Data-intensive architecture for scientific knowledge discovery

Malcolm Atkinson · Chee Sun Liew · Michelle Galea · Paul Martin · Amrey Krause · Adrian Mouat · Oscar Corcho · David Snelling

Abstract This paper presents a data-intensive architecture that demonstrates the ability to support applications from a wide range of application domains, and support the different types of users involved in defining, designing and executing data-intensive processing tasks. The prototype architecture is introduced, and the pivotal role of DISPEL as a canonical language is explained. The architecture promotes the exploration and exploitation of distributed and heterogeneous data and spans the complete knowledge discovery process, from data preparation, to analysis, to evaluation and reiteration. The architecture evaluation included large-scale applications from astronomy, cosmology, hydrology, functional genetics, imaging processing and seismology.

Keywords Knowledge discovery · Workflow management system

1 Introduction

The scientific community is facing an imminent flood of data from the next generation of experiments and simulations, as recognised in [20]. The demand for data-analysis tools and computing resources is increasing even faster than the data volume, as more sophisticated algorithms are used which comprise more and deeper analysis [17]. Beside the data deluge, another challenge is the diversity and complexity of both applications and execution environment. New scientific applications involve execution on distributed and heterogenous computing resources across organisational and geographical boundaries, processing gigabytes of live data streams[1] and petabytes of archived and simulation data,[2] in various formats and from multiple sources. Managing the data deluge not only requires larger storage space and more computational power, but also demands new technologies, e.g. scalable data-processing algorithms that can handle massive datasets, new data-management technologies for distributed and heterogeneous data sources and high-speed networks for transferring large volumes of data [5, 6, 15, 16, 34].

There is a consequent growth in the number of research applications and researchers who wish to exploit data-intensive methods. Hitherto, the solution has been to engage experts to build the tools in each case. It is infeasible to grow the body of experts sufficiently quickly. Consequently, our research focuses on raising the level of discourse so that the work of experts can be more easily reused and the domain scientists can be more self-sufficient. This requires a new architecture to separate concerns and engineering advances to replace hand-crafted optimisation.

The exploratory nature of scientific experiments requires fast modelling, prototyping and easy enactment. In general, there are three types of users involved in running scientific workflows: domain experts, data-analysis experts and data-intensive engineers. *Domain experts* are scientists who are interested in scientific discovery,

[1] The Square Kilometer Array (http://www.skatelescope.org) will generate about 200 GB of raw data per second and the LOFAR (http://www.lofar.org/) low band antennas generate 1.6 TB raw data per second.

[2] The Euclid Imaging Consortium (http://www.ias.u-psud.fr/imEuclid) will generate 1 PB data per year and the Large Synoptic Survey Telescope (http://www.lsst.org) will generate several petabytes of new image and catalogue data every year.

and who use various tools to interpret their experiments. *Data-analysis experts* are knowledge discovery workers who are expert in extracting information from data. They know the data-analysis methods, data-mining techniques and statistical methods, which help domain experts to understand their data. They have the skills to design data-analysis algorithms, but may not be familiar with handling distributed computation. Thus, they rely on the computer scientists, software engineers and systems engineers who are knowledgable in distributed computing infrastructure to manage the data and computations. This last type of user is made up of *data-intensive engineers*. All three groups work well in their own domain, but may or may not be capable of performing each other's tasks. Domain experts know what data are needed for flood forecasting, but may not know how to retrieve and integrate data from all distributed monitoring stations. The data-intensive engineers can execute the forecasting workflows in an optimised environment, provided that the data-analysis experts have already created the required prediction modules. The successful story of the Sloan Digital Sky Survey[3] is a tremendous combination of the efforts of astronomers and database engineers, to design the data-handling mechanism of the large database built up over the years.

Our architecture achieves the separation of concerns with three crucial components: a novel and powerful process engineering language (DISPEL), a registry that provides rich semantic descriptions, and an extensible and robust enactment platform that supports the data-intensive computations on distributed and heterogeneous environments. This architecture was developed under the ADMIRE project[4] and its prototype is available in open source[5]. Related work is summarised in Sect. 3. Section 4 presents the data-intensive architecture. The DISPEL language is described in Sect. 5, followed by the data-intensive platform in Sect. 6. Section 7 discuss the evaluation of the architecture. Section 8 concludes with an assessment of progress and the plans for further work.

2 Data-intensive applications

Our data-intensive approach was stimulated by growing data-handling challenges in diverse applications. We draw our examples from a mix of scientific (Astronomy, Biology, Seismology, Environmental Management) and business domains (Customer Relationship Management).

Quasar Classification (Astronomy) Investigation into whether the use of more than one sky survey improves the accuracy of quasar classification.

Gene Annotation (EURExpress-II) Machine learning for automated annotation of mouse-embryo gene-expression images.

Seismic Ambient Noise Processing (Seismology) Automated cross-correlation and aggregation of distributed seismic wave forms.

---

[3] Sloan Digital Sky Survey: http://www.sdss.org/.

[4] ADMIRE project: http://www.admire-project.eu/.

[5] ADMIRE prototype: http://sourceforge.net/projects/admire/.

Rainfall Prediction (Radar) Short-term prediction of rainfall using radar.
Reservoir Characteristics Prediction (Orava) Prediction of water reservoir level and temperature.
Reservoir Inflow Prediction (SVP) Prediction of inflow to a water reservoir.
Customer Churn Prediction (ACRM Churning) Data mining to predict customers most likely to leave.
Customer Cross-selling (ACRM Cross-selling) Analysis of services/product purchase correlations.

Between them, these use cases span all key steps in the knowledge discovery process, including data selection, integration, pre-processing (including cleaning, feature selection, transformation), analysis and mining, interpretation and presentation of results, and reiteration over the whole process.

