Journal of Computational Science 5 (2014) 51–62

Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

Flexible composition and execution of large scale applications on distributed e-infrastructures

Stefan J. Zasada, David C.W. Chang[1], Ali N. Haidar[2], Peter V. Coveney*

Centre for Computational Science, University College London, 20 Gordon Street, London WC1H 0AJ, United Kingdom

ARTICLE INFO

Article history:
Received 18 February 2013
Received in revised form 9 September 2013
Accepted 24 October 2013
Available online 1 November 2013

Keywords:
E-infrastructure
High performance computing
Application virtualization
Usability

ABSTRACT

Computer simulation is finding a role in an increasing number of scientific disciplines, concomitant with the rise in available computing power. Marshalling this power facilitates new, more effective and different research than has been hitherto possible. Realizing this inevitably requires access to computational power beyond the desktop, making use of clusters, supercomputers, data repositories, networks and distributed aggregations of these resources. The use of diverse e-infrastructure brings with it the ability to perform distributed multiscale simulations. Accessing one such resource entails a number of usability and security problems; when multiple geographically distributed resources are involved, the difficulty is compounded. In this paper we present a solution, the Application Hosting Environment,[3] which provides a Software as a Service layer on top of distributed e-infrastructure resources. We describe the performance and usability enhancements present in AHE version 3, and show how these have led to a high performance, easy to use gateway for computational scientists working in diverse application domains, from computational physics and chemistry, materials science to biology and biomedicine.

1. Introduction

Today's computational scientists face a growing number of challenges which affect their ability to fully exploit the computational resources, made available to them via so called *e-infrastructures* (such as PRACE, EGI or EUDAT in Europe, or XSEDE in the USA). Firstly, they have an unprecedented amount of computational power available to them, which will continue to grow inexorably in the future, presenting many opportunities as well as challenges to an increasing number of scientific disciplines that rely on computer based modelling and simulation.

Secondly, the architectures of these large scale high performance computing (HPC) machines point to a growing trend of computers comprised of hybrids of scalar and vector processors [1,2]. This requires application scientists to ensure their code is optimized to take full advantage of the hybrid architecture of a specific machine. Grid computing [3,4] has sought to simplify end user access to and use of HPC resources, but the middleware tools developed to realize the computational grid concept have seldom provided the transparency and ease of use envisaged [5]. The challenges described above are compounded when one attempts to invoke multiple resources, in order to achieve more than just the sum of their individual parts [6].

Alongside grid computing we have witnessed the development of cloud computing. Cloud computing represents a fast growing business model that seeks to commoditize computational infrastructure, and provide access to various distributed resources such as CPU, memory and storage (known as infrastructure services) and applications (software as services). It is a rapidly growing area due to major strategic investments from global software companies such as Microsoft, Amazon, Google and IBM. Cloud storage today is growing in popularity, particularly due to its shared data at low cost capabilities. Nonetheless, there are many security and legal issues in cloud computing that are yet to be resolved.

The Application Hosting Environment [7,8] is a middleware layer designed to simplify the user's ability to exploit computational resources beyond the desktop, greatly facilitating the use of e-infrastructure. It has been deployed in support of a diverse set of projects, including HIV-1 protease modelling [9], immune system simulation [6], and large scale materials modelling [10]. AHE seeks to converge the Software as a Service model of cloud computing with high performance grid computing. In this paper we will discuss the concepts behind AHE, and describe in detail the latest version of the Application Hosting Environment, AHE 3.0, which has been reimplemented using RESTful services [11] rather than WSRF services [12]. We will demonstrate how the work we have done to redesign AHE 3.0 has led to a significant increase in performance

* Corresponding author.
E-mail address: p.v.coveney@ucl.ac.uk (P.V. Coveney).
[1] Now at: The Graduate School of Biomedical Engineering, University of New South Wales, Sydney, Australia.
[2] Now at: HSBC, Canada Square, London, UK.
[3] AHE is available to download under the LGPL license from: https://sourceforge.net/projects/ahe3/.

1877-7503/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.jocs.2013.10.009

compared to AHE 2.0 [8], and we show how this new version of AHE is benefiting various ongoing research projects.

2. Service oriented computational science

Virtualization is a broad term used in computer science to describe the abstraction of resources. Application virtualization describes a range of technologies designed to separate an application from the operating system that it runs on. In many cases this is achieved by introducing compatibility layers around underlying operating system features and libraries, for example the WINE system used to run Windows applications on UNIX systems [13].

The key aim of virtualization is to abstract away all the details of an underlying hardware or software system from the concern of the user. The benefits are manifold: developers can code to a single virtualized interface or system rather than for a specific hardware implementation; multiple virtual instances of a system can often be run side by side on a single physical system (in machine virtualization for example); and physical resources can be protected.

The growth of virtualization technologies, along with service oriented architectures (SOA), has also driven the development of cloud computing. The use of virtualized interfaces and systems means that the specific details of a cloud's architecture are hidden from consumers of the cloud resources. Several cloud computing models exist; the Infrastructure as a Service (IaaS) cloud paradigm typically takes the form of virtualized servers running on hardware platforms managed by the cloud hosting company, where each user is given access to one or more virtual servers, solely under their control. This also provides a degree of *elasticity*, as the number of virtual machines in a cloud environment can be greater than the number of physical servers available to the hosting entities. The Software as a Service (SaaS) cloud paradigm delivers access to applications centrally hosted on a cloud platform, typically via a web browser.

While virtualization technologies certainly reduce the complexity of using a system, and especially when working across multiple heterogeneous computing environments, they are not widely deployed in high performance computing scenarios. As its name suggest, HPC seeks to obtain maximum performance from computing platforms. Extra software layers impact detrimentally on performance, meaning that in HPC scenarios users typically run the applications as close to the 'bare metal' as possible. In addition to the performance degradation introduced by virtualization technologies, choosing what details to abstract in a virtualized interface is itself very important. Grid and cloud computing support different interaction models. In grid computing, the user interacts with an individual resource (or sometimes a broker) in order to launch jobs into a queuing system. In cloud computing, users interact with a virtual server, in effect putting them in control of their own complete operating system. Both of these interaction models put the onus on the user to understand very specific details of the system that they are dealing with, making life difficult for the end user, typically a scientist who wants to progress his or her scientific investigations without any specific usability hurdles obstructing the pathway.

To address these problems, we have developed a software layer designed to implement the Software as a Service cloud paradigm for scientific applications that rely on high performance computing, mediated by the *Application Interaction Model* which we describe in Section 3, derived from the user requirements also discussed in Section 3. This model is based on the insight that many e-infrastructures impose a steep learning curve on the majority of end users, who do not possess the technical expertise for the most part to compile, optimize, install, debug and finally launch their applications; they simply want to run their applications, obtain results and focus on their scientific endeavours. While an application may consist of a single execution of a computational code, it could also consist of a complex set of operations involving multiple codes, connected as a workflow; AHE enables all kinds of applications to be treated as simple "atomic" units, helping realize the original vision of a grid as "distributed computing performed transparently across multiple administrative domains" [14].

3. User requirements

For supercomputer class applications, the user generally has to install his/her own application, if that application is not one of the few community codes pre-installed on the machine; it is not possible simply to stage an executable to the target resource as it requires too much bespoke tailoring to the particular hardware setup of the resource. Generally a group of researchers will want to use the same application on a resource. However, many users will not know where a particular application is installed on a target system, nor will they necessarily know the best way to run the application on a particular system. Often, with supercomputer class systems, applications have to be run in specific ways to achieve the best performance. The community's expert users must spend time educating other users on the vagaries of different queuing systems and machines. Typically, the end user will need to stage data to the supercomputer before he/she is able to execute her application. Therefore, the supercomputer must provide accessible interfaces over which data can be staged. In order to launch an application, the users have to prepare a description of the job that they want to run, which is submitted to the queue management system on their target resource, in a format that the queue management system understands and which is potentially incompatible with other instances of the same queue management system running on other resources. Once the job has been submitted, users monitor the progress of their jobs through the queuing system, using interfaces provided by the resources.

Distributed applications can consist of multiple computational codes launched on multiple resources, connected together as workflows of operations, as well as single codes launched on single resources. Applications can get their data from multiple sources, such as online data repositories and databases, and store their output data in similar resources. Typically, users will be given allocations of time on individual grid resources, or the e-infrastructure as a whole, through awards made to their project's principal investigator. These allocations will have a notional associated cost, the cost per CPU hour, derived by the resource operator from their running costs and a projected resource utilization. Such allocation models inhibit the most creative use of and ways of exploiting distributed e-infrastructure.

