogue Vocabulary (DCAT). Currently there are only low-level interfaces to these datasets, provided by the RDF database Openlink Virtuoso (SPARQL, ODBC, JDBC, ADO.NET, OLE DB, etc.). The complexity and low-level nature of these interfaces inhibits the full use of the Data Catalogue’s information.

The Semantic Data Discovery service should enable a user to access the data stored in the Data Catalogue conveniently. To achieve this goal the application scans recursively through the existing datasets, indexes all known information patterns found and provides an interface to search this data. The Python code provides functions, such as trigger indexing, deleting the index and starting a search. These are accessed via a REST API using the Python web

development framework Flask[54] and are exposed by OpenAPI Swagger[55]. To create, manage and use an index, the search engine Apache Solr[56] is used. This offers a wide range of functions including: simple text-based search, a search by date or by geo-location. It is easy to add vocabularies[57] to meet the need to be open-ended, as a wide range of specialised linked data vocabularies may be used. The implementation architecture is shown in Figure 4.8.

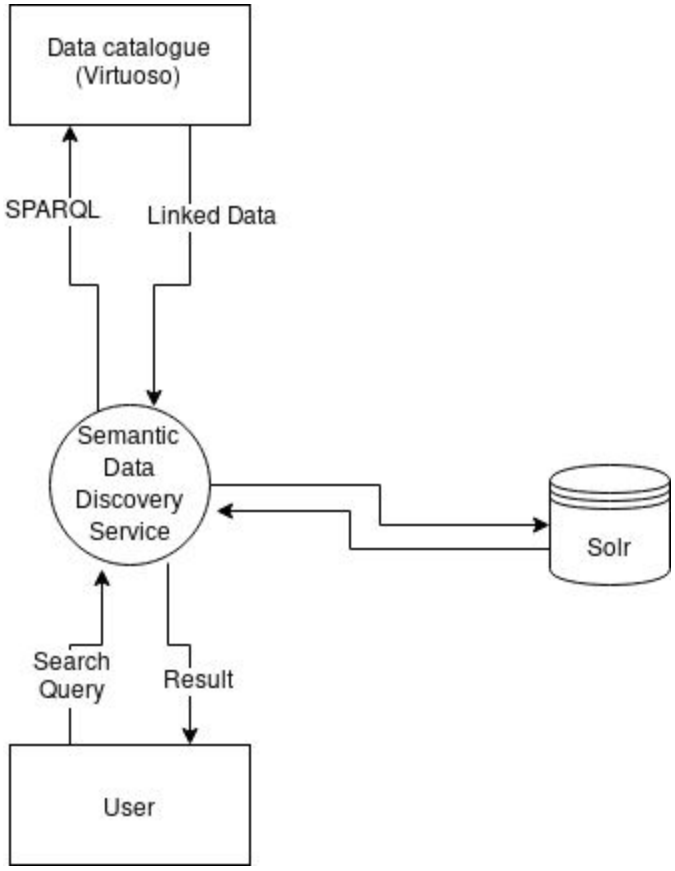

Figure 4.8: The architecture of the Semantic Data Service extension to the Data Catalogue.

In the future, a web GUI should be placed on top of the search, for example comparable to the European Data Portal[58]. An integration with the provenance database is also planned. A further extension could be the integration of external Data catalogues to meet known requirements for the climate or seismology communities. This can be done as a change of configuration, as long as the data exists with the vocabulary DCAT and are provided through a SPARQL endpoint.

[54] Flask https://www.fullstackpython.com/flask.html
[55] Swagger https://swagger.io/docs/specification/about/
[56] Apache Soir https://lucene.apache.org/solr/
[57] This is done using a configuration file.
[58] https://www.europeandataportal.eu/

Registry

The registry is a part of the DARE platform that coexists alongside the DKB. As do the provenance system and the Data Catalogue. The previous item discussed the relationship between the Data Catalogue and the DKB. Here we focus on the Registry's relationship with the DKB. We then consider the relationship with provenance.

The registry was developed in the VERCE project[59], where authors of [Klampanos *et al.* 2015] prototyped the dispel4py information registry to facilitate consistency and collaboration in workflow development[60]. Consequently, the current state of the registry is strongly linked to dispel4py. The current version is implemented in Django[61], a Python-based Web framework and is linked to a relational MySQL database server. The current usage allows for the development of workspaces, and the storage and production of information regarding workflows.

The registry is currently used in the DARE platform via an API, to register workflows, (i.e., register PEs of the workflow). It also can execute and monitor the runs of the workflow, i.e., use the provenance system to stream provenance traces at run time. The main functions of the API are given in Section 3.3 and recalled in Table A2.8.

As mentioned, the registry deals with the notion of workspaces. In its design, workspaces refer to a snapshot of whole sets of components linked to registry (including the registry) to allow for refined, specific, user-defined context work. In that sense, the idea of workspaces in the registry is closely related to the definition of contexts in the DKB.

In order to make use of the DKB and registry together, it is critical to link those two definitions. It is also important to add more specificities either to the registry or to the DKB so that its uses cover more than dispel4py. Indeed, the idea is for users and developers to be able to register methods in any format they desire, including: bash script, python script, CWL, dispel4py, etc.

At the present stage, we can integrate the registry in the DKB as a proxy server to access information of methods (PEs, workflows, runs).

The registry is also mentioned in the following sections:

- In §3.3: the use for development, where we can clearly see that developers and users make use of the registry API to register dispel4py workflows.
- Which is also expressed by §4.1, that describes how to use the registry for dispel4py workflows. For example, a method must be registered in the registry before it can be run on the DARE platform.

[59] http://www.verce.eu
[60] More information can be found in https://zenodo.org/record/3361395#.XfjPVOvgrUZ.
[61] https://www.djangoproject.com

Relationship with P4

The provenance system runs a database into which it collects information from many runs and from different technologies into a standard form [Spinuso 2018]. It also collects metadata associated with runs specified by users, supports tools for examining and visualising these records. It can be accessed by a set of web-service functions including specific requests to export selections of its data in standard formats. Consequently, it is performing the role like that of the DKB: there are therefore two directions of development to consider:

1. *Independence*, each develops independently and users/developers tackle the integration of their functionalities. Even with this approach they still need to coordinate, cross-reference and align their treatment of entities that they both handle.
2. *Integration*, where they converge through co-design, so that eventually users and developers see them as one system that they use unaware of the two subsystems.

Integration is clearly the desirable long-term goal, but it is so challenging that a period[62] of *independence* but *convergence* is needed before it is attempted. The necessary *interdependence* will need to progressively deliver:

1. *Coordination*, e.g.,
    a. notification to P4 from DKB when it starts an Action that needs provenance,
    b. notification to the DKB and P4 when the WaaS detects completion or failure.
2. *Cross-reference*, the following examples indicate what will be needed:
    a. References DKB can use to refer to traces in P4, preferably PIDs
    b. References (PIDs) P4 can use, to refer to entities identified via the DKB, preferably with known fixity.

    The timing of PIDs being allocated and being used in each subsystem, as well as the decisions as to which entities require PIDs, will need to be worked out as the alignment is developed.
3. Alignment of representations, such as:
    a. Identification of individuals and representations of attributes both use.
    b. Identification of sessions and representations of attributes both use.
    c. Ditto for computational environments, containers, deployments, software, data and services, etc.

This convergence should be achieved incrementally in co-design and co-development closely aligned with pressing requirements from use cases, e.g., as identified in §3.4.

4.2.5 DKB R&D planning

The current status of the DKB is that there are three mature, operational subsystems: data catalogue, registry and the provenance system independent from one another and a prototype general-purpose DKB that has yet to contribute to the DARE platform's use. It is clear that

[62] Almost certainly longer than that remaining in the DARE project.

users, particularly research developers, would benefit from increased automation and integration (see §3.4). As explained above, the DKB should help those extending and developing data-intensive and computationally demanding applications in the context of existing systems and established practices. This is precisely what DARE provides and we should therefore use the opportunity to develop the DKB:

1. To help the research developers in our user communities and in successors to DARE, and
2. To use DARE to progress to the point where the integrated version of the DKB has proved to be useful and feasible, so that it is both sustainable and a good foundation for future work.

This requires decisions on:

1. The information needed for the automation priorities.
2. The information needed for optimisation, deployment choices and target flexibility.
3. The best ways of presenting DKB information to research developers as they program.
4. Their preferred was of recording information that interests them.
5. How to fluently combine innovation and production.

This in turn requires development decisions leading to inclusion in platform releases.

1. How best to develop and deploy the semantic extensions of the data catalogue.
2. How to align Contexts in the DKB with workspaces in the registry.
3. How to combine the provenance and DKB systems.
4. How to deliver method authoring to use simply, e.g., for Python scripts.
5. How to roll out the DKB functionality incrementally with good integration.
6. How to deploy the common Concept library and other common resources.

Detailed discussions on these topics are developed in §A2.9.

4.3 The P4, tools and interaction interfaces

P4 includes the provenance functionality that enables the acquisition and exploitation of provenance data. DARE focussed initially on capturing lineage information about the execution of a method. This is described by the initial inputs, the method's processing elements, and the computational resources used. We acquire provenance from different types of systems (CWL and dispel4py). They are mapped to S-PROV, in order to be interactively explored and visualised using DARE's S-ProvFlow tools and lineage API. The lineage information recorded from the execution of a dispel4py workflow can be tuned adopting a provenance configuration and contextualisation system developed during the first half of the DARE project [Spinuso *et al.* 2019]. For CWL, we are currently focussing on managing provenance information that is specifically addressing the execution of *specfem3d* workflows, which for simplicity of use, are bundled in dedicated calls of the DARE API. These have been implemented according to the requirements of the use cases defined in WP6. Here, we are interpreting and summarising the

CWLProv information produced by such API calls in order for it to be integrated and accessible through S-ProvFlow.

Ultimately, CWL and its provenance component aims at the generation of research-objects[63]. Pursuing this model depends on the implementation and policy of the the DKB in terms of the storage and description of the results generated through DARE. We foresee provenance data to be linked from DKB entities describing the final dataset. This will require the proper generation of the results' PID and the reference to the endpoint that will extract from P4 (s-ProvFlow) the full lineage trace associated with the data product. This would comply with the RDA indication for a pattern enabling the linkage between metadata and provenance (Figure 4.9).

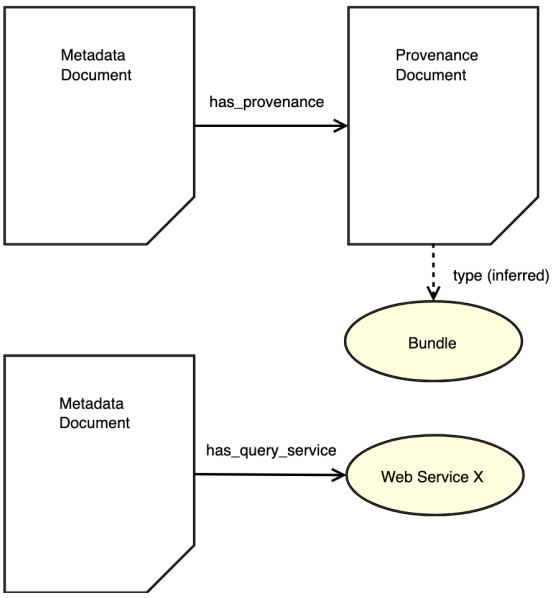

Figure 4.9**:** Schematic representation for referenced inclusion of provenance information within metadata document or in structured data files (such as NetCDF). Formal provenance can be linked as external self-contained documents or provided by external services. The approach fosters decentralised management of lineage information, which can be queried to extract and combine information about the software dependencies consistently. We should consider though to include a textual and human readable summary (image extracted from the RDA provenance pattern database[64]).

Once the lineage has been stored we provide three kinds of exploration functionalities. Live monitoring, Lineage queries, Discovery and comprehensive visual Analytics. These have been described in DARE literature [Spinuso *et al.* 2019, Atkinson et al. 2019, Klampanos *et al.* 2019] from the point of view of their conceptual functionalities and integration within the platform. More technical insights are provided within the DARE deliverable produced by WP3.

[63] Research Objects http://www.researchobject.org/
[64] https://patterns.promsns.org/pattern/12

Current developments are addressing improvements to the lineage query methods. We aim at providing a more usable and powerful set of methods that will allow users to search over combinations of terms' using value-ranges or value lists. These improvements will be reflected on the s-ProvFlow viewer too. In Figure 4.10 we show the prototypical interface that allows users to search for workflow executions with a short explanation of the new syntax.

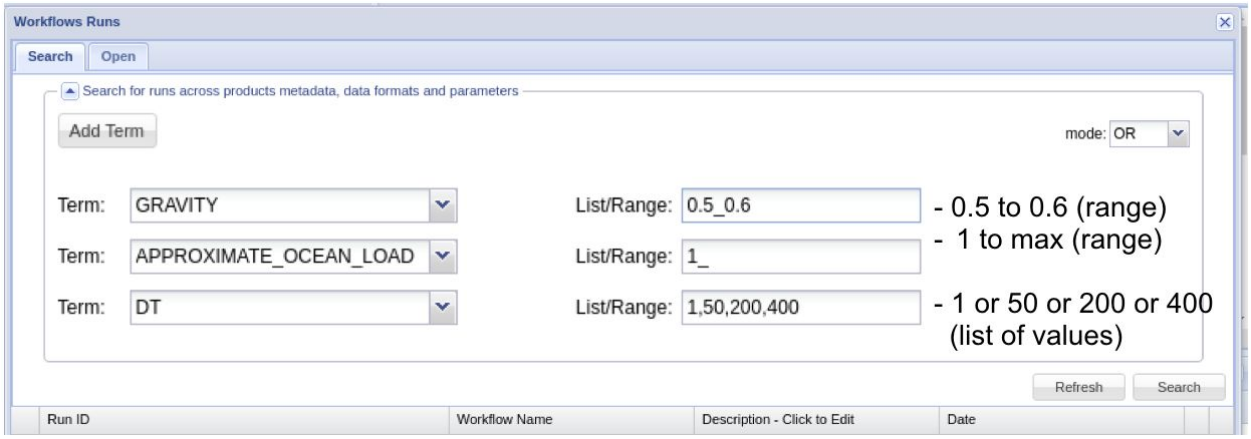

Figure 4.10: Mockup of the improved MVV GUI that allows to search for workflow executions adopting multiple metadata and parameter value-range and lists. The expressions are explained on the right side of the input box. The query terms are suggested among the ones present in the user's collection of runs, as already introduced in D3.7 [Spinuso & Klampanos 2018].