Note, that though these steps are common to the KDD (Knowledge Discovery in Databases) process, several of our use cases have to work on and consolidate heterogeneous data from distributed sources. For instance, the Seismology use case integrates data from different data centres, where the raw data in stored in 100s of 1000s of files in a file system, but associated metadata is stored in a database; a further complication is that different data centres are likely to use different file systems and database schemas. The Astronomy use case integrates data from two sky surveys stored on different databases, to perform quasar classification. In the Radar rainfall prediction use case, radar images in binary format are selected, extracted and processed, and then merged with meteorological sensor data from a relational database. All the use cases are further described in detail in [18], in conference and journal publications,[6] and in Part IV of [1].

3 Related work

A wide range of workflow management systems has been developed over the last two decades, e.g. Pegasus [11], Kepler [28], Taverna [21], Triana [35], Swift [40], Trident [3] and Meandre [27]. Reviews of these systems can be found in [8, 10, 39]. They use a bottom-up approach in describing the experiments. Executable programs and web services already exist in most cases. A workflow specifies their composition and the required interconnections. Lack of separation between the abstract workflow and the implementations mechanism introduces dependency between the experiments and the execution platforms. As a consequence, workflows need to be rewritten each time platforms or resources change. What the community needs is a workflow language that provides separation of concerns, which supports creativity of both workflow creation and platform implementation through a standard and robust mapping interface.

Some of these systems have their own intermediate language, e.g. Meandre's ZigZag [27], Taverna's Simple Conceptual Unified Flow Language (SCUFL) [30], Kepler's Modelling Markup Language (MoML) [25] and Swift's SwiftScript [37].

[6] ADMIRE publications: http://www.admire-project.eu/admire-library/index.html.

From our viewpoint, many of these languages allow their users to say too much, in the sense that they can specify nitty gritty detail which exposes ephemeral implementation mechanisms. This reduces the platform independence, which is the foundation of our strategy for providing separation of concerns, diversity, resilience to change and optimisation using local and up-to-date information. More importantly, these XML-based coding languages, including other workflow description formats, e.g. DAX used in Pegasus, are mainly used for the communication within the enactment engines. We argue for a language designed to support collaboration between experts when solving data-intensive problems by facilitating precise dialogue about the processes required. It is also one mechanism for man-machine and workflow-distribution communication. The human-readable (non-XML and exclusion of execution details) orientation of DISPEL makes it an ideal notation for discussing, publishing, teaching and implementing data-intensive methods.

There are many commercial workflow systems, with BPEL as a standard [23], as well as standards aimed at managing business processing in a variety of domains, e.g. ebXML [13]. The majority of commercial workflow systems are tuned to orchestrating business processes, including human activity, whereas the scientific workflow systems are more orientated to controlling computations and managing data movement. They therefore almost invariably give a high priority to identifying and exploiting data dependences. Data streaming is key to scalable and continuous computation—a facility relatively rare in workflow languages, e.g. it was recently added to Kepler [4].

The streaming-process model has a great capability to perform data-intensive computations with modest computing resources with the use of one-pass algorithms, also known as streaming algorithms. A good survey of data-streaming algorithms and applications used in various domains, e.g. network traffic monitoring, text mining, and real-time streaming applications on the web, can be found in [29]. Another significant advantage of the streaming-process model is the capability of performing *parallel execution* of independent tasks [19]. The advance of multicore architectures and high-speed communication networks has opened up the opportunity of executing streaming tasks in a parallel and distributed environment. The creation of high-level languages for streaming applications (e.g. StreamIt [36]) and stream-processing middleware (e.g. SPADE [14] and Granules [32]) enables users to write applications that are automatically parallelised and mapped onto multiple computing resources. Our research adopts the streaming-process model for workflow enactment.

4 Data-intensive architecture

The architecture has three levels, as shown in Fig. 1. The upper layer (the *tool* level) supports the work of both domain experts and data analysis experts. It houses an evolving set of portals, tools and development environments, sufficient to support the diversity of both of these communities of experts. The lower layer (the *enactment* level) houses a large and dynamic community of providers who deliver data and data-intensive enactment environments as an evolving infrastructure (called the *data-intensive platform*), which supports all of the work done in the upper layer. Most

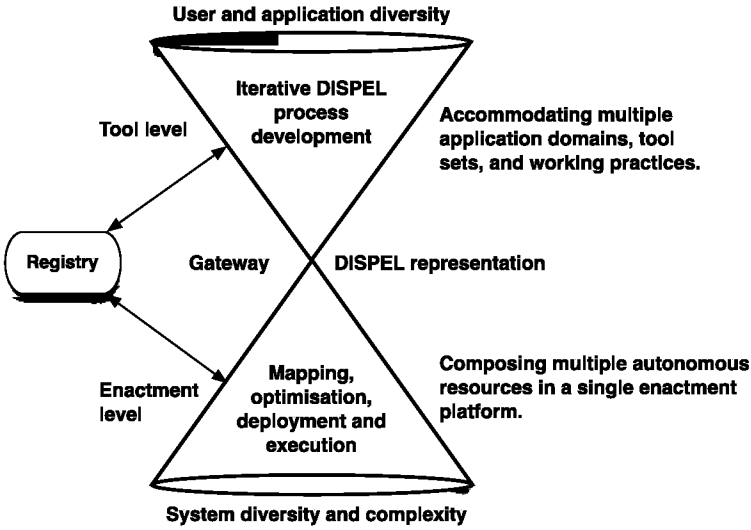

Fig. 1 Hourglass architecture separating the complex contexts of users and providers

of the work done by data-intensive engineers goes on here. Data-analysis experts can also develop generic libraries optimised for a provider's enactment environment at this level should they so desire.

