## Project Title
ADMIRE

## Document Title
Internal report on goals, scope and draft definition of AA3

## Deliverable Number
D2.4

## Authorship
Ally Hume, Liangxiu Han, Jano van Hemert and Malcolm Atkinson

## Document Filename
ADMIRE-D2.4-architecture

## Document Version
3.0

## Distribution Classification
Private

## Distribution List
European Commission

## Approval List
Ally Hume, Alex Wöhrer, Project Manager, Executive Board

---

**Copyright © 2008–2009 The University of Edinburgh and others within the ADMIRE Project**
<table>
<thead>
<tr>
<th>Personnel</th>
<th>Date</th>
<th>Comment</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ally Hume</td>
<td>28/05/2008</td>
<td>First draft</td>
<td>0.1</td>
</tr>
<tr>
<td>Liangxiu Han</td>
<td>03/06/2008</td>
<td>Second draft</td>
<td>0.2</td>
</tr>
<tr>
<td>Jano van Hemert and</td>
<td>05/06/2008</td>
<td>Third draft</td>
<td>0.3</td>
</tr>
<tr>
<td>Liangxiu Han, Jano van Hemert and Malcolm Atkinson</td>
<td>09/06/2008</td>
<td>Fourth draft</td>
<td>0.4</td>
</tr>
<tr>
<td>Liangxiu Han</td>
<td>12/06/2008</td>
<td>New diagram in Fig 2</td>
<td>0.42</td>
</tr>
<tr>
<td>Jano van Hemert</td>
<td>12/06/2008</td>
<td>Changes made while discussions with WP4</td>
<td>0.43</td>
</tr>
<tr>
<td>Jano van Hemert</td>
<td>24/07/2008</td>
<td>Converted 2.1a from Word to 2.1 in LaTeX, restructured and started adding info from other sources</td>
<td>0.43</td>
</tr>
<tr>
<td>Liangxiu Han</td>
<td>02/08/2008</td>
<td>Adding introduction, related work, sequence analysis in section 4.5.5 and creating bibtex file of references in the document</td>
<td>0.52</td>
</tr>
<tr>
<td>Liangxiu Han</td>
<td>12/08/2008</td>
<td>Adding semantic integration part</td>
<td></td>
</tr>
<tr>
<td>Liangxiu Han</td>
<td>14/08/2008</td>
<td>Adding data modelling</td>
<td></td>
</tr>
<tr>
<td>Jano van Hemert</td>
<td>15/08/2008</td>
<td>Final check before internal review</td>
<td>0.6</td>
</tr>
<tr>
<td>Liangxiu Han</td>
<td>26/08/2008</td>
<td>Modifications based on David Snelling’s comments</td>
<td>0.7</td>
</tr>
<tr>
<td>Jano van Hemert</td>
<td>28/08/2008</td>
<td>Final tweaks</td>
<td>0.71</td>
</tr>
<tr>
<td>Rob Baxter</td>
<td>28/08/2008</td>
<td>Approved</td>
<td>1.0</td>
</tr>
<tr>
<td>Liangxiu Han</td>
<td>02/02/2009</td>
<td>Articulating with the white paper and restructuring the document</td>
<td>1.2</td>
</tr>
<tr>
<td>Jano van Hemert</td>
<td>12/02/2009</td>
<td>Completely restructured and aligned with the whitepaper</td>
<td>1.3</td>
</tr>
<tr>
<td>Ally Hume</td>
<td>13/02/2009</td>
<td>Added chapter mapping architecture to prototype</td>
<td>1.4</td>
</tr>
<tr>
<td>Liangxiu Han</td>
<td>18/02/2009</td>
<td>Incorporating comments from Peter</td>
<td>1.41</td>
</tr>
<tr>
<td>Rob Baxter</td>
<td>27/02/2009</td>
<td>Final edit and signoff</td>
<td>2.0</td>
</tr>
<tr>
<td>Jano van Hemert</td>
<td>18/08/2009</td>
<td>Added Chapter 8 Evaluation of the architecture</td>
<td>2.1</td>
</tr>
<tr>
<td>Jano van Hemert</td>
<td>18/08/2009</td>
<td>Added first evaluation paper to evaluation, removed Appendix B, added n-Fold Cross Validation explanation and results to Appendix A</td>
<td>2.2</td>
</tr>
<tr>
<td>Liangxiu Han</td>
<td>02/09/2009</td>
<td>Incorporating comments from Alex</td>
<td>2.3</td>
</tr>
<tr>
<td>Liangxiu Han</td>
<td>04/09/2009</td>
<td>Incorporating comments from Ally</td>
<td>2.4</td>
</tr>
<tr>
<td>Liangxiu Han</td>
<td>09/09/2009</td>
<td>Incorporating comments from Rob</td>
<td>2.5</td>
</tr>
<tr>
<td>Rob Baxter</td>
<td>17/09/2009</td>
<td>Final edit and signoff</td>
<td>3.0</td>
</tr>
</tbody>
</table>
# Contents

Executive Summary 3

1 Introduction 7

2 ADMIRE Architecture 11
   2.1 Goals of the architecture ............................................. 11
   2.2 Architecture design decision ........................................... 11
      2.2.1 DMI processes .................................................. 11
      2.2.2 Partitioning conceptual concerns ................................. 14
      2.2.3 Partitioning user communities .................................... 14
      2.2.4 Partitioning computational concerns ............................... 15
   2.3 Summary of ADMIRE architecture ...................................... 17
   2.4 Architectural principles ............................................... 20
   2.5 Prioritising architecture principles .................................. 23

3 DMI Gateway 25
   3.1 Components of the gateway .......................................... 26
      3.1.1 DMI Validator .................................................. 27
      3.1.2 DMI Planner .................................................... 27
      3.1.3 DMI Optimiser .................................................. 28
      3.1.4 DMI Executor ................................................... 28
      3.1.5 DMI Monitor ..................................................... 28
      3.1.6 Gateway Registry ................................................ 28
   3.2 Receiving DMI-language sentences .................................... 29

4 DMI Language 31
   4.1 Primary features .................................................... 31
      4.1.1 Type Systems ................................................... 32
   4.2 Secondary features ................................................... 37
   4.3 DMI Tools ............................................................ 39

5 DMI Registry 41

6 DMI Enactor 43
   6.1 DMI Enactor Functionalities Related to Data Processing .......... 43
      6.1.1 Data Access .................................................... 43
      6.1.2 Data Storage ................................................... 45
10 Conclusion and Future Work

10.1 Future Work: the Next Six Months

10.2 Future Work: Lifecycle of the ADMIRE Project

10.2.1 Security framework

10.2.2 Outstanding issues

A DMI Usage Patterns

A.1 n-Fold Cross Validation
Executive Summary

This document describes a new iteration of the ADMIRE Architecture, covering the progress of WP2 during last six months (PM12-PM18). The main objective of the WP2 during this period has been to explore parallel processing models in the ADMIRE architecture through the implementations of prototypes and to perform the first evaluation of the architecture based on a real Data Mining and Data Integration (DMI) use case.

In terms of the explicit research goals set for PM18, progress can be summarised as follows:

A major feasibility study of the ADMIRE architecture has been conducted around the existing EURExpress-II use case. The aims of this exercise were a better understanding of performance and concurrency issues within the proposed architecture and an understanding of how concurrency and parallelism within a DMI workflow can be identified, implemented and expressed in the DMI language under development in workpackage 1 (cf. Section 4 of ADMIRE deliverable D1.5 [61]).

Capture Processing Elements in Registry. The first set of Processing Elements from the EURExpress, Flood and ACRM use cases has been defined and captured in the prototype ADMIRE Registry (cf. Section 5 of [61]).

About this document

This document is a revision of “ADMIRE D2.1-Public—ADMIRE Architecture”. The major changes of this iteration are:

- a new chapter, Chapter 8 “Evaluation of the architecture”, which starts on page 60; it outlines our approach to evaluation in conjunction with the use cases and presents the results of the first evaluation of the architecture (also written up as a paper and submitted to the IEEE International Conference on e-Science);

- in Chapter 10 future work has been divided into short-term goals (i.e. six to twelve months) and long-term goals (i.e. the lifecycle of the ADMIRE project);

- Appendix B was removed. An explanation of n-Fold Cross Validation and corresponding results were added to Appendix A on page 91.
Chapter 1

Introduction

There is a rapidly growing wealth of data. The number of sources of data is increasing, while, at the same time, the diversity, complexity and scale of these data resources are also increasing dramatically. This cornucopia of data offers much potential; a combinatorial explosion of opportunities for knowledge discovery, improved decisions and better policies. Today, most of these opportunities are not realised because composing data from multiple sources and extracting information is too difficult. The demanding challenges that face today’s citizens, such as coping with climate change, improving international security, predicting the behaviour of global economic systems or delivering better wellbeing for an ageing population, are a pressing motivation for improving our exploitation of data. Every business, organisation and government faces problems that can only be addressed successfully if we improve our techniques for exploiting the data we gather.

This data-rich environment with a growing commitment to the effective exploitation of data leads to ADMIRE’s vision that future DMI architectures must simultaneously address a number of sources of scale and complexity. The following list is indicative of that multi-dimensional challenge and ADMIRE’s strategic response:

- The scale and complexity of each data source grows. ADMIRE addresses this with data-flow technology to reduce data handling and to move data reduction and transformation operations closer to data sources.

- The number and variety of data sources is growing. ADMIRE addresses this by proposing dynamic composition processes as warehousing and static global schema are infeasible.

- The computational complexity of extracting information grows as a result of the above and of increasingly sophisticated requirements. ADMIRE addresses this by enabling the work of data-aware distributed computing engineers.

- The number of application domains using DMI grows, becomes more diverse and engages more users. ADMIRE addresses this by recognising communities of users, by supporting them with their own environments and by delivering packaged production versions of DMI processes.

- The number of experts involved in developing new DMI processes and supporting application domains grows. ADMIRE addresses this by separating support for DMI experts
from that for Data-Aware Distributed Computing (DADC) engineers and application-domain users.

- The number of providers of data and DMI services grows. ADMIRE separates the organisation of environments for DMI process development from the complexities of DMI service provision by interposing DMI gateways using a canonical language.

- The growing sophistication of information extraction from large bodies of data requires ever more complex and refined workflows. ADMIRE addresses this by structuring the predefined components into libraries that correspond to a conceptual structure captured in ADMIRE ontologies and supports the incremental refinement of libraries and the DMI processes that use them. It allows greater contemporaneous effort by supporting concurrent independent development by three separate categories of experts working both for providers and users.

- The providers of data and services autonomously change their offered services and schema at a rate which defies manual adaptation when many resources are in use. ADMIRE proposes that this should be addressed by exploiting type systems, semantic description, community effort and light-weight composition to semi-automatically adapt to change and to pool the intelligence of human interventions.

The ADMIRE project aims to significantly improve the exploitation of data by delivering three categories of output: a framework, an architecture and a set of use cases that illustrate how they can be used to improve DMI. These will be built on a consistent set of principles that will emerge and be validated by the project’s research; for example:

- The partitioning of concerns based on a stratification of interests with a variety of application domains on top, an intermediate layer containing the various experts in data integration and data mining and a foundation of distributed systems engineers that build, operate and optimise the computational services.

- The eventual scale and diversity of DMI activity will mean that globally consistent services will be undesirable and unachievable due to concurrent autonomous change.

- The scale and diversity of activity will benefit from increasing the independence between DMI-process developers and DMI-service developers.

- Communities of interest within application domains will need to be supported, e.g. their domain-specific prevalent standards will need to be honoured and their independent vocabularies, ontologies and developed DMI-processes supported.

- Support for controlled sharing of information is essential.

Within ADMIRE these high-level principles are explored in more detail as they are applied in the specific explorations of the project. Within the scope of the project it is not possible to explore their full generality. They are discussed as they arise within the sub-topics covered below.

The ADMIRE framework will include in prototypical form:

- a coherent and consistent model in which data exploration, data integration and data mining are integrated and well supported;
• mechanisms for extracting information from such integrated data by accommodating a wide variety of data-analysis methods and (legacy) services;

• a robust and efficient underpinning distributed-computing framework that accelerates the integration and interpretation of data from multiple autonomous data resources;

• tools that support data mining and knowledge engineering experts as they develop strategies, algorithms and workflow patterns to extract information or test hypotheses against integrated data;

• an expansion of the range of domain experts who successfully exploit the data, achieved by delivering easily understood tools, so that domain experts may use the strategies and workflow patterns developed by DMI experts to discover new evidence, inform policies and analyse previous behaviour; and

• a distributed computing environment that expects and handles the wide variety of changes that are encountered as data resources are re-organised and evolve their services in response to advances in technology or in their business and research activities.

The ADMIRE architecture will provide a specification and definition of the structure and behaviour of the ADMIRE system, validated through prototypes and preliminary evaluations, for those data-aware distributed-systems engineers who will build production versions of the tools, services and computing environments to deliver DMI for widespread professional use. The architecture is intended to guide the construction and operation of the full computing environment needed for DMI by supporting all aspects of the DMI framework. The ADMIRE architecture will include:

• a separation, via a canonical form for DMI process definition, between a diverse and extensible domain of DMI tools and a range of DMI-enactment platforms;

• a model for describing all the components participating in DMI processes that supports the range of DMI tools, DMI-enactment optimisation and automated adaptation to changes in data sources and services;

• DMI gateways that mediate requests in the canonical form and hide the transformations, delegation and heterogeneity of the distributed underlying data resources, DMI processing elements and services; and

• efficient direct data paths for delivering results and monitoring DMI-process enactment.

This document mains purpose is to convey the design of the architecture to support data integration and mining in a service-oriented context in terms of requirements, functionalities and implementation. While this document is still under construction, it has a secondary purpose, which is to define the boundaries between the work on the architecture and the work on the language design (WP1), the execution platform (WP2), and the tools that make use of the architecture (WP5). Please note this document is work in progress; it will contain inconsistencies and is incomplete. It also contains pointers to keep track of open issues and questions that we hope to resolve later.

The remainder of this document is organised as follows: Chapter 2 describes the design of ADMIRE architecture including goals, principles and a high-level overview of the architecture; Chapter 3 introduces the central idea of gateways and lists the major components a
gateway should have; these gateways will accept requests for data mining and integration in a
standardised language briefly discussed in Chapter 3 and the detail work on language aspect
can be found in WP 1 deliverable D1.2 [59]; registries are needed to allow semantically correct
communication between gateways, which are described in Chapter 5. Chapter 6 introduces
processing components that are assumed to be available in the execution engine. To show how
these components interoperate, Chapter 7 contains descriptions and examples of the current
prototypes of the ADMIRE platform. Chapter 8 describes evaluation principle and a specific
prototyped use case. Chapter 9 highlights related work on existing data mining architectures
and data integration frameworks; and current issues and future work are kept in Chapter 10.
Chapter 2

ADMIRE Architecture

2.1 Goals of the architecture

The ADMIRE architecture has to contain all of the components and deliver all of the functions introduced in ADMIRE white paper [44, Section 2].

The ADMIRE architecture has to be scalable, resilient and deliver good performance for the envisaged range of distributed data mining and integration (DMI) scenarios and have sufficient simplicity and regularity that its development and maintenance is feasible. It intends to provide guidelines for constructing and operating the full computing environment needed for DMI. These guidelines will be validated through prototypes and preliminary evaluations.

A crucial contribution of ADMIRE is to take an integrated view of all of the stages in the DMI process. This is intended to have two benefits:

- the elimination of boundaries that have to be negotiated by domain and DMI experts, removing technical clutter so that they can focus on the challenges specific to their domain or category of DMI operations; and

- the extension of the scope for architects and engineers to impose structure, recognise and support patterns and deliver optimisations.

2.2 Architecture design decision

2.2.1 DMI processes

The focus of the ADMIRE architecture is the computational treatment of DMI processes. The requirements of the DMI process are described as follows.

- The full life cycle of a DMI process must be supported:
  - its creation using design and specification tools or languages;
  - its revision using tools or editors;
  - its storage and transmission to other contexts where it may be revised and re-used;
  - its packaging for routine and repeated use, possibly with parameters that must be substituted before use, in which case it is a DMI-process template;
  - its submission to DMI gateways; and
• A DMI process must be capable of describing any step that DMI practitioners want
describe and enact. This must include:
  – the exploration of individual or integrated data resources;
  – the extraction of data from individual or combinations of data resources;
  – the cleaning of data derived from the previous steps;
  – the processing of data derived by any of the preceding steps with any composition
    of data analysis processes; and
  – the visualisation and presentation of results from any of the above stages.

It must also include any valid concatenation of the above stages of a complete DMI
process that extracts information from data and management operations to package
and deploy DMI-process descriptions and to monitor, steer and terminate running en-
actments.

• The computational steps in a DMI process are accomplished by processing elements
(PE). Each PE corresponds to code that the enactment may activate via some mech-
anism, such as a web-service invocation or an object’s method invocation. A single
enactment may contain multiple instances of a particular PE. Each PE has the follow-
ing properties:
  – a PE-class name that is indicative of the function of the PE;
  – a PE-variant description that is indicative of a particular way of implementing the
    function and variations in its behaviour or results\footnote{In time, this may be a sophisticated structure support users as they select using tools and enactment planning algorithms as they select using rules and heuristics.} this will normally contain a
    UUID to ensure unambiguous interpretation;
  – a PE-instance identity that allows operations specific to that instance to be per-
    formed, this will contain a UUID to ensure unique identification in logs, inter-
    gateway communication and so on;
  – a set (possibly empty) of named inputs, each of which has a specified type\footnote{An initial discussion of these types appears in Section 4.1.1} that
    may restrict the permitted input values;
  – a set (possibly empty) of named outputs, each of which has a specified type that
    may be explicitly related to one or more of the input types;
  – a specified auto-iteration behaviour;
  – a specified termination behaviour; and
  – additional rules relating inputs to outputs, e.g. relating cardinality, ordering and
    identity behaviour.

Several forms of PE are recognised:
  – primitive PE, which have been pre-coded and made available in the computational
    environment;
– composite PE, that are made by composing PE using the standard ADMIRE composition mechanisms (see below);

– data elements (DE), such as data resources, data bases, file stores, etc. that have persistent state, that is the results provided by their outputs depend on the history of operations performed on them, both from within and external to the ADMIRE systems, these may be primitive or composite; and

– standard PE, that are available by default in every ADMIRE environment (they may be examples of any one of primitive, composite or data elements).

• The PE are composed by connecting the outputs of each PE to the inputs of other PE using data streams. Data streams communicate a sequence of values that are delivered in the order that they are provided. The stream has a flow-control function as a receiver can wait on data becoming available and can signal to the sender that it no longer requires values. Similarly, the sender can signal that there are no more values. A stream may operate with arbitrary granularity, may take advantage of pass by reference and may offer an arbitrary amount of buffering.

• Inputs may also be provided as literal values.

• Outputs may also be sent directly to requester-specified destinations or to a data sink.

• Recurring parametric and abstract DMI patterns may be recognised and encoded, so that they may be stored and passed as requests in a pattern form, that can be realised differently depending on the data and computational context. A pattern may be parameterised with the PE that occur within it, with numerical values, with data elements and with input and output data streams. A pattern with all of its parameters specified behaves logically as a composite PE but remains as a pattern during DMI-process storage, transmission and submission. The commonly used patterns will be predefined and available in ADMIRE environments. New ones may be defined and added dynamically.

• A DMI-process may be submitted to a DMI gateway as a request for its enactment.

