Journal of Grid Computing **00:** 1–16, 2004.

Relating Requirements and Architectures: A Study of Data-Grids

Anthony Finkelstein, Clare Gryce and Joe Lewis-Bowen
University College London, Gower Street, WC1E 6BT London, UK
E-mail: {a.finkelstein,c.gryce,j.lewis-bowen}@cs.ucl.ac.uk

Key words: architecture, data-Grid, evaluation, requirements, style, suitability, survey

Abstract

The requirements and architecture of any complex software system are highly interdependent. We have studied the relationship between these two concerns in several data-Grid systems. Data-Grids are characterized by an infrastructure that focuses on the coordinated management of, and access to distributed data resources. We survey current data-Grid projects to demonstrate that a set of general requirements for data-Grid systems can be identified. Architectural styles are a way of highlighting design and engineering similarities between software systems. We consider the styles that are exhibited by current data-Grids and use a lightweight methodology to analyze how these styles support general requirements. Our conclusions provide guidelines to assist the data-Grid developer in making informed architectural choices.

1. Introduction

In this paper, we present an extended case study of the relationship between the requirements and architecture of data-Grid systems. The architectures of current data-Grids can be shown to exhibit characteristics of various architectural styles. By analyzing how these styles support the core requirements of the domain, we can identify those styles that offer 'best-fit' and provide guidelines for the engineering of data-Grid systems. The relationship between requirements and architectures is not a concern unique to the Grid domain, but an area of active enquiry in the wider software engineering community. By means of this detailed study we hope to contribute to this discussion.

Within an informal taxonomy of Grid systems, data-Grids are concerned with the generation of new information from distributed data repositories. Data-Grids yield new information in various ways, by making available to scientists an unprecedented volume of useful data. They allow more rigorous statistical analysis, and enable the application of new data-mining techniques and the cross-correlation of sets of data that have not previously been compared. Data-Grids are characterized by an infrastructure that focuses on the coordinated management of distributed data resources and the provision of data access mechanisms.

Data-Grids present many challenges to the systems developer. Many requirements are subject to change, whilst the development environment is populated with new technologies, tools and paradigms. Grids cover many scientific domains and generally include stakeholders with a diversity of skills and experience.

The requirements and architecture of any software system are highly interdependent. The architecture is the first artefact in the development process that addresses the requirements of the system. In particular, many of the desired qualities of the system commonly referred to as 'non-functional' requirements such as security, performance etc., can be largely determined by architectural choices. Conversely, architectural decisions can feedback to the requirements, constraining the system under development. For a system as complex as a data-Grid, understanding system requirements and using them to make informed architectural choices, is crucial to project success.

Architectural styles are a way of abstracting architecture instances to highlight design and engineering similarities [13]. They can be used to help the architect make informed choices about system design. A number of architectural styles for distributed software systems have been identified and documented. Each supports specific qualities, offering certain guarantees about the attributes and behavior of the deployed system. Styles can also be combined to reflect the relative importance of desired system qualities and design trade-offs. By using architectural styles as a starting point for system design, the architect can exploit the benefits of re-use, reducing risk and improving the efficiency of the development process.

In the next section of the paper, we review the key architectural styles of distributed systems and examine which current data-Grids use these styles. In Section 3, we review the system requirements of current data-Grid projects, deriving a set of general requirements for the data-Grid domain. We then analyze how the documented architectural styles support the general requirements of the domain. After this detailed review, targeted at data-Grid practitioners, we present a novel, lightweight method, of especial interest to software engineers. Section 5 describes the method for quantifiable evaluation of style suitability for fulfilling requirements. Observations, conclusions and a summary of future work follow.

1.1. *EGSO*

We are directly involved in EGSO, the European Grid of Solar Observations [21], and refer to this project as a test case. This data-Grid is being developed by 8 European and 2 United States institutions to enable a 'virtual observatory' for the worldwide solar physics community. It will provide unified access to distributed, heterogeneous solar observations and related scientific data, and form a platform for their analysis.

Since the project's launch, requirements have been gathered from data providers and scientific users. We have also collaborated with other astronomy data-Grid projects, especially AstroGrid [17] and VSO [29]. The elicited EGSO requirements were analyzed – following software engineering best practice – with use cases, MSCW prioritization and goal decomposition [5, 6].

We then investigated whether other data-Grids had identified similar requirements to those of EGSO, hoping to reuse suitable technology and good design patterns. It became clear that there were common, challenging requirements that characterized data-Grids. However, we did not find any documentation that abstracted these and fitted them to generic solutions. Reuse, especially of distributed system technology, does commonly occur in data-Grids, but in an informal way, without clear application of engineering principles.

This paper documents our review of data-Grids and their common requirements. The analysis of architectural style suitability also presented is a technology independent solution to many of these challenges. It should therefore serve the wider community of data-Grid planners, managers, architects and developers.

1.2. *Projects Surveyed*

Our review of data-Grids includes eight further projects, listed below. These were identified as suitable for study owing to their focus on the federation of data resources, and their combined coverage of a variety of application domains (particle physics, biomedical and bioinformatics, astronomy and Earth observation). The ready availability of information about project requirements, architecture and services was also a determining factor; information gathered and presented in this paper was from the project websites at the time of writing, including informal project documents and web content, as well as more formal papers.

1.2.1. *AstroGrid*

AstroGrid [17] aims to build a data-Grid for UK astronomy, ultimately contributing to a global Virtual Observatory. It aims to deliver a working data-Grid for key selected databases, with associated data-mining facilities, by late 2004. AstroGrid will cover astronomy, solar physics, and space plasma (solar terrestrial) physics, through a partnership between UK archive centers and astronomical computer scientists.

1.2.2. *BIRN*

The Biomedical Informatics Research Network (BIRN) [18] is a US based project that aims to foster large-scale Biomedical science collaborations. This will be made possible through an infrastructure enabling data integration, high speed networking, distributed high-performance computing and application software. Three 'test bed' projects including groups working on a variety of applications will be used to drive the definition, construction, and use of a 'federated data system'. The vision of the project is to enable the testing of new hypotheses through the analysis of larger patient populations and multi-resolution views of animal models through data sharing and the integration of site independent resources for collaborative data refinement.

1.2.3. *EDG*

The European DataGrid (EDG) [20] is an expansive EU funded project. It aims to enable access to geographically distributed compute power and storage facilities belonging to different institutions across Europe. The project uses three scientific disciplines with different application and domain specific needs as drivers; high-energy physics, biology and Earth observation. Running from 2001–2003, the first and main objective for the project was the sharing of huge amounts of distributed data over the existing network infrastructure.

