

The Grid Lite DREAM: Bringing the Grid to Your Pocket

Chris A. Mattmann^{1,2}, Nenad Medvidovic¹

¹ University of Southern California
Los Angeles, CA 90089, USA,
{mattmann, neno}@usc.edu

² Jet Propulsion Laboratory, California Institute of Technology
Pasadena, CA 91109, USA
chris.mattmann@jpl.nasa.gov

Abstract. The emergence of small, mobile, inexpensive computing platforms has made computation possible virtually anywhere, and has opened up countless opportunities for distributed and decentralized collaboration and information sharing among a wide range of actors. The software-intensive systems of today are increasingly shaped by their decentralized, resource-constrained, embedded, autonomic, and mobile (*DREAM*) computing environments. In this paper we present GridLite, a software architecture-based grid platform suitable for deployment in *DREAM* environments. Our prototype implementation of GridLite represents an effective and highly efficient marriage of our OODT data grid and Prism-MW architectural middleware solutions. The ultimate goal of GridLite is to extend the reach of the grid all the way to people's "pockets". Our initial experience suggests that this goal is achievable and worthy of further active pursuit.

Keywords: GLIDE, GridLite, *DREAM*, Software Architecture, Grid Computing.

1 Introduction

The emergence of small, mobile, embedded, inexpensive computing platforms (e.g., PDAs, cell phones, GPS receivers) has made computation possible virtually anywhere. In turn, this has opened up countless possibilities for distributed and decentralized collaboration and information sharing among a wide range of individuals and organizations, including engineers, scientists, health and humanitarian workers, emergency response teams, law enforcement agencies, and average citizens. Fleets of mobile devices are, or will soon be, employed in complex scenarios such as land and sea exploration, environment monitoring, traffic management, fire fighting, and damage surveys in times of natural disaster. The software-intensive systems of today are increasingly shaped by their *decentralized*, *resource-constrained*, *embedded*, *autonomic*, and *mobile* (*DREAM*) computing environments.

In parallel with this development, another exciting and promising direction in modern computing has emerged – *the grid*. Grid computing connects dynamic collections of individuals, institutions, and resources to create virtual organizations, which support sharing, discovery, transformation, and distribution of data and computational resources. Distributed workflow, massive parallel computation, and knowledge discovery are only some of the applications of the grid. Grid applications involve large numbers of distributed devices executing large numbers of computational and data components. As such, they require techniques and tools for supporting their design, implementation, and dynamic evolution.

The grid has revolutionized the manner in which both computation-intensive and data-intensive software systems are constructed and deployed. The grid paradigm, however, makes a number of limiting assumptions that curtail its adoption, utility, and deployment in the emerging DREAM computing environments. These assumptions include availability of powerful processors, large amounts of memory, capacious and reliable network links, and stability of software systems deployed on the grid. Two independently conducted studies [2, 3] to date have indicated that existing grid technologies suffer from several recurring shortcomings, which are particularly magnified in the context of the emerging DREAM environments. We will highlight several of the shortcomings for illustration. First, existing grid solutions are implemented using technologies (e.g., CORBA, Web services) that are unsuitable for DREAM environments. Second, grid protocols (e.g., Grid Resource Allocation Management, GridFTP, Meta Directory Service) require heavy-weight processing and memory resources to poll and monitor nodes in a grid-based system. Third, the grid solutions assume stable network connectivity and bandwidth. Fourth, the topology of a deployed grid-based system is essentially static, and any modification to an existing deployment may require a (manual) restart of the entire system. Finally, the existing grid technologies provide no system design, implementation, deployment, and evolution guidance to their users; instead, they implicitly assume that the users will somehow “figure it out”. This is particularly problematic when one considers that the systems deployed on the grid may be highly complex, and that the typical users of the grid (e.g., scientists, health workers) may have no formal training in software development.

While most of the above difficulties may be overcome by engineering more efficient underlying infrastructure, the lack of development guidance for grid-based software systems also requires enriching grid computing with an appropriate body of software development concepts, constructs, principles, and techniques. We believe that the area of *software architecture* [6] provides such a body of knowledge. Software architectures are high-level abstractions for modeling the structure, behavior, and key properties of software systems. These abstractions involve descriptions of elements from which systems are built, interactions among the elements, patterns that guide their composition, and constraints on those patterns. In general, a system is defined as a set of *components* (elements that encapsulate computations and state in a system), *connectors* (elements that embody interactions), and a *configuration* (overall organization of components and connectors). Furthermore, software architectural *styles* are key design idioms that embody best practices in the design of systems in specific domains (e.g., DREAM).

Software architecture plays an important role in our proposed research agenda, as we illustrate throughout the rest of this paper. We will argue for an architecture-based approach to use grid computing as a means of constructing software systems in the DREAM environments. Our position is that, in order to address the aforementioned limitations of the grid paradigm, a new type of software solution must be constructed. To that end we propose, and have implemented an early prototype of, *GridLite*, a software architecture-based grid platform suitable for deployment in DREAM environments [1]. GridLite is heavily influenced by, and its early design and implementation directly rely on, our OODT data grid [7] and Prism-MW architectural middleware [4] platforms. The ultimate goal of GridLite is to extend the reach of the grid all the way to people's "pockets".

The remainder of the paper is organized as follows. Section 2 provides the background and related work in the areas of grid computing, software architecture, and the relationship of each to DREAM environments. Section 3 lays out the architecture of GridLite, including its objectives and architectural principles. Section 4 describes and evaluates the current status of the GridLite research and the prototype implementation and infrastructure of GridLite that we have constructed called GLIDE. Section 5 rounds out the paper with our conclusions and an overview of future work.