The crucial innovation is the neck of the hourglass, which is a tightly defined and stable interface through which the two diverse and dynamic upper and lower layers communicate. This has a minimal and simple protocol and language, ultimately controlled by standards, into which the upper and lower communities can invest, secure in the knowledge that changes to this interface will be carefully controlled. This interface is analogous to the HTTP and HTML interface that has powered the Web's technological and business successes. It has separated the enormous body of business and technical innovation that lies behind the interface to respond to all the diversity of Web requests (corresponding to the enactment level) from the equally significant body of tools and portals that generate those requests and handle responses (corresponding to the tool level).

We have explored our interface by creating a new workflow composition language, named DISPEL. The primary function of DISPEL is to express how a data-intensive application uses processing elements (e.g. that provide noise filtering algorithms and perform pair-wise cross correlation of time-series data), and how these elements communicate with each other. In other words, DISPEL is a language for expressing a directed graph, where processing elements represent the computational nodes and the flow of data between them is represented by connections. Thus, DISPEL provides an abstraction technique for a data-streaming execution model. At the lower level, DISPEL also handles validation, and provides the required model for carrying out workflow optimisations. It is designed to be comprehensible to expert humans so that it is also a medium for dialogue between experts. It is hoped that the development of DISPEL will introduce new ideas into the continuing dialogue around workflow composition languages.

The architecture also has its own registry which is used to store descriptions of all components available for the construction of data-intensive tasks; the registry serves to relate the lightweight entities used by the tool level to the various possible implementations of those entities at the disposal of the enactment level. Thus the semantic descriptions stored in the registry provide consistent functionality across the tool and enactment levels. The registry is a key component of the architecture for three reasons:

1. it holds and validates all of the descriptions discussed above, and expands as descriptions evolve;
2. it acts as a consistency foundation and database for all of the subsystems (tools, language processing and enactment) in the architecture; and
3. it provides a foundation for sharing and cooperation using web-based tools, ontologies and information models.

5 A Data-Intensive Systems Process Engineering Language

The Data-Intensive Systems Process Engineering Language (DISPEL) is a data-flow workflow construction and optimisation language for distributed data-intensive applications (for a full definition see the DISPEL reference manual [24]). DISPEL is imperative, rather than declarative (making it similar to Pig Latin [31]), so a DISPEL script essentially describes how to *construct* a workflow rather than merely specifying the workflow itself directly—this allows the use of imperative constructs such as iteration, selection and functions to be employed to concisely specify arbitrarily complex workflows. It also means that scripts can account for external factors at execution time, which may affect the composition of workflows. DISPEL also permits the modification of existing workflow elements in order to specify new elements, as well as the arbitrary composition of such elements in order to create more powerful composite elements.

The key idea however is that DISPEL definitions can be mapped onto arbitrary computational platforms (whether they be based on e.g. OGSA-DAI [12], Hadoop or Dryad [22]) by merely specifying the logical properties of workflow components and how they connect together rather than to a specific platform (such as for Pig Latin, Sawzall [33] or languages like ZigZag, SCUFL or MoML mentioned in Sect. 3); as long as a system can provide implementations of certain core elements, the correct generic behaviour can be inferred from the script, with optimisation deferred to the platform. A DISPEL workflow is thus an abstract network of processing elements through which data can be streamed:

- A *processing element* (PE) describes a persistent computational entity. Every PE has a number of connection interfaces through which data is either consumed or produced. Data is streamed between PE instances via connections made between output and input interfaces.
- A *connection* streams data from one output interface to at least one input interface.

A DISPEL script may declare functional or abstract definitions or describe a workflow. A script is submitted to a *gateway* and interpreted using a registry as described

```
package dispel.manual {
  // Import existing PE from the registry and define domain namespace.
  use dispel.db.SQLQuery;
  namespace db
    "http://dispel-lang.org/resource/dispel/db";

  // Define new PE type.
  Type SQLToTupleList is
    PE( <Connection:String::"db:SQLQuery"      expression> =>
        <Connection:[<rest>]::"db:TupleRowSet" data> );

  // Define new PE function.
  PE<SQLToTupleList> lockSQLDataSource(String dataSource) {
    SQLQuery sqlq = new SQLQuery;
    |- repeat enough of dataSource -| => sqlq.source;
    return PE( <Connection expression = sqlq.expression> =>
               <Connection data       = sqlq.data> );
  }

  // Create new PEs.
  PE<SQLToTupleList> SQLOnA = lockSQLDataSource("uk.org.UoE.dbA");
  PE<SQLToTupleList> SQLOnB = lockSQLDataSource("uk.org.UoE.dbB");

  // Register new entities (dependent entities will be registered as well).
  register SQLOnA, SQLOnB;
}
```

Fig. 2 A DISPEL script which constructs a new workflow element

in Sects. 6.2 and 6.3. DISPEL is statically-typed, with strict, call-by-value evaluation of expressions.

5.1 Anatomy of a DISPEL script

DISPEL uses a notation similar to that of Java. Figure 2 demonstrates the four main stages in constructing and registering new workflow elements:

- The definition of abstract types (`SQLToTupleList`). Abstract PE types can be used to define classes of PE which can be inserted into a workflow; any implemented PE matching the abstract type can be used.
- The specification of a constructor for an abstract type (`lockSQLDataSource`). As well as being used to describe particular classes of PE, abstract types can be implemented using compositions of existing components (such as `SQLQuery`), producing new composite PEs.
- The construction of new processing elements (`SQLOnA` and `SQLOnB`). Multiple implementations of a given abstract type can be created using different constructors or different parameterisations of the same constructor.
- The registration of components for later use. Dependent components are also registered, allowing new abstract types, constructors and constructed types to be recovered for later workflows and shared with other users.

