GLIDE: A Grid-based Light-weight Infrastructure for Data-intensive Environments

Chris A. Mattmann[1,2], Sam Malek[2], Nels Beckman[2],
Marija Mikic-Rakic[2], Nenad Medvidovic[2], Daniel J. Crichton[1]

[1]Jet Propulsion Laboratory, California Institute of Technology
[2]University of Southern California

Abstract. The promise of the grid is that it will enable public access and sharing of immense amounts of computational and data resources among a large number of individuals and institutions. However, the current grid solutions make several limiting assumptions that curtail their widespread adoption in the emerging decentralized, resource constrained, embedded, autonomic, and mobile (DREAM) environments: they are designed primarily for highly complex scientific problems, and therefore require powerful hardware and reliable network connectivity; additionally, they provide no application design support to grid users (e.g., scientists). To address these limitations, we present GLIDE, a prototype light-weight, data-intensive middleware infrastructure that enables access to the robust data and computational power of the grid on DREAM platforms. GLIDE embodies a number of features of an existing data grid solution within the framework of an existing DREAM middleware solution with extensive application design capabilities. We illustrate GLIDE on an example mp3 file sharing application. We discuss our early experience with GLIDE and present a set of open research questions.

1 Introduction

One of the most exciting and promising technologies in modern computing is the grid [8,12]. 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.

Current *grid* technologies provide extensive support for describing, modelling, discovering, and retrieving data and computational resources. Unfortunately, they are implemented using middleware infrastructures that leverage both heavyweight and computationally intensive protocols and objects [6]. As such, the current grid software systems are not applicable to the domain of decentralized, resource constrained, embedded, autonomic, and mobile (DREAM) environments. The existing grid technologies also

1. Earth Science Data Systems Section, Pasadena, CA 91109 Tel: +1 (818) 354.9155.
Email: {chris.mattmann,daniel.crichton}@jpl.nasa.gov. WWW: http://oodt.jpl.nasa.gov
2. Computer Science Department, Los Angeles, CA 90089-0781 Tel: +1 (213) 740.6504.
E-mail: {mattmann,malek,nbeckman,marija,neno}@usc.edu. WWW: http://sunset.usc.edu/~softarch/

lack support for systematic design, implementation, and evolution of grid-based software systems. Finally, current development, deployment, and runtime adaptation support for the grid is ad-hoc: shell scripts abound, makefiles are the common construction and deployment tool, and any adaptation is usually handled by restarting the entire system.

Given the central role software architectures have played in engineering large-scale distributed systems [21], we hypothesize that their importance will only grow in the even more complex (grid-enabled) DREAM environments. This is corroborated by the preliminary results from several recent studies of software architectural issues in embedded, mobile, and ubiquitous systems [15,24]. In order for architectural models to be truly useful in any development setting, they must be accompanied by support for their implementation [15]. This is particularly important for the DREAM environments: these systems will be highly distributed, decentralized, mobile, and long-lived, increasing the risk of architectural drift [21] unless there is a clear relationship between the architecture and its implementation. To address these issues, several light-weight *software architecture*-based solutions [18,24] supporting the design, implementation, and evolution of software systems in DREAM environments have recently emerged. However, these solutions are still not directly supporting the grid: they have not focused on, and uniformly lack facilities for, resource and data description, search, and retrieval.

A recent focus of our work has been on addressing the limitations of the grid by bridging the two approaches described above. To that end, we have drawn upon our previous experience in developing the OODT data grid middleware [4,14], along with our experience in developing Prism-MW, a lightweight, mobile middleware for resource constrained devices [15,18], to arrive at GLIDE, a *g*rid-based, *l*ightweight *i*nfrastructure for *d*ata-intensive *e*nvironments. GLIDE was built with a focus on addressing both the resource limitations and lack of systematic application development support of the current grid technologies. GLIDE strives to marry the benefits of Prism-MW (architecture-based development, efficiency, and scalability) with those of OODT (resource description, discovery, and retrieval). We have performed a preliminary evaluation of GLIDE using a series of benchmarks, and have successfully tested it by creating a mobile media sharing application which allows users to share mp3 files on a set of distributed PDAs. While this work is still in its early stages, our initial results have been promising and have pointed to several avenues of future work.

The rest of this paper is organized as follows. Section 2 describes the existing grid middleware infrastructures, including OODT, and presents an overview of Prism-MW. Section 3 describes the design, implementation, and evaluation of GLIDE and is illustrated using an example MP3 sharing application. The paper concludes with an overview of future work.