2. Background and Related Work

Our work on GridLite has been inspired by a set of related projects and draws upon three fundamental areas of research: *computational and data grid computing*, *light-weight middleware and protocols*, and *implementation support for software architectures*. We discuss GridLite and its relationship to each area below. We then describe representative approaches to large-scale data sharing, with a specific focus on OODT, the grid technology in whose development we have participated and which is used by NASA and the National Cancer Institute. Finally, we summarize Prism-MW, our light-weight middleware platform that explicitly focuses on implementation-level support for software architectures in DREAM environments; we also overview a cross-section of representative light-weight middleware platforms.

2.1 Computational Grid Technologies

The *Globus Toolkit* [8, 9] is an open-source, research-off-the-shelf middleware framework for constructing and deploying grid-based software systems. It combines a middleware transport layer (reified in the form of the Simple Object Access Protocol (SOAP) [20]), a suite of Grid-services and protocols (e.g. Grid Resource Allocation Management or GRAM [9], GridFTP [9], and so on) and a web services-based implementation infrastructure for constructing and deploying grid-based software systems using various programming languages, including Java, C++, and Perl.

The adoption of Globus has to date primarily been at the level of scientific and research institutions, although commercial adoption of Globus is currently occurring at IBM, Sun, and Microsoft [21]. In the large, Globus realizes the basic goal of grid

systems: the establishment of *virtual organizations* (VOs) sharing computing, data, metadata, and security resources. In the small, however, Globus lacks many of the salient features that would ease its adoption and use across a more widespread family of software systems and environments. These salient features include (1) the marriage of architecture-based software development (which has been shown to facilitate and improve large-scale, distributed software construction), with the great promise and potential provided by the grid and (2) the decoupling of Globus protocols and services, such as GridFTP and GRAM, from their heavyweight origins, File Transfer Protocol (FTP) and Lightweight Directory Access Protocol (LDAP) respectively.

In addition to Globus, several other grid technologies have emerged recently. Alchemi [10] is based on the Microsoft .NET platform and allows developers to aggregate the processing power of many computers into virtual computers. Alchemi is designed for deployment on personal computers: computation cycles are only shared when the computer is idle. JXTA [11] is a framework for developing distributed applications based on a peer-to-peer topology. Its layered architecture provides abstractions of low-level protocols along with services such as host discovery, data sharing, and security.

2.2 Data Grid Technologies

GridLite is directly motivated by our own work in the area of data-grids, specifically on the Object Oriented Data Technology (OODT) system [7]. We have adopted an architecture-centric approach in OODT, in pursuit of supporting distribution, processing, query, discovery, and integration of heterogeneous data located in distributed data sources. Additionally, OODT provides methods for resource description and discovery [12] based on the ISO-11179 data model standard [13], along with the Dublin Core standard for the specification and standardization of data elements [14].

There are several other technologies for large-scale data sharing. Grid Data Farm [15] project is a parallel file system created for researchers in the field of high energy acceleration. Its goal is to federate extremely large numbers of file systems on local PCs and, at the same time, to manage the file replication across those systems, thus creating a single global file system. Similar to OODT, the SDSC Storage Resource Broker [16] is a middleware that provides access to large numbers of heterogeneous data sources. Its query services attempt to retrieve files based on logical information rather than file name or location, in much the same way that OODT maintains profile data.

2.3 Prism-MW

Prism-MW [4] is a middleware platform that provides explicit implementation-level support for software architectures. The key software architectural constructs are *components* (units of computation within a software system), *connectors* (interaction facilities between components such as local or remote method calls, shared variables,

message multicast, and so on), and *configurations* (rules governing the arrangements of components and connectors) [6, 17]. Prism-MW's core object model consists of several canonical classes allowing for the expression of complex architectures using a well-defined set of architectural primitives. *Brick* is an abstract class that encapsulates common features of its subclasses (*Architecture*, *Component*, and *Connector*). The *Architecture* class records the configuration of its components and connectors, and provides facilities for their addition, removal, and reconnection, possibly at system runtime. A distributed application is implemented as a set of interacting *Architecture* objects, communicating via *DistributionConnectors* across process or machine boundaries. *Components* in an architecture communicate by exchanging *Events*, which are routed by *Connectors*. Finally, Prism-MW associates the *IScaffold* interface with every *Brick*. Scaffolds are used to schedule and dispatch events using a pool of threads in a decoupled manner. *IScaffold* also directly aids architectural self-awareness by allowing the runtime probing of a *Brick's* behavior.

Prism-MW enables several desired features of software development in DREAM domains. First, it provides the needed low-level middleware services, including decentralization, concurrency, distribution, programming language abstraction, and data marshalling and unmarshalling. Second, unlike the support in current grid-based middleware systems (including OODT), Prism-MW enables the definition and (re)use of architectural styles, thereby providing design guidelines and facilitating reuse of designs across families of DREAM systems. Third, Prism-DE [18], an architecture-based (re-)deployment environment that utilizes Prism-MW, can be extended to aid users in constructing, deploying, and evolving grid-based DREAM systems.

A number of additional middleware technologies exist that support either architectural design or mobile and resource constrained computation, but rarely both [18]. An example of the former is Enterprise Java Beans, a popular commercial technology for creating distributed Java applications. An example of the latter is XMIDDLE [19], an XML-based data sharing framework targeted at mobile environments.

