



UNIVERSITEIT VAN AMSTERDAM

Problem Solving Environment for Medical Image Analysis Application Development

A thesis submitted to the Board of Examiners in partial fulfillment of the requirement of the degree of Master of Science in Grid Computing

Ketan C Maheshwari

Supervisors:

Silvia Olabarriaga and Adam Belloum

Contents

| | |
|-----------------------------------------------------------------|------------|
| Contents | iii |
| Acknowledgment | v |
| Abstract | vii |
| 1 Introduction | 1 |
| 1.1 MIA Application Development Lifecycle | 2 |
| 1.1.1 Stages in MIA-ADL | 2 |
| 1.1.2 Users | 2 |
| 1.1.3 Feedback during the MIA-ADL | 3 |
| 1.1.4 VL-e and Support Systems | 3 |
| 1.2 Motivation and Goals | 4 |
| 1.3 Generalization and Application to Other Domains | 4 |
| 1.4 Thesis Overview | 5 |
| 2 Problem Description and Requirements | 7 |
| 2.1 Introduction | 7 |
| 2.2 MIA-ADL Development Phases and Users | 7 |
| 2.2.1 MIA-ADL Development Phases | 7 |
| 2.2.2 MIA-ADL Users | 11 |
| 2.3 The VL-e Proof-of-Concept Environment (VL-e PoC) | 12 |
| 2.4 Support Systems | 12 |
| 2.4.1 DeVIDE | 12 |
| 2.4.2 Nimrod/G | 14 |
| 2.4.3 AMC-DWMS | 15 |
| 2.5 Requirements | 17 |
| 2.5.1 Functional Level Requirements | 17 |
| 2.5.2 Technical Level Requirements | 18 |
| 3 Literature Review | 19 |
| 3.1 Introduction | 19 |
| 3.2 Problem Solving Environments | 20 |
| 3.2.1 Workflow Management Systems | 23 |
| 3.2.2 Rapid Application Development (RAD) Environment | 27 |
| 3.2.3 Parameter Sweep Environment | 27 |

| | | |
|----------|--------------------------------------------------|-----------|
| 3.3 | Provenance Management | 29 |
| 3.4 | Discussion | 30 |
| 4 | Proposed Architecture | 33 |
| 4.1 | Introduction | 33 |
| 4.2 | Modeling Requirements | 33 |
| 4.2.1 | Concepts and Terminology | 33 |
| 4.2.2 | Search and Provenance | 36 |
| 4.2.3 | Other Requirements | 36 |
| 4.3 | The Data Model | 36 |
| 4.3.1 | Entities, Attributes and Relationships | 37 |
| 4.4 | PSE Architecture | 41 |
| 4.5 | Usage Scenario | 42 |
| 5 | Implementation | 45 |
| 5.1 | Introduction | 45 |
| 5.2 | Overview | 45 |
| 5.3 | Choice of Technology | 46 |
| 5.4 | API Classes Implementation | 46 |
| 5.5 | APIs Functions Implementation | 47 |
| 5.5.1 | Functional Specification | 47 |
| 5.6 | Grid-Service Implementation | 48 |
| 5.7 | Grid-Service Clients Implementation | 49 |
| 6 | Test Case | 51 |
| 6.1 | Introduction | 51 |
| 6.2 | Experimental Setup | 51 |
| 7 | Conclusions and Future Work | 57 |
| 7.1 | Introduction | 57 |
| 7.2 | Conclusions | 57 |
| 7.3 | Future Work | 59 |
| | Bibliography | 61 |

Acknowledgment

I am grateful to my supervisors Silvia D. Olabbarriaga and Adam Belloum for their invaluable guidance and support throughout the project. Thanks to our collaborators Charl Botha, Jeroen Snel, Johan Alkemade for their support and invaluable inputs during the project. I acknowledge gratefully the contribution of Piter T. de Boer in the programming and implementation part of this work.

Thanks to my family who has been always helpful and supportive. Thanks to friends, especially Dhiraj, Elise, Ruta, Joost, Thomas for always being helpful, motivating and inspiring. I acknowledge Stanley Jaddoe and Arthur van Dam for providing a beautiful L^AT_EX template for writing the thesis. I extend my gratitude towards the open-source developer's community for providing valuable software and supporting tools.

I would like to extend my gratitude to the wonderful Dutch people for providing me warm and friendly environment in a foreign land.

Abstract

The development of Medical Image Analysis (MIA) applications that can successfully be applied in clinical practice is difficult for several reasons, one of them being the large amount and variety of resources involved (people, data, methods, computing). The application goes through several phases (development, parameter optimization, evaluation and clinical deployment) usually supported by different systems. The lack of support for information flow from phase to phase puts extra logistics burden on the lifecycle of MIA applications. The present report describes efforts to develop a Problem Solving Environment (PSE) for MIA applications using the three systems available at the proof-of-concept environment of the Virtual Laboratory for e-Sciences project. The proposed PSE implements data provenance mechanisms that support information flow among systems, facilitating navigation across phases of the application lifecycle. A prototype implementation of PSE is tested on a small representative MIA test-case.

The thesis is partly published in and presented as “K. C. Maheshwari, S. D. Olabarriaga, C. P. Botha, J. Snel, J. Alkemade, and A. Belloum, “Problem solving environment for medical image analysis,” in CBMS 2007: IEEE Symposium on Computer Based Medical Systems, 2007, Maribor, Slovenia, 2007”

Introduction

The field of Medical Image Analysis (MIA) [1] enables knowledge extraction through computational post-processing of digital medical images. Processing of digital images using MIA modalities, greatly assist in disease monitoring, diagnosis and preoperative planning. With advanced medical image acquisition instruments, the resolutions and growing sizes of medical images. Handling the growing amount of medical image data and managing the related applications across a collaborative multi-department environment is difficult and tedious. Figure 1.1 shows a high level overview of a MIA application. Medical images from an image acquisition modality are sent to the MIA methods for enhancements and the resulting image is sent to the users' workstation for further analysis and diagnosis. PACS (Picture Archiving and Communication System) is a storage system at the hospital for image archival and distribution purposes. The box representing MIA in the figure is a high level view of a whole range of MIA applications that work in coordination to achieve a specific task. The viewing workstation can also be user's personal computer.

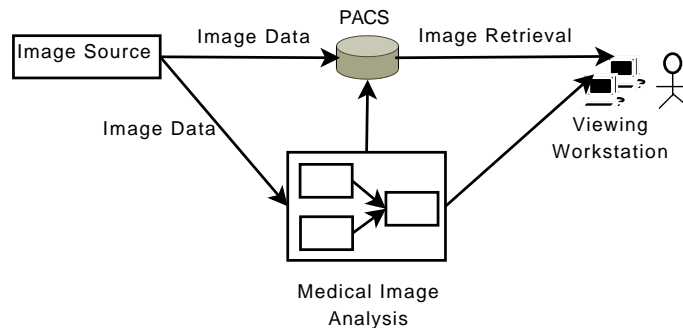


Figure 1.1: General Scheme of an Ideal Medical Image Analysis in a Clinical Setting illustrating a Medical Image Analysis Application, Picture Archiving and Communication System (PACS) and a Viewing Workstation

1.1 MIA Application Development Lifecycle

The development of successful MIA applications that can be used in a clinical setup is a difficult process. Beginning from the incubation of an application until a successful deployment into clinical setup forms a MIA Application Development Lifecycle (**MIA-ADL**). A successful completion of MIA-ADL requires a diverse set of skills and heterogeneous development and testing environments. This leads to a considerable amount of information exchange among people and stages during the lifecycle. A typical MIA-ADL requires information flow through these stages and collaborative work by the users involved may span over a considerable amount of time. One of the important aspect of the MIA-ADL is to keep track of the application executions and their results. This involves keeping track of a range of information including the application data, input and output parameters, and the image data. New versions of application may be developed and stored or an application might be tested on a different dataset during the feedback cycle.

1.1.1 Stages in MIA-ADL

A MIA application goes through a series of stages during its lifecycle before it matures to be usable into the clinical environment. Each stage has a different environment and requirements in terms of handling of data, user interaction (interactive vs. automated) and the user-expertise required. These stages involve development of application components, their assembly into a complete application, the testing of application using a range of parameters and images, and clinical deployment. Stages are linked to each other through information flow paths. The information that flows along these paths can be the applications, parameters and notification messages. A moderate to very high computational power is required that makes it essential to use the High Performance Computing (HPC) environment like the grid [2]. The inputs of the MIA applications can be large image files of the order of from a few megabytes to a few gigabytes. The image files are stored and retrieved to be processed upon at various stages. A conventional computing facility currently does not have such large storage capacities. This makes it necessary that handling of data among stages be done by a large scale data storage facility. The environment varies across stages from purely software development to clinical and the information flows across these environments.

1.1.2 Users

The above mentioned stages involve different departments, users and development environments. Users with a diverse set of skills are involved in the complete MIA-ADL. These users do different activities during different stages yet are connected as part of the development of a same MIA application. At the development stage, a technical set of skills are required while at the clinical deployment stage the clinical and medical domain skills are required. These users need to work in a collaborative environment and be communicated of the events taking place during the lifecycle. A “Development User” is responsible for the development of an application. This user may develop software components or assemble “off-the-shelf” components from a repository. A “Parameter Optimizer” user searches for the optimal parameters for the developed application. This activity is done in an automated fashion with some kind of support system that enables parameter sweep to find optimal parameters. An “Evaluation User” performs statistical evaluation for this application using a large number of test image data-set. Evaluation is also an automated and repetitive activity carried out using a support system that can perform computation over a large amount of data. A “Clinical User” performs the clinical deployment of the

developed application. Clinical deployment involves introducing the MIA application into the clinical workflow in order to run it on a regular basis.

1.1.3 Feedback during the MIA-ADL

MIA-ADL is an iterative process that may require to revert back to a previous stage in the case of failure of a stage to meet its development requirements and evolution of imaging modality or protocol. The information flows backwards from any given stage to a previous stage. This process of feedback enables the developers and users involved to revert to a previous stage in the cycle and correct and refine the problems. These iterations can continue for any number of times required. For example a user should be able to reconstruct a past application execution that is to “rerun” a previously run application with the same set of parameters, recreating the environment of a previously executed application. This could be done by “fetching” the information pertaining to a run from the system to a local computer. The purpose of rerun is to return to a previous stage of the MIA-ADL in order to detect and correct flaws or compare the performance of a previous execution with a current one.

1.1.4 VL-e and Support Systems

The current work is carried out as a part of the VL-e project at the Informatics Institute of the University enabling a host of support systems and HPC infrastructure available readily.

VL-e The Virtual Laboratory for e-Science¹ (VL-e) is a Dutch project started in 2004 with the mission of boosting e-Science by creating an e-Science environment and carrying out research on methodologies [3]. The project is carried out by a consortium of academic and industrial partners, being organized into subprograms that address different components of an e-Science environment: Applications, Virtual Laboratory (VL) methodologies and Infrastructure. The vision is at the same time application- and technology-driven [4]. On the one hand, the Applications subprograms (six) try to design and implement various problem solving environments (PSEs) for different domains based on the e- Science paradigm. On the other hand, the VL subprograms (six) try to develop new methodology to be used as generic components of the various PSEs under construction. The interaction among (subsets of) subprograms takes place in the context of application-driven use cases, in a set-up that stimulates and facilitates multidisciplinary research. All activities share resources provided by a generic e-Science infrastructure consisting of computing clusters, data storage, and services hosted in distributed locations across The Netherlands. The infrastructure is organized into two types of environments for Rapid Prototyping (RP) and for Proof-of-Concept (PoC). The RP environment is provided by the Distributed ASCI Supercomputer 32 (DAS-3), and mainly used for experimentation on VL methodology. In the PoC environment³, the different tools and services used by and developed in the project are available, being bound together in a service-oriented approach. The PoC is the main platform for development and implementation of a variety of application-specific PSEs.

Support Systems A MIA application is developed, tested and deployed with the help of support systems. These support systems have a rich set of domain specific functionality and specialized for a specific area such as image processing or parameter sweep applications. Adapting the usage of available infrastructure makes the tasks at different stages easier and at times automated. Each supporting system has its own domain of specialty and complement each other during the

¹<http://www.vl-e.nl>

lifecycle. As much as these systems are useful at certain stages during the MIA-ADL they also exchange information and are required to work in an integrated manner with each other.

1.2 Motivation and Goals

A complete MIA-ADL is a data and compute intensive process with diverse requirements at each stage. To fully support MIA-ADL multiple software systems are needed. The feedback in the MIA-ADL cycle and its long time span calls for a need for information organization and management. This in turn also accelerates the adaptation of new MIA techniques and tools. This work is motivated by the Medical Image Analysis lifecycle development model, as proposed in the article titled “Integrated Support for Medical Image Analysis Methods: from Development to Clinical Application” by Olabbarriaga et. al. in [5]. The goals of this work are as follows:

- **Problem Solving Environment:** To propose and design a “Problem Solving Environment” (**PSE**) architecture in order to manage the MIA-ADL elements, capture and retrieve the information flowing across the MIA-ADL. The proposed architecture should interface to the existing infrastructure including the support systems. The users should interface the system in batch as well as interactive mode. The architecture must provide for batch and interactive user modes as required. Additionally it must provide mechanisms to use the large scale storage and the grid infrastructure that is available within the VL-e project.
- **Structuring and Modeling:** The elements involved in the MIA-ADL are spread across stages, users and loosely structured. They need to be structured and grouped into meaningful components. This will help capture, store and retrieve efficiently the relevant information from the system. Structuring the information makes it easier to retrieve and maintain over a long time period.

The MIA-ADL elements need to be modeled to bring up the relations among them and to maintain the information integrity.

- **Enabling Support Systems:** One of the main goals of this work is to bring together the complementary support systems together and enable a problem solving environment in which all these systems can work together. The VL-e infrastructure [3, 4] acts as a common platform that provides an access and facilitates the usage of these support systems.
- **Implementation:** To implement and test the proposed architecture for the PSE. The implementation will serve as a proof-of-concept for the MIA-ADL model.

Considering the above points our goal could be stated concisely as follows: *Propose, implement and test an extensible Problem Solving Environment architecture in order to control and manage the elements of MIA-ADL.*

1.3 Generalization and Application to Other Domains

The MIA-ADL shows several characteristics that can be found in other e-science domains:

- Requirement of a heterogeneous mix of support software tools for different parts (stages in this case) of the same problem.
- A diverse range of human skills and roles.
- Requirements for the usage of large scale data storage and HPC/Grid support.

- Information flow from one part of the problem to other parts.

The MIA-ADL acts as an example application case for the proposed PSE. The goal is to design and develop a simple and generic architecture. This means the same PSE can be applied to other e-science domains with minimum modifications. The characteristics of the PSE are suitable for applications that require a diverse range of supporting software tools and human skills distributed over different departments.

1.4 Thesis Overview

The present chapter gives an introductory view to the problem and the goals of this work. Chapter 2 presents a detailed description and analysis of the problem and describes the relevant resources available within the VL-e project. Chapter 3 presents a study of related literature. Chapter 4 presents the requirements and proposed architecture and description. Chapter 5 presents the implementation of the proposed architecture. Chapter 7 wraps up the thesis with conclusion and future work.

Problem Description and Requirements

2.1 Introduction

This chapter elaborates on the description of the problem introduced in chapter 1. A detailed discussion of the complete Medical Image Analysis Application Development Lifecycle (MIA-ADL) is presented. This follows by a description of the available VL-e infrastructure highlighting the support systems that play important role during the MIA-ADL. The chapter concludes with a set of requirements are elicited at a high level of abstraction.

2.2 MIA-ADL Development Phases and Users

This section presents a detailed description of the MIA-ADL development phases and users.

2.2.1 MIA-ADL Development Phases

During the MIA-ADL an application evolves through stages. We term these stages formally as *MIA-ADL phases* or *phases* for short. These phases are termed as (*development*, *parameter optimization*, *evaluation* and *clinical deployment*) according to their functionality. Each phase is described in detail with its characteristic features in the following paragraphs of this section. Figure 2.1 depicts the phases of MIA-ADL with involved users and the information flow. The MIA-ADL starts with the development of components and ends with a successful clinical deployment of a MIA application. The arrows from left to right indicates the forward information flow from one phase to the next. During any phase the control can shift backwards to a previous phase in the form of feedback. This is shown in the figure using arrows from right to the left.