• A DMI-process description has to be complete and self-consistent before it is ready for submission. It will then be a directed graph of PE nodes interconnected by directed arcs that denote data streams from a source to one or more destinations. The completeness requirement means that the source of data for every input of every PE must be specified and at least one destination must be specified for every output of every PE. There is a destination denoted by discard that will throw away unwanted output of any type and a source named empty that will provide an empty stream of any type. Subsets of DMI-processes (i.e. subgraph) may be enacted concurrently with other fully connected subgraph that is in the same request provided there are no data streams between them. Enactment systems may also discover other opportunities for parallel enactment. When a gateway delegates work to another gateway it may send a number of subtasks, each fully connected apart from data streams connected to subtasks enacted as part of the same requests in other gateways. This requires inter-gateway data streams with suitable capacity and efficiency. To set these up names for the source and destination must be generated that are suitably unique. These names will contain a UUID [103] to avoid erroneous connections. Cycles may appear in a request only if they include one of the
special PE that can control the cycle, e.g. in master-worker mode, to avoid indefinite enactment.

2.2.2 Partitioning conceptual concerns

As a first stage of imposing structure, the ADMIRE architecture recognises two dimensions and chooses to separate concerns into three categories in each dimension. ADMIRE hypothesises that these partitions are appropriately separable, i.e. that they will allow different communities to work reasonably independently with well defined interfaces for their common concerns. The first dimension, called the **expert dimension**, corresponds to three communities of experts, who have to be mutually supportive. The second dimension, the **abstraction dimension**, corresponds to three levels of abstraction and detail—different universes of discourse with carefully controlled overlaps for digital communication. Figure 2.1 summarises the conceptual domains which are the primary concerns of each category of expert.

![Figure 2.1: Conceptual concerns of each ADMIRE community](image)

2.2.3 Partitioning user communities

The first dimension spans three communities of experts:

- **Domain experts**. The focus of domain experts is the particular domain in which they work, such as: “increasing sales in a particular retail sector”, “recognising the phenotypical implications of correlations in gene expression patterns” or “predicting geographic localities which will overtax current emergency plans in times of flooding”. They are presented with DMI tools tailored for their domain, integrating the set of data resources, domain ontologies, methods, interchange standards, (legacy) services and presentation/visualisation systems appropriate for their domain. Technical and operational effects are well hidden. A domain expert will become familiar with the DMI capabilities presented and will select prepackaged integrated DMI tasks, choose source data, set control parameters and specify destination and presentation forms for outputs. The ADMIRE system will take responsibility for deploying, resourcing, executing and monitoring the requested enactment and will report progress and other information in
terms appropriate for the domain expert. A domain expert may observe intermediate results presented in an appropriate form and steer the enactment of DMI processes.

- **DMI experts.** The DMI experts will understand algorithms and methods established in DMI and may well specialise in supporting particular application domains or particular classes of method. They will be presented with a tool set and workbench, which they will use both to develop new methods that can be installed in the ADMIRE context and to re-factor, compose and tune existing components that they then test and package for use by domain experts. They will be reasonably aware of the capabilities of the underlying implementations, i.e. the ADMIRE framework, and will expect detailed diagnostics to support DMI-process development and tuning. They are aware of the structures, representations and type systems of the data they manipulate and may also exploit that data’s statistical properties. Their tools will exploit DMI-specific semantic descriptions developed in ADMIRE to validate or automate their tasks. They may observe DMI-process enactment and analyse traces of previous enactments to evaluate performance and to improve process definitions.

- **Data-Aware Distributed Computing (DADC) engineers.** These engineers are concerned with the engineering that implements the ADMIRE platform that supports DMI enactment, resource management, DMI-system administration, and delivers interfaces for the tools required by the other two groups. The term *Data-aware Distributed Computing* is used because the technology will need to adapt schedules, deployments and process mappings as a consequence of variations in the data, such as volume, format, representation, selectivity and content [34]. The DADC engineers’ role will be pioneered for DMI in the context of ADMIRE. It will include the dynamic deployment, configuration and optimisation of the data movement, storage and processing. As far as possible such adaptivity will be automated through autonomic strategies. The DADC engineers will also organise the library of data handling and data processing activities, providing them with an appropriate computational context, giving them regular structures and determining how they should be described. Similarly, the descriptions required for registries of data resources and mechanisms for composing methods will be organised by these DADC engineers but populated by DMI experts. The DADC engineers also need to understand the distributed-data system’s failure modes and develop recovery strategies to resume long-running DMI activities as well as to gracefully recover resources after local failures. The framework that detects data-resource evolution and limits its impact will also be the responsibility of DADC engineers. DMI or domain experts may be called upon to handle detected changes that automated adapters cannot accommodate. The data-aware technologies developed will be useful in other application areas beyond those explored in ADMIRE.

### 2.2.4 Partitioning computational concerns

The three levels of the abstraction dimension represent different aspects of a DMI process as it moves from creation or refinement, via a canonical intermediate model to a detailed representation that has sufficient information for enactment.

- **Tool.** At this level data sources and their components are nameable elements of the discourse as are the “white-box” representations of DMI-process elements, patterns,
templates and services—“white-box” in the sense that users will have some knowledge of their parameters and processing stages. There is information to support sharing, presentations suited to individuals or communities, and representations of work in progress. A diversity of form and content is expected at this level. The exposure of information via the tools should enable domain experts to steer analyses, e.g. terminate processes that are not producing useful results or adjust parameters to improve results and visualisations.

For the domain experts the exposed information is high-level structure of data resources and the pre-packaged DMI-process templates already tailored for their needs. They work by using the tools to study and interconnect these, to specify the data resources to be used, to provide parameters, to request enactments, to observe enactment progress, to view results and to control, e.g. terminate, their running enactments.

For the DMI experts, the tools provide details of representations, formats and semantics of the data resources, DMI services and activities. They can use the tools to specify generic patterns and to design, debug and tune DMI processes, possibly using some of those patterns. They may use the tools to provide annotation to clarify the type, structure, representation and statistical properties of specific data streams and may specify third-party data transfers. In mature versions of the architecture, they will not need to provide such annotations when they can be inferred, nor will they normally specify the physical mapping to the distributed computational infrastructure. They may use all of the tools used by domain experts with whom they are working, and use tools to package processes for domain experts.

• **Canonical.** The canonical domain has textual representations of DMI-process specifications, that can be generated by the tools or by an ADMIRE expert, and which can be sent to an ADMIRE gateway (see below) as a request for an enactment. DMI processes in this representation should be storable for reuse and refinement, and should be a convenient comprehensible notation for DMI experts and DADC engineers—we call this notation the DMI Language (DMIL). It will have textual representation and will be translatable into internal forms. Various user-oriented languages and graphical design notations will be translated into DMIL. DMI-process descriptions in the language are abstract. That is, they normally use abstract names to refer to DMI-process elements, so that DMI gateways and enactment services behind them have many options as to how to implement the process and how to map it to physical facilities. These DMI-process descriptions must be precise. For example, sufficient type information about data inputs and outputs must be given that the enactment system can validate for internal consistency and detect that changes in external data resources have invalidated type expectations. This is the foundation for adapting to change. That precision will be based on a carefully defined abstract machine underpinning the DMI language. Tools and textual presentations will normally hide much of this detail so that the DMI and domain experts are not distracted by it. However, the tools are responsible for providing that detail when they generate a DMI-process specification for enactment or storage.

• **Enactment Level.** This domain deals with all of the engineering necessary to support the full range of DMI enactments required, over all possible data resources and delivering...
to all possible tools, visualisation systems and data destinations. It must also deal with issues of performance, reliability, security and autonomous evolution. It is therefore potentially complex; high-quality DADC engineering should confine the inevitable operational and system complexity to this level, preventing its intrusion into the high-level language, the normal use of tools by DMI experts or the purview of domain experts. It is impossible to explore fully these engineering issues within the duration of ADMIRE, so we will focus on how to structure and organise this engineering, demonstrating that it is feasible to do so by exploring in detail selected critical issues.

### 2.3 Summary of ADMIRE architecture

The consequences of the above architectural decisions are summarised in this section. The three layers of abstraction lead to a communication via a restricted canonical form between a user-oriented tools layer and a system and provider-oriented platform layer, as is illustrated diagrammatically below.

![Figure 2.2: Separating DMI levels of diversity](image)

Figure 2.2 shows how the complexities of matching the diversity of user requirements at the tools level can be separated from the complexity of the enactment level, accommodating the diversity of data resources and services, by interposing the single canonical domain of discourse represented by the DMI language (see Section 4). The ADMIRE hypothesis is that, by enforcing this logical decoupling, both the tools development and the platform engineering will proceed rapidly and independently. Of course, this depends on the quality of the abstract machine and the language operating at the gateway. Developing that quality is one of ADMIRE’s research goals.

Figure 2.3 shows two important specialisations that occur in ADMIRE. The upper cone
denotes the result of DMI experts developing and refining a DMI process. Once it has met a particular target group of domain experts' requirements it will be packaged and presented as a tool for that group, e.g. a portal. Many such packages would coexist, which would correspond diagrammatically with many upper cones all connecting with one lower cone. They would all communicate with platforms through the standard gateway. The lower cone denotes the specialisation corresponding to ADMIRE's DMI platform. There will be one of these per ADMIRE research cycle—only one exists at any one time in an ADMIRE experiment, whereas, in the envisaged future many alternative versions would exist.

These experimental prototype platforms will not reach the scale or complexity of comprehensive production-quality DMI platforms. Instead, they will each comprise prototypes of aspects of fully fledged platforms required by ADMIRE research experiments. In the future we envisage many fully fledged DMI platforms engineered to meet their user communities' workloads, their computational environment and their service providers' business. They will all communicate through DMI gateways using exactly the same canonical DMI language and abstract machine.

However, as is shown in Figure 2.4 the DMI gateway is not the only channel for information flow between the enactment system and the tools. It remains the only channel for submitting requests for DMI enactment and the control of running enactment processes. However, two other significant channels are introduced for two reasons: (1) to obtain sufficient performance, and (2) to exploit established and emerging standards. One channel, direct bulk data delivery, delivers data directly from wherever it is generated in the depths of the distributed hidden DMI-process elements to which the enactment has been mapped, in the form and protocol required by the tool. For example, a 3D representation of gene expression or
Figure 2.4: Communications between DMI tools and DMI enactment systems

the frames of a simulation of a flood development sequence would be sent in the appropriate format to the tool that submitted the enactment request or the visualisation service it specified. The efficiency gains are in the avoidance of multiple data hops. An important additional gain here is that the tools are able to exploit standards prevalent in each domain. The other channel, *dialogue about enactment status*, is a bi-directional path for sending requests from the tool to the platform to obtain status information, or sending notifications to the tool in accord with interests specified by the tool registering interest. There are families of standards for such dialogues which should be used [49, 84, 85].

Figure 2.5: Original model and the new extended version to incorporate data mining and integration, and highlight data is not in one physical location
It is important to recognise that the model of data mining and integration used by ADMIRE is fundamentally different from the existing CRoss Industry Standard Process for Data Mining (CRISP-DM) shown in Figure 2.5(a) which focuses around a fixed and delimited data resource. In ADMIRE data resources are found and combined from a growing and dynamic cloud of data resources that we term the data space. ADMIRE users work with unpredictable selections and extractions from that evolving space as shown in Figure 2.5(b) (source: [62]). Frequently, users of DMI systems will undertake studies similar to previous studies, so that DMI process designs and patterns can be reused and previous tuning and descriptions are well suited to the task. However, as explained above, the space of options is huge, so that data will be used in unpredictable combinations and in unpredictable ways. This means that elements of a DMI system must be described without knowledge of how those descriptions will be used and the enactment and optimisation systems must work acceptably well for all of these unanticipated applications.

A high-level overview of the ADMIRE architecture is shown in Figure 2.6. DMI gateways are connected together over the Internet and Grid. The gateways communicate with one another using standard internet communications technologies such as WSRF-compliant SOAP messages. Each gateway provides a core set of DMI services for accessing to data sources and custom services, and facilitating communication with third-party tools and middleware, which can be driven using a high-level language that provides the canonical notation for communicating about the DMI processes. The domain experts use the high-level structure of data resources and pre-packaged DMI-process templates already tailored for their needs (e.g., presented within a portal). They work by using the tools to study and interconnect these, to specify the data resources to be used, to provide parameters, to request enactments, to observe enactment progress, to view results and to control, e.g. terminate, their running enactments. DMI experts work with tools and workbench to develop new methods that can be installed in the ADMIRE context and to re-factor, compose and tune existing components that they then test ad package for use by domain experts. The DADC engineer are mainly concerned with the engineering that implements the ADMIRE platform that supports DMI enactment, resource management, DMI-system administration, and delivers interfaces for the tools required by domain and DMI experts. The DADC engineer’s role will be pioneered for DMI in the context of ADMIRE. The ADMIRE envisages a future where many communities are developing processes and using DMI services. To share definitions and selections from the standard definitions of DMI-processing components and support continuous DMI-process research and development and persistent provision of services, the architecture therefore uses registry for each community and a common repository to store the established and shareable DMI workflow designs for re-use by members of the community and others to whom they grant access.

2.4 Architectural principles

The ADMIRE architecture develops from service-oriented architectures and imposes further discipline based on the following principles, many of which are familiar from service-oriented architectures [76] and the fundamental principles long established in distributed computing [110], however they need interpretation and application in the context of DMI.
• **DMI Gateway.** A DMI gateway is a software component that represents an arbitrarily complex set of resources behind its offered services. In ADMIRE these services are all concerned with supporting DMI processes. A gateway hides the internal complexity so that it may be re-engineered and polices the acceptance of requests in conformance with resource limitations, load levels, security enforcement and other factors.

• **Delegation hiding.** Once a request for enactment has been submitted via a DMI gateway it may be delegated in part or in whole to other DMI gateways, internal enactment “subcontractors” or legacy services, without the requester needing to take action nor being aware of the delegation. Delegation should not impose unnecessary data transfers or control messages. Monitoring and status data must be mediated via aggregators, so that tools can obtain progress, diagnostic and provenance information without becoming aware of the details of delegation and enactment mapping.

• **Recursive composition.** A set of composed activities may be treated as a single activity in another composition. A DMI gateway may integrate services from a set of DMI gateways.

• **Data-flow composition.** The DMI activities are composed by connecting data sources with data consumers. Data sources may be a specified data extraction from a data resource, a third-party transfer or an output from a DMI activity. Data consumers include an update operation on a data resource, a third party transfer or an input to a DMI activity. Normally, the order of enactment and the opportunities for parallelism are inferred from the dependencies in these data flows.
• **Typed composition.** The DMI system will use a multi-layer type system to validate the composition of components in all DMI-process definitions, particularly their use of recursive and data-flow compositions. In later versions, the type system may support selective presentation of options in DMI-process design tools and automatic insertion of adaptations to handle representational changes.

• **Abstracted data flow.** Logically, a data flow is a pipe carrying a sequence of elements, a data stream from its source to its consumer, with buffering to cope with different production and consumption rates. This has the same logic whatever the scale and type of the data transfer. The implementation may use any granularity of transfer without change of semantics. Similarly, data may be passed by reference or by copy without affecting the semantics.

• **Data substitutability.** Any data flow, i.e. the sequence of values passing along a data stream, may be captured and saved in a data store for diagnosis or re-use. The graph of interconnected activities and data resources producing a data flow may be treated as a dynamically evaluated view of the specified extractions from those data resources, i.e. the normal enactment model re-evaluates that graph of activities. The stored data is a snapshot of that view. It is always possible to replace a view with a stored snapshot and *vice versa*. The only change in semantics should be the difference in the times when the originating data resources are accessed.

• **Data-scale invariance.** The composition mechanisms and the enactment mechanisms must be capable of implementation over a wide range of data scales (from tens of bytes to tens of terabytes) without a change of semantics. Short-running enactments for data exploration must yield the same semantics as production runs over large volumes of data that take days to process. Similarly it must be possible to test and validate methods or compositions of methods on small samples of data and then use them in production mode.

• **Sampling support.** Mechanisms should be available for effectively and efficiently sampling data, so that a specified proportion of the available data is used (random, initial, terminal and periodic sampling should be supported) with unchanged semantics, except that which results from the different data contents. Sampling is important to encourage data exploration and DMI process development, for example, Alex Szalay when reporting on developing astronomical data queries at the Microsoft e-Science conference, December 2008, indicated that algorithms and systems should be explored with $1/10000$ of the data and then with $1/100$ before investing in the full analysis cost, based on experience reported in [50].

• **Provenance support.** The DMI system must support mechanisms for collecting provenance information so that, when a user specifies that some data is saved to a data resource, sent to a third party, or delivered to a visualisation mechanism, it is possible to obtain the provenance information and send that to a user-chosen destination.

• **DADC patterns.** Certain frequently used patterns will be recognised so that they can be exploited by DMI experts, and can be supported by specialised efficient implementation strategies. Examples of such patterns include:
– pipelines;
– distributed map reduce;
– distributed tree-integration of replicated or partitioned data collections;
– parallel all-meets-all computation; and
– multi-site visit and collect.

- **Semantic and type description** DMI sources and activities should be well described in terms of their semantics and types. This will allow validation of the correctness of compositions of DMI activities and use of DMI resources, as well as facilitating ways for increasing the efficiency of compositions by transforming them to equivalent compositions. The description data will be represented in standard languages like RDF(S) and OWL, and stored in RDF stores that will be accessed using standard mechanisms, according to ontologies defined at different levels of abstraction.

### 2.5 Prioritising architecture principles

The ADMIRE project will need to keep all of these principles in mind to ensure that its architectural designs and implementation experiments will still be valid when any postponed principles are eventually considered. However, the project will give priority to demonstrating approaches to the following selected priorities:

- DMI-gateways as a means of separating the complexities of the enactment provision from the complexities of tools provision, based on the architecture pioneered by OGSA-DAI and USMT.

- Dynamic mapping and optimisation from requests submitted to a DMI-gateway in a canonical form. In later versions of the ADMIRE experiments this will include delegation hiding and optimisation.

- Support for incrementally defining and refining DMI-processes and the components from which they are constructed using data-flow composition, recursive composition and abstracted data-flow. The mechanisms implementing these principles will be demonstrated in successive versions of the ADMIRE prototypes.

- A small number of commonly encountered DADC patterns will be explored to demonstrate the DADC-engineering processes.

- A type system will demonstrate the key elements of the multi-level description and its use in validating the requests to gateways and the compositions they include, initially using structural equivalence validating the graph-composition rules needed for data-flow composition and later including consistency of the semantic use of the values transported along the data streams. In the final prototype, these two levels of types may also be used to support selective presentation by design tools and semi-automated insertion of simple representational transformations.

- Description in registries, that includes and is coupled with the above type information, that can be exploited by tools and by DMI gateways as described in [43]. Successive versions of ADMIRE prototypes will explore an increase in the information carried by
registries and a consequent increase in the use of these descriptions by tools and by the delegation and optimisation systems.

Given the above goals and the priorities selected for the ADMIRE architecture, at the first stage, the ADMIRE architecture mainly focuses on four important functional requirements and components: DMI Gateway, DMI Enactor, DMI Language, and DMI Registry. In the following chapters, we will detail the functional requirements for each of these.
Chapter 3

DMI Gateway

The DMI gateway is a software component that separates the complexities of the enactment provision from the complexities of tools provision, based on the architecture pioneered by OGSA-DAI and USMT. The DMI gateway service is responsible for receiving DMI queries and ensuring they are executed on the services known to that gateway. There are four requirements for the gateway service as follows:

• The gateway service should provide an interface that accepts a DMI-process requests. This request will be represented in a canonical notation, namely, DMI language.