3. The GridLite Architecture

In order to address the above-stated shortcomings of existing grid technologies and bring the grid "to your pocket", a new type of grid must be developed, which we dub "GridLite". GridLite is an adaptable, light-weight grid platform that is equipped with the appropriate support for application design, implementation, deployment, and evolution. To realize GridLite, we have drawn upon our experience in the areas of distributed computing [4], software architecture and in particular its role in DREAM environments [5], and distributed data management [1]. In this section, we describe the architecture of GridLite, including the three key objectives that motivated its separation into three distinct layers. GridLite is a layered solution comprising facilities for software architecture-based application development that leverage a set of core grid services, both of which are implemented on top of a light-weight middleware platform, as depicted in Figure 1. In the subsequent sections, we will

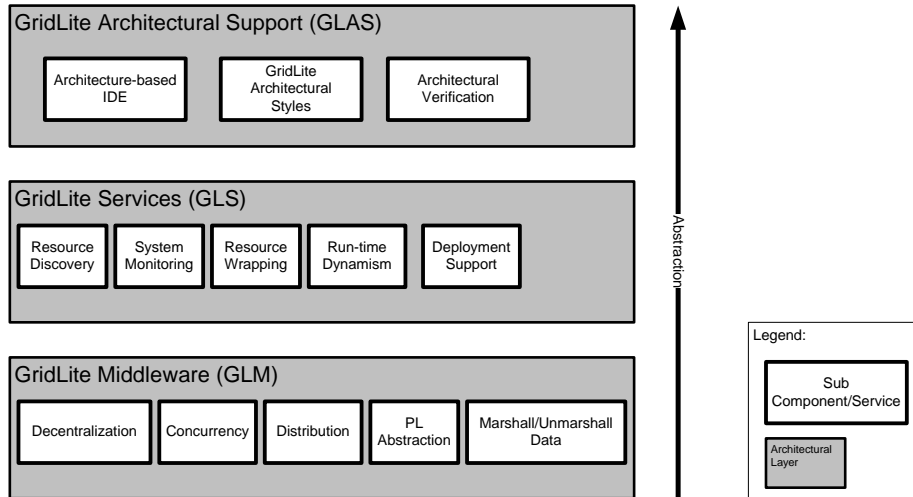


Figure 1. The Grid Lite Architecture

explain each of GridLite’s layers (and each layer’s relationship to the objectives) in detail.

3.1 Middleware Layer (GLM)

GridLite’s architectural objectives were directly inspired by the need to address the aforementioned shortcomings of the grid. To attack the first shortcoming (the current grid technologies heavy-weight nature), we arrive at the first objective of the GridLite architecture:

Objective 1. (Light-weight) Middleware Support – Create an efficient and adaptable middleware platform to support implementation, deployment, and runtime monitoring and evolution of GridLite systems.

The middleware support in GridLite requires several features including:

1. *Decentralization* – since fleets of mobile devices are highly dynamic and distributed, and since network links are inherently unreliable in DREAM environments, the GridLite architecture should include middleware that directly aids a system’s loose-coupling and decentralization.
2. *Concurrency* – grid systems typically involve massive amounts of parallel computation and data distribution. As such, any middleware support for the GridLite architecture needs the ability to support concurrency that takes advantage of the large amounts of hosts and services available.
3. *Distribution* – DREAM environments are characterized by highly mobile, dynamic, and distributed applications, unlike the existing grid environments which are comparatively stable and stationary. The GridLite architecture

needs to have the ability to support distribution at the architectural level, including distributed deployment.

4. *Programming Language Abstraction* – Both DREAM and grid environments typically involve software systems communicating via protocols across programming language (PL) boundaries (e.g., Java and C++ communicating via SOAP). Thus, GridLite should natively abstract away the underlying PL.
5. *(Un-)Marshalling of Data* – Large amounts of scientific data are typically exchanged in grid-based software systems. Consequently, the large amount of data transferred by GridLite applications operating in DREAM environments needs first-class services. Such services include packaging and un-packaging data using standard formats (e.g., XML).

To summarize, the middleware layer (GLM) builds on our prior work in the area of light-weight middleware [4], and encapsulates low-level communication on devices whose connectivity is limited and unstable. GLM is intended to abstract different communication protocols (e.g., Internet, Bluetooth) to enable transparent connections; different operating systems (e.g., PalmOS, WindowsCE, Symbian) to ensure platform portability; and different programming languages (e.g., J2ME, EVC++, Brew) to foster development flexibility and interoperability. GLM should also leverage light-weight software components (e.g., “tiny” XML parsers) to provide application extensibility, resource discovery, and data conversion. Ultimately the GLM layer should provide compatibility with existing grids, extending the reach of grid computing to pocket-sized devices. Design, implementation, and evaluation of GLM have been the early focus of our work on GridLite.

3.2 Services Layer (GLS)

We needed a means of empowering GridLite systems with native, light-weight grid services for use in DREAM environments. From this need was born our second major objective for the GridLite architecture:

Objective 2. Grid Service Development – Support development of an extensible set of grid services (e.g. resource discovery, system monitoring, and so forth).

GridLite requires several key grid services that are provided by the GLS layer. We briefly summarize each of them below:

1. *Resource Discovery* – this service enables GridLite to discover resources in its given deployment environment. A *resource* is defined as a unit of computation (e.g., an object or component), data, metadata, data-producing software, or is a computation-providing system. Resource discovery should support peer-to-peer (see [22]) or client-server based resource discovery.
2. *System Monitoring* – this service will provide the capability of monitoring data and metadata within a GridLite system. Such data has the ability to shape the way in which a system is configured. We have recently shown [4] that monitoring data can affect a mobile software system’s dependability given a particular deployment of its components. The system monitoring

service uses both the data and metadata it collects to affect a GridLite software system's deployment providing maximum quality of service.