Recent developments have been largely dedicated to the integration of security mechanisms for the authorised access and storage of the provenance data (AAI). This is pursued in a way that meets the GDPR regulations in terms of the separation, "by design", between the recorded lineage traces and any deducible information about the users themselves (e.g., username, email, identity). In this respect P4 relies on the DARE AAI (see §5.2) and the DKB (see §4.2) for the complete resolution of users' personal information. Implementation details will be reported in the official deliverables about the platform deployment in WP5 and the updated lineage services in WP3.

Aspects of P4 also concerns direct interaction with the setup and incremental customisation of development environment, for instance based on Notebook services such as Jupyter. The KNMI is working on a new API, SWIRRL[65] (Software for Interactive and Reproducible Research Labs). This work is conducted in the context of the ENVRIFair project in close collaboration with DARE. The API automatically manages the deployment of a computational environment offering integrated notebook, execution of workflows and visualisation services. The provenance information describing the API actions for the creation and updates of the environment, the execution of data-staging workflows and the generation of repdocuble snapshots are captured within formal provenance documents. These are stored within a dedicated database and made accessible through the SWIRRL API. Ultimately, the provenance data will allow users to trace

[65] https://b2drop.eudat.eu/s/ENLJZmxxt2q2MNL

the evolution of changes within the environment itself and to restore past setups. The latter action may include data and notebook pages, according to user's needs, fostering reproducibility and sharing of research progress among peers. Aiming DARE at being fully controllable through such interactive notebooks, we foresee great potential for the integration of the services offered by the two projects, with a particular focus over reproducibility and traceability of the research progress. This will extend the period of support and amortise support over a much wider community, as required for sustainability - see §5.

4.4 Analysis, Conclusions & Summary

As reported in §3 the DARE platform delivers significant new power to our user communities. However, DARE does not have the resources to run these at scale as frequently as users require. As the DARE-enabled methods are polished ready for productive use and as courses are run to expand DARE's user community, this becomes ever more pressing. It is therefore urgent to facilitate DARE platform deployment on eInfrastructures user communities can gain access to, such as institutional facilities. This requires enabling relatively inexperienced operational teams to set up, run and help DARE users with a local instance of the DARE platform. This priority is revisited in §5.2.

Currently the three subsystems on which DARE depends are not interworking as effectively as they need to. A protocol for using a message handling system, e.g., ZeroMQ[66], and agreements on what events warrant messages and what those messages should contain is urgently needed.

Reviewing the individual subsystems, the following developments are necessary.

The *WaaS* has a powerful system but exclusively for dispel4py.

1. It needs to accommodate others, at least plain Python scripts, and perhaps CWL or other notations.
2. It currently requires too much input from workflow and system experts along the path from method creation to method deployment. This needs to be overcome by automation and good defaults.
3. DARE is meant to push scalability limits in three dimensions, yet the optimisation of mappings, and the informed choice of targets for enactment has still to be implemented.
4. Adaptation to different to different target architectures, different data handling services, different distributions of data, and different communication channels is needed.

All of these will help users and help with sustainability.

The *DKB* is currently fragmented and much of it is in prototype form only. The following priorities need to be addressed with improved alignment and coordination.

1. Alignment with the registry including reconciling workspaces with Contexts.
2. Synergy with the data catalogue.

[66] ZeroMQ https://zeromq.org/

3. Delivery of Contexts or their equivalent.
4. Enabling convenient update by a wide range of researchers and software for their own purposes.
5. Releases of the conceptual library.

The *provenance system, P4*, is fully operational, however it needs to consider the following issues:

1. Making provenance configuration easy, flexible and well supported.
2. Stimulating the use of the integrated metadata and provenance by improving query support and the tools that exploit it.
3. Widening the range of resources from which provenance is collected.
4. Enabling technically capable users to add more provenance collectors.
5. Establishing agreed cross-reference models with PIDs with other systems, including the WaaS and DKB.

There is also an ethical challenge we face. The purpose of the platform is to help individuals, teams and wider federations pursue complex research campaigns by pooling their ideas and efforts, i.e., we must support their collaboration, i.e., the DARE platform should have a CSCW role. But that depends on individuals recognising each other in the system, acknowledging each others contribution and respecting each others wishes. This cannot be done if identities are hidden. The following is a suggestion. We should either adopt this or propose an alternative.

Trivial initial use, e.g., by a citizen scientist testing whether an instance of the DARE platform would help them, should be anonymised as currently implemented to comply with personal data regulations. Extensive and long running use would require that a user has gone through a process that gives them a personal PID, e.g., on ORCID[67]. Any such personal PID allocation procedure would suffice. As allocation of the PID will have included establishing informed consent to their identity being public in an ethically and legally acceptable way. We can then use their PID and reference and use information from that information source. I cannot imagine the powerful visualisations of provenance being useful if the person(s) behind all that work is anonymised. At least, not when we reach larger scale consortia.

[67] ORCID https://orcid.org/about/what-is-orcid/mission

5 Future and Sustainability

Sustainability is crucial for our partners using the DARE platform. They are developing new methods and working practices that depend on the platform. They would suffer severe disruption if the sophisticated software and integrated systems were not supported after the DARE project concludes. There are three aspects to achieving such sustainability:

1. *Minimising the cost and effort* required by using shared systems, standard software and careful engineering; but the cost can never be made vanishingly small - this is addressed in §5.2.
2. *Building the commitment to invest* in the required maintenance by developing expertise and advocates across the user communities - this is addressed in §5.1.
3. *Amortising the costs* widely by expanding the user communities and the number of application areas, organisations and funders who contribute. This depends on and contributes to an improved *return on investment* (RoI); a bootstrap challenge - see §5.1.

The drive for sustainability influences every aspect of DARE's work. Sustainability has long been understood as a pressing issue for software, e.g., quoting an ENVRI report accepted by DARE's two RI communities, EPOS and IS-ENES.

> "**Software sustainability**: ... The decision to depend on software is as important as the decision to depend on an instrument and it should be taken equally carefully. This will lead to an identified list of mission-critical software. Each RI ... should establish mechanisms for determining that critical list. The list should be minimised by careful use of commercial and *well-supported* open-source software. The members of the residual list of software must be maintained or replaced throughout each RI's lifetime. This requires appropriate resources, particularly software engineering staff and processes with appropriate quality controls. Wherever possible these should be met through alliances." (From Section 5.2 "Impact on Stakeholders" p193 [Atkinson *et al.* 2016])

One source of underfunding for sustaining critical research software is the lack of realisation of the costs involved, as most people do not have experience of software going into production and being used by multiple users, for many purposes, some not originally envisaged, running in many different and changing computational contexts. The support needed is software maintenance and provision of help to installers and users. Maintenance includes: bug fixes (~10%), accommodation of computational context changes (~50%) and enhancements (~40%).

The lack of research-software sustainability, led to the establishment of the Software Sustainability Institute (SSI)[68]. Its mission is to change the culture so that the Research Software Engineers (RSEs) making and sustaining well-engineered software are respected and resourced. SSI now delivers global leadership for this cause.

[68] Software Sustainability Institute (SSI) https://www.software.ac.uk/

Taking these viewpoints into account, we identify the critical sustainability steps. We combine the viewpoint of RIs and their user communities with the viewpoint of research developers and platform engineering teams.

Research Infrastructures[69], represented by WP6 and WP7 in DARE, need to:

1. Establish an agreed process by which they *decide on which software they will depend on*. This has to balance two factors:
    a. Research agility enabling them to explore new ideas which may depend on and develop new software, and
    b. Dependability from using established software and limiting new software to that which they know they or others can maintain.

    This requires continuous governance and operational procedures. Allowing experiments and exploration, but filtering which are carried through to production with the concomitant obligation for long-term support.
2. Develop a *mutual-respect* ethos when interacting with RSEs, expecting professionalism to develop in software and systems engineering as it does in their own discipline.
3. *Share the responsibility* of finding resources in the short, medium and longer term.
4. *Actively develop broad adoption* of the software they choose to depend on.

The builders of the DARE platform and the research developers pioneering new uses take on the following responsibilities as they proceed:

1. *Minimise the use of bespoke software* so as to reduce the sustainability burden.
    a. This requires re-engineering once requirements and solutions are understood to eliminate the effects of agile processes delivering quick solutions.
    b. This requires broadening the functionality of key elements to avoid additional elements, to take over from bespoke software and to extend amortisation.
2. Use *existing well-maintained software* whenever possible, and build any new software with well-disciplined professionalism.
3. *Deliver self-sufficiency* through intellectual ramps; users start by using your provided solutions with their anticipated variations, but can, with modest effort move to more radical variations when they need to.
4. *Reduce to a minimum* the elements in a platform, subsystem or software stack that is included in its sustainability phase.
5. *Document* with guidelines, patterns, technical information and tutorials the minimum from step 4 for each role that will be involved in use or maintenance.

5.1 User communities and engagement

DARE Sustainability pillars:

[69] Both EPOS and IS-ENES, in conjunction with their related global and long-running campaigns, recognise the importance of software for their research and have relevant resources.

1. *analysing the market sectors where the results may be applied,*
2. *identifying self-contained DARE outcomes that are exploitable, and*
3. *identifying the appropriate distribution channels to promote and sustain the different results of the project.*

5.1.1 Market Sectors

DARE seeks to address needs concerning:

1. the **European domain specific eInfrastructures** (such as EPOS, IS-ENES2, ICOS65, SKA66, etc.) which can exploit DARE in order to create new data-driven services more easily.
2. **science and technology professionals** that can use DARE-powered infrastructures more easily, without being concerned with technicalities, enabling them to focus on how to improve their methods, results, synergies and innovation potential, and
3. the **"long tail of science"**, including research institutes, research teams, individual researchers, SMEs, etc., who due to lack of tools, methodology or resources, are unable to make the most of even today's wealth of data, scientific advances and the power of the e-infrastructures Commons.

DARE will support this last category of users by enabling them to design and implement experiments, services and products that can still be supported by the same local, small-scale infrastructures but have the inherent ability to scale up to exascale resources, when such a need arises. A crucial element of this sustainability is increasing DARE users' self-sufficiency by reducing technical hurdles, establishing intellectual ramps based on adaptable examples, and delivering self-education aids, such as online courses.

Especially for SMEs, we will initiate activities fostering SME engagement through targeted focus groups, workshops and other events aiming to increase awareness, enabling SMEs to access the knowledge they require.

5.1.2 Identifying self-contained DARE outcomes that are exploitable,

Asset 1: dispel4py A high-level data-streaming dataflow specification API, reusable components and Python library.
Asset 2: S-ProvFlow A set of tools and components in support of Reproducibility as a Service (RaaS) based on and delivering W3C provenance standards.
Asset 3: Exareme A system for large-scale dataflow processing on the cloud.
Exareme provides SQL and Python interfaces, that can readily federate queries over relational DBs. It supports data streaming and on-line compression.
Asset 4: Semagrow A linked-data query federator over remote sources, enabling complex querying in a transparent and seamless way.

Asset 5: BDE platform A Docker-based, cloud-ready and modular integrator platform, bringing together commercial and research, production-ready components for big-data analytics.
Asset 6: DARE Hyper Platform An integrated operation-ready, Cloud-based platform to advance data-driven agile innovation at the extremes of modern science.
Asset 7: DARE Registry An automatically populated semantic registry covering DARE methods, reusable method fragments and patterns, tools, data, context and e-infrastructures information.
Asset 8: DARE Methodology A set of principles, guidelines and best practices that enable innovators to create exascale products and services using the DARE approach.
Asset 9: DARE EPOS pilot An operational EPOS-IP use-case pilot meeting societal requirements for producing innovative earthquake risk assessment and response.
Asset 10: DARE IS-ENES pilot An operational IS-ENES use-case pilot meeting societal requirements for producing innovative climate-change risk assessment via the Climate4Impact portal.

5.1.3 Distribution plans

Initial analysis reveals appropriate distribution channels to promote and sustain results from the DARE project.

1. Structural engagement with EOSC

Services offered via EOSC portal, EOSC public project, EOSC channels to publish on-line deliverables and various kits of information, EOSC project in the EOSC Landscape Analysis Report.

2. Partner Assets Exploitation Intentions

- NCSRD #4, #5, #6, #7, #8
  Creation of opportunities for internal, national and international collaborations across neighboring scientific fields and promoting technologies that have been developed, such as the big data management infrastructure technologies and the Semagrow engine.
- INGV #6, #7, #8, #9
  The EPOS Use Case aims at exploiting the powerful services and tools of the DARE platform to agilely handle big data HPC simulations and analytics in the solid-Earth field with a strong impact also for societal emergency contexts.
- UEDIN #1, #6, #7, #8
  Supporting technology in many other contexts, including research contexts with environmental, Life, and Astronomical sciences. This will include grant-funded work and commercial work through EPCC and the Data Technology Institute.
- CERFACS #6, #7, #8, #10
  Promoting the use of the tools and the DARE platform within the climate research domain.
- KIT #6, #7, #8, #9

The technologies developed will be essential to many other areas in solid-Earth research. Especially in the area of seismic hazard better quantification of uncertainties are needed to be able to use advanced decision support systems that are heavily relying on robust and quantifiable information uncertainty.

