Supporting QoS-Based Discovery in Service-Oriented Grids

Rashid J. Al-Ali, Ali ShaikhAli, Omer F. Rana and David W. Walker
Department of Computer Science
Cardiff University, UK
{Rashid.A.ShaikhAli,O.F.Rana,David.W.Walker}@cs.cardiff.ac.uk

Abstract

In this work the Service abstraction is extended to support Quality of Service (QoS) attributes – and implemented in the Grid QoS Management (G-QoSM) framework. The framework supports three main functions: (1) providing mechanisms for establishing QoS guarantees via Service Level Agreements (SLAs), (2) enabling QoS management on allocated QoS-aware services, and (3) supporting discovery of services based on QoS attributes. The framework relies on each service offering both a “functional” interface and a “management” interface. The functional interface provides attributes on how a service is to be invoked and how it returns results, whereas the management interface is used to provide QoS attributes and performance characteristics associated with a service. The focus of this paper is on implementing a QoS-based service discovery system, which utilises an extended version of the Universal Description, Discovery and Integration (UDDI) registry.

1 Introduction

Service-oriented computing has become an important paradigm for deploying software. A “service” may be defined as a behaviour that is provided by a software component, for use by another, based only on an interface. The interface acts as a contract between services, enabling the discovery, advertising, delegation and composition of services [1]. Most contracts at present are addressing the functional interface of a service – generally describing aspects such as called and return parameter types, and exception handlers. In our framework we emphasise non-functional attributes of a service – primarily its QoS. We define QoS at three different levels: at the application level, at the middleware level, and at the network level – and suggest how these three different aspects can be combined to support service management. This work is based on our G-QoSM framework, described in detail in [3].

In service-oriented Grids we assume we are dealing with scarce and expensive resources, which often highlights the need for QoS provisions such as advance/immediate resource reservation and provision of certain resource capability. We envision the application area to be scientific computing services, such as computation steering and visualisation services, whereby interactions may take place between single or multiple services within a given session. Examples of application domains include structure modelling, telemedicine and collaborative working.

In this paper we address QoS domains in service interface definition based on the OGSA architecture, whereby service providers state QoS attributes as part of their service interface. The services are then published into a registry system – we extend UDDI and implement UDDIe, which is capable of supporting search based on QoS properties by providing (3) advanced features: service properties, service leasing, and range-based search. This paper addresses (3) main contributions as follows: i) extending the UDDI to support service discovery based on QoS properties in Grid environments, ii) implementing a UDDIe with advance search features and iii) demonstrating the feasibility of such extension in the context of G-QoSM framework.

2 Applications

QoS guarantees are necessary for some applications to function adequately – such as the commonly used multimedia applications quoted in literature [9]. In such applications, a session may comprise either interactions between two services, or concurrent data transfers between multiple services. For instance, a user may download multimedia content from a single server (offering video, audio and text), or each of the content types may be downloaded from different servers and then synchronised at the server end. In both these instances, QoS guarantees from one or all of the service providers are necessary.

Similarly, in the context of scientific computing, the primary application area being addressed here, interactions may be between single or multiple services within a given session. An example is computational steering, whereby a
session may involve repeated interactions between two services. Each interaction in this context must support QoS guarantees for the application behaviour to be meaningful to a user. In such applications, a simulation service must be coupled with a visualisation service, as both services need to exchange control parameters and data. Hence, the visualisation service must be able to transfer control parameters to the simulation service within a limited time frame, and update the visual representation from simulation results as a result of the control parameters. In this instance, service discovery is undertaken prior to the two services interacting with each other, by specifying the visualisation/simulation service along with their QoS requirements to the service discovery system. However, once the interaction has ceased (i.e. the session is completed), summary statistics on service execution may be recorded to determine whether the same simulation or visualisation service may be used.

3 Related Work

GARA [6] provides a framework to support QoS functions such as advance/immediate resource reservation and co-allocation support in Computational Grids. Specification of QoS associated with a particular service is however not addressed in the GARA framework. Nevertheless, we could utilise GARA as the underlying resource manager to provide resource QoS. The myGrid project [8] is a middleware project that aims to provide a toolbox to help biologists and bioinformaticians to perform workflow-based in silico experiments, and subsequently to automate the management of such workflow. The concepts of service leasing and QoS registration for service instances are explored in the “Service Directory” of the myGrid project and UDDI-M [5]; however, the main difference from our work is that we extend the UDDI primarily to support service discovery for services based on QoS – thereby providing specialised support for Grid Services. The SDS[4] system provides a service discovery architecture consisting of clients, services and SDS servers. This architecture supports a number of interesting features such as security, scalability and the notion of a capability manager. In SDS, clients search for services based on their capability and not based on clients’ QoS requirements. Furthermore, the SDS supports search based on the XML query-model. The hierarchical approach adopted in SDS has resulted in a number of different service discovery schemes, and adopted widely in the Peer-2-Peer systems.