The scientific end user's primary concern is running their application in a timely fashion, in order to obtain results that further their scientific objectives. All the services and facilities provided by a grid should be subservient to this end. Typically, the user does not even care which machine on the grid their application is run on, as long as results are delivered within a time frame that makes them useful, whether that is the time to publish a scientific paper, or the time to conduct a potentially life-saving medical simulation [15].

A further problem faced by end-users and administrators of computational e-infrastructures arises in connection with the usability of the security mechanisms usually deployed in these environments, in particular identity management. Many of the existing computational grid environments use Public Key Infrastructure (PKI) and X.509 digital certificates as a cornerstone for their security architecture. However, it is well documented that security solutions based on PKI lack user friendliness for both administrators and end-users, which is essential for the uptake of any grid security solution [16,17]. The problems stem from the process of acquiring X.509 digital certificates, which can be a lengthy one, and generating proxy certificates to get access to remote resources as part of the authentication process [17]. As a result, many users engage in practices which weaken the security of the environment, such as the sharing of the private key of a single personal certificate, to get on with their tasks.

From many years of working with high-end computing applications, we have been able to derive a set of requirements for a system designed to make the use of distributed and HPC applications transparent to the end user. Current e-infrastructures focus on submitting jobs to batch schedulers on computational resources, meaning the user has to interact at both job and resource levels. Since users' predominant interest is running their application within a useful time frame, the first requirement of our usability model is that it should promote applications as a first class resource concept. *All user interactions should be with the application, rather than the machine, scheduler and job.*

Current e-infrastructure job submission mechanisms put the onus on the user to manage and curate their application's output data. Our model preserves the full state of each instance of an application, including all parameters and data used to launch the application, and all simulation output. This assists with tracing the provenance of simulation results, and is key to simulation reproducibility. Current grid middleware tools require the user to perform a number of steps in order to launch their code. Our model reduces the number of steps required to the minimum number possible in order to successfully run an application.

Current systems require the user to generate complicated job description documents in order to submit their application. Our model allows the user to launch their application using the simplest set of requirements possible, and takes care of generating whatever job descriptions the middleware on the underlying resource requires [17]. Users of supercomputer class resources may have access to a number of such resources via different computational grids, running different grid middleware stacks, requiring them to learn how to use different middleware tools to submit their jobs [18]. Our model presents a uniform interface to users to access resources running different middleware stacks, allowing them to transparently access not only single grids but also federated resources from multiple grids.

3.1. The Application Interaction Model

Based on our user needs analysis we have derived the Application Interaction Model, designed to allow users to easily control virtualized applications running on remote e-infrastructures. Traditionally, HPC focuses on the concept of 'jobs' to describe distinct workloads submitted to a batch queue. We purposefully focus on the concept of *applications*. An application is a higher level concept than a job; although an application could be realized by a single HPC job, it could equally correspond to a coupled simulation, where two codes (launched as two HPC jobs) pass parameters between themselves, or a steered application which requires steering Web services to be initialized before the code is launched, or a workflow of arbitrary complexity. However the application is composed the user should still interact with a single entity to control the execution of all components of the application.

We define the Application Interaction Model as follows:

(i) The virtualized application is the central entity in the Application Interaction Model.
(ii) An application does not necessarily correspond to a single computational code – it could be composed of multiple computational codes linked together in a workflow, or a computational code and associated steering Web services. However, it is presented to the user as a single application.
(iii) The application encapsulates all of the details of how to launch it, such as where the binaries that constitute the application are located, how to interact with individual resources and so on. These details are shielded from the user, who does not need to know anything about the underlying details.
(iv) Each instance of an application is controlled by a separate application instance, through which it is controlled. The application instance encapsulates all of the state associated with that run of the application, such as the input and output data, the application parameters and so on.
(v) All user interaction occurs through the virtualized application instance, which causes the computational code(s) that constitute the application to be launched on back-end computational resources.
(vi) Operations on the application instance allow the user to stage data associated with the application to the resource where it is needed, launch, monitor and terminate the application. These operations have an effect on the codes running on remote grid resources.

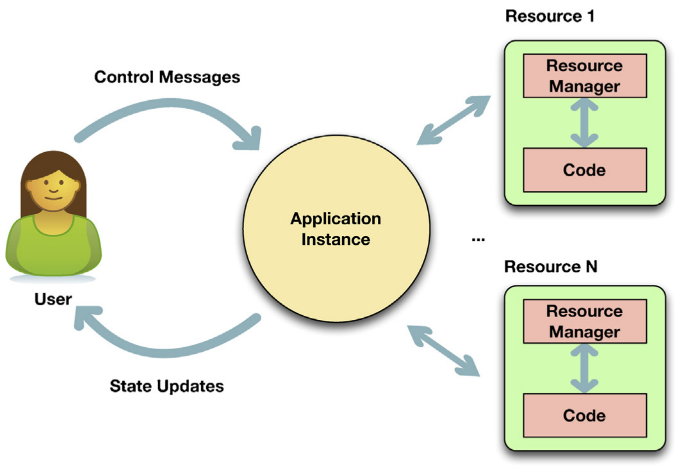


Fig. 1. The Application Interaction Model underpinning AHE. The application instance is the central entity representing each instance of an application that a user launches. All user interaction is mediated via the application instance, which supports operations to launch, monitor and terminate the application, and to manage data sharing.

A schematic representation of this interaction model is shown in Fig. 1. The principal motivation behind this approach is to simplify the use of e-infrastructures, by introducing an abstraction layer between the users and the high end computing resources available to them which hides the complexity of the latter, providing an abstract interface to scientific applications deployed on a grid. This abstraction layer takes care of the process of launching the application on one or more HPC resources, and reduces the interaction with an application to those operations most relevant to the user.

The Application Interaction Model implies that the task of deploying and configuring an application is taken care of by a system administrator, or a community's 'expert user'. This draws a parallel with many different communities that use applications on high performance computing resources, such as the UK Collaborative Computational Projects (CCPs) [19], where a group of programmers develop a code, which they then distribute to an end user community. Once the expert user has configured AHE to share an application, end users can invoke clients installed on their workstation, tablet or mobile phone to launch and monitor the application across a variety of geographically distributed computational resources.

4. The Application Hosting Environment

The Application Hosting Environment (AHE) is our implementation of the Application Interaction Model. AHE is based on two key concepts to promote usability: application virtualization and community application sharing. Application virtualization allows developers to code against a single virtualized interface instead of the specific underlying software or hardware system, and it also allows multiple virtualized instances to be run side by side on a single physical system. The concept of an application in AHE does not necessarily imply a single computational code executed on a single HPC resource. A virtualized application in AHE can be comprised of more complex workflows, such as coupled simulations where multiple applications are required to pass data to each other, for example coupled quantum and molecular level simulations, made up of separate codes that exchange data via files [20].

AHE is built around the idea of a community model. In this paradigm, an expert user is required to set up and configure AHE with details of a scientific application, the distributed (grid) infrastructure it is deployed on and then uses AHE's RESTful interface to share this scientific application transparently with a group of end users. The resources that a particular application is deployed onto are chosen based on the characteristics and requirements of the application. An end user can then launch and monitor applications through the AHE desktop GUI client, web client or command line client (described in Section 4.2), and any combination of these clients can be used simultaneously. AHE has gone through several software releases, principally AHE 1.0 [7] and AHE 2.0 [8]. AHE 3.0, which we describe in this paper, adds many significant new features to those found in AHE 2.0, with an unremitting focus on usability and reliability.

4.1. The architecture of AHE 3.0

AHE 3.0 is a complete re-implementation of AHE 2.0 in Java (AHE 2.0 and earlier versions were implemented in Perl). AHE 3.0 introduces a new workflow engine based on JBPM [21] allowing complex workflows to be created and integrated into AHE, and accessed by users as single applications. AHE 3.0 also incorporates an object relational mapping framework using Hibernate [22], which simplifies installation and development of AHE by decoupling AHE from the database used to maintain state. A RESTful web service interface based on the Restlet [23] library furnishes a simple and concise HTTP based interface for clients to access AHE services, compared to the WSRF [12] based services used in AHE 2.0.

AHE 3.0 is a departure from AHE 2.0 and earlier releases, having undergone a complete redesign. AHE 3.0 comprises a number of modules which implement the core functionality. AHE runtime controls the start-up and shut-down of the AHE application life cycle; AHE engine implements the core functionalities including the workflow engine as well as facilitating interactions between the different components; AHE connector module implements the functionality required to connect to different types of middleware; AHE security module handles the security component as well as user management of AHE application and grid middleware; AHE interface module provides a RESTful Web service interface as well as command line access to AHE; the file module provides mechanisms to transfer files between different storage resources using GridFTP. A schematic showing the interaction of these different components is presented in Fig. 2.