Component Development A new MIA application component is developed in this phase. MIA application components are developed using support systems through a component based development paradigm. This could be a new basic algorithm or a combination of algorithms to perform a series of tasks. A component has input and output *ports* that interface to other components. These components may be assembled into a network of components through ports in order to form a MIA application. As figure 2.2 shows the component accepts the data as input and performs processing, generating and transferring data to the output. Component Development phase acts as a prototype formulation for some specific MIA method. This phase demands

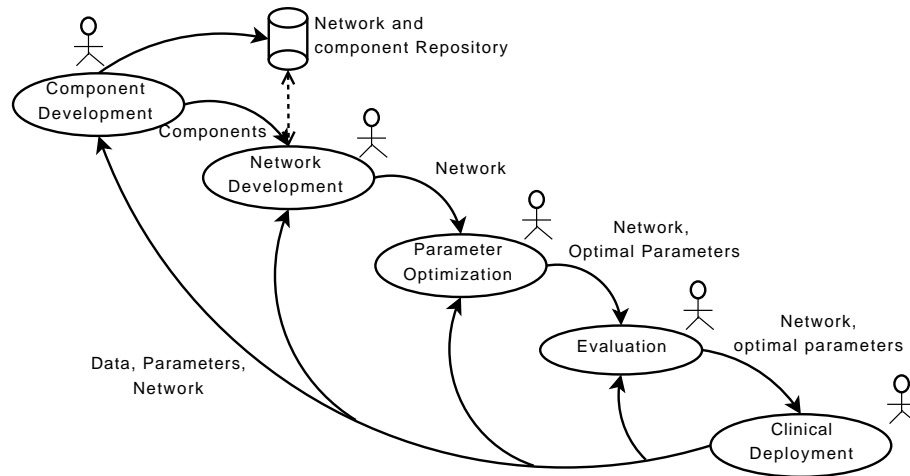


Figure 2.1: Information flow among phases of the MIA-ADL, from development to clinical deployment showing the application repository and users at various phases

technical specialization in the area of scientific programming. A manual and interactive testing using a limited number of parameters is done until desired results are obtained. The component is tested individually (independent of the application) for correctness during this phase. The components developed at this stage must provide interfaces so as to be able connect to other components. The developed component is stored into the applications repository. The applications repository stores the applications and components (see figure 2.1). The amount of computation and data required is limited at this stage. An integrated development environment dominates the component development phase.

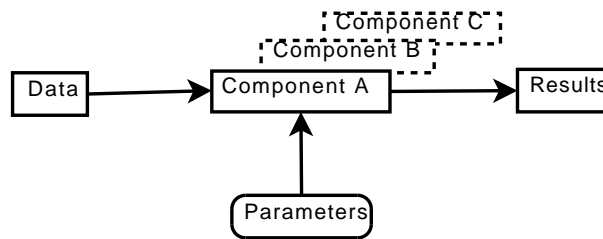


Figure 2.2: The Component Development phase showing the input data, parameters and output results

Network Development The Network Development phase is performed. A processing network is constructed by combining the components or existing networks developed previously using a legacy development tool. As shown in figure 2.3, the key activity in this phase is to combine components in order to form a network that performs the task. Appropriate components are identified and combined using their interfaces to form a network. This network is tested for correctness with a limited number of input images. The process of network development is an interactive task and the debugging and testing is done manually. A satisfactorily developed network is stored into the repository as a potential MIA application. The phases component and network development concludes with a preliminary version of a MIA application to be evaluated

in the further phases.

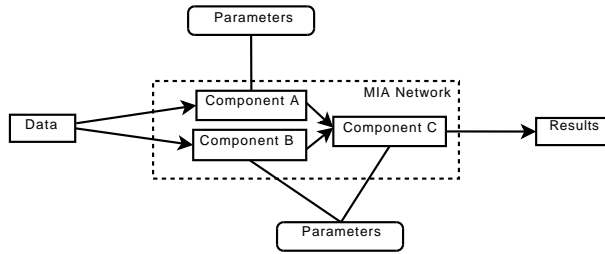


Figure 2.3: The Network Development phase showing a MIA network with components, parameters, input and results

Parameter Optimization The optimal parameters for the MIA application developed in the previous phase are identified in this phase. As shown in figure 2.4 an automated parameter sweep operation is setup that executes the MIA application (developed in previous phase) against a large number and combinations of parameters. The results are compared against an available benchmark resultset. It is not practical to take into account all the results and some of them may be selectively given less attention in order to speed up the task. parameter optimization is performed through parameter sweep. This phase is largely automated because of the large number of parameters that needs to be taken into account. However the setup of the parameter sweep, planning of the evaluation criteria and managing the storage of the results are performed manually.

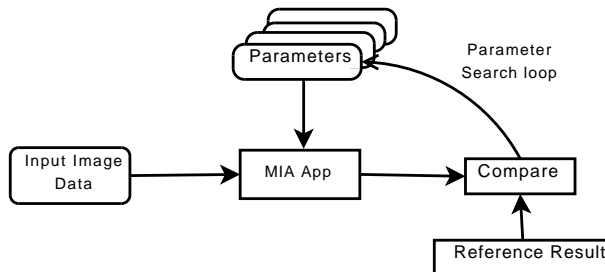


Figure 2.4: The Parameter Optimization phase showing a MIA application executing against a range of parameters (parameter sweep) and comparing results against a reference benchmark

Evaluation As shown in figure 2.5 the MIA application is evaluated against a large number of images in this phase. An image sweep is setup wherein the MIA application is run with optimal parameters as supplied by the parameter optimization phase. The application is evaluated against a number of qualitative and quantitative criteria. These include patient comfort, cost and time among others. The parameter optimization and evaluation phases are characterized by their large scale computational and storage requirements. A distributed resource usage becomes necessary in order to perform computations and storage at this scale.

Clinical Deployment In this final phase of the MIA application, a thoroughly evaluated and approved MIA application is introduced for practice into the clinical routine. The new MIA

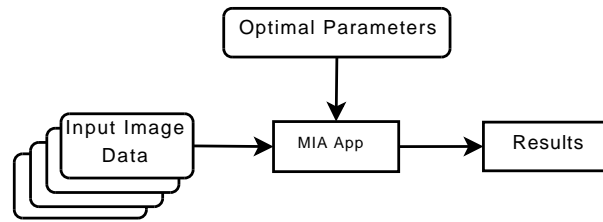


Figure 2.5: The Evaluation phase involving evaluation of the MIA application against a large number of input images (image sweep)

method becomes a part of the existing clinical environment. As shown in figure 2.6, the MIA application performs its tasks and the results are archived to the hospital's archival system. The users are notified of uncommon events during the execution. The developed component is used as a part of workflow within the clinical practice to support the daily clinical tasks. A Clinical User performs the clinical routine phase. A purely clinical environment of the hospital dominates this phase. Changes in the clinical environment or requirements may trigger further refinement of the MIA application. These changes may be brought about by upgrading of instruments, change in protocols or refinements and error corrections in the clinical workflow.

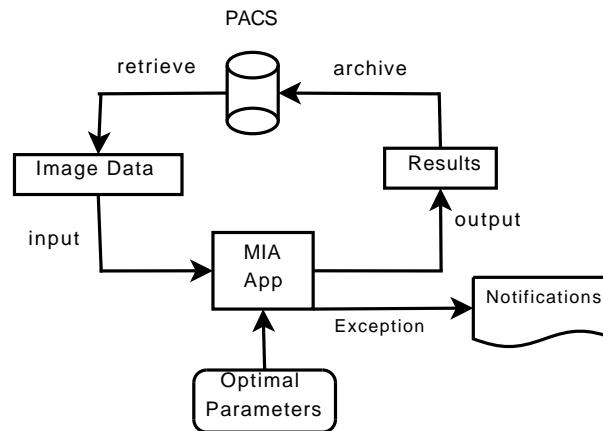


Figure 2.6: The Clinical Routine phase where in a MIA application is adopted as part of daily clinical practice; users are notified of exceptions

Navigation Between Phases The aforementioned phases are part of an evolution process that results in a successful MIA application. During this process, the focus may shift from one phase to other in forward as well as the backward direction (Figure 2.1). This backward navigation is required for error correction, feedback and quality refinement purposes for a particular phase during the MIA-ADL. The purpose of feedback is to enhance and tune the Developed Component in order to suit the application requirements. The MIA-ADL is a continuous lifecycle because of changing requirements and other developments such as new image acquisition and processing modalities. As a result of these changes new versions of the same MIA applications might be introduced or a new set of optimal parameters might be found. This motivates the need to enable the information flow not only in the forward but in backward direction as well. Information

generated during previous iterations of MIA-ADL must be stored in a searchable and retrievable repository. A user must be able to comfortably revert back to a previous phase in order to correct the problem. During the first iteration, the MIA application might need to revert back to a previous phase from any of the phases because of lack of conformance to the requirements for that phase. However, it can also revert back to a previous stage after a completed cycle because of unforeseen circumstances. These could be simple as fine tuning the application or as complex as upgrading the application to adapt to a new or a modified set of requirements.

2.2.2 MIA-ADL Users

A diverse set of users and skills are involved with the MIA-ADL, that influences the usability of the environment. User expertise shifts along the phases of MIA-ADL from purely technical to purely clinical background. Figure 2.7 illustrates the users and associated phase through a UML use-case diagram. The figure shows a dependency among various phases and the users involved with each phase. The user *Component Developer* develops application components. A Component Developer is usually a scientific programmer. The user *Network Developer* assembles components into a MIA Application that performs the intended MIA task. A Network Developer has an expertise in programming as well as in the area of image processing. A Component Developer may play the role of Network Developer as the expertise domains of both these users overlap. *Parameter Optimizer* finds the optimal parameters associated with the MIA application for a particular problem at hand. A Parameter Optimizer has a mixed expertise with programming and image processing. An *Evaluation User* performs the statistical and qualitative evaluation of the MIA application. For an Evaluation User the expertise shifts more towards the clinical field and less on the technical field. A *Clinical User* is associated with the clinical deployment phase and is the one who uses the MIA application in daily practice. A Clinical User has purely clinical expertise. The dashed arrow between phases denote a sequential and chronological dependency among the MIA-ADL phases. To simplify, all the users are referred to as ‘user’ when not mentioned otherwise.

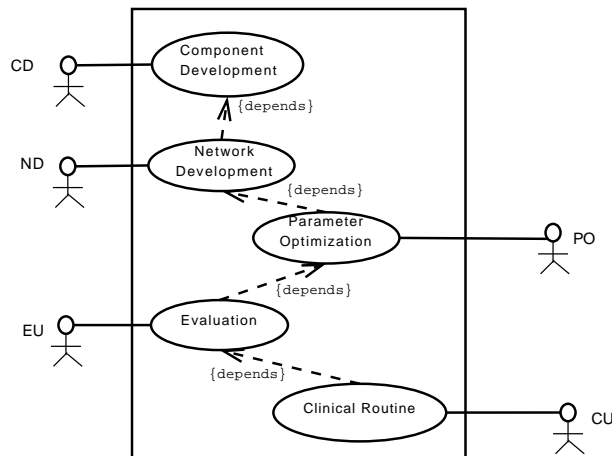


Figure 2.7: A UML use-case for MIA-ADL depicting the users associated with the phases and the dependency among the phases, CD=Component Developer, ND=Network Developer, PO=Parameter Optimizer, EU=Evaluation User, CU=Clinical User

2.3 The VL-e Proof-of-Concept Environment (VL-e PoC)

As a part of this work, it is desired to leverage the available infrastructure within the Virtual Laboratory for e-science (VL-e project). This chapter describes the available infrastructure and their working in brief.

The VL-e Proof of Concept (VL-e PoC) environment is a common, shared hardware and software support platform within the VL-e project. The VL-e PoC provides the physical infrastructure (storage, computing), services and systems shared by different application areas. The VL-e PoC resources are currently hosted by the SARA Computing and Networking Services and the Dutch Institute for Nuclear Physics and High Energy Physics (NIKHEF) at Amsterdam. Currently they consist of computational grid resources, a Storage Resource Broker (SRB, www.sdsc.edu/srb), and a large number of services, including the systems adopted for MIA.

The Storage Resource Broker (SRB) The Storage Resource Broker (SRB) [6] is the datagrid middleware storage facility hosted by the SARA Computing and Networking Services at Amsterdam. SRB provides an easy-to-use web-based as well as the UNIX-like commandline tools. Large scale data transfers can be done programmatically using java API provided by jargon. Globus provides a grid interface to SRB through the SRB-DSI using GridFTP for data transfer between grid nodes and the SRB.

2.4 Support Systems

This section describes the legacy systems used for this work: DeVIDE for the development, Nimrod/G for optimization (parameter sweeps) and evaluation (image sweeps), and the AMC-DWMS for clinical deployment.

2.4.1 DeVIDE

DeVIDE or the Delft Visualization and Image Processing Development Environment is a platform independent software environment that facilitates for the rapid creation, testing and application of modular image processing and visualization algorithm implementations [7]. DeVIDE provides, what is called a modular development environment, wherein, the modules provide a visual input/output and connectivity modality, through which, a pipeline or 'network' could be developed. This network accepts an image and optionally several parameters and performs the series of operations as specified by the network modules. Every module in DeVIDE has its own 'properties' that control its runtime behavior. DeVIDE facilitates to interactively alter the values of module properties. The network hence developed could also be used as a new third-party module by another application. A GUI for visualizing and interacting with the processed image data makes the experimentation with alternative algorithms faster and effective.

Architecture Figure 2.8 shows a high level view of the DeVIDE architecture. A DeVIDE module implements the algorithmic functionality and makes it available to the DeVIDE environment. This functionality is exposed to the rest of the system through the module API. The module API provides and enables the input and output ports on the modules that form an external interface for a module to connect to other modules. The module library is a collection of functions that can be used by programmers to integrate a new module into the API. For instance, a VTK class can be used to be wrapped into the module through available wrapper behavior in the library. The module manager controls the modules and associates resources. The module

manager provides necessary functionality including cataloging, instantiation, execution, monitoring and serialization/deserialization. The external libraries contain auxiliary tools that are accessible to the modules. These modules can use the functionality provided by these tools. Currently, DeVIDE includes VTK (Visualization ToolKit), ITK (Image processing ToolKit) and wxWidgets/wxPython support.

The Visualization ToolKit (VTK)[8] is an open source, freely available software system for 3D computer graphics, image processing, and visualization. VTK consists of a C++ class library, and several interpreted interface layers including Tcl/Tk, Java, and Python. The National Library of Medicine Insight Segmentation and Registration Toolkit (ITK)[9] is an open-source software system for medical image processing in two, three and more dimensions. Both VTK and ITK are application libraries, and provide a potentially powerful set of functionality through wrapped usage from the Python environment. DeVIDE integrates all functionality in both of these libraries and so makes all this functionality available through a data-flow application builder model. It is possible to experiment with any combination of VTK and ITK elements at runtime. Effectively, one can experiment not only with parameters and input data, but also with different code paths.

The graph editor is a client application for the module manager and provides a flexible user interface in order to create, configure and connect the modules. In the situations where a simpler interface is required to make a subset of functionality available to the user, the mini app extension of the graph editor is used.

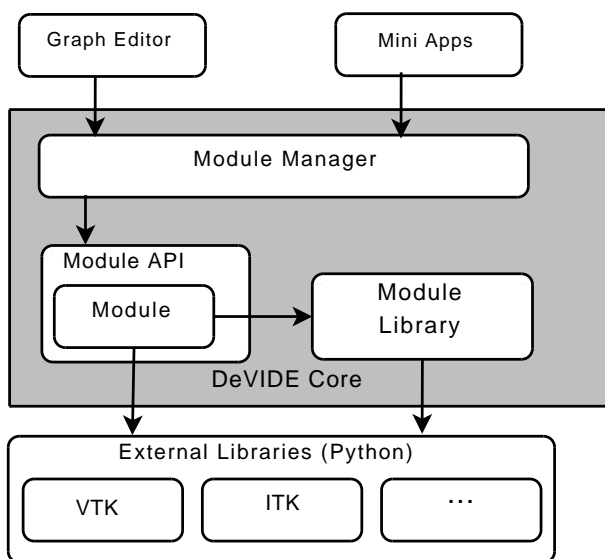


Figure 2.8: A high level view of the Delft Visualization and Image processing Development Environment (DeVIDE) Architecture showing DeVIDE core key components and external libraries ref. [7]

DeVIDE User Interface Figure 2.9 shows a snapshot of the DeVIDE user interface. Shown in the figure is a visual interface to a simple DeVIDE network consisting of three modules. A DicomRDR module provides the image slices from the Dicom data files it reads as input. DicomRDR sends the output to the resampleImage module where the sampling rate could be adjusted by modifying the spatial parameters. Finally, the resampled output data goes as input to the slice3DViewer

module. A user can activate/execute the network and visualize the image through slice2DViewer. Figure 2.9 shows a snapshot of the image as visualized through the slice3DViewer module. The interface also provides an interactive facility using which a user can navigate through the image in four dimensions using standard I/O devices such as a mouse and keyboard.

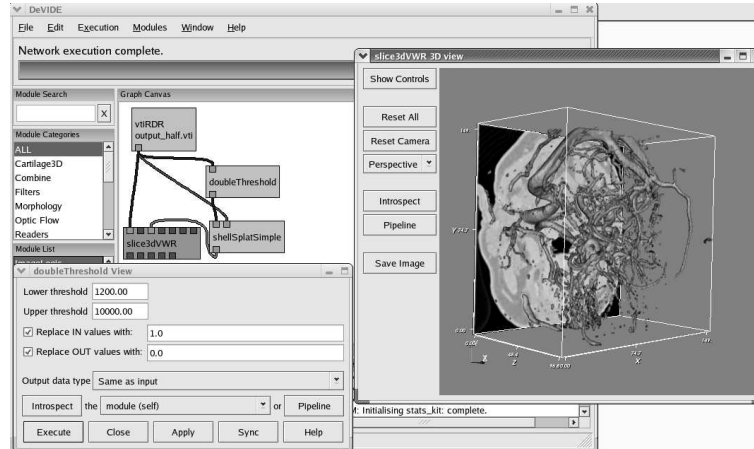


Figure 2.9: DeVIDE network in interactive mode during the development phase.