3. *Resource Wrapping* – this service allows structured retrieval of data from semi-structured resources (e.g., resources that do not necessarily adhere to a single data model and software interface) via the well-known technique of *software wrapping* [24]. In the information integration community this service is offered at the level of returning XML-formatted data from heterogeneous web sites [23].
4. *Run-time Dynamism* – this service is responsible for accepting requests and understanding how to dynamically change a running GridLite software application given its modeled architecture. The service provides an interface for the addition, removal, replacement, and redeployment of running application-level components exported by the GLAS layer above it. Its task includes ensuring component “quiescence” before disconnection, logging requests intended for components that are temporarily unavailable, updating a reinserted component's state before allowing it to engage in interactions with other components in the system, rolling back (i.e., “undoing”) changes that cannot be completed because of new system events, and ensuring system integrity throughout this process.
5. *Deployment Support* – this service is responsible for physically distributing both GridLite components (e.g., different GLS components, including the deployment service itself) as well as application components to GridLite-enabled hosts. This service is a consumer of deployment commands from GridLite's GLAS architectural layer.

In summary, the services layer (GLS) encapsulates the lower-level interfaces of GLM into a set of basic GridLite services. The services include discovering resources on the grid (such as data or computation provision), monitoring grid and grid node performance, wrapping structured resources to enable access in a heterogeneous setting, dynamically “morphing” deployed system architectures, and deploying uniform grid software components in the face of different execution platforms. Experience in resource and metadata description [1] and software architectures [4] is necessary to enable GLS to utilize GLM effectively and efficiently.

3.3 Software Architectural Support Layer (GLAS)

The final, and what we would argue to be the most important, limitation of existing grid systems is the lack of any (native) support for architectural abstractions and their relationship to a grid-based software system's implementation. This issue has been identified by two independently conducted studies [2, 3], and has directly motivated our third objective with the GridLite architecture:

Objective 3. Software Architectural Support – Formulate a set of software architectural principles for constructing GridLite-based applications.

The software architectural support layer (shown in detail in Figure 2) of GridLite is reified in the form of three critical services:

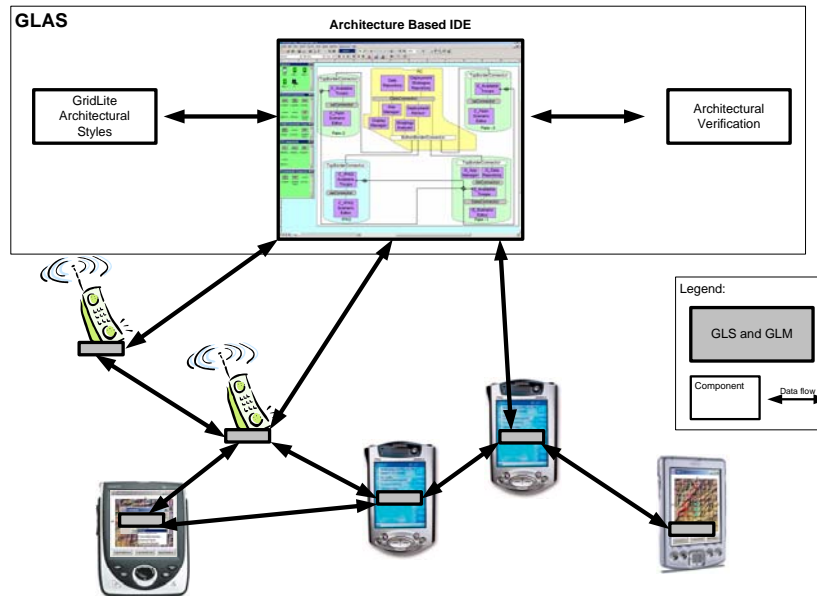


Figure 2. The GridLite Architectural Support Layer

1. *Architecture-based Integrated Development Environment (IDE)* – the IDE provides a visual interface for system design of GridLite software applications using the systematic primitive services detailed in the preceding sections. The IDE also provides facilities for (partially) automated system implementation from an architectural description.
2. *Architectural Styles* – in order to provide support to application developers for effective architectural compositions, GLAS must be able to model, ensure adherence to, and implement appropriate architectural styles for DREAM environments. An architectural style is a set of design heuristics and guidelines for composing and constructing a family of software systems [6, 17]. Our discussion thus far may have implied that peer-to-peer (P2P) is the ideal style for DREAM systems. However, P2P does not enforce topological constraints on applications and may result in “component soup” architectures. We postulate that there may be (a number of) other architectural styles that are well suited to DREAM environments, and it is the primary responsibility of the architectural styles service to provide support for designing GridLite software systems with a multitude of different architectural styles.
3. *Architectural Verification* – analysis of architectural models allows engineers to discover or verify critical system properties early in the development life cycle. GridLite leverages an *architecture-description language* (ADL) [17] as the basis for architectural analysis of GridLite systems. Analysis should be conducted both in the traditional manner, at

architecture specification-time [17], and by leveraging (meta-level) architectural model events, at system run-time.