- KNMI #2, #6, #7, #8, #9, #10
  Repurposing of basic services within an advanced computational platform to foster the realisation of innovative and sustainable products in line with the KNMI duties and R&D activities, supporting experimental data-driven research in the context of the newly established KNMI-DataLab. Extending RaaS in the ENVRIfair and Copernicus communities. Continuing the curation of Concepts in the EPOS context.
- GRNET #6, #7, #8
  Exploitation of the outcomes by deploying them into production, thereby expanding the company's portfolio of offered cloud and HPC services to its end-users: research and academic community including Earth Sciences, Life Sciences and Engineering.
- FRAUNHOFER #6, #7, #8
  Knowledge gained through the project and parts of the solution, especially on service integration, Cloud services and automatic deployment methods, will be utilized for in-house solutions that are provided to scientists and developers; newly gained expertise will flow into further research projects and to let other scientific disciplines benefit from the results.
- ATHENA #3, #6, #7, #8
  Transfer of results towards further development of the infrastructures it participates in: OpenAIRE, CLARIN, META-SHARE, ELIXIR-EXCELERATE and HELIX.

Specific Priorities and actions

- Training Kit:
  - Best Practice guidelines for DARE in high tech industry and in particular for SMEs
  - Handbook for *Getting Started*
  - Methods, tools and techniques for working with target groups

- Packaging and releasing software
  - offer each of WaaS, the persistent provenance system RaaS and the DKB as separate services
  - offer the framework for composing them as a Software framework that others can adopt and adapt, optionally replacing some of the DARE components
  - offer the abstracted general purpose DARE platform with one or both of two tailorings for user communities developed within DARE
  - DARE platform *https://gitlab.com/project-dare*
  - Design and implementation of the components of the DARE toolkit following a Microservices approach and Virtualization techniques on multiple levels, *i.e. Kubernetes Container Orchestration* see §5.2.

- Promoting DARE assets
    - Social media including: blogging, tweeting and ResearchGate
    - free promotion and recruitment and training of active advocates
    - developers events (webinars, hackathons)
    - standard software engineering documents and code commenting tools: doxygen, class diagrams,
    - ü Ambassador Programme (chance to build adoption in other communities, networking opportunities, etc.)

5.2 Individual and Combined services

Deployment and operations effort required can often present an obstacle for the adoption of new platforms like the DARE toolkit. Therefore, care has been taken to design and implement the components of the DARE toolkit from the start in a way that eases the burden on IT-administrators and self-empowers and non-expert users to be able to run and manage a DARE deployment without the help of IT-operations experts. Using microservices facilitates deployment of the DARE platform when a community wishes to select some of them. In these cases and for the complete platform virtualisation on multiple levels assists with deployment.

The components, as presented in the preceding sections, are designed as loosely coupled, concise services. Communication / Interworking is realised through APIs, mostly based on REST interfaces using JSON responses, allowing for distribution and horizontal scaling of the services (e.g. through the automatic functionalities of Kubernetes[70]). DARE is undogmatic when it comes to the microservices philosophy, though. Where it was required to directly address user needs, for example, deviations from this approach are made. For example, to support some simulation codes that use the Message Passing Interface (MPI), more tightly coupled interdependencies are required, e.g. a shared POSIX filesystem between the Kubernetes pods that take part in the computational job. However, care has been taken to address these cases with native Kubernetes functionality, e.g. in this case with a Kubernetes Operator.

To avoid duplication of effort and sustainability challenges after the project's end, well established components have been used wherever possible. Where multiple implementations where available, well-known and widely used community-supported solutions have been preferred. Such components include MySQL, nginx, Virtuoso, Keycloak and Kubernetes.

APIs of the DARE components are designed to be as simple as possible, preferably as REST APIs using JSON for communication. API documentation is available in the form of OpenAPI/Swagger descriptions. Where available, (pseudo-)standards have been used, such as W3C-Prov, DCAT-AP and CWL.

As has been described before, the DARE components make use of operating-system level virtualisation. They are distributed as Containers, and their deployment is managed using the Kubernetes Container Orchestration system. Where possible, existing community-maintained container images are used to avoid replication of work. For custom containers, care has been taken to follow best-practices such as relying on community-maintained base images, carrying

[70] Kubernetes https://kubernetes.io/

only necessary software in small containers running only single applications, using APIs between different applications instead of interweaving, etc, which reduces the effort for maintenance and operations. For deployment, Kubernetes descriptors have been prepared that allow selective as well as collective deployment of the DARE components.

The above described containers and descriptors allow easy deployment on existing container infrastructures, such as managed Kubernetes Clouds (e.g. Google Kubernetes Engine). On top of that, the DARE project provides tools to ease the deployment on IaaS-Clouds such as Amazon and the EOSC-provided IaaS Clouds (e.g. EGI FedCloud). By using the Terraform[71] tool, DARE can provide Infrastructure-as-code level configuration files that allow automation of deployment on various Cloud technologies, both managed and on-premise, e.g., locally installed OpenStack Clouds. For this approach, DARE relies on the work of the Kubernetes project Kubespray[72] to automate the deployment of Kubernetes on Cloud infrastructures and then deploying the DARE components on top.

Most of the DARE components can be used separately as well as in combination as a part of its design philosophy. The components should be as independent as possible, but allow for strong synergy effects when used in combination. For example, the provenance system can be run independently from the rest of DARE as an autonomous system. However, there are many advantages when it is used with other components. As described in §4, when used with dispel4py for example, a lot of provenance information is collected and recorded automatically. Another such case is the Search & Discovery service, which can but does not need to make use of the information collected from the Provenance service. In this way, DARE encourages further use of its components to improve long-term sustainability.

Authentication and Authorisation

Including support for Authentication and Authorisation in the DARE platform is a particular challenge due to the dual objectives described above, as they result in partly contradicting requirements. To give just one example: while on the one hand, local deployments (on-premise) should be independent of external services and separate components should be usable without too many dependencies, typical requirements of community-driven hosted services go in the opposite direction. Users would like to be able to use their existing accounts to log-in and want to benefit from Single-Sign-On solutions instead of entering their credentials multiple times. For this purpose, community-driven infrastructures like the ESFRIs often employ or are on their way to implementing the strategy depicted in the AARC Blueprint Architecture[73]. Examples include DARIAH[74], EPOS[75], LifeWatch[76] and many others. This is also the model that the EOSC-portal[77] is currently using and the proposal for the EOSC AAI from EOSC-Hub[78,79] project.

[71] https://www.terraform.io/
[72] https://kubernetes.io/docs/setup/production-environment/tools/kubespray/
[73] https://aarc-project.eu/architecture
[74] https://wiki.de.dariah.eu/display/publicde/DARIAH+AAI+Documentation
[75] https://aarc-project.eu/aarc-in-action/epos/
[76] https://wiki.geant.org/display/AARC/LifeWatch+-+Pilot+Overview
[77] https://eosc-portal.eu/
[78] https://www.eosc-hub.eu/
[79] https://confluence.egi.eu/display/EOSC/Authentication+and+Authorization+Infrastructure+-+AAI

To strike a balanced compromise, WP5 has evaluated multiple standards (OpenID Connect[80], SAML2[81]) and available implementations (among them Keycloak, Unity IDM, Perun, Shibboleth) and has finally decided to implement a solution based on the Keycloak Open Source Identity and Access Management solution[82]. Keycloak is a widely supported Open Source solution with backing from Red Hat, as it forms the upstream project of their commercial Red Hat Single Sign-On solution. It allows Identity Brokering, acting as an AAI Proxy, Single-Sign On, as well as local user databases. With SAML, OAuth2.0[83] and OpenID Connect, it supports the most important Standards, allowing easy integration and wide compatibility with standards-compliant software and infrastructures (such as EOSC). Additionally, client adapters for multiple programming languages and application servers are available to facilitate integration. For the use with microservices in particular, an authenticating (reverse) proxy called Gatekeeper is available, that can be used to outsource the protocol implementation from the application to this ---proxy. Due to its popularity, a curated helm chart for Kubernetes is available and upgraded regularly on which DARE can easily rely.

On this foundation, DARE uses the OpenID Connect/OAuth 2.0 Standard with access tokens. Even though this meets the requirements nicely, there are still challenges to solve which are currently under investigation. These include pure API access (OAuth2.0 is browser-focused) and secure delegation for long-running batch jobs (OAuth 2.0 Token Exchange is under investigation[84]).

[80] https://openid.net/developers/specs/
[81] https://tools.ietf.org/html/rfc7522
[82] https://www.keycloak.org/
[83] https://tools.ietf.org/html/rfc6749
[84] https://tools.ietf.org/html/rfc8693

6 Summary, Vision and Impact

The power and usability of the DARE platform and its supported applications is already significant (§3) and is going to be extended substantially during the project's final year (§4.4). Work is already underway in each work package of DARE and for each subsystem contributing to the platform's power (§4.1, §4.2, §4.3 & §5.2). There is no doubt that this will deliver more of what our user communities need. However, there is a *critical issue that must be taken into account* as this work is planned and progresses, namely *sustainability*. It recurs repeatedly in every planning discussion with users, developers and funders. Without it being achieved our user communities will suffer and the DARE advances will be lost. Current DARE investment in and planning for sustainability was presented in §5.

With due regard for sustainability principles, engineering practices and constraints needed for sustainability the following steps will be undertaken in the DARE project's final year.

1. *Enable easy deployment* onto a wider range of hosting infrastructures so that user communities can find and sustain their required operational resources. This is needed almost *immediately* to sustain DARE availability during and after courses and to let researchers run DARE-enabled methods *repeatedly* to build statistical evidence.
2. *Introduce AAI controls* compatible with user practice and organisational requirements and use these to meet personal data privacy requirements and tailored research contexts supporting CSCW in conjunction with DARE.
3. *Increase automation and optimisation* to extend the range and durability of DARE-supported methods, to increase portability and to aid self-sufficiency.
4. Pioneer an integrated and flexible DARE KB to encourage self-sufficiency among research developers and application experts and to deliver easily sustained abstract wrapping of methods.
5. *Deliver provence-powered tools*, easily controlled by users, to incentivise the configuration and use of provenance and to improve research and operational practices.
6. Develop new and *challenging applications* for each collaborating domain and ensure that these are made accessible to a wide community with similar interests to assess their value and utility. This will provide evidence of which DARE functionalities are worth sustaining.
7. Run training sessions to stimulate the use of these methods and to conduct the initial assessment of worth, usability and cost-effectiveness.
8. Conduct a trial of self-sufficiency (the volcanic-ash modelling example - §3.3) to discover the extent to which application communities can conduct their own pioneering R&D within DARE, observing the rate of progress and the nature of the help needed.
9. In preparation for future *minimum cost sustainability* and *maximum self-sufficiency* conduct three finalisation procedures in parallel:
    a. Limit as precisely as possible the functionality the application communities need to continue after DARE and make a strong case for retaining those.

   b. Reduce to the smallest set of elements possible the platform and software stacks to be maintained *for those functionalities*; including if possible reducing the technologies used.
   c. Maximise self-sufficiency by delivering all aspects of documentation as well as tools and defaults.
10. Application researchers and system providers jointly form a committed consortium to deliver a sustained five-year future with the governance described above.

This will pioneer a new approach to supporting demanding collaborative research well.

Acknowledgements

The DARE project is a Horizon2020 project, 777413, funded by the European Union. It builds on a succession of prior EU projects, including: ADMIRE, VERCE, ENVRI, ENVRIplus, and on nationally funded projects. It depends on the coordination of e-Infrastructures by EOSC. The authors thank the research software engineers and systems teams that have delivered and tested a succession of releases of the platform, and the research developers and application-domain scientists who have worked with them to shape and demonstrate the power of DARE's approach and products.

References

[Aki & Richards 1980] K. Aki & P. G. Richards, *Quantitative Seismology, Theory and Methods. Volume I and II,* 1980.

[Anselemi & Gaujai 2009] Anselmi, J., Gaujal, B.: *Performance evaluation of work stealing for streaming applications*. In: Abdelzher, T., Raynal, M., Santoro, N. (eds.) OPODIS 2009. LNCS, vol. 5923, pp. 18–32.Springer, 2009.

[Atkinson *et al.* 2019] M.P. Atkinson, R. Filueira, I. KLampanos, A. Kourkourikos, A. Krause, F. Magnoni, C. Pagé, A. Rietbrock and A. Spinuso, *Comprehensible control for researchers and developers facing data challenges*, proc. IEEE eScience conf. 2019.

[Atkinson *et al.* 2018] M.P. Atkinson, E. Casaroti, Ewering, R. Filgueira, A. Gemünd, I. Klampanos, A. Koukourikos, A. Krause, F. Magnoni, Pagani, C. Pagé, A. Rietbrock, A. Spinuso and C. Wood, *DARE Architecture & Technical Positioning*, Deliverable D2.1 DARE project. URL https://zenodo.org/record/2613550#.Xe-l5r_grUb

[Atkinson *et al.* 2016] M.P. Atkinson, Alex Hardisty, Rosa Filgueira, Cristina Alexandru, Alex Vermulen, Keith Jeffery, Thomas Loubrieu, Leonardo Candela, Barbara Mangagna, Paul Martin, Yin Chen, Margareta Helström, *A consistent characterisation of existing and planned RIs*. Deliverable 5.1 of the ENVRIplus project URL http://www.envriplus.eu/wp-content/uploads/2016/06/A-consistent-characterisation-of-RIs.pdf

[Atkinson & Filgueira 2020] *Planning dispel4py developments*, in preparation, 2020. URL https://drive.google.com/open?id=1N6UGU8J47FHpnWaLTNpRLCAFguGIQYmanmO4pyB8ixM

[Atkinson & Levray 2020] Malcolm Atkinson and Amélie Levray, *DKB Design*, in preparation, 2020. URL https://docs.google.com/document/d/1hCyoqeB8R0Ov5ZcBZVxyCOAiOST7QEP90n37PnghxGs/edit?usp=sharing

[Bell *et al.* 2013] Bell, Ray & Strachan, Jane & Vidale, P.L. & Hodges, Kevin & Roberts, Malcolm. (2013). Response of Tropical Cyclones to Idealized Climate Change Experiments in a Global High-Resolution Coupled General Circulation Model. *Journal of Climate*. **26**. 10.1175/JCLI-D-12-00749.1.