RESTful Web service provides a simple abstraction of AHE's functionalities to the user by exposing AHE components as resources, each of which is identified by a global identifier (URI). This provides a clean and simple mechanism for end users to access AHE, making client tooling less complicated; it also means that AHE 3.0 can either be deployed via a servlet container such as Tomcat, or as a standalone server. A detailed discussion of the AHE 3.0 server components is presented below.

4.1.1. AHE runtime module

The AHE runtime module is responsible for starting up and shutting down the server in the standalone mode. It also initializes all the components and user configurations as well as the basic registry information relating to users. In standalone mode, AHE uses an embedded Jetty Server to provide web server functionality, including HTTPS with mutual user certificate authentication. In this mode, AHE 3.0 can be started from an executable file. AHE can also be deployed as a Java servlet into a servlet compliant server such as Apache Tomcat [24]. AHE can be deployed with an embedded H2 database [22] or use an external database through the Hibernate framework. More details about the deployment are discussed in Section 6.

4.1.2. AHE engine module

The AHE engine module encapsulates the functionality through which AHE 3.0 virtualizes access to scientific applications. It provides methods to create an Application Instance object, used to represent an instance of a virtualized application. In addition, methods are provided to run and maintain the execution workflow for each virtualized application instance. The workflow describes how the data and computational code(s) associated with this application instance are processed, including details such as the back-end connector (cf. Section 4.1.3) and security mechanism to use.

The AHE engine module also allows higher level workflows to be implemented. These workflows can control multiple application instances to create parameter sweep applications or complex chained application scenarios, in which data created by an application is used as the input for a second one and so on.

4.1.3. AHE connector module

The connector modules provide a set of classes that invoke external Java libraries which allow AHE to act as a client to distributed resource managers (DRMs) such as Globus GRAM. The connector module provides a generic Java interface (using the adapter pattern) which adapters for different DRMs have to implement. This Java interface is used by AHE 3.0 to access external computational resources, providing a loosely-coupled relationship between AHE and external client libraries. Connectors currently exist to allow AHE 3.0 to run applications via Globus 5.0 [25], Unicore 6 [26] and QCG OGSA-BES [27]. Each connector implementation translates the AHE's internal application state model into specific directives to the relevant DRM, such as the number of cores to use. The extensible interface framework means that interfaces to other DRM systems can easily be added as necessary. Each connector module is responsible for trapping errors from the underlying DRM and mapping it to an AHE error state, which is presented to the user.

4.1.4. AHE interface module

This module contains library code used to interface with AHE 3.0. This includes a bridge between the RESTful Web service interface (cf. Section 4.1.7) and the AHE runtime module (cf. Section 4.1.1). The AHE REST Web services exposes the main AHE functionalities and components as resources which can be controlled by performing operations on those resources.

4.1.5. AHE security module

The AHE security module provides a number of important functions, including user management, authorization and authentication control, platform credential management and integration with Audited Credential Delegation [28] (see Section 5). AHE provides a mechanism to delegate security control to ACD; these security functions include user authentication and management as well as virtual organization support and proxy generation for any specified virtual organization. In ACD mode, AHE contacts ACD

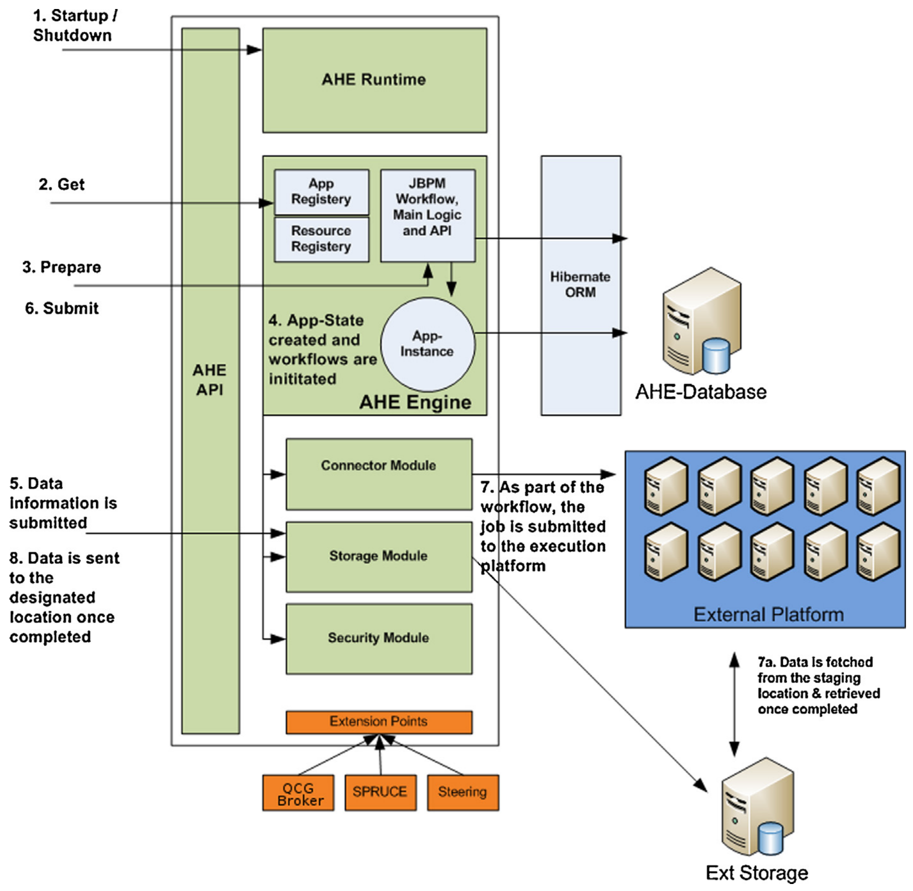

1. Startup / Shutdown
AHE Runtime
2. Get
App Registery
Resource Registery
JBPM Workflow, Main Logic and API
3. Prepare
6. Submit
Hibernate ORM
AHE-Database
4. App-State created and workflows are inititated
App-Instance
AHE Engine
AHE API
5. Data information is submitted
8. Data is sent to the designated location once completed
Connector Module
Storage Module
Security Module
7. As part of the workflow, the job is submitted to the execution platform
External Platform
7a. Data is fetched from the staging location & retrieved once completed
Extension Points
QCG Broker
SPRUCE
Steering
Ext Storage

Fig. 2. The architecture of the AHE 3.0 server, showing the relationship between the different software modules.

using RESTful Web service calls to authenticate users as well as to request the generation of proxy credentials on a per user basis. AHE itself is also able to authenticate users via SSL certificates or the more standard username/password credential combination. The security module is able to perform command level authorization, as well as platform credential management such as maintaining private key and certificate information for a user which may be required for him/her to be granted access to particular computational resources and data. Additionally, the security module is able to request updated proxies from a MyProxy server when a certificate is about to expire.

4.1.6. *AHE transfer module*

In AHE, input data is transferred directly from a location specified by the user to the computational resources where an application is to be run. Once a job is completed, the AHE server takes care of staging any output data back to the user specified location. The AHE Transfer module provides a mechanism to set up the security credential used to authenticate transfers and then initiate transfers between two different storage components through the AHE Java file transfer interface. Different transfer mechanisms have been implemented using this Java interface. Currently, file transfer functionalities make use of the jGlobus and UCC libraries to stage data using the GridFTP [29] and Unicore transfer protocols. The Java interface makes it easy to add new transfer protocols in future should they become necessary.

4.1.7. *RESTful web service*

One of the main features of AHE 3.0 is the implementation of the RESTful Web services interface, which replaces the WSRF interface used in AHE 2.0 and earlier versions. RESTful Web services expose resources addressable via HTTP and operated on using HTTP operations such as POST and GET. This provides a secure and straightforward universal end point for AHE to provide services to users. AHE 3.0 uses the Java Restlet library for its RESTful implementation [23]. The Restlet library was chosen to underpin the AHE 3.0 server due to the many features it provides, including the ability to develop services that run as standalone services or which can be deployed in a servlet container such as Tomcat (using either the J2SE or the J2EE version of the library), multiple native data representation formats such as XML and JSON, and scalability as well as security support.