• The gateway must be able to optimise DMI language. The optimisation stage constructs and optimises a DMI plan that will execute the process expressed in the DMI language. This plan consists of a workflow of invocations of data operations. The optimisation stage makes use of descriptions of data sources and processing elements, and statistics obtained from a variety if sources including the data access services and monitoring the enactment of previous DMI process plan.

• The gateway should be responsible for the execution of the DMI-process plan at the enactment stage. This may include the DMI-process plan, for example to recover from an error or make use of new resources that become available.

• The gateway should provide semantic registry to maintain a catalogue and description of the services and components it has available, which may be used in a DMI-process. The advantages for having semantic registry include:
  – providing information to domain and DMI experts in terms suited to each community;
  – selection of targets when mapping abstract element names in a DMI-process request to references to concrete elements that will be used during enactment.
  – checking the self consistency of a DMI-process request;
  – verifying that the properties of data resources, data services and elements in a DMI-process request haven’t changed so as to cause failures unless the changes are dealt with; and
  – informing the enactment planning and optimising algorithms with enough data to determine which alternative DMI-request graphs are equivalent.
- The gateway need to make sure data movement and delivery services deliver data to third parties. These third parties may be client applications or other services involved in the DMI enactment. Both push and pull models of data movement and delivery should be supported.

Figure 3.1 shows the the main basic components of a gateway service and the relationship between gateways and other basic components of outside the gateway in the ADMIRE architecture. Each gateway can communicate with each to determine processing requests in an independent or coordinate way. For example, when receiving a request, probably one gateway can process a request independently or alternatively, two or more gateways work together to process the request.

Within the gateway, there are five main components: a DMI Validator, a DMI Planner, a DMI Optimiser, a DMI Executor, a DMI Monitor and a DMI registry.

Outside the gateway, the DMI enactors are services that provide the functionality that can be used to implement data mining and integration processes. DMI processes will include workflows that are sent to the DMI enactor via the DMI Executor. The DMI process will therefore consist of workflows that chain together the operations provided by DMI enactors. More abstract versions of these operations will be used in the logical query plans and probably also in the DMI language as well. The detail description of the DMI enactor will be presented in Chapter 6. To submit a DMI-language sentence, and communicate between tools and different enactment engines through DMI gateways, a canonical representation will be provided. Chapter 4 discusses the requirements and features of the DMI Language and the ways in which sentences in this language are produced. Besides the registry of the gateway itself, there are also number of registries to maintain catalogues and descriptions of the services and components from domain and DMI communities. The detailed information regarding the purpose of a DMI registry and the relationship between the registry and other system components will be discussed in Chapter 5.

### 3.1 Components of the gateway

Internally a DMI gateway will contain a validator, a planner, an optimiser, an executor and a monitor and a registry.

The DMI gateway service will construct a query plan for each query. The construction of this query plan will apply many application (i.e. data mining) specific optimisations. It is hoped to build a framework that will support many plug-in optimisation strategies. For example, if a new set of services or OGSA-DAI activities were added that supported a type of parallel, distributed data mining it would be desirable to be able to plug-in a component that caused this functionality to be included in the query plan.

In order to construct efficient query plans it is likely that the DMI gateway service will need to utilise statistics gathered from previous query plan executions. It is planned that the DMI gateway service will be able to get such statistics from a DMI monitor service.

If a DMI query contains a reference to an OGSA-DAI service that provides a data set then it is possible that the query plan may invoke any data mining and integration activities available at that service. Thus it is possible for the query plan to actively involve services that are unknown to the gateway but are specified in the DMI query. The alternative to this would be to simply use such OGSA-DAI services to obtain rather than process the data.
This has a major drawback when large volumes of data need to be transferred to do relatively simple processing.

It may be that users explicitly wish to run a particular version of a data mining algorithm. For example, users may wish to ensure that C5.0 from Rulequest Research is the data mining algorithm used to produce decision trees. In such situations the query language ought to be able to annotate the specification of a decision tree classifier with an identifier of the particular algorithm to use. It may be that the DMI gateway will need to publish information about the implementations of the various data mining algorithms it supports.

3.1.1 DMI Validator

The DMI Validator is responsible for checking the validity of the DMI query. This will include type and semantic checking that the operations can be connected as specified in the query.

3.1.2 DMI Planner

The task of the DMI Planner is to compile the DMI request into a DMI logical query plan that is a plan of operations that correctly implement the DMI request. The logical query
plan will not be optimised in any with. Optimisation of the query plan occurs in the DMI Optimiser.

3.1.3 DMI Optimiser

The DMI Optimiser must optimise the logical query plan for efficient execution. The optimiser will first apply logical optimisations and then apply physical optimisation. Logical optimisations are optimisation that can be made to the query plan simply due to the known properties of the operators in the plan. For example, if is usually considered sensible to execute any data filters as early as possible to reduce the amount of data that is processed at subsequent stages. One logical optimisation strategy is therefore to attempt to move filter operations as lose to the data source as possible. Whether and how one operation can be moved ahead of another operation without changing the logic of the query plan is the type of information that a logical optimiser requires.

After logical optimisation the DMI Optimiser will apply physical optimisation. At this stage the physical properties of the data sets and the available enactors will be used to optimise the query plan to minimize the overall execution time. It is the task of the physical data optimiser to choose appropriate strategies for data merging, data movement, parallel execution, data mining etc. The physical optimiser will use continuously updated statistics obtained from monitoring the execution of previous plans to choose optimal execution strategies.

3.1.4 DMI Executor

The DMI Executor is responsible for executing the plan and physical tracking that execution. If the execution fails the DMI Executor should be able to detect this and possibly instigate a means of rectifying the problem. This may involve reinvoking the DMI Optimiser to produce another plan that bypasses the cause of the error.

The DMI query plan executor may be a service in its own right or simply a component within the DMI gateway service. It is not yet clear what advantages would be gained by this being a service.

3.1.5 DMI Monitor

The task of the DMI Monitor is to track the execution of physical query plans on the DMI enactors. By tracking the execution of the query plans the monitor is able to provide information that can be used by the DMI Optimiser to influence future optimisation decisions.

The DMI Monitor will also pass information onto the gateway to be consumed by client. For example clients will be able to monitor the current status of their queries. There will need to be a mapping between the level of detail available at the DMI Monitor and the more abstract view that is presented to clients.

3.1.6 Gateway Registry

Each gateway has its own registry to maintain the descriptions of the services and components it has available. This may include information about DMI services that it is able to provide by delegating work to other gateway. The detail information about the use of a registry is described in Chapter 5.
3.2 Receiving DMI-language sentences

On receipt of a DMI-language sentence denoting a request to enact a DMI process, a gateway typically executes the following steps (these are also presented from the registry point of view in [43]):

1. It decides whether the request is one which this enactor is prepared to run. This may examine credentials, authority and entitlement to resources. It may compare available services and data resources required with those available. A gateway will normally consider as accessible resources that can be obtained by it delegating whole or part of the request. It may consider current commitments and reservations. Then, if the enactment can go ahead, it sends the requester an acceptance acknowledgement and a reference to the session that will correspond to the enactment. If it cannot be accepted, it sends a negative acknowledgement and an explanation.

2. It scans the description and verifies that:
   
   (a) the request is self consistent and a valid DMIL sentence, e.g. that all the inputs and outputs of PE are defined and that all their types match,
   
   (b) that physical elements exist or can be created that can perform as expected by each component, and
   
   (c) that the current descriptions of those elements are compatible with the annotation in the request.

3. It performs high-level pattern expansion (evaluation of DMIL functions) and planning (high-level optimisation), producing DMI processes to be run within the regime of this gateway, and coupled DMI processes to be submitted to other gateways that are federated with this gateway. It carries out the submission of those delegated processes setting up the mechanisms for any required inter-process or inter-gateway data streams.

4. It optimises the local DMI process elements to run locally, mapping each abstractly named element to locally managed corresponding physical elements, taking account of those elements’ data properties, locality, resource requirements and performance. Such mapping may be to any executable form, e.g. to some standard workflow language such as WS-BPEL [96, 51], to OGSA-DAI request documents [39], to DagMan [124], to an abstract code for the DMI-language virtual machine, or to some mixture of these. Note that optimisation can address any cost function, such as response time or energy used.

   In the ADMIRE project the prototypes will map to mechanisms supported by the ADMIRE platform, a combination of USMT, OGSA-DAI and bespoke or standard services—see Chapter 7. The experiments will focus on the effect of optimising a cost function that represents response time, i.e. how much delay there is before a developer or user gets a useful response. On modern architectures we believe the cost of data movement will dominate.

5. It initiates the enactments, and replaces any that are not accepted by the gateway to which they were delegated.

6. It transactionally updates the registry associated with the gateway to include all of the modifications to the set of DMI-component definitions that were specified by the
request, e.g. it adds definitions for all the new PE, function and library values that were defined in this enactment and for which there was a register operation. The transaction property is required as these may be an interdependent set and so other enactments will need all of them, or should see none of them—see [43] for more details.

7. It coordinates, monitors and supervises the enactments, maintaining integrated status information, by collecting data and aggregating it via a reverse of the mappings conducted in steps 3 and 4 that can then be interrogated by the requester. It deals with issues that arise during enactment, such as failures or under performance by dynamically rearranging the enactment.

8. It terminates the enactment, preserves data still to be collected and tidies up using resource lifetime management [121].

Whilst a wide variety of implementations of the DMI-Language are possible, the ADMIRE research will initially map DMIL encoded DMI-process requests to OGSA-DAI requests. The OGSA-DAI repertoire and enactment engines will be progressively expanded to become a powerful DMI-enactment system. These prototypes will only enable exploration of a small subset of the potential DMI capabilities. That subset will be chosen to be sufficient for the ADMIRE experiments (see Section 7). The architecture and hence the design of DMIL, in contrast to the prototypes, is intended to offer guidelines for implementing a wide range of production systems.
Chapter 4

DMI Language

The DMI Language (DMIL) has a central and specific role, which is to provide the canonical notation for communicating about DMI processes. It is the form used to submit requests for DMI process enactment through DMI gateways. It may also be used as a stored representation of DMI processes by tools, and as the input and output of DMI process optimisers.

The DMI Language is a key component of the ADMIRE architecture. It is the notation used to communicate DMI processes and is crucial for tools and for enactment engines. It can be used to communicate between tools and to communicate between different enactment engines. It must be used to communicate through DMI gateways. A great deal, therefore, rests on its design and implementation. The progressive design and evaluation of this language is therefore a central part of the ADMIRE research programme. Whilst a wide variety of implementations of the DMI-Language are possible, the ADMIRE research will initially map DMIL encoded DMI-process requests to OGSA-DAI requests. The OGSA-DAI repertoire and enactment engines will be progressively expanded to become a powerful DMI-enactment system. These prototypes will only enable exploration of a small subset of the potential DMI capabilities. That subset will be chosen to be sufficient for the ADMIRE experiments as shown in [44, Section 6]. The architecture and hence the design of DMIL, in contrast to the prototypes, is intended to offer guidelines for implementing a wide range of production systems. To guide the design of the DMIL, from the ADMIRE architecture point of view, the DMIL should posses the following features:

4.1 Primary features

- The type system of this language, the structural types, is kept separate from the type system identifying the semantics of data-mining or application domain values, the DMI and domain types respectively, so that it can organise DMI for application domains that use different types of data without itself having a complex system.

- The interconnection of elements are described in terms of connections that, conceptually at least, transmit data as a stream.

- The language supports the incremental design and installation of (libraries of) components that support DMI so that computational context provided by a DMI gateway can be incrementally manipulated to better serve a community’s needs during its operation.
4.1.1 Type Systems

There are three type systems that have to be accommodated in DMIL, corresponding to the three conceptual domains of interest:

- **the graph-construction types** that are used to constrain all of the operations used in describing, constructing and manipulating the graphs of processing-element nodes interconnected by streaming connections—this type system is the same whatever DMI-application domain—it uses structural type equivalence and is implemented and exploited by DADC engineers;

- **DMI types** that are used to specify the inputs and outputs of the data-mining specific PE, these are mathematically based and are of primary interest to DMI-experts—these types describe the data streaming along connections to and from PE that perform data mining algorithms—their semantics and equivalence or transformation rules are precisely defined by the DMI-experts, their implementation is achieved by the joint efforts of DMI-experts and DADC-engineers; and

- **the domain types** that describe the data input into and output from processing elements, and transmitted through connections that corresponds with values in an application domain, such as a brain image or a water temperature time series—there may be many versions of this type system for different application domains.

A valid DMIL sentence will be constructed only if every operation and parameter substitution is compliant with the graph-construction type system and every streaming connection has an input connection and set of output connections that also comply with a DMI or domain type system. For all implementations of DMIL the compliance with graph-construction type system is mandatory. There may be some implementations of DMIL, though it is not advised, where a DMI or domain type system is treated as advisory. In this case, the connections which require an implicit type cast to achieve compliance should be detected and appropriate warnings issued. In all cases, the DMI or domain-type system must provide enough structure to allow auto-iteration systems to function correctly.

DMI and domain types serve the following purposes:

- They are used in the specification of PEs as follows:
  - to specify the permitted structure of domain values that may be supplied to each input;
  - to specify the structure of domain values that are emitted from each output;
  - to specify the relationship between the types supplied to inputs and those emitted from outputs; and
  - to specify the iteration behaviour of a PE, e.g. it may perform its sort algorithm for each of the lists of values that are provided on input data with an order determined by the corresponding representation of a less-than test provided on input ltCondition.

- They describe the sequence of values being transmitted along a connection. This will be inferred from the domain-type specification of the output to which it is attached unless a specific annotation has been attached to the connection to make the type more specific.
This description may be used to select appropriate pipes along which to transmit the stream.

- They may be used in an annotation to define the types emitted from an instance of a PE’s output or a particular connection. This is useful where the inferred output type is too general to inform validation and processing.

- They may be used to validate the DMI-type or domain-type consistency of a DMIL graph; in particular, to verify that the DMI or domain type of an output that is supplied to a connection is compatible with the DMI or domain type of each input to which its values are delivered.

- They may be used to specify casts that transform the interpretation of a stream of values prior to supplying them to an input.

- They may be used in textual representations both in stream literals (see below) and in the default presentation of results or errors.

**DMIL graph-construction types**

The purpose of the graph-construction type system is to (a) define the permitted graphs that may be constructed and (b) the permitted operations used in the construction process. For purpose (a) we introduce two categories:

- **Processing Elements** (PE), denoted by the reserved identifier `PE`, that have, inter alia\(^1\) a name, an identity, a tuple of input connectors and a tuple of output connectors; and

- **Connections**, denoted by the type identifier `Connection`, that transfer a stream of values from an output connector connected to their input to one or more input connectors that all receive those values. The connection type may be annotated with a description in terms of DMI or domain types of the values that may be transmitted along it. To permit flexibility and extensibility checks on this annotation must support inclusion polymorphism and type `any`.

The name associated with a PE denotes the class of algorithms it performs. This name may vary in specificity and may be abstract or concrete depending on the stage of DMIL processing. It is designed to communicate the purpose of the PE to humans and therefore remains consistent for successive versions of the PE and may be re-used by alternative implementations in different libraries. It should be discriminated from the identity of an instance of that PE. The identity of the PE is used to provide a good approximation to uniquely identifying a PE, i.e. it contains a UUID. This identity is different for each version and each alternate implementation and is used internally to identify the PE. The registries maintain mappings between names and identities and also record relationships between different identities—see \[43\].

Figure 4.1 illustrates these two categories.

There are three examples of PEs in the left hand column, illustrating variety in number of inputs and outputs, and the naming of these connectors; so that the PE type with identifier

---

\(^1\)See Section 2.2.1 and \[43\] for a full list; it is not yet clear which of those properties are bound to the type and which to the instance of a PE.
PE3 has two inputs, with identifiers $i_1$ and $i_2$, and two outputs, with identifiers $o_1$ and $o_2$. The input and output identifiers taken together form a set. Although many algorithms encapsulated in PE and used in DMI are described in the literature as having parameters, e.g. to set thresholds, these are all modelled as streams in DMIL. This is to permit such parameters to be supplied via any DMIL graph, e.g. one that obtains them from a database.

There are three examples of connections in the right hand column, showing a simple point-to-point connection, and connections to two and three inputs respectively.

DMIL has a number of base graph-construction types that are used in functions manipulating DMIL graphs and in constructing stream literals:

- the booleans, with type identifier Boolean, that are used in iteration and conditional expressions;
- the identifiers, with type identifier Identifier, that are used in reflections over DMIL graphs;
- the integers, with type identifier Integer, that are used for counting, indexing and in stream expressions;
- the reals, with type identifier Real, that are used for control calculations and in stream expressions; and
- the strings, with type identifier String, that are used for error reporting, monitoring and stream expressions.

DMIL has three type constructors:

- the function constructor, Function(p parameters) $\mathcal{T}$, where the parameters are a sequence of type, identifier pairs, and $\mathcal{T}$ is the result type (the type, Void, enables the definition of functions with no result);
- the array constructor, $\mathcal{T}[]$, defines the type which is a one dimensional array of elements with type $\mathcal{T}$; and

Figure 4.1: Elements of a DMIL graph
the tuple constructor, \( \langle T_1 \ \text{id}_1, \ \text{id}_2, \ldots, \ \text{id}_{1+i_1}; \ldots; T_n \ \text{id}_m, \ldots, \ \text{id}_{m+i_n} \rangle \), a tuple value is a set of triples: identifier, type, value of that type; the type of that tuple is the corresponding set of identifier, type pairs; the identifiers must themselves be a set.

The transitive closure of these three type constructors over all forms of PE, all forms of connection, and the base types, generates the set of possible types that may occur in DMIL sentences constructing DMIL graphs. These types are strictly enforced using structural equivalence by all DMI architectural components.

**DMI types**

To make it possible to reason about the graphs constructed with types in the previous section, we need to know about the semantics of the PEs that are connected by these graphs. A first step in this direction is to define data types that are commonly used in data mining and integration. In other words, we define a type system for dealing with inputs to and outputs from common data mining operations.

**Continuous** A possibly infinite number of values where interim values are allowed. Often these will be the output of measurements. Typical examples are time, temperature, weight and speed. Data of this type can be ordered and measured.

**Discrete** A finite number of values with no continuum between values, i.e., the values are distinct and separate. This type cannot have an implicit ordering, so only equality can be defined. This is often referred to as nominal data, but the strictly taken this is only true if the values are mapped to codes. This data cannot be ordered and cannot be measured.

**Ordinal** This is data that can be ordered, but not measured. For example, one could take user preference as numbers 1 to 6. These preferences can be ordered, but the difference in between two preferences cannot be measured.

**Categorical** This is data consisting of values that can be sorted according to category where each value is chosen from a small set of non-overlapping categories. A simple example is the type gender where the categories are ‘male’ and ‘female’.

**Discretised** Related to the Discrete type, but this defines data that has undergone a process to make continuous data discrete. Most popular method for this is binning.

**Normalised** Data from a continuous range is mapped to another continuous range.

**Unique** This ensures the values in the associated data are all unique. For example, keys in a relational databases adhere to this type.

**Sequence** This type includes any data that is ordered in some way. It can be a sequence of discrete data and it can be a sequence of continuous values. The crucial point is that the ordering over the data has been made, for example be a PE that sorts, and the data is now considered a sequence where order needs to be preserved.

**Unique Sequence** This is quite similar to the sequence, with the additional constraint that every item is unique and the data is a sequence.
**Periodical** This data type is concerned with time that repeats itself. Examples are full moons, and the tide. Sometimes this type is referred to as *periodical*.