In summary, the top-most (GLAS) layer of GridLite provides software architectural support. We believe that a software architecture-based approach is essential in deploying successful applications onto hundreds or thousands of DREAM nodes. GLAS should enable the expression of application architectures using a number of systematic primitives (e.g., component, connector, port, communication event, data stream) that are directly implemented in the GLM. From these primitives, complex architectures could be shaped and then dynamically reshaped based on environment events, changing requirements, dynamically discovered resources, and so on.

4. Current Status

4.1 GLIDE – A Reference Implementation of GridLite

Our reference implementation of the GridLite architecture is called “GLIDE”, which stands for a grid-based, *lightweight, infrastructure for data-intensive environments*. GLIDE is a hybrid grid middleware which combines the key architectural facilities of Prism-MW and core data grid services provided by OODT. The implementation-level class diagram of GLIDE is shown in Figure 3. At first blush, it appears that this design does not correspond directly to the GridLite architecture from Figure 1. However, that is only a by-product of our intent to reuse the facilities provided by Prism-MW and OODT to the greatest intent possible. In fact, the OODT components, placed along the periphery of the diagram, provide the GLS-level grid services (recall Section 3.2), while the Prism-MWCore section provides both the GLM-level infrastructure support and, due to Prism-MW’s native support for architectural constructs, part of the GLAS-level architectural support (other aspects of the GLAS layer within GLIDE are discussed in Section 4.3).

We specialized Prism-MW to implement the core components of GLIDE shown in Figure 3. Our intent was to retain the key properties and services of Prism-MW and provide basic grid services (recall the GLS layer described in Section 3.2) across dynamic and mobile virtual organizations. Additionally, we desired GLIDE to support architecture-based design, implementation, deployment, and evolution of data-intensive grid applications in DREAM environments. Finally, we desired that GLIDE at least partially interoperate with a heavy-weight grid counterpart: because of our prior experience with the OODT data grid platform, it seemed the most appropriate choice; indeed, OODT directly influenced our design of the key grid services provided by GLIDE. Below we describe GLIDE’s realization in light of these goals and the description of the GridLite architecture from Section 3.

Inspired by OODT’s architecture, GLIDE’s *Data Components* include the *Resource Profile*, a data structure which describes the location and classification of a resource available within a grid-based software system. Resources include data granules (such as a File), data-producing software systems (including the below described profile servers, product servers, query servers, and so on), computation-

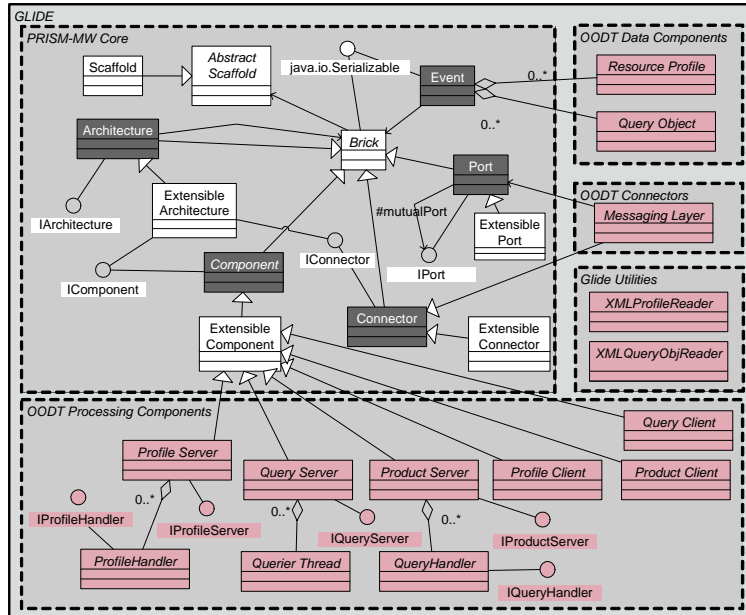


Figure 3. GLIDE's design.

providing software systems, and resource profiles themselves. Resource profiles may contain additional resource-describing metadata [14]. Resource profiles directly address the GridLite *grid service development objective*, including *resource discovery* (recall Section 3.2). The *Query Object* is a data structure which contains a query expression. A query expression assigns values to a predefined set of data elements that describe resources of interest to the user and a collection of obtained results.

Again, inspired by OODT's architecture, GLIDE's *Processing Components* include *Product Servers*, which are responsible for abstracting heterogeneous software interfaces to data sources (such as an SQL interface to a database, a File System interface to a set of images, an HTTP interface to a set of web pages, and so on) into a single interface that supports querying for retrieval of data and computational resources. Users query product servers using the query object data structure. *Product Clients* connect and send queries (via a query object) to product servers. Product clients and servers are implementation-level artefacts of *grid service deployment*, including *resource wrapping*. A query results in either data retrieval or use of a remote computational resource. *Profile Servers* generate and deliver metadata [2] in the form of resource profile data structures, which are used for making informed decisions regarding the type and location of resources that satisfy given criteria. *Profile Clients* connect and send queries to profile servers. After sending a query, a profile client waits for the profile server to send back any resource profiles that satisfy the query. *Query Servers* accept query objects, and then use profile servers to determine the available data or computational resources that satisfy the user's query. Once all the resources have been collected, and processing has occurred, the data and processing results are returned (in the form of the result list of a query

object) to the originating user. *Query Clients* connect to query servers, issue queries, and retrieve query objects with populated data results.

GLIDE contains one key *software connector* type. The *Messaging Layer* connector is a data bus which marshals resource profiles and query objects between GLIDE client and server components.