2 Background and Related Work

GLIDE has been inspired by a set of related projects along with our own existing work in three areas: computational and data grids, light-weight middleware and protocols, and implementation support for software architectures. In this section, we first briefly summarize Globus, the *de facto* standard grid toolkit, as well as several additional com-

putational grid solutions, and their most obvious limitations that motivated GLIDE. We then describe OODT, the grid technology used by NASA and the National Cancer Institute, along with other representative approaches to large-scale data sharing. Finally, we summarize Prism-MW, a light-weight middleware platform that explicitly focuses on implementation-level support for software architectures in DREAM environments; we also briefly overview a cross-section of representative light-weight middleware platforms.[1]

2.1 Globus

The Globus Toolkit [8,12] is an open-source middleware framework for constructing and deploying grid-based software systems. It combines a middleware transport layer [9], a suite of grid-services and protocols (e.g. Grid Resource Allocation Management or GRAM [1], Grid File Transfer Protocol, or GridFTP [1], and Metadata Catalog Service, or MCS [23]), and a web-services based implementation infrastructure [9] to support construction and deployment of grid-based software systems. Globus realizes the basic goal of the grid: the establishment of virtual organizations sharing computational, data [3], metadata, and security resources. However, Globus lacks several features that would ease its adoption and use across a more widespread family of software systems and environments.

First, Globus lacks architecture-based software development support. This severely hinders its utility as a development platform and middleware technology: Globus relies on shell scripts and makefiles as the only means of supporting deployment, configuration, and evolution of grid-based systems. Secondly, Grid protocols and services implemented in Globus all rely heavily on many interactions with multiple components, even while the service is idle. For example, Grid-FTP mandates that separate, parallel control channels exist between the Grid-FTP client and each one of the servers from which it transfers the files using the Globus infrastructure. Maintenance and coordination of these control channels requires a much broader constituent of resources than a simple socket connection. As another example, the MCS service relies on a MySQL data store that provides metadata used to describe resources in a Globus-connected grid system. For these reasons, Globus is not suitable for use in DREAM environments.

In addition to Globus, several other grid technologies have emerged recently. Alchemi [1] 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 [13] is a framework for developing distributed applications based on a peer-to-peer topology. Its layered architecture provides developers with abstractions of low-level protocols along with services such as host-discovery, data sharing, and security.

2.2 OODT

GLIDE is directly motivated by our work in the area of data-grids, specifically on the Object Oriented Data Technology (OODT) system [4,14]. We have adopted an archi-