**Text** This type is important in the context of text mining, but it is hard to define without going into technical details, e.g., strings of characters where the set of characters is from an iso standard.

**Domain types**

These types describe the values that arise from a domain and which may be passed along connections, may be supplied to the inputs of PEs and which may be obtained from the outputs of PEs. They are, in principle, specific to a domain. For example, the domain types `DigitalElevationModel` in an environmental application, `ISHgeneExpression3Dimage` in a gene expression application are never expected to meet and so do not need to be fitted into a common framework of types. However, there will be a set of *universal* domain types that are so pervasive that they are considered available in every domain. Similarly, there will be a standard set of application-domain type constructors that are available for recursive application in all domains. The universal domain types and the domain type constructors will be drawn from an ISO standard for general purpose data types [78].

An open-ended set of domain-defined types will allow each domain to add its own types independently from other domains. Each domain should define its own type matching rules, which will normally use type subsumption or inclusion polymorphism to allow processes definitions to have appropriate specified flexibility—this will not be partially explored within the ADMIRE project by using the *isa* relationship in ontologies. Where data is passed through the system and just needs to be faithfully transmitted, it can be given type *any*, which any other type matches.

The universal domain base types, `UDBT`, cover the usual values that are frequently required, such as: boolean, integer, real, number, date, time, char, text, etc. Mappings to and from their representations in databases and programming languages are defined by the standard, as is their textual representation. These `UDBT` are used in writing literals specific to a DMI-type or domain-type input according to standard rules. Similarly, they are used for presenting diagnostics when no DMI or domain mapping has been defined for a presentable form.

A domain may introduce other primitive types. Note that a type is primitive by convention, as in other contexts, e.g. within PEs that have algorithms to deal with that type, internal structure may be used in the processing of these values.

Every domain type, including those introduced by a domain, must have at least the following properties:

- A domain-type identifier, e.g. *Number* or *CustomerStatus* that is unique for the scope in which it is used, i.e. all of the DMI process specifications in which it is used. This means that no domain-defined type may have a domain-type identifier that is used in `UDBT` as they are universally available. Sentences in DMIL may be used to define types, give them names, collect them in libraries and register them in registries [43]. Once this has been done DMI and domain experts may work exclusively with these named types and avoid underlying detail.

- At least one method for injecting values of this domain type into the computation. This may be as a literal, e.g. 17 and -23 inject values into the `Integer` set of values, whereas
3.2756 and -1.00001E-8 inject values into the \textit{Real} set of values and \textit{Date}(2008,12,6) injects a value in the domain type \texttt{Date} but assumes a Gregorian calendar. Note that the injection methods can convert from another type and so may include the casts defined for the domain (see below).

- At least one method for extracting values of this type so that they are available in an external world. Typically, for simple types this will be a conversion to a text string, but more powerful transformations may be specified, e.g. for a 3D time series denoting a storms development and movement across a catchment area.

Other operations may also be useful, such as an equality test, a hash function that approximates the same function and a less-than test. Both equality and less-than are defined for all types in \texttt{UDBT} though those operators may be precluded from use by a type mapped to these values (see DMI-types above for examples). A mechanism is provided for defining these methods and other methods for the domain-specified types.

Difficulties are inevitable as ADMIRE systems must deal with legacy collections of data and with autonomous and diverse data resources over which the designers of DMI processes have little control. Therefore it will explore the extent to which descriptions of data resources, domain-type relationships and PE can enable automated type insertion. It will always be possible to assert that transformations apply at the input or output of a connection, so that (a) designers of DMI processes can control the trade-offs precisely and (b) so that the automation can be introduced incrementally. Asserted or automatically inserted transformations need only specify how to deal with one value. They are automatically applied to each element of the sequence.

The DMI-language processors must support the logical type systems which accommodate all of the types that can be constructed by the recursive composition of the type constructors, such as list, array and tuple, over the set of standard types \texttt{UDBT} and all domain-defined sites. The underlying implementations must represent these types in their metadata and represent the values corresponding to those types in the data streams passed between PE. One aspect of the representation of values is to divide them into quanta of the appropriate granularity to permit large values to be streamed when they cannot be passed by reference. Where the data streams are internal to a gateway the representation can be optimised. When transmission is between independently managed gateways a standard representation must be used, such as [78].

### 4.2 Secondary features

- It manipulates DMI processing elements (PE) that are provided by environment, are defined and registered by users or are obtained from libraries of related PE and components that work with them. These are normally described by abstract names represented in compliance with the WS-Addressing proposed standard [57] and the OGF recommended standard [86], so that mapping to specific PE implementations or services equivalent to the specified PE can be postponed and managed optimally by the enactment system at the time of enactment.

- Instances of PE are normally allocated or created by the enactment engine, however, some specific instances may be explicitly named in DMIIL sentences.
• Each PE has named input and output connections. These names of the inputs and outputs of a PE must form a set, i.e. the same name may not occur as an input and as an output for a particular PE. Variable numbers of inputs or outputs all with the same purpose are provided by using an array notation.

• Typically a PE will support automated iteration; i.e. it will repeat its operations for successive values appearing in its inputs and generate corresponding values on its outputs. The ways in which it may do this, e.g. in-lock-step, where one value is consumed from each input per cycle, or in merge-mode, where one value is obtained from a selected input and then the next input from which to select a value is determined, are varied and are specified in the description of each PE. This description also states how outputs relate to inputs and whether there are restrictions on the way in which an output can be used, e.g. pull-only.

• Composition is achieved by using connections, i.e. data streams, to connect outputs to inputs.

• Encapsulation is achieved by creating new elements, from a graph of connected elements, by specifying the inputs and outputs of the higher-level element, and connecting these to inputs and outputs of the internal implementation—other inputs and outputs are hidden from external view so that multiple instances of these encapsulations may occur in one DMIL sentence. It is recommended that this encapsulation is achieved by using a function—this allows the usual hiding mechanisms as well as control parameters.

• Functions may be used to name, parameterise and encapsulate any sequence of DMIL sentences optionally ending with a return expression. They can yield an encapsulation, be used to program patterns of composition and to represent DMI-process templates. Using functions, DMI experts and DADC engineers can encode repetitive and data-adaptive process patterns.

• Recursive application of composition and encapsulation can support arbitrarily complex DMI-process specifications. An environment that provides a suitable repertoire of pre-defined encapsulations and patterns can deliver a high-level and abstract view of DMI process that frees its users from DMI process details. It also opens up more scope for mapping and optimisation during enactment planning. The language should facilitate the provision of a variety of such environments by collaborations of DADC engineers and DMI experts. In particular they are able to define libraries each of which contains a set of components, mainly PE, intended to work together to deliver a coherent set of commonly required DMI functions, such as data cleaning or classification.

• The PE will be described, using a registry facilities, so that composition can be validated. These descriptions will include:
  – structural requirements over the structural types on inputs and outputs;
  – DMI or domain semantic compatibility over the format and interpretation of the values provided by outputs and consumed by inputs
  – relationships between inputs and outputs;
  – auto-iteration and termination behaviour; and
– restrictions on the use of outputs.

• Potential concurrency and permissible non-determinism will be inferred from the graph of connections and the description of elements.

DMIL is not the language used for coding the DMI algorithms in a primitive PE though it is intended for use by DADC engineers and perhaps by DMI-experts working with them to define DMI patterns and to define and refine DMI processes by composing PE. Many DMI-experts will work with graphical tools that ultimately generate DMIL, however, large-scale and detailed process design and refinements may be better served by using a syntactically supported text editing tool and working directly with DMIL—exploration of this aspect of DMIL utility is beyond the scope of ADMIRE.

4.3 DMI Tools

As the canonical representation instances of the DMI Language will be produced by many DMI-process design or generation tools and will be used by many DMI-submission systems. These generators of DMIL process descriptions will include:

• high-level graphical DMI-process design tools;

• portlets, and other tools tailored for domain experts, that submit DMI-process templates after they have been parameterised;

• high-level DMI text-based language processors that compile to DMIL; and

• DMI planning and enactment systems delegating tasks or subtasks to other enactment services.

Only some of these will be explored within the ADMIRE project. All of the submission systems will generate DMIL texts (sentences in the DMI language) by the following stages, often with repeated passes over these steps as developers iterate on the design of a DMI process:

1. Manipulate a high-level representation of a DMI process, in the first three cases in accordance with the directions given by the users of the DMI tool. The forms used at this level are the most easily read, but some of the information required for their interpretation will be implicit.²

2. Scan the representation of the DMI process and for each DMI component (PE, PE instance, data resource, data collection or function) obtain from an ADMIRE registry descriptive annotation of the element and annotate the component with that description. Successive versions of ADMIRE will increase the information in this annotation. It will include information about the number of inputs and outputs, type structures with which they must comply, the relationship between inputs and outputs, new properties of the outputs, iteration behaviour and so on, see [43].

²Initially the omitted information will be supplied by standard defaults, then it will be provided by ordered lists of options obtained by querying ADMIRE registries and in final prototypes selection from these options will be performed by an optimisation component in the enactment planing subsystem of DMI gateways.
3. Submit the fully annotated DMI-process description in an envelope that meets other requirements of the gateway. For example, the envelope will carry data on which statistics collection, authentication, authorisation and accounting depend. That information will include:

(a) identification of the user on behalf of whom the submission is made;
(b) identification of the tool or gateway making the submission;
(c) identification of the workbench on which that tool is mounted; and
(d) identification of the registry that workbench is bound to, thereby identifying the developer or user community.

The full annotation is essential information for the enactor. It will be used (1) to validate the DMI process for self consistency, (2) to check that the assumptions about elements that held at the time of construction still hold, and (3) to guide some aspects of optimisation and mapping. The second check is important as the elements may have changed since the DMI process was defined, e.g. because the process has been stored for re-use or because autonomous changes occurred since information about the elements was last collected; an inevitable distributed system phenomenon.
Chapter 5

DMI Registry

As discussed in [44, Section 5] and [43], the ADMIRE architecture envisages a future where many communities are developing DMI processes and using DMI services. Each community will share its specialised definitions and its selections from the standard definitions of DMI-processing components by using its own registry. The purpose of a registry is to define certain aspects of computational context of some stage of a DMI process. For example, in the context of a DMI workbench that houses a set of tools, including at least one DMI process design (PD) tool, it will keep track of set of Processing Elements (PE) that may be used via the PD tool. As the tool may be used to generate new PE it will also track these as they are iteratively developed, even before they are ready for execution. Conversely, a registry supporting a DMI gateway will keep track of all of the PE that can be used via that gateway; these must all be in some sense complete and valid in that context so that they are capable of being enacted. The process design inevitably occurs at a different (earlier) time from process enactment. Consequently the state of the computational context as represented by a registry during PD may be different from the state during enactment.

There is a many-to-many mapping between DMI workbenches being used by design teams and DMI gateways being used by design teams or application-domain user communities to enact the DMI processes. Therefore the computational context will differ depending on location. Consequently, separate registries are needed for each location. For the moment, we assume the following locations:

1. one for each community of DMI workbenches in use for related DMI purposes, i.e. a registry may support many workbenches but each workbench uses only one registry, and
2. one for each DMI gateway.

The roles of registries and their deployment in the ADMIRE architecture are illustrated in Figure 5.1. It shows registries in use in two contexts, outside of a gateway serving the needs of a community using one or more workbenches, and behind a gateway serving the needs of enactment implementation. As far as possible, the functions and representation used by ADMIRE registries should be the same in every location, so that the same semantics, code and service interface can be used throughout. The specific usage of a registry has been clarified in [43].
The ADMIRE will also need repositories to hold the actual definitions of the implementations of the objects named in the registries. It is as yet unknown whether these will be integrated within the registries or whether they will use some other service, such as a file store or myExperiment

In the latter case, entries in a registry will need to refer accurately to stored definitions in a repository.

The design of the repository is currently undertaken in a separate document internal to the project called “Advanced Data Mining and Integration Research for Europe-Defining Registry Requirements”.

---

1www.myexperiment.org
Chapter 6

DMI Enactor

DMI enactors are services that provide the functionally that can be used to implement data mining and integration requests. DMI physical query plans will include workflows that are sent to the DMI enactor via the DMI Executor. The DMI physical query plan will therefore consist of workflows that chain together the operations provided by DMI enactors. More abstract versions of these operations will be used in the logical query plans and probably also in the DMI query language as well.

Section 6.1 presents some of the data processing related functionality that must be provided by DMI enactors. Section 6.2 presents other functionality that DMI enactors must provide. Section 6.3 discusses the ADMIRE enactment engine that will provide much of this functionality. Section 6.4 describes the primary functional components of the DMI enactor supported by the ADMIRE platform.

6.1 DMI Enactor Functionalities Related to Data Processing

DMI enactors must provide a range of data mining and integration functionality. The following sections categorise and describe some of this functionality.

6.1.1 Data Access

Data pertinent for a particular DMI goal may reside almost anywhere. It is therefore necessary to provide (at least in principle) access data from any data source. However, an ADMIRE implementation may choose to have some preferred categories of data access, which are convenient and particularly well supported.

Data access [98] means 1) access to data from whatever storage system holds it in whatever format, data model and representation it is stored, this includes access to data in databases and access to data in files or bulk storage systems; 2) access to data using the protocols that its host storage management required. Data access involves querying and extracting data from:

- relational databases; using appropriate drivers
- native XML databases; using appropriate drivers
- semi-structured text files; e.g. using Lucene indexing library/language [2]
• structured binary files; e.g. using DFDL
• bulk storage systems, e.g. SRM, SRB and iRODS
• file systems
• web pages
• web services

Databases

Regarding data access to relational and XML databases, please refer to Sections 4.1, 4.1.1 and 4.1.2 in the document of redesigned and new activities for more detail. RDF databases should be supported, both as a source of primary data and to support semantic information used by ADMIRE.

Files and file systems

File-based access includes access to files, file systems and file content. By access to files we mean functionality similar to FTP’s get and put operations, i.e. the ability to pull down a whole file identified by a path name or URL of some sort, and likewise the ability to push a file onto a specified location on a file server. Access to file systems means being able to query and modify the directory structure of local or distributed file systems, such as AFS and NFS, including information about file ownership, access permissions, etc.. This access to files should comply with the emerging standards, e.g. OGSA-naming, Posix and the work of the OGF GFS-WG and may be mediated via the encapsulation of file operations in SAGA. While access to file contents means, for files of an appropriate type, such as XML or files with an appropriate description (e.g. DFDL, BinX, BFD, ESML) to access logical parts of the file though the use of a query string (e.g. XPath).

Some operations of access to files and semi-structured text files have been already defined in Sections 4.1.3 and 4.1.4 in. There is also an interface of access to files like OGSA-Byte-IO.

Web Pages

Web pages can be treated as data sources in several ways. At the simplest level data can simply be accessed using HTTP or HTTPS. Web databases can be accessed in a Grid related manner using OGSA-WebDB. Data can be obtain from websites by writing dedicated web scrapers. In all of these cases these will be considered read-only data sources.

Other Web Services

Including output from data mining and GIS services. These may be accessed through ADMIRE proxy or specialised gateway components. There will also be ADMIRE components using OGC standards that facilitate the composition and visualization of ADMIRE data in its standard forms, e.g. SEE-GEO.
6.1.2 Data Storage

Data storage functional component is to store intermediate results. This includes explicit use of external data services and storage systems. These would include the file stores, database and bulk storage identified in Section 6.1.1. There may be some storage systems closely integrated with the ADMIRE architecture so as to make such storage more convenient. Naming of these stored data items will be under the control of other ADMIRE components, but for convenience there may be an integrated unique-name generation system. Resource management will be the responsibility of the storage systems and services used.

Intermediate Results

Temporary results from data access. This may take various forms, such as in-memory storage of standard forms, relational data, etc., on-disc storage in files, buffers, databases and persistent objects, e.g. using hibernate. The intermediate storage requirement may be specified in a DMI high-level request or it may emerge during specialisation, mapping, optimisation and enactment. It must therefore support both explicit and implicit operations and the ADMIRE systems must be responsible for resource management and access control.

Named File Storage

Named file storage will be supported by mechanisms that correspond with those in Section 6.1.1.

Named DB Storage and Updates

The update services will include the update of the access provided in Section 6.1.1 and will in addition include the dynamic creation of databases, e.g. to accumulate results, and the bulk load of databases.

6.1.3 Data Movement

Data movement covers both bulk data movement and background data movement.

Bulk Data Movement

Bulk data movement can support movement of data to archive, for replication, for recovery, to bulk storage from operational storage, etc.. From data size and format point of view, the bulk data movements mainly include:

- High-performance large volume data.
- High-performance short messages.
- High-performance streamed data.

There are mainly five data movement technologies for large volume data movement such as FTP (File Transfer Protocol), SCP (Secure Copy Protocol), GridFTP, bbFTP, UFTP (UDP-based file transfer protocol). The Open Grid Services Architecture Data Movement Interface (OGSA-DMI) specification has been also defined. A comprehensive
evaluation criterion for different data movement technologies was proposed based on seven key dimensions in [107] as follows, which could be used for our reference for the ADMIRE architecture.

**Scalability** The amount of data (the volume) that must be delivered to a particular number of nodes (such as from point-to-point).

**Reliability** The measure of the amount of faults per data transfer.

**Ease of Use** The ease with which to install and configure, and use the data movement technology.

**Transfer Rate** The amount of data (bits/second) that can be transferred point-to-point over a fixed period of time.

**Cost to Operate** The estimated cost to operate a data movement technology once it has been deployed (estimated using a numerical scale, with 0 indicating negligible costs, and 5 indicating extremely high operational costs).

**Cost to Implement** The cost of procuring the data movement technology, along with its deployment cost (estimated using the same 1-5 scale above).

**Industry Adoption** The pervasiveness of the data movement technology (e.g., pervasive, bleeding edge). Initially, ADMIRE should use data transport mechanisms that are already supported by either USMT or OGSA-DAI.

**Background Data Movement**

Background data movement is characterised as low-cost and less urgent for replication, etc. A production system would require this and it should be described and used in the ADMIRE architecture, but it is unlikely to be used in prototypes.

### 6.1.4 Data Transformations and Filters

DMI enactors should be able to support a variety of data transforms. Some data transforms will convert data from one format to another while preserving most of the content. These transforms will be used to convert data to efficient internal format for processing and also to convert data into other formats used by specific services, external applications and visualisation tools. Example of a relevant external standard would be PMML (Predictive Model Markup Language) [88] to capture data mining results and provenance.

Other transforms will be computational and will be mainly used in the pre-processing phase of data mining, for instance, data cleansing, data obfuscation, data aggregation by computing averages, data normalisation and data discretisation. These transforms will typically operate on element by element bases for a single input stream.

The data transformation framework must also support generic operations. Generic operations are operations where part of the implementation of the operation is provided as one of the inputs. For example, the code to specify how to convert a data element in one date format into another may specified as a script that the operation executes. Other options for mobile code include passing Java jar files that includes classes containing the methods to be executed.
6.1.5 Data Combining

Data combing includes functionality such as join, merge, union and difference could be used here. Any data combing activities have been designed for OGSA-DAI, see Section 4.3 and appendix A.3 in the document of redesigned and new activities [98] for more detail. These introduce new control and buffering requirements compared as the rate at which data becomes available on each input stream may vary.

6.1.6 Collection Analysis

Collection analysis is the analysis of collections of data to produce new data, typically summary statistics. Examples of collection analysis include the generation of histograms [115] and the calculating statistics such as mean or standard deviation.