DISPEL benefits from three distinct type systems: *language* types refer to the types of variables in scripts; *structural* types refer to the syntactic structure of data el-

```
package dispel.manual {
  // Import existing and newly defined PEs.
  use dispel.manual.SQLOnA;
  use dispel.lang.Results;

  // Construct instances of PEs for workflow.
  SQLonA  sqlona  = new SQLOnA;
  Results results = new Results;

  // Specify query to feed into workflow.
  String query = "SELECT * FROM littleblackbook WHERE id < 10";

  // Connect PE instances to build workflow.
                        |- query -| => sqlona.expression;
  |- "Little Black Book, page 1" -| => results.name;
                        sqlona.data => results.input;

  // Submit workflow (by submitting final component).
  submit results;
}
```

Fig. 3 A DISPEL script which submits a workflow

ements streamed between PE instances; *domain* types refer to the semantic (principally ontological) meaning assigned to data elements. Language types (such as `String` designating a string of characters and `Connection` designating a connection object) permit validation of operations in scripts before execution, whilst structural types (`String` designating a string of characters and `[<rest>]` designating a list of tuples of any internal composition) permit validation of connections between workflow components. Structural types can be arbitrarily complex compositions of arrays, lists and tuples; processing elements can consume and produce arbitrary units of data (for example by use of the `Any` type). Domain types (such as `"db:SQLQuery"` and `"db:TupleRowSet"`) can be associated with external ontologies (such as found at `"http://dispel-lang.org/resource/dispel/db"`) and can be freely attached to data elements of any constructed structural type.

Figure 3 demonstrates the process of building and submitting a workflow to a gateway:

- Components (`SQLOnA` and `Results`) are imported from a registry and the instantiated (`sqlona` and `results`).
- The workflow is then constructed by connecting together all component instances and feeding in any initialisation data (`query`). DISPEL permits the denotation of arbitrarily complex data streams, bridging the gap between the script and workflow data-spaces.
- Finally, the workflow is submitted by submitting any part of the workflow.

DISPEL is oriented around data-flow rather than control-flow. As a result, no specification of how data should be produced or consumed is required; instead, data is pushed out of or pulled into processing elements based on the balance of their respective implemented behaviours, regulated by the enactment platform.

6 Data-intensive platform

The lower layer of the data-intensive architecture (see Fig. 1), the enactment level, is intended to host a large and dynamic community of providers who deliver data and data-intensive enactment environments as an evolving infrastructure, called the *"data-intensive platform"*, that supports the work of the upper layer. The DISPEL request is produced using facilities at the tool level and then sent to a gateway, which acts as the entry point to the data-intensive platform. A data-intensive platform comprises: (a) an *application development environment* (including libraries of processing elements, functions, and data types), (b) a *gateway* as the entry point of enactment which accepts DISPEL request, (c) a DISPEL *language processor* that compiles the DISPEL request into graph representation, (d) an *enactment engine* that optimises those graphs, deploys them, executes them in a controllable framework that permits interaction with the end user, and finally terminates them and cleans up the environment, (e) *execution engines* that deploy and execute workflows, and (f) *data sources* that are connected and made available through this platform.

Once a DISPEL request for enactment has been received, it has to be transformed and mapped to selected parts of the data-intensive platform. This involves analysing the request and determining whether it can be run, whether it is best run on the local platform, or better delegated to another, or whether it should be partitioned and each part delegated to platforms that better match its balance of resource requirements. Additionally, the data-intensive platform takes full responsibility for buffering and optimising the flow of values along each connection, e.g. passing by reference when possible; or serialising, compressing and encrypting long haul transmission. The system will automatically buffer, spilling to disk when this is unavoidable.

6.1 Application development environment

Currently, the ADMIRE workbench is the primary development platform for DISPEL workflows, used by data-analysts and data-intensive engineers, and occasionally by domain experts. Data-analysts include the software developers working on knowledge discovery projects and knowledge discovery experts who are tasked with implementing solutions for domain experts. Data-intensive engineers may use the workbench to investigate and optimise movement and computation of data, or analyse the effectiveness of PEs, functions and patterns. They need a complete and familiar environment to become productive quickly.

The workbench is based on the Eclipse platform,[7] which provided the project with a professional, feature-rich IDE that would have otherwise taken years to build. The project developed plug-ins to support DISPEL development in the workbench. The primary tools for editing workflows are the DISPEL-aware text-editor, which provides syntax and error highlighting for workflows, and the graphical DISPEL editor which provides a simple GUI that can be used to quickly construct workflows. Other plug-ins interface with the platform itself, for instance workflows can be submitted

[7] Eclipse: http://www.eclipse.org/.