Discovery of services with QoS has also been explored in [12], where the authors demonstrate how the inclusion of service feedbacks could improve QoS of a session. The Wide-Area Discovery Framework was enhanced to provide a better query responsiveness and QoS awareness. Feedback, in this context means that during a service session a software component monitors the QoS, and generates the numerical average QoS level observed. The QoS is highly service-specific and does not have a generic specification as is the case with our model. Furthermore, their work targets queries which have to traverse a number of discovery services in hierarchal fashion. Madja et al [7] propose a data model for QoS management in the Web. The data model is a set of QoS characteristics for multimedia (MM) audio/video documents. This data may be stored in a database, as text files, or as an extension of HTML tags. The client specifies the desired quality of the MM document and the QoS manager accesses the MM document’s meta-data to negotiate the requirements identified by the client. This work is conceptually similar to our approach, with the exception that their focus is limited to the MM documents, whereas our focus is on Grid services.

4 G-QoSM Background

The Grid QoS management framework (G-QoSM) [3], is a model that aims to operate in service-oriented Grids and to provide three main functions: 1) support for resource and service discovery based on QoS properties, 2) support for providing QoS guarantee at application, middleware and network level, and establishing service level agreements (SLAs) to enforce these QoS parameters, and 3) providing QoS management on allocated resources. The G-QoSM delivers three types of QoS levels: guaranteed QoS, controlled-load QoS and best-effort QoS. In the case of guaranteed QoS, constraints related to QoS parameters of the client need to exactly match the server provision. Controlled-load QoS is similar to guaranteed QoS with the exception that the QoS levels are somewhat more relaxed and the notion of range-based QoS attribute is used along with range-based SLAs. On the other hand in best-effort QoS the resource manager has full control to choose the QoS level without any constraints.

4.1 G-QoSM System Architecture

G-QoSM consists of three main components as in Figure 1: 1) Application QoS manager (AQoS) – there being a single AQoS for each application that is to be deployed, 2) middleware Resource Manager (RM) and 3) Network Resource Manager (NRM). AQoS is the main focus within our system, and is required to interact with clients, RMs, NRMs and neighbouring AQuoSs. The AQoS also negotiates SLAs with clients and communicates parameters associated with an SLA to the corresponding resource manager. Furthermore the AQoS is responsible for ensuring SLA conformance on the allocated resources, and provides scaling actions in case of SLA violations. In addition, the middleware resource manager (RM) exists within a given administrative
domain. A domain can be defined via an IP mask or as an administrative domain in Globus, for instance, and contains a set of services over which the RM has administrative and configuration control. A RM is considered in this context as a combination of Globus Resource Allocation Manager (GRAM) and a Universal Description and Discovery Integration (UDDI) registry. Globus is used to host services and to create an execution environment with specific QoS specifications; UDDI on the other hand is used as a registry service, where services can be registered and discovered based on their capability and QoS attributes.

The Network Resource Manager (NRM) is conceptually a Bandwidth Broker (BB) (the concept of a BB can be found in [10]), and manages the QoS resources within a given domain based on the SLAs that have been agreed upon in that domain. The NRM is also responsible for managing interdomain communication, with the NRMs in neighbouring domains, with a view to coordinating SLAs across the domain boundaries. The NRM may communicate with local monitoring tools to determine the state of the network and its present configuration.

A scenario that illustrates a typical operation within the G-QoSM is illustrated in Figure 2 – showing a sequence diagram of interactions between the various components of the system. All interactions are encoded as XML messages. A client sends a message to the AQoS containing service information along with QoS requirements such as reservation time and budget constraints. The AQoS queries the UDDI registry for services with the specified QoS capabilities. The UDDI registry replies to the AQoS with a list of matching services. The AQoS contacts the corresponding resource managers, namely the NRM and Globus GRAM, for resource availability with the specified QoS levels. Once the resource managers reply, this concludes the service and resource discovery phase based on QoS properties.