2.4.2 Nimrod/G

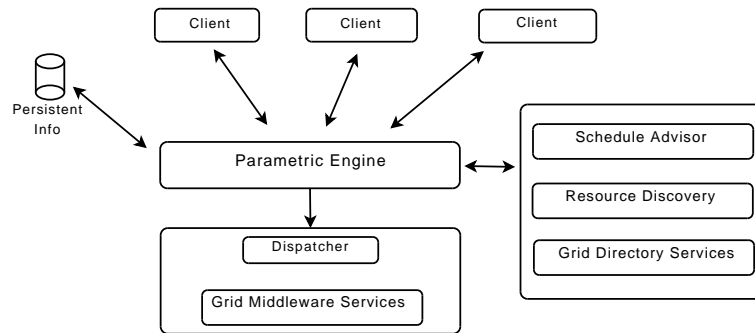


Figure 2.10: A simplified Nimrod/G Architecture showing interaction among client, parametric engine, scheduler and persistent information manager

Nimrod/G [10] is a high throughput computing environment that performs job management and resource scheduling based on a cost and time optimization strategy. It provides a queue management mechanism; a job submission system to communicate with other resources, and an integration facility with resource management entity. In this way the resources are chosen from a resource pool for individual jobs hence the jobfarming. The job is wrapped in a software container called nimrod agent that contains job scheduling and dispatch specific information. Therefore, one agent per node per job makes all job independent of each other.

Architecture Figure 2.10 presents a summarized outline of the Nimrod/G architecture. A Parametric Engine performs the parameter sweep calculation in consultation with the schedule adviser component. The Schedule Adviser provides with the resource availability information through querying the Resource Discovery and the Grid Directory Services. The Persistent Information component is an information repository in the form of relational database in order to store and retrieve the information about status of job executions and their resource utilization. The parametric engine passes the job submission instructions to the dispatcher and the dispatcher, in turn dispatches the jobs on the grid environment using the grid middleware services. Nimrod/G

```

parameter THRESH integer range \
from 1200 to 1600 step 1;

task main
  copy ./rundevide.sh node:.
  copy ./dev_nw.dvn node:.
  copy ./img_in.vti node:.

  node: execute ./rundevide.sh $THRESH img_in.vti \
  ./test_network.dvn img_out.vti

  copy node: stdout stdout.${THRESH}
  copy node: stderr stderr.${THRESH}
  copy node: dev_nw.dvn dev_nw${THRESH}.dvn
  copy node: img_out.vti img_out${THRESH}.vti
endtask

```

Figure 2.11: Example of a Nimrod/G plan file showing the staging-in, execution and staging-out

performs the job submission and execution activities with the help of a batch-like plan file submitted to its parametric engine (see figure 2.11). It sends out a number of parameter sweep calculations at the start of the execution of the tasks. Next, these calculations are executed in parallel on available resources. When finished, the calculation results are collected back from the resources and sent back to the client. During the execution of these tasks, the client computer only needs to dispatch and collect the jobs to and from alternative nodes in accordance with the commands on a plan file. A sample plan file is shown in figure 2.11. Here a shell script called ‘rundevide.sh’ is executed across a range of 1200 to 1600 for 400 parameter values hence forming 400 independent jobs. In job farming, each node and, therefore, each analysis runs independent of all other nodes. This method is particularly suited to examining large parameter spaces.

2.4.3 AMC-DWMS

The AMC-DWMS (Academic Medical Center Distributed Workflow Management System) [11] is a software framework developed at AMC for the deployment of the custom MIA applications in a clinical setting. The system provides logistics facilities to support various clinical activities. Based on a Service Oriented Architecture (SOA), AMC-DWMS uses loosely coupled modules specialized to perform a specific task. A workflow is specified using a specialized XML vocabulary as shown in figure 2.13. These ‘templates’ are composed of logical units that correspond to various activities within the workflow. The system supports image import-export, caching, analysis and notification services among others. A relational Database is used to store and retrieve the information related to workflow actions. A semi-graphical interface, called ‘control-center’ is used to install, configure, update, synchronize and monitor module activities. The system is specialized in handling DICOM (Digital Image Communication) server data, a proprietary image format and protocol.

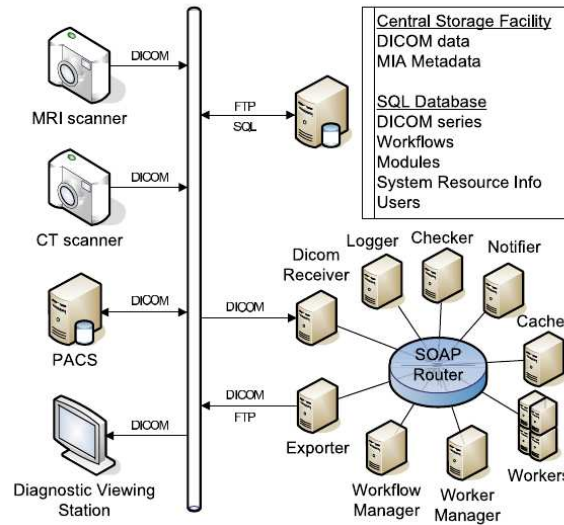


Figure 2.12: The Academic Medical Center Distributed Workflow Management System (AMC-DWMS) embedded in the hospital network including imaging devices, archives and viewing stations, ref. [11]

AMC-DWMS Environment Figure 2.12 depicts a simplified hospital network with embedded AMC-DWMS. The AMC-DWMS is a SOA based workflow manager composed of loosely couples software components. The modules support several generic functionality eg. call to operating system services, multi-threading and relational database connectivity. The communication between these modules is based on SOAP message exchanges. Important modules are the DICOM Receiver, Workflow Manager and Worker manager. The DICOM receiver receives DICOM images from the storage and/or scanning devices. The Workflow Manager instantiates the workflow and instructs the Worker Manager to select a Worker that meets the specific requirements on the available resources to perform the target task. After the task is executed, the results are sent back to the viewing workstation/PACS archive via FTP/DICOM protocols. A relational database is used to store the workflows, image data and MIA metadata.

```
<unit xsi:type="xwf:DicomSeries" description="CTA Blanco">
  <dicomTagCondition>
    <tag tagId="0" description="Acquisition Protocol">
      <group>24</group><element>4144</element>
      <regEx>MMBE | CTA | CAROTIDEN | WILLIS</regEx>
    </tag>
    <tag tagId="1" description="Manufacturer Model">
      <group>8</group><element>4240</element>
      <regEx>MX8000</regEx>
    </tag>
    <tag tagId="2" description="Contrast Bolus Agent">
      <group>24</group><element>16</element>
      <regEx>CONTRAST</regEx>
    </tag>
    <boolEx>tag(0) & tag(1) & !tag(2)</boolEx>
  </dicomTagCondition>
</unit>
```

Figure 2.13: Example of an XML used as workflow specification with AMC-DWMS

2.5 Requirements

This section presents an overview of the requirements that are imposed in order to control and manage all elements involved in a MIA-ADL. These requirements are elicited from a higher level of abstraction and derived from the description of MIA-ADL above. The requirements are classified as the functional level and technical level. Functional level requirements are those that needs to be fulfilled because of the inherent nature and working of the system, the MIA-ADL in this case. Technical level requirements are those that are imposed on the implementation of the system. These requirements are necessary to be met for a correct usage of the system.

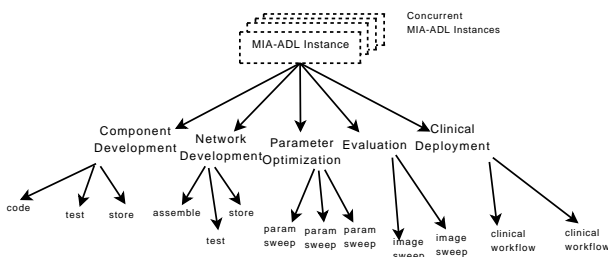


Figure 2.14: A hierarchical scheme showing the association of activities with phases corresponding to concurrent MIA-ADL instances

2.5.1 Functional Level Requirements

This section presents the requirements elicited at the functional level. These requirements will dictate the functionality of the proposed environment.

1. **User Collaboration:** MIA-ADL is a collaborative process with users involved belonging to different organizations. The users with clinical and technical backgrounds must be able to use the environment with easy to use interfaces. The interface must provide for a batch as well as interactive mode of user interaction. These users need to be informed about the events that take place during the MIA-ADL. These events are related to the normal or exceptional conditions occurring during the lifecycle. A mechanism to notify these users of events based on their participation and interest is desired.
2. **Structure:** In a research and development environment, work on several independent problems are initiated and handled concurrently over a period of time. This leads to the presence of independent MIA-ADL instances concurrently as suggested by figure 2.14. Hence it becomes practical to structure and group information related to all elements associated with each problem separately. At a low level of abstraction, information related to each phase of MIA-ADL also requires structuring.
3. **Application Management:** The MIA applications that evolve during the MIA-ADL leads to the creation of new versions of applications. In a development environment these applications and their versions coexist and are required to be preserved and stored in the system. These applications have their own call signatures consisting of a number of parameters and execution options. This makes it necessary to associate types and numbers of parameters with these applications and their versions. The compatibility among versions needs to be traced.

4. **Enabling Feedback:** The MIA applications are executed as a part of all phases. These executions and their results are evaluated during the lifecycle. A developer might want to re-evaluate and/or review the results at a future time. This makes it necessary to keep record of all executions of MIA applications performed during the MIA-ADL. These executions need large amount of processing and storage resources to process and store a number of large medical images. An HPC and large scale storage facility is required to handle such executions.
5. **Provenance:** Feedback in the MIA-ADL means that the origin and generation of data during the MIA-ADL needs to be traced and retrieved. For instance, from a given result image, it should be possible to trace the executions through which it has been produced.
6. **Events and Notifications:** MIA-ADL is a collaborative process with more than one users belonging to different departments being participants. These users might be interested in a common event occurring during one or more phases. A notification mechanism checks the events occur ed at any phase and notifies via email to the users “subscribed” to that event.
7. **Security:** The field of medical image analysis emphasizes the need for a secure access to all the information associated with it. This makes it important to have proper user authentication mechanisms for any supporting environment that interacts with the MIA applications and images. Especially in a distributed environment like the grid, provision of secure access to users via authentication is a high priority requirement.

2.5.2 Technical Level Requirements

1. **Distributed Resource Usage:** During the MIA-ADL, especially at the optimization and evaluation phases, a large amount of computational power is required. Such computational power can not be provided with a limited number of localized resources at the hospital and needs an extended computational environment. Using grid computing infrastructure available within the VL-e PoC gives a readily available HPC resources for the MIA-ADL.
2. **User Interface:** Depending upon the requirement of the application a batch-mode or interactive user interface must be provided. Functionality that require inputs at intermediate stages (eg. creation of a new session) from users must be implemented as interactive interface whereas those requiring a one-off inputs (eg. recording a run) must be batch mode.
3. **External Systems Interfaces:** The MIA-ADL involves usage of several support systems at various phases during its lifecycle. It is important for providing appropriate interfaces so as to accommodate these legacy systems and potentially unknown new support systems.
4. **Standards:** In order to make the proposed environment consistent with the existing and ongoing projects within VL-e, the standards must be followed. While designing the system, a standard methodology for representation and communication should be used. While implementation, complying technical standards for security, communication and storage protocols, database creation and grid services must be used.
5. **User Authentication and Delegation:** In a user collaborative grid environment, proper user authentication mechanism must be enabled. Services that require users to delegate their credentials must be able to use appropriate credential delegation mechanisms.

Literature Review

3.1 Introduction

As more science is done with the help of HPC infrastructure (e-science or enhanced Science), the complexity of experiments and the number of required computing tools has increased. These include hardware tools like storage and network devices, instruments and their adapters as well as software tools like Problem Solving Environments(PSE), Workflow Management Systems(WfMS) and an integrated framework that enables the utilization of software and distributed infrastructure in a transparent fashion. This chapter presents a literature review of various software tools and techniques involved in doing e-science.

Motivation and Scope This literature review is motivated by the problem requirements as mentioned in chapter 2. The envisioned problem solving environment requires a study of tools, techniques and concepts associated with existing systems. Figure 3.1 shows a partial classifica-

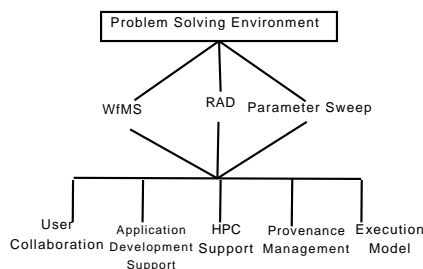


Figure 3.1: A conceptual classification of the Problem Solving Environments with Workflow Management Systems (WfMS), Rapid Application Development (RAD) environment and Parameter Sweep Applications at top level followed by their characteristic properties at the lowest level

tion and properties of a PSE. We propose this hierarchy keeping in mind the requirements for our system. The classification at first and second level is not exhaustive and may include more items. However, we only consider the relevant features here. The first level presents the systems that fall under the hood of PSE. The second level presents the characteristic features present in

a typical PSE for e-science. User collaboration has increasingly become evident and relevant in large-scale scientific applications [12]. Domain scientists form what is known as virtual organizations in order to collaborate to achieve short or long term goals. Execution Model or Execution Specification describes how the system in question allows user to specify and represent the executable portion of an application. The scope of literature review includes the study of concepts and systems that relate to the classification as shown in figure 3.1. The selection of systems and environments for the literature review was motivated by their prominence in the particular category and by the investigation of possibilities for integration and interoperability of several systems under the hood of a single PSE. Environments from other domains such as chemistry are chosen in order to identify the components and mechanisms generic to these systems. Provenance management, being a vast area in its own right, is treated as a separate section for literature review.

3.2 Problem Solving Environments

In this section we discuss a generic architecture for Problem Solving Environments (PSE) and investigate *Extensible Computational Chemistry Environment (Ecce)* [13], Triana [14] and Lab-Grid [15] PSE. These systems are chosen because of their prominence, diversity of application and popularity.

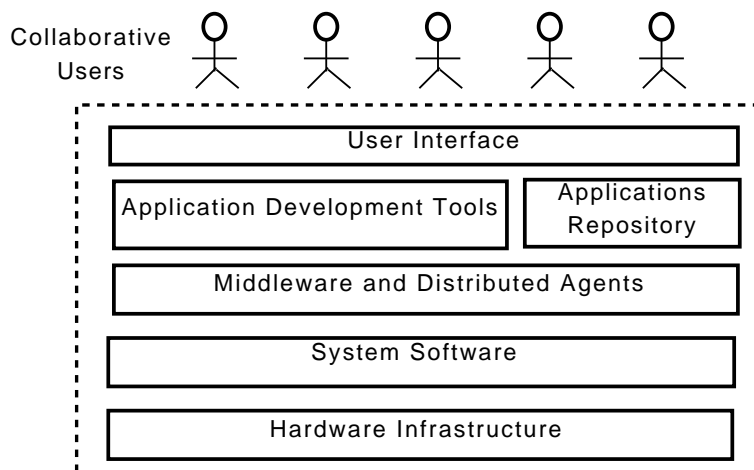


Figure 3.2: A general PSE model

Architecture There is no single architecture for a PSE. The components that make up a PSE depends upon the range of problems a PSE is intended to solve. The architecture differs from situation to situation and depends largely upon the problem domain [16]. A PSE is a computational framework that solve a target class of problems, support rapid prototyping and can be used for e-science applications. A PSE brings together all the capabilities a given domain of problems might require. Certain components are must in a PSE using HPC and/or Grid support. Figure 3.2 attempts to capture a set of common components and their role in a PSE and illustrates a “generic” PSE architecture. An intuitive user interface that makes the human-computer interaction more productive is a salient feature of PSE. Application Development Tools enable users to develop prototype applications rapidly to solve simple or complex scientific problem

at hand. These applications serve as reusable off the shelf components that could reside in a library. Middleware acts as a broker for scheduling and managing the underlying HPC resources. Distributed agents acts a glue to tie the distributed applications and provides auxiliary support eg. provenance, debugging, logging among others. The last layer comprises of the hardware devices and instruments. These devices are controlled and interfaced through the low-level system software.

Triana is described as a PSE capable of acting as a complete integrated computing environment for composing, compiling and executing application for a specific problem domain. It is dominantly used as a WfMS in scientific community. The architecture of Triana is based on a third-party interfacing to Grid and p2p components. It is capable of visually developing workflow based applications that can use the underlying grid and/or p2p resources.

The Extensible Computational Chemistry Environment (ECCE) architecture supports a component-based application development, distributed code execution, and data and metadata management. Figure 3.3 shows ecce PSE model. Ecce provides a suite of tools to perform computational Chemistry integrated within a PSE. Separate tools are responsible for performing separate tasks that range from high level operations like project-management and visualization to low-level operations like calculation and job management. Integration among the components is based on an event-based publish/subscribe model. Ecce jobs can be launched on distributed resources such as UNIX workstations, Portable Batch System (PBS), Novell Queue System (NQS) type batch systems.