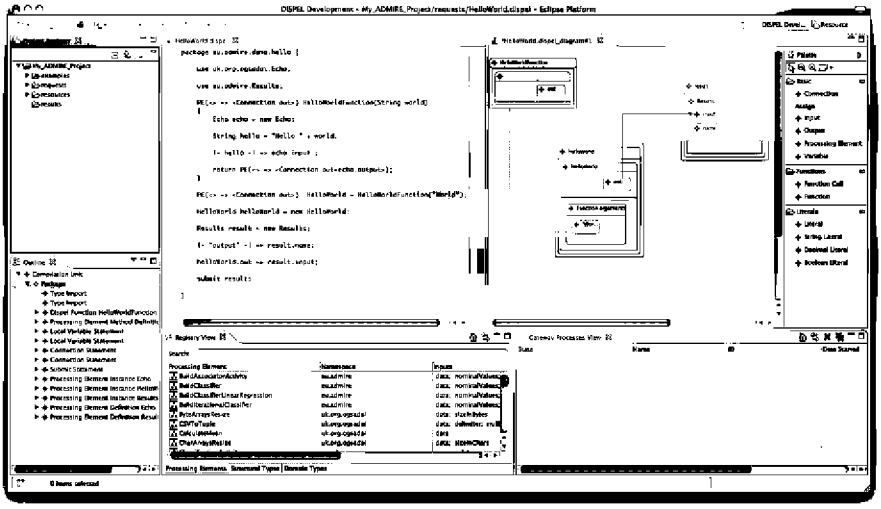

Fig. 4 The workbench

directly from the workbench and their progress monitored from the process viewer. The results from completed workflows can be viewed in a range of visualisers, which can interpret results as charts and diagrams as well as plain text. When building workflows, the registry plug-in allows users to quickly identify available PEs and verify their inputs and outputs. A screenshot of the workbench can be seen in Fig. 4.

New users often use the GUI editor to create workflows. This environment allows them to quickly "plug" together workflows in a few mouse clicks. The GUI also provides the user with a visual indication of the "flow" of a DISPEL script. When they hit the limitations of the GUI editor, they progress to the text editor, which supports a richer set of commands and constructs.

The workbench is designed to reduce the burden on new users and to provide developers with a familiar environment that is rich in features. It allows developers to adapt the environment (for example by adding PEs and plug-ins to handle new data types).

6.2 Registry

The registry contains descriptions of all categories of DISPEL components (processing elements, connections, types and functions). It has two main roles: to provide persistent storage and communication regarding the elements of DISPEL sentences between the tools used for development and the enactment systems, and to support communication, information sharing and collaboration among the various communities using the architecture.

For example, a data-analysis expert using the application development environment may be interested in retrieving descriptions of all of the processing elements that are capable of performing a specific transformation or analysis, so as to select

one to incorporate into a DISPEL script. A domain expert may be interested in knowing functions that have been used to perform a given data-analysis task, together with whom has used it already, and whether its use was successful. A data-intensive engineer may contribute to the registry libraries of widely used or carefully tuned components, and share information about the physical computational context.

As an example of the type of information recorded in the registry, both for human and system consumption, processing elements are described with:

1. A unique name (a URI).
2. A short natural language description.
3. An ontology-based classification of their purpose.
4. A precise description of their input and output connections, including their structural and domain types, as described in the previous section.
5. The consistency and propagation rules for structural and domain types in their input and output connections.
6. Their known relationships in the sub-type hierarchy.
7. Their patterns of data consumption and production.
8. Their termination behaviour and error modes.
9. Information useful for placing instances and optimising enactment.
10. Information about version relationships that may be used by automated change adapters.

The registry is also equipped with type propagation and checking functionalities that can be used by the application development tools or the enactment engines so as to verify that a workflow is well constructed (in terms of the types used in each connection) and to characterise the results of a workflow execution [38].

6.3 DISPEL processing model

There are four stages in the enactment process for data-intensive computations, as shown in Fig. 5:

1. DISPEL *Language Processing*, which includes parsing and validating a DISPEL program [7, 38] and creating the data-flow graphs.
2. *Optimisation*, which includes selection of PEs, transformation of the data flow graph, substitution of PEs, identification of available resources, and the mapping of PEs to resources.
3. *Deployment*, which includes translation into platform-specific form and initialising resources and connections.
4. *Execution and Control*, which includes instrumentation and performance measurement [26], failure management, delivering results and clean up.

This enactment framework provides a high-level abstraction of data-intensive applications. This is achieved through a *separation of concerns*, where the software details are abstracted at various levels: e.g., the application level, algorithmic level, and execution level.

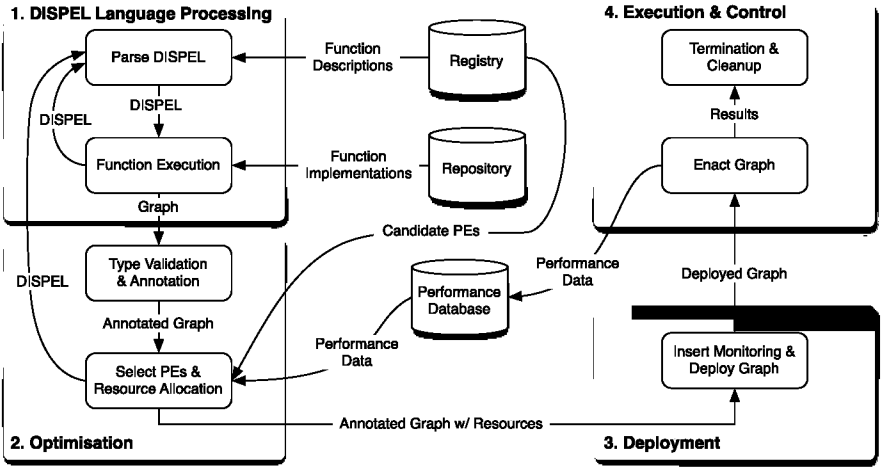

Fig. 5 Diagram that shows the steps involved in processing DISPEL programs

7 Evaluation of the prototype architecture

7.1 Experimental aims and method

We expect a data-intensive platform to:

- cover a variety of domains from academia, business and government;
- span the knowledge-discovery life cycle;
- cope with heterogenous data, large data volumes and multiple distributed data sources.