After this the AQoS and the client enter a negotiation phase aimed at reaching a mutual agreement on resource QoS levels and to establish a service level agreement (SLA).

Once the SLA is established, its parameters are deployed to the corresponding resource managers, namely Globus GRAM and NRM, for resource allocation. Eventually, Globus GRAM invokes the service for execution, and this concludes the second phase, which is resource QoS specification and SLA establishment.

Once resources have been allocated and the service invoked for execution, the last phase of the G-QoSM execution process – QoS management – is initiated. In this phase, the AQoS queries status information on allocated resources for monitoring purposes to ensure SLA conformance, and applies appropriate scaling actions, such as adaptation techniques, in case of SLA violation. These adaptation techniques are only applicable for controlled-load or best effort QoS, and results in some of the constraints being reevaluated or an alert event being sent to the client service. In case of a guaranteed QoS, an error message is returned.

5 QoS Properties for Grid Services

A Grid service is a software entity that needs some resource capability for its execution, and conforms to the Grid Services Specification [11]. In the G-QoSM framework we classify the service QoS properties into (5) QoS domains that could be used to describe the service capability, as follows:

Figure 1. The G-QoSM Architecture

Figure 2. A Sequence Diagram Shows the Interaction between the Various G-QoSM Components
1. Accounting QoS: This property is assumed to be supplied by the service provider and considered as the cost of the software that makes up the service. The accounting service assumes that some monitoring mechanism is already in place.

2. Service QoS: This class of QoS is provided by the service provider and its primary objective is to state what main QoS attributes and their values are absolutely necessary to run the service. For example, in multimedia services such as video streaming, bandwidth and packet jitter are necessary QoS characteristics to be specified in order for the service to run effectively.

3. Provisional QoS: This type of QoS is very similar to service QoS with the exception that these attributes do not affect the end result of service execution. For example, if a service is doing a bulk data transfer, increasing the bandwidth would result in a better execution but the end result is not affected if the user chooses a smaller bandwidth.

4. Service Reliability: This QoS attribute defines the degree of service reliability as a transaction – specifying actions to be taken if the service fails in the middle of its execution, e.g. whether it will roll back, or what fault tolerance action would be taken.

5. Service Security: This QoS attribute deals with service security issues, primarily with who has the right to access this specified service. Such information could be associated with services in the form of access control lists (ACLs).

These attributes are different from the functional interface offered by a service – as QoS attributes may still hold even if a service were to return an incorrect result. The correctness or accuracy of a result are therefore not part of the G-QoSM framework, and must be dealt with by the service client.

6 UDDIe: Supporting Discovery Based on Service QoS Properties

UDDIe forms the service registry system in the context of the G-QoSM framework – Figure 3. It is used primarily to publish services with their QoS information and subsequently to search for services based on QoS attributes. The publication process, as per the Open Grid Service Architecture (OGSA) specifications, requires that service providers supply two separate WSDL documents, namely, service interface and implementation documents that describe the syntax and semantics for accessing the service. Current OGSA specification does not specify a QoS structure to be included in the services’ WSDL documents. Every service in our framework has two interfaces: functional and management interfaces. The functional interface describes how the given service could be accessed, whereas the management interface describes attributes related to QoS and performance characteristics associated with the service. We incorporate the QoS properties defined in section 5 in the service management interface specifically within the service interface definition that is specified in the OGSA specifications [11]. With this extension to the WSDL document, the service provider would be able to describe their services in terms of QoS properties. In order for this extension to be recognised by a registry system such as UDDI and hence searchable, the UDDI needs to be extended to recognise these additional attributes. We design and implement 3 search capabilities, by extending UDDI: service properties, service leasing and range-based search. These properties together can be used to search for stored documents based on their properties – rather than their keys or tmodels – as undertaken in a standard UDDI implementation. Documents stored within the registry also have a lease associated with them, and an event manager to support these leases has been implemented in UDDIe. In this way, we are able to search for documents which match a given range of QoS attributes – for supporting controlled-load QoS.