[Casarotti *et al.* 2019] Casarotti E., Magnoni F., Pagé C., Filgueira R.,, Klampanos I., *D8.4 Training and Consulting Report I*, DARE D8.4, July 2019.

[Corcho 2019] Oscar Corcho, *Management of versions of ontologies*. Personal communication, 2019

[Doltz *et al.* 2018] Dolz M.F., Del Rio Astorga, D., Fernández J., Garcia, J.D. and Carretero, J., *Towards automatic Parallelization of stream processing applications*, IEEE Access, Vol. 6, pp 39944-39961, 2018. DOI 10.1109/ACCESS.2018.2855064. URL https://doi.org/10.1109/ACCESS.2018.2855064.

[DXWG 2020] W3C Data Exchange Working Group, *Data Catalog Vocabulary (DCAT) - Version 2 W3C Recommendation* 04 February 2020, URL https://www.w3.org/TR/vocab-dcat-2/

[Filgueira *et al.* 2017] Rosa Filguiera, Amrey Krause, Malcolm Atkinson, Iraklis Klampanos and Alexander Moreno, dispel4py: *A Python framework for data-intensive scientific computing*, The International Journal of High Performance Computing Applications 2017, Vol. 31(4) 316–334. DOI: 10.1177/1094342016649766. URL https://journals.sagepub.com/doi/pdf/10.1177/1094342016649766

[Filgueira *et al,* 2016] R. Filgueira, R. Ferreira da Silva, A. Krause, E. Deelman, and M. Atkinson, *Asterism: Pegasus and dispel4py hybrid workflows for data-intensive science*, in 7th International Workshop on Data-Intensive Computing in the Clouds (DataCloud'16), 2016, p. 1–8.

[Frigo *et al.* 1998] M. Frigo, C. E. Leiserson, and K. H. Randall. *The implementation of the cilk-5 multithreaded language*. In Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '98, pages 212–223, ACM 1998

[Garijo *et al.* 2019] Daniel Garijo, Maximiliano Osorio, Deborah Khider, Varun Ratnakar and Yolanda Gil, *OKG-Soft: An Open Knowledge Graph for Describing, Composing and Reusing Software*, IEEE eScience Conf., 2019

[Klampanos *et al.* 2019] Iraklis Angelos Klampanos, Athanasios Davvetas, André Gemünd, Malcolm Atkinson, Antonis Koukourikos, Rosa Filgueira, Amrey Krause, Alessandro Spinuso, Angelos Charalambidis, Federica Magnoni, Emanuelé Casarotti, Christian Pagé, Mike Lindner and Vangelis Karkaletsis, *DARE: A Reflective Platform Designed to Enable Agile Data-Driven Research on the Cloud*, in BC2DC workshop proc. IEEE eScience conf., 578-585, 2019. DOI 10.1109/eScience.2019.00079

[Klampanos *et al.* 2015] Klampanos, Iraklis Angelos, Martin, Paul, & Atkinson, Malcolm. *Consistency and Collaboration for Fine-Grained Scientific Workflow Development: The dispel4py Information Registry*. Zenodo. http://doi.org/10.5281/zenodo.3361395.

[Lannon *et al.* 2020] Larry Lannom, Dimitris Koureas, and Alex R. Hardisty, *FAIR Data and Services in Biodiversity Science and Geoscience*. Data Intelligence 2 (2020), 122–130. doi: 10.1162/dint_a_00034

[Levray 2020] Amélie Levray, *DKB Requirement Specification Document,* In preparation. 2020.

[Magnoni *et al.* 2019a] Magnoni F., Casarotti E., Artale P., Lindner M., Rietbrock A., Klampanos I., Davvetas A., Spinuso A., Filgueira R., Krause A., Atkinson M., Gemund A., Karkaletsis V., *DARE to Perform Seismological Workflows*, IN13C-0726, American Geophysical Union, Fall Meeting 2019.

[Magnoni *et al.* 2019b] Magnoni F., Casarotti E., Davvetas A., Klampanos I., *D6.3 Pilot Tools and Services, Execution and Evaluation Report I*, DARE D6.3, July 2019.

[Martin *et al.* 2019] Paul Martin, Laurent Remy, Maria Theodoridou, Keith Jeffery, and Zhiming Zhao, *Mapping heterogeneous research infrastructure metadata into a unified catalogue for use in a generic virtual research environment*. Future Generation Computer System, 101, 1–13. http://doi.org/10.1016/j.future.2019.05.076

[Mattheis *et al.* 2012] Sebastian Mattheis, Tobias Schuele, Andreas Raabe, Thomas Henties and Urs Gleim, *Work Stealing Strategies for Parallel Stream Processing in Soft Real-Time Systems*, in proc. Architecture of Computing Systems -- ARCS 2012, 172-183, Springer 2012

[Myers *et al.* 2015] James Myers, Margaret Hedstrom, Dharma Akmon, Sandy Payette, Beth A Plale, Inna Kouper, Scott McCaulay, Robert McDonald, Isuru Suriarachchi, Aravindh Varadharaju, Praveen Kumar, Mostafa Elag, Jong Lee, Rob Kooper and Luigi Marini, *Towards sustainable curation and preservation: The SEAD project's data services approach*, in: e-Science, IEEE pp. 485-494, 2015.

[Navarro *et al.* 2009] Navarro, A., Asenjo, R., Tabik, S., Cascaval, C. *Analytical modeling of pipeline parallelism*. In: International Conference on Parallel Architectures and Compilation Techniques (PACT).IEEE (2009)

[Pagé *et al.* 2019a] Pagé, Christian; Plieger, Maarten; Som De Cerff, Wim; De Vreede, Ernst; Drost, Niels; Klampanos, Iraklis Angelos; Karkaletsis, Vangelis; Atkinson, Malcolm and Pivan, Xavier. *Climate Data Access: Re-thinking our Data Analysis Workflows*, poster, in Proc. Scientific Gateways Conference (2019), DOI: 10.17605/OSF.IO/T8Y3H URL: https://zenodo.org/record/3546232#.Xfeh4S2ZOUk

[Pagé *et al.* 2019b] Pagé, Christian; Som de Cerff, Wim; Plieger, Maarten; Spinuso, Alessandro; Pivan, Xavier, *Enabling Transparent Access to Heterogeneous Architectures for IS-ENES climate4impact using the DARE Platform*, in Proc. IEEE eScience Conf. (2019) DOI: 10.17605/OSF.IO/WKM93, URL: https://zenodo.org/record/3546219#.XfelrS2ZOUk

[Python IDEs 2019] The Python IDEs, *The Python Language Reference* URL https://docs.python.org/3/reference/

[Ramakrishnan 2018] V. Ramakrishnan, Gene Machine, Oneworld Publications, 2018.

[Rietbrock *et al.* 2018] Andreas Rietbrock, Federica Magnoni and Emanuele Casorotti, Alessandro Spinuso and André Gemünd, *D6.1 Requirements and Test Cases I*, DARE D6.1, July 2018. URL https://drive.google.com/open?id=1ZISbDjphDR7gYNiQ24TQOM-npKxx169A.

[Rodriuez & Buyya 2018] Maria A. Rodriguez and Rajkumar Buyya, *Scheduling dynamic workloads in multi-tenant scientific workflow as a service platforms*, Future Generation Computer Systems, **79**, 2018, 739-750

[Rule, Adam, *et al*. 2018] "T*en simple rules for reproducible research in Jupyter notebooks.*" *arXiv preprint arXiv:1810.08055* (2018).

[Spinuso *et al.* 2019] Alessandro Spinuso, Malcolm Atkinson and Federica Magnoni, *Active provenance for Data-Intensive workflows: engaging users and developers*, Proceedings of the BC2DC workshop IEEE eScience conf. 2019. URL https://bc2dc.github.io/presentations/ActiveProvenanceASpinuso.pdf **Replace with DOI**

[Spinuso & Klampanos 2018] Alessandro Spinuson and Iraklis Klampanos, *Integrated Monitoring and Management Tools*, D3.7 DARE project deliverable URL https://drive.google.com/open?id=1QRDAQOoyPX13Uue_L3c_WnLNhapQID3z

[Trani 2019] Trani L. *A methodology to sustain common information spaces for research collaborations*, PhD thesis, University of Edinburgh, 2019.

[Trani *et al.* 2018] L. Trani, R. Paciello, M. Sbarra, D. Ulbricht and the EPOS IT Team, *Representing Core Concepts for solid-Earth sciences with DCAT – the EPOS-DCAT Application Profile*, Geophysical Research Abstracts 2018.

[Trani *et al.* 2018] Trani L., Atkinson M.P., D. Bailo, R. Paciello, Filgueira, R., *Establishing core concepts for information-powered collaborations*, Future Generation Computer Systems 89, 421–437, 2018.

[Trani *et al.* 2018] Trani, L., Paciello, R., Bailo, D., and Vinciarelli, V. (2018). EPOS-DCAT-AP: a DCAT Application Profile for solid-Earth sciences. In 2018 Fall Meeting AGU. Abstract IN31B-33.

Appendix 1 Abbreviations and Definitions

Table A1.1: *Abbreviations used in this document*

Abbreviation	Meaning
§	Section or paragraph
AAI	Authentication Authorisation and Identity
API	Application Programming Interface. the means by which software and developers use the capabilities a software subsystem or services offers
C3S	Copernicus Climate Change Service a service run by Copernicus
C4I	Climate for Impact a service run by IS-ENES
CMIP	
CSCW	Computer-Supported Collaborative Working
CWL	Common Workflow Language a W3C standard https://www.commonwl.org/
DCAT	Data Catalogue, a W3C standard describing the content of data catalogues
DKB	DARE Knowledge Base an open-ended place to leave and access information
DXWG	Data eXchange Working Group a W3C group developing a vocabulary to describe data, DCAT
EPOS	European Plate Observing System
FDSN	Federation of Digital Seismometer Networks that deploy seismometers on a long-term basis to collect and make available their wave-form observation time series
KB	Knowledge Base an organised repository of information used by people and software
LOD	Linked Open Data used to represent the semantic web
MIP	
MPI	Message Passing Interface used in HPC systems for parallelisation
MS	Mile Stone that marks project or research campaign progress
MT3D	Moment Tensor in 3D, a seismological method
P4	Protected Pervasive Persistent Provenance a means of recording what has been done
PE	Processing Element, a computational process or processes forming part of a data-streaming workflow

RaaS	Reproducability-as-a-Service the collection and use of provenance to facilitate repeating a computational experiment or analysis
RA	Rapid Assessment a seismological method estimating ground motion
RDF	Resource Description Framework (RDF) a W3C standard for the semantic web
Registry	The dispel4py Information Registry that manages information about dispel4py workflows and their component PEs
RI	Research Infrastructure computational, storage and networking facilities to support research or domain specific facilities to support research
RoI	Return on Investment the value obtained compared with the effort or cost needed
RSE	Research Software Engineer a designation of competence awarded by SSI
SSI	Software Sustainability Institute https://www.software.ac.uk/
SWIRRL	Software for Interactive and Reproducible Research Labs (ENVRIfair project)
UTC	Coordinated Universal Time https://en.wikipedia.org/wiki/Coordinated_Universal_Time
WaaS	Workflows-as-a-Service an automated support for authoring and using formalised methods
WMS	Workflow Management System that supports developing and running workflows
WPS	Web Processing Service an OGC developed standard for geospatial data
URI	Universal Resource Indicator a W3C standard
URL	Universal Resource Locator a W3C RDF-related standard

Appendix 2: Summary of DKB functionality

We present here tables, listings and figures summarising the functionality of the DKB subsystem, including the Registry and the Data Catalogue, presented in §4.2. In the context of DARE there will be one instance of the DKB per instance of the DARE platform. All users of that platform will share this instance of the DKB. New releases of the platform may include new releases of the DKB, its Python library and the Conceptual library. Procedures, human and computational, will be needed to install releases without losing information accumulated in the DKB by users of the previous version of the platform and DKB.

Table A2.0a: Functions applicable to instances of the DKB. Python call details will be in [Levray 2020][85].

Function	Description
newDKB	Make a new instance of the DKB with the supplied instance name (the first part of every PID in this DKB instance) and populate it *only* with the standard conceptual library as Context kb. **NB** if this is a re-use of an existing instance name it will throw away all of the existing state in this instance. Therefore, if the DKB finds it is re-use of a previous identity, it will do nothing, unless the optional parameter reset has been set to True. The new instance will have had the kb Context set up. Most users will want many other things set up for them. As there is one instance of the DKB per instance of the DARE platform, this platform/DKB identity setting and initialisation using newDKB is performed as part of the installation of a DARE platform.
login	Start using an existing instance of the DKB specified by name, with the supplied user name, currently as preferredName, and sessionId, a String meaningful externally, e.g., issued by the AAI service. This resumes use of the DKB instance, potentially concurrently. It will set up the DKB's *internal* operational state, such as current Context and User, for this Session or run.
close	Terminate the session or run. Attempts to continue use result in errors. Make any new or revised state persistent, where this hasn't already happened. Release resources and discard the session or run internal state.
status	Return a dictionary, which is a snapshot of aspects of the DKB it is prepared to reveal, e.g., for diagnostic purposes, as name: value pairs. Minimal until evidence extra things are needed emerges.
deprecate	Change the state, so that attempts to log in to this DKB get a warning.
delete	Throw away this instance of the DKB, releasing all associated resources. NB have you made alternative arrangements for PID resolution or are they unnecessary?