Each GLIDE processing component was implemented by subclassing Prism-MW's *ExtensibleComponent* class, using the asynchronous mode of operation. Asynchronous interaction directly resulted in lower coupling among GLIDE's processing components. For example, the dependency relationships between GLIDE's *Client* and *Server* processing components, which existed in OODT, was thereby removed. GLIDE's components use Prism-MW's *Events* to exchange messages. GLIDE data components are sent between processing components by encapsulating them as parameters in Prism-MW *Events*, directly addressing the GridLite (*light-weight*) *middleware support objective*, including (*un-marshalling of data*). Leveraging Prism-MW's *Events* to send and receive different types of data enables homogenous interaction among the possibly heterogeneous processing components.

We found OODT's connectors not to be suitable for DREAM environments because of their heavy-weight (they are implemented using middleware such as RMI and CORBA). Furthermore, they only support synchronous interaction, which is difficult to effect in highly decentralized and mobile systems characterized by unreliable network links. To this end, we have leveraged Prism-MW's asynchronous connectors to implement the messaging layer class in GLIDE. GLIDE's connector leverages Prism-MW's port objects that allow easy addition or removal of TCP/IP connections. This allows the system's topology to be adapted at runtime, directly addressing objectives 1 and 2 of the GridLite architecture (*light-weight*) *middleware services* and *run-time dynamism*, respectively). GLIDE's connector also implements event filtering such that only the requesting client receives responses from the server.

Lack of ability to easily adapt a system's software architecture is a key limitation of current grid systems, including OODT. If present, such a facility could be leveraged to improve a system's functionality, scalability, availability, latency, and so on. For example, our recent studies [4] have shown that the availability and latency of software systems in DREAM environments can be improved significantly via dynamic adaptation.

Finally, to support interoperability of GLIDE with OODT, we provide two additional utility classes: *XMLProfileReader* and *XMLQueryObjReader* parse an XML representation of a resource profile and query object data structure, respectively. For brevity, we do not provide a detailed example of the XML Profile structure here, but its full treatment can be found in [12]. Each string is parsed into a GLIDE data object. Similarly, resource profiles and query objects can be serialized into XML. Thus, the level of interoperability with OODT is at the level of resource description and retrieval, and currently resource profiles and data can be exchanged between GLIDE and OODT. As part of our current work, we are investigating the Web Services Resource Framework (WS-RF) as a means of enabling interoperability between GLIDE and Globus, which uses WS-RF in its latest release (GTK 4.0).

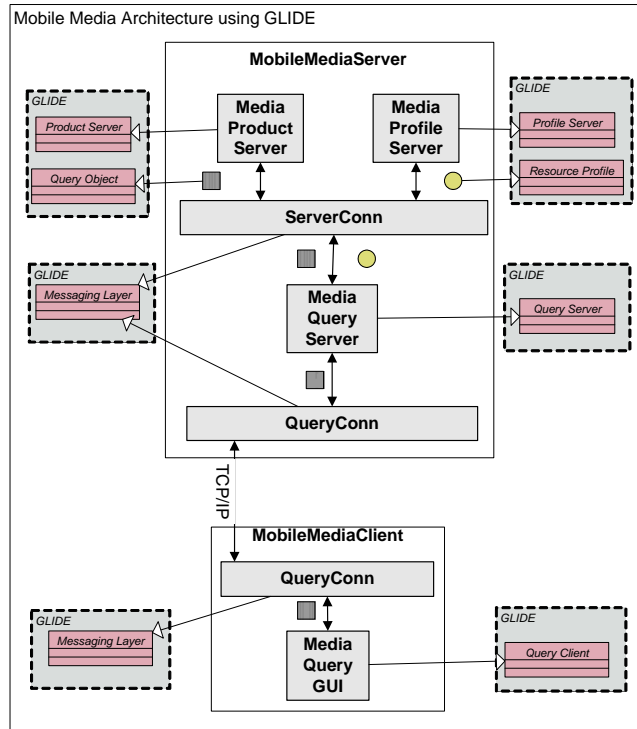


Figure 4. Mobile Media Application Architecture

4.2 Sample Application Using GLIDE

In order to evaluate the feasibility of GLIDE, we designed and implemented several representative applications. Here we discuss a prototype *Mobile Media Sharing* application (MMS), shown in Figure 4. MMS allows a user to query, search, locate, and retrieve MP3 resources across a set of mobile, distributed, resource-constrained devices. Users query mobile media servers for MP3 files by specifying values for genre and quality of the MP3 (described below), and if found, the MP3s are streamed asynchronously to the requesting mobile media client.

Figure 4 shows the overall distributed architecture of the MMS application. A mobile device can act as a server, a client, or both. *MobileMediaServer* and *MobileMediaClient* correspond to the parts of the application that are running on the server and the client devices.

MobileMediaClient contains a single component called *MediaQueryGUI*, which provides a GUI for creating MP3 queries. MP3 queries use two query parameters, *MP3.Genre* (e.g., rock) and *MP3.Quality* (e.g., 192 kb/s, 128 kb/s). *MediaQueryGUI* is attached to a *QueryConn*, which is an instance of GLIDE's messaging layer

connector that forwards the queries to remote servers and responses back to the clients.

MobileMediaServer is composed of three component types: *MediaQueryServer*, *MediaProductServer*, and *MediaProfileServer*. *MediaQueryServer* parses the query received from the client, retrieves the resource profiles that match the query from *MediaProfileServer*, retrieves the mp3 file(s) in which the user was interested from the *MediaProductServer*, and sends the MP3 file(s) back to the client.