The ease with which an application may be defined, implemented, executed, extended or redefined, is an important factor. We therefore designed the evaluation of the ADMIRE prototype data-intensive architecture around the implementation of the eight use cases listed in Sect. 2, identifying and defining criteria that would help us judge whether the architecture was able to meet the requirements outlined above.

These evaluation criteria are shown in Table 1. Each use case tested several criteria, and each criterion was tested by several use cases. Use case owners and developers were asked to complete a questionnaire about progress made in the implementation. The questionnaire was also designed to capture the qualitative experience of collaborative work between the three different types of experts, and of using the architecture.

7.2 Discussion

A summary of the evaluation of the prototype architecture is presented in Table 2, which maps each use case to some of the evaluation criteria listed in Table 1. Full details of the evaluation are available in [2].

The criteria in Table 2 are all quantitative, and the figure in brackets beneath each criterion indicates the point at which we may consider the data-intensive architecture

Table 1 ADMIRE prototype architecture evaluation criteria

Criteria	Description
Data volume	The raw volume of the data
Dimensions	Number of rows in a data set and complexity (e.g. number of columns)
Data sources	This evaluates the integration of data from multiple sources
Concurrent processes	This evaluates the potential for parallelisation
Heterogeneous and physical sites	This is the integration of data that may be heterogeneous, and from different physical sites
Real-time	This is relevant to domain experts who wish to observe partial results from workflow while it is still being processed. This can potentially save much wasted computation
Sophisticated workflow	This tests optimisation and deployment of complex graphs
Use of abstract language patterns	This evaluates the re-use of common and complex data integration and mining patterns, such as all-meets-all, k-fold cross-validation and decision-tree building
Use by domain experts	This evaluates quick and easy interfaces for domain experts
Steps in workflow	This is the number of PEs in the longest path from source data to output results in a DISPEL graph

Table 2 Evaluation criteria mapped to use cases. Colour indicates how well suited a use case is to evaluate a specific feature of the architecture–*white* is excellent, *light grey* is borderline, *dark grey* is poor

	Data volume (>1 TB)	Dimensions (>1 M tuples)	Heterogeneous data + physical sites ($x + y > 5$)	Steps in workflow (>20)
Astronomy	256 MB	65 M	1 + 2	
EUREXpress-II	170 MB		5 + 3	57
Seismology	20 TB	2.3 M	4 + 2	>20
Radar	200 GB	1 M	4 + 2	
Orava	500 GB	0.2 M	6 + 3	32
SVP	100 GB	0.2 M	3 + 2	25
ACRM Churning	1.5 TB	35 M	3 + 3	
ACRM Cross-selling	1.5 TB	60 M	3 + 3	8+

to be presented with a real challenge. For instance, a data set is considered to be large and sufficiently challenging to the architecture, if the raw volume is larger than one Terabyte. The colour of a cell of the table is also an indication of the extent to which a feature of the architecture is tested by a use case—white is well-tested, light grey is moderate, and dark grey indicates a poor test.

An element of the architecture worth noting is the use of abstract language patterns. Several use cases originally were defined in terms of primitive PEs (Orava, Radar, SVP, ACRM Churning and Cross-selling). However, as DISPEL evolved to

meet the requirements of users, almost all use cases were refined and then rerun using higher-level constructs such as composite PEs and functions.[8] These enabled easy re-use of common patterns between different workflows, for instance, the reuse of data-filter functions between the Orava, Radar and SVP use cases.

Since the recognition of the different categories of experts is a key factor in the design of the architecture, it is important to question how and to what extent a separation of roles and concerns between the categories has been achieved. Use case owners and developers report that in practice this separation of roles provides an effective approach.

Domain experts discuss the data processing tasks with data-analysis experts, without a requirement for the domain expert to understand DISPEL. Once a task has been designed and executed, and a portal developed, a domain expert may use the portal to specify problem parameter values, execute predefined DISPEL and visualise results (e.g. the SVP use case). If required, the domain expert can iterate over this process, observing results and tuning parameter values (e.g. ACRM Cross-selling).

No restrictions are imposed on a domain expert with regards to their requirement for specific software (e.g. Seismology incorporated domain-trusted Python libraries), ways of accessing data (e.g. Astronomy built upon current web services used by astronomers for access to databases, or the type and location of data (six of the use cases integrated heterogenous data from different physical locations).

Data-analysis experts can use their preferred programming and scripting languages to specify algorithms that are later wrapped as DISPEL PEs (e.g. use of Ruby script in Radar and SVP), and tools such as an interactive development environment including the workbench DISPEL editor, process designer and registry viewer. DISPEL allows them the facility to simply extend workflows to solve different problems (e.g. extension of Seismology workflows for seismic interferometry currently in progress), or to reuse existing components in new workflows (e.g. composite PEs for reading data from binary files used in SVP and Orava).

Crucial interaction is sometimes required between a data-analysis expert and a data-intensive engineer, in order to resolve technical issues or improve performance (e.g. for ACRM Churning and Cross-selling).

Limitations were encountered by the developers, including a partly populated registry, lack of automated optimiser for handling parallel execution, lack of automated help in analysing errors in workflows, and a workbench that was under development and unstable. These limitations and other gaps were a necessary consequence of the nature of the project—an experimental development aimed at rapidly pioneering a new architecture, developing in so far as it was necessary to test new ideas. The successful implementation of the use cases, however, provides compelling evidence of the viability of this approach to data-intensive research and application—they demonstrate that the architecture provides a collaborative framework within which the different experts may interact, caters for both scientific and business applications, promotes the exploitation and exploration of large-scale distributed and heterogeneous data, and spans the complete knowledge discovery process. The further development of the architecture to a production level platform is discussed in Sect. 8.

[8] These are functions that when supplied with parameters such as PEs, generate graphs with those PEs in them. The graph is then treated just like any other.