[85] Tables in this Appendix are in two parts: the (a) part: things that are being or have been implemented, the (b) part: things that *might* one day be implemented.

Table A2.0b: Possible future developments for managing DKB instances.

Function	Description
backup	Preserve the state of the DKB in a suitable independent reliable storage system. There may be controls, e.g., for selectivity or incrementality.
restore	Recover the state. There may be similar controls. Where partial restoration is used, there may be checks on the impacts on the other DKB state.

A2.1 Entry

The underpinning role of Entries in the DKB was presented in §4.2.3 as a generic term for any Entry in the DKB, including Concept definitions and instances of Concepts. They also include Entries that do not comply with these additional structures in order not to overconstrain users. However, we would prefer them not to use Entry functions directly.

*Table A2.1a: Functions applicable to all Entries in a DKB. They may not all be implemented in a DKB instance. The category of Entry may determine their interpretation and whether they are available. They are based on the data-lifecycle developed as the ENVI reference model[86]. The use of direct Actions on Entries is **deprecated**. We recommend that users work via the Conceptual library features - see §4.2.3 and §A2.4. Python call details will be in [Levray 2020].*

Function	Description
newEntry	Introduce a new Entry with its initial values, relationships or attributes. Some values will be added automatically. The values are supplied as a Python dictionary, They must include a value for name not already used in the current Context and may not specify the value for any automatically set attribute.
get	Obtain the current state of an Entry identified by name, prefix:name or PID as a Python dictionary, including the automatically set attributes.
update	Change, add or delete some of values in an Entry identified as in get iff updates are permitted for this Entry in its Context by the current user or run. The values to change are supplied as a Python dictionary. They may not change any of the standard attributes. Existing additional attributes may be changed and new ones may be added. The attributes to delete are identified by an optional list. A new uniqueness counter value will be used in the pid and the subsequent attribute in the updated Entry will be set to this pid and the previous attribute will be set in the new Entry to the old pid.

[86] ENVRI Reference Model https://confluence.egi.eu/display/EC/ENVRI+Reference+Model

A2.2 Context[87]

A Context is not represented as a DKB Entry because it is managed differently from Entries, e.g., mutation does not generate new versions. The set of Entries defined in a particular Context is inferred from their prefix attribute in their Entry and PID. The other attributes of a Context are:

```
prefix: String,          # the prefix that identifies this Context
owner: Person,           # the individual who created and controls this Context
user: Person[],          # the current individual working in this Context, empty if no one[88]
searchPath: String[],    # when a name not found locally ordered Contexts to search next
state: String,           # one of a small list of DKB-determined states
…                        # attributes below here may be needed later
readers: Person[],       # a possible privacy/authorisation control feature
writers: Person[],       # a possible mutation and authorisation control system
history: Session[],      # the current and previous sessions, current first then order by age
```

For each user session or method run using the DKB, the DKB keeps track of the current Context, in *private* variables. We give examples:

```
_home: Context,          # the current Context; first place to look
_contexts: Context{},    # a dictionary mapping prefix to Context for all contexts available
_stateKB: String,        # current status of this DKB instance
```

When the DKB wants to get an Entry by quoting its name, this initiates a search with the following algorithm, which determines the nature of inheritance.

```
def get(name):                  # try to find an Entry with the supplied name[89]
   pr = _home.prefix            # look first using the current prefix
   res = lookup( pr + ':' + name )# use prefix<standard separator>name for existing Entries
   If res:                      # lookup returns None if not found
      return res                # it was found => return result
   path = _home.searchPath      # not found - prepare to search using search paths
   for pr in path:              # look first using the current prefix
      res = lookup( pr + ':' + name )       # use prefix<standard separator>name for existing Entries
      if res:                   # lookup returns None if not found
         return res             # it was found, return result
      path = setAppend( path, _contexts[pr].searchPath )  # append to path any prefixes not in it
   raise EntryNotFoundError( name + ' not found starting from ' + _home.prefix )
```

[87] Consider recording owner Person, who can manage authorisation lists.
[88] Normally an additional user would not be permitted, but if they are, the list holds the most recently joined first.
[89] Need to sort out explicit prefix and explicit PID first in real code.

This algorithm ensures that the search path of the current Context dominates, and if a user just wants to work by extending a Context, however it is built, they only need to put it in their search path. Putting the current Context as the search path is a sensible default, when a user creates a new Context.

Table A2.2a: Functions operating on Contexts that are implemented or being implemented. See [Levray 2020] for specifications.

Function	Description
newContext	Introduce a new Context with a specified prefix (check unique) and specified search path/ The search path defaults to the Context in which this Action is performed. Set counter to zero for local Entry discrimination. Do not automatically enter the new Context.
~~update~~	Add, update or discontinue Entries, changing the set available. This is done by modifying Entries with the relevant prefix. *No explicit function is needed!*
getSearchpath	Return a list of Strings corresponding to the current, locally held search path, i.e., perform **return** _home.searchpath
setSearchpath	Set the search path of the current Context, i.e., perform _home.searchpath = searchpath
clone	Generate a new Context with a copy of the supplied Context[90]. Auto-discriminate prefix by appending an integer when the caller does not supply one.
enter(r|w)	Move into the Context with the intention of reading or writing. It becomes the new start of name searches. Re-entry on next login if this was the last Context. If there was a previous Context that has been left. The Context administration code is not required to remember previous Contexts visited.
leave	Explicitly leave the operational Context. Move into limbo. Methods may need this to minimise the risk of propagating information unintentionally.
discard	Explicitly stop this Context from being used further. The extent to which its resources and Entry can be discarded is an open question, but a sufficient RIP is needed. It is a research question, what this needs to contain.

Table A2.2b: Functions operating on Contexts that may be implemented when needed and resources are available.

Function	Description
branch	As for clone form a copy, but retain a relationship with the origin as master[91].

[90] May implement using copy-on-write and accelerate using Bloom filters.
[91] May use Git algorithms.

pull	Obtain updates from the master.
push	Send updates to the master - may need other Git commands.

The interaction between a platform's login procedures, for users and for their delegated methods, will need to link in with the Context management initialisation, and Context owner identification. This may need to be co-developed in each new deployment Context, and comply with the privacy and collaboration regulations prevailing in that community. See §5.2 for our immediate plans for AAI implementation. This will deliver an identifiable Session instance associated with a known Person instance.

The Session instance will be automatically created and uniquely identified by the platform. The DKB will capture information that is needed by others or that needs to persist in this instance. See §5.2.

There are several issues here, for example that the token delivered by an AAI procedure may be very short-lived and could expire before Method run finishes. The standard also describes that the token should be per audience and per client, but doesn't offer a ready-to-use solution for "batch" kind of jobs that run in the background on distributed resources. There is the possibility of getting a second type of token ("Refresh token")[92] to retrieve new access tokens, but some entity would have to manage this and redistribute it to the distributed processes that want to use it (e.g. during a dispel4py workflow running on multiple servers), which then connects with the a Session. We are currently leaning towards a solution where the DARE API would "hold" the original user token (short-lived JWT) and would, upon job execution, exchange this for a delegated token for backend processing which is valid for a longer period of time but is scoped, i.e. only allows for limited actions such as insertion but not retrieval of provenance data. This could be combined with Session information to make it even more secure.

The Person instance will contain a consistent, persistent but pseudonymised to be normally uninterpretable in human terms identity by default to comply with GDPR and local data-protection regulations[93]. If, during some Session, when identified in this way, a user performs an act supplying a public person PID, e.g., their ORCID identity, then their Person instance will be updated to include this information. When it has been so supplied the DKB will keep and reveal their human-interpretable identity, since it now stores a mapping from their pseudonym used by the platform to the identity they have just supplied. For this revelation to be permitted the naming service used must have a compliant informed consent procedure. A

[92] https://openid.net/specs/openid-connect-core-1_0.html#RefreshTokens

[93] The individuals need a consistent pseudonymised identity, so that all their work and actions can be appropriately attributed and gathered. Often, there is an authority, capable of reversing the mapping from human identity to hard-to-interpret pseudonym, e.g., in medical contexts, in case a research subject is found to have a serious condition through the research. Some communities may require this in case the person's behaviour is posing a problem. That mechanism and decision is external to DARE and the DKB.

simpler initial option, is to allow users, when they create a Person instance to specify a preferredName that they wish to be known by, leaving this empty would imply anonymity.

A2.3 Concept

The requirement for creating, managing and using Concepts is described in §4.2.3. Here we begin to outline how those Concepts are supported and their use is facilitated. In the listings using a mocked up language, the keyword **concept** introduces a Concept definition. It may be a new Concept as shown in Listing A2.2[94].

The representation of a Concept instance is defined by the 'Concept' instance itself, as shown in Listing A2.1. Readers not concerned with these representational issues are advised to skip to Listing A2.2.

The Concept definitions are all Entries and so they have the attributes of every Entry. The attributes are then defined by a list of statements each introducing one attribute with the identifier given at the start of the statement, followed by a colon, followed by a description of the values that attribute may take. That description is constructed using an expression that yields a built-in Python type, a named Concept, or the name or literal for an AttrDescr instance. The expression may also show Collections of these or alternatives. Universally, None is an alternative, so that unknowns may be represented, possibly to be filled later. Any permits unconstrained values to be associated with the attribute in instances.

Attribute names may not repeat within a Concept. The list is partitioned. Attributes before --- are required, every instance must have them. Attributes after --- and before … are recommended, and those after … are optional. The fact we are defining Concept makes this recursive. It is easier to understand Concept definition by looking at Listing A2.2 below.

```
concept Concept:                    # define a Concept, start with text for its description
        "The means of conceptually organising using named Concepts with attributes specified"
                                    # these attributes are inherited from Entry and would not appear
                                    # we show the values that would be automatically supplied
        name: 'Concept',            # name is Context unique in dare
        prefix: 'dare',             # its Context is the shared library in every DKB instance
        pid: 'ex:dare:1:Concept',   # persistent exportable identifier for Concept
        instanceOf: 'ex:dare:1:Concept',   # Concept is an instance of Concept
        timestamp: xxx,             # when Concept was defined at the start of the DARE epoc
                                    # there are two required attributes for every Concept definition
        extra: {String: Any},       # a free form dictionary of any extra attributes a user adds
        mutability: String,         # a term, e.g., "mutable", "immutable", "no hiding" => immutable
        ---                         # after 3 minus signs, we get the recommended attributes
        description: String,        # every concept should have a description saying what it is for
        …                           # beyond here optional attributes have their form defined
```

[94] These listings are illustrative. In fact, this notation is not yet supported. At present, everything is done using the supplied Python library calls.

```
required: {String, AttrDescr}       # Dictionary of attribute names and their specification
recommended: {String, AttrDescr},   # Dictionary of attribute names and their spec.
optional: {String, AttrDescr},      # Dictionary of attribute names and their spec.
translation: {String, String[]},    # External equivalents, e.g., ontology terms, for any
                                    # of the attribute names in the previous 3 dictionaries
                                    # so that experts can specify a mapping to RDF
                                    # The value mapping may be in an AttrDescr instance
```

Listing A2.1: The notation for defining a Concept: i) What does it specialise and ii) What are its required, recommended and optional specified attributes in addition to those required by Entry in each of its instances. Instances may have additional attributes in extra.

A Concept may be a specialisation of another Concept. This is indicated by the following notation.

```
concept SpecialC(C):    # define SpecialC  as a specialisation of C
```

The SpecialC instances may be used anywhere an instance of C is required. In addition, the specialisation inherits the specification of attributes from the Concept it specialises. It may add more or modify existing ones using the same notation for required, recommended and optional attributes. Attribute names may not be duplicated, so if one in the Concept being specialised appears in the specialisation specification, it redefines the original.

```
concept Widget:          # a small example Concept definition
   "An example"          # the description c.f. _doc
   shape: String         # a required attribute

concept SpecialWidget(Widget):      # define its specialisation
   "A specialisation of Widget"     # and its description
   ---                              # no more required attributes
   length: Real,                    # recommend specify length
   weight: Real                     # and weight
```

Listing A2.2: Examples defining a Concept Widget and a Concept SpecialWidget.