6.1.7 Data Sampling

Data sampling will be an very important part of the DMI process in the ADMIRE architecture. Data sampling can be used to reduce a data set in size in order to speed up initial investigations. Data sampling plays a vital part of implementing k-fold cross validation which a technique commonly used when data mining.

Several data sampling methods have been used for the reduction of data size by the selection of individual observations with a certain probability, such as Hoeffding bound [75], Load shedding [47] and Sketching [111]. Hoeffding bound to measure the sample size according to loss functions has been used in machine learning. The problem with sample is the unknown dataset size. The key to deal with sequence data thus is to find the error bound to obtain right sample rate. Load shedding refers to the process of dropping a sequence of data streams that has been successfully in querying stream. The method has introduced random sampling operators, or load shedders at various points in the query plan. Each load shedder flips a coin for each tuple that passes through it. With probability \( p \), the tuple is passed onto the next operator, and with probability \( 1 - p \), the tuple is discarded. To compensate for the lost tuples caused by the introduction of load shedders, the aggregate values calculated by the system are scaled appropriately to produce unbiased approximate query answers. The problem with load shedding is that it may not be suitable for mining patterns because the discarded stream could represent a pattern of interest in time series analysis. Sketching is the process of randomly projection a subset of the features by using projection along random vectors. The random vectors are generated by space-efficient computation of random variables. These projections are called the sketches. Synopsis refers to the process of applying summerization techniques.

6.1.8 Data Modelling

Data modelling is an important process in the ADMIRE architecture. Because users of the ADMIRE are from different domains and often have different analysis goals, different data modelling methodologies are thus required. The ADMIRE architecture should be able to meet different requirements and support the common data modelling techniques.

Generally, based on the execution strategy of data mining, data modelling in the context of data mining can be classified into two groups: Task-driven data mining and Learning-mode-based data mining.
Task-driven data mining includes classification, clustering, association rule, outlier and evolution analysis. Classification is the process of assigning new objects to predefined categories or classes. Typical classification algorithms include decision tree, rule induction, regression analysis, genetic algorithm, Bayesian network, neural network and Wavelet analysis. Clustering is used to place data elements into related groups without advance knowledge of the group definitions. K-means clustering and expectation maximization (EM) clustering are typical clustering algorithms. Association rule is to analyse and present strong rules discovered in databases using different measures of interestingness, which can predict occurrence of an item based on the occurrences of other items. Outlier analysis aims to analyse some elements of data sets that are inconsistent without expectations, based on the majority of other elements of the datasets. Trend and evolution analysis normally contain trend and deviation (e.g. regression analysis), sequential pattern mining, periodicity analysis, and similarity-based analysis.

Learning-mode-based data mining contains supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning. Supervised learning is to learn a function from training data (with labelled data). The training data consist of pairs of input objects (typically vectors), and desired outputs. The task of the supervised learner is to predict the value of the function for any valid input object after having seen a number of training examples (i.e. pairs of input and target output). Classification is also a supervised learning. Unsupervised learning is to learn a function or model from unlabeled data. Clustering is a typical unsupervised learning mode. Semi-supervised learning falls between supervised and unsupervised modes, which make use of both labelled and unlabelled data for training, typically a small amount of labelled data with a large amount of unlabeled data. Reinforcement learning is the machine interacts with its environment by producing actions so as to maximise some notion of long-term reward. Reinforcement learning algorithms attempt to find a policy that maps states of the world to the actions the machine ought to take in those states.

6.1.9 Data Visualisation

DMI enactors should include functionality for data visualisation. In some cases visualisation may simply require the results to be transformed into a format suitable for a client side visualisation tool. In other cases the visualisation can be done at the server and the visualisation result transferred to the client. Whether the visualisation is performed at the client of at a DMI enactor will depend on factors such as whether the visualisation output is larger in size than the raw data that produced, or whether the computation required to produce the visualisation to trivial enough to be left to the client, or and also whether the user needs to interact with the visualisation.

Currently, this is difficult to describe in specific subcomponents, as it is unclear what requirements these visualisations should have. An initial starting point could be the visualisations for a set of well-defined data mining tasks, such as decision trees, neural networks and association rules.

One of the types of visualisation we hope to support in ADMIRE is dynamic visualisation of a data stream that is part of a currently executing DMI task. This type of dynamic visualisation would allow users to take more control over the task. For example, a user could terminate a DMI task if the dynamic visualisation showed that it was not doing what the user expected.
6.2 DMI Enactor Management Functionalities

This section details functionality that the DMI enactor must provide that is not directly related to data processing but are instead higher level discovery and monitoring style operations.

6.2.1 Operations for Description

Many things in ADMIRE will need to be described so that can be discovered and used. This includes fairly static things such as data sources, services and operations and also more dynamic things such as virtual resources and stored workflows.

6.2.2 Operations for Discovery

DMI enactors should allow for discovery of the available data resources, the available transformation functionality and also the set of currently executing DMI evaluations.

Set of Available Data and Sources

It should be possible to inquire what data resources are available of varying sophistication, from simple name matching to queries that specify acceptable ranges of properties on support equivalences or approximate matching. USMT has a repository that contains all the services running. Also repositories connect to build up any topology of these. Best to put the semantic description in the properties field of the service. Then any service can go in and pull out this description. Suggestion from WP4 is to build a semantic registry in USMT. This will be a RDF version of OGSA-DAI. USMT registry for exact match is fully distributed, the semantic description is centrally located.

Set of Available Transformation Services

It should be possible to inquire what DMI enactors are accessible using queries of varying sophistication as outlined in Section 6.2.2. The same questions as those above in Section 6.2.2 are raised. This links to workpackages WP1, WP4 and WP5.

Set of Currently Executing DMI Evaluations

It should be possible to inquire (perhaps of a specific DMI gateway or perhaps of their P2P community) what enactments are currently executing. The query may range between the presentation of an enactment identifier to approximate match queries in various forms. The results should include the mapping of the returned enactments to a set of data resources and the mapping to the set of AEEs. We want to be able to monitor instances of OGSA-DAI activities. For example three instances of the join activity are running, then we want to monitor each to detect if they are all running efficiently. Individual activities are resources, which we can then discover directly.

6.2.3 Status Enquiries

These enquiries, which may name a specific target or may use range queries or approximate matches, obtain status information about their subjects. That status information can be
read by scheduling systems, progress monitoring systems and by problem detection and resolution tools. Note: All responses may need to limit the information required to match the authorisation of the enquirer. Requirement is that USMT provides access to usage records (as defined OGF-style). Also required are an event aggregator service and an alarm service. Perhaps not directly in USMT, but may be implemented in another work package. Most of the things below are life cycle data, which USMT always gathers. For specific properties, such as storage size, this should be in the properties of the resource.

The Status of Data Resources
These obtain the status of data and storage resources, including information about their capacity, organisation and current workload. Links to Section 6.1.2

The Status of Enactment Resources
These obtain the status of enactment services including their functional and non-functional properties, their current load and operational status.

The Status of Active Enactments
These obtain information about enactment requests from those that are queued, those that are being transformed, those that are being enacted, and those that are delivering their results.

6.2.4 Organising Enactment
In order to organise a specific enactment of a DMI request we need a component that allows to:

1. Parameterise a DMI request, e.g., bind it to a specific data source or specify a threshold.
2. Submit a DMI request to a specific DMI gateway or the community of collaborating gateways.
3. Monitor a DMI enactment, receiving error reports, warnings and progress information.
4. Stop a DMI enactment, e.g., when progress is not sufficient, and collecting diagnostics before cleaning up.
5. Steer a DMI enactment in more advanced ADMIRE architectures it may be possible to dynamically modify a live enactment to steer it towards a more useful result.

6.3 ADMIRE Enactment Engine
During the prototype phase ADMIRE will use OGSA-DAI as the basis for the implementation most of the enactors. The ADMIRE Enactment Engine (AEE) will essentially be USMT versions of OGSA-DAI servers with plug-in functionality and enhancements that support the type of operations discussed previously in this chapter and other functionality required by ADMIRE. The AEE must provide:

1. Loading and instantiation of activities
2. Inter-activity data communication

3. Dynamic generation and revision of activity graphs

4. External data transfers to and from external data resources, other AEEs and specialised services or computations.

5. The execution of activities with coordination between sources and sinks of data.

6. Monitoring the execution of activities for status reporting, logging and accounting.

7. Oversight of executions to ensure failure detection and resource clean up.


6.3.1 Activities

OGSA-DAI servers allows clients to execute data related workflows that are comprised of components called activities. Activities provide data related functionality of the type described in Section 6.1. Activities in OGSA-DAI are categorized into a number of functional groups in [98]. These activities may use an ad-hoc input and output format to accommodate the prevailing form of data they manipulate or they may use the OGSA-DAI standard internal tuple-stream format. This latter form is to be preferred unless there are compelling reasons for using an ad-hoc form. There are standard models for iterating over streams containing potentially nested sequences [98]. Again, these iteration forms should be respected when ADMIRE specific activities are developed.

Activities to access data sources

Activities must be available (potentially via suitable adaptors such as those provided by OGSA-DAI) to access data in databases, file stores, bulk stores and other data sources. They may need to deal with security and access control issues in order to access the data. They will need to convert the data to the standard forms used by other activities and to deal with any mismatch between the externally available units of access and those required by the internal ADMIRE activities. A particular special case that must be well supported is the recursive federation of ADMIRE networks. This is required so that one network can be used to provide a virtual data warehouse and then another ADMIRE network can access that exactly as if it were a fully materialised view of the warehouse’s data.

Activities to deliver to data sinks

When DMI activities generate data, they should be able to store the data for further use in standard forms, e.g. in a database ready to be used by some standard data processing tool. This means that as well as interfacing to the data writing mechanisms described in Section 6.1.2, there should be activities that can transform data to the required forms, essentially the inverse of the operations supported by Section 6.3.1.
6.3.2 Compositions of Data Activities

Activities composition is currently supported in OGSA-DAI by request workflows that define a directed acyclic graph (DAG) of activity instances. The ODE then generates a corresponding graph of activity instances interconnected by buffering pipes that carry streams of data between activities. The ODE then initiates and monitors the enactment. It is anticipated that in ADMIRE the ADMIRE Enactment Engines (AEE) will receive instructions in the form of a similar document that specifies a DAG of ADMIRE activities for that AEE. The form of that specification will be a dialect of the ADMIRE DMI Intermediate Language (DMI-IDL). This will contain all of details to configure, initiate, interconnect, run and control the corresponding enactment. It may also be annotated with information, e.g. input and output types, so that an AEE or intermediary may validate properties of the request. The activities may be connected by external pipes to activities running in another AEE instance, and via specialised protocol to data sources and data storages.

6.3.3 Operations to Manage Architecture

Create, monitor, start, stop, configure, and destroy instances of ADMIRE Enactment Engines, i.e. services delivering USMT life cycle management. Deploy is to create a specific instance of a service. Uninstall is a complete wipe of etc.

6.3.4 Operations to Facilitate Dynamic Environments

In ADMIRE it will be necessary to allow dynamic changes to the set of activities deployed in an instance of an ADMIRE Enactment Engine. It should be possible to both install and remove activities in the operational context. This in turn requires that the AEE treats this transactionally so that currently running enactments are not corrupted. This will allow execution plans to be re-optimised for example based on a change in available resources.

6.3.5 Managing Enactment Frameworks

The ADMIRE Enactment Engines need to be able to continue operation virtually indefinitely as other AEE and other services will depend on their continuity. This means that many configuration and maintenance tasks must be carried out while their services are effectively maintained. This requires tools that support such systems administration. Possible functionalities required are:

1. Moving an AEE to a different hardware platform.
2. Adding capacity to an AEE.
3. Upgrading the elements hosted by an AEE
4. Changing the configuration parameters of an AEE
5. Flushing an AEE’s workload and restarting it.
6. Moving live enactments onto a different AEE.
7. Flushing the work from an AEE and shutting it down.
6.4 The primary DMI enactor functionality supported by the ADMIRE platform

Based on the priority, the current ADMIRE platform will provide several services as the main implementation of the DMI enactor:

6.4.1 Data services

Data Integration data-integration services provide functionality required to link across data sources. This may involve everything from data transforms to achieve common schemas to statistical matching services that determine whether two records refer to the same person.

Data cleaning Data-cleaning services provide functionality required to prepare data for data mining. Examples of data cleaning include the insertion of an average value when there is missing data and the removal of anomalies in the data, e.g. pregnant males.

Data Mining Data-mining services provide the ability to execute data-mining algorithms to discover knowledge hidden in large volumes of data. Data-mining algorithms include classification algorithms that classify or cluster records in a data set. Additionally data-mining services provide functionality to evaluate the accuracy of the models produced by the data-mining algorithms.

Data Federation Data-federation services provide the ability to create new virtual data sources that are derived by combining selections from other data resources. These virtual data sources do not necessarily store any data, instead they use distributed query processing to materialise the required data each time they are used.

Distributed Query Processing Closely related to data federation, distributed-query processing supports the creation and execution of queries that that extract and combine data from several distributed data resources.

Persistent Storage DMI tasks may wish to store data sets or models within the gateway as intermediate results or for use in subsequent tasks or for later collection. The enactment technology will also use this storage for logs, audit trails and provenance information. The persistent storage infrastructure provides a means of storing, managing and referring to these data.

6.4.2 Core services

Data Access: Unlike the Data Services, this service simply provides access to data sources. Only simple queries and data extraction and transfer management are included in these functions.

Security: The security service provides a number of facilities to grant authorisation to individuals to obtain access to data and processing resources. It is important to note that these security services may function at many different granularities within data mining processes. Thus a general purpose security service provides a needed level of flexibility.
Resolution: Closely tied to Naming, the resolution service obtains access information (address) for named resources \[86\]. This service is nest-able, meaning that the DMI Platforms can themselves be the target of a resolution request. Resolution is critical for fail over and mobility of resources and services.

Monitoring: Closely tied to Notification, the monitoring services identify a number of high-level concepts, such as life cycle, and provide a monitoring service for these properties of services and resources. The framework is of course extensible and therefore DMI-specific concepts can also be the target of monitoring.

Audit and Provenance: In later stages of the project, it will be important to track and audit many of the transactions that take place in conjunction with data, particularly personal data. By incorporating this capability in the Core Services framework, its provision will be virtually universal once it is added to any DMI function.

6.4.3 Infrastructure management service

Lifecycle: This management capability defines the limits of existence of services and resources in the platform. Before a service or resource is created and after it is destroyed, it is not subject to any management functions.

Lifetime: This basic management function includes provisioning, configuration, and activation of resources. Most activities and data sources exhibit lifecycle states that map easily onto this pattern, as an example, it can be used to provide a uniform mechanism for controlling all components in a workflow ensuring that the resources they allocate are eventually recovered \[121\].

Naming: Distributed systems have always needed a rigorous concept of naming \[86\]. This functionality provides a naming infrastructure for all distributed resources that operate in and around the ADMIRE platform.

Discovery: There are several aspects to discovery. The two most important are:

1. identifying that a resource exists; and
2. providing a registry with some information about it.

The second of these normally provides some indication of the relationships between resources through the platform’s extensive self discovery mechanisms. The end result is a peer-to-peer like network on DMI Platforms performing a collective discovery function.

Notification: The Notification infrastructure is the core of all (non-bulk) communication between platform instances and resources controlled by them. This function includes subscription creation and management as well as distribution \[83\] \[68\] \[126\].

One of the main types of DMI enactor services will be OGSA-DAI services with data mining and integration activities. The idea of implementing the data mining tasks as OGSA-DAI activities is to move the computation close to the data and hence minimise the amount of data that has to be transferred. This is desirable as data transfer can be slow and can often be the performance bottleneck. The granularity of the OGSA-DAI data mining activities will be investigated in the early prototypes. It is anticipated that some data mining algorithms
such as sequence rules can be implemented by connecting several simpler activities together while other data mining algorithms will be implemented as a single activity.

Figure 6.1: Example OGSA-DAI workflows that together perform a data integration and data mining. Many of the functional components of the ADMIRE architecture can be seen in these co-operating workflows.

In addition to OGSA-DAI services other services that implement data mining algorithms will also be included. These services will typically require the data to be moved to the service before it can be processed.

Figure 6.1 shows an example of two OGSA-DAI workflows that together implement a data integration and data mining task. This example shows many of the functionality components of the ADMIRE architecture including data access, data transformation, data movement, data integration, data mining and data visualisation.
Chapter 7

ADMIRe Prototype Platform

This section discusses how the ADMIRe architecture maps on the prototype platform with the OGSA-DAI implementation. In this version of the document what in included here is simply a sketch of initial thoughts on this mapping and is far from being complete In future releases this section will become more concrete.

OGSA-DAI will used to implement the ADMIRe execution engine in the ADMIRe pro-
totype. OGSA-DAI shares many concepts with the ADMIRe architecture but also has some important differences that will require alterations to OGSA-DAI or additionally functionality added to the ADMIRe prototype outside of OGSA-DAI.

7.1 Data elements

The ADMIRe architecture defines data elements to represents the various data resources that can be accessed in ADMIRe. Data elements map closely onto OGSA-DAI’s data resources. Data resources is an extensibility point of OGSA-DAI so it should be easy to add any new types of data resource required by ADMIRe that is not currently available in OGSA-DAI.

The USMT version of OGSA-DAI will expose data resources as WSRF resources. This will allows clients to control the lifetime of, and access the resource properties of, data resources. Different data resource types can have a set of resource properties that are specific to that type. Currently OGSA-DAI’s generic data resource functionality does not provide a mechanism for a data resource to have its own type specific service operations.

OGSA-DAI does provide data sink and data source resources that can be considered to be data resources but these are exposed by services that provide operations specific to the data resource type. For example, the data sink resource has operations that allow data to be sent to the resource and the data source resource has operations that allow data to be pulled from the resource.

7.2 Processing elements

The ADMIRe architecture defines Processing Elements (PEs) to represent the atomic units of computation from which ADMIRe’s DMI tasks are constructed. Processing elements map very closely to OGSA-DAI’s activities. Activities are an extensibility point of OGSA-DAI so it should be easy to develop and add new activities to OGSA-DAI for each of ADMIRe’s processing elements if a suitable match does not already exist.
Processing elements are defined as having:

- a PE-class name that is indicative of the function of the PE;
- a PE-varient description that is indicative of a particular way of implementing the function and variations in its behaviour or results, this will normally contain a UUID to ensure unambiguous interpretation;

In OGSA-DAI an activity is exposed with an activity name and also an optional set of activity contracts. It may be possible to map activity contract to the PE-class name and activity name onto the PE-varient description.

To explore how this mapping works it is best to consider an example process element and see how it maps onto both models. Consider an implementation of the C4.5 algorithm to produce decision tree classifiers. An implementation of this algorithm could be a processing element in the ADMIRE architecture or an activity in OGSA-DAI. The PE-class name and activity contract name would be used to say the processing element was a decision tree classifier and the PE-varient description or activity name would be used to say that the version was the C4.5 decision tree algorithm.

Currently OGSA-DAI provides only two levels of naming. If the ADMIRE architecture requires more then either OGSA-DAI will need to be extended or this level activity metadata can reside in an additional database stores details of call ADMIRE processing elements and contains their mappings to OGSA-DAI activities.

Processing elements have a set of named inputs and a set of named outputs. OGSA-DAI activities provide are identical in this regard. The inputs and outputs of processing elements must specify the type of values that they read or write. OGSA-DAI activities are no so strongly typed. In OGSA-DAI the definitions of an activity’s inputs and outputs and the types they must take reside only in on-line user-level documentation. If a workflow given to to OGSA-DAI passes invalid types into an activity this would result in a runtime error. There is no pre-execution check of a workflow’s validity with respect to the type system.