1. Recall that OODT and Prism-MW form the foundation for GLIDE.

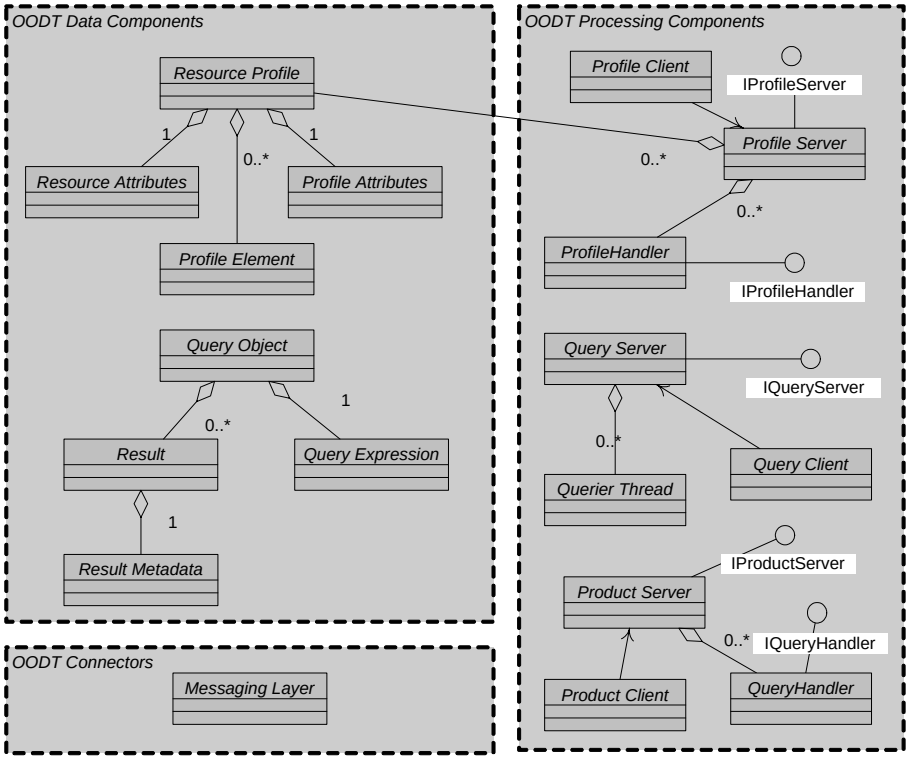

Figure 1. UML class diagram of OODT

tecture-centric approach in OODT, in pursuit of supporting distribution, processing, query, discovery, and integration of heterogeneous data located in distributed data sources. OODT contributes a set of software components, implementation-level connectors, and software configurations, which are used to integrate existing data-intensive systems, and to ingest and retrieve data. Figure 1 shows the high-level design of OODT. Below we describe the major components of OODT:

OODT Data Components

- **Resource Profile** is a data structure which describes both 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-providing software systems, and resource profiles themselves. Resource profiles may contain additional resource-describing metadata [4].
- **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.

OODT Processing Components

- **Product Servers** 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. A query results in either data retrieval or use of a remote computational resource.
- **Profile Servers** generate and deliver metadata [5] 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 Profile*s that satisfy the query.
- **Query Servers** accept *Query Object*s, 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 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.

OODT Connector

- The **Messaging Layer** connector is a data bus which marshals resource profiles and query objects between OODT client and server components.

Although OODT has shown to be quite effective at leveraging its architectural focus on data-grid systems, its heavyweight approach to implementing connectors (e.g., using off-the-shelf middleware such as CORBA [20] and RMI [25]) is not suitable for deployment in DREAM environments. At the same time, several of the core capabilities of OODT (e.g. resource discovery, query optimization, and resource description using metadata) provide a sound basis for the services in GLIDE discussed in this paper.

There are several other technologies for large-scale data sharing. Grid Data Farm [26] 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 [22] 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 [18] is a middleware platform that provides explicit implementation-level support for software architectural constructs. The top-left diagram in Figure 2 shows the

class design view of Prism-MW's core. *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 run-time. A distributed application is implemented as a set of interacting *Architecture* objects, communicating via *DistributionConnector*s across process or machine boundaries. *Component*s in an architecture communicate by exchanging *Event*s, which are routed by *Connector*s. 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 run-time probing of a *Brick*'s behavior.

Prism-MW's native support for architectural abstractions and extensive separation of concerns allow it to be tailored to different architectural styles [21]. To that end, Prism-MW provides the following:

1. the ability to distinguish among different architectural elements of a style — *Brick* has an attribute that identifies its style-specific type (e.g., `Client`, `Server`, `Pipe`, `Filter`, and so on).
2. the ability to specify the architectural elements' stylistic behaviors — *ExtensibleComponent*, *ExtensibleConnector*, and *ExtensiblePort* classes provide tailorable event routing, synchrony, and distribution support, respectively.
3. the ability to specify the rules and constraints that govern the architectural elements' valid configurations — *ExtensibleArchitecture* provides tailorable support for ensuring the topological constraints of a given style (e.g., in the client-server style, `Clients` can connect to `Servers`, but not to one other `Clients`).
4. the ability to use multiple architectural styles within a single application — *ExtensibleArchitecture* implements the *IComponent* interface, thereby allowing hierarchical composition of components. Each hierarchical component is internally composed of subarchitectures that can adhere to different architectural styles.

Prism-MW also comes with explicit support for component deployment [16] in the form of Prism-DE, an environment for deploying and monitoring Prism-MW based software systems.

Prism-MW directly enables several desired features of GLIDE. First, it provides the needed low-level middleware services for use in DREAM environments, 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 can be extended to aid GLIDE users in constructing, deploying, and evolving grid-based DREAM systems.

A number of additional middleware technologies exist that support either architectural design or for mobile and resource constrained computation, but rarely both [18]. An example of the former is Enterprise Java Beans (EJB), a popular commercial technology for creating component-based, distributed Java applications. An example of the latter is XMIDDLE [27], an XML-based data sharing framework targeted at mobile en-

vironments. XMIDDLE provides data replication across mobile devices and pays particular attention to the frequent disconnection of these devices.