The AHE REST command structure is grouped into a number of resources including: User, AppInstance, AppReg (application registry), Resource, PlatformCredential and Cmd (general commands). Each of these resources can be viewed or modified using the GET, POST or DELETE HTTP operations when applied to a suitable AHE resource URI. A typical AHE URI consists of several components; the domain URI followed by the user identifier and the AHE resource that will be operated on. The URI is followed by the command and argument if required.

4.1.8. *AHE workflow engine*

A key component of AHE 3.0 is the workflow management system built on the JBPM framework. JBPM is a lightweight Java workflow engine, with workflows described using the Business Process Modelling Notation (BPMN) 2.0 specification which calls specific Java classes, scripts or Drool rules to perform arbitrary functions. This allows new complex workflows and scripting functionalities to be introduced quickly to extend AHE. JBPM supports workflow persistence using the Hibernate framework meaning that, in the event of a server crash, the workflow can be recovered quickly and seamlessly. There is also a wide range of tools available to plug in to the JBPM framework, including workflow editors, which eases the integration of JBPM with AHE.

By using a workflow engine, further functionalities and workflows can be introduced into AHE applications. This allows the expert user to tailor customized workflows to complex tasks, such as coupled model applications. It also allows additional features or functionalities to be added, such as fault tolerance, and to integrate AHE with external services such as SPRUCE [30] in order to submit urgent computing jobs, and RealityGrid Steering [31] which allows scientists to interact with running applications.

4.2. AHE client

The AHE server maintains all state information about a particular application instance. This means that client tools need to store no information about individual application runs, and consequently very simple clients containing little configuration data can be created. It also means that clients can interoperate, with one client used to launch an application and another used to monitor it for example.

The simple REST endpoints exposed by AHE 3.0 server mean that in practice any tool which can perform HTTP POST and GET operations (such as the UNIX `curl` command) can be used as clients. However, a Java client API has been developed which not only provides methods to call AHE 3.0 server commands, but also provides auxiliary functions such as data staging. This API has been used to produce both graphical and command line clients. It also allows applications hosted in AHE to be accessed from high level tools, and integrated with workflow engines such as GridSpace and Taverna (see Section 8).

In addition, an AHE web client has been developed to provide a simple interface for the end user when interacting with the AHE server via a web browser. The web client interface has been developed using the Google Web Toolkit (GWT) and communicates with the AHE 3.0 server through its RESTful interface. The web client can be deployed on Java servlet compliant servers such as Tomcat or JBoss AS. The web client also allows the user to administer and configure an AHE 3.0 server, providing capabilities to manage users, certificates, applications, target computational resources and the server itself. The web client also allows end users to transfer files, launch and monitor AHE jobs through a web browser.

4.3. AHE application life cycle

AHE manages the whole life cycle of an application when invoking AHE, from input data staging, through job execution, to output data staging; during this process the application transitions through a number of different states. This life cycle is shown in Fig. 3.

The process starts when a prepare command is received by AHE. This puts AHE in a waiting mode, allowing the user to set up additional configuration details required for the application or workflow submission. Once a start command has been submitted, AHE server proceeds to first stage any input data that the user has attached to the application instance; once that is completed, it is then submitted to the execution platform. Once the application has been submitted to an external execution platform, AHE goes into a polling state, checking regularly for the completion of the application. When the job has completed, any output data is staged to the location specified by the user and the job submission process comes to an end.

If errors occur during certain stages of the AHE workflow process, AHE captures the error and allows the user to fix this error and

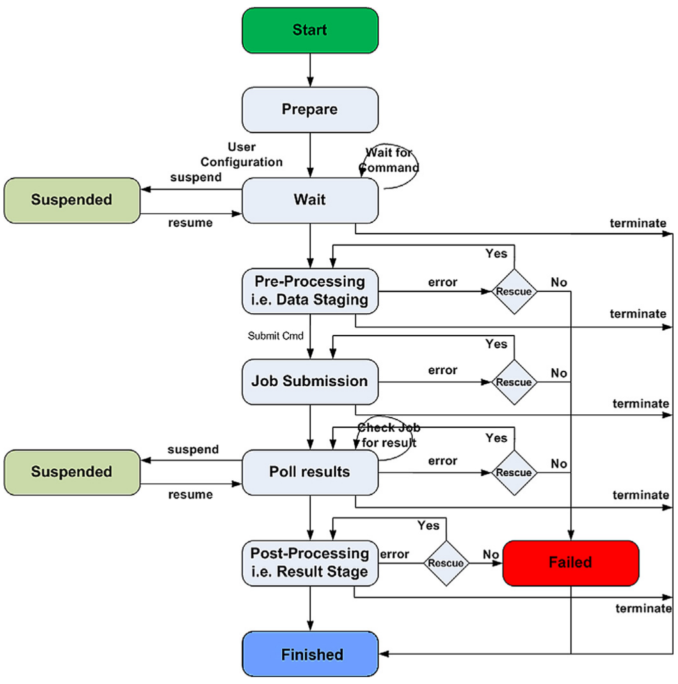


Fig. 3. The application life cycle. AHE server manages the transition of an application instance through a number of states, in order to stage data, execute an application, and handle failures.

attempt to execute the same step again. This workflow is modelled and executed using the JBoss jBPM workflow library and additional components can be added to the workflow if necessary.

In practice, a user has to go through the following steps in order to run an application:

(i) AHE runtime initializes all components, populates the internal data structures and ensures that the data held in the state database is synchronized with the AHE data structures.
(ii) The user queries the application registry to see what applications are available.
(iii) The Prepare command is issued which tells AHE Engine to create a persistent App-Instance Object that keeps track of the status and state of an executing application, which in turn initiates the AHE workflow process. An App-Instance object is a representation of a virtualized application initiated by the user. This allows AHE to keep track of the state and progress of the virtualized application.
(iv) This App-Instance object is persistent and stored in a local database using the Hibernate Framework, which allows AHE server to be database agnostic. In particular:
    (a) the App-Instance object is associated with a user/group and has a unique identifier;
    (b) active App-Instance data/objects are held in a registry and checked by AHE Engine to see which processes can be operated on each App-Instance, such as when and how they can be run, when data can be checked or retrieved, and so on.
(v) Input data files required by the application are staged to the target resource. AHE Server records the location and transfer protocol specified for each individual data file and passes that information to the relevant connector module so the job manager knows how to stage the data and retrieve the results if necessary.
(vi) The user next issues the submit command.
(vii) AHE workflow module then schedules the execution of the application using JBPM and quartz scheduler [32]. This allows

complex workflows to include asynchronous tasks, as well as multi-thread/concurrency support.

(viii) AHE Engine deals with the security interface requirements and submits tasks to external execution platforms. AHE polls the external execution platform (if it is configured to do so) and retrieves any output data once the application is completed. JBPM allows additional features to be added in order to create more complex workflows incorporating AHE plug-in components. JBPM is persistent so that all events are logged. If the server crashes, the workflow state stored in a database can be retrieved and reinitialized.

(ix) Once the application has completed, the data is retrieved and sent to the scratch disc (temporary file storage) or copied to an external storage resource specified by the user, allowing him/her to access it.

5. Securing AHE 3.0 with audited credential delegation

Efforts to address the usability of e-infrastructures are hampered by existing security mechanisms imposed on users. Typically, these require a user to obtain one or more digital certificates from a certificate authority, as well as to maintain and renew these certificates as necessary. The difficulty in doing this leads to widespread certificate sharing and misuse and a substantial reduction in the number of potential users [16]. In order to remove this barrier, we have coupled AHE 3.0 to Audited Credential Delegation (ACD) [28]. ACD is a usable security system that accommodates the security requirements of both end-users and resource providers, offering facilities to authenticate, authorize and audit all transactions.

The main advantage of ACD is that it entirely removes the use of digital certificates from end-users' experience, minimizing the usability problems caused by such credentials while addressing resource providers' concerns regarding securing access to their shared resources, tracing the users responsible for performing specific tasks on their resources. ACD enables users to invoke security credentials they are familiar with such as their institutional username/password combination (using Shibboleth [33], for example); assuming that they are authenticated it issues a digital certificate to them when necessary in the background.

ACD can be used to set up multiple virtual organizations (VO) to manage dynamic groups of users wishing to access e-infrastructure based resources, and to provide VO administrators with tighter control of users' actions as well as identity management. Existing solutions such as MyProxy [34], Shibboleth, and SARoNGS [35] on their own only provide credential repositories to store short lived X.509 certificates (MyProxy), web based single sign-on (Shibboleth), and web portals to access grid resources using a combination of Shibboleth and VOMS [36] (SARoNGS). None provides a holistic VO controlled security solution in the way ACD does.