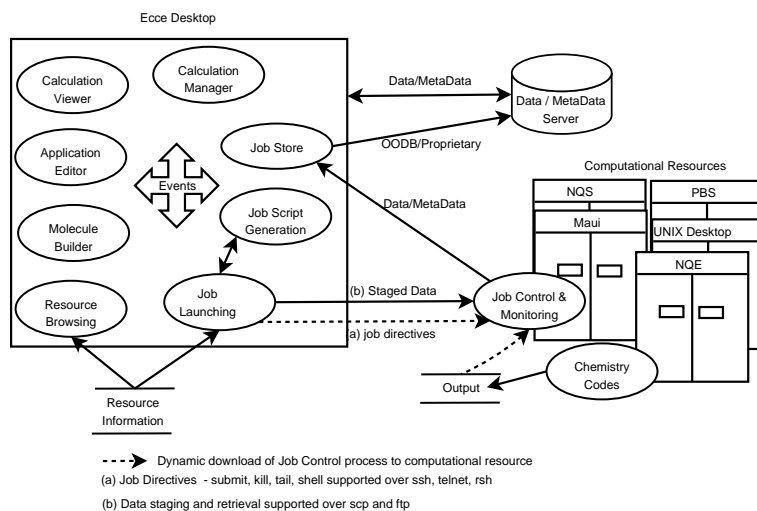


Figure 3.3: The Extensible Computational Chemistry Environment (Ecce) PSE model for Computational Chemistry ref. [13]

LabGrid is an effort towards building an integrated PSE in order to solve scientific and engineering problems for the university members of Kanazawa University. It attempts to provide a High Throughput Computing (HTC) environment through aggregation of a vast amount of underutilized computing resources at the university. These resources then could be used to solve a range of problems including Primer DNA Design, Soil Retaining Wall Placement Optimization and Velocity Vector Calculation. The PSE deploys and makes available, several application packages through a web-browser based user interface.

User Collaboration A PSE that enables the usage of HPC infrastructure is likely to support multiple users, often across departments, organizations and specializations. Users could be classified based on the technical expertise they possess or based on their requirements. In most PSE-based environments end-users, application-developers and domain experts are the most commonly found classes of the users.

Ecce uses a publish/subscribe event system to coordinate activities between applications. This provides a strong basis for collaborative work environments. A notification agent serves messages to users as well as interested applications whenever the output of subscribed job changes.

LabGrid provides an easy-to-use and collaborative web-browser based environment. Concurrent users can access the LabGrid facilities from their own PC in order to monitor some computation or visualize results.

Triana does not seem to be supporting any explicit user collaboration.

Application Development Support A wide range of tools could be used to support application development for PSE. The choice of a tool depends upon the requirements the PSE in question is addressing. These tools could be WfMS, Rapid Application Development (RAD) tools, Parameter sweep applications, compilers, Legacy hardware and software toolkits and libraries. Essentially, they make up the core components of a given PSE.

LabGrid is one such example wherein a range of “Application Packages” combine to form a PSE infrastructure. Triana integrates various tools and technologies in order to achieve a range of functionality under a single user interface. Notable components include java tools, Legacy applications Grid and P2P toolkits.

Ecce provides support for python and TCL. It includes various domain specific tools like calculators for chemistry, visualization tools and a molecule builder application. Both Triana and ecce are extensible, in that additional components could be added to extend their functionality.

Execution Model A PSE could often have more than one execution modalities depending upon the problem at hand. On one hand a simple problem involving a single job could be handled through a command-line job-submission method. On the other hand a complicated problem might need a team of users composing a complex workflow using GUI-based tools for the deployment. Certain problems with intermediate complexity could be solved by a technical expert using an xml-like “meta-language” to compose an execution sequence.

Triana can be used as a dataflow or distributed workflow system depending upon the application and user requirements. It has a rich GUI-based execution model. A user can graphically compose the execution scheme using the “box-and-arrow-modality”. A box is a representation for processing and an arrow represents flow of data and/or control.

LabGrid has no specific execution model. Jobs could be submitted through a web-browser based console. Every job is specified by its associated properties including application package, output filename, portnumber and job specific parameters. The results of this job are written on the specified output file.

Ecce uses a combination of globus [17] based globusrun and the Globus Resource Specification Language (RSL) as its execution specification. The Globus RSL provides a common syntax in order to describe resources. Various components can use this syntax in order to provide the resources to executions.

HPC Usage The amount of computation power needed and the data produced is increasing rapidly in most scientific application domains, including the Medical Image Analysis [2]. Acquiring, processing and archiving of large amount of data requires the usage of sophisticated devices

and supporting software. Further, the development and testing of application programs in such Fields takes considerable time which could be minimized through HPC.

Triana uses the underlying grid and p2p resources through Grid Application Toolkit (GAT) [18] and Jxta ¹.

LabGrid uses an existing computing infrastructure at the university including personal computers, routers and other network instruments. This provides sufficiently high distributed computing power to perform the scientific computing. A special job control and monitoring component is responsible for launching and migrating jobs dynamically among the resources. Job submission and scheduling is performed using a number of existing tools including Globus [17], Legion and NetSolve.

Ecce includes an advance reservation-based metascheduler called *Silver*. Silver manages a queue of jobs and dispatches the jobs based on a job and target resources' constraints match.

3.2.1 Workflow Management Systems

This section presents a discussion on generic and grid based workflows. This is followed by a detailed investigation on various WfMS including *Kepler* [12], *WS-VLAM* [19] and *Taverna* [20]. Other workflow systems such as *Grid-flow* [21], *GridAnt* and *Business Process Execution Language (BPEL)* are also briefly discussed. By far, the Workflow Management Systems (WfMS) is the most popular category of PSE for e-science. The Workflow Management Coalition [22] defines a WfMS as

The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.

The OGSA-WG (Open Grid Services Architecture-Working Group) glossary [23] describes the term *workflow* as closely associated with *choreography* and *orchestration* and defines it as

Workflow is a pattern of business process interaction, not necessarily corresponding to a fixed set of business processes. All such interactions may be between services residing within a single data center or across a range of different platforms and implementations anywhere.

In simplest terms, a workflow could be treated as a “program” at higher level of abstraction intended to run on a specific execution-enabled environment [24]. A workflow should be able to support common “patterns” [25], including sequential, repetitive and conditional execution.

Architecture Figure 3.4 shows an architectural model of a generic WfMS as proposed by Workflow Management Coalition. This acts as a common reference model for WfMS products. A set of workflow engines lie at core of this architecture. These engines communicate to the external world including other workflow systems through several dedicated interfaces. The process definitions are provided through ‘interface 1’. Client applications are interfaced through ‘interface 2’. External applications are invoked through ‘interface 3’. External workflow engines are interfaced through ‘interface 4’. Monitoring and administrative activities are carried through ‘interface 5’. The grid-based WfMS architecture proposed by Yu and Buyya [26] as shown in Figure 3.5 is based on the WfMS model. An important feature of this model is the classification of functions into build-time and run-time functions. The build-time functions are concerned with describing

¹<http://www.jxta.org>

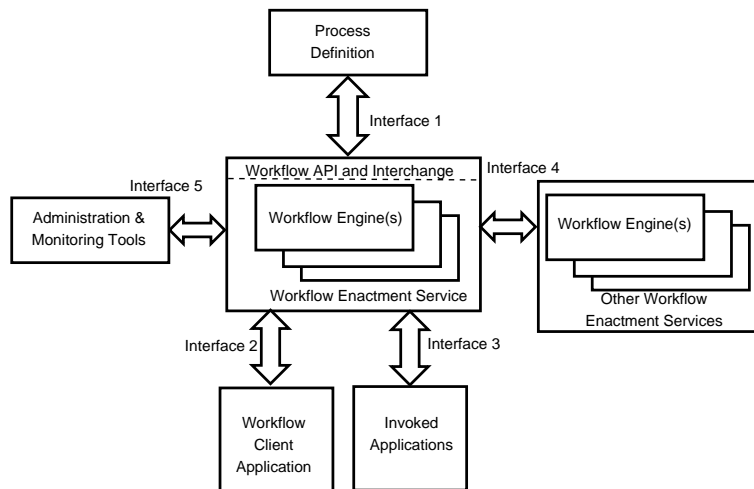


Figure 3.4: A general WfMS architectural model ref. [22]

tasks and dependencies while the run-time functions are concerned with executing and monitoring of these tasks. The model replaces the normal computing resources with grid resources and a grid middleware layer to manage these resources.

WS-VLAM is a grid-based distributed scientific workflow management system developed within the VL-e project. WS-VLAM is a service based architecture that supports execution of large scientific workflows constructed using grid-services. Hierarchical workflows with different levels of granularity are supported. This hierarchy helps in resource management by mapping the assignment of work from different level to appropriate resources. These granularity range from highest level at workflow function level to the lowest level at the individual task level. Job farming and parameter sweep applications benefit greatly by the hierarchical workflow model. A high level task can be formed as a parameter sweep and can be assigned to independent resources leveraging the embarrassingly parallel nature of such tasks.

WS-VLAM provides interoperability features through which a WS-VLAM workflow can be called and executed from within Kepler or Taverna.

Grid-flow is a petri-net based WfMS. The architectural model is depicted in figure 3.6. The Grid-Flow Description language acts as an interface between the graphics based petri-net interface and the grid-flow engine. The data and program integration layer manages the underlying distributed resources. The program integration layer works on top of the WebRun system, a unified platform that invokes remote programs using grid computing modality. A grid-flow repository helps in storing a number of compiled programs, meta-data, system property info and other necessary context information in order to run the applications. Krishnan et. al. [27] proposes an OGSA compliant grid-based workflow framework called *Grid-Services Flow Language or GSFL*. GSFL is based on a set of grid services. It has the following important features:

- *Service Providers* is a registry of the services taking part in the workflow.
- *Activity Model* is a list of all operations belonging to the individual service providers.
- *Composition Model* describes the composition of grid services into a new service.
- *Lifecycle Model* is concerned with the runtime activities of the services.

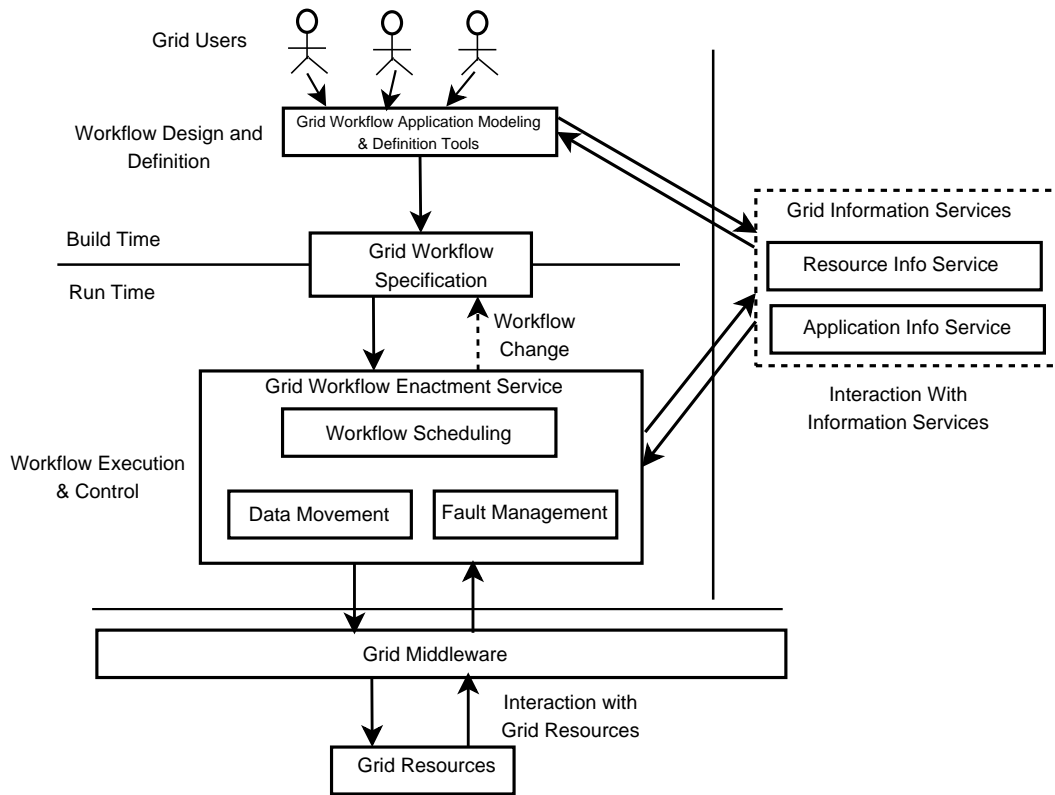


Figure 3.5: A grid based WfMS model [26]

Kepler is a scientific workflow management system built on top of the PtolemyII² system. Webservice extensibility is one of the highlights of Kepler. The “actor” modality of Kepler allows to create actors representing webservices/grid services facilitating an extension to the existing functionality. “Box and pipe” model of user interface makes it easy to use for scientific workflows. *Planning for Execution in Grids or Pegasus* [28, 29, 30] is a scientific workflow management system that maps complex and abstract scientific workflow specification onto the grid resources. A user can create an abstract workflow using *Chimera* [31] and Directed Acyclic Graph (DAG) tools. Chimera is a “virtual data system” that could be used for the generation and management of derived data in scientific experiments. This abstract workflow is submitted to the Pegasus engine. Pegasus, in turn consults the grid based resource brokers to locate appropriate resources, data and applications to execute the workflow tasks. Taverna from the *myGrid*³ project is webservice based tool to create and execute workflows in the field of lifesciences. A workflow is formed of tasks or “processors” connected with each other in a data flow that is represented by arrows. A processor can consume multiple input data and produce multiple output data. The workflows are written in a new, xml-based conceptual language called Simple conceptual unified flow language (Scuff). *Karajan* [32] is a derivative of gridant [33]. It is an engine as well as workflow language that supports a number of workflow patterns [25]. Karajan is scalable, parallel and extensible language that can be used to manage a large number of tasks

²<http://ptolemy.eecs.berkeley.edu/ptolemyII>

³<http://www.mygrid.org.uk>

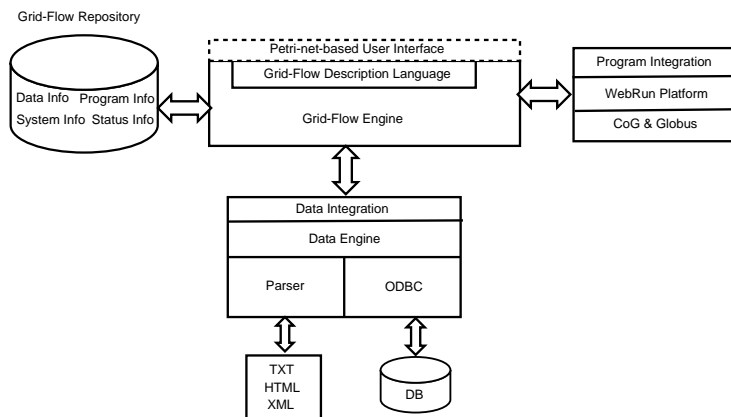


Figure 3.6: The grid-flow WfMS model ref. [21]

as a workflow.

User Collaboration Modern science is an era of collaborative work. With the emergence of sophisticated computing infrastructure, more and more science is done across organizations in a collaborative fashion. Workflows becomes pivotal in sharing resources, data, applications and knowledge, fuelling collaborative environment. A workflow may be considered as a societal entity supporting work coordination. One of the important component for a workflow enactment engine is a *human control interface* [24]. Its functionality include supervision of the overall execution, distribution of the results and manage collaborative activities. Collaboration in WfMS is about providing the following:

- Co-editing of the workflow specification.
- Simultaneous receipt of the results of execution.

Currently, no WfMS seems to provide a fully collaborative environment. However, several WfMS provide a semi-collaborative environment. Most WfMS provide an automated offline-sharing and reusability mechanisms at the workflow as well as resource level. Taverna enables the sharing of resources through webservices. These shareable services are used by the bioinformatics community. Non-sharable scripts could also be used for individual purposes using Taverna. GSFL incorporates a *serviceprovider* modality implemented as a list of services identified by a unique name. These services are used as working units in GSFL. Users can use control,data and notification models as described in section 3.2.1, in order to communicate with processes as well as other users. Communication in GSFL includes control messages as well as large amount of data. The Grid Workflow Definition Language (GFDL) proposed as part of Grid-flow in [21] is a translatable language by design with most of the scripting workflow languages. This enables construction of a translator an easy and mechanical task. This makes an integration and collaboration with other workflow systems and hence usage can become easy. GridANT and Karajan doesnt provide any specific features for collaboration *per se*, however they could be used collaboratively through their underlying service based modality. Users can use shareable grid services in the form of workflow tasks.

Application Development Support Most WfMS provides a minimal set of auxiliary or integrated tools for application development. Depending upon the problem domain, they might or might

not be sufficient. This calls for an external development environment suitable to the user needs along with the workflow system. Several systems provide mechanisms for saving and reuse of previously created workflow and legacy applications. These repositories, often called “application factories” [34] acts as a collection of invocable applications through a registry. User may compose new workflows using a combination of these applications with appropriate parameters.

Kepler [12] provides some support through its user-friendly graphical workflow generation interface. Other workflow systems does not seem to provide any significant application development support with an exception of gridAnt, which provides an ‘application factory’ construct.