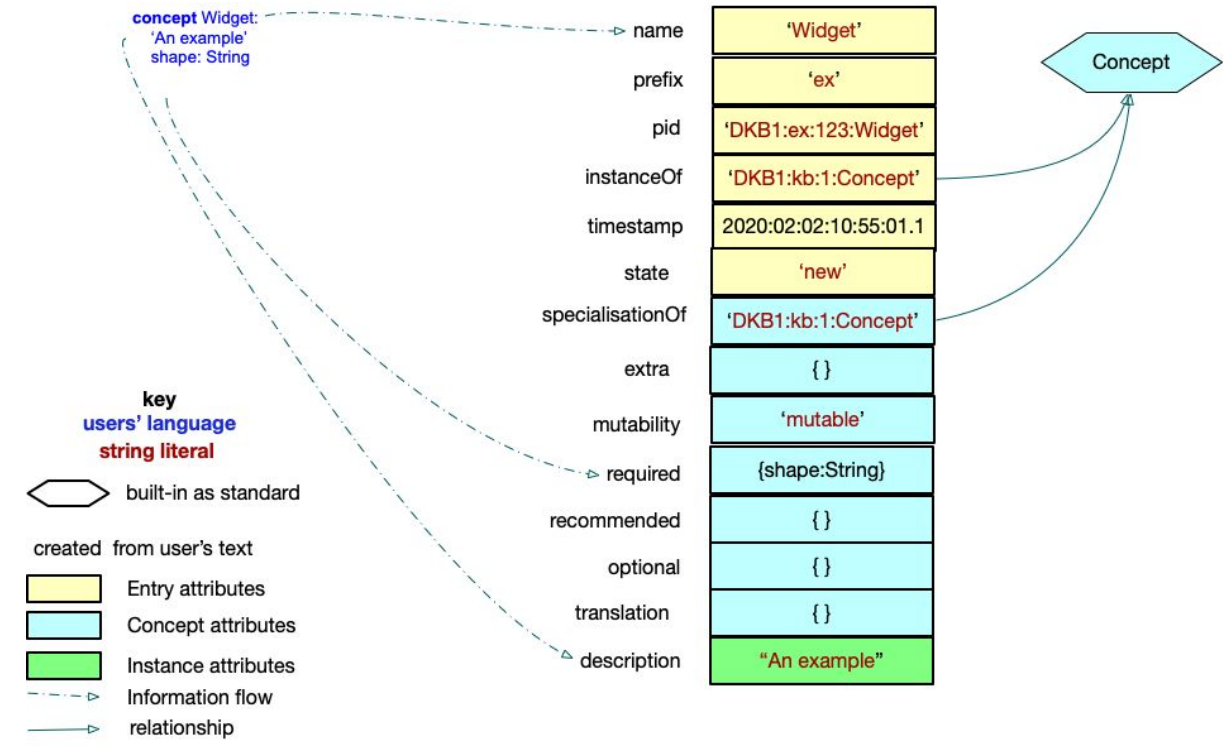

Figure A2.1: The corresponding DKB Entry for the Concept Widget. Interpretation on platform instance DKB1 in the context with prefix ex.

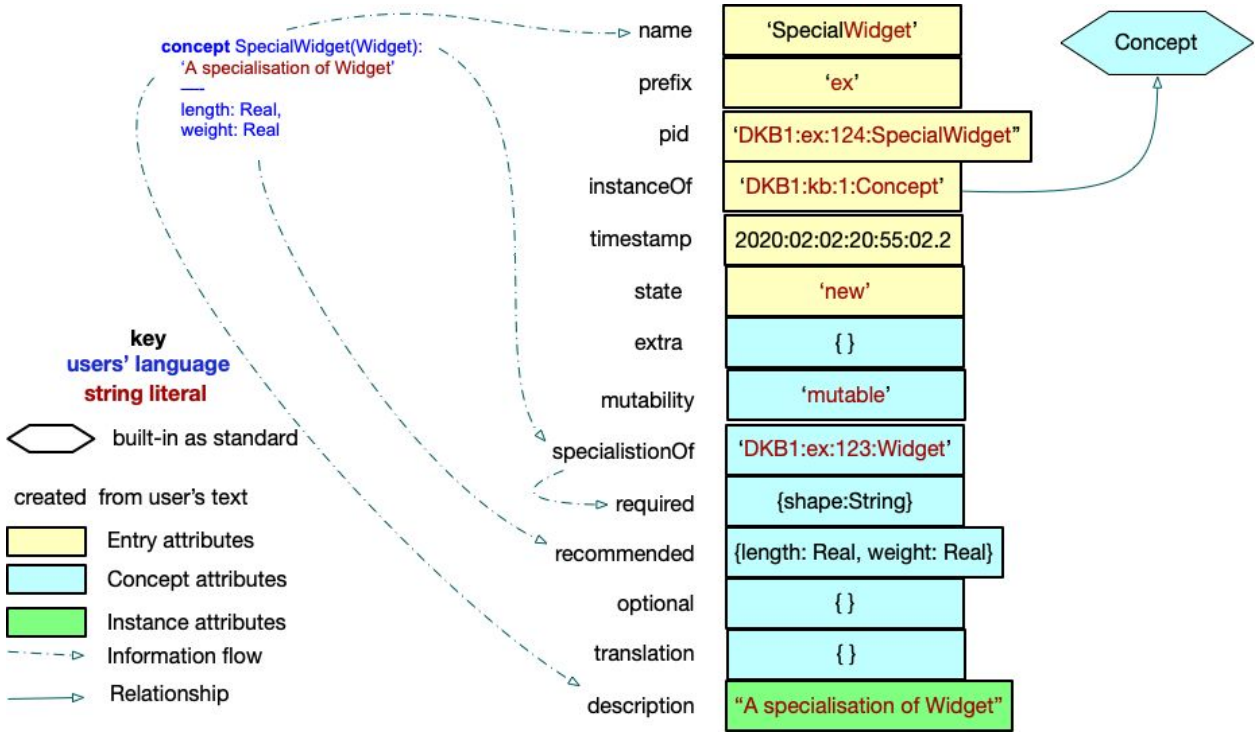

Figure A2.2: The corresponding DKB Entry for the Concept SpecialWidget. Note the relationships, e.g., with a Concept that was specialised, is stored as a PID.

```
aWidg = Widget{          # construct an instance of Widget
   shape: 'cube' }       # with one attribute in a dictionary

aSWidg = SpecialWidget{        # and an instance of SpecialWidget
   shape: 'sphere',      # with required shape attribute
   length: 12.37,        # and one of the recommended attributes
   value: 100 }          # and an additional attribute the user wants
```

Listing A2.3: An example constructing instances of Widget and SpecialWidget. In the second case not all of the recommended attributes are supplied but an extra one is.

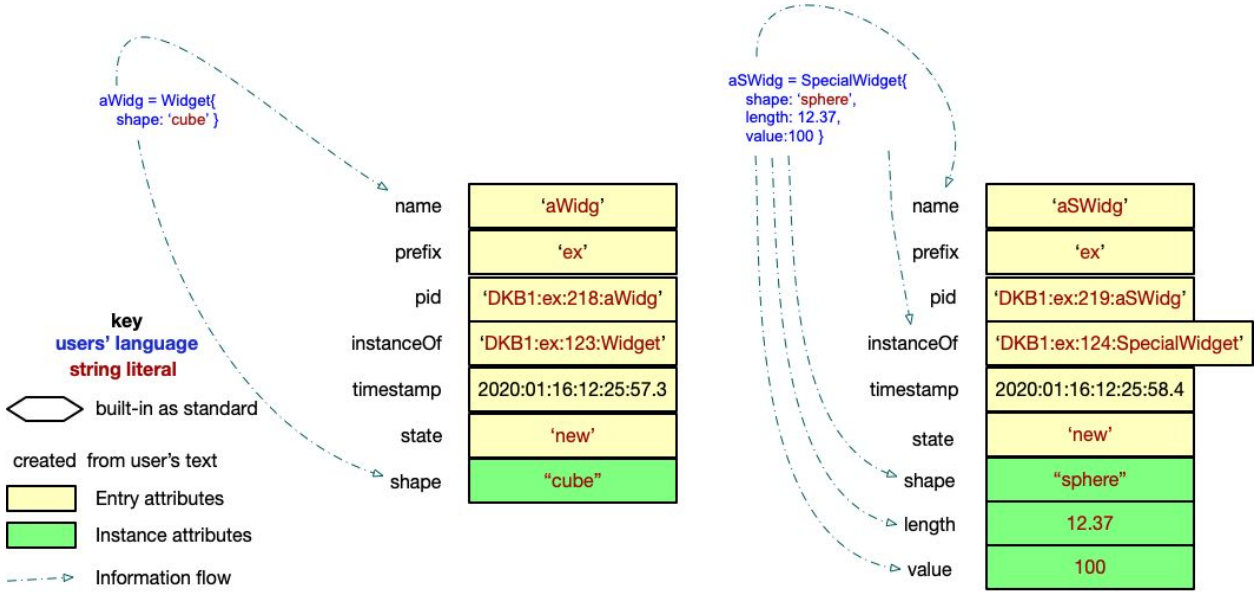

Figure A2.3: The two Entries in the DKB corresponding to aWidg and aSWidg. These show the user using the Concepts they have introduced to inform interpretation and to organise information. However, they can still use their own judgement and not be over-constrained.

Table A2.3a: Functions applicable to Concepts in a DKB. See [Levray 2020] for implementation details and Python function calls.

Function	Description
newConcept	Create a new Concept. If the Concept being specialised parameter is omitted, it defaults to Concept. The three dictionaries, defining required, recommended and optional attributes, have the attribute identifier followed by an expression that may restrict the values the attribute may have, as described below. The Concept Concept may not be modified, redefined or hidden as it forms the key to the rest of the organised DKB information. The rules for attributes and inheritance are defined below.
newInstance	Create a new instance of the specified Concept supplying the attributes for this instance of that Concept as a dictionary. Any attribute not mentioned is omitted or gets its default value. There will be checks as to whether this complies with

	the Concept's definition.
get	The identifier for an instance[95] in the DKB is supplied as a name, a prefix:name or pid. Some KB implementations may also allow a Python object pointer. The result returned is a dictionary of name: value pairs in the same form as used for newInstance or newConcept. If a non-empty list is supplied as the onlyThese parameter, then only the intersection of the attribute names that would be returned and names as Strings in this list are returned. See §A2.2 for the search algorithm if just a name is supplied.
find	Return an enumerable Collection (for the moment a materialised Python list of PIDs) of all of the entries that match the supplied query. The query development plan is described below.
update	The instance to be updated (this includes a Concept definition) is identified in the same way as it is for get. The new values are supplied in the same way as they are for newConcept and newINstance, i.e., as a dictionary of attribute-name, value pairs. If any checks are necessary, they are all performed before any change occurs. If a problem is found, an appropriate **exception** is raised, with the DKB state unchanged. Otherwise, all of the changes are made.
instances	Return an enumerable Collection (currently a materialised Python list) of all the instances of exactly this Concept.

Attribute specification can be partitioned into attribute naming and specification of the permitted values that attribute may take.

1. The **attribute names** / identifiers normally start with a lower-case letter and follow Python identifier lexical analysis rules (§2.3 in [Python IDEs 2019]) as far as possible. They must be unique within the Concept being defined, including those present by default and those inherited.
2. The specification of permitted values is given by an expression that:
    a. names (under the get rules) of a specific Concept, instances of which supply this attribute's values, taking into account specialisations.
    b. names (under the get rules) of a specific instance of AttrDescr which contains a description of the permitted values and optionally mutation rules.
    c. **Any** to imply that the permitted values are unconstrained.
    d. In due course, there may be expressions to indicate compositions, collections or choices between these, but they fall within AttrDescr instantiation at present.

Inheritance during the creation of a Concept CS that is a specialisation of Concept C, is as follows:

1. All of the attributes of C become attributes of CS unless otherwise specified by an attribute name in C appearing in one of the three supplied dictionaries.

[95] NB this retrieves a Concept, as that is an instance of Concept. It may also be used to retrieve an Entry that is an instance of Thing.

2. All attribute definitions given for CS that do not appear in C are included straightforwardly.
3. Where a name in CS already appears in C the following actions are taken in the given order.
    a. If the name now maps to **None**, the attribute is dropped.
    b. If the name now appears in a different inclusion group, e.g., moves from recommended to required, it is removed from the old inclusion group and added to the new inclusion group in the form specified in the CS creation call.
    c. Otherwise, the new specification overrides the old specification if it is different.

The formulation and interpretation of **queries** used in find calls will consider the following and incrementally introduce the capabilities, not necessarily in the order given:

1. The possibility of using a Python library that helps with the relevant query evaluation.
2. The need to perform comparators on scalar values:
    a. == and !=
    b. >, >=, =< and <
    c. range tests, e.g., 0 < x <= 100
    d. inclusion tests, e.g., x **in** [1, 2, 3, 5, 8, 13, 21, 34] or x **in** "the lazy brown dog"
    e. Approximate matches, e.g., x ~= "DARE"
3. The need to support **isa** on Concept values and hierarchies.
4. The need to support time comparisons, e.g., timestamp **after** 2020:02:02.
5. The need to support string matching, potentially including contains, regular expressions, approximate matching, etc.
6. The need to permit composition with **and**, **or** and **not**[96].
7. The need to handle subqueries, but see Collection expressions.
8. The need to support expressions combining attribute values and literals.
9. The need to provide users with a notation and/or tool that helps form queries over the DKB.
10. Ditto for the DKB plus contemporary services.

Update rules that *may*[97] be checked include the following:

1. That the instance is mutable.
2. That the actor (user or software) is permitted to update in this Context.
3. That this attribute is mutable.
4. That all the required attributes will be given a value.
5. That the values given match the rules for each attribute.

Supporting **extra attributes** to deliver open-endedness and the ensuant flexibility.

[96] This is the probable limit of implementation achievable in DARE.
[97] Early versions will only check a few of these.

Table A2.3b: Functions considered but they will be implemented only when required and if resources are available.

Function	Description
validate	Record that this instance (including a Concept) has passed quality controls.
publish	Arrange that this instance (including any Concept) is publicly usable.
freeze	Cocoon this instance (Concept) so that it can no-longer be changed.
deprecate	Arrange that attempts to use this instance (Concept) trigger warnings.
discard	Prevent further use of this instance (Concept).
promote	Copy the definition of an instance (Concept) into a specified Context, dealing with or warning about any consequences for current uses of this Entry.
export	Provide an externally usable translation of this Entry. There may be options to choose the external representation.
preserve	Make a back-up copy in a preservation service.
restore	Recover a backed-up copy.
archive	Make a preserved copy with appropriate metadata and quality controls.

The functionality given in A2.3 is directly usable. However, a KB is prepopulated with a library of Concepts that help in its use and administration. The initial library is described in the next section. It is envisaged that communities will build on this with further tailoring work to facilitate their work, normally as additional Contexts. This tailored prepared libraries can continue to prepare for specialisations and groups within the community, concurrently with the KBs other uses.