6.1 Service Properties

The UDDI registry system has also been extended to support a search based on service properties. The idea behind this is an attempt to provide Grid service users with a mechanism to discover services based on some QoS properties. Since the OGSA specifies WSDL conventions to be used...
to describe user interactions with Grid services, we extend the Grid service interface WSDL document to include the QoS properties discussed in section 5. This addition of service properties is similar in concept to the serviceData described in the Grid Service Specification [11]. However, the main distinction here is that our focus is primarily on QoS information related to the Grid service, whereas serviceData is meant to refer to descriptive information about Grid service instances – and is more generic in scope. In order for this newly defined service property to be searchable by the registry system, the UDDI registry replies to queries based on QoS attributes, such as service properties. More details can be found in the implementation section 7.

6.2 Service Leasing

Service leasing is another feature that we add to our registry system. Often, service providers are interested in leasing their services with certain QoS constraints, such as cost, for a predefined time frame and then advertising the same service with different QoS provisions at some other time. This is similar to the way telecommunication companies introduce different charges according to different time frames, for instance peak and off peak times. Another motivation for this feature is the introduction of Grid service lifetime management in the OGSA specification [11], which refers to the validity of the service from its creation to destruction. The actual mechanism by which the Grid service is created or destroyed is a property of the hosting environment. Therefore, we incorporate this feature with our registry system to allow the service providers to state the validity of their service, along with its QoS properties, and likely times after which the service is no longer valid, so it can be removed from the registry database. This soft-state mechanism also allows entries in the database to more appropriately represent the current state of the system. We also include other query functions to allow the registry system to be queried based on this property.

6.3 Range-Based Search

We introduce a novel feature to our registry system, which is the capability of searching for QoS attributes based on numerical ranges, utilising operators such as “less-than”, “greater-than” and “between”, in addition to logical operators such as AND/OR. This range-based search idea comes from the range-base service level agreements (SLAs) introduced in the G-QoSM framework [3]. Our framework establishes SLAs with clients which include QoS attributes that are specified in range-oriented relations. For example a range-oriented SLA, for a multimedia service, might have the following relations: \( R_1 = \text{Bandwidth} > 512\text{Kbps}, \) \( R_2 = \text{Memory} \geq 20MB. \) Using these extensions it is possible to locate services in the registry system that have QoS attributes specified in \( R_1 \) and \( R_2. \) On the other hand, the logical operations such as AND/OR, adds extra flexibility to the registry/discovery system by allowing queries with a number of QoS attributes to be combined, such as the following example:

\[
\text{Query} = ((\text{CPUnodes} \leq 2) \land (\text{Bandwith} \leq 2Mbps))
\]

6.4 Mapping of Services’ QoS into the Service-Interface Definition

In order for the extensions mentioned above to take effect, there a service provider (SP) must explicitly publish their services with QoS attributes. In the OGSA specification document, there is no explicit provision for associating QoS attributes with the service definition; however, one could use the ServiceData tag within the service interface definition to include QoS attributes. We utilise this tag to represent services’ QoS information. The ServiceData tag is mapped to our Service Property extension in the UDDIe, which eventually makes this QoS information searchable. We have developed a service publishing tool wizard that automates the publishing process by taking in two WSDL-based documents: i) service interface definition, and ii) service implementation and publishes the service in the UDDIe. The tool helps service providers to include services’ QoS attributes in the service interface definition document and publishes the service in the UDDIe registry – without having to write any XML code. The tool automatically generates the relevant WSDL documents, that can then be published with a lease period in the UDDIe registry.

```
<ServiceRequest>
  <ServiceName> MathService </ServiceName>
  <ServiceQoS>
    <ServiceName> MathService </ServiceName>
    <ServiceCost> 10 LOW </ServiceCost>
    <CpuQoS> 5 nodes HIGH </CpuQoS>
    <MemoryQoS> 64MB </MemoryQoS>
  </ServiceQoS>
</ServiceRequest>
```

Figure 4. A client/application requesting a service with QoS specifications along with the QoS importance level encoded in an XML message

0-7695-1926-1/03/$17.00 (C) 2003 IEEE
Figure 5. The Clients/applications, the AQoS and the UDDIe interaction model. The client/appl. sends a request as an XML message to the AQoS servlet, the AQoS parses the request, processes it and sends it to the UDDIe servlet for service discovery.