Execution Model Workflow definitions for large-scale problems may get reasonably complex [35]. Such complexity calls for a sufficiently usable execution model. A workflow execution model can be as sophisticated as an intuitive box-and-pipe network or can be as raw as a set of customized scripting statements. At times the execution model might be difficult to learn and apply. Thus requiring user-training for more than simplest of problems. XML and XML-based vocabularies are most popular execution specification models for workflow systems. Business Process Execution Language (BPEL) is increasingly becoming popular for the representation of scientific workflows [35]. BPEL is Based on Service Oriented Architecture (SOA) wherein, every process acts as a service. The execution specification is based on the XML schema. This schema is responsible for the typed variable system and name resolution through namespace management. It provides constructs for activities such as variable declaration, service invocation and SOAP message receipt, dataflow and controlflow. BPEL can maintain and communicate temporary data structures to other services during execution.

The Academic Medical Center Distributed Workflow Management System (AMC-DWMS) is the WfMS used in the current work. For more information about AMC-DWMS see section 2.4.3 or [11].

3.2.2 Rapid Application Development (RAD) Environment

Scientists develop applications which need high level of abstraction. This motivates a need for rapid prototyping and the reuse of existing components in an easy fashion. A Rapid Application Development (RAD) facilitates quick development of new applications and easy integration among existing components. In this section we study the *Java CoG kit* [36, 37] RAD environment.

Java CoG kit is a Rapid Application Development environment for Computational Grids. A layered schematic diagram of the Java CoG kit is shown in figure 3.7. External applications from various scientific domains such as nanotechnology, bio-informatics and disaster management systems interface the CoG kit at different layers depending upon their requirements. These requirements include application development and testing, data and job management, scheduling and resource management. A CoG abstraction layer enables transparent access to underlying local and distributed grid resources including third-party schedulers such as Condor’s DAGMan.

The Rapid Application Development Environment used in the current work is the Delft Visualization and Image Processing Development Environment (DeVIDE). For more information on DeVIDE see section 2.4.1 or [7].

3.2.3 Parameter Sweep Environment

Many scientific applications need a repeated execution of the same program over a very large range of parameters. These kind of applications, known as parameter sweep applications provide results that are studied for statistical and analytical purposes. This section discusses the *AppLeS*

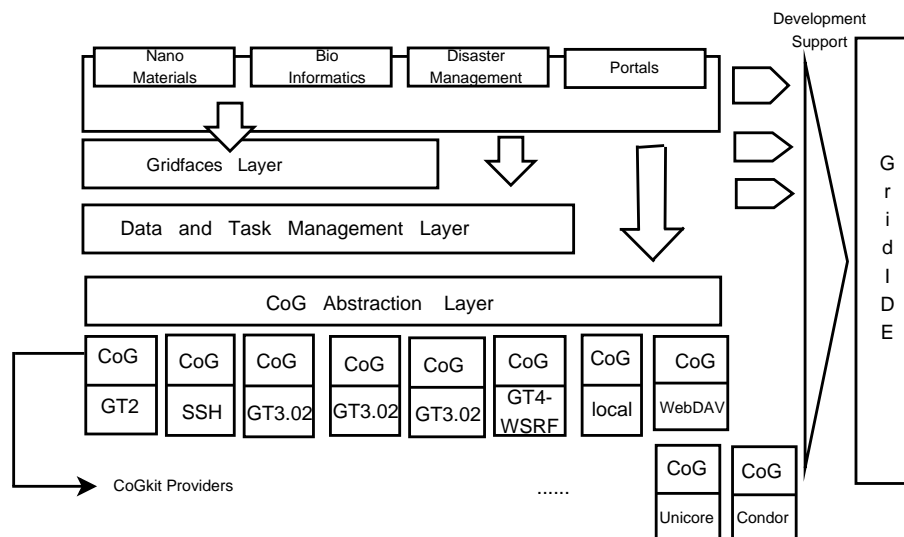


Figure 3.7: The java CoGkit model for Rapid Application Development (RAD) on the grid ref. [37]

or the *Application Level Scheduler* [38] parameter sweep environment. *AppLeS* or the *Application Level Scheduler* [38] is a parameter sweep system that takes into account the system and application requirements in order to adaptively schedule the application jobs. It attempts to exploit the embarrassingly parallel constructs within the application and deploys them on the grid. Figure 3.8 shows an architectural model of the AppLeS parameter sweep system. A component based architecture provides components for different activities. Four type of parameter sweeps are available viz. ‘MaxMin’, ‘MinMin’, ‘Sufferage’ and ‘XSufferage’. A ‘workqueue’ algorithm handles the job scheduling. An ‘actuator’ component performs, staging-in-out and executions on underlying resources. A meta-data manager performs querying the status of parameter sweeps. The grid executable resources are interfaced through *GRAM* and *netsolve* job submission systems. File transfer for staging-in/out is handled through the *Network File System (NFS)* and *Global Access to Secondary Storage (GASS)* service provided by Globus for data movement and access. AppLeS supports the development of large scale parameter sweep applications through what is called the AppLeS Parameter Sweep Template or APST. AppLeS uses the underlying grid resources in an optimal way by enabling colocation of data by parallel jobs.

The parameter sweep environment used in the current work is Nimrod. For more information on Nimrod see 2.4.2 or [10].

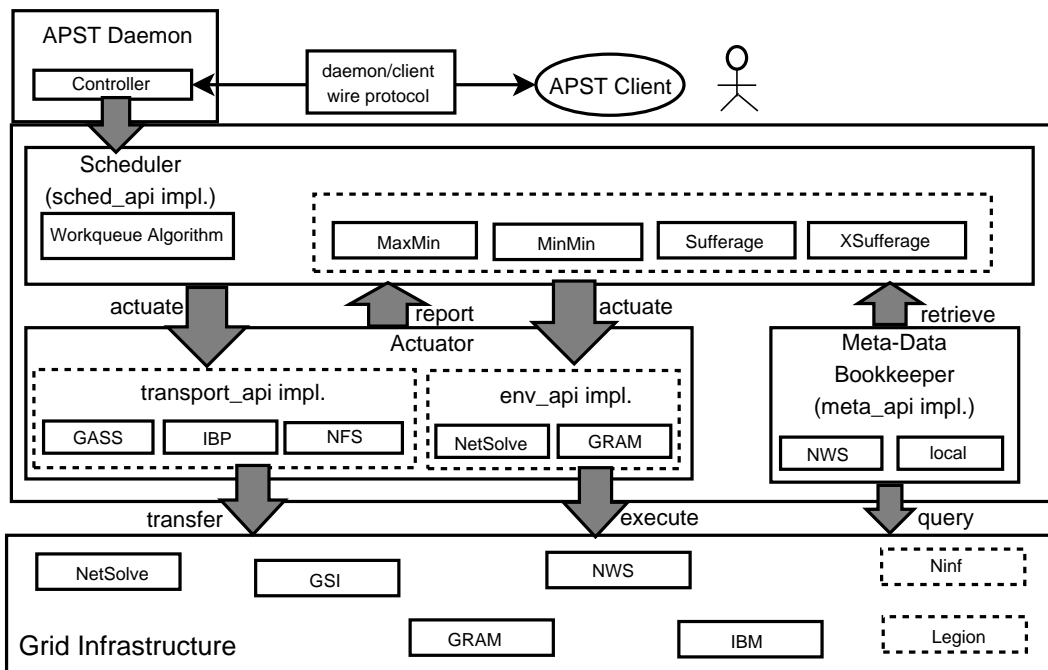


Figure 3.8: The Application Level Scheduler (AppLeS) parameter sweep model ref. [38]

3.3 Provenance Management

Scientific application need provenance mechanisms to trace back the derived data for its origin and the process of its evolution. The level of sophistication for provenance varies and depends upon the application requirements. In this section we investigate two provenance management systems for scientific applications. These systems are *Chimera* [31] and briefly discuss the Adaptive Disclosure (AID) ⁴ Project being developed within the VL-e project.

Chimera Chimera is used for representing, querying and automating data derivation. It is used for tracking the derivation of processed data.

Chimera is based on a so called “virtual data system” for its operations and features its own query language “virtual data language”. Figure 3.9 shows a schematic of the Chimera architecture. It has two major components, the Virtual Data Language (VDL) Interpreter and the Virtual Data Catalog (VDC). VDC implements the virtual data schema, a relational database schema responsible for storing the relationships and tracking information about the data. External applications access Chimera via the Virtual Data Language, a query language that can be used for defining data as well as querying the system. A VDL Interpreter manipulates the data derivations and transformations and interacts with the schema using plain SQL. Grid resources as used by the applications remain external to the system.

The Adaptive Information Disclosure (AID) Project The AID project provides the AIDA (AID Adapter) service can be used to support provenance. It currently works as a firefox plug-in that let’s user ‘annotate’ the URL being viewed with either established jargon (e.g. imported

⁴<http://adaptivedisclosure.org/>

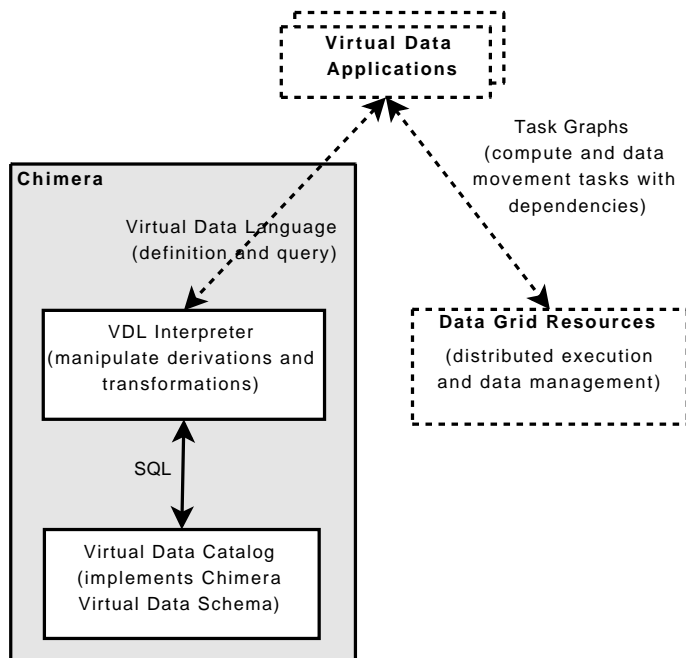


Figure 3.9: Schematic of the Chimera Architecture showing interaction between Chimera system, Virtual Data Applications and Data Grid Resources ref. [31]

as OWL into the repository) or with ad hoc tags. AIDA is a generic toolkit developed by the AID group, aimed at groups of knowledge workers that cooperatively search, annotate, interpret, and enrich large collections of heterogeneous documents from disparate locations. It consists of a set of services for metadata, learning, and storage and retrieval. A few example clients are provided such as an annotation client, query construction client, and a named-entity recognition client that show how the services can be applied in practical applications. The services perform the following tasks: text indexing, text statistics, meta-data storage and querying, thesaurus reasoning, annotation, text retrieval, spelling correction, synonym detection, and model learning. The AID toolkit is based on web services and W3C standards such as Web Ontology Language (OWL), Resource Description Framework (RDF) as well as other open source technologies such as SRB, Lucene, Sesame, and Jena.

3.4 Discussion

Several systems were studied in the context of literature review.

Problem Solving Environments (PSE) are seen as key tools for e-science. One of the characteristic feature of all the PSE is a component based architecture. These PSE are extensible and scalable. Most provide mechanisms for distributed computing and collaborative development. The degree of sophistication varies depending upon specialized (ecce for chemistry) or generalized (LabGrid) requirements.

Workflow Management Systems (WfMS) make the automation of a series of tasks very easy. They act as high level programming interfaces for scientific applications. Generic frameworks of WfMS was studied. Several WfMS and their characteristics were studied. An important

characteristic of WfMS observed is that they leverage the underlying distributed architecture based on the target tasks (WS-VLAM, kepler). Also data-flow based workflows are very common in scientific application area (taverna, Grid-flow). Specialized WfMS also provide mechanisms for meta-data and provenance management (Grid-flow).

Parameter sweep applications provide a way to execute applications for large range of parameters. They provide mechanisms for scheduling, submitting, and collecting results for jobs. An important feature of parameter sweep applications is an ability to query and monitor the executions meta-data. The AppLeS system was studied as parameter sweep application.

Rapid Applications Development (RAD) environments facilitates quick application prototyping. Java CoG kit was studied as a representative RAD environment.

Chimera and Adaptive Information Disclosure (AID) project were studied under provenance management. Chimera is a system that provides for backtracking for derived data in an external system. Chimera uses a sophisticated relational database schema to provide provenance mechanisms. The AID project provides AIDA services that act as firefox browser as an annotation tool. Note that both provenance management systems treat applications as external environment.

Proposed Architecture

4.1 Introduction

This chapter presents a detailed description of the proposed Problem Solving Environment (PSE) architecture. Architectural framework that support the MIA-ADL must provide a way so as to make a way for interfacing a diverse set of support systems together. In order to support new and unforeseen legacy systems the architecture must be extensible. Further, it should be able to make use of the underlying grid resources. The approach for proposed architecture is a component based paradigm where-in customized and glued interfaces of existing systems talk to each other in a predefined way. One of the goals of this work is to enable the support systems to interact with the PSE. One way of achieving this could be through a tight integration of support systems. However, the support systems used for the current work are highly heterogeneous and complementary in their functionality, environment and invocation requirements. This would make it tedious and resource intensive in order to load an environment for each system into the PSE. Further, it will make the architecture more rigid and less scalable as accommodating new systems would require additional work of studying and integrating the environment for the new system. Considering these difficulties, a “passive” architecture was proposed where in the PSE exposes a number of grid service calls and the external systems passively call these services according to the operational requirements.

4.2 Modeling Requirements

The requirements as described in chapter 2 are modeled using the Relational Database Management System (RDBMS). The database stores information in the form of values and references to the remote resources. These references are stored in the form of URL links pointing to the virtual file system. The system maintains entity and referential integrity through primary and foreign-key relations respectively.

4.2.1 Concepts and Terminology

This section presents the concepts and terminology used for modeling the requirements. These terms are used throughout this chapter while discussing architectural and data models.

1. **User** A user is a person or an external system (eg. support systems) interfacing with the problem solving environment.
2. **Project** A project is an aggregation of the information associated with a single MIA problem, containing sessions. Users can subscribe to the project. Project contains sessions which in turn contains runs. A project has an associated MIA application.
3. **Session** Session is a logical entity that belongs to a project. A session may correspond to a phase in MIA process.
4. **Application** An application is an abstraction of an executable program containing the information of the program, its parameters and a description. An application version is a concrete program that contains the information about its location, the parameters associated and the compatibility with other versions.
5. **Parameter** A parameter is an argument passed to the application when it is executed. The formal parameter is associated with the application and parameter instances are associated with application execution.
6. **Run** A run is collection of information associated with an instance of an application execution.
7. **Event** An event is a condition that occurs during the execution of the application in any of the MIA-ADL phase. An event normally corresponds to an error condition or a normal execution of an application.
8. **Attachment** An attachment with any of the above entities can be associated. Depending upon the context, attachment may be a document indicating dependencies or an auxiliary file or an administrative document.

User The PSE should be able to maintain the information about users and their affiliations. A “super-user” is a special type of user-role in order to manage the users. A user may have subscription with projects and sessions and is notified via email about the events (s)he is subscribed to. A user also is the author of one or more applications and versions of an application. One or more users may be associated to the application.

Project A project represents a MIA problem. Project is at the highest level entity that contains many other entities including users, sessions and runs. All the users that are part of a project form a project team. A team member of a project is able to create, access, update and remove (only super-user is allowed) a project. An active project is the one a user is interacting and working in. At a given time not more than one project should be active for a given user. However, more than one project can be active simultaneously for different users.

Application An application is a program that is created using the support systems. The applications along with their versions are registered, searched for and retrieved from an application repository. Application registry process involves setting the attributes of application and placing it in the database. These attributes include:

- Name of the application.
- Version of the application.

- Backward compatibility of the application.
- Author of the application.
- Parameters associated with the application.
- Return status of the application.
- Location of the executable program.
- A description of the application.

The PSE should automatically keep the date of creation of the application and the date the application was last called. The user should be able to search and retrieve an application from the database through registry. The PSE manages versions of applications. All versions of an application are related to each other. A user should be able to run different versions of same application for same set of parameters provided they are compatible. The PSE identifies compatible versions of an application.

Run A run is an instance of application execution. The PSE records the information related to runs through the support systems. It captures information in the database corresponding to following attributes of each run:

- The application and version involved in the run.
- The input and output parameters and their values.
- Completion status of the run.
- The date of execution.

A user may also record the run with additional attributes as follows:

- A description of the run.
- The author of the run.
- Whether the results are inspected.

Session A session contains a related set of runs. Session is managed by user in that he can access, update and remove a session. At a time more than one session might be active and the user must have flexibility to associate a run to any session. However, one run can belong to exactly one session. Events related to runs are generated inside the sessions and all users in the sessions who are subscribed for the events are notified via email for the event.

Parameters The parameters for an application must be specified and stored on a per version basis. parameters can take values as well as reference to a remote resource. Instances of these parameters are stored as part of run.