The ADMIRE architecture requires and uses much more metadata regarding processing elements than OGSA-DAI has for activities. In addition to the above type information the architecture also specifies that process elements have specified auto-iteration behaviour, specified termination behaviour and detailed rules relating to inputs and outputs ordering and cardinality. None of this information is explicitly stored for OGSA-DAI activities so it is not available for automatic processing and reasoning. In the prototype we will construct a database of processing element metadata that holds all of this information. This information will be used at compile and optimisation time by the gateway. The database will also contain mappings to corresponding OGSA-DAI activities. This will allow the gateway to validate any request it receives according to the type system etc. and, if valid, compile it to an OGSA-DAI workflow. Thus OGSA-DAI is providing the pipeline workflow execution engine but not the higher level request validation and optimisation functionality of the gateway.

In the ADMIRE architecture a PE has:

- a PE-instance identity that allows operations specific to that instance to be performed, this will contain a UUID to ensure unique identification in logs, inter-gateway communication and so on;

This is not something that OGSA-DAI currently supports with OGSA-DAI activities. Activity instances are named but they are not exposed as WSRF resources. In order to provide
this functionality they would need to be exposed as resources. This will require changes to
OGSA-DAI. Exposing OGSA-DAI activities as WSRF resources with a set of resource prop-
erties specific to that activity and a set of operations common to all activities could probably
be achieved without too much alteration to OGSA-DAI’s design. To provide activity-specific
operations that can be executed on the activity resources would be considerably harder to
do.

Currently the set of operations you can execute on OGSA-DAI resources is static and it
not an extensibility point of OGSA-DAI. This is because OGSA-DAI runs with many different
presentation layers (Globus Toolkit, UMST etc.) and each presentation layer has to be coded
differently. If activities resources are to provide operations specific to the activity then an ac-
tivity developer would need to provide the WSDL for the service that exposes the operations
and also an implementation of that service for each of the presentation layers. Alternatively
OGSA-DAI could try to create these services dynamically for each presentation layer. It first
glance this sounds like a lot of work. It would be good revisit the design of the ADMIRE ar-
chitecture to further investigate this requirement and look to see if any alternative approaches
can be found.

7.3 Connecting processing elements

In the ADMIRE architecture, the processing elements are composed by connecting the outputs
of each PE to the inputs of other PE using data streams. Data streams communicate a
sequence of values that are delivered in the order that they are provided. The stream has
a flow-control function as a receiver can wait on data becoming available and can signal to
the sender that it no longer requires values. Similarly, the sender can signal that there are
no more values. A stream may operate with arbitrary granularity, may take advantage of
pass by reference and may offer an arbitrary amount of buffering. This functionality maps
on exactly to OGSA-DAI’s pipes that connect activities.

The architecture specifies that the outputs of one PE can fan out and become the inputs
of multiple other PEs. OGSA-DAI provides this functionality by automatically inserting a
Tee activity into the workflow when such fanning out is detected.

As well as obtaining inputs from the outputs of other PEs, PEs can also be given literal
inputs. In OGSA-DAI the set of literal inputs is restricted to a few primitive types. The
architecture plans to support a greater set of literals. These additional literals could be
mapping to OGSA-DAI simply by adding explicit serialization and deserialization activities
into any workflow that requires literals not supported by OGSA-DAI.

7.4 Other functionality

The ADMIRE architecture specifies that processing elements that require data elements will
have a reference to the data element passed to then like any other input values. This currently
cannot be done with OGSA-DAI’s data resources which must be specified as literals in the
workflow send to OGSA-DAI. Extending OGSA-DAI to provide this functionality will need
further investigation. There is nothing in OGSA-DAI other than a convention of activity
design that prevents the desired functionality but reimplementing every activity to support
this would be considerable effort. It would be better if a simple mechanism could be discovered
to provide this without re-writing activities.
The architecture mentions as an example a timestamp activity that outputs the current time. In OGSA-DAI’s push mode of operation such an activity would be useless as the time would get buffered and hence could be out of date when actually used. One solution may be to use a lazy evaluation time object that outputs the current time when first read. If this type of functionality was required often it may require support for activities that operate on a pull rather than push model, i.e. the activity outputs data only when asked rather than continually attempts to write (push) as much data as possible as all activities currently do.
Chapter 8

Evaluation of the Architecture

8.1 Evaluation Principles

We evaluate the design of the ADMIRE architecture through the implementation of prototypes, which are then used in several use cases. Each use case is chosen because it exhibits certain properties that aid in evaluating specific aspects of the architecture. We first discuss these properties and aspects and then map these on to the use cases.

**Large data sets** are simply collections of data where the raw volume, i.e. size in bytes, is more than one Terabyte. At present, in distributed systems we would find it hard to move one Terabyte of raw data and expect processing to handle this easily within most processing steps. One Gigabyte is a plausible size to transport and process on with current systems, but will still present problems with certain processing steps.

**Complex workflows** have at least two branches and at least two of these branches must be connected back into one processing element later on. This allows us to test large farming and non purely sequential cases.

**Real-time visualisation** is where the end-users, either domain specialists or DMI experts, want to observe partial results from a DMI request while it is still processing. This is useful to evaluate streaming models, as the output of streams must then be visualised as soon as possible.

**Large row sets** are defined as sets that have more than one million rows. Generally speaking, when data are gathered for input, they are converted into sequences of tuples. We take the largest number of rows in such a sequence when it is considered as input to a significant integration or mining step. From previous experiments we know these numbers are useful to evaluate the robustness of the architecture in terms of memory models. Also, different streaming models can be evaluated with these large numbers of rows.

**Data integration** is of interest only if there is more than one source. If there are two sources, we deem the use case moderately suitable for testing this aspect. Note however, that two sources may still have a very complex integration patterns.

**Abstract language patterns** are required to allow the use of common and complex data integration and mining patterns. These allow us to evaluate how well the architecture...
can handle the abstraction introduced by the DMI Language, and in turn how well the separation of concerns can be facilitated by the architecture.

**Parallel DMI operations** are defined as processing elements that can be parallelised in well-understood ways. These are interesting operations for testing the architecture as they are naturally compute-resource heavy and are difficult to distribute over loosely coupled compute resources.

**Data availability** is important to ensure we can use the data during the project to perform our experiments. Moreover, it is useful to know whether the data are available publicly to enable other researchers to use the data for their own experiments or to simply repeat our experiments.

### 8.2 The Use Cases

Six use cases are described here. Two of them — Flooding prediction and Customer Relationship Management (CRM) — are core parts of the ADMIRE project; the others arise from additional collaborations and funding sources.

**Gene expression in the developing embryo** This usage scenario involves data acquired from high-throughput studies in the context of understanding the normal embryonic development of the house mouse (*Mus musculus*) in terms of spatial gene expression patterns. This study, EURExpress-II a 13.5 million EURO EU-funded project, aims to generate spatial gene expression patterns for all genes in one developmental stage of the mouse embryo. To make these patterns useful to researchers world-wide, they need to be annotated to allow searching for appropriate images. This entails a team of anatomical experts manually curating each image by annotating it with all of the anatomical components that exhibit gene expression. This time consuming aspect of the EURExpress project is costly and will not be completed by the end of the project, which will leave about 20% of the images without annotation. By constructing a classifier that predicts, given an image, which anatomical components exhibit gene expression, the remaining images, worth about 2.9 million EURO, can be annotated automatically. The challenge here is the large volume of data, currently more than 5 TB, in combination with the high level of parallelism required for the various processing steps.

**Flooding prediction** This usage scenario involves integrating data related to several environmental aspects that influence the management of the river Váh in Slovakia and then using data mining to predict several important parameters about potential risk of floods. These data include meteorological data, definitions of catchment areas, river-flow models, water measurements (e.g. temperature, level, discharge) and soil types. The challenge here is the large amount of data in various forms originating from several sources which need to be integrated into a spatial framework. Furthermore, several of the input sources will come from models used by water-management authorities, which will require dynamic steering from active DMI processes. The important question is whether one can predict sufficiently accurately the risk of flooding for particular areas. A further possible outcome is to investigate the influence of forestry on the water level of the river. Essentially, using such predictions, one could influence future water levels by deforestation and forestation of particular catchment areas for precipitation.
CRM This usage scenario is set in the context of the mobile phone market, where it is vital to understand why customers change their subscription plans or worse, change to another provider. It is in conjunction with Comarch (Poland), which provides customer relationship management for several mobile phone providers. The challenge posed by this domain is in the large volume of records produced on a daily basis. Millions of phone customers make several phone calls and send several text messages a day. These records need to be sampled, preprocessed and then mined over a period of several months to build accurate predictors. All of this will happen in a dynamic environment as new customers are added, and records increase on a daily basis.

FLE data centre This usage scenario takes data from several data centres where the main goal is to predict future operational incidents. These data centres log much data about their usage as well as operational incidents such as hardware failures and overloads of the system. By integrating all of these data, the objective is to predict future incidents to allow timely handling of these incidents or even dynamic re-allocation of resources to prevent an incident from occurring. The challenge here is the vast amount of data produced by large distributed data centres, which will require much filtering and preprocessing steps before a predictor can be established.

Chinese medicine The context of this usage scenario is traditional Chinese medicine (TCM) and the application of its approach to life, health and illness to diagnose health conditions. Based on the meridian theory, novel non-invasive measurement and prediction methods for several vital parameters, e.g. blood pressure and blood glucose level, will be elaborated. Analysing the data obtained from meridians using advanced data mining techniques can lead to fruitful information about the human health state or other health relevant knowledge. The challenge here is to integrate the vital parameters stored as signal data measured by a device specially developed for this purpose and values of blood glucose obtained from a blood sample taken from the patient. The non-invasive measurement device produces raw signal data ranging from 100KB up to 1MB in volume depending on the device settings from one measurement. The measurement is performed several times per day as is typically done by diabetic patients who continuously control their blood sugar level. This leads to large volumes of data that need to be collected and integrated. Furthermore, for each patient an individual model has to be created using a neural network modelling approach that can be applied only to well preprocessed data. The preprocessing phase involves several iterations of compute intensive filtering methods which are common in signal data processing applications.

AstroGrid This usage scenario involves a data mining scenario from the UK’s AstroGrid Virtual Observatory Project that utilise the data gathered by the Virtual Observatory, which is governed by the International Virtual Observatory Alliance. The specific data mining and integration task is to integrate data derived from sky surveys at different frequency bands and then train a classifier that is able to detect quasars in these data. These data are large in terms of numbers, with up to 100M rows; these data require distributed compute resources in order to train such classifiers.

In Figure 8.1 we map the aspects to properties of the use cases. In this matrix we denote with colours how well each aspect can be evaluated using a particular use case. If in red, the use case is not suitable for evaluating that aspect. In green, it is suitable for evaluation. If
denoted in yellow, then it is moderately suitable, but considered a border case or perhaps needs more specific information to validate its suitability. For each aspect we show the facts on which the decision about suitability is based.

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Large data set (&gt; 1TB)</th>
<th>Complex workflow (branches)</th>
<th>Realtime visualisation (&lt; 1 min)</th>
<th>Large row sets (&gt; 1M)</th>
<th>Data integration (2+ sources)</th>
<th>Abstract language patterns</th>
<th>Parallel DMI operations</th>
<th>Data availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>EURExpress</td>
<td>4 TB</td>
<td>Signal processing</td>
<td>N/A</td>
<td>500,000X50</td>
<td>2</td>
<td>n-Fold, Fisher-ratio</td>
<td>Wavelets</td>
<td>Copied / public</td>
</tr>
<tr>
<td>Flooding</td>
<td>100 MB</td>
<td>Standard Weka</td>
<td>N/A</td>
<td>21X50,000</td>
<td>-20</td>
<td>Model selection</td>
<td>N/A</td>
<td>Copied / non-public</td>
</tr>
<tr>
<td>CRM</td>
<td>Unknown</td>
<td>Simple statistics</td>
<td>Statistics whilst processing</td>
<td>&gt;1M</td>
<td>2</td>
<td>No</td>
<td>Probably</td>
<td>Synthetic data / commercial</td>
</tr>
<tr>
<td>FLE data centre</td>
<td>1 GB</td>
<td>N/A</td>
<td>N/A</td>
<td>Likely</td>
<td>24</td>
<td>Unknown</td>
<td>Depends on task</td>
<td>Hard to get / commercial</td>
</tr>
<tr>
<td>Chinese medicine</td>
<td>100 MB</td>
<td>Loop, signal processing</td>
<td>N/A</td>
<td>5X10,000</td>
<td>1</td>
<td>Novel</td>
<td>Neural nets</td>
<td>Copied</td>
</tr>
<tr>
<td>AstroGrid</td>
<td>~100TB</td>
<td>Integration, several algorthms</td>
<td>Continues updating</td>
<td>10^8</td>
<td>2</td>
<td>n-Fold</td>
<td>Kernel density estimation</td>
<td>Public services</td>
</tr>
</tbody>
</table>

Figure 8.1: Use cases mapped against the aspects of the ADMIRE architecture. The colour coding denotes how suitable a use case is to evaluate a certain aspect; red is not suitable, yellow is moderately suitable and green is suitable.

In the next sections we will accumulate evidence that helps us to validate the design of the architecture in terms of aspects by applying prototypes of the architecture to the use cases.

8.3 Facilitating DMI using pipeline on the EURExpress-II use case

The ADMIRE architecture is designed using the pipeline streaming paradigm. The architecture allows execution to take advantage of parallel processing where it is available but hides most complexities from end-users.

OGSA-DAI [41] implements data streaming graphs of interconnected PEs (termed “activities” in OGSA-DAI) and is an inspiration for the ADMIRE DMI architecture. Thus, our initial prototype being used to explore data streaming for DMI maps DMI requests to sets of OGSA-DAI requests. Each OGSA-DAI request represents a subgraph of the DMI request that will be run on one OGSA-DAI service. These are coupled together using OGSA-DAI’s inter-process data streaming. All of the activities in one OGSA-DAI request are started simultaneously and execute in a pipelined manner as data arrives at their inputs. Within an OGSA-DAI request, data are passed between activities as references to Java objects. These are serialised for inter-process data streaming. This leads to the efficient processing of arbi-
trarily large data with a small memory footprint.

At the early stage of this project, we have chosen the EURExpress-II use case as an initial target prototype to investigate DMI tasks using pipeline parallel processing model. Here, we describe the result of the first evaluation of the architecture, which has also been written up and submitted to the IEEE 5th International Conference on e-Science.

8.3.1 Pipeline parallel processing model representation and OGSA-DAI

We propose a DMI task as a directed graph \[ G = (V, E) \] that consists of a set of processing elements (PEs), which can be represented as \( G = (V, E) \). A node \( V_i \in \{ V_1, \ldots, V_i, \ldots, V_n \} \) is represented as a PE. Each PE is a unit of data processing functionality, is implemented as a service (e.g., data mining and data integration operations) and has inputs and outputs. An edge is represented as \( E_i \in \{ E_1, \ldots, E_i, \ldots, E_n \} \). The edge connecting a pair of PEs is a data flow, viewed conceptually as a data stream. The streaming computational paradigm provides a powerful capability for processing large-scale data efficiently; a single pass over massive amounts of data using a small working memory produces great computing power. Each edge describes a data dependency between the PEs. The PEs are processes of the DMI task.

Based on the proposed graph model, there are two ways to perform parallel processing on the DMI task and achieve a measure of speedup: data and task parallelism pipeline processing.

**Data parallelism pipeline processing** means that the execution of the DMI task has a sequence and PEs are connected as a pipeline. The dataset can be partitioned into subsets so that some subtasks or PEs can be executed on different portions of the data in parallel (there is no interdependence of input data between executions), as shown in Figure 8.2 — the DMI task is composed of \( n \) PEs; the dataset is partitioned into two subsets; two instances of PE1 can execute on two sub-datasets in parallel.

**Task parallelism pipeline processing** represents the execution of the DMI task as a sequence and PEs are connected as a pipeline. Some sub-tasks or PEs can be independently executed in parallel (parallel branches), as shown in Figure 8.3 — the different PE2 and PEj are independent and can be executed entirely in parallel.

![Figure 8.2: Data parallelism pipeline processing for the DMI task](image1)

![Figure 8.3: Task parallelism pipeline processing for the DMI task](image2)
This proposed model can be captured and represented as a workflow. The advantages of a workflow include the automated execution of the processing pipeline in order, and allow easy and automated construction of the complex processes at various levels. There are different workflow implementations such as Taverna [112], Pegasus [97] and OGSA-DAI [41]. Taverna is a tool for the composition and enactment of bioinformatic workflows and Pegasus is suitable for managing any complex scientific workflows. OGSA-DAI is a pipelined data workflow system. OGSA-DAI can be used as a workflow that capture data streams through a set of activities (i.e. processes or tasks) and allow them to be composed in a pipelined manner. For instance, three OGSA-DAI activities, including data conversion, data transformation and data delivery, form a workflow and can be executed simultaneously, with each processing a different portion of the data stream. This can result in the efficient processing of arbitrarily large data with a small memory footprint. Additionally, because the activities can be described in terms of different granularities instead of only different types based on the inputs and outputs, this will make activities more composable as the workflow.

An example of OGSA-DAI workflow is shown in Figure 8.4. There are five activities: two queries (Q₁ and Q₂) on different databases, a data transformation (T), a join (J) and data delivery (D). Q₁, the aggregate output of T, together with Q₂ and other two activities J and D can be executed in a pipeline. In this case, we have implemented the proposed method using OGSA-DAI 3.0.

![Figure 8.4: An example of OGSA-DAI workflow](image)

8.3.2 Performance metrics

Since the DMI pipeline connects data streams sequentially, subtasks or PEs of this pipeline can be executed in parallel. To evaluate whether the parallel processing pipeline model functions well or not, performance measurement has been considered using the following three indicators:

**Speedup** (S) is defined as a ratio of the execution time of the DMI task (Tₜ) on one single processor and the execution time of the DMI task on multiple processors (Tₘ), represented as

\[ S = \frac{Tₜ}{Tₘ} \]  \hspace{1cm} (8.1)

**Efficiency** (E) represents the average utilisation of the number of processors, which measures the relationship between the speedup (S) and the number of processors (Pₙ) used. It can be described as

\[ E = \frac{S}{Pₙ} \]  \hspace{1cm} (8.2)
Throughput measures the overall performance of the DMI task execution. It can be defined as an inverse of the maximum execution time among subtasks or PEs and can be described as follows:

\[
\text{Throughput} = \frac{1}{\max\{T_{PE_i}\}}
\]  

(8.3)

8.3.3 DMI use case study

To demonstrate the feasibility of this approach, we apply the proposed pipeline approach to a real DMI use case: EURExpress-II [4][90]. An initial prototype using an OGSA-DAI workflow has been implemented for facilitating data mining and integration.

The EURExpress-II project aims to build a transcriptome-wide atlas database for the developing mouse embryo established by RNA in situ hybridisation. The project uses automated processes for in situ hybridisation on all genes of one stage of the development. The outputs are many images that represent different genes, which are then annotated by human curators. The annotation consists of tagging images with terms from the ontology for mouse anatomy development. If an image is tagged with a term, it means that anatomical component is expressing as a gene. So far, 80% of images (4 Terabytes in total) have been manually annotated by a human curator. For cost-effectiveness, a typical DMI task in this case is to automatically perform annotation by classifying the remaining 20% into the correct terms of anatomical components (this would be still 85,824 images to be annotated with a vocabulary of 1,500 anatomical terms). The input is a set of image files and metadata of the images. The output will be an identification of the anatomical components that exhibit gene expression patterns in the image.