7 Implementation

We have implemented a prototype of a service discovery system utilising the UDDIe registry - and based on a public domain implementation of UDDI from uddi.org. The implementation uses the following technologies: Java (JDK1.3.1), SOAP, J4UDDI APIs, WSDL, and XML. The system is implemented in the context of our G-QoSM framework, whereby clients/applications send their requests for services with QoS properties to our QoS broker – the AQoS as in Figure 4. The AQoS processes the request and submits the service request portion to the UDDIe. The UDDIe replies with a list of services that support this particular query. Finally, the AQoS applies a selection algorithm to select the most appropriate service with respect to the client/application request and sends the result to the client/application. The selection algorithm that the AQoS uses is based on the Weighted Average (WA) concept. In this algorithm, we introduce the notion of QoS importance level, whereby the client/application is requested to associate a level of importance, such as High, Medium or Low, with every QoS attribute. Based on this QoS importance level and QoS attribute’s value, the algorithm computes the WA for every returned service and select the service with the highest WA.

7.1 The UDDI Extension

We introduce the Application Programming Interface (API) for interacting with the registry system, as discussed below. These sets of APIs are the interfaces methods, e.g. via which the UDDIe services could be invoked, as described in Figure 5.

• `saveService`: This set of APIs is mainly used for publishing service details. This has been extended from the original UDDI system to introduce dynamic metadata for services. We utilise them to represent QoS information, and they also could be used to represent various service information.

• `findService`: This set of APIs is mainly used for inquiry purposes. In particular we extend this set of API to include queries based on various information associated with services, such as Service Property and Service leasing.

• `getServiceDetails`: This set of APIs is mainly used for requesting more detailed information about services, such as BusinessKey, information, etc. We extend this set of APIs to include Service Property information as well.

• `renewLease`: This newly defined set of APIs is used by both the operator/UDDI administrator to control leasing information, and by the service provider (SP) to renew and set leasing information. The leasing concept works as follows; every service is associated with a lease information, either for limited duration or for an infinite time period. The maximum number of infinite leased services is controlled by the operator to maintain the database efficiently. In the case of limited duration, the SP provides a start-from date and an expiration date for the lease period. The operator has control on setting up the default leasing period. Moreover, if a lease expires the SP could always renew the expired lease, provided that the request falls within the allowed number of times to renew a particular lease, which is controlled by the operator/UDDI administrator. When the lease period expires, the service becomes invalid and clients cannot make use of the expired services. It is important to continually renew a lease or request an infinite lease – and an event manager is used to alert all connected users if the lease of the service is about to expire.

• `startLeaseManager`: This newly defined set of APIs is used to monitor the lease constraints by starting

1The UDDI registry service is available as an open-source software from http://www.cs.cf.ac.uk/user/A.Shaikhali/uddie/
processes to monitor expired leases and delete them from the database. The operator has control over how often to run these processes.

In addition to these sets of APIs, we introduce support for a Qualifier-based search, that is, to find services based on some property along with a qualifier value, such as EQUAL, LESS THAN and GREATER THAN. Support for logical operations is also introduced to the extended UDDI, which enables querying for properties with logical AND OR. We believe these extensions to the UDDI registry and query mechanisms would add a great search flexibility, making UDDI a more powerful search engine. The ability for UDDIe to co-exist with standard UDDI version is also an important aspect of this work – as we do not break compatibility with existing UDDI deployments.

8 Experiment

The main objective of undertaking this experiment were to observe the performance of the UDDIe registry, to determine whether it is acceptable for real-time like applications – such as the applications which necessitate QoS provision. Also, the experiment is an attempt to identify any bottlenecks in the query processing path – so these could be factored into the SLA. The experiment infrastructure is as follows: the AQoS manager, which processes the clients’ requests, the UDDIe and its database management system are run on the same server. The queries are issued from a client workstation. The client workstation and server are located in the computer science department at Cardiff, and connected via a 100Mbps Ethernet network. We measure the query round trip time (RTT) as the time it takes for the query to be submitted from the client workstation and processed by the AQoS and UDDIe, and the results sent back to the client, (Figure 6 depicts the query logical path). Therefore, the RTT could be computed according to the following equation:

$$QueryRTT = \sum_{i=1}^{4} T_i$$

The experiment consists of a mixture of queries for services with QoS properties: Query1 requests services with QoS properties, making use of the Service Property extension, and the result of this query is (No services found), for response time observation purposes. Query2 requests services with QoS properties along with service validity constraints, making use of the newly defined Service Leasing extension. Query3 requests services with more QoS properties and leasing information. It introduces logical operations, making use of the Range-Based Search mechanism that is incorporated into the UDDIe. We populated the UDDIe with a number of services and we submitted the (3) types of queries to the prototype system in different cases: each case with different service name and QoS attributes aiming to observe the system response time. Table 1 shows the average RTT times for the submitted queries in each case.