1.2.4. *ESG*

The Earth System Grid (ESG) [22] is a US project with the primary goal of addressing current challenges in the analysis of, and knowledge development from global Earth System models. The project will use generic Grid technologies and application-specific technologies, distributed supercomputing resources and large-scale data and analysis servers to create a seamless and powerful environment for climate research.

1.2.5. *GriPhyN*

The Grid Physics Network project (GriPhyN) [24] has the primary objectives of providing the IT advances required to enable Petabyte-scale data intensive science. Driving the project are four physics experiments that produce extremely large volumes of data, and the need for scientists to be able to extract complex information from this data independent of geographic location. To meet these challenges, GriPhyN focuses its research on realizing the concept of Virtual Data; the definition and delivery to a large community of a virtual space of data products derived from experimental data.

1.2.6. *myGrid*

The myGrid project [25] is targeted at developing middleware to support in-silico experiments in biology on a Grid. In contrast to other projects based around Biomedical or bio-informatic applications, myGrid focuses on the resolution of issues arising from the semantic complexity of data and services, such as resource discovery, workflow enactment and distributed query processing.

1.2.7. *NVO*

The US National Virtual Observatory program (NVO) [26] is collaboration aiming to investigate frameworks for the construction of a virtual observatory. This includes research into and development of standards and protocols for data exchange and access. The project has built several application prototypes to drive this research, working cooperatively with the astronomical community.

1.2.8. *PPDG*

The Particle Physics Data Grid collaboration (PPDG) [27] is driven by the needs of current and near-future research in particle and nuclear physics. It draws on the requirements of a wide range of experiments, aiming to develop an early Data Grid architecture and evaluate prototype Grid middleware. Project goals and plans are ultimately guided by the immediate, medium-term and longer-term needs and perspectives of these representative experiments, some of which will run well beyond 2010.

2. Architectural Styles

Architectural styles [13] are high level design patterns [10] that describe software systems in terms of logical components and connectors. Their abstract description assigns key properties, relationships and responsibilities in a decomposed view of the system.

Five well established, distributed system styles are introduced below and applied to data-Grids. Key architectural features are described for each. As the application of an architectural style is commonly and sometimes ambiguously stated, they are defined here in terms of component communication. Examples of existing technologies that use the styles are also given.

Subsequent evaluation of reviewed projects use of each style has typically been inferred from available documentation. Where a style is not explicitly cited, it may be inferred by the function and interaction of systems' components. In some cases, technology choice imposes architecture, so a few data-Grid technologies are also reviewed. Our findings are summarized in Table 1.

2.1. *Layered*

A system may be simplified by dividing it into layers with interfaces. Each layer has unique responsibilities, and distributed instances have a direct virtual communication path. In this way, programs at one

Table 1. Summary of projects' architecture styles.

		Architecture				
		Layered	n-tier	Peer-to-peer	Dataflow	Agent
Tools	Condor-G		Y		Y	
	Giggle		Y			
	Globus 2	Y	Y			
	Spitfire		Y			
Projects	AstroGrid		Y	Y		
	BIRN		Y		Y	
	EDG	Y	Y	Y		
	ESG		Y		Y	
	EGSO		Y	Y		
	GriPhyN		Y	Y	Y	
	myGrid		Y			Y
	NVO		Y			
	PPDG		Y	Y	Y	

layer can ignore issues handled in other layers, simply relying on their service. At the highest layer, the application may use an API without coupling to its implementation, whilst at the lowest layer the physical operation may be implemented mechanically, ignoring the variety of use and design subtleties at higher levels.

The logical content of data and control messages (information and commands) are translated by the layers to diverse representations. Enterprise databases (integrating the heterogeneous schema of distributed repositories) and high level programming languages (supported by compilers and virtual machines) are examples of layered architectures.

Observed Application

The use of true, layered architectures is not evident, though some projects use conceptual layers to describe the system from a functional perspective. EDG follows the layered Grid Architecture of Foster and Kesselman [8]. This is a reference model in which layers are defined by the general function of their components and the interactions between them. In common with true layered architectures, components in each layer can use the capabilities provided by lower layers. However, the Grid architecture is actually a variation of a true layered architecture, as it allows some degree of layer 'bridging', with higher layers communicating directly with lower layers rather than through intermediate layers. Also in common with layered architectures, the Grid Architecture describes how layers are defined by communication protocols.

2.2. *n-tier*

Business logic (functionality associated with a user's needs) may be separated from process logic (technical solutions for classes of application) using tiers. This architecture allows flexibility and transparency from the front end user driven behavior to the back end system administration. Transparency allows homogenous use of diverse distributed, and the redundancy and growth that supports reliability and scalability. This is enabled by components' platform independent interfaces. The middleware that enables tier abstraction typically provides minimal basic services via core component interfaces. Systems may reuse components within this framework to build their functionality.

Interaction about a tier is independent but connected; messages used by the application have a many-to-many relationship with messages using back end resources. CORBA and J2EE provide component based middleware for diverse distributed systems in conceptual tiers. Generic interfaces define web services on application servers such as WebSphere.

Observed Application

The layered EDG model also has n-tier characteristics, with functional components that can be deployed independently. It reuses components from the Globus [23] project alongside EDG specific initiatives, including the following core components. The Replica Location Service, instantiated by Giggle distributed components [4], maps logical to physical file names flexibly and hierarchically. The Metadata Catalog uses Spitfire [28], a web service with local and a global layers, to provide a uniform interface to distributed metadata resources. Reptor, a reference implementation of the Replica Management Service that offers a single point of entry to the core capabilities, exposes web service interfaces with a configuration API.

Other projects make their underlying architecture of distributed components more explicit. PPDG and GriPhyN re-use Globus, Condor [19] and other data-Grid projects' components. Their Virtual Data Toolkit subsystem uses a workflow framework for data-product discovery and re-derivation. ESG also reuses Globus Metadata Catalog Services (MCS) and Giggle components with domain specific analysis and visualization components. BIRN reuses Metadata Catalog (MCAT) and Storage Resource Broker

(SRB) [3] for data retrieval, with other functional components. These include plug-in visualization servers and the Data Mediator that maps between knowledge domains.

AstroGrid, myGrid and NVO adopt web service technologies. AstroGrid is OGSA [15] compliant, and the NVO testbed integrates web services with the Grid technology components MCS and Giggle. NVO also off-loads large computational tasks to subsystems, whilst both use a registry component for resource discovery. The myGrid project wraps existing domain tools in web services alongside Globus components, Condor and SRB – used for Grid task management and uniform data access.