The MMS application helps to illustrate different aspects of GLIDE: it has been designed and implemented by leveraging most of GLIDE's processing and data components and its messaging layer connector, and has been deployed on DREAM devices. In the next section we evaluate GLIDE using MMS as an example.

4.3 Architecture-Based Development and Deployment Support in GLIDE

GLIDE inherits architecture-based development and deployment capabilities, including style awareness, from Prism-MW and deployment support, from PRISM-DE. Unlike most existing grid middleware solutions (e.g. OODT), which provide support for either peer-to-peer or client-server styles, GLIDE does not impose any particular (possibly ill-suited) architectural style on the developers of a grid-based application. As a proof of concept, we have implemented several variations of the MMS application in different architectural styles including client-server, layered client-server, peer-to-peer, and C2 [25]. The variations of MMS leveraged existing support for these styles and were created with minimal effort. For example, changing MMS from client-server to peer-to-peer required addition of three components and a connector on the server side, and one component and one connector on the client side. Figure 5 shows the peer-to-peer variant of MMS.

4.4 DREAM Support

Resource scarcity poses the greatest challenge to any grid solution for DREAM environments. We have leveraged Prism-MW's efficient implementation of architectural constructs [17] along with the following techniques to improve GLIDE's performance and minimize the effect of the computing environment's heterogeneity: (1) MinML [25], a lightweight XML parser, to parse the resource profiles and query object data structures; (2) W3C's Jigsaw Web Server Base64 Encoding Library [27] to compress (at the product server end) and uncompress (at the product client end) the exchanged data; (3) Filtering inside the Messaging Layer to ensure event delivery only to the interested parties, thus minimizing propagation of events with large data loads (e.g., MP3 files). Specifically, GLIDE tags outgoing request events from a client with a unique ID, which is later used to route the replies appropriately; and (4) Incremental data exchange via numbered data segments for cases when the reliability of connectivity and network bandwidth prevent atomic exchange of large amounts of data.

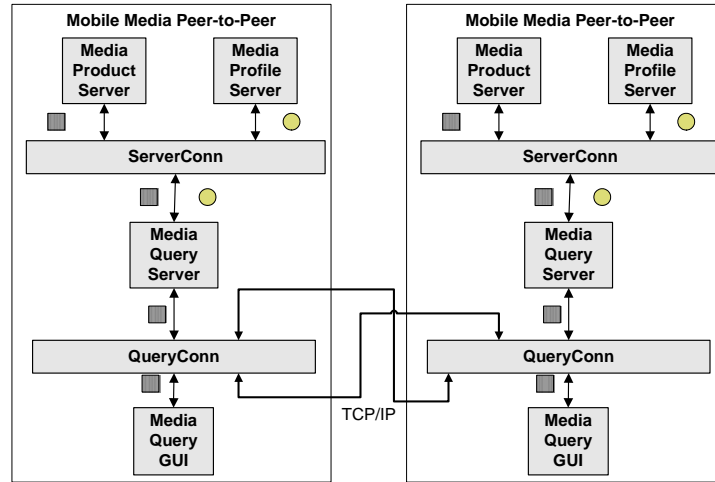


Figure 5. Peer-to-Peer variation of the Mobile Media Application

As an illustration of GLIDE's efficiency, Table 1 shows the memory footprint of *MobileMediaServer*'s and *MobileMediaClient*'s implementation in GLIDE. The total size of the *MobileMediaServer* was 5.7KB and *MobileMediaClient* was 4.1KB, which is two orders of magnitude smaller than their implementation in OODT (707KB and 280KB, respectively). The memory overhead introduced by GLIDE on the client and server devices was under 4KB.

5. Conclusion

The GridLite architecture that we describe in this paper has the potential to shape the role and utility of existing mobile computing platforms such as PDAs, cell phones, and laptops used in everyday life. As networked computing systems must operate in DREAM environments, several avenues of research must be significantly advanced. In this paper, we have argued that grid computing provides a promising approach for engineering some classes of networked computing systems, but several limitations of the grid must be addressed before its widespread deployment and use as a computing platform for future networked environments.

In addition to proposing the architecture for GridLite, we described a reference implementation of GridLite, called GLIDE that addresses many of the described limitations of the grid using the proposed objectives of the GridLite architecture as a guide. We summarized the current status of GLIDE, and identified further avenues of research that must be explored in order to ultimately "bring the grid to one's pocket". These future avenues include assessing GLIDE's suitability, as well as possible shortcomings, as *the* reference implementation for GridLite; studying the limits of the envisaged GridLite architecture's adaptability, scalability, and efficiency; and, finally,

Table 1. Memory footprint of MobileMedia Server and Client in GLIDE

<i>Mobile MediaServer</i>	Java Packages	# Live Objects	Total Size (bytes)
Java	java.lang	36	2016
GLIDE's implementation of OODT's components	glide.profile	1	24
	glide.product	1	24
	glide.query	1	32
	glide.queryparser	1	160
	glide.structs	8	232
Application	mobilemedia.product.handlers	1	32
	mobilemedia.profile.handlers	1	8
PRISM-MW	glide.prism.core	26	1744
	glide.prism.extensions.port	1	40
	glide.prism.extensions.port.distribution	4	216
	glide.prism.handler	2	32
	glide.prism.util	18	1200
Total Size			5760
<i>Mobile MediaClient</i>			
Java	java.lang	28	1568
GLIDE's implementation of OODT's components	glide.structs	7	208
Application	mobilemedia	2	384
PRISM-MW	glide.prism.core	18	1304
	glide.prism.extensions.port	1	40
	glide.prism.extensions.port.distribution	3	136
	glide.prism.handler	1	16
	glide.prism.util	7	480
Total Size			4136

expanding this work to other domains, such as computational grids, high-performance computing, and ubiquitous and embedded systems.