A2.4 Concept library

We present here the initial library that is made available in Context kb when an instance of the KB is created. The Tables below are partition in the usual way, with the (a) parts, presenting the currently available and developing features, and the (b) parts presenting the features under consideration for future development. Each of these is partitioned by the category of Concepts being presented, numbered with a Roman numeral. A category partition may be followed with a Table of functions additional to those presented in Table 2.3 that apply to Concepts in this category. Empty Tables are omitted.

Table A2.4a.i: Library of Concepts provided as common organising foundations in every KB.

Concept name	Description
kb:Entry	The unstructured Entry into the DKB, with no defined user attributes. Its direct use is deprecated.
kb:Concept	The basis for all other Concepts with zero or more mandatory, recommended or optional attributes of specified form - see A2.3.
kb:AttrDescr	A description of the permitted values of an attribute may take; a relationship to an instance of this specifies the form and optionally the use of an attribute - see notes below.

The creation of AttrDescr may be automatically arranged in future developments based on a Python expression or a parsable String, for the moment it needs to be explicitly created as an instance using the functions in Table A2.3a. There are however a set of instances, in kb with a standard definition corresponding to each of the Concepts in Table A2.4a, or Any, with the names given by strings of the same name, e.g., 'Concept', 'String' and 'Any'. These specify that the value must be an instance or literal corresponding to that Concept, that is mutable. The AttrDescr should eventually be able to specify:

1. Collections of such Concepts, i.e., initially at least list String[] and set Method{}.
2. Compositions of such Concepts, e.g., tuples (String, Number).
3. Choices between value options, e.g., [Real, Integer, Number, Complex, Time].
4. Enumerated options, e.g., Bool: 'True', 'False'

AttrDescr will also specify constraints on mutability and *may* specify constraints on visibility, default values and preferred toString or visualisation methods. The details and representation of AttrDescr will be developed incrementally, when they are needed.

Method foundations

Table A2.4a.ii: Library of Concepts provided as foundations for Method authoring, management and use in every DKB instance.

Concept name	Description
kb:Method	The highest level and most general description of a method, giving at least its name, a description and normally its inputs and outputs.
kb:Action	A specialisation of Method that is simple and locally enacted, so that it *may* be treated as atomic and instantaneous, e.g., its runs *do not need to be explicitly recorded*, only its effects. All of the Python functions in the tables in Appendix 2 are instances of Action, unless otherwise specified.
kb:PythonMethod	Specialisation of a Method encoded in Python, e.g., as a function or class method. The kb will be populated with published instances of PythonMethod with the name of the function the same as that given in the Tables in §A2.

dare:D4pMethod	Specialisation of Method encoded in dispel4py that will generate a graph of interconnected processors, PEs, via arcs of data streams. It includes stand-alone PEs. These are currently described in the Registry and the implementation *may* cross-reference Registry entries. **NB**, this Concept is in the Context dare because not all KB uses will use dispel4py.
kb:InputDescr	Description of one input to a Method - see below.
kb:OutputDescr	Description of one output from a Method - see below.
kb:Run	The records of an active or completed enactment of a Method instance.

Table A2.4b.ii: Library of additional Concepts that may in the future be provided as foundations for Method authoring, management and use in every DKB instance. This will depend on requirements and resources.

Concept name	Description
dare:CWLMethod	Specialisation of Method encoded in CWL.
dare:CompReq	A computational requirement, normally of a Method

The design of Method and its specialisations and related Concepts is underway. It will place most of its attributes in the recommended category to facilitate an initial idea being incrementally developed and refined. These attributes will cover:

1. Input specification for each input that eventually defines parameters, data requirements, and defaults. This may specify the semantics (meanings) of each input, as well as formats and explanations in a fully developed method.
2. Output specification for each output, eventually specifying semantics, format and usage.
3. An implementation or list of implementations, e.g., as Method Python codes or calls, dispel4py programs or compositions of other Methods.
4. Version and/or PID to identify precisely the Method. The available implementations may be independently varied and identified, i.e., precisely defining what a Method does is separate from defining how it does it, the former may be focused on human perception for collaborating experts, the latter on software interpretation optimised and mapped onto currently available platforms.
5. Instructions (computer interpretable) for a human interacting with a Run of the Method, or more often, for the software preparing, deploying and monitoring the chosen implementation for this Run.
6. Possible failures.
7. Data to help with optimisation.

The definition and representation of the Concept Method is currently under development, and the specialisations may be restricted to particular forms of Method and may contain additional details in their attributes. We plan co-design and co-development in conjunction with the

development of dispel4py optimisations. Hence the early inclusion of dare:D4pMethod and related Concepts.

The Run Concept will record whatever is needed for developers and users to discover what has been done and how it was done. For example, it will refer to the Method instance being run, note at least the start and end times, the authorised identity requesting the run, and the Session within which that occurs. For submitted run requests that do not originate from a current interactive Session, the run implementation may create a Session instance. Further performance of Actions and Methods within the same Method enactment, do not create more Run and Session instances, instead these are accumulated at a level of detail that may be changed as the run request is configured[98]. In DARE this is done by the provenance service P4. The Run instance, refers to that P4 enactment trace, which itself may refer to the Run instance, or to some of the key aspects of the enactment context.

Interleaved with Table A2.4 partitions are partitions of Table A2.5 showing corresponding additional functions, beyond those shown in Table A2.3, or refining the meaning of those functions when applied to instances of Concepts in the category that has just been described.

Table A2.5a.ii: Additional functions applicable to Method instances in a DKB. See [Levray 2020].

Function	Description
newInstance	Introduce a new Method or Method specialisation instance that may be refined until ready to run by a series of updates to complete or correct it. It may be imported from a reliable external source. *Import has yet to be supported.*
run	Start the enactment of the Method instance, generating a Run instance that may link to traces in P4.
publish	Make the Method instance available for use via the API.

Table A2.5b.ii: Yet to be implemented functions applicable to Method instances in a DKB. These will only be implemented if required and resources are available.

Function	Description
test	Run the set of tests associated with this Method.
validate	Record that this Method has passed its tests and is approved for use.
deprecate	Arrange that a specified warning / reason, e.g., replacement available remedying an identified fault, is given to anyone attempting to use this Method from now on. Those who have expressed interest in this Method or built composite methods using this Method, should be sent the warning message.

[98] This will need to align with arrangements for provenance configuration.

Data handling

The Concepts denoting data, cover many kinds of files, databases and *may* include literal representations stored directly in the DKB. The varieties of data that may usefully be distinguished are unbounded as every discipline may want to discriminate different semantic content and different representations/formats to refine and guide the use of data they import or produce. This discrimination extends all the way to individuals developing new science.

A well-developed notation has been developed for describing such data by the W3C Data Exchange Working Group as the "*Data Catalog Vocabulary*" [DXWG 2020]. These powerful descriptive ontologies need to underpin our description of data in order that interworking with other systems and catalogues is facilitated. However, the users require vocabularies that match their own community's handling of Concepts. In the context of EPOS, this has already been well developed [Trani 2019] and this should influence our choice of specialisations of the Data Concept for the geoscientists associated with EPOS[99]. The climate community almost certainly has similar work underway under the aegis of IPCC or NetCDF. DARE already has a substantial investment in its Data Catalogue - see §4.2.4. The development of the Data Concept therefore needs to navigate a complex pre-existing and co-evolving space. Initially we will focus on the high-level notion of Data and expect a substantial family of specialisations to emerge.

There is a significant difference between the DKB treatment and other treatments:

- Other treatments bundle together Data instances and Collections of Data instances, e.g., a directory of files may be viewed as a single dataset.
- We treat Collections separately, as we believe this has conceptual and engineering advantages - see below.

The current Data development is a first step that may be revised significantly, as the practical use provokes change. That practical use will include developing an alignment with the existing Data Catalogue, and exploring with WP6 and WP7 the extent to which they benefit from describing the data they use. The Data Concept will include the following aspects of a Data entity:

1. Its name (often repeatedly the same, as a bundle of Data instances will be collected in a manufactured Context, e.g., as the data input to a Method run or the results produced).
2. Its PID or DOI, either imported from its definitive source or locally constructed.
3. Its source, including originators, if made locally.
4. Its creation date/time (this may be very similar to the Entry time if made locally).
5. Its semantics/meaning/intended use in a form understood by humans.
6. Its format in a form understood by software.
7. Its volatility and preservation/discard plan.

[99] The formalisation of this work as EPOS-DCAT-AP provides the definitive source https://github.com/epos-eu/EPOS-DCAT-AP/.

8. Usage controls.
9. Locations where a copy of this data can be found if it is not a local literal, otherwise the local literal value. This will allow the DKB to act as a proxy for data held elsewhere.

Table A2.4a.iii: Library of Concepts that will be provided as foundations for Data production, management and use in every DKB instance. See [Levray 2020] for further details.

Concept name	Description
kb:Data	Any data item may be described by an instance of this Concept or its specialisations.
is:Document	A specialisation of Data meant for reading and printing. There may be specialisations of this for each way of structuring and formatting. It may have additional attributes, such as language, page size and number of pages, etc.
dare:Dataset	A DCAT compliant Data specialisation, e.g., as a proxy for an entry in the Data Catalogue

Table A2.4b.iii: Library of additional Concepts that may be provided in the future as foundations for Data production, management and use in every DKB instance.

Concept name	Description
is:JSON	A JSON document, e.g., as used in seismology methods
is:Code	Any code text. This may often be a proxy for code stored in a Git.
is:PythonCode	A specialisation of Code referring to or containing a Python script

There may be a need for ancillary Concepts, e.g., to represent data use rules or mappings to formats.

Table A2.5a.iii: Additional functions applicable to Data instances in a DKB. See [Levray 2020].

Function	Description
newInstance	Introduce a new Data or Data specialisation instance that may be refined until ready to be used by updates. It may be imported from a reliable external source and exported for external use, but see import and export below.
publish	Make the Data instance available for download.

Table A2.5b.iii: Yet to be implemented functions applicable to Data instances in a DKB. These will only be implemented if required and resources are available.

Function	Description
proxy	Virtually import a Data instance, so that it may be used as if held locally, but without paying the cost of moving the data and without losing access to subsequent updates.
import	Import a snapshot of an external data item as a local copy.
export	Deliver a copy, e.g., as a result of an http request, or as a result of an internal imperative in the form specified to the destination specified,
preserve	Place a copy in reliable storage and add this to the places where this data item can be found.
archive	Place a copy with required metadata in the specified archival service. This may be performed on bundles of items, assembled as a Collection or in a Context.
test	Run the set of tests associated with this Data.
validate	Record that this Data has passed its tests and is approved for use.
deprecate	Arrange that a specified warning / reason, e.g., replacement available remedying an identified fault, is given to anyone attempting to use this Data from now on. Those who have expressed interest in this Data or use it repeatedly should be sent a warning message.
discard	Arrange that this Data is no longer available, and release storage resources where it is appropriate to do so.
expunge	Remove all copies of this Data and eliminate the chance of reconstruction and access. .NB this is very hard and possibly impossible to do for all cases.

Collection handling

The use of Collections is so prevalent in human behaviour and in computational practice that significant gains can be made by making them a feature in Conceptual space - *digitally using the tray or trolly when clearing the table instead of making many trips to the kitchen*. Precisely which forms of Collection scientists want is an open question. We start by supporting those commonly used in programming and in databases we will then explore the exact forms these should take and which other specialisations of Collection are needed. We observe three uses of Collection instances:

1. *Marshalling their members*: incrementally collecting or excluding members sometimes using queries over other Collections, databases or external sources, and sometimes excluding members that fail to meet specified criteria. This may be assembling samples

or representatives as input to processes and Methods. It may be accumulating evidence to reach a statistical threshold or to support a compelling visualisation.
2. *Representing logical and mathematical structures*: the assembly into representative Collections, e.g., set, array, table or time series, that matches the required structure.
3. *Bundelling to reduce work*: saying do m for every member of Y is a lot easier and less error prone than trying to manually organise each application m applied to y where $y \in Y$. This also provides more optimisation and systems-engineering opportunities.

Researchers want to deal effectively and efficiently with two categories of Collection:
1. *Their own*, where they directly design and control them, deciding what they should contain, and adding and removing members until they are satisfied they contain the required population. Myers *et al.* [2015] showed the benefits and improvements of providing researchers with their own easy to instal and use catalogues to facilitate this in the early stages of research. It proved an adoption incentive and it improved the quality of results archived, because it accelerated progress and reduced labour.
2. *External reference sources*, these include the catalogues offered by their community's archival services, repositories of simulation results, and results of other researchers.

By delivering a usable abstraction of Collections that gives users the direct control they want and that maps well to the underlying Collection representations we expect to deliver benefits to researchers and to those undertaking the engineering necessary as the number, population and size of the individuals increases. The next round of the IPCC analyses, clearly takes us into that territory. Higher-resolution seismic-wave propagation simulations would also require such engineering. We hope that these abstractions will help us address the boundary between databases and scientific workflows as they both accommodate user-defined functions (UDF) and exploit parallelisation and data streaming.

The information needed to describe a Collection is still work-in-progress. The aspects being considered include:
1. Constraints on the members, e.g., they are all instances of particular Concept.
2. The structure, e.g., set, list, array, table/relation, dictionary.
3. Is the order significant and implicit?
4. The location(s) and representation(s) of an instance.
5. Population statistics, current numbers, growth rate, distribution of members' sizes.
6. Enumeration mechanism.
7. Parallelisation patterns.
8. Anticipated forms of change, e.g., a curated archive normally only adds to its catalogue.
9. Mutability.

Table A2.4a.iv: Library of Concepts that will be provided for Collection production, management and use in every DKB instance. See {Levray 2020] for details.