EGSO's architecture has three tiers of subsystems, each built from encapsulated components, for participating functional roles – Consumer, Broker and Provider. The abstract architecture is not tied to specific technologies.

Tiered architecture is further supported by data-Grid components that go beyond Grid functionality; Globus components also provide monitoring and security capabilities. The thin, web-based clients typically provided for data-Grid users also demonstrate that designers have adopted the *n*-tier style.

2.3. *Peer-to-Peer*

Peer-to-peer nodes have symmetrical relationships, for example functioning both as client and server when creating and performing service requests. In a peer-to-peer network, a large number of nodes may share resources without dependence on central points of control.

Communication sessions in peer-to-peer networks are typically a triangular sequence of requests for service until a match is made, then service invocation, before the reply to the origin. IP networks have peer-to-peer characteristics, though file sharing services such as Gnutella are the paradigm of this architecture. JXTA is a flexible middleware for peer-to-peer resource sharing.

Observed Application

Data-Grids are rarely explicitly described as peer-to-peer networks, though descriptions of subsystem interaction suggest emergent peer-to-peer architecture. In particular, components for resource discovery and metadata management are generally distributed implementations in which peers forward messages.

The EDG metadata catalog uses Spitfire with a global layer for transparent access to metadata resources – in distributed implementation this could exhibit peer-to-peer behavior. The MCS of PPDG/GriPhyN is distributed by partitioning and replicating metadata – as this would be transparent to the user, queries must be forwarded between nodes in peer-to-peer fashion. Conversely AstroGrid's tiered resource registries forward metadata updates. EGSO explicitly describes the Broker subsystem as a distributed infrastructure for marshalling user requests and managing metadata resources. Broker instance interaction supports fault tolerance whilst presenting a homogenous service for other subsystems.

2.4. *Dataflow*

Processing components may be organized in sequence, so that the output of one forms the input of the next. Branching is possible to allow concurrent progress, but may require later synchronization if paths rejoin. Different scheduling strategies may be used to suit the functional requirements, and may require some intelligence to make the best use of resources.

The messages between components for one job have different content after each transformation. A pipeline of processes or filters (such as in a Unix shell script) is an example of data-flow architecture, and many parallel computing tasks (such as finite element simulation) run in a data-flow sequence.

Observed Application

Several projects include subsystems with the data-flow architecture's characteristic interaction pattern. BIRN uses a data pipeline processing architecture for analysis and visualization in modular toolkits integrated with other components. ESG also specifies analysis and visualization components, including 'filtering servers' for running user-specified analysis routines.

Particle Physics' key requirement for derived data products has driven the PPDG/GriPhyN Virtual Data Toolkit, in which datasets are defined by transformations. This enables data product re-creation through workflows, with parallel task management supporting by Condor-G.

2.5. *Blackboard/Agent Based*

Complicated tasks can be tackled by dividing work amongst software agents, running concurrently on distributed platforms and using a shared 'blackboard' data area. This architecture may solve a problem by applying a variety of analytic or heuristic methods to one data set, or find information in the content or relations of distributed data sets.

Simple agent protocols only pass messages (only differing in content) via the blackboard (a shared critical resource). Artificial intelligence and data mining applications use this architecture for information processing.

Observed Application

The Bioinformatics community has a large number of heterogeneous, rapidly evolving data resources. The myGrid project architecture uses agent technology to notice changing 'views' of project resources. The 'open platform' for data and tool interoperability acts as a domain blackboard; changes trigger user notification events.

2.6. *Hybrid Styles*

It has been demonstrated that many projects use several architectural styles. Characteristics of different styles may be legitimately combined in a software system. Cumulative benefits may be gained, and hybrid styles are pragmatic when systems are built of subsystems (including legacy architecture). Even though pure architecture is rarely implemented, well chosen styles should still help to meet non-functional requirements. Evaluating the relative benefit of each style is made harder, however, by overlapping design solutions.

3. Current Projects – Requirements

A set of 83 general requirements for data-Grids were derived from our gathered information of the requirements of the representative projects listed in Section 1. These are summarized under 18 headings in this section, organized in three classes: characteristic requirements, functional requirements and non-functional requirements. The first are the broadest, representing properties that characterize a distributed system as a data-Grid. The second group are more specific, describing what the system must do to fulfill its characteristic requirements. The third represent other traits that the system should demonstrate, frequently constraints on the former. Though some avoid the terms 'functional' and 'non-functional' requirements, saying their respective use is contextual, our definitions make them clear and useful in this work.

We found a notable lack of documentation describing requirements in a formal or systematic manner. Instead, requirements were typically stated as high-level system goals or application specific objectives. Though we assumed this informal information is incomplete, similar high-level system objectives emerge. From these we draw conclusions about the general, domain-independent requirements for data-Grids. Relative priorities were also abstracted, and recorded for each derived requirements according to the popular MSCW (or MoSCoW) scheme – for 'must', 'should', 'could' and 'would like'. Often projects' documents used these terms informally, or used other phrases that implied priority such as "it is important that". When requirements are frequently stated in diverse projects, we also judged them higher priority paradigm data-Grid requirements. Conversely, we ranked rare, potentially domain specific requirements 'could'.

The complete set of general data-Grid requirements with their priorities is given elsewhere [1]. That table also shows which projects referred to each requirement. Some of those requirements are shown here in Table 4. The following summarizes and discusses them.

3.1. *Characteristic Requirements*

1. Data Resources

The primary purpose of a data-Grid is to include distributed, possibly heterogeneous data resources in a single networked system. The resulting data-Grid may be considered as one, logical resource. A data-Grid is required to be able to include data resources that are distributed across normal boundaries of access; i.e. geographical, administrative or organizational.

2. Access to Resources

The users of the data-Grid require access to its resources. Specifically, they need to discover and use the available resources. Access is generally required to be location transparent; from the user's perspective, the data-Grid offers a single, 'virtual' data resource.

3.2. *Functional Requirements*

3. Data and Data Management

All projects require the ability to include data of various formats and structures. This may be commonly used data formats (e.g., for images), or domain specific (e.g., Astronomy FITS files). Data can also be categorized as raw, processed or annotation data. The relative representation of each of these types varies between projects.

Most projects require multiple copies of included files or data set. In the majority of cases these are replicas for network optimization. In the Biomedical domain there can be multiple proprietary formats of each file. Several projects distinguish underlying media, with special treatment of tape archives. An associated requirement is for data to have both logical and physical identifiers. Many projects also require that users are able to create their own logical views or collections of data.