High-Level DMI processes of the DMI task

To annotate the images automatically requires:

- the training stage to train these sample images with annotations and build classifiers;
- the test stage to test and evaluate the performance of classifiers; and then,
- the deployment stage to apply the classifiers to perform the classification of un-annotated images.

The high-level DMI process of the DMI task for this case is shown in Figure 8.5.

Applying the proposed parallel processing model into the DMI task

Based on the high-level DMI process in Section 8.3.3 we consider this DMI task as two subtasks: a subtask for the training stage and a subtask for the test stage. Each subtask is composed of various processing elements (PEs). For the training stage, there are nine major PEs that are related to data mining operations:

- \(PE_1\) (SQLq): SQL query for integrating images with annotations;
- \(PE_2\) (SSp): sample split for splitting the dataset as training datasets and test datasets;
- \(PE_3\) (RIF): read image file for obtaining the specified images to be processed;
Figure 8.5: EURExpress-II abstract DMI process

$PE_4$ (IRe): image rescaling for standardising the size of images;

$PE_5$ (IDe): image denoising for the reduction of the noise; $PE_4$ and $PE_5$ are subprocesses of the image processing in the High-level DMI task in Figure 8.5.

$PE_6$ (FG): feature generation;

$PE_7$ (FS): feature selection;

$PE_8$ (FE): feature extraction;

$PE_9$ (Clas): classifier construction.

For the testing stage, it contains all of other PEs that appear in the training stage except for feature selection $PE_7$ and classifier construction $PE_9$. It is noted that the testing stage only uses the results of $PE_9$ and $PE_7$ from the training stage to perform a classification and does not need classifier construction again. By composing the PEs, we have wrapped the training and testing stage into a DMI workflow, as shown in Figure 8.6.

To speed up this task we consider parallel processing, in this case from both data and task parallelism aspects as suggested in Section 8.3.1. For data parallelism, image samples can be divided and dispatched onto distributed computing nodes. The processing elements $PE_3$, $PE_4$, $PE_5$ and $PE_6$ on subsets of image data are executed in parallel. In addition, task parallelism can be employed in this case. Feature generation $PE_6$ in the testing stage and feature selection $PE_7$ in the training stage are purely independently tasks. In the test stage, $PE_8$ only uses the result of the $PE_7$ in the training stage to extract the most significant features.

The processes in the training stage consist of image integration, image processing, feature generation, feature selection and extraction, and classifier construction.
Figure 8.6: Applying the parallel processing model into the DMI task
**Image integration:** Before starting data mining we need to integrate data from different sources: the manual annotations have been stored in the database and the images are located in the file system. The outputs of this process are images with annotations.

**Image processing:** The size of the images is variable. We apply median filtering and image rescaling to reduce image noise and rescale the images to a standard size. The outputs of this process are standardised and “denoised” images, which can be represented as two-dimensional arrays \((m \times n)\).

**Feature generation:** After image pre-processing, we generate those features that represent different gene expression patterns in the images. We use wavelet transform methods to obtain features; by using wavelet transformation, the image can be transformed into the wavelet domain and represented by both the spatial and frequency domains simultaneously. This can identify the characteristics of images at different scales. The numbers of features generated using wavelet transformation are large and depend on the image size. For example, an image with dimension as 300 \(\times\) 200 will generate 60,000 features. These features are represented as 2-dimensional arrays and are stored in files.

**Feature selection and extraction:** Due to the large number of features, the features need to be reduced and selected for building a classifier. This can be done either by feature selection, feature extraction or both. Feature selection selects a subset of the most significant features for constructing classifiers. Feature extraction performs a transformation on the original features to reduce the dimensionality of the feature set and obtain representative feature vectors for building up classifiers.

**Classifier construction:** The main task in this case is to classify images into the right gene terminologies. The classifier needs to take an image’s features as an input and, for each of the anatomical features, output a rating as ‘not detected’, ‘possible’, ‘weak’, ‘moderate’ or ‘strong’ (in the current experimental stage, we use two types ‘detected as a gene’ and ‘not detected as a gene’). We have built separate classifiers for each of the anatomical components and consider them independently.

The test stage comprises image integration, image processing, feature generation, feature selection and extraction, and prediction validation. At this stage, we use the result from the training stage and do not have to build the classifier again. To evaluate further the performance of the classifier, we also use \(k\)-fold cross validation. The data set is divided into \(k\) subsets. For each validation, one of the \(k\) subsets is used as the test set and the other \(k - 1\) subsets are put together to form a training set. Then the average error across all \(k\) trials is computed.

The deployment stage involves the deployment of the classifiers onto the system and the application of the classifiers to annotate the images automatically with the correct terms, and deliver results to the users.

**Implementing the proposed parallel processing model using OGSA-DAI**

We have prototyped our workflow system within the ADMIRE Testbed using the OGSA-DAI data workflow engine. All the required PEs have been implemented as OGSA-DAI activities. There are two categories of PEs in the system: major PEs and supporting PEs. Here, the major PEs represent operations that are related to data mining, for example, \(PE_1\) (SQLq)
for obtaining data samples that integrate images with annotations, $PE_2$ (SSp) for splitting data samples, $PE_3$ (RIF) for reading image files, $PE_4$ (IRe) for rescaling images, $PE_5$ (IDe) for denoising images, $PE_6$ (FG) for generating features, $PE_7$ (FS) for selecting features ($PE_7$ is composed of $FS_1$, $FS_2$ and $FS_3$ that partially execute the function of this PE), $PE_8$ (FE) for extracting features and $PE_9$ (Clas) for constructing classifiers. The supporting PEs represent operations to connect and compose the workflow, e.g. the $FM$ activity is used to merge features streams generated from distributed computing nodes to form a single data stream for classifier construction $PE_9$.

Figure 8.7 and Figure 8.8 show the architecture and specific parallelisation implementation of the DMI workflow for the EURExpress-II use case.

Three main components in this architecture shown in Figure 8.7 are client, controller ODS (OGSA-DAI Server) and executor ODSs. ODS represents an OGSA-DAI server that runs activities of a workflow. The client is responsible for submitting a request or a workflow to a controller ODS. The controller ODS has two tasks: the first task is to partition and dispatch data to the executor ODSs running on distributed computing nodes; the second is to submit the workflows to the executor ODSs. The executor ODSs execute the workflows. For instance, in Figure 8.8 the executor ODS at the computing node P1 execute the workflow that contains $PE_3$, $PE_4$, $PE_5$, and $PE_6$ and part of $PE_7$ (i.e. $FS_1$). Similarly, the same workflows are run on other distributed computing nodes from P2 to P7. The executor ODS at processor P8 runs a workflow consisting of major PEs (i.e. $PE_3$, $PE_4$, $PE_5$, $PE_6$, part of $PE_7$ [i.e. $FS_2$ and $FS_3$], $PE_8$, and $PE_9$) and supporting PEs (i.e. FM). We have adopted a multithreaded TCP socket implementation for parallel communication across multiple machines. The corresponding supporting PEs are represented as Socket Sender for sending the parameters and Socket Receiver for receiving the parameters from distributed computing nodes.

**Performance evaluation**

To show the feasibility of the proposed parallel processing model we measure the performance of the prototype under different conditions:
Figure 8.8: The detail implementation of the DMI workflow

- changing the size of data set (i.e. number of image samples);
- varying number of processors (i.e. machines).

In addition, since our system follows the streaming philosophy we have also investigated the total execution time of the workflow and the execution time for each major PEs, to give an indication of how well this streaming model works.

We have set up nine machines for experimental use based on different scenarios. One machine (an Intel 2 GHz Core 2 Duo MacBook with 2GB RAM) acts as a client to submit a request. The other eight machines act as one ODS controller and multiple ODS executors respectively. These machines are all Intel Core Duo 2.4 GHz PCs with 8GB RAM. Our workflow system is deployed onto these machines with prerequisite software packages including OGSA-DAI 3.1 GT, Java 1.6, Globus Toolkit 4.2 Web Services Core, Jakarta Tomcat 5.5 and Apache ANT.

Figure 8.9 shows the speedup of the workflow system running on a number of distributed computing nodes with 6,400 images and 12,800 images respectively. The result shows that speedup increases with an increase in the number of computing nodes in both cases. Interestingly, when the number of images increases from 6,400 to 12,800, the speedup gain is greater. We also find that there is a maximum speedup with input images as 6,400 when the number of computing nodes is equal to 3. When the number of computing nodes is greater than 3, the speedup levels off. The main reason behind this is that the average size of images to be processed on each machine is relatively small when we increase the number of computing nodes: for example, 800 images on each of 8 computing nodes for a total 6,400 images. The time overhead from communication, middleware and data transfer [117] comes to dominate such that the system cannot achieve greater speedup.
Figure 8.9: Speedup of the system

Figure 8.10 shows the efficiency that measures the average utilisation of distributed computing nodes. It demonstrates the average utilisation decreases with the decrease of the number of computing nodes.

Figure 8.11 shows the throughput. It demonstrates that the throughput increases with the increase of the number of computing nodes under both image size as 6,400 and 12,800.

To investigate how the streaming model works in this case, we also compare the total execution time of the system and the execution time of each of the individual processes as shown in Figure 8.12 and Figure 8.13. The result shows the streaming model works very well — for 12,800 images input, the total execution time in Figure 8.12 at one computing node is equal to 1,382 seconds, while the accumulation of the execution time of each PE in the system under the same conditions is 3,473 seconds. This means the data in the system are accessed in a streaming way so that the whole workflow executes quickly even if each PE takes more time for processing. In Figure 8.13, we also find that the execution time for each PE decreases with the increase in the number of computing nodes under both inputs of 6,400 and 12,800 images.

8.4 Summary and discussions

The preliminary experimental results demonstrate our proposed parallel processing model functions well. With the increase of the number of computing nodes, the speedup of the system increases. When the number of images increases, the system gains more speedup, which demonstrates the advantage of this system for processing large-scale data sets. The efficiency of the system across machines decreases with an increase of the number of computing...
Figure 8.10: Efficiency of the system

Figure 8.11: Throughput of the system
Figure 8.12: Execution time of the system

Figure 8.13: Processing Elements execution time of the system
nodes. Under both input cases (6,400 and 12,800 images) the efficiency is essentially the same. This means, relatively speaking, the processors can be well utilised under workloads of large data sets. The throughput of the system increases with the increase of the number of computing nodes. However, when processing large data sets, the throughput is smaller than that of processing small data sets. The execution time of individual PEs decreases with the increase of the number of computing nodes. Additionally, the experimental results also prove the concept of the streaming model, in this case by a comparison between the total execution of the system and the execution time for individual PEs. The accumulated time for each individual PEs is greater than the total execution time of the system.

This experiment above provides a starting point for understanding how a DMI task is executed in a pipelined way and how data parallelism and task parallelism patterns are applied for efficient parallel execution. On the other hand, during the experiment we have also recognised several key questions that may influence the ADMIRE architecture design as follows:

- How can the DMIL language help to describe a DMI workflow at the abstract level and then map it to a concrete workflow, automatically splitting and executing the DMI task in parallel from a top-down view?

The automation of composing, splitting and executing a DMI task is a primary goal for the ADMIRE architecture. To do so, it requires both support from the DMIL language and an understanding of workflow patterns. The current experiment is performed from a bottom-up perspective without using the DMIL language. The composing and splitting of the DMI task is currently done manually, a process which is obviously not suitable for large-scale distributed data intensive systems.

The work of Workpackage 2 in the near future will be to investigate how the DMIL language can be used in the current architectural experiments; specifically how best to describe a DMI request using DMIL, how to map this request into physical resources and what information is needed in the Registry to allow this mapping. Additionally, future work will also collect common workflow patterns and run trials on these acyclic directed graphs for finding optimal split patterns so that the system can automatically make a decision based on an optimisation strategy of splitting patterns.

- What is the categorisation granularity of the PEs?

The Registry in the ADMIRE architecture is used to describe PEs. Some PEs can be split into smaller PEs for efficient parallel execution; some others cannot. For a given optimisation strategy, the system needs to find the best suitable PEs and compose them for efficient execution of a DMI task. This composition of PEs may take place at different levels; thus the categorisation granularity of PEs is important in the automation of the composition of the PEs.

For example, during our experiment, feature selection is calculated based on Fisher’s Ratio, represented as follows:

\[
F = \frac{(m_1 - m_2)^2}{v_1^2 + v_2^2}
\]

\[(8.4)\]

where \(m_1\), and \(m_2\) are the means of class 1 and class 2, and \(v_1\), and \(v_2\) are the variances. Based on the equation above, this PE is a collective activity, because it has to collect all data in question and obtain a final result. This requires intensive computing. Since
this PE (Fisher’s Ratio) requires means and variances of features for all input samples, to solve the computing problem we split this PE into smaller PEs at a finer level (such as mean and variance) that perform parts of the function of the Fisher’s Ratio PE, and which are then implemented in parallel, as shown in Figure 8.8. In this way, computing time is reduced and performance improved.

To do this automatically in the ADMIRE architecture requires an understanding of a PE not only at an abstract level but also at a detailed level: we need to know what the PE is doing (i.e. what specific algorithm is it implementing) so that the PE may be split into smaller PEs for efficient execution.
Chapter 9

Related Work

Distributed data mining architectures are the underlying fabric to allow the extraction of knowledge from distributed data sources by making use of distributed computing and communication resources. In this chapter we will represent the state-of-the-art in these architectures and discuss their advantages and disadvantages with respect to the high-level requirements we introduced in the previous chapter.

9.1 Distributed data mining architectures

To date, a significant effort has been done on structuring distributed data mining systems. Generally, distributed data mining systems have been developed as Agent-based, Client/Server and Peer-to-Peer infrastructures.

9.1.1 Agent-based Model

Agent-based distributed data mining systems employ one or more agents to analyze and model local datasets, which generate local models. These local models generated by individual agents can then be composed into one or more new 'global models' based on different learning algorithms, for instance, JAM [122] and BODHI [92]. JAM is a Java-based distributed data mining system that uses a meta-learning technique. It has been applied to detect fraudulent behaviours. The architecture consists of local databases of several financial institutes, learning agents and meta-learning agents. Agents operate on a local database and generate local classifiers. These local classifiers then are imported to a data location where they can be aggregated into a global model using meta-learning. Similarly, BODHI is a Java and agent based distributed data mining system. BODHI also notes the importance of mobile agent technology. As all of agents are extensions of a basic agent object, BODHI can easily transfer an agent from one site to another site, along with the agent’s environment, configuration, current state and learned knowledge. Figure 9.1 shows the BODHI architecture.

9.1.2 Client/Server-based Model

The client/server model of distributed applications has been applied into distributed data mining systems. Considering cost effectiveness, existing data mining systems under the Clien-
t/Server model can be classified into three-tier client/server, clusters, grid and cloud-based infrastructures.

**Three-Tier Client/Server architecture**

Kensington [69] is a distributed data mining solution based on a three-tier client/server architecture as shown in Figure 9.2. The client application provides interactive visual programming of data mining tasks, and three-dimensional visualisation of data and analytical models. The application server handles user login and authentication, provides services for one aspect of the application, namely user object management, task execution, mining component management, database access and data storage. The third-tier server provides high-performance data mining services located high-end computing facilities that include parallel systems.

Discovery net [38], shown in Figure 9.3, has built on the work of the Kensington system, extending its architecture to enable its capabilities to be used in a grid environment. It provides the middleware for knowledge discovery services for a wide range of high throughput informatics applications including drug discovery, remote sensing and geo-hazard prediction. Discovery net consists of three types of servers: Knowledge Discovery Look-up and Registration servers that support the publication and retrieval analysis services and provide a store of service descriptors including functionality and input/output types; Meta-information Servers that provide services for data type management including type checking and data composition; and Knowledge Servers that include services for warehousing knowledge discovery workflows and generating reports in addition to application generation services that allow servers to deploy their own workflows as new services.

The IntelliMiner [113] optimises the execution of tasks using task farming and the distributed DOALL parallel primitive over clusters of SMP workstations. The architecture,
Chapter 9. Related Work

Figure 9.2: Kensington Client and Server Components (source: [69])

Figure 9.3: Discovery Net Architecture (source: [38])
shown in the Figure 9.4 separates the data server and computing server.

Figure 9.4: InteliMiner Architecture (source: [113])

PerfExplorer [80] is a performance data mining framework for large-scale parallel computing. The architecture supports both a Client/Server model and also a multi-threaded, standalone option. It is built upon an open performance data management framework [93] that provides a common, reusable foundation for performance results storage, access and sharing. The architecture of perfExplorer has integrated data mining techniques in the analysis of large-scale parallel performance data and provided interfaces with a set of statistical analysis and computational packages, including the R system [19], WEKA [30] and Octave [13]. The architecture is shown in Figure 9.5.

Figure 9.5: PerExplorer Architecture (source: [80])
Cluster-based model

Papyrus [38] is an architecture that enables users to mine distributed data sources and find optimal mining strategies over meta-clusters or super-clusters. Each node of the cluster decides whether data should process locally, or send data to another node for processing, or a hybrid approach where the data is processed to produce an intermediate result that can then be sent to another node for further processing. The architecture supports three strategies: Move Data, Move Results, Move Models, as well as combinations of these strategies. Papyrus is built over a data-warehousing layer that can move data over both commodity and high performance networks. IntelliMiner also supports data mining over clusters.

An All-Pair production system [109] has been proposed for data intensive applications such as data mining and biometrics. The work provides users a high-level abstraction, while hiding low-level system details. A prototype of an All-Pair engine that runs on top of a conventional batch system and exploits the local storage connected to each CPU has been constructed. The engine first selects the minimum data transfer time, computing time and turnaround time by using predefined functions that provide performance prediction of executions, and then distributed data to the compute nodes via a spanning tree. After that, the engine constructs batch submit scripts for each of the grouped jobs and queues them in a batch system with instructions to run on the nodes where data is available. The engine can monitor if the resources are available or overload and provides an estimation of completion time. Once the batch system completes jobs, the engine collects results and cleans up.

Grid-based model

Knowledge grid [65] uses basic grid services such as communication, authentication, information, and resource management to build parallel and distributed knowledge discovery tools and services. The architecture is composed of two layers: Core K-Grid and high-level K-Grid. The Core K-Grid layer implements basic services for the definition, composition and execution of a distributed knowledge discovery application over the Grid. Its main goals are the management of metadata describing features of data sources, third party data mining tools and algorithms. Moreover, the layer coordinates the application execution by attempting to fulfill the application requirements on the available grid resources. The high level K-Grid layer includes services used to compose, validate, and execute a parallel and distributed knowledge computation. This layer also offers services to store and analyze the discovered knowledge.

GridMiner [60] is an infrastructure for distributed data mining and data integration in Grid environments. The architecture consists of basic elements: a Service Factory for creating and managing services; a Service Registry for registering services; a Data Mining Service that provides a set of data mining, data analysis algorithm; a Pre-processing Service for data cleaning, integration, handling missing data; a Presentation Service and an Orchestration Service for handling and long-running jobs.