The design of ACD is based on the concept of wrappers. A wrapper is a connector between a component and the outside world. It enables controlled access to the functionalities of a component. The ACD security wrapper comprises authentication, authorization and auditing components. Any request by a user to perform an operation using a service secured by ACD is intercepted by the security wrapper to establish the identity of the requester, to check whether or not the user is allowed to perform the task, to record the results of these checks in an audit log, then to perform the task in the distributed environment and, finally, to return results to the user. ACD has been developed in Java and exposes a RESTful interface. This allows its integration with any tool developed in a programming language that is capable of accessing RESTful services. Prior to its implementation, a model of ACD was developed based on formal notation [37], which is used for building safety critical systems, using the recommendations of the Open Web Application Security consortium for developing secure software [38]. This provides rigorous validation of ACD's security features.

The principal features of ACD's architecture are described below.

- *Local authentication service*. The current implementation supports a username-password database specifically for non Shibboleth ACD support. To be authenticated, a user has to provide a username/password pair that matches an entry in the database or use their local institution credentials via Shibboleth. To avoid known vulnerabilities involving usernames and passwords we adopted OWASP best security practices [39] such as storing passwords in encrypted form, rejecting weak passwords chosen by users, forcing the password length to be a minimum of eight characters including special characters, and changing the password on a regular basis. The Shibboleth support in the latest version of ACD provides users with more familiar authentication mechanisms. Shibboleth is currently used by many universities in the UK, EU and beyond to allow students and researchers to access online publishers' resources by invoking their local institutional username/password credentials.
- *Authorization component*. This component controls all actions performed in the VO. It uses the Parametrized Role Based Access Control (PRBAC) model in which permissions are assigned to roles [40]. The VO policy designer associates each user in the VO with the role that best describes his/her job function. The policy is defined at VO set up because it depends on the VO functionalities. The tasks (permissions) assigned to roles are drawn from the VO functionality.
- *Credential repository*. This component is responsible for managing the delegation of identity from the user to ACD via a proxy certificate. It stores the certificates acquired by the VO administrator (known as robot certificates) and their corresponding private keys in order to communicate with the target e-infrastructure. To allow the members of a VO access to computational resources, the VO is assigned a digital certificate, which is used behind the scenes to authenticate requests issued by the VO at the resource provider site. The component also maintains a list of issued proxy certificates (delegated identities), their corresponding private keys and the association between users and proxies in order to trace which proxy was used by which user.
- *Auditing component*. This component records all actions within the VO, including authorized and unauthorized requests to perform tasks within the VO, the username that requested them, the number of login attempts and login times. This allows the VO management to identify those ACD users responsible for having performed any tasks within an e-infrastructure environment.

5.1. Integration with the Application Hosting Environment

When run without ACD, the AHE security model requires each individual user to have a digital certificate, which carries with it the need to go through a lengthy credential acquisition process. To remove the need for such a certificate, we have integrated ACD with AHE. The first step of the integration requires understanding the interaction between AHE and ACD, in other words the functional and administrative tasks that can be performed within the integrated system. The administrative tasks offered by ACD include VO creation, certificate assignment, adding users, resetting user passwords, creating user roles, assigning tasks to roles, and assigning users to roles. The functional tasks offered by AHE include: Prepare Job, Submit Job, Monitor Job, download and Terminate Job. AHE's functional tasks are the same as the tasks permitted for any authorized user on a computational resource that uses the Globus or UNICORE middleware, for example. Therefore, the permissions assignment to the VO is done by the resource owner first, then the

VO administrator re-assigns these permissions to the roles in the VO according to the VO authorization requirements.

In the integrated ACD + AHE environment, the authorization requirements determined by the VO administrator are expressed through the introduction of two roles: VO Administrator and Scientist. The former is permitted to perform all the administrative operations above in addition to terminate, monitor and download any job submitted to a resource. The latter is permitted to perform all AHE operations in such a way that a person who submitted a specified job can only perform AHE functional operations on this application. As a result, two users running applications invoking different data will not be able to view the results of each other's activities. In addition, the scientist's role only permits a user to change his/her own password.

The construction of a VO requires that a system administrator goes through the process of acquiring a digital certificate. Once done, the VO administrator creates a VO and assigns the certificate to the named VO using the ACD client. It then becomes possible to add users instantly to the VO and give them genuinely transparent access to e-infrastructure resources. To illustrate how this system works, consider a user named "John Smith" who is a member of a research group in a UK university and would like to use UK National e-Infrastructure Service (NES) [41] resources to run scientific applications using AHE. The user contacts the local VO administrator and requests to join a specific VO. The user can opt to use their local username and password, if their institution is part of the Shibboleth Federation, or requests the creation of a dedicated ACD account. The VO administrator assigns the user to the "Scientist" role described above and assigns the user to a VO that has an access to e-infrastructure resources. The communications between the AHE + ACD client and the wrapped AHE server, as well as between the latter and the grid resources, are protected by the SSL security protocol.

In order to launch an application on a computational resource, the user invokes a request to perform the "Submit Job" task using an AHE client with ACD extension. This request is intercepted by the ACD authentication component that checks if the username and password match an entry in the database or can be authenticated against Shibboleth. The result of the authentication is recorded in the auditing component. The role of the user is picked up from the authorization component, in this case "Scientist". The authorization checks whether the "Submit Job" operation is permitted for the "Scientist" role held by the user. The result of the access control check is recorded in the audit log, and the operation "Submit Job" is invoked on AHE server. Once the request is granted, ACD picks the certificate associated with the VO the user wants to use and checks whether the user is assigned to this VO. If the check is successful ACD generates a proxy certificate from the VO assigned certificate, uploads it to a MyProxy server and records the issued proxies in the credential repository.

ACD then sends the randomly generated username/password pair needed to access MyProxy to the AHE server to download the session proxy. Finally, the AHE server sends the request to the computational resource site along with the proxy. At the target site, the proxy is validated. Certificate authentication succeeds, and the distinguished name on the proxy (VOName) is checked against the resource's authorization system to determine the role of the VOName, which is Scientist. Since this role is allowed to submit an application to resources the task will be invoked. From the e-infrastructure's perspective, it is the VOName that submitted the task, not "John Smith". In order to find out who invoked the "submit job" task on the resource using a specific proxy, the resource administrator passes the public key of the proxy to the VO administrator who can identify the name of the user.

In this way, requests from within the combined ACD/AHE system can be audited. It is thus possible to identify legitimate users and to ensure that only such users are allowed access to resources, in conformance with the policies enforced by the e-infrastructure management. In addition, it is possible to detect unauthorized attempts to access resources from within the VO and to identify persons responsible for such attempts. This form of accountability is an essential requirement for resource providers to be prepared to accept the ACD security model.

6. Deployment of AHE 3.0

AHE can be deployed as a standalone application via the Jetty Server using an embedded database or, in a more complex environment, AHE can be deployed as a Servlet hosted within a Servlet compliant server such as Apache Tomcat and configured to use databases supported by the Hibernate framework.

In the simplest configuration, the standalone mode allows AHE to be executed as an application which launches the Jetty Server with the option of invoking an embedded database or any external database supported by the Hibernate Framework. In this configuration, the user simply downloads the AHE executable, configures the Hibernate configuration file to set up the database connectivity and runs the programme.

With server or network constraints, AHE can be hosted inside a Servlet compliant server such as Apache Tomcat and be configured to use any databases supported by the Hibernate framework. A user should then download the AHE servlet version, deploy it on the Servlet server and configure the database configuration file to ensure AHE runs correctly. Once AHE is running, the system administrator configures user management, hosted applications as well as resources and credentials.

Whichever way the server is deployed, end users can access it either using a web browser, via the web client interface, or using the GUI or command line client tools. The client tools simply require Java to be available on the client machine; after setting an environment variable and running a configuration script these can be easily run.

7. AHE 3.0: comparison with AHE 2.0

Our efforts to refactor AHE 3 to expose a restful interface, as well as redesign the AHE server in version 3.0 have not only been done to enhance user experience, but also to improve performance. In order to evaluate the benefits of this work, we ran performance tests comparing the performance of AHE 2.0 against AHE 3.0.