4. Metadata

All projects require support for existing domain metadata standards. Most require easy interchange between domain standards to create a comprehensive metadata framework. Many require that this framework be extensible, with users able to create their own metadata at various levels of granularity.

Some projects require the automatic extraction or generation of metadata for given datasets or new data products. This implies automatic catalogue update.

5. Data Querying and Data Access

Most projects require both ‘push’ and ‘pull’ data querying techniques. Users should be able to submit queries based on attributes of data (through the use of catalogues and indices), or based on pattern matching or data mining methods. Several also require support for user-built complex queries, termed ‘pipelines’ or ‘workflows’.

Most discuss the capability to run queries that span multiple data resources as an ‘advanced’ requirement, though it is given high priority by the Biomedical and Astronomy domains. Projects in Physics and Astronomy domains require data-access granularity within files. The Biomedical, Earth observation and Sun–Earth domains require rapid and frequent access to their volatile data.

6. Data Processing

All projects require processing resources to be available as part of their Grid system. Projects and testbeds serving the Physics community place the greatest emphasis on this, to incorporate the many distributed, heterogeneous compute resources currently used within the community.

Commonly there are special requirements for processing data stored on resources remote from computation resource; Earth Observation and Astronomy projects also emphasize portable user code. Another special case is for processing across multiple data resources.

Processing resources are generally required to support computationally intensive and lengthy tasks. Some projects explicitly specify parallel processing capabilities or pipeline support. In some cases temporary local storage resources are required for data staging.

7. Data Transfer

Most data-Grids need to transfer entire datasets. Particle Physics projects require continuous network traffic from data production centers to tiered data resource nodes.

8. User Interface and User Functions

Several projects require usable interfaces for users in a variety of roles, including some that are not IT literate.

Key user functions include: data browsing, data selection and query, local data visualization, browsing and access to analysis services, uploading user code, data management, account management, tracking and organizing active jobs. The interface should support several of these in the same user session through an integrated workbench.

On-line help and, in some projects, collaborative workspace are also required. In all cases, interfaces must by highly interactive. Graphic web portals are typically specified. Some projects require that a user tasks persist after disconnection.

9. Applications and Tools

Most projects require integration with existing applications. Users may be able to create new functionality via APIs or by composing-services. Astronomy projects go beyond reusing existing visualization tools; users should be able to browse synoptic images that summarize data.

10. System Information, Monitoring and Tracking

All projects explicitly require that users or administrators can access information about the system itself, including static resources metadata and dynamic information about system state. This information is used for higher-level capabilities: error detection and tracing, application and job monitoring, performance optimization, task evaluation and scheduling, resource management, metering and accounting. Particle Physics projects have notably detailed requirements for the such capabilities.

11. Resource Management and Scheduling

A related ubiquitous requirement is for the management of work over distributed resources. At the most basic level, jobs need to be matched to resources in an optimal manner. Many projects also require job priority management, and bottleneck identification and correction. Particle Physics projects emphasis this, specifying interactive resource allocation that allows re-negotiation of running jobs. A requirement for check-pointing is also typical.

12. Interoperability

Many projects need to work with other Grids in related domains. The noted need to support existing metadata standards is partly motivated by this intercommunication requirement.

3.3. *Non-functional Requirements*

13. Security

Most projects do not state security requirements in depth. All specify a need for authentication (verifying the declared identity of a system user or resource) and authorization (linking that identity to a set of permitted actions), sometimes with auditing (recording the actions carried out by system entities). Auditing is usually refined to accountability (of users and resources for their actions) and management of usage quotas (or billing).

Further requirements for security are documented in general terms, usually referring to 'ideal', networks. Though specific requirements are not described, the following are implied by such discussion: the system should respect all types of local security policies, should allow users to be mobile, and should ensure the integrity of data. The problem of exposing all data, while ensuring robust security services, is also noted.

14. Load, Capacity and Scalability

Projects commonly state the required data volume of systems to be 'Petabyte scale', Particle Physics being generally higher. Individual file and data set sizes varies widely; 20 MB to 2 GB files, and 1 TB to 100 TB data sets are mentioned. Expected growth rates are also domain dependent, from over 1 PB per year in Particle Physics to 10's of TB in other domains.

Required capacity can be given by the number of included data resources. Few projects include a known number of existing resources; others state open-ended requirements, indicating a requirement for scalability. Capacity is also represented by the planned number of users, with figures of between 1000 and 100,000 cited.

Anticipated system load is rarely stated in data-Grids. It is generally suggested that systems should support 10 to 100 times the number of processes of standard computing nodes.

15. Performance

Most requirements for performance framed as resource management and scheduling, described above. These indicate a need for optimum service levels to be maintained as system load and system state change – relative rather than absolute terms.

Some projects specify query response times between 5 and 10 seconds. Earth observation and Biomedical domains state the need for 'near real-time' processing of data.

16. Fault Tolerance and Robustness

General requirements for fault tolerance, or robustness, are not specified in detail. Where given, they are stated with reference to particular services: security services should not have any possible single point of failure, data access services should show some degree of fault tolerance. It is a general requirement that the system should offer capabilities for the recovery of jobs that are running in the event of system failure.

17. Extensibility and Modifiability

Requirements for extensibility and modifiability vary between projects. Where stated, adding new functionality to a system once deployed is given a high priority. This is usually described as adding new services, discovered via standard mechanisms. Most projects require portability of some system components, notably user and data resource interfaces.

18. Integrability

All projects require heterogeneous component integration, whether project specific or legacy. Some projects plan to integrate components or tools that are in development, at various release stages.

4. Style Suitability

In Section 2 we introduced and discussed the architectural styles demonstrated by current data-Grid projects. In this section, we summarize the ways in which these styles variously support the general requirements for data-Grids presented above. A complete, more formal presentation of this information, used for the analysis described in Section 5, is given in elsewhere [1].

1. Data Resources

Layered and tiered styles support the transparency required to present distributed, heterogeneous resources as one logical entity. n-tiered and agent architectures may support a single point of entry. Implemented peer-to-peer networks also host diverse data types.

2. Access to Resources

Tiered and peer-to-peer networks support location transparency, allowing users access to unknown resources. Peer-to-peer networks may go further to render location anonymous, whilst n-tier middleware also hides resource duplication and migration. Agents may indirectly support resource discovery by creating catalogues in advance of user resource look-up.

3. Data and Data Management

Detailed data management requirements are largely resolved at lower levels of design. However, a layered paradigm could help data format abstraction. n-tier middleware typically uses basic data types abstractly and marshals data structures at the OSI presentation layer.