4.2.2 Search and Provenance

Locating and retrieving data as a result of a series of transformation is termed as data provenance. Provenance requirement states that the PSE should facilitate a user to be able to search ‘backwards’ and discover information related to a particular run, session and application. This includes the application parameters, session and project information, the run responsible to produce a given output and the user associated. Provenance requirement involves maintenance of logs, retrieval of historical data, execution trace of MIA applications, parameters, metadata and clinical interpretation of archived data for the purpose of reproduction, evaluation, and/or recovery. The PSE should provide a search feature. This feature should be tuned to be able to perform search at different levels of scope. A system-wide broad search based on a generic keyword from all entities and their attributes to a refined search for a particular attribute and advanced search for a combination of values. Following are the classes of search that should be supported:

1. **Open Search** An open search is a system wide search based on the user-provided keyword. For e.g. search for a keyword “contrast”, “elimination”, “mask” from the descriptive fields related to entities.
2. **Attribute Based Search** An attribute based search is based on the value a user provides for an attribute of an entity. For e.g. search for all runs whose description contains the keyword “soft tissue”.
3. **Advanced Search** Advanced search is based on a combination of attributes. A logical “or” or “and” combination of the above search options could be provided in the advanced search.

Fetch Run A user should be able to retrieve an existing run from the database and create a new run using existing one. The user must be provided with the information if a different version of the application is compatible with the version used in existing run. A rerun executes in the same manner as the run and same information as the run is stored in the log. A rerun may be executed with exactly the same information as the existing run or it can have change in parameters and/or application version.

4.2.3 Other Requirements

Authentication Only authenticated users should be able to use the PSE. User credentials should be validated by the PSE before the service calls are made. Credentials should be delegated by the user to access the underlying distributed resources where appropriate

Distributed Resource Usage The PSE should be able to transparently use the available VL-e PoC provided distributed resources for storage and execution. These resources include distributed database and filesystem as well as a grid based HPC environment. An environment for the storage and automated (*as opposed to manual*) transfer of large amount of data should Provide execution control management as per the user requirements.

4.3 The Data Model

Choice of Technology Ontologies are becoming increasingly popular among scientists for data modeling for information systems [39]. Information exchange requirements among different e-

science disciplines drive the need for a flexible and adaptable approach to manage scientific data. Ontologies provide structural and methodological flexibility that fulfills unforeseen requirements in an interdisciplinary environment. Ontologies based data model also help in knowledge representation and extraction from the database. An ontology associated with a database can provide for a wide range of flexible query options. An Resource Description Framework (RDF) based database can be associated with such ontologies. Apart from the advantages as mentioned above, there are a few challenges facing the ontology based information systems. These challenges are summarized by Yildiz and Miksch in [40]. One of the main challenge is that it takes much more time, effort and cost to develop an ontology-based information system than a normal one.

The RDF based and SQL based Relational Database Management (RDBMS) technology was studied keeping the provenance management in focus. Since the provenance requirements of the PSE are predefined and constant a relatively rigid and fast information design is best suited. RDBMS is a proven and robust approach to management of a database with such requirements. Predefined associations among entities in RDBMS lends itself into intuitive and hence stable design. Retrieval of data from an RDBMS is computationally cheap, flexible and quick. This is better than the RDF technology which is based on a semantic discovery of information based on an exhaustive relationships among the data items. Ontologies and RDF based information system can provide an extensive provenance facilities at the cost of more development efforts and computational time which was thought to be an “overkill” for the limited provenance requirements of the current work.

The Entity/Relationship Model The Entity/Relationship (E/R) model has been extensively researched and documented approach to capture the data domain requirements of a system into the RDBMS model database schema. A graphical representation of the elements of the domain and the relationships among them are modeled through various diagrammatic artifacts as follows:

- **Entity:** An entity is a basic element and corresponds to the “nouns” of the system. It represents a basic class of data eg. user, item, person etc.. An entity is represented as named rectangle box.
- **Attribute:** An attribute is a property of entity. One entity may have many attributes representing the information that an entity possess. An attribute is represented as an oval or “balloon” attached to the entity.
- **Relationship:** A relationship corresponds to the associations among the entities. Relationships represent the “verbs” of the system. A relationship is represented as a diamond shaped box that connects two or more entities through a line.
- **Cardinality:** Cardinality or multiplicity of a relationship is the number of entities that can participate into a relation. Cardinality can be either of ‘one to one’, ‘one to many’ or ‘many to many’. Depending upon the cardinality the entities are mapped into the tables when the schema are converted into the RDBMS implementation.

4.3.1 Entities, Attributes and Relationships

As figure 4.1 shows, the information for the complete environment is modeled and expressed using an Entity/Relationship model. The entities are mapped into tables of the database at the time of implementation. Each attribute of an entity is mapped as the column of the table. Each entity has an implicit unique id (primary key) that is managed by the database system. Following is a brief description of the entities and their attributes from the system.

User A user is a person associated with the MIA-ADL. A user is associated with project. User subscribes to session, creates application and executes runs. Following attributes define a user

- *name*: The name of the user used for user registration and search purposes.
- *organization*: The organization, the user belongs to. Used for registration and search purposes.
- *email*: email id of the user. User will receive notifications on this email id.
- *isactive*: Is the user active? If the user is not active, he will not receive notifications.
- *description*: A brief description about user. This mentions the role

Application An application is a MIA application that is stored, along with possible versions into the system. A single application is associated with project. However, more than one application versions may be associated with a project. An application is authored by a user. Different users may author different versions of the same application. Following are the attributes associated with an application and its version.

- *name*: Name of the application. Used for registration and retrieval purposes.
- *version*: Version of the application. Used to identify a version and assign it to the execution.
- *date_created*: Date of creation of application.
- *description*: Description of application. Description may contain characteristics of application such as the comments on quality or usage of the application.
- *iscompatible*: Compatibility with previous version. This essentially means the application can run with the same signature as its previous version.
- *location*: The location of the application in the application repository. This location is a URL to a distributed storage resource.

Parameter A parameter is an input or output value/file that is associated with one or more application. An instance of parameter (parameter-instance) is a value of parameter at runtime. Parameter is associated with application and a parameter-instance is associated with run. Following attributes define a parameter and parameter-instance.

- *name*: Name of the parameter.
- *type*: Type of the parameter.
- *input_output*: Whether the parameter an input or output.
- *value*: The value of the parameter.
- *isreference*: Whether the parameter is a value or a URL reference to a remote location.

Project A project is an entity that glues together several other entities and represents one MIA problem at a higher level of abstraction. Following attributes form a project.

- *name*: The name of the project. Used for storage, search and retrieval purposes.
- *description*: A description of the project. Includes the feature, goals and status related remarks about the project.
- *date_created*: Date of creation of the project.

Session A session is an entity that represents one MIA-ADL phase and is a collection of runs. Following attributes define a session:

- *name*: Name of the session.
- *date_created*: Date of creation of the session.
- *date_accessed*: Date, the session was last accessed.
- *description*: Description of the session.

Event An event is a condition that occurs during the course of an execution of application. Events are defined in order to let users subscribe and receive notifications. An event is triggered when a run is added to the system. Following attributes define an event:

- *name*: The name of an event. A fixed number of events are defined as: OK, ERROR, and ABORTED representing different conditions that can occur during execution.

Run A run is an instance of execution of an application. A run is associated with user, session and application and parameter-instance. Each run is formed of the following attributes:

- *completion_code*: An exit status code that determines if the execution completed successfully.
- *ischecked*: Weather the outcome of this run is checked by a user.
- *usernotified*: Weather the user is notified of the outcome of this run.
- *date_created*: The date of creation of this run.
- *description*: Description of the run.

Attachment An attachment is one or more files associated with other entities. Attachments are associated with users, projects and sessions. They can be user profile, or project/session documents.

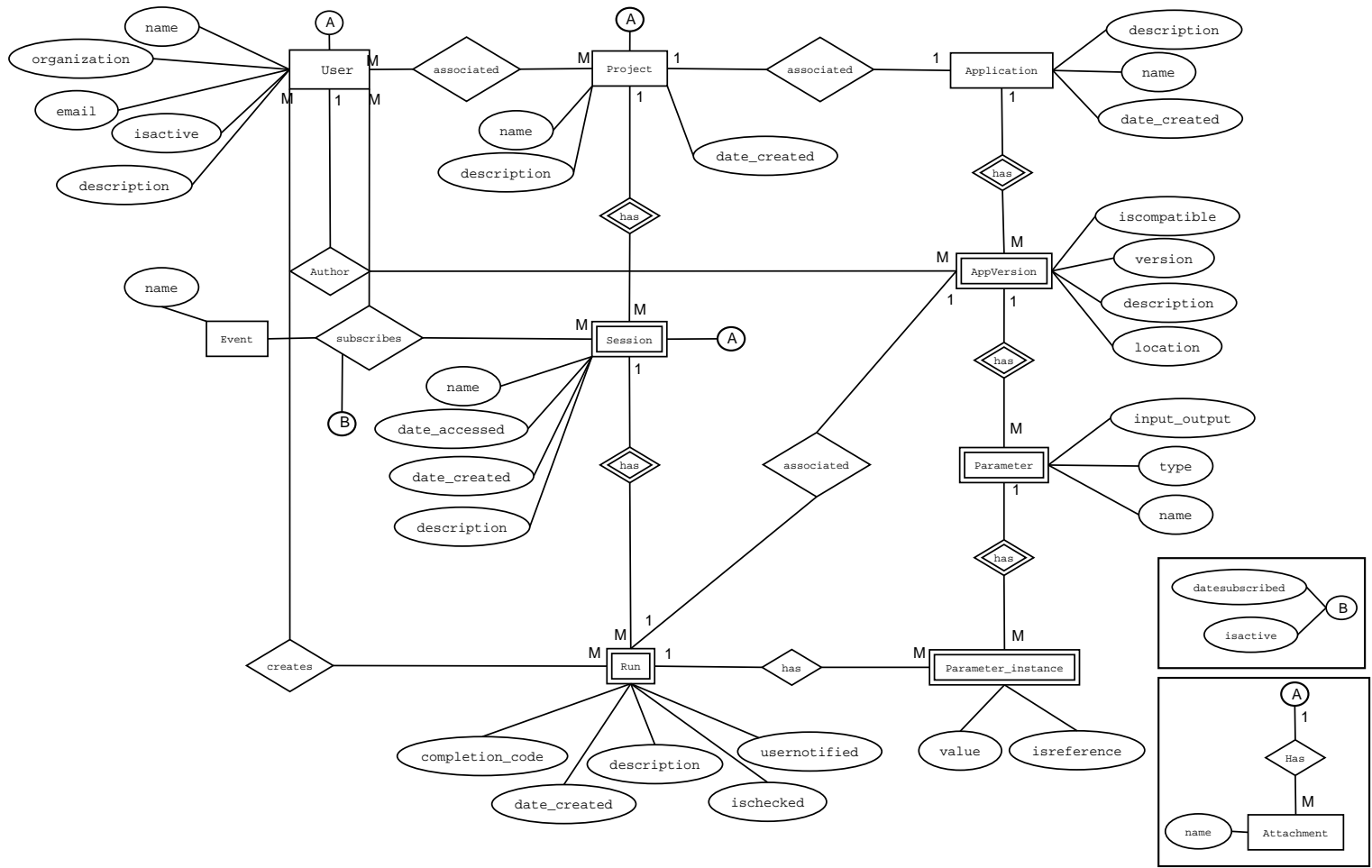


Figure 4.1: An Entity/Relationship Model for the MIA Information System; Entities and Relationships are shown as boxes and connecting lines respectively. The attributes are shown as ovals attached to the entities. The multiplicity of relationships are shown with “1” and “M” symbols for a “one” or “many” kind of relationship

4.4 PSE Architecture

This section presents the evolution of the PSE architecture. The architecture supports a “passive” information management. It supports the information flow among different MIA-ADL phases via data provenance mechanisms [41]. Its passive in the sense that the information is stored and retrieved by the external systems and users retroactively after the events happen. This enables a long term management and diagnosis of activities along the MIA-ADL. Three layers (User Interface, Web-Services and Distributed Resource) along with support systems (AMC-DWMS, Nimrod and DeVIDE) form a framework for managing and organizing the information associated with a MIA application.

User Interface Layer The user interface layer provides interactive and batch-mode command-line and web-browser based interaction with the system. A command-line interface is used with the batch mode functions and a web-browser based interface can be used for interactive mode functions. The commandline calls are passed as services calls to the next layer. The proposed architecture is illustrated in 4.2.

Support Systems and Service Call Stubs The support systems perform application execution operations. These executions are recorded in the system through service calls placed in the service-call stubs. The service-call stubs is service call signature code inserted into the support systems. This call can be automatically or manually made when a support system finishes an execution operation.

Grid-services Layer (APIs) The grid-services layer consists of APIs providing Application Description, Data Provenance, Project and Application Execution management services. These services are invoked through client by the user or through the stubs extended from the support systems. Following APIs serve the functional requirements for the environment:

- *App Description Management* API handles the task of creating and registering new applications, creating, adding and associating versions of applications and retrieving applications. The services provide a mechanism to describe, store, access and retrieve MIA applications and their versions.
- *App Execution Management* API performs the task of recording the executions of the applications writing the parameters, status and sending notifications to users. The services are used to store information that needs to be tracked from phase to phase, including applications, parameters, image data, and standard logs.
- *Project management* API manages the creation and deletion of projects, sessions and users. It also manages the associations between projects, sessions, users and runs. The services facilitate the creation, access, subscription and cleaning up of projects.
- Using the *Data Provenance Management* API services, the user can retrieve provenance data to navigate back during the MIA lifecycle. The complete settings used to generate a given result can be restored and used to run the application interactively for diagnosis and backtracking purposes.

Resource Access Middleware A middleware layer abstracts the available resources and presents them transparently irrespective of the physical location or access protocols. Distributed resources are managed through the VFS. A relational database is accessed through a connectivity paradigm through a load-sharing connection pool.

VL-e Resources Layer Distributed storage resources available within the VL-e are used to store persistent data. The data and references are controlled through PostgreSQL database. SRB, gridFTP and local file systems form the data storage. These are transparent to the user and accessed through regular URLs via the VFS.

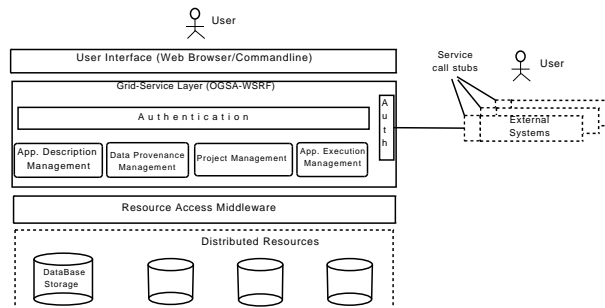


Figure 4.2: A complete Architecture of the proposed PSE for MIA-ADL showing User Interface, Grid-Services, Resource-Access Middleware and Distributed Resources Layer along with the support systems and service call stubs

4.5 Usage Scenario

Figure 4.3 shows a typical usage scenario (in chronological order) with the system. A typical sequence of functional calls is shown in the figure. A user authenticates himself with appropriate credentials presented to the Authentication service. An authenticated user performs project management to create the basic elements of the PSE including, project, session and application. Further, a user can add a run to the PSE via a call to the addrun function. A run is downloaded using the fetchrun call. The provenance data can be obtained through fetchrun, or one of the several forms of the find function. Each function call has a return status that determines the success of the call.