FREERIDE-G [82] is a framework for rapid implementation of data mining engines in the Grid, that supports a high-level interface for developing data mining and scientific data processing applications that involve data stored in remote repositories. A performance prediction model [81] has been incorporated into the framework, in which a set of performance metrics such as data retrieval, communication and processing time including interprocessor communication time and global reduction are defined. The system therefore can predict and evaluate performance of resource selection, i.e. choosing computing nodes and replica of the
A data diffusion framework [119], shown in Figure 9.6, has been proposed for supporting large-scale data exploration that acquires compute and storage resources dynamically, replicates data in response to demand, and schedules computations close to data. The architecture is built on Falkon [118] and is a hybrid architecture. The system consists of a set of executors (dynamically allocated) that cache and analyze data; a dynamic resource provisioner that manages the creation and deletion of executors; and a dispatcher that dispatches each incoming task to executors. The provisioner uses tunable allocation and de-allocation policies to provision resources adaptively. Data caches and data-aware task scheduling mechanisms have been also incorporated into the executors and dispatcher to achieve a separation of concerns between the core logic of data-intensive applications and the complicated task of managing large data sets. Individual executors manage their own caches, using local eviction policies, and communicate changes in cache content to the dispatchers. The dispatcher sends tasks to nodes that have cached the most needed data, along with the information on how to locate needed data. An executor that receives a task to execution will access required data from its local cache or request it from peer executors.

**Cloud-based model**

Grossman et al. [87] have developed a cloud-based infrastructure for data mining large distributed datasets. The architecture consists of the Sector storage cloud [21] and the Sphere compute cloud [25] that are designed for analyzing data sets using the computer cluster connected with wide area high performance networks (10+ Gb/s). The architecture uses Sector to provide long-term persistent storage system to large datasets that are managed as distributed indexed files. For processing data stored in the Sector/Sphere model, shown in Figure 9.7, a stream programming model is developed in the architecture. Sphere splits the input files (Sphere Streams) into data segments and the Sector slave nodes process the segments in parallel. The output of each process can be either sent back to the client, written to local disks, or sent to different ‘bucket’ files. Neither Sector nor Sphere provides support for data semantics. It is up to the user to decide how to explain a data file. In order to allow Sphere
to understand the record structure and hence split the files, users need to provide an index file for each data file.

The approach used in the architecture is to store the data persistently and to process the data in place without moving. The data waits for the task or query. In contrast, a grid system generally transfers the data to the processes prior to processing, which can incur a heavy data transport time overhead.

Some other cloud-based infrastructures for data-intensive application include BigTable [67], Google File System (GFS) [79] as well as MapReduce [74] and Hadoop [55] that couple data and computing resources to accelerate data-intensive applications. MapReduce and Hadoop with underlying file systems GFS and HDFS are especially designed for racks of computers in data centers. Both systems use information about clusters and racks to position file blocks and file replicas. These tightly coupled systems mean that users have to adapt them for their own applications.

9.1.3 Peer-to-Peer-based Model

Research in Peer-to-Peer (P2P) distributed data mining [73] describes both exact and approximate distributed data mining algorithms such as majority voting, frequent item-set mining, and K-means clustering for potential use in solving distributed data mining problems in P2P networks. Some research on primitive computing operations have been done. Kempe et al. [99] investigate gossip based randomized algorithms and prove the error will go to zero if the algorithm runs uninterrupted. A newscast model based on epidemic algorithm [102] has been also developed to address the problems of analysing data which are scattered over P2P networks by calculating basic data statistics. The common feature of these approaches is that they all require resources that scale directly with the size of the system. This is different from local algorithms [46] that compute the result using just a handful of nearby neighbours and the number of resources required is independent from the size of the system.
9.2 Data integration

Data integration is the process of combining data residing at different sources and providing the user with a unified view of these data by shielding the underlying multiple heterogeneous and autonomous data sources from the users [103, 128]. Data integration can be classified into two types: architectural data integration and semantic data integration.

9.2.1 Architectural data integration

Architectural data integration mainly focuses on structuring the architecture of the integration to create the unified view to facilitate data access and data reuses from different data sources based on the transformation of queries or data by using mappings as if the users are accessing through a single access point.

From the system architecture point of view, there are four main approaches to the architectural data integration that happen at different levels in the architecture: data warehouse, federated database, peer-to-peer integration and programmatic integration.

Data warehousing

Data warehousing is a popular integration approach, where data from several sources are extracted, transformed, and loaded into a single structure and stored in a centralized locations and can be queried with a single schema. This can be perceived architecturally as a tightly coupled approach because the data reside together in a single repository at query time [95, 100]. This method is not only unifies schema, it also relocates the actual data permanently. User queries are answered from the relocated data source that may not be updated when underlying original data sources have changed. Data warehouse is a kind of physical integration by transferring data to a new data storage. Figure 9.8 shows a typical data warehousing architecture and process [56]. The CDW represents the corporate data warehouse and contains highly aggregated data and can be organised into a multidimensional structure. The ODS(Operational Data Store) reflects source data in a uniform and clean representation.

![Data Warehousing Architecture and Process](source: 56)

Figure 9.8: Data Warehousing Architecture and Process (source: 56)
Federated database

A federated database is a way to transparently integrate a set of distributed, heterogeneous and autonomous databases [64] [106] [120]. It provides a uniform data access solution by logically integrating data from underlying autonomous data sources into a federated database. Users of a federated database system observe a single global schema that includes the data from all the databases in the federation. In a federated database system, the original data sources are not relocated, instead the federation layer makes calls to the original data sources and combines the results. The queries that are answered are always up-to-date. Figure 9.9 shows a sketch of a federated database architecture.

Peer-to-Peer integration

Peer-to-peer data integration means the integration of distributed data sources in a peer-to-peer network. Peer-to-Peer data integration has no global schema. Mappings between peers are defined and form a logical association between peers [123]. During query evaluation, queries can be passed along the logical associations, thus to the entire network.

Programmatic integration

Workflows and software APIs can support the development of bespoke complex data integration scenarios by defining the process step by step. Some steps fetch data from certain data resources; other steps can transform data or merge data. Workflows support modelling, execution and maintenance of processes. OGSA-DAI [98] supports data related workflows ideal for data integration tasks. Other workflow systems that co-ordinate the invocation of web services, such as Taverna [28], Kepler [12] and ActiveBPEL [1], can also be used for data integration if a suite of web services are available that provide data integration functionality. Current workflow systems often require a great deal of detailed specification, although myGrid and OntoGrid have demonstrated the use of automatic data matching [94]. The work in [125] has described a distributed dataflow processing system under development that is designed to run on top of different architecture in an adaptive way. It processes data queries that involve

Figure 9.9: Federated Database System (source: [64])
user-defined operators. By using the ADFL( Athena Data Flow Language), these operators can be described in a ‘semantically, technically, and operationally domain agnostic manner’, which are specified in associated profiles. The system can optimise the queries at several level. The current optimisation in this work mainly focuses on the conventional CPU work parameters. The final goal of this work is to map high-level dataflow semantics to flexible runtime structures.

Technologies to facilitate data integration

IBM’s Information Integrator [9] is a software solution that supports federated databases. OGSA-DAI [98] is a web service based approach that at supports access to heterogeneous data resources, such as relational or XML databases and also provides a powerful set of data transformation, data merging and data delivery functionality. OGSA-DAI can be used to execute data integration workflows or can provide data federation functionality through its extension OGSA-DQP [37][39].

Other technologies that may support data integration include Transaction Process Monitors [54], Message Oriented Middleware [105] and Object Request Broker [127]. Many data integration middlewares build until standard technologies such as XML [31], SOAP [23], Web Services [29] and Service-Oriented Architecture (SOA) [53]. SQL technologies such as such as OLE DB and Java Database Connectivity (JDBC) database access driver can be used to provide a connection between an application and a database.

9.2.2 Semantic Data Integration

The semantic data integration is to solve semantic conflicts between heterogeneous data sources. The reason is that heterogeneous data source nodes at different sites may be captured by different tools, platforms and methodologies, and described in different languages, which causes difficulties to exchange information among different data nodes and have the meaning of that information automatically understood and interpreted among each other for discovering useful information or knowledge. Therefore, it is necessary to provide standardized ways to describe the meanings of many more things that are identical or can be accurately interconverted so that the use of data or context of applications can be understood and unambiguously defined. Ontologies are a widely accepted state-of-the-art knowledge representation, and have thus been identified as the fundamental building block for the Semantic Web. The extensive usage of ontologies allows semantically enhanced information processing and help to resolve semantic conflicts. Examples of semantic integration are from early multidatabase systems over mediator systems [70] [66] to ontology based approaches such as single-domain ontology [42] that map the contents of data sources a single ontology, multi-domain ontologies [108] [116] [33] that provides one or more ontologies for each data source so that query can be formulated based on user-selected ontologies, Semantic web [71][72][89] that supports semantic-based query handling for semi-structured information from the world wide web, Semantic Grid [71][72][89] that and user- specific semantic integration [129] that integrates data according to specific information needs without having to cope with low-level heterogeneity and technical details of underlying data sources.
Technologies to facilitate semantic data integration

Oracle Database 11g [33] has incorporated native RDF/RDFS/OWL support, enabling application developers to benefit from scalable, secure, integrated, efficient platform for semantic data management. This semantic database support is part of Oracle Spatial 11g, an option to Oracle Database. Application developers can add meaning to data and metadata by defining a set of terms and relationships between them. These sets of terms ("ontologies") enable enhanced query, analysis and actions based on semantic content, rather than simply data values. The current Oracle 11g semantic database can support:

- Storage, Loading, and DML(Data Manipulation Language) access to RDF/OWL data and ontologies; each model is an RDF/OWL graph consisting of directed labelled edges. The edge is labelled by a predicate and connects a subject node to an object node.
- Inference using OWL and RDFS semantics and also user-defined rules; Oracle database 11g include a native inference engine for efficient and scalable inferencing using major subsets of OWL.
- Querying of RDF/OWL data and ontologies using SPARQL-like graph patterns embedded in SQL.
- Ontology-assisted querying of enterprise (relational) data.

S-OGSA [72] is a reference Semantic Grid Architecture, extending the Open Grid Services Architecture, by explicitly defining a lightweight mechanism that allows for the use of semantics and defining the associated knowledge services to support a spectrum of service capabilities. S-OGSA has focused on three aspects: the model (the elements that it is composed of and its relationships), the capabilities (the services needed to deal with such components) and the mechanisms (the elements that will enable communication when deploying the architecture in an application). S-OGSA model introduces the notion of semantics into the model of the Grid defining Grid Entities, Knowledge Entities (e.g. ontologies, rules, text), Semantic bindings between these two for a Grid Entity to become Semantic Grid entities. Semantic bindings are metadata assertions on Grid entities and are Grid resources with their own identity, manageability features and metadata. S-OGSA also adds the set of capabilities that Grid middleware should provide the Semantic Provisioning Services and Semantically Aware Grid Services(Figure [9.10]). Semantic Provisioning Services can dynamically provision an application with semantic grid entities. The services support the creation, storage, update, removal and access of different forms of Knowledge Entities and Semantic Bindings. Semantically Aware Grid services exploit knowledge technologies to deliver their functionality, for example, metadata aware authentication of a given identity by a VO manager service or execution of a search request over entries in a semantically enhanced resource catalogue. S-OGSA implementation mechanism is based upon Globus Toolkit 4, grounding to the WS-RF (Web Services Resource Framework), incorporating S-OGSA entities into the Common Information Model(CIM) resource model.

OGSA-DAI-RDF [101] has extended OGSA-DAI access to RDF database system, e.g., Jena [11]. Several OGSA-DAI activities for handling RDF data and ontology are implemented. RDF handling activities include SPARQL [21] QueryStatement Activity, RDF Bulkload Activity, RDF ResourceManagement Activity, and RDF CollectionManagement Activity. The
Figure 9.10: The S-OGSA semantic provisioning services positioned in the OGSA services

SPARQL QueryStatement Activity receives the W3C SPARQL representation and gets the result in RDF XML format. The ResourceManagement Activity provides functions of creation and deletion to manipulate RDF statements. The CollectionManagement Activity supports functions, such as create, delete and list, operating on a RDF data repository. In addition to RDF activities, Ontology handling activity is also implemented. The activity is based on reasoning function of Jena and supports OWL and RDFS.

9.3 Discussion and issues

As we have seen, these existing distributed data mining systems have made much effort on the improvement of high performance data mining: from data mining algorithms, programming models to exclusive hardware platforms. They have their own strengths. Some of them [92] have targeted how to aggregate local models to global models with minimum communication cost; some of them have used dedicated computing resources and special communication protocols [87] or third party commodity platforms [79] to persistently store data in order to access it in place without moving data, while some of them relocate data at data warehouse layer [48]; Some of others have focused on finding the optimal parallelisation of a workload [109] by a high-level abstraction or data cache technology [119]. Some of them [80][81] have incorporated performance cost models into the architectures. It is also worth mentioning these systems do not support semantic nor consider privacy preserving data mining.
Chapter 10

Conclusion and Future Work

This document describes an iteration, AA3, of the ADMIRE Architecture. The previous report on architecture version AA2 clarified the goal and rationale of the ADMIRE architecture, provided the specification and definition of the structure and behaviour of the ADMIRE system and each component within the ADMIRE architecture. It also defined the boundaries between the architecture work and the work on language design (WP1), the execution platform (WP3/4), and the tools that make use of the architecture (WP5).

In this iteration of this document, we have presented a set of principles for evaluating the architecture based on different properties of typical data mining applications and reported the initial evaluation result of applying parallel processing model into a real DMI use case as shown in Chapter 8.

Future work can be divided into two types:

• short-term, i.e. six-months or 12 months;

• long-term, i.e. the lifecycle of the ADMIRE project.

10.1 Future Work: the Next Six Months

As discussed in section 8.4, the experimental results of the first evaluation of the ADMIRE architecture provide initial ideas on how to parallelise manually data and tasks on a real DMI use case. It has also introduced ideas and raised key questions regarding the design of the architecture, which in turn can be viewed as a to-do list for further exploration of the ADMIRE architecture. Therefore, our work in the next 6 to 12 months will focus on:

• how can the DMIL language help to describe a DMI request and then map the DMI request to physical resources, automatically splitting and executing the DMI task in parallel from a top-down view?

• What is the “right” categorisation granularity for PEs?

In the next 6 months we will refine the design of our current experiment described in Section 8.3. The current experiment is designed from the bottom to up and is still a little “rough and ready”. We manually partition a DMI workflow and run that workflow without using the DMIL language; this does not help us in realising the automation of composing and partitioning the workflow to meet ADMIRE’s primary goal. Our next step will thus be
to investigate how we might use the DMIL language to run and split a DMI workflow to support parallel execution. Specifically, how should we describe a DMI request using DMIL, how should we map this request onto physical resources and what information is needed in the Registry to allow this mapping? Additionally, the refinement of the experiment will also include parallelisation of PEs in the DMI EURExpress use case. For example, the PEs in the current experiment are partially implemented as parallel versions. Some of PEs still use sequential versions — for example KNN, image processing, etc. — which obviously influences efficient parallel execution of the DMI task.

Furthermore we will expand the DMIL description to the other use cases, such as flood prediction.

10.2 Future Work: Lifecycle of the ADMIRE Project

10.2.1 Security framework

The ADMIRE architecture will require a security infrastructure that controls access to services. Due to the nature of service orchestration, one service could call another service on behalf of the original user; thus a delegation model will be required in the architecture.

10.2.2 Outstanding issues

- Collect DMI Usage Patterns in Appendix A.

- How to store data — persistently stored using third party commodity storage service or common storage?

- How to partition, segment, cache, and move data? Large amounts of data movement should consider communication costs. Good data partitioning algorithms and cache technologies should be developed or borrowed from examples.

- How to parallelise workload and tasks?

- How to access distributed data dynamically?

- How to monitor performance — what is the performance cost model, and how best should we incorporate the performance model into the architecture?

- How to ensure the reusability and openness of data mining components so as to easily compose them for better data exploration?

- How to perform privacy-preserving data mining?
Appendix A

DMI Usage Patterns

We list a number of usage patterns that are common in the context of DMI.

A.1 n-Fold Cross Validation

To validate a classification task in a data mining exercise, it is common to make use of \( n \)-fold cross validation. This process randomly splits the set of input tuples into \( n \) distinct subsets of equal size (or as close to equal as possible). We then repeat the training phase of the classifier, i.e., the model-building phase, \( n \) times, where each time we use \( n-1 \) subsets to train on and then validate, i.e., test, the resulting model on the \( n \)-th subset. For each iteration we choose a different subset to do the testing. The accuracy of a model is then reported as the average accuracy calculated over the \( n \)-th tests. A common setting for \( n \) is 10 or 12 folds.

Below is the DMIL code for producing a distributed version of the n-Fold Cross Validation. Once compiled into an OGSA-DAI workflow it looks like the diagram in Figure A.1. This pattern naturally has much structure to it. We show this structure in Figure A.2.

```plaintext
function kFoldCrossValidationPattern(
  Integer k,
  PE(<Connection dataIn>, <Connection output>) BuildClassifier,
  PE(<Connection classifier; Connection dataIn> => <Connection score>) Evaluator)
PE(<Connection inputData, inputVar, outputVar>, <Connection score>)
{
  Connection inputVariables;
  Connection outputVariables;
  ListRandomSplit lrs = new ListRandomSplit(k);
  UnlimitedBuffer [] buffer = new UnlimitedBuffer[k];
  ListMerge [] listMerge = new ListMerge[k];
  TupleProjection [] projectInputVariables = new TupleProjection[k];
  TupleProjection [] projectOutputVariables = new TupleProjection[k];
  BuildClassifier [] buildClassifier = new BuildClassifier[k];
  Evaluator [] evaluator = new Evaluator[k];
  Classify [] classify = new Classify[k];
  TupleMaker [] tupleMaker = new TupleMaker[k];
  ListMaker listMaker = new ListMaker();
  for (Integer i = 0; i < k; i++)
  {
    for (Integer j = 0; j < k; j++)
    {
      if (i == j)
      {
        /* connect test set */
        ->dataOut[j] => dataIn->buffer[i];
      }
```
else
{
  /* connect training set */
  lrs->dataOut[j] = dataIn[j]->listMerge[i];
}

/* training phase */
listMerge[i]->dataOut = dataIn->buildClassifier[i];
inputVariables = columnIds->projectInputVariables[i];
buffer[i]->dataOut = dataIn->projectInputVariables[i];
buildClassifier[i]->classifier = classifier->classify[i];
projectInputVariables[i]->result = dataIn->classify[i];

/* classify */
projectInputVariables[i]->result = desiredClass->evaluator[i];
projectOutputVariables[i]->result = element[0]->tupleMaker[i];
evaluator[i]->score = element[1]->tupleMaker[i];
tupleMaker[i]->output = input[i]->listMaker;

/* testing phase */
classifier[i]->proposedClass = proposedClass->evaluator[i];
projectOutputVariables[i]->result = desiredClass->evaluator[i];
buildClassifier[i]->classifier = element[0]->tupleMaker[i];
evaluator[i]->score = element[1]->tupleMaker[i];
tupleMaker[i]->output = input[i]->listMaker;

/* form and return a PE comprising this DMI process subgraph */
return !deleted: new PE(
    <Connection> inputData = dataIn->lrs,
    Connection> inputVar = inputVariables,
    Connection> outputVar = outputVariables>,
    <Connection> score = listMaker->result >);
Figure A.1: n-Fold Cross Validation pattern compiled into an OGSA-DAI workflow
Figure A.2: Overview of the structure inside the n-Fold Cross Validation pattern
Bibliography

[4] EURExpress-II.
[23] SOAP. http://www.w3.org/tr/soap/.
[31] XML. http://www.w3.org/xml/.
[34] DADC ’08: Proceedings of the 2008 international workshop on Data-aware distributed computing (New York, NY, USA, 2008), ACM. Program Chair-Tevfik Kosar.