4. Metadata

Strong requirements for diverse, flexible metadata schema further support styles that offer abstraction, notably layered and tiered architecture. Both allow heterogeneous, volatile low level or back-end schemas to be presented homogeneously. Though a tiered middleware introduces additional metadata, it should be very flexible. A peer infrastructure that separates discovery from content may support diverse metadata too.

The requirement for automatic metadata generation could be met by the “divide and conquer” method of peer-to-peer and agent based architectures. Filters or agents may be employed for the metadata transformation requirement, possibly helped by a standard layer connection protocol.

5. Data Querying and Data Access

The client–server solution to traditional query services is a simplification of n-tier architecture. An additional middle tier could coordinate distributed queries and handle different granularities transparently. Agent and filter methods are well suited for pattern and data-mining queries (and may work within files). For rapid access, concurrent task management in a pipeline coordinated by a middle tier would be more suitable than agents.

6. Data Processing

Tiered architecture decouples the application from back end activity, allowing a variety of distributed resources to be used for lengthy or concurrent operations. Peers and agents may support mobile tasks that make progress on diverse resources in parallel. Pipeline architecture is also well suited for executing lengthy tasks in parallel, exploiting variation in the capabilities of resources.

7. Data Transfer

Well-established protocols satisfy reliable data transfer by implementing the OSI layers. Pipeline architecture may also be applied for parallel data stream control.

8. User Interface and Functions

The abstraction provided by tiered (and layered) architecture allows separation of client roles, and may provide a virtual platform for mobile code and host mechanisms for account management. Both tiered and peer-to-peer networks support transparent service discovery and use, and therefore allow task distribution, decentralized data management and user collaborations. Offline task progress may be managed by any architecture that decouples the current job state from the application.

9. Applications and Tools

Tiered systems satisfy the requirements for transparent access to legacy and future services (or tools that build service based applications). Layers support abstraction of diverse back end services, whilst peer infrastructure helps advertisement of new services. Abstract service descriptions presented in peer and n-tier networks may be composed into pipelines presented to the client.

10. System Information, Monitoring and Tracking

Tiered middleware components typically maintain metadata about a distributed system’s configuration and state, and offer core services to access them. Peer networks are intended to be dynamic, minimizing static data requirements; nodes typically only maintain accurate data of the current local environment. Agents may also be deployed to discover distributed status information.

11. Resource Management and Scheduling

All architectures that separate the client from other system components support distributed task management. Dispatching and managing jobs on suitable resources are basic operations for n-tier and peer networks, parallel pipeline schedulers and even mobile agents. The n-tier architecture is well designed to simplify management across heterogeneous resources.

Queuing may be implemented in any of these architectures, possibly using a ‘time to live’ attribute in peer-to-peer networks. Peer networks are intended to be free of centralized bottlenecks, whilst a pipeline scheduler and tier configuration management should make it possible to avoid them. Task recovery and renegotiation is supported by pipeline mechanisms (by check-pointing, steering and staging) and middle tier management components.

12. Interoperability

A common protocol at one layer could hide differences at lower or higher levels. A portal to a different network may be presented homogeneously in a tiered architecture. Pipelines may also be used to transform communication between diverse resources (as demonstrated in compute-Grid systems).

13. Security

By separating users from resources, tiered systems offer a mechanism for enforcing security measures. The middleware may organize a certification process (possibly involving third parties) and manage heterogeneous policies for reliable authentication and authorization. This functionality forms the foundation of other security requirements for accounting, auditing, single sign on and data integrity checks. However, using a separate tier may reduce the availability of underlying resources, whilst heterogeneous or changing policies may impede pipeline tasks across resource boundaries.

Peer networks may enforce signature exchange and generate ‘crumbtrails’ to support audits, though these techniques have typically been used to ensure anonymization and integrity. A security layer may also be employed to validate certificates and data integrity.

14. Load Capacity and Scalability

Tiered architecture separates control from back-end interactions, and therefore manages growth and large data resources well. However, the required distribution configuration and potential bottlenecks make tiered solutions weaker than peer-to-peer networks when scaling to very large numbers of tasks and resources. Pipeline schedulers have proven to scale to large numbers of tasks, and may facilitate very large data set access with parallel streaming.

15. Performance

Pipeline task schedulers can optimize resource usage, and may ensure synchronous progress when processing a real-time data stream. Tier networks also offer mechanisms for performance monitoring and configuration management to optimize a system. However, both these styles and peer-to-peer task distribution are less suitable than direct resource control for rapid interaction as their response times may be slower. A mobile agent architecture is likely to offer even worse performance as activities may take an arbitrarily long time to end.

16. Fault Tolerance and Robustness

Tiered middleware coordinates shared responsibility, allowing managed fail-over to keep security mechanisms and services operational. Pipeline checkpoints would facilitate job transfer from a failed resource. Peer networks are designed to provide fault tolerant routing and may also support redundant service nodes, ensuring elastic service degradation.

17. Extensibility and Modifiability

Abstraction layers help extension and portability of lower level facilities, providing common presentation to applications. n-tier and peer networks also hide underlying heterogeneity, supporting flexible platforms and service extension; tier middleware and peer advertisement services present abstract meta-data descriptions of underlying functionality.

18. Integrability

The primary goal of n-tier architecture is the integration of heterogeneous components. To support extension, they may enforce constraints on new components with compatible configuration. Peer networks also integrate diverse nodes. Layered abstraction helps the integration of diverse low-level elements using a common protocol.

5. Style Evaluation

5.1. *Method*

Section 2 noted how some documented architectural styles are suitable for data-Grids. We gave an informal impression of how general data-Grid requirements may be met by high-level design. In this section we describe our method for making a quantified evaluation. The 5 architectural styles are scored against the 83 data-Grid requirements described in Section 3. The complete matrix [1], partially reproduced in Table 4, is summarized by the 18 requirement headings of Section 3 in Table 2.

Style suitability was judged intuitively, and this method is therefore subjective and not necessarily reproducible. However, it is equivalent to industrial best practice, whereby experienced software developers decide to reuse components (including function libraries, sub-systems and design patterns) on how they expect them to fit requirements. This method efficiently covers a very large design space, considering whether a rich variety of possible systems would meet many requirements. A more thorough, tractable method would be prohibitively laborious, requiring experimental proof of design properties and their formal association to requirements. Our method is efficient and reliable, assuming the styles achieve that which they're designed for.

A strong positive score (2) was awarded to styles whose explicit purpose was the satisfaction of the given requirement. There are several requirements that data-Grids share with other distributed (data-intensive and high performance) systems, and therefore established styles have been created specifically to resolve some of these.