Our experimental set up consisted of a server running both AHE 2.0 and AHE 3.0, with both systems configured to launch applications via the QCG-Computing middleware onto a 96 node cluster within the Centre for Computational Science at UCL. The tests we performed used a workstation to submit batches of applications to AHE 2.0 and AHE 3.0 in turn, measuring the time taken to submit these batches. The application launched was a simple code designed to sort a list of words into alphabetical order, but since we are only interested in the time performance of the AHE server itself, we only measured time taken to submit the application rather than measuring the time the application takes to execute (which would be affected by the cluster load), and the cluster was dedicated to the experiment while the tests were performed. The tests themselves were implemented as JUnit tests calling the AHE client API, while JUnit, executed via the Eclipse development platform, was used to measure the time taken to perform the tests. Each test was repeated three times, and the mean time taken for each test calculated. The results are plotted in Fig. 4, with error bars showing the standard deviation of each result.

As Fig. 4 shows, AHE 3.0 performs far faster than AHE 2.0, and the time taken to submit jobs using AHE 2.0 is much more variable,

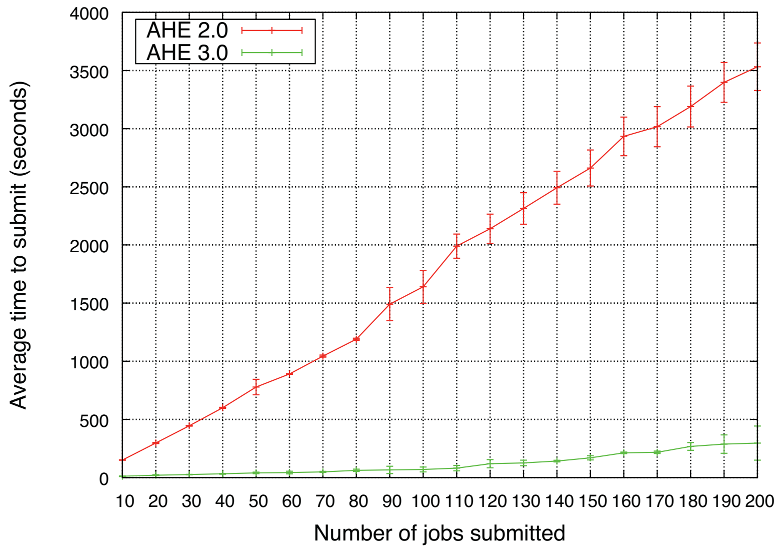


Fig. 4. Comparison of the mean time and standard deviation required to submit using AHE 2.0 and AHE 3.0 for batches of 10–200 jobs.

meaning that it is less reliable from a user perspective. This is due to the fact that AHE 3.0 is developed in Java whereas AHE 2.0 was developed with Perl/WSRF::Lite; AHE 3.0 exploits a simple RESTful interface, whereas AHE 2.0 uses the far more complicated WSRF extension to Web services, which increases the complexity of both client and server. Application submission is also faster in AHE 3.0 because the system implements a buffered queuing system between the AHE server and the connector modules, which has the effect of allowing the submission interface to process more simultaneous requests, compared to AHE 2.0. In order for us to better investigate the nature of the performance differences between AHE 2.0 and AHE 3.0, in future work we plan to instrument the server code of both AHE versions with timing routines, and then perform a set of experiments to determine the performance of different parts of the applications launching process, from data staging to job submission.

7.1. Evaluation of the Application Interaction Model

To validate our hypothesis that the Application Interaction Model, by simplifying the process of launching applications on high performance computing resources, is more usable than the traditional grid interaction model, we conducted a rigorous usability study. We have reported this study fully in Zasada et al. [17], and for a comprehensive account of the study methodology we refer readers to that publication.

We compared the AHE command line client with the Globus command line client and the AHE graphical client with the Unicore graphical client. By all of our measurements, the AHE clients were judged to be significantly more usable than either Globus or Unicore [17].

In addition to the usability tests comparing AHE with common middleware tools, we also compared the standard AHE release with the new version of AHE, integrated with the security solution Audited Credential Delegation [37] (discussed in Section 5). Our usability tests clearly established that AHE with ACD is more usable than AHE alone.

8. AHE 3.0 in action: e-infrastructure based multiscale simulation

Like its predecessors, AHE 3.0 is being actively used by several large research projects. AHE provides the principal HPC access tool in the VPH-Share project [42], a currently funded endeavour within the Virtual Physiological Human (VPH) initiative [43], concerned with patient-specific biomedical modelling and simulation. The aim of this project is to develop a set of intelligent services and supporting network infrastructure that will facilitate the exposure and sharing of data and knowledge. In particular, it is developing a multi-scale framework for the composition of new biomedical workflows to promote collaboration within the VPH community.

As part of this infrastructure, VPH-Share is developing a cloud platform that will allow users to easily access computational as well as data resources. AHE and ACD together constitute the HPC gateway service for VPH-Share, allowing simulations that require more computational power than the VPH-Share cloud infrastructure is able to provide to be seamlessly run on HPC resources. AHE and ACD are deployed based on the Software as a Service (SaaS) model. AHE is responsible for handling the execution life cycle of virtualized applications on computational resources, while ACD bridges the gap between different security infrastructures used by the execution platform and those remote resources. This allows simulation workflows to be deployed which combine resources from a cloud provider such as Amazon in order to execute single core and small scale parallel simulations, but that can switch to high performance computing, accessed via AHE, to run parts of the workflow that require more computational power. The ability of the Taverna workflow system used by VPH-Share to call AHE's RESTful interface allows applications hosted in AHE to be included as components in Taverna workflows.

The system is being used in production runs by VPH-Share scientists to run the Chaste code [44] to model personalized treatments of cardiac arrythmias in patients. AHE allows the researchers to launch Chaste simulations on the HECToR HPC machine in the UK (part of PRACE), marshal input and output data and manage parameter sweeps. It also allows data to be staged in and out of the EUDAT [45] data storage infrastructure as necessary.

Additionally, the AHE client API has been integrated with the GridSpace2 workflow engine [46] within the MAPPER project, and as part of the Virtual Imaging Platform (VIP) [47]. The MAPPER project aims to develop a persistent production infrastructure for distributed multiscale computing [48], making use of resources from multiple European e-infrastructures. AHE provides a key interoperability layer allowing higher level MAPPER services and interface tools to seamlessly access and connect these distinct resources.

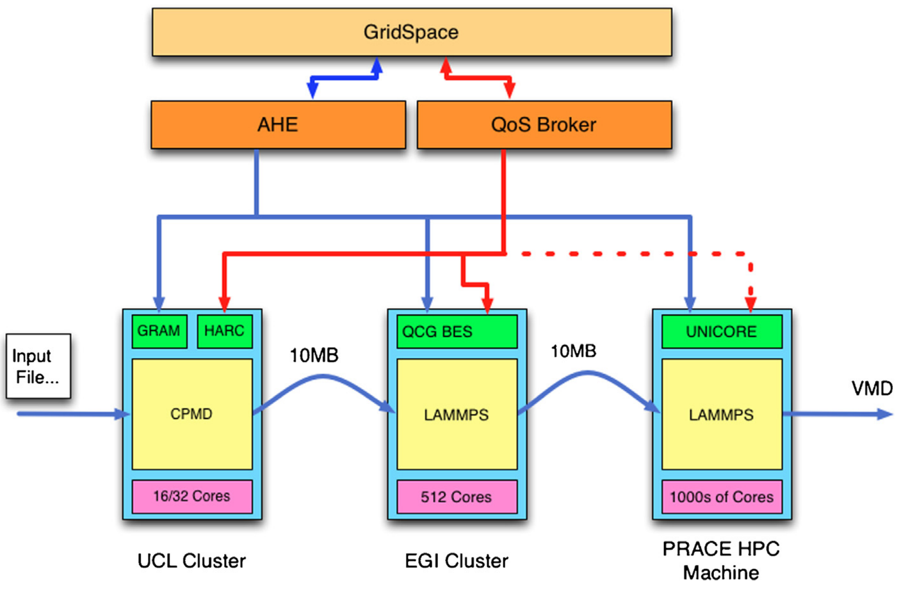


Fig. 5. The architecture of a MAPPER loosely coupled application. AHE provides an interoperability layer to launch codes across a range of platforms.

MAPPER applications couple codes operating at different temporal and spatial resolutions together. An illustration of such an application is given in Fig. 5, which shows an application from the materials science domain [49]. This application invokes parameters generated at the quantum level (using Car–Parrinello molecular dynamics (CPMD)) to build an atomistic simulation of a material using the Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS)), and then uses parameters generated at this level to perform a coarse grained molecular simulation (again using LAMMPS).