8 Conclusion and future work

As is evidenced above, the architecture has proved very valuable as a collaboration framework for the three categories of expert: domain experts, data-analysis experts and data-intensive engineers, who employ and enable distributed data-intensive methods to advance science, commerce and government. It has shown the appropriate admixture of autonomy and interaction. The registry has proved its worth as a knowledge base for users and as a consistency enabler for the separate and distributed subsystems. The canonical language DISPEL proved to be precise and a suitably abstract *lingua franca* for data-intensive computing, from applications to platform engineering, and there is preliminary evidence that it can be efficiently enacted.

The prototypes are experimental—a sufficient implementation to test the ideas—but agile development to pioneer a radically new architecture has ineluctably meant that the majority of subsystems are not complete, have not had sufficient engineering to make them sustainable and have not tackled non-functional QoS issues, such as security and dependability in the presence of partial system failures. However, the architecture makes good use of standards, and of existing subsystems, such as OGSA-DAI [12], which are already established, well-engineered and sustained. So it is only some aspects of the total system that need further work in order to meet our expectations of using the data-intensive architecture in the longer term with many applications warranting the sustainability investment. Several projects (e.g. VERCE[9]) are continuing to develop and use the architecture.

Future work will include:

1. advancing the data-intensive tools (e.g. accommodating more development contexts and work styles, while meeting operational requirements);
2. advancing the canonical language (e.g. refining the DISPEL definition to be complete and consistent and exploring multiple enactment strategies);
3. advancing the enactment platforms (e.g. extending to heterogenous platforms from mobile handheld devices to large data-intensive machines, incorporating other execution environments, integrating with other workflow management systems, expanding the component libraries, exploiting automatic and dynamic optimisation); and
4. advancing the registry and ontologies (e.g. improving the abstract description notation, addressing the socio-economic issues, such as: controlled release for collaboration between peers, group identity and attribution, that have been explored in other platforms such as myExperiment [9]).

The architecture is a first step in developing sufficiently powerful and easily used environments for exploiting the emerging data bonanza.

---

[9]Virtual Earthquake and seismology Research Community e-science environment in Europe: http://www.verce.eu/.

References

1. Atkinson, M.P., Baxter, R., Besana, P., Galea, M., Parsons, M., Brezany, P., Corcho, O., van Hemert, J., Snelling, D.: The DATA Bonanza—Improving Knowledge Discovery for Science, Engineering and Business. Wiley, New York (2012). To be published
2. Atkinson, M.P., Galea, M., Liew, C.S., Martin, P.: Final report on the ADMIRE architecture, with an assessment and proposals for its development. Tech. rep., The ADMIRE Project (2011)
3. Barga, R., Jackson, J., Araujo, N., Guo, D., Gautam, N., Simmhan, Y.: The Trident scientific workflow workbench. In: Proceedings of the 2008 Fourth IEEE International Conference on eScience, e-Science '08, pp. 317–318. IEEE Comput. Soc., Los Alamitos (2008)
4. Barseghian, D., Altintas, I., Jones, M.B., Crawl, D., Potter, N., Gallagher, J., Cornillon, P., Schildhauer, M., Borer, E.T., Seabloom, E.W., Hosseini, P.R.: Workflows and extensions to the Kepler scientific workflow system to support environmental sensor data access and analysis. Ecol. Inform. **5**(1), 42–50 (2010)
5. Bell, G., Hey, T., Szalay, A.S.: Beyond the data deluge. Science **323**(5919), 1297–1298 (2009)
6. Berriman, G.B., Groom, S.L.: How will astronomy archives survive the data tsunami? Commun. ACM **54**(12), 52–56 (2011)
7. Buil-Aranda, C., Arenas, M., Corcho, O.: Semantics and optimization of the SPARQL 1.1 federation extension. In: Proceedings of the 8th Extended Semantic Web Conference on the Semanic Web: Research and Applications—Volume Part II, ESWC'11, pp. 1–15. Springer, Berlin (2011)
8. Curcin, V., Ghanem, M.: Scientific workflow systems—can one size fit all? In: Cairo International Biomedical Engineering Conference, CIBEC '08, pp. 1–9 (2008)
9. De Roure, D., Goble, C., Stevens, R.: The design and realisation of the myExperiment virtual research environment for social sharing of workflows. Future Gener. Comput. Syst. **25**, 561–567 (2009)
10. Deelman, E., Gannon, D., Shields, M., Taylor, I.: Workflows and e-Science: an overview of workflow system features and capabilities. Future Gener. Comput. Syst. **25**(5), 528–540 (2009)
11. Deelman, E., Singh, G., Su, M.H., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman, G.B., Good, J., Laity, A., Jacob, J.C., Katz, D.S.: Pegasus: a framework for mapping complex scientific workflows onto distributed systems. Sci. Program. **13**(3), 219–237 (2005)
12. Dobrzelecki, B., Krause, A., Hume, A., Grant, A., Antonioletti, M., Alemu, T., Atkinson, M.P., Jackson, M., Theocharopoulos, E.: Integrating distributed data sources with OGSA-DAI DQP and views. Philos. Trans. R. Soc. Lond. A **368**(1926), 4133–4145 (2010)
13. ebXML Business Process Technical Committee: ebXML business process specification schema technical specification (version 2.0.4). Tech. rep., OASIS (2006)
14. Gedik, B., Andrade, H., Wu, K.L., Yu, P.S., Doo, M.: SPADE: the system S declarative stream processing engine. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, SIGMOD '08, pp. 1123–1134. ACM, New York (2008)
15. Gorton, I., Greenfield, P., Szalay, A., Williams, R.: Data-intensive computing in the 21st century. Computer **41**(4), 30–32 (2008)
16. Gray, J.: Jim Gray on eScience: a transformed scientific method. In: Hey, T., Tansley, S., Tolle, K. (eds.) The Fourth Paradigm: Data-Intensive Scientific Discovery, pp. xix–xxxiii. Microsoft Research, Washington (2009)
17. Gray, J., Liu, D.T., Nieto-Santisteban, M., Szalay, A., DeWitt, D.J., Heber, G.: Scientific data management in the coming decade. SIGMOD Rec. **34**, 34–41 (2005)
18. Habala, O., Jarka, M., Laclavik, M., Simo, B., Tran, V.: Report on pilot applications deployment and platform evaluation. Tech. rep., The ADMIRE Project (2011)
19. Han, L., Liew, C.S., van Hemert, J.I., Atkinson, M.P.: A generic parallel processing model for facilitating data mining and integration. Parallel Comput. **37**(3), 157–171 (2011)
20. Hey, T., Tansley, S., Tolle, K. (eds.): The Fourth Paradigm: Data-Intensive Scientific Discovery. Microsoft Research, Washington (2009)
21. Hull, D., Wolstencroft, K., Stevens, R., Goble, C.A., Pocock, M.R., Li, P., Oinn, T.: Taverna: a tool for building and running workflows of services. Nucleic Acids Res. **34**(web–server-issue), 729–732 (2006)
22. Isard, M., Budiu, M., Yu, Y., Birrell, A., Fetterly, D.: Dryad: distributed data-parallel programs from sequential building blocks. In: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems, EuroSys '07, pp. 59–72. ACM, New York (2007)
23. Jordon, D., Evdemon, J.: Web services business process execution language, version 2.0, OASIS standard. Tech. rep., OASIS (2007)