Concept name	Description
kb:Collection	A number of entities that may be treated together as well as individually. A minimum requirement is to be able to enumerate the members.
kb:Directory	A specialisation of Collection corresponding to files in a storage system. This is necessary to support incremental introduction of Collections, as many Collections are currently assembled using directories.
kb:List	A specialisation of Collection as a sequence.

Table A2.4b.iv: Library of Concepts that may eventually be provided for Collection production, management and use in every DKB instance. Need and resources will determine which ones.

Concept name	Description
kb:Set	A specialisation of Collection that excludes duplicates and may be parallel processed.
kb:Dictionary	A set of String, Any pairs, looked up by supplying String values.
kb:Catalogue	An (often autonomously updated) Collection of items. It may be curated and managed, e.g., as part of a curation and archival service. It may be local and temporary.

The functions that are useful for Collections are many and varied. Exploring precisely what is useful and necessary is a research topic that cannot be fully explored in the context of DARE, i.e., they include:

1. *Bulk operations*, such as set union, intersection and difference, list concatenation and loss-less compression to a Data instance, and the inverse. These may lead to a complete algebra, e.g., the relational, in the long term.
2. *Membership operations*, e.g., insert, remove, exclude fc1, include fc2, the latter two include or exclude those that satisfy the boolean criterion supplied.
3. *Repetition*, as in **do** f(m) **for all** m **in** M, or **do all** f(m) **for all** f **in** F **and all** m **in** M, the latter being a significant saving, when bringing each m to a place for computation is expensive.

The provisional initial plan is a small sample of the potential space and it will be co-developed with applications and should need to design patterns aiding self-sufficiency. Implementations may be virtual and lazy to achieve optimisations.

Table A2.5a.iv: Additional functions applicable to Collection instances in a DKB. See [Levray 2020].

Function	Description
newInstance	Introduce a new Collection or Collection specialisation instance. By default it starts empty, but if supplied with an initial expression yielding a Python collection it will populate the instance from that collection, subject to any prevailing criteria.
insert	Insert the supplied member (collection of members) into the instance subject to criteria and ordering if specified.
remove	Remove the identified member (collection of members) from the instance.
apply	Apply the supplied function or Method to every member of the instance. This may produce a map as a result, or a reduce as a result or ...
supply	Given a Method with n inputs, supply the members from n Collections so that the members from the ith Collection are fed to the ith input and gather the result(s). **NB** This parameter-space exhaustive exploration or cross-product correlation can generate a very large volume of work and output.
doall	Apply a Collection of Methods, e.g., a set of validation checks, to an instance.

The above are an extensive body of work and may only be undertaken gradually. There, the speculative Table of possible extras is omitted.

Sundry Concepts

These are Concepts that have proved necessary when working on other Concepts.

Table A2.4a.v: Library of Concepts that will be provided as sundries that have emerged as necessary for use in every DKB instance. See [Levray 2020] for details

Concept name	Description
kb:Person	A representation of an individual. Instances of Person will hold what the KB needs to know about a user in order to support that user and their *modus operandi*. It may also accommodate standard attributes needed by external administrations or recommended by DCAT or a relevant DCAT-AP. These instances will be persistent for at least as long as the KB needs to recognise an individual over repeated visits and serve that individual taking into account prior visits. Eventually, these may be used in a KB authorisation system, but not during DARE. A Person instance may contain formal AAI-related information, a preferred public name and an official PID, see the end of A2.2.
kb:Session	A period during which an individual interacts with the system via one interface or during which a Method enactment interacts with the system via one protocol. Each instance of Session will represent one user logging in, working an arbitrary period of time and then ending the session. Similarly, an instance may be constructed when a queued Method runs, to record the AAI information

	governing that enactment, as the Session where the enactment was initiated may no longer be active, or it may submit a series of run requests, potentially using different AAI credentials.

Table A2.4b.v: Library of Concepts that may be provided as sundries. This depends on need and resources.

Concept name	Description
kb:Group	A set of individuals and groups, e.g., those authorised to do something
kb:Software	A body of code treated as an entity [Garijo *et al.* 2019].
kb:Service	A (web) service running for a period to deliver some functionality.
kb:Rule	A rule that specifies what should be done - generated Obligation instances (normally by a human) when something else is done (normally by software steered by a human).
kb:CompReq	Computational requirements, normally of a Method (see above).
kb:CompCap	The CompReq-meeting capabilities of a particular service.
dare:Container	A Docker container normally managed and described by Kubernetes (§5.2)
dare:Deployment	A graph of interconnected Container instances placed on instances of a Hosting Service (§5.2).

There are, as yet, no specific functions for these Concepts that differ from those generally available.

Built in types as Concepts

As always, these are provisional lists. They need to include all of the primitive types in use for which the attribute values are stored as a literal. They are all immutable.

Table A2.4a.vi: Library of Concepts that will be provided to incorporate standard types in every DKB instance. See [Levray 2020]. Others will be added as needed for attribute values.

Concept name	Description
kb:Integer	The mathematical concept's representation as a Concept.
kb:Real	The mathematical concept's representation usually using floating point
kb:String	A standardised sequence of standardised characters.
kb:Time	A standardised representation of UTC time (as users are geographically

	distributed). Communities may have different ones. Users will normally have local time and may prefer to see local times. The KB does not deal with that; user facing code will, but the KB may hold user's preferences.

A2.5 Coexisting information subsystems

Table A2.8: Functions in the registry API. A description of some of the data required for each function can be found in the GitLab link: https://gitlab.com/project-dare/dare-api

Function	Description
login	Get dispel4py registry credentials by logging.
create_folders	Create the working environment
get_auth_header	Return the authentication header
get_workspace	Get a workspace URL by name
create_workspace	Create a workspace using dispel4py registry api
create_pe	Create ProcessingElement / dispel4py workflow using d4p registry api
create_peimpl	Create ProcessingElement/ Workflow Implementation using d4p registry api
auth	Generate user "access token" / Simulate user login
submit_d4p	Spawn mpi cluster and run dispel4py workflow
debug_d4p	Debug a dispel4py workflow in "playground mode"
exec_command	Allow for running a command in "playground mode".
upload	Upload data into a working environment
myfiles	List the uploaded files.....
download	Download a file using exec-api filesystem reference.
delete_workspace	Delete a workspace
submit_specfem	Spawn mpi cluster and run specfem workflow)
my_pods	Return user created pod properties (name and status)
send2drop	Upload a file from the exec-api shared filesystem to the project-dare b2drop account in order to get a shareable link for a single file
pod_pretty_print	Monitor the container status

monitor	Monitor a dispel4py workflow run

A2.6 Planning DKB development

Our challenge is to find the best steps to develop and deploy an effective, easily used and integrated DKB as described in §4.2. We need to take an agile approach to deciding on the functional priorities while choosing implementation paths, while retaining a direction that leads towards a powerful and complete DKB. Initial suggestions follow and are depicted with some scheduling and interdependency suggestions in Figure A2.4:

These will be radically revised during the Toulouse plenary. The order is significant.

1. Develop a releasable version of the semantic data catalogue. {FRAUNHOFER + NCSRD} [M26]
2. Align the registry workspace and DKB Context designs and implementations and then deploy a version of the platform that supports that alignment for evaluation. {UEDIN+NCSR} [M26]
3. Compose registry and DKB, perhaps by proxying the registry in the DKB and add to the release. {UEDIN+NCSR} [M27]
4. Develop a strategy for harmonising with the data catalogue. Possibly via partial delegation, e.g., query and existing metadata functionality in the data catalogue and easy additional data in DKB. {NCSR + UEDIN} [M27]
5. Agree the model for Python programmers to interact with the DKB, the criteria for automatic additions to the conceptual library & API. Whether to go directly to a REST web service wrapping the DKB engine & storage. Used via a Python library programmers instal and use in the normal way. {UEDIN + KIT + CERFACS + KNMI} [M28]
6. Develop the Concept and Method support and test it in with WaaS WP4 developers to support:
    a. Descriptions of methods using simple Python scripts, i.e., a text-based system.
    b. User support for authoring, testing and moving into production such methods.
    c. Application-expert use of these to parameterise and control such methods (application developers and experts should get all the way from idea to production-ready release into the available API without help from IT experts.) {UEDIN + ?} [M28]
7. Release second version of the Concept library emerging from the above and supporting operations on Concepts that can be used in the above methods. {UEDIN + NCSR} [M28]
8. Evolve the above to describe and handle PEs working in conjunction with the dispel4py optimisation effort. This will include:
    a. All necessary properties of input and output streams that limit or open-up optimisation.

b. All necessary properties of the encapsulated algorithms, e.g., whether they accumulate state.
c. Optimisation-costs input data mined from the provenance data from previous runs.
d. Platform and subsystem performance and capability data - hand crafted.
e. Plans for incremental deployment and decision making. {UEDIN + ? + FRAUNHOFER + KIT} [M30]

9. Combine the above with selective implementation of virtualised and lazy Collections of data:
   a. Include operations to create, manage and use those collections
   b. Include at least one lazy mapping strategy
   c. Integrate the above optimisation to co-optimise the data handling and workflow enactment in a way that is scalable with data unit size and with the number of data units. {UEDIN + CERFACS + NCSR + ?} [M32]
10. Third release of the conceptual library and associated API.

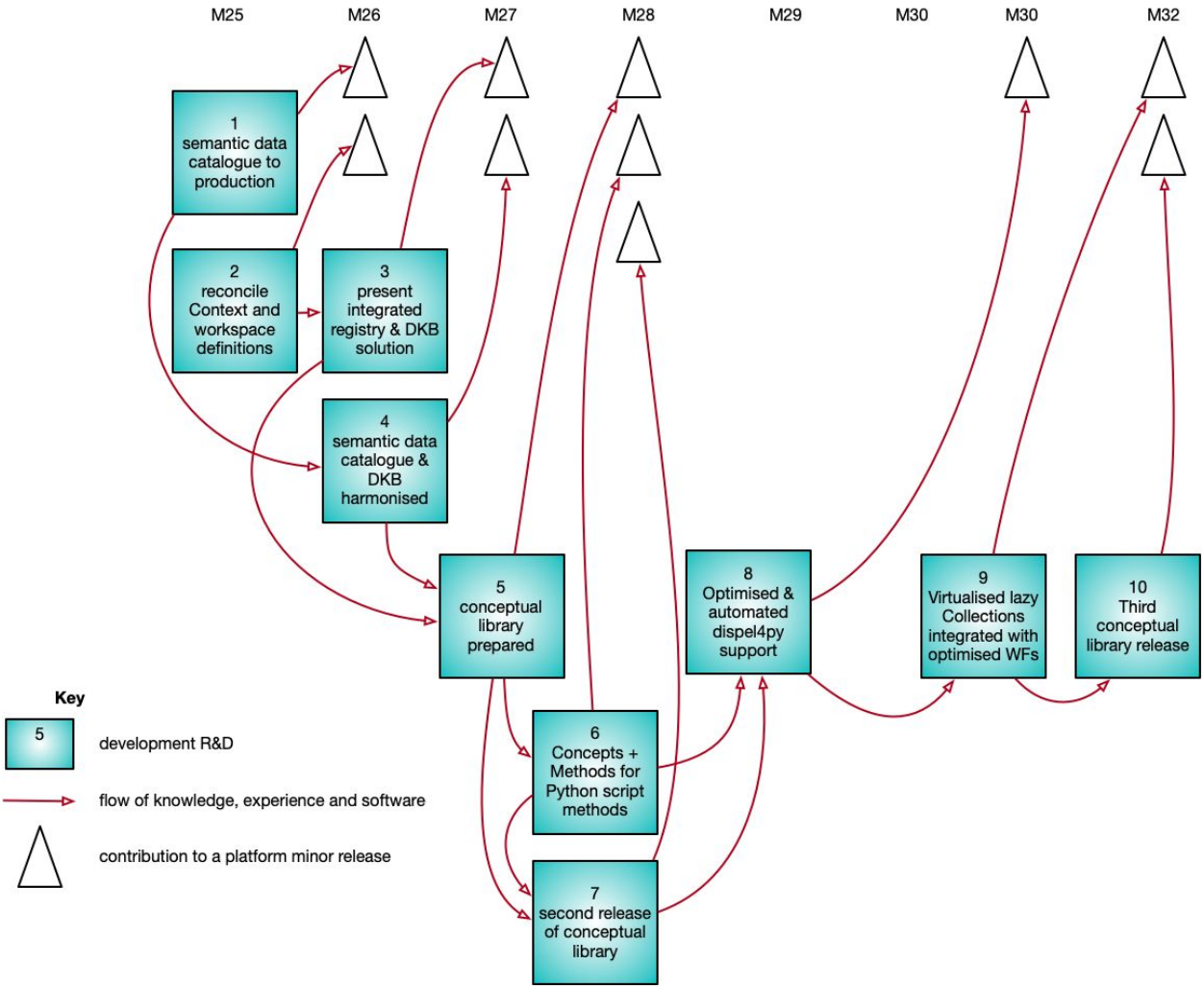

Figure A2.4: Possible plan for developing the DKB to provide key functionality and be well integrated. Timing and order needs discussion and clarification. Groups engaging in the work need to be identified.

This poses a number of substantial and critical questions:

1. Should the DKB support a SPARQL endpoint?
2. Can we use the prototype before we have wrapped it as a web service with co-located Python?
3. Can we provide a Python library that research developers understand and use?
4. Will there be a useful part of that understandable *and used* by application experts?
5. Do we operate as a proxy for the registry?
6. How do we collaborate with the data catalogue?
7. How do we progress alignment with provenance?
8. What is the message queuing system we use to coordinate the pillars? (and contemporaries?) ZeroMQ?
9. What are the events we send between pillars? (and contemporaries?)