Where this was not true, a positive score (1) was indicated for styles that should still help to satisfy the given requirement. This score may be given if technology associated with the style have historically exhibited the required behavior, or if primary features of the architecture may be adapted to satisfy the requirement.

A negative score (-1) was given to a style that undermines a requirement. This may be because the goal of the architecture contradicts the requirement, or

Table 2. Architecture fit summary.

Requirement		Architecture						
		Layered	*n*-tier	Peer-to-peer	Dataflow	Agent	Sensitivity	
1.	Data resources	++	++	+		−	6.0	High
2.	Access to resources		++	++		+	5.0	High
3.	Data management	+	+				0.3	Low
4.	Metadata	+	+	+		++	2.7	Medium
5.	Data querying	+	++		+	+	1.7	Low
6.	Data processing	+	+	++	++	+	3.8	High
7.	Data transfer	+			++		2.0	Low
8.	User interface	+	++	++	++	+	1.9	Low
9.	Applications tools	+	++	−	+		2.0	Low
10.	System information		+	−		+	2.5	Medium
11.	Resource management		++	++	++	+	3.2	Medium
12.	Interoperability	++	++		+		5	High
13.	Security	++	++	+	−	+	1.8	Low
14.	Load capacity	−	−	++	+		2.6	Medium
15.	Performance		+	−	++	−	2.3	Medium
16.	Fault tolerance		+	++	+	−	3.0	Medium
17.	Extensibility	++	++	+			3.5	High
18.	Integrability	+	++	+			3.5	High
Suitability		27	63	41	24	16		

Table 3. Demonstration of method for deriving architectural sensitivity, with style suitability summary and running total, from matrix of requirements and styles detailed in [1].

Requirement	Architecture					Absolute total	Average sensitivity
	Layered	n-tier	Peer-to-peer	Dataflow	Agent		
16.1		1	2		−1	4	
16.2		1	2			3	
16.3		1		1		2	9–3
Summary		+	++	+	−		
Total	0	3	4	1	−1		

mechanisms typically implementing the style would have a negative impact on the required behavior.

A neutral ranking (0) was given when the architecture has no obvious impact on the requirement, or has balancing positive and negative effects. Many data-Grid requirements were neutral for several styles, as the abstract systems described by the styles and the core technology that implements them would not fulfill the requirement; the behavior would be implemented within a component or performed by a related subsystem.

The detailed requirements are listed with the score given for each architectural style that impacts its resolution. A brief reason for the score is given in each case. A section of the complete table given in [1] is shown in Table 4. (In this the original matrix of requirements against styles has been flattened for presentation.)

The symbols in Table 2 indicate the strongest score in the given group of requirements. For example, the tiered style is marked with '++' for requirements group 18 as it scored 2 for requirement 18.1, even though it only scored 1 for 18.2.

The average absolute value of the scores across styles for each requirements group indicates that class of requirement's architectural sensitivity; a low score indicates architecture choice does not much influence whether a requirement can be met. The values are given in the right-hand column of Table 2, with the words 'high', 'medium' and 'low' indicating which third of architectural sensitivity scores the requirement group falls into.

By simply summing the scores for each style for all requirements, the style's overall suitability for data-Grid architecture is indicated; a low score either indicates that the style cannot meet the requirements or actually hinders their fulfillment. The values are given on the bottom row of Table 2.

To demonstrate these operations, a fragment of the matrix is given in Table 3 with the averaged architectural sensitivity scoring, the partial architecture suitability sum and the summary symbols. As noted, the details of requirements and style scores' justification are in [1], but the relevant section is shown here in Table 4.

This method was inspired by design space analysis [11], the 'House of Quality' [11] and simpler methods of linking requirements to system components. The candidate designs to be placed were the five styles. However, this analysis did not specify a system property space. The dimensions of behavior would have to be built from simplified abstractions of the 83 stated requirements, hiding the detail that defines the data-Grid problem. That is why we scored each style was scored against every requirement.

5.2. *Observations*

The total scores of architectural style suitability for data-Grids across the bottom of Table 2 were all positive. As the five styles considered were those observed in current data-Grid projects, this demonstrates that wholly unsuitable architecture are avoided. The most commonly used style, n-tier, actually scores highest, supporting the choice of real projects and this methodology.

The peer-to-peer architectural style is also highly ranked, supporting the convergence of data-Grids and peer networks noted in the community [9]. This method has identified specific requirements met by a peer-to-peer solution.

Table 4. Fragment of the matrix [1] of detailed requirements (with priority and origin) against the architectural styles' suitability score (with justification).

16.1		The security services of a data-Grid should not have a single point of failure.	2
Project:		(Inferred)	
Tier:	1	Tiers may coordinate shared responsibility, so a validation task may fail over to a redundant resource when the primary service point fails.	
Peer:	2	Decentral peer networks are ideal for elastic degradation of service as sub-sets of the network may continue to make progress on node failure.	
Agent:	−1	If a blackboard were used for any part of the security process, this would be potential single point of failure.	
16.2		The data access services of a data-Grid should be faulty tolerant to some degree.	2
Project:		(Inferred)	
Tier:	1	A middle tier may handle transfer to redundant nodes on primary failure to ensure continued data access service.	
Peer:	2	Peer networks are highly fault tollerant with respect data routing.	
16.3		A data-Grid should have capabilities for job recovery in the event of system failure.	2
Project:		(Inferred)	
Tier:	1	Middleware may coordinate transfer of task state from a failed resource to store or another resource.	
Pipe:	1	Workflows may include checkpoints that allow for job recovery.	
18.1		A data-Grid must allow existing heterogeneous components to be successfully integrated, as necessary.	1
Project:		EDG, PPDG, GriPhyN, BIRN, ESG, NVO, AstroGrid, MyGrid, EGSO.	
Layer:	1	Layered abstraction of low-level platforms enable component integration.	
Tier:	2	A primary aim of the transparency enabled by a middle tier is heterogeneous component integration.	
Peer:	1	Peer networks typically integrate heterogeneous nodes (which may host heterogeneous components).	
18.2		A data-Grid could allow heterogeneous components that are not yet available, to be successfully integrated.	3
Project:		EDG.	
Layer:	1	Layer abstraction also enables integration with future diverse low-level elements.	
Tier:	1	Middle-tiers should enable future integration, but may enforce component responsibilities to allow compatibility.	
Peer:	1	Peer network flexibility should extend to future uses.	