24. Language and Architecture Team, ADMIRE project: DISPEL: data-intensive systems process engineering language users' manual (version 1.0). Tech. rep., School of Informatics, University of Edinburgh (2011)
25. Lee, E.A., Neuendorffer, S.: MoML—a Modeling Markup Language in XML–version 0.4. Tech. rep., University of California at Berkeley (2000)
26. Liew, C.S., Atkinson, M.P., Ostrowski, R., Cole, M., van Hemert, J.I., Han, L.: Performance database: capturing data for optimizing distributed streaming workflows. Philos. Trans. R. Soc. Lond. A **369**(1949), 3268–3284 (2011)
27. Llorá, X., Ács, B., Auvil, L.S., Capitanu, B., Welge, M.E., Goldberg, D.E.: Meandre: semantic-driven data-intensive flows in the clouds. In: IEEE Fourth International Conference on eScience, pp. 238–245. IEEE Press, New York (2008)
28. Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., Lee, E.A., Tao, J., Zhao, Y.: Scientific workflow management and the Kepler system. Concurr. Comput. **18**(10), 1039–1065 (2006)
29. Muthukrishnan, S.: Data streams: algorithms and applications. Found. Trends Theor. Comput. Sci. **1**(2), 117–236 (2005)
30. Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M.R., Wipat, A., Li, P.: Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics **20**(17), 3045–3054 (2004)
31. Olston, C., Reed, B., Srivastava, U., Kumar, R., Tomkins, A.: Pig Latin: a not-so-foreign language for data processing. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, SIGMOD '08, pp. 1099–1110. ACM, New York (2008)
32. Pallickara, S., Ekanayake, J., Fox, G.: Granules: a lightweight, streaming runtime for cloud computing with support for map-reduce. In: Proceedings of the IEEE International Conference on Cluster Computing and Workshops, CLUSTER '09, pp. 1–10 (2009)
33. Pike, R., Dorward, S., Griesemer, R., Quinlan, S.: Interpreting the data: parallel analysis with Sawzall. Sci. Program. **13**(4), 227–298 (2005)
34. Stonebraker, M., Becla, J., Dewitt, D., Lim, K.T., Maier, D., Ratzesberger, O., Zdonik, S.: Requirements for science data bases and SciDB. In: Conference on Innovative Data Systems Research (CIDR) (2009)
35. Taylor, I., Shields, M., Wang, I., Harrison, A.: The Triana workflow environment: architecture and applications. In: Taylor, I., Deelman, E., Gannon, D., Shields, M. (eds.) Workflows for e-Science, pp. 320–339. Springer, London (2007)
36. Thies, W., Karczmarek, M., Amarasinghe, S.: StreamIt: a language for streaming applications. In: Horspool, R. (ed.) Compiler Construction. Lecture Notes in Computer Science, vol. 2304, pp. 49–84. Springer, Berlin (2002)
37. Wilde, M., Hategan, M., Wozniak, J.M., Clifford, B., Katz, D.S., Foster, I.: Swift: a language for distributed parallel scripting. Parallel Comput. **37**(9), 633–652 (2011)
38. Yaikhom, G., Atkinson, M.P., van Hemert, J.I., Corcho, O., Krause, A.: Validation and mismatch repair of workflows through typed data streams. Philos. Trans. R. Soc. Lond. A **369**(1949), 3285–3299 (2011)
39. Yu, J., Buyya, R.: A taxonomy of workflow management systems for grid computing. J. Grid Comput. **3**, 171–200 (2005)
40. Zhao, Y., Hategan, M., Clifford, B., Foster, I., von Laszewski, G., Nefedova, V., Raicu, I., Stef-Praun, T., Wilde, M.: Swift: fast, reliable, loosely coupled parallel computation. In: Proceedings of the 2007 IEEE Congress on Services, SERVICES '07, pp. 199–206. IEEE Comput. Soc., Los Alamitos (2007)