Acknowledgments. This material is based upon work supported by the National Science Foundation under Grant Numbers CCR-9985441 and ITR-0312780. Effort also supported by the Jet Propulsion Laboratory, managed by the California Institute of Technology. The authors are especially thankful to Daniel Crichton and Steve Hughes for their contributions to the OODT grid platform, as well as to Sam Malek, Marija Mikic-Rakic, and Chiyong Seo for their work on the Prism-MW middleware platform.

References

1. C. A. Mattmann, S. Malek, N. Beckman, M. Mikic-Rakic, N. Medvidovic, and D. J. Crichton. GLIDE: A Grid-based Light-weight Infrastructure for Data-intensive

- Environments. *Proceedings of the 2005 European Grid Conference (EGC 2005)*, Amsterdam, the Netherlands, February 2005.
2. A. Finkelstein, C. Gryce and J. Lewis-Bowen, Relating Requirements and Architectures: A Study of Data-Grids. *Journal of Grid Computing*, vol. 2, pp. 207-222, September 2004.
 3. C. A. Mattmann, N. Medvidovic, P. M. Ramirez, and V. Jakobac. Unlocking the Grid. *Proceedings of the 8th International Symposium on Component Based Software Engineering (CBSE-8)*, St. Louis, MO, May 2005.
 4. S. Malek, M. Mikic-Rakic, and N. Medvidovic. A Style-Aware Architectural Middleware for Resource-Constrained, Distributed Systems. *IEEE Transactions on Software Engineering*, vol. 31, no. 3, pp. 256-272, March 2005.
 5. N. Medvidovic, M. Mikic-Rakic, N. Mehta, and S. Malek. Software Architectural Support for Handheld Computing. *IEEE Computer, Special Issue on Handheld Computing*, vol. 36, no. 9, pp. 66-73, September 2003.
 6. M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice-Hall, 1996.
 7. C. A. Mattmann, D. Crichton, N. Medvidovic, and S. Hughes. A Software Architecture-based Framework for Highly Distributed and Data-intensive Scientific Applications. *Proceedings of 28th International Conference on Software Engineering (ICSE)*, Shanghai, China, May 2006.
 8. I. Foster et al. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Globus Research, Work-in-Progress 2002.
 9. C. Kesselman, I. Foster, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations, *International Journal of Supercomputing Applications*, pp. 1-25, 2001.
 10. Alchemi .NET Grid Computing Framework. http://www.alchemi.net/doc/0_6_1/index.html
 11. N. Maibaum, T. Mundt. JXTA: A Technology Facilitating Mobile Peer-to-Peer Networks. *Proceedings of MobiWac 2002*. Fort Worth, TX, October 2002.
 12. J. S. Hughes, D. Crichton, S. Kelly, C. A. Mattmann, J. Crichton, and T. Tran. Intelligent Resource Discovery using Ontology-based Resource Profiles. *Data Science Journal*, vol. 4, pp. 171-188, December 2005.
 13. ISO/IEC, Framework for the Specification and Standardization of Data Elements, Geneva, 1999.
 14. DCMI, Dublin Core Metadata Element Set, Version 1.1: Reference Description, 1999
 15. O Tatebe et. al. The Second Trans-Pacific Grid Datafarm Testbed and Experiments for SC2003. *2004 International Symposium on Applications and the Internet*, January 2004, Tokyo, Japan.
 16. A. Rajasekar, M. Wan, R. Moore. MySRB and SRB - Components of a Data Grid. *Proceedings of High Performance Distributed Computing (HPDC-11)*. Edinburgh, UK, July 2002.
 17. N. Medvidovic and R. N. Taylor. A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Transactions on Software Engineering*, vol. 26, no. 1, pp. 70-93, 2000.
 18. M. Mikic-Rakic and N. Medvidovic. Adaptable Architectural Middleware for Programming-in-the-Small-and-Many. *Proceedings of ACM/IFIP/USENIX Middleware Conference*, Rio De Janeiro, Brazil, 2003.
 19. S. Zachariadis et. al. XMIDDLE: Information Sharing Middleware for a Mobile Environment. *Proceedings of 24th International Conference on Software Engineering (ICSE)*, Orlando, FL, May 2002.
 20. M. Gudgin, M. Hadley, et al., Simple Object Access Protocol Version 1.2, 2003
 21. The Globus Alliance, <http://www.globus.org>, 2005.

22. R. Fielding, Architectural Styles and the Design of Network-based Software Architectures, *Ph.D. Dissertation*, University of California, Irvine, 2000.
23. I. Muslea, S. Minton, and C. Knoblock, Hierarchical Wrapper Induction for Semistructured Information Sources, *Autonomous Agents and Multi-Agent Systems*, vol. 4, pp. 93-114, 2001.
24. H. M. Sneed. The Rationale for Software Wrapping. In *Proc. of the International Conference on Software Maintenance*, 1997.
25. R. N. Taylor, N. Medvidovic, et al. A Component- and Message-Based Architectural Style for GUI Software, *IEEE Transactions on Software Engineering*, vol. 22, no. 6, pp. 390-406, 1996.
26. MinML A Minimal XML parser. <http://www.wilson.co.uk/xml/minml.htm>.
27. Jigsaw Overview. <http://www.w3.org/Jigsaw/>.