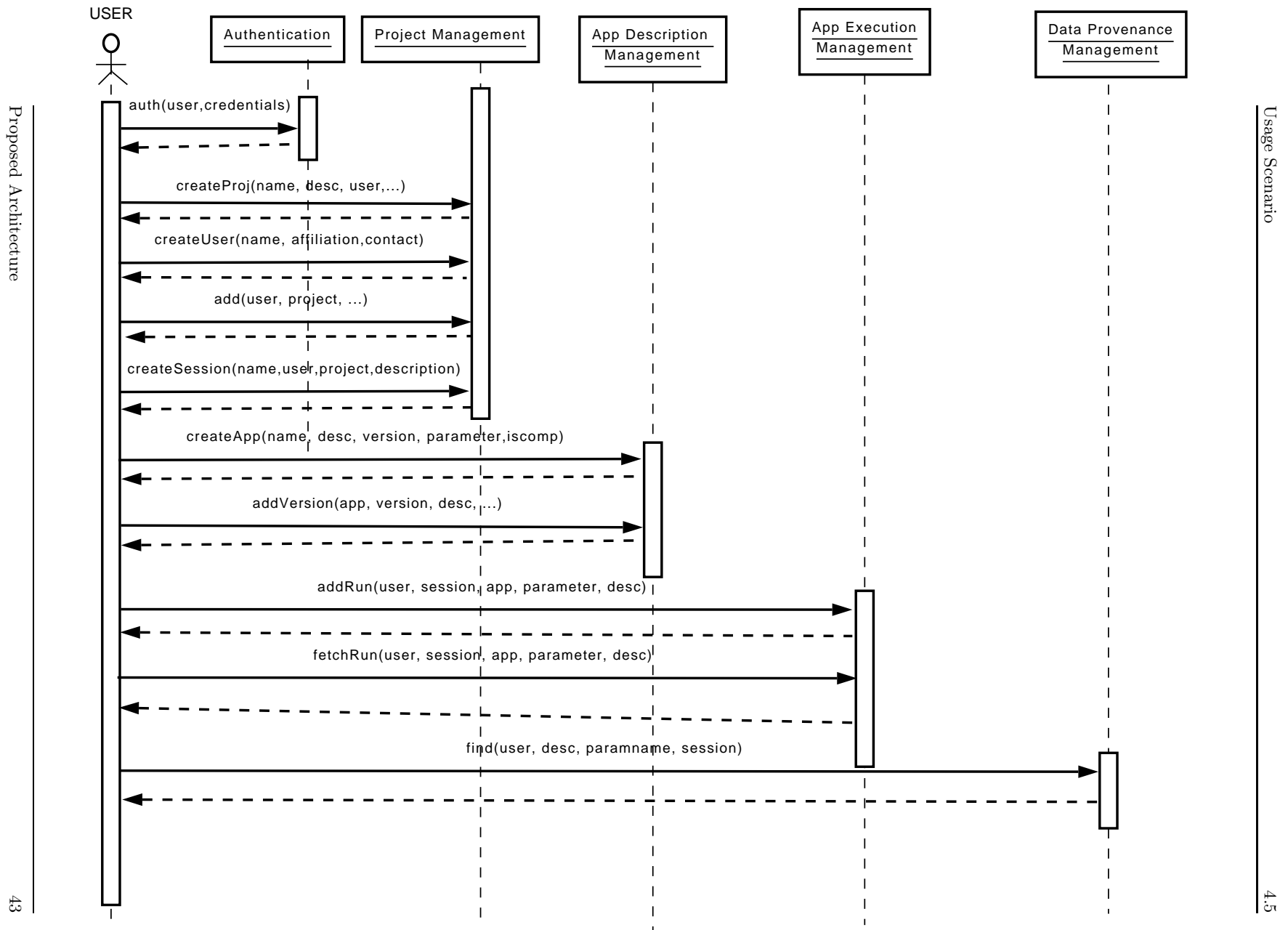


Figure 4.3: Sequence Diagram showing a typical usage scenario with the system; A user makes function calls to the system and receives the responses to accomplish a given task

Implementation

5.1 Introduction

This chapter presents the description of implementation for the architecture proposed in the previous chapter. The architecture was implemented using a classical bottom-up object-oriented software development paradigm. The functionality of MIA-ADL was segregated into APIs and mapped into functions. The functions were developed using the standard java platform. Next the functions were wrapped into the Globus Toolkit version 4 (GT4) grid-services. Finally the clients were developed corresponding to these services. The chapter begins with an overview of the implementation followed by a description of the functions and their implementation. Finally, section 5.6 describes the implementation of grid-services.

5.2 Overview

An implementation of the APIs for the proposed architecture was carried out. Various software tools and technologies used and interfaced at different levels of implementation depending upon the suitability. The distributed resources were treated as transparent to the grid-services layer and interfaced with the appropriate access middleware. This enables the user to access, store and retrieve resources using a single URL based addressing. The service clients were implemented and can be invoked as call signatures (as explained below). This makes it consistent to call services by a human user as well as by the support system stubs. A call signature that takes in a well defined parameters makes it easy to extend the command-line client into a web-based or standalone rich GUI client.

Call Signatures The services are called by high-level clients using a “call signature”. This call signature provides the name of the service, the input parameters, the output parameters, the options for execution, and the return status. Each call must conform to this signature to invoke the service correctly. These signatures conforms to the application-specific signatures and takes into account additional parameters required to record the executions into the system. For example, an application may have 2 input and 3 output parameters, but the session of which this execution is part of must also be taken into account.

5.3 Choice of Technology

Figure 5.1 shows the architecture as proposed in chapter 4 with the choice of technologies. The APIs as described in chapter 4 were implemented on Java platform. Each API was implemented as a Java class with a set of Java functions. These functions correspond to the activities done as part of the MIA-ADL. The Virtual File System (VFS) API is used as a part of the VL-e toolkit (vlet) API developed within the VL-e project. The standard java database connectivity (JDBC) API was used to interface with the Postgres database. The Jakarta Commons Collection API provides the facilities to pool the connections to the database for an efficient and fast interaction with the database via a lightweight Java Naming and Directory Interface (JNDI) interface. The standard JavaMail API was used to send email notifications to users. The functions were in turn wrapped into grid services. This was done by exposing the function call signature through the Web Services Description Language (WSDL) and building and deploying the corresponding functions into grid services. The grid services are Globus Toolkit version 4 (GT4) compatible and deployed on a secure container running on a linux operating system. The clients for these services were developed working in interactive and batch (commandline) mode. These clients can connect to the services via service call using a published service endpoint URL. The factory-instance grid service pattern enabled efficient creation of service instances on multiple concurrent calls by clients. The function and service code was developed using eclipse version 3.2 Integrated Development Environment (IDE) platform.

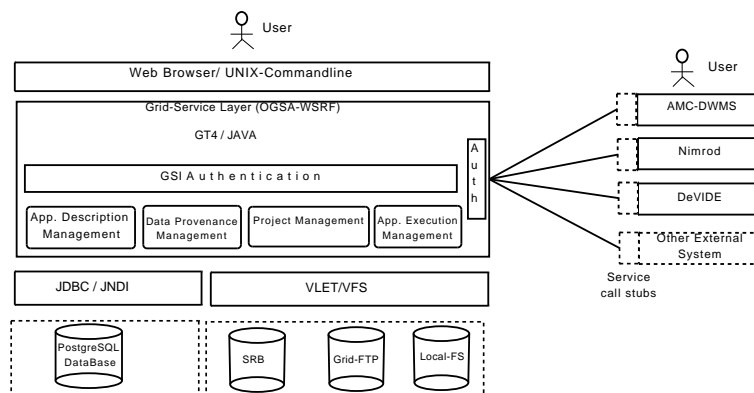


Figure 5.1: Architecture highlighting the choice of technology for implementation

5.4 API Classes Implementation

In an object oriented software development paradigm, a number of entities co-exist concurrently and makes a “state” of the system at any given time. This state can be represented as an instance diagram that demonstrates the dynamics of the system. As shown in the figure 5.2 an instance diagram for the key elements of the system are shown at any arbitrary point in time. Shown in the picture are the associations between these “live” elements of the system. Any function call will affect the state of the system with manipulation of information within the classes or the information between the classes. For example an update in the *description* attribute of *application* class will change the internal state of the class where as a call to the *addversion* will change the state of the relation between the *application* and *appversion* classes in that a new

version will be associated with the application.

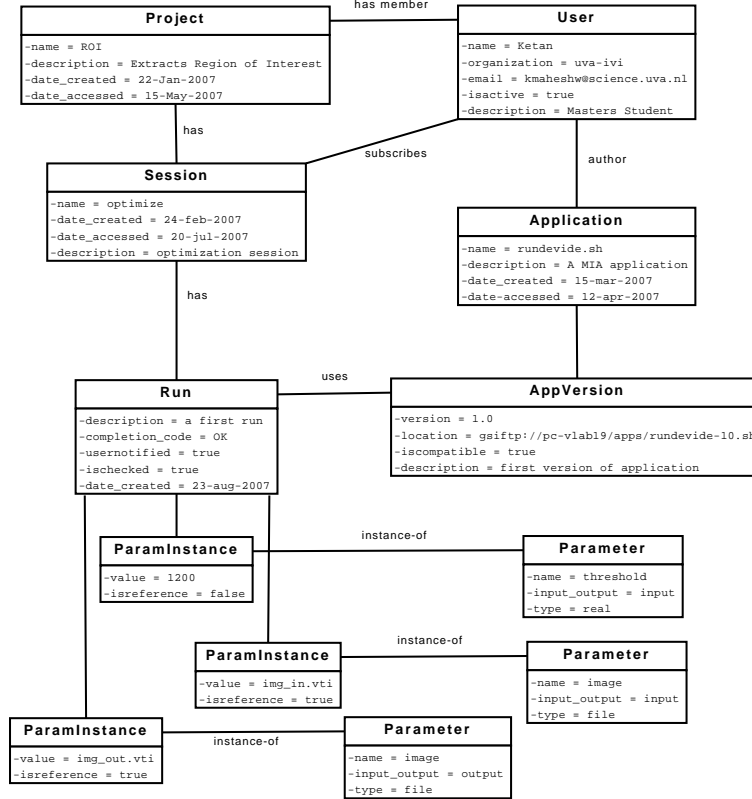


Figure 5.2: An Instance diagram of Implementation showing the values the classes/entities assume when in operation

5.5 APIs Functions Implementation

Each of the APIs as described in chapter 4 contains several functions each performing a specific task. Table 5.1 shows an overview of the key functions implemented and mapped to the APIs.

5.5.1 Functional Specification

Table 5.1 shows the mapping of APIs into functions. Several functions are overloaded with a similar signature but applying on different entities. Following is a brief description of these functions.

- **Create:** Create functions create an instance of the entity in question. This involves setting up of values of the properties of that entity and recording them into the database for future retrieval.
- **Update:** Update functions alter the values of one or more properties. Update method is either triggered by the user or triggered automatically based on events.

| API | Functions |
|------------------------------------|---------------------------------------------|
| Project Management | create/update/delete, subscribe/unsubscribe |
| Application Description Management | regapp, addversion |
| Application Execution Management | addrun, fetchrun |
| Data Provenance Management | get, find (User,Session,Application,Run) |

Table 5.1: APIs and Functions for the PSE implementation

- **Delete:** Delete functions are used to delete an entity.
- **Get:** Get functions are used to locate the id of an entity given one or more properties associated with it. It returns more than one entities if they match with the provided parameters.
- **Find** Find Performs a search based on the criteria given by the user. A basic form of find performs searching of entities and an advanced form performs a search on the associations among entities.

Apart from the above mentioned generic functions, several other functions with special functionality were implemented. Following list briefly describes these functions:

- **addrun** This function performs the recording of an execution into the system. The addrun is a batch mode function. The information recorded as part of an addrun function is the user, project and session associated with the run and parameters and user notification status for the outcome of this run. Depending upon the options given by the user, addrun triggers events associated with run and notifies the users subscribed via email.
- **fetchrun** As the name suggests a fetchrun action performs fetching of information related to the run to a local computer. The associated parameters are also downloaded from the remote locations. This information is fetched into a 'run-profile' file and includes information about associated user, project and session and parameters. Remotely located files associated with this run are downloaded into a designated directory of the user.
- **Register-application** This method registers the application into the repository. The application could be retrieved at a later time using its URL. Additional versions of the application can also be registered with the application.
- **adduser** This method enables a user to be subscribed to a project and session on specific events. Subscription information is used to notify users via email when the events occurs.
- **execquery** This method provides a user friendly facility to submit a free query to the system.

5.6 Grid-Service Implementation

A factory-instance pattern based Globus 4 compliant grid-service was implemented. The functions available with the service is described by a standard Web-Services Description Language (WSDL) document. The strategy of wrapping the code into grid-services and the *endpoint reference* of the service for invocation was used. This makes it convenient to separate the services

implementation code from the functions code as opposed to a tight integration of the programming logic into the service code. A separate service and function code also makes the programs easy to maintain and upgrade the service and functionality part independent of each other. The service here acts as a black box that performs the task upon invocation. Figure 5.3 depicts the invocation mechanism for the service. An authenticated client invokes the factory service. The factory service in turn creates an instance of the service. The instance serves the service call and is destroyed by the container at the end of the call.

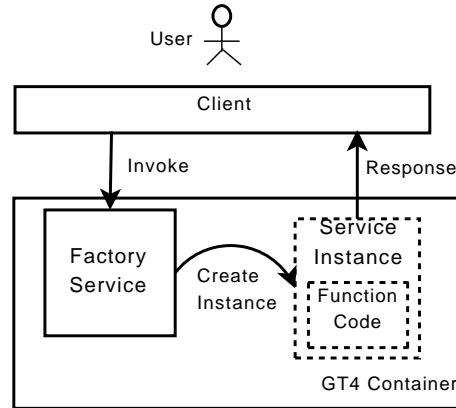


Figure 5.3: A Factory-instance pattern of grid-service invocation

5.7 Grid-Service Clients Implementation

Different clients performing activities corresponding to each API were developed. These clients are as follows:

1. *PrepareEnv*: An interactive menu-driven client that calls the services on the *Project Management* API in order to manage the project and related entities.
2. *Addrun*: A batch mode client that accepts parameters related to execution of application and calls the services in the *App Execution Management* API.
3. *ManageApp*: An interactive menu-driven client that performs the registration of applications and manages new versions of existing applications using services in the *App Description Management* API.
4. *Find*: An interactive menu-driven client that takes care of search and provenance management. This client invokes services that belong to the *Provenance Management* API.

Test Case

6.1 Introduction

The implementation was tested with a representative case that demonstrates a MIA-ADL over a long time period, say one year. An experimental setup is described followed by its handling of the system. Various activities are described with illustrations of the working of the environment and the snapshots of the client calls to the grid-services.

The environment will be used by calling the clients to perform various activities related to the MIA-ADL (see figure 2.1). The system is used for preparation before the actual MIA-ADL begins. Entities corresponding to users, projects and sessions are created. Usage of the system corresponding to each phase and feedback cycle is described as follows:

- **Component and Network Development:** The main activity here is to register the developed applications and their versions with parameters in the application repository.
- **Parameter Optimization:** The main activity during this phase is to add the runs to the system. During this phase one might want to search for the previous runs and rerun them.
- **Evaluation:** The activities during this phase are same as the above phase.
- **Clinical Deployment:** The main activity during this phase is to add the runs and to fetch the results of existing runs to see the stored results.
- **Feedback in Lifecycle:** The activities performed during the feedback in lifecycle correspond to the provenance requirements. The existing executions, their results, sessions, users and projects are tracked and the information is transferred (manually) to the previous phases.

6.2 Experimental Setup

An experimental setup was implemented to realize all phases of a simple MIA application using the proposed PSE architecture implementation. Both Nimrod and AMC-DWMS run DeVIDE networks in command-line mode. Before running the DeVIDE application, the data and DeVIDE network are staged in according to the Nimrod plan or the AMC-DWMS workflow. When the application is completed, the output results are staged back.

Example Application A region-of-interest (ROI) is segmented from an image using a threshold operation. A DeVIDE network accepts the image and a threshold range as parameters, and outputs a binary mask for the voxels within the chosen threshold range (see Figure 2.9) . Different ROIs can be selected by choosing a proper (optimal) threshold range. Here we focus on segmentation of blood vessels from contrast-enhanced CT Angiography scans of the head.

Development A DeVIDE network was developed to implement the segmentation application using available components in DeVIDE’s library. The network contains a VTK image reader, a ‘DoubleThreshold’ image processing operator, a custom ‘Slice3DViewer’ and a VTK image writer. During development, the user runs DeVIDE interactively, setting the TR on the GUI and observing the generated result in a viewer. As shown in figure 6.1, application registration action interactively performs the registering of a new application into the system. The system records the author of the application and application creation time. A new version can be associated to an existing application and registered into the system. The system keeps information about the version number, compatibility and author of the version.

```

====Main Menu====
1. Prepare Application
2. Add Version
3. Add Parameters
4. Quit

Choose one: 1

Application Name: rundevide.sh
Application Description: Commandline DeVIDE for ROI

Application prepared, registering ...

Application Registered.

```

Figure 6.1: Application Registration: An interactive client accepting the details of application and registering it

Project Management Activities The project management activities involves tasks such as create project, create user, create session, subscribe users to sessions etc. A user can subscribe to a project and session. This means that the user is subject to receive notifications based on the events he has been subscribed to the project and session. Common examples of events are OK, ERROR, ABORT meaning a normal execution, an error in execution and execution abnormally terminated respectively. Figure 6.2 shows an interactive subscribe to a session.

Optimization/Evaluation Nimrod was used to investigate the optimal TR to segment blood vessels. A parameter sweep on the upper threshold value was performed, while keeping the lower threshold fixed (1200 HU). Figure 6.3-left shows the Nimrod plan for varying the threshold between 1200 HU and 1600 HU. The data and executable are staged-in, DeVIDE is executed in command-line mode, providing the image, parameters and the network as arguments, and the output data is staged-out. The resulting images are visually inspected by an expert to determine the optimal threshold value to segment blood vessels (i.e. 1210 HU). In a similar fashion, an image sweep was performed with Nimrod to run the method with the optimal settings for many images.

```

====Main Menu====
1. Create Project
2. ...
....
6. Add User to Session
Choose one: 6
User Name: Ketan
User Description: Programmer
Session Name: Optimization
Subscribe to event (OK/ERROR/ABORT): ERROR
User Active?: true
    
```

Figure 6.2: User Subscription: A user is being subscribed to a session interactively. Note that the user is subscribed for the event “ERROR”

```

parameter THRESH integer range \
from 1200 to 1600 step 1;
task main
    copy ./rundevide.sh node: .
    copy ./dev_nw.dvn node: .
    copy ./img_in.vti node: .
    node: execute ./rundevide.sh $THRESH img_in.vti \
    ./test_network.dvn img_out.vti
    copy node: stdout stdout.${THRESH}
    copy node: stderr stderr.${THRESH}
    copy node: dev_nw.dvn dev_nw${THRESH}.dvn
    copy node: img_out.vti img_out${THRESH}.vti
endtask
        
```

Import Unit
DICOM Data

Worker Unit
DICOM to VTK

Worker Unit
MIA Network

Worker Unit
VTK to DICOM1

Export Unit
To DICOM Node

Notify Unit
Send Message

Figure 6.3: Nimrod plan for parameter optimization and AMC-DWMS Workflow performing a typical image processing operation in a simplified clinical setup

The executions as described above are added to the system using the `addrun` function. The `addrun` is a batch mode function that records the details of an execution into the system. The information recorded as part of an `addrun` command is the user, project and session associated with the run and parameters and user notification status for the outcome of this run. The `addrun` method accepts information related to the run as shown in figure 6.4 and registers the run into the system. In this case, a DeVIDE application running on a VTI image to obtain a region of interest is added by user with username “ketan” and as a part of “optimization” session. The ‘`true`’ option with the parameters specifies to save the parameters to a local folder.