| Table 1. Round Trip Time (RTT), in milliseconds, for the (3) queries in the (3) different cases |
|-------------------------------|-------------------------------|-------------------------------|
| Query | RTT – case1 | RTT – case2 | RTT – case3 |
| Query1 | 2749 | 4421 | 5031 |
| Query2 | 9250 | 11469 | 13422 |
| Query3 | 9703 | 9407 | 10703 |

8.1 Discussion

The main purpose of this experiment is to show performance characteristics as a result of integrating the developed UDDIe into the G-QoS framework. Therefore, we observe, from the experiment, that the minimum time taken by the AQoS and the UDDIe to process a request, coming from a client/application, is about 5 seconds. We also realize that if the returned list of (found services) is greater than 30 services and the AQoS has to choose from amongst these returned services based on the client’s constraints, then the response time is quite high, and therefore we adjust the maximum number of returned services to 5, which then results in an acceptable response time. It is observed from the experiment that the average response time of a successful request takes about 9 seconds; we also think that this response time is acceptable for a prototype system being tested in a distributed computing environment where 3 systems interact together in an internet environment using XML and SOAP messaging protocol. The overall response time could also be enhanced by tuning the selection algorithm further – we leave this for future work as currently we are focusing on delivering a prototype QoS broker (AQoS).
9 Conclusion

In this paper we propose a service discovery system (UDDIe) based on the UDDI specification [2], that is capable of discovering services based on QoS attributes. We extended the UDDI registry to support 3 advanced functions: service property, service leasing, and range & logical operation based search capability. The client/application could utilise these advanced functions to help them locate services with QoS properties in service-oriented Grids. The main objectives of this paper are to introduce advanced features to the UDDI registry, to demonstrate the feasibility of such extensions, to observe the performance of the UDDIe and argue whether the proposed approach is appropriate or not for Grid application requiring QoS guarantees. It is also useful to observe any bottlenecks in the system. The experiment has shown that the system is appropriate for Grid realtime applications.

For future work, we plan to integrate this service discovery system with the other underlying resource managers of the G-QoSM framework, such as the network resource manager (NRM) and the middleware resource manager, Globus GRAM.

References


Biographies

Rashid Al-Ali is a doctoral candidate at the Computer Science Department, in the Parallel and Scientific Computation Research Group at Cardiff University, UK. His research interests are in QoS and resource management in Grid computing, and Grid services. He received his BS degree from the University of the Pacific, Stockton CA, USA, in 1992, and his MS degree from the George Washington University, Washington DC, USA, in 1997. His work experience covers foremost Mainframe Computers and UNIX systems, Network administration and IT project management.

Ali Shaikh Ali is an undergraduate student at the Computer Science Department of Cardiff University, UK. His research interests are in the areas of Distributed Systems,Grid computing ,Web services, P2P applications and the convergence of computing and mobile communications. His works experience covers the design and implementation of Web services, Grid computing clients and remote network administration.

Dr Omer Rana is a Senior Lecturer in Computer Science at Cardiff University, and the Deputy Director of the Welsh E-Science/Grid Computing Centre. He also acts as advisor to Grid Technology Partners – a US based company specialising in Grid technology transfer
to industry. He holds a PhD in Computer Science from Imperial College, London University in parallel architectures and neural algorithms, an MSc in Micro-electronics from Southampton University, and a BEng in Information Systems Engineering from Imperial College, London. His research interests are in the areas of High Performance Distributed Computing, Multi-Agent Systems and Data Mining. Prior to joining Cardiff University, he worked for over 5 years in the use of neural algorithms in a number of fields, including biotechnology, instrumentation and control.

**Professor David W. Walker** is the Director for the Welsh E-Science/Grid Computing Centre at Cardiff University. He received a B.A. degree in Mathematics from Jesus College, University of Cambridge, in 1976. His M.Sc. degree in Astrophysics was obtained from Queen Mary College, University of London, in 1979, and his Ph.D. from the same institution in 1983. He has held appointments at the University of London, the Jet Propulsion Laboratory, California Institute of Technology, University of South Carolina, and at Oak Ridge National Laboratory. Professor Walker's research interests focus on software, algorithms, and environments for computational science on high performance computers. He has been closely involved in the development of the ScaLAPACK parallel software library, and the MPI message passing standard. In recent years he has also been active in Problem Solving Environments and Agent based Computing.