Each of the MAPPER applications requires access to suitable resources to run, often concurrently or in a particular sequence. AHE's ability to couple with advanced reservation tools such as QCG Computing means that MAPPER applications can be scheduled to run in advance (co-)reservations on HPC resources. This is of course dependent on the machines providing advanced reservation capabilities; it is essential for resource providers to implement such policies in order for these kinds of applications to be run.

The VIP project targets multi-modality, multi-organ and dynamic medical image simulation, integrating proven simulation software to simulate four main imaging modalities. This project builds on the Virtual Imaging Platform, a French ANR (National Agency for Research) project aiming at building a multi-modality simulation platform for the main medical imaging modalities, namely Magnetic Resonance Imaging (MRI), Ultrasound imaging (US), Positron Emission Tomography (PET) and Computed Tomography (CT).

Typically the simulations considered by VIP have been executed in an "embarrassingly parallel" fashion on cluster machines and are managed via the VIP platform. The platform can perform complex simulations, such as whole-body CT scans, in a reasonable time thanks to its connection to EGI, which provides access to a distributed set of clusters. However, some imaging simulation codes can be parallelized and thus benefit substantially from access to high performance computing resources. By integrating calls to AHE, via the AHE client API, with the VIP portal and workflow engine, this requirement has been satisfied. AHE provides a bridge between EGI and the HPC facilities provided by PRACE, so that simulations which require greater power than EGI can provide can instead be run on the PRACE platform.

Common to all of these projects is the need to use AHE's capabilities as an interoperability layer to bridge across disparate e-infrastructures, meaning that AHE provides a single interface to a variety of resources, from PRACE, through EGI to local campus based clusters. AHE's ability to connect to a wide range of different backend middleware tools makes it an ideal candidate to federate resource access from the user's perspective. Integration with ACD means that appropriate security credentials are presented to the target infrastructure by AHE when a user needs to execute a simulation, making the use of multiple e-infrastructure platforms transparent to the user.

9. Conclusions and future work

Since its initial release, AHE has been taken up by various user communities, where its usability features have proved extremely important. AHE has been employed to host computational codes from different scientific domains, including widely used codes such as NAMD, CHARMM, LAMMPS, VASP, LB3D and DL_POLY. A key strength of AHE is its flexibility. Since all of its complexity resides on the server side, and all of AHE's functionality is exposed as RESTful Web services, AHE can be used as a building block for systems of much greater complexity.

As our performance tests have shown, the redesign of AHE 3.0 has greatly improved performance over older AHE versions. Our usability results have also confirmed the benefit of the Application Interaction Model in that user interaction is reduced to the most essential components: namely a user interacting with his/her application. Users do not need to worry about the details of particular batch queuing systems, or how to stage data back from particular HPC resources; the specifics of how to launch an application are encapsulated within the Application Interaction Model. The approach virtualizes the HPC resources from a user's point of view. Indeed AHE virtualizes the entirety of a grid's HPC resources, and federates resources stemming from multiple different e-infrastructures.

AHE 3.0 provides a number of capabilities including a workflow engine that allows complex simulations to be created, including coupled simulations where data is automatically transferred from one application to another. ACD provides a security suite that includes support for Shibboleth authentication, as well as user auditing. ACD supports virtual organization management and is able to provide access to grid proxy credentials through RESTful web services.

This combination of usability and performance embedded into a feature-rich platform have led AHE and ACD to become an important cornerstone of many research projects, from materials science through computational physics and chemistry to life and medical sciences. These projects have in common a need for a computational platform to provide access to high performance computational resources and links to cloud computing infrastructures. ACD and AHE allow e-infrastructure to be accessed in a similar manner as an IaaS cloud resource. This is achieved by virtualizing applications using an SaaS model, exposing their functionality as simple RESTful web services, and by abstracting the security mechanism of the e-infrastructure middleware through ACD. Although they may be and often are deployed in combination as part of a project specific e-infrastructure, AHE and ACD are also standalone tools, and can be deployed within any similar e-infrastructure that requires transparent access to high end computing resources.

AHE 3.0 and ACD have been released under the LGPL licence and can be downloaded from: https://sourceforge.net/projects/ahe3/.

Acknowledgements

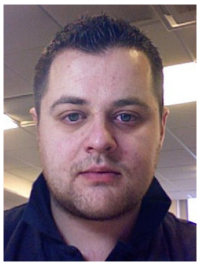
The developments of AHE and ACD reported in this paper have been funded by the EU FP7 VPH-Share (no. 269978), VPH-NoE (no. 223920), MAPPER (no. 261507) and ContraCancrum (no. 223979) projects. Previously they were funded by the EPSRC projects RealityGrid (GR/R67699), Rapid Prototyping of Usable Grid Middleware (GR/T27488/01), User-Friendly Authentication and Authorization for Grid Environments (EP/D051754/1), and also by OMII under the Managed Programme Robust Application Hosting in WSRF::Lite (RAHWL) project. SJZ's PhD studentship is funded by EPSRC.

We would like to thank Maciej Pawlik and Grzegorz Dyk from Cyfronet, Krakow, Poland, and William Romero from CNRS Lyon, France, for their contributions to the code, and also Mariusz Mamonski and Tomasz Piontek from Poznan Super Computing and Networking Center, Poland for useful discussions relating to advance reservation.