Pipeline and layered architectures score well too. These styles are closely associated with parallel computing and communication, and to some extent with filter transformation and data management; all key components of data-Grids. These styles only offer a partial solution to the problems that must be resolved in a typical data-Grid though. Though agent technology scores lowest and apparently only offers specific behavior at the fringes of data-Grid operation, it does help to meet many requirements.

Conclusions may also be drawn from the requirements sensitivity to architectural style sensitivity to the right of Table 2. The characteristic data-Grid requirements proved highly sensitive to architectural style. As these describe top-level system goals, this demonstrates that high-level architecture design determines overall data-Grid behavior.

Other data-Grid requirements that were highly sensitive to architectural choice concern flexibility (interoperability, extensibility and integrability) and data processing (another key data-Grid function). Most other non-functional (load capacity, performance and fault tolerance) and data-Grid management (metadata, system information and resource management) requirements showed medium architectural sensitivity. That left security and straight-forward functional requirements (data management, querying and transfer, and user interface and application tools) with low demonstrated sensitivity; they can more easily be encapsulated to fit any style. The ranked architectural sensitivity of requirements reinforce the importance of sound architecture for data-Grids.

6. Discussion

We have conducted a semi-formal investigation into the general requirements of an emerging domain, the data-Grid. We have also examined the architectural styles evident, and their suitability with regards to the fulfillment of these requirements. Some particular observations and limitations of the study are outlined below.

6.1. *Style Evaluation*

We have proposed a methodology by which architectural styles may be analyzed to evaluate their suitability for the fulfillment of stated requirements. The study has examined five architectural styles that are dominant in the distributed applications domain from which data-Grids emerged. Other styles could have been evaluated with these. The n-tiered architecture could have been refined into distributed component and web service styles; this decomposition would lead to discussion of specific technologies' suitability – beyond the scope of this review. The pipe and filter style could be separated from various parallel execution paradigms; the reliable scheduling strategies of these may satisfy specific computation requirements, but at a detailed design level not considered here. A simple client–server architecture was not discussed as it is so weak in an Internet scale distributed context. Mobile agent architecture, exemplified by Internet 'bots, are related to the agent style (least applicable of the five styles); they would also only satisfy a subset of the requirements related to their functional purpose.

The five styles examined could also have been more strictly defined, possibly by their abstract interfaces or event transitions. For example, essential operations for agent architecture may include spawning new agents and agent writing to a shared blackboard. At this level of detail it is unambiguous how a style supports required behavior. However, such strong characterization would exclude existing projects that use a hybrid style or hide the characteristic style interfaces in design detail. Our technique allows real-world solutions to be informally classified against the five styles discussed.

6.2. *Formalism*

If architectural styles were strictly expressed, the suitability of candidate architectures for the fulfillment of requirements could be determined by formal analysis methods. Architectural properties could be proven to satisfy declarative requirements by finding consistency. However, this method is of limited value in the current data-Grid domain, as declarative constraints cannot be reliably derived from lengthy, ill-defined requirements.

Formal methods have been applied to analyze the architecture of specific projects [7] and to generally distinguish Grid systems from distributed systems [14]. The former work used an event transition language rather than a calculus for static relations. In this way, models could be rapidly generated and easily interpreted from informal high-level designs. However, it is not clear what benefit models of generalized Grid architecture would be.

6.3. *Non-Functional Requirements*

We have seen that although the functional requirements for data-Grids are fairly well defined, the non-functional requirements are expressed rather more informally. Despite the low emphasis given explicitly to non-functional requirements in documents available, in many cases appropriate architectural choices can be inferred from stated functional requirements. For example, the requirement for interoperability with other Grids implies the need for extreme flexibility and the decoupling of applications. These requirements are well met by the abstraction enabled by n-tiered systems.

Some types of non-functional requirement are not approached directly at all. For example, usability is discussed only in so far as buy-in for less technical users must be low, and in the operation of interfaces. However, no explicit guidance is given on interface complexity, training and documentation requirements, or the boundary between front-end service composition or administrative interfaces and back-end hacking. Likewise, security and reliability are given limited attention, and so do not inform architectural direction as much as might be expected. This is particularly true of security, where requirements are frequently described in very coarse-grained terms, or may be stated in terms of solutions.

6.4. *Generality*

It is apparent that as well as strong core requirements, data-Grids share some other lower priority requirements. It is interesting that the domain can also be characterized by its weak positive requirements, and this may reflect that all projects currently have similar long-term goals. However, a light overall weighting is also given when a single project has a strong need for a requirement that is specific to an application domain; for example, extra sign-on points for authorization to use medical records. These therefore appear to be low priority general requirements when they are actually critical for just a subset of projects. Such differences in emphasis point to the need to develop a more refined taxonomy of data-Grids, before undertaking a more comprehensive study of appropriate architectural styles for the domain.

6.5. *Context*

In order to fully understand the relationship between requirements and architectures in the data-Grid domain, this relationship must also be considered in a broader context. The Architecture Business Cycle (ABC) [2] is a model that describes three key factors that influence the architecture of a software system throughout the development lifecycle: Requirements, the Architect's Experience and the Technical Environment of the development. The Requirements of the ABC model include not only 'system requirements' such as those described in Section 4, but also the requirements of the developing organization(s). In the data-Grid domain, several such influences can be readily identified. Firstly, some projects have very close relationships with teams or organizations that are developing tools or technologies for data-Grids. Such associations can result in implicit or explicit requirements for a project to leverage such tools or technologies in their system. Second, the fact that most data-Grid projects are geographically distributed leads to a requirement for a highly modular architecture.

The previous experience and expertise of the system architect(s) may also affect the architecture of their current project, through an inclination towards a particular architectural choice or approach. Finally, any software development project exists in a technical environment of trends, paradigms, technologies and existing infrastructure that may influence the development process and hence the architecture. These may be specific to certain scientific domains, such as existing networked applications and data services, or non-specific such as the tools and software packages of the Globus Toolkit, the Web services paradigm and the Open Grid Services Architecture (OGSA). As discussed in Section 2, the nature of the distributed computing systems from which data-Grids have evolved must also be considered. The architecture of these systems will inevitably influence that of the data-Grids that emerge from them.

6.6. *Information Quality*

We have indicated throughout that, being drawn from public websites and other such available documents, the information upon which this study is based may be incomplete, or of uncertain currency. We have considered these limitations in the observations made, and conclusions drawn, describing our reasoning at each stage.