Clinical Deployment AMC-DWMS was used to insert this application into a simulated clinical setting. The complete workflow consists of the following tasks: import the CT scan (DICOM) data from a server, convert it to the appropriate VTK data format, start the DeVIDE application with the optimal parameters, convert output from VTK to DICOM format, and export results to a DICOM node.

```

add-run \
--user name=ketan \
--project name=ROI \
--session name=optimization \
--app name=rundevide.sh, version=1.0 \
--param threshold;1100;ffalse,infile;./image-in.vti;ftrue, \
network;./testnetwork.dvn;ftrue;outfile;./image-out.vti;ftrue \
--status OK \
--notify true

```

Figure 6.4: Add Run: An example of a command-line version of the adding a run into the system; the switches are optional and can be provided in any order

Feedback in Lifecycle During the inspection of results produced during optimization, evaluation or deployment, a medical or technical expert can detect problems (e.g., degraded output, algorithmic instability). Using the information saved in the log files, it is possible to reload the DeVIDE network in interactive mode with the input image and parameters that generated the problem. The user can then add more components to the network (e.g. other 3D viewers) to comfortably investigate the situation at hand. The past executions can be retrieved using the fetchrun function. As the name suggests a fetchrun function performs fetching of information related to the run to a local computer. This information is fetched into a 'run-profile' (see figure 6.5) file and includes information about associated user, project and session and parameters. Remotely located files associated with this run are downloaded into a designated directory of the user. A registered run can be fetched through the interactive fetch-run. As figure 6.6 shows, the fetched run is saved in a 'profile' file marked with the id of the run fetched. The profile contains relevant information about the run downloaded and path to the associated referenced parameters downloaded. In this case the parameters associated with the run would be downloaded to the "/tmp" folder of local machine. In the case where multiple hits are found the system returns an error message.

```

run id=4
description= Run executes with ref params.
completioncode=OK
checkstatus=NOT CHECKED
username=ketan
sessionname=optimization
Projectname=ROI
Parameter=1200
Parameter=1600
Parameter=ftp://pc-vlab19/parameters/imagein.vti
Parameter=ftp://pc-vlab19/parameters/imageout.vti

```

Figure 6.5: A typical Run Profile generated as part of the fetchrun; Note the URL for the parameters that are referenced to a remote location.

```
====Main Menu====
1. Find run by attributes
2. Find run by user
3. Find run by session
4. Find run by application

Choose one: 1

Enter Description: test
Enter Completion Status: OK
Is user notified?: true
Enter Destination directory: file:///tmp
Fetching run done. runprofile: file:///tmp/run_06.pprofile
```

Figure 6.6: An interactive fetch run client fetching the specified run to a destination folder and creating a profile for that run.

Provenance The interactive find function is used to address the provenance requirements. A basic form of find performs the search on the entities. An advanced form of find performs the search on the associations of the entities. Several options for find are available. For example a user could be found based on the runs he is responsible for or the application he has authored. As shown in figure 6.7 one can interactively find a run based on the users who executed it. After selecting an appropriate option, a sub-menu allows to tune the search. The find function is connected to other functionality. For example a run found using the find method can be fetched in the same interaction using the fetchrun function. Figure 6.8 shows a free query executed on sessions. Free queries provide a flexible way of retrieving the information from the underlying database by querying the tables and their attributes directly.

```

===== Main Menu=====
1. Find Project
2. Find Session
...
6. Find Run
Choose One: 6
1. Find Run by Attributes
2. Find Run by User
3. Find Run by Session
4. Find Run by Application
Choose One: 2
Finding Run by User
Enter user name: Ketan
Enter User Description: Programmer

Runs found...

-----
id || completion code || description
--||-----||-----
6 || OK                || null
8 || OK                || MMBE final
12 || ABORTED           || DeVIDE First
-----

```

Figure 6.7: Usage of the find function to find the information about a run

```

Free Query
Enter the name of entity: session
List Attributes (q to end list):
name
description
q
select name, description from session ...

-----
name      || description
-----
devel     || development session
eval      || Evaluate
opt       || Optimize
-----

```

Figure 6.8: A flexible “Free query” facility: user provides an entity and attributes to find the sessions

Conclusions and Future Work

7.1 Introduction

The present work analyzed the MIA-ADL inspired by the system envisioned and proposed in [5]. The key features of MIA-ADL were studied:

- Phases in the MIA-ADL
- Feedback in the lifecycle
- Users and their activities
- Application Management
- Search and Provenance
- Support Systems

7.2 Conclusions

The requirements for a PSE were elicited that can realize the above features. An architecture for the PSE was proposed to facilitate the information flow among systems supporting different phases of a MIA application lifecycle in the VL-e project. The layered design and grid-service based implementation of architecture gives it several advantages:

1. **Simple:** A component based architecture that treats the support systems as external environment becomes simple to understand, implement and validate. The dynamics of the operations within the architecture does not affect and are not affected by the external systems.
2. **Robust:** The proposed architecture is robust and fault tolerant because of the underlying exception handling mechanisms of the grid services. A faulty call by the user or an external system is handled by the loosely integrated services and hence does not affect the performance or working of rest of the system.
3. **Flexible** Adding, removing and modifying the functionality becomes easier through the use of grid services as black box wrapper around the PSE functions.

4. **Secure** The Open Grid Services Architecture (OGSA) compliant grid services allows access with proper user authentication through user-credentials and delegations where appropriate.
5. **Lightweight** The architecture is lightweight as it excludes the legacy systems and their environment as external. This makes it easy to ship and deploy the services to a new environment.
6. **Extensible** Any number of new systems that are capable of interfacing the grid services can be added without making any changes to the architecture itself. This makes it easy to extend the PSE with adding new systems which in turn adds to the functionality and benefits the MIA-ADL.
7. **Scalable** Scalability can be classified as computational, storage and incremental scalability. With the interfacing of VL-e PoC resources the architecture can tolerate increasing concurrent calls and data storage loads. The proposed architecture is incremental systems scalable. This means that since independent calls to webservices generate a new instance any number of new systems does not affect the performance of the architecture.

With the prototype implementation and simple application presented in this work, we have demonstrated that it is possible to construct a PSE that assists across all phases of MIA development for clinical applications. The prototype implemented the APIs fulfilling the requirements imposed. A proof of concept was given by implementing these APIs and using the Grid-based services to expose them. The function of these services were tested through clients for a typical test case.

A new algorithm can be interactively developed as a MIA application in a rich graphical environment, then seamlessly transported through parameter optimization and evaluation phases, and finally deployed in clinical practice, all using a unified set of tools. At any stage return to a previous phase in the lifecycle is possible, with the prospect of facilitating the maintenance of MIA applications that meet the standards of health care.

To manage the heterogeneity of different development environments is a tedious and laborious task. The PSE can be used in a clinical environment supporting the existing clinical procedures through workflows. Lack of interoperability standards among diverse software systems makes it difficult to achieve a tightly coupled interoperable systems and requires a customized solution.

This work was started with a goal of facilitating an “active” integration among support systems in order to support information flow across the MIA-ADL. However, the support systems in question present entirely complementary functionality which are heterogeneous in nature. This prompted us to design an architecture that is more “passive” in nature. As an added advantage we get better extensibility as any number of independent systems can be hooked onto the architecture without any hindrance. This is more desirable since the MIA applications keep on evolving adapting new standards and tools. The data model can be converted into a more elaborated and flexible model by associating ontologies with the concepts. However, this may require more efforts in testing the correctness of the model and should be driven by strong requirements. The architecture presented in this work is implemented using grid based services. This makes it a strong candidate for integration with other larger systems in a large scale e-science project such as the VL-e.

The idea demonstrated in this work can be applied not only to the healthcare field but to such diverse research areas wherein a phased development and testing requires multiple independent and complementary systems.

7.3 Future Work

The current architecture provides a passive facilities for information tracking and management which requires manual intervention to some extent. An “active” and automated mechanism will improve the performance of the PSE. An integration of the current PSE with a larger system will facilitate smooth adaptation in the clinical setup. The system can be plugged-in a larger system in order to enrich its functionality.

In the current work a set of commandline clients were developed to satisfy the requirements. A rich graphical user interface with previews and virtual file select facilities will surely add immense value to the PSE. Development of generic interfaces in order to use the current PSE as a plug-in to an existing larger system will help a deployment with minimum efforts. A sophisticated mechanism to interface with the underlying database instead of the current ‘id’ based mechanism will make the working of the system fault tolerant as it will hide the database intricacies from he user.

Since the work was done in the context of masters thesis, an extensive and realistic test over a long period of time was not possible. We anticipate that the observation of system behavior over long period of time will give more insights on the requirements and could lead to a stronger and stable environment.

Bibliography

- [1] A. Dhawan, *Introduction to Medical Image Analysis*. Wiley Interscience, 2003.
- [2] R. Kikinis, S. K. Warfield, and C.-F. Westin, "High performance computing in medical image analysis at the surgical planning laboratory," in *High Performance Computing Asia'98*, (Singapore), pp. 290–297, September 22-25 1998.
- [3] "Virtual Lab e-Science: Towards a new Science paradigm. A BSIK Proposal," tech. rep., Wetenschap & Technologie Centrum Watergraafsmeer, 2003.
- [4] L. Hertzberger, "e-Science and the VL-e approach," in *Transactions in Computational Systems Biology IV*, vol. 3939 of *LNCS*, pp. 58–67, Springer Verlag, 2006.
- [5] S. Olabarriaga, J. Snel, C. Botha, and R. Belleman, "Integrated support for medical image analysis methods: from development to clinical application," *IEEE Transactions on Information Technology in Biomedicine*, January 2007.
- [6] C. Baru, R. Moore, A. Rajasekar, and M. Wan, "The SDSC storage resource broker," in *CASCON'98 Conference*, 1998.
- [7] C. Botha, "DeVIDE - The Delft Visualization and Image Processing Development Environment," tech. rep., TU Delft, May 2005. <http://cpbotha.net/DeVIDE>.
- [8] <http://www.vtk.org/>.
- [9] <http://www.itk.org/>.
- [10] R. Buyya, D. Abramson, and J. Giddy, "Nimrod-G Resource Broker for Service-Oriented Grid Computing," *IEEE Distributed Systems Online*, vol. 2, no. 7, 2001.
- [11] J. G. Snel and S. D. Olabarriaga and J. Alkemade and H. G. van Andel and A. J. Nederveen and C. B. Majoie and G. J. den Heeten and M. van Straten and R. G. Belleman, "A Distributed Workflow Management System for Automated Medical Image Analysis and Logistics," in *accepted to IEEE-CMBS special track on Grids for Biomedical Informatics*, 2006.
- [12] B. Ludscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the kepler system."
- [13] K. Schuchardt, B. T. Didier, and G. Black, "Ecce - a problem-solving environment's evolution toward grid services and a web architecture.," *Concurrency and Computation: Practice and Experience*, vol. 14, no. 13-15, pp. 1221–1239, 2002.

-
- [14] I. Taylor, "Triana generations," *e-science*, vol. 0, p. 143, 2006.
- [15] Y. J. Choi, T. Oroguchi, Y. Kato, M. Takeda, and Y. Tago, "Labgrid: Integrated problem solving environment system for high throughput computing," *e-science*, vol. 0, p. 103, 2006.
- [16] E. Gallopoulos, E. Houstis, and J. R. Rice, "Computer as thinker/doer: Problem-solving environments for computational science," *IEEE Comput. Sci. Eng.*, vol. 1, no. 2, pp. 11–23, 1994.
- [17] I. T. Foster, "Globus toolkit version 4: Software for service-oriented systems.," in *NPC* (H. Jin, D. A. Reed, and W. Jiang, eds.), vol. 3779 of *Lecture Notes in Computer Science*, pp. 2–13, Springer, 2005.
- [18] G. Allen, K. Davis, T. Goodale, A. Hutanu, H. Kaiser, T. Kielmann, A. Merzky, R. V. van Nieuwpoort, A. Reinefeld, F. Schintke, T. Schott, E. Seidel, and B. Ullmer, "The grid application toolkit: Towards generic and easy application programming interfaces for the grid," in *Proceedings of the IEEE*, vol. 93, pp. 534–550, March 2005.
- [19] A. Wibisono, D. Vasunin, V. Korkhov, Z. Zhao, A. Belloum, C. de Laat, P. W. Adriaans, and B. Hertzberger, "Ws-vlam: A gt4 based workflow management system.," in *International Conference on Computational Science (3)* (Y. Shi, G. D. van Albada, J. Dongarra, and P. M. A. Sloot, eds.), vol. 4489 of *Lecture Notes in Computer Science*, pp. 191–198, Springer, 2007.
- [20] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li, "Taverna: A tool for the composition and enactment of bioinformatics workflows," *Bioinformatics Journal*, vol. 10, no. 17, pp. 3045–3054, 2004.
- [21] Z. Guan, F. Hernandez, P. Bangalore, J. Gray, A. Skjellum, V. Velusamy, and Y. Liu, "Grid-flow: a grid-enabled scientific workflow system with a petri-net-based interface: Research articles," *Concurr. Comput. : Pract. Exper.*, vol. 18, no. 10, pp. 1115–1140, 2006.
- [22] <http://www.wfmc.org/>.
- [23] <https://forge.gridforum.org/sf/projects/ogsa-wg>.
- [24] D. C. Marinescu, "A grid workflow management architecture." White paper.
- [25] W. M. P. van der Aalst, , B. Kiepuszewski, and A. P. Barros, "Workflow patterns," *Distributed and Parallel Databases*, vol. 14, pp. 5–51, July 2003.
- [26] J. Yu and R. Buyya, "A taxonomy of workflow management systems for grid computing," 2005.
- [27] P. Wagstrom, S. Krishnan, and G. von Laszewski, "GSFL: A Workflow Framework for Grid Services," in *SC'2002*, (Baltimore, MD), 11-16 Nov. 2002. (Poster).
- [28] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, vol. 13, no. 3, pp. 219–237, 2005.
- [29] E. Deelman, S. Callaghan, E. Field, H. Francoeur, R. Graves, N. Gupta, V. Gupta, T. H. Jordan, C. Kesselman, P. Maechling, J. Mehringer, G. Mehta, D. Okaya, K. Vahi, and L. Zhao, "Managing large-scale workflow execution from resource provisioning to provenance tracking: The cybershake example," *e-science*, vol. 0, p. 14, 2006.

-
- [30] E. Deelman, “Mapping abstract complex workflows onto grid environments,” 2003.
- [31] I. Foster, J. Voeckler, M. Wilde, and Y. Zhao, “Chimera: A virtual data system for representing, querying, and automating data derivation,” 2002.
- [32] M. Hategan, G. von Laszewski, and K. Amin, “Karajan: A grid orchestration framework.” Supercomputing 2004, Pittsburgh, 6-12 Nov. 2004. (Refereed Poster).
- [33] *GridAnt: a client-controllable grid workflow system.*
- [34] D. Gannon, L. Fang, G. Kandaswamy, D. Kodeboyina, S. Krishnan, B. Plale, and A. Slominski, “Building grid applications and portals: An approach based on components, web services and workflow tools.,” in *Euro-Par*, pp. 1–8, 2004.
- [35] W. Emmerich, B. Butchart, L. Chen, B. Wassermann, and S. L. Price, “Grid service orchestration using the business process execution language (bpel),” *Journal of Grid Computing*, vol. 3, no. 3-4, pp. 283–304, 2005.
- [36] G. von Laszewski, “Java CoG Kit Workflow Concepts,” *accepted for publication in Journal of Grid Computing*, 2006.
- [37] *Grid Workflow*, ch. Grid Workflow with the Java CoG Kit. 2006. in preparation.
- [38] H. Casanova, G. Obertelli, F. Berman, and R. Wolski, “The apples parameter sweep template: user-level middleware for the grid,” in *Supercomputing '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, (Washington, DC, USA), p. 60, IEEE Computer Society, 2000.
- [39] N. Guarino, “Formal ontology and information systems,” 1998.
- [40] B. Yildiz and S. Miksch, “Ontology-driven information systems: Challenges and requirements,” in *Proceedings of the International Conference on Semantic Web and Digital Libraries (ICS-D-2007)*, Bangalore, India, 2007.
- [41] P. Buneman, S. Khanna, and W.-C. Tan, “Data provenance: Some basic issues,” in *Foundations of Software Technology and Theoretical Computer Science*, 2000.