References

[1] D. Turek, High performance computing and the implications of multi-core architectures, CTWatch Quarterly 3 (2007) 31–33.
[2] J. Dongarra, D. Gannon, G. Fox, K. Kennedy, The impact of multicore on computational science software, CTWatch Quarterly 3 (2007) 1–10.
[3] P.V. Coveney (Ed.), Scientific Grid Computing, Vol. 363, Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 2005.
[4] I. Foster, C. Kesselman, S. Tuecke, The anatomy of the grid: enabling scalable virtual organizations, International Journal of Supercomputer Applications 15 (2001) 3–23.
[5] J. Chin, P.V. Coveney, Towards Tractable Toolkits for the Grid: a Plea for Lightweight, Useable Middleware, Tech. rep., 2004 http://nesc.ac.uk/technical_papers/UKeS-2004-01.pdf
[6] M. Halling-Brown, D. Moss, C. Sansom, A. Shepherd, A computational grid framework for immunological applications, Philosophical Transactions of the Royal Society A 367 (2009) 2705–2716.
[7] P.V. Coveney, R.S. Saksena, S.J. Zasada, M. McKeown, S. Pickles, The application hosting environment: lightweight middleware for grid-based computational science, Computer Physics Communications 176 (6) (2007) 406–418.
[8] S.J. Zasada, P.V. Coveney, Virtualizing access to scientific applications with the application hosting environment, Computer Physics Communications 180 (12) (2009) 2513–2525.
[9] S.K. Sadiq, D. Wright, S.J. Watson, S.J. Zasada, I. Stoica, P.V. Coveney, Automated molecular simulation based binding affinity calculator for ligand-bound HIV-1 proteases, Journal of Chemical Information and Modeling 48 (9) (2008) 1909–1919.
[10] J.L. Suter, P.V. Coveney, H.C. Greenwell, M.-A. Thyveetil, Large-scale molecular dynamics study of montmorillonite clay: emergence of undulatory fluctuations and determination of material properties, The Journal of Physical Chemistry C 111 (23) (2007) 8248–8259.
[11] R.T. Fielding, Architectural styles and the design of network-based software architectures, 2000 (Ph.D. thesis).
[12] S. Graham, A. Karmarkar, J. Mischkinsky, I. Robinson, I. Sedukin, Web Services Resource Framework, Tech. rep., 2006 http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf
[13] WINE, http://www.winehq.org/
[14] P.V. Coveney, Scientific grid computing, Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 363 (1833) (2005) 1707–1713.
[15] S. Manos, S.J. Zasada, P.V. Coveney, Life or death decision-making: the medical case for large-scale on-demand grid computing, CTWatch Quarterly Journal 4 (2) (2008) 35–45.
[16] B. Beckles, V. Welch, J. Basney, Mechanisms for increasing the usability of grid security, International Journal of Human–Computer Studies 63 (1/2) (2005) 74–101.
[17] S.J. Zasada, A.N. Haidar, P.V. Coveney, On the usability of grid middleware and security mechanisms, Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 369 (1949) (2011) 3413–3428.
[18] B. Boghosian, P.V. Coveney, S. Dong, L. Finn, S. Jha, G. Karniadakis, N. Karonis, NEKTAR, SPICE and Vortonics: using federated grids for large scale scientific applications, Cluster Computing 10 (3) (2007) 351–364.
[19] UK Collaborative Computational Projects, http://www.ccp.ac.uk/
[20] S.J. Zasada, M. Mamonski, D. Groen, J. Borgdorff, I. Saverchenko, T. Piontek, K. Kurowski, P.V. Coveney, Distributed infrastructure for multiscale computing, in: Proceedings of the 2012 IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications, IEEE Computer Society, 2012, pp. 65–74.
[21] JBPM – JBoss Community, http://www.jboss.org/jbpm
[22] Hibernate – JBoss Community, http://www.hibernate.org/
[23] Restlet – RESTful web services framework for Java, http://www.restlet.org/
[24] The Apache Tomcat Servlet Container, http://tomcat.apache.org
[25] The Globus Project, http://www.globus.org
[26] The UNICORE Project, http://www.unicore.org
[27] K. Kurowski, W. de Back, W. Dubitzky, L. Gulyás, G. Kampis, M. Mamonski, G. Szemes, M. Swain, Complex system simulations with QosCosGrid, Computational Science-ICCS 2009 (2009) 387–396.
[28] A.N. Haidar, S.J. Zasada, P.V. Coveney, A.E. Abdallah, B. Beckles, M.A.S. Jones, Audited credential delegation: a usable security solution for the virtual physiological human toolkit, Interface Focus 1 (3) (2011) 462–473.
[29] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, S. Tuecke, GridFTP: Protocol extensions to FTP for the grid, Global Grid Forum GFD-RP 20, http://www.ggf.org/documents/GWD-R/GFD-R.020.pdf
[30] P. Beckman, S. Nadella, N. Trebon, I. Beschastnikh, SPRUCE: a system for supporting urgent high-performance computing, Grid-Based Problem Solving Environments (2007) 295–311.
[31] S. Pickles, R. Haines, R. Pinning, A. Porter, A practical toolkit for computational steering, Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 363 (1833) (2005) 1843–1853.
[32] C. Cavaness, Quartz Job Scheduling Framework: Building Open Source Enterprise Applications, Prentice Hall, Upper Saddle River, NJ, USA, 2006.
[33] R.O. Sinnott, J. Jiang, J. Watt, O. Ajayi, Shibboleth-based access to and usage of grid resources, in: IEEE International Conference on Grid Computing, Barcelona, Spain, 2006, pp. 28–29.
[34] J. Novotny, S. Tuecke, V. Welch, An online credential repository for the grid: MyProxy, in: 10th IEEE International Symposium on High Performance Distributed Computing, 2001. IEEE Proceedings, 2002, pp. 104–111.
[35] X. Wang, M. Jones, J. Jensen, A. Richards, D. Wallom, T. Ma, R. Frank, D. Spence, S. Young, C. Devereux, et al., Shibboleth access for resources on the National Grid Service (SARoNGS), in: Fifth International Conference on Information Assurance and Security, 2009, IAS'09. Vol. 2, IEEE, 2009, pp. 338–341.
[36] R. Alfieri, R. Cecchini, V. Ciaschini, L. dell Agnello, A. Frohner, A. Gianoli, K. Lorentey, F. Spataro, VOMS, an authorization system for virtual organizations, in: Grid Computing, Springer, 2004, pp. 33–40.
[37] A.N. Haidar, P.V. Coveney, A.E. Abdallah, P.Y. Ryan, B. Beckles, J.M. Brooke, M. Jones, Formal modelling of a usable identity management solution for virtual organisations, in: Proceedings of Formal Aspects of Virtual Organisations, 2009, pp. 41–50.
[38] The Open Web Application Security Project, http://ww.owasp.org
[39] OWASP top 10: The ten most critical web application security vulnerabilities (2007), http://www.owasp.org/index.php/Top_10_2010-Main
[40] A. Abdallah, E. Khayat, Formal *z* specifications of several flat role-based access control models, in: Software Engineering Workshop, 2006, SEW'06. 30th Annual IEEE/NASA, IEEE, 2006, pp. 282–292.
[41] UK National e-Infrastructure Service (NES), http://www.ngs.ac.uk
[42] The VPH-Share Project, http://www.vph-share.eu
[43] P. Hunter, P.V. Coveney, B. de Bono, V. Diaz, J. Fenner, A. Frangi, P. Harris, R. Hose, P. Kohl, P. Lawford, et al., A vision and strategy for the virtual physiological human in 2010 and beyond, Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 368 (1920) (2010) 2595–2614.
[44] J. Pitt-Francis, P. Pathmanathan, M. Bernabeu, R. Bordas, J. Cooper, A. Fletcher, G. Mirams, P. Murray, J. Osborne, A. Walter, et al., Chaste: a test-driven approach to software development for biological modelling, Computer Physics Communications 180 (12) (2009) 2452–2471.
[45] The EUDAT Project, http://www.eudat.eu
[46] M. Malawski, T. Bartynski, M. Bubak, Invocation of operations from script-based grid applications, Future Generation Computer Systems 26 (1) (2010) 138–146.

[47] The VIP Project, http://www.creatis.insa-lyon.fr/vip/
[48] The MAPPER Project, http://www.mapper.eu
[49] J. Suter, D. Groen, L. Kabalan, P.V. Coveney, Distributed multiscale simulations of clay-polymer nanocomposites, MRS Online Proceedings Library 1470 (1) (2012).



Stefan J. Zasada is a software engineer in the Centre for Computational Science at UCL, developing lightweight grid middleware and enabling tools for e-Science. He has a first degree in Computer Science from the University of Nottingham and a Masters degree in Advanced Software Engineering from the University of Manchester, where he was responsible for implementing the WS-Security specification in Perl for use by the WSRF::Lite toolkit. Currently he is lead developer on the AHE project. He is involved in developing medical data sharing solutions in the the EU FP7 p-medicine and UK MRC Farr projects, and also lightweight grid middleware and enabling tools for e-Science. He is currently completing his PhD in Computer Science, investigating the design and development of lightweight application virtualization toolkits and market based resource allocation solutions. His research interests cover many aspects of high performance and grid computing, and their application in the medical and life sciences domain.

David Chan-Wei Chang is currently a research associate in the Graduate School of Biomedical Engineering, University of New South Wales, Sydney, Australia. He is working in the area of tele-health and tele-care, clinical decision support and machine learning. He was previously at the Centre for Computational Science at University College London, working in the VPH-Share and p-medicine projects. He completed his BSc, MSc and PhD degrees at the University of New South Wales, where he worked in the Biomedical System Laboratory on a number different projects, including the development of clinical decision support systems, non-intrusive wireless monitoring systems for the elderly and signal analysis. His research is focused on cardio electrophysiology, medical IT infrastructure and cloud and grid computing.

Ali Nasrat Haidar is an Application Security Consultant at HSBC Head office in London. He has a PhD in web services security and a Masters degree in information security from Royal Holloway, University of London. He is also a Visiting Research Fellow in the Centre for Computational Science at UCL, a part-time Research Fellow at Birmingham City University in the Cyber Security group, and a member of the UCL Cyber Security Centre of Excellence awarded by GCHQ in partnership with the Research Councils' Global Uncertainties Programme (RCUK) and the Department for Business Innovation and Skills (BIS). Prior to his current appointment, Ali was a Research Fellow at UCL and a Research Associate at the Centre for Software Reliability at Newcastle University. He was involved in developing secure e-Science applications, data sharing platforms, capturing user and security requirements for computational grid environments, providing formal models and analysis of these requirements to assist the design of security prototype. Ali was involved in a number of EU e-health research projects, such as the VPH-NOE (Virtual Physiological Human Network of Excellence), the ContraCancrum (Clinically Oriented Translational Cancer Multilevel Modelling) and p-medicine (Personalized Medicine). His research interests include secure software development, identity and access control management, grid and cloud security, information assurance and compliance.

Peter V. Coveney holds a Chair in Physical Chemistry, and is Director of the Centre for Computational Science, Director of the UCL Computational Life & Medical Sciences Network, an Honorary Professor of Computer Science, and a member of CoMPLEX at UCL. He is also a founding member of the UK e-Infrastructure Leadership Council, a Medical Academy Nominated Expert on Data, Algorithms, and Modelling for the UK Prime Minister's Council for Science and Technology, and Professor Adjunct within the Yale School of Medicine at Yale University. He is active in a broad area of interdisciplinary theoretical research including condensed matter physics and chemistry, materials science, and life and medical sciences including collaborations with clinicians. He is a founding editor of the Journal of Computational Science and to date has published more than 300 scientific papers, edited 20 books, and coauthored two best-selling popular science books (The Arrow of Time and Frontiers of Complexity, both with Roger Highfield).