The data used is also necessarily qualitative in nature. Qualitative research methodologies have been used extensively within the Information Systems community, and our work may be considered as a Case Study; "an empirical study that investigates a contemporary phenomenon within its real-life context" [16]. Our work studies the requirements and architectures of data-Grid systems within the context of a project-based development environment. We also note that many of the steps involved in the process of style evaluation were dependent upon heuristic judgments. Given these considerations, our results are presented as being indicative rather than definitive. Our conclusions may be used by data-Grid projects to guide their initial stages of development, or to suggest direction and focus for a more in-depth study of data-Grid systems.

7. Conclusions

In consideration of the above discussion, we are able to draw certain conclusions from this investigation. We have demonstrated that data-Grids are an emerging domain with a well-defined set of core functional requirements, though poorly defined non-functional requirements. From this, we have derived a set of general requirements for data-Grid systems. We have identified requirements that are particularly sensitive to system architecture. We have considered the architectural styles prevalent in the distributed systems from which data-Grids have emerged, and analyzed the fitness of these styles for fulfilling the derived general requirements. We have determined that n-tier architectures offer the best fit to these requirements, suggesting a baseline architecture for data-Grid projects. It has also been noted that the peer-to-peer style also offers significant benefits. Our examination of current data-Grid projects has indicated that the n-tier style is being use extensively, though implicitly by the vast majority of projects. Use of other styles is also apparent. Most systems are hybrid in style, with many tiered components decomposing to reveal an internal structure that conforms to an alternate style.

It is generally the case that individual projects have made decisions about technologies and component solutions to be used very early in the development lifecycle, effectively freezing some high-level design decisions. With this equal focus on requirements and architecture, the development process evident in the data-Grid domain roughly follows the Twin Peaks model of system development [12]. This model focuses explicitly on requirements and architecture, allowing for their concurrent and semi-independent evolution, thus addressing specific characteristics of the data-Grid domain: the need for elaboration and improved definition of requirements through early prototyping, the need to match existing units of architecture (technologies and components) to requirements, and the fact that lower priority requirements may be subject to rapid change. Twin Peaks also allows for the evaluation of alternative design solutions offered by existing software packages or components.

From the above conclusions, guidelines have emerged that may serve to inform the very early stages of design and development for data-Grid systems. In a domain where software engineering and requirements analysis expertise are not always available, projects may use a set of general, core requirements and a coarse-grained, *n*-tier model to explore and refine the properties of the system through successive, rapid iteration between the requirements and architecture of the system.

8. Future Work

The EGSO project is currently moving from requirements analysis for architecture choice to component and interface design. At this level, formal techniques are successfully employed to test the design [7]. Concurrent component event transitions support progress analysis, and rapidly developed prototypes facilitate evaluation by the user community. In this way, the high-level hybrid architectural description, mapped to vague requirements, is refined to a workable design that demonstrates how user requirements have been interpreted. This approach further clarifies user understanding about how a data-Grid may satisfy their goals.

Will EGSO (and other projects) really implement their stated architectural style? As data-Grids are implemented and deployed (reality bites) components and relationships may be implemented that are not represented in the initial architecture. It is possible that a new architectural style will emerge, or a novel high-level view will be useful. For EGSO we will be able to trace implementation detail directly to a revised architecture. We will also be able to examine other projects via their user documentation, user experience and reported engineering experience.

References

1. A. Finkelstein, C. Gryce and J. Lewis-Bowen, "Appendix to Relating Requirements and Architectures: A Study of Data-Grids". http://grid.ucl.ac.uk/file/datagrid-appendix.pdf
2. L. Bass, P. Clements and R. Kazrnan, *Software Architecture in Practice*. Addison-Wesley, 1998.
3. C. Baru, R. Moore, A. Rajasekar and M. Wan, "The SDSC Storage Resource Broker", in *Proceedings of CASCON'98*, Canada, 1998.
4. A. Chervenak et. al., "Giggle: A Framework for Constructing Scalable Replica Location Services", in *Proceedings of the IEEE Supercomputing Conference*, 2002.
5. N. Ching and C. Gryce, "Descending the Twin Peaks: Requirements and Architecture in the EGSO Project", in *Proceedings of the UK e-Science All Hands Meeting*, September 2003.
6. A. Dardenne, A. van Lamsweerde and S. Fickas, "Goal-Directed Requirements Acquisition", *Science of Computer Programming*, Vol. 20, April 1993.
7. A. Finkelstein, J. Lewis-Bowen and G. Piccinelli, "Using Event Models in Grid Design", forthcoming in J.C. Cunha and O.F. Rana (eds.), *Grid Computing: Software Environments and Tools*, Springer.
8. I. Foster, C. Kesselman and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", *The International Journal of Supercomputer Applications*, 2001.
9. I. Foster and A. Iamnitchi, "On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing", in *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.
10. E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns: Elements of Reusable Object Oriented Software*. Addison-Wesley, 1995.
11. T.G. Lane, T. Asada, R. Swonger, N. Bounds and P. Duerig, "Architectural Design Guidance", in [13], Chapter 5.
12. B. Nuseibeh, "Weaving the Software Development Process between Requirements and Architectures", in *Proceedings of the ICSE 2001 STRAW Workshop*, Toronto, 1996.
13. M. Shaw and D. Garlan, *Software Engineering – Perspectives on an Emerging Discipline*. Prentice Hall, 1996.
14. V. Sunderam and Z. Nemeth, "A Formal Framework for Defining Grid Systems", in *Proceedings of the Second IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2002.
15. S. Tuecke et al., "Grid Service Specification", February 2002. http://www.globus.org/research/papers/gsspec.pdf
16. R.K. Yin, *Case Study Research, Design and Methods*, 3rd edn. Sage: Newbury Park, 2002.
17. AstroGrid. http://www.astrogrid.org/
18. BIRN. "Biomedical Informatics Research Network". http://birn.ncrr.nih.gov/birn/
19. "Condor Project". http://www.cs.wisc.edu/condor/
20. EDG. "European DataGrid Project". http://eu-datagrid.web.cern.ch/eu-datagrid/
21. EGSO. "European Grid of Solar Observations". http://www.egso.org/
22. ESG. "Earth System Grid". http://www.earthsystemgrid.org/
23. "Globus Project". http://www.globus.org/
24. GriPhyN. "Grid Physics Network". http://www.griphyn.org/
25. myGrid. http://mygrid.man.ac.uk/
26. NVO. "US National Virtual Observatory". http://www.us-vo.org/
27. PPDG. "Particle Physics Data Grid". http://www.ppdg.net/
28. "Spitfire EDG Task." http://edg-wp2.web.cern.ch/edg-wp2/spitfire/
29. VSO. "Virtual Solar Observatory". http://vso.nso.edu/