3 Arriving at GLIDE

GLIDE is a hybrid grid middleware which combines the salient properties of Prism-MW and the core services of OODT, with the goal of extending the reach of the grid beyond super-computing and desktop-or-better platforms to the realm of DREAM environments. To this end, the myriad of heterogeneous data (music files, images, science data, accounting documents, and so on) and computational (web services, scientific computing testbeds, and so on) resources made available by heavy-weight grids can also be made available on their mobile counterparts. Thus, mobile grids enabled by GLIDE have the potential to be both *data-intensive*, requiring the system to provide rich metadata describing the abundant resources (and subsequently deliver and retrieve large amounts of them), as well as *computationally-intensive*, focused on discovering and utilizing data, systems, authorization, and access privileges to enable complex, distributed processing and workflow.

Existing grid solutions such as Globus [12] and OODT [4] take a completely agnostic approach to the amount of hardware, memory, and network resources available for successfully executing, deploying, and evolving a grid-based software system. These technologies consider the system's architectural design to be outside their scope. In addition, they also fail to provide sufficient development-level support for building, deploying, and evolving software applications. A solution that overcomes these limitations is needed to realize the widely stated vision of "data and computation everywhere". By implementing the core grid components of OODT using Prism-MW, we believe to have created an effective prototype platform for investigating and addressing these limitations.

3.1 GLIDE's Design

The core components of OODT include six processing components, two data components, and one messaging layer connector (recall Section 2.2). We used Prism-MW to implement OODT's core, thus arriving at GLIDE, shown in Figure 2. Our main objective was to retain the key properties and services of Prism-MW and OODT, such that GLIDE would support architecture-based design, implementation, deployment, and (runtime) evolution of data-intensive grid applications in DREAM environments. Below we describe GLIDE's architecture in light of this objective.

Each OODT 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, as Figure 2 shows, the dependency relationships between GLIDE's *Client* and *Server* processing components, which existed in OODT (recall Figure 1), are removed.

GLIDE's components use Prism-MW's *Event*s to exchange messages. OODT data components are sent between processing components by encapsulating them as parameters in Prism-MW *Event*s. Leveraging Prism-MW's *Event*s to send and receive differ-

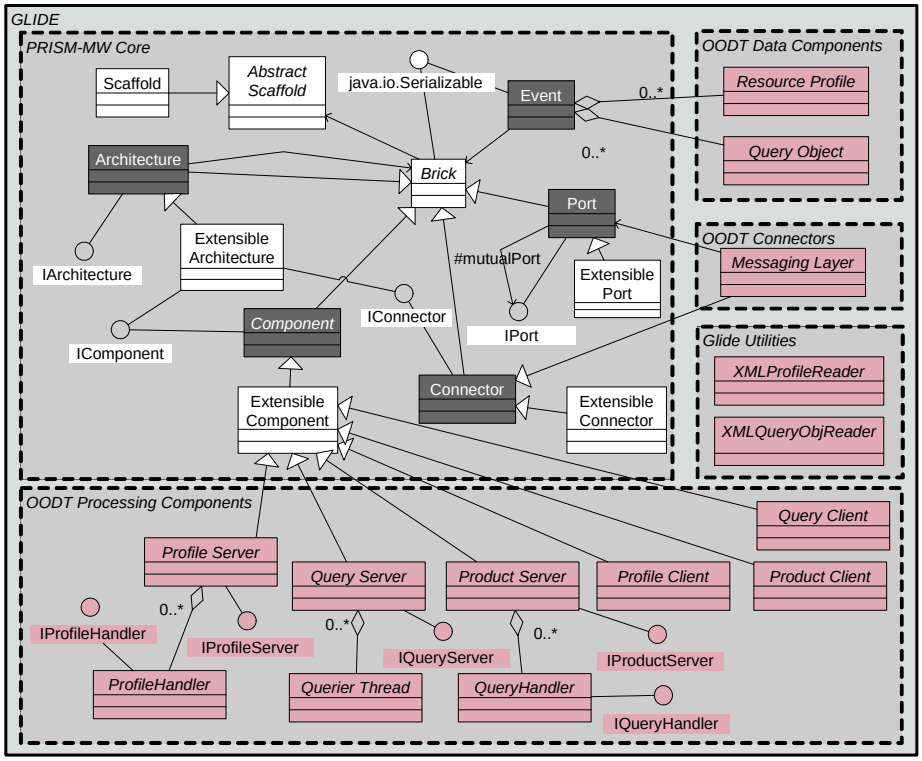

Figure 2. UML class diagram of GLIDE showing its Prism-MW and OODT foundation.

ent types of data enables homogenous interaction among GLIDE's processing components.

As discussed in Section 2.2, OODT's connectors are not suitable for DREAM environments because of their heavy-weight. 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 OODT's messaging layer class in GLIDE. GLIDE's connector leverages Prism-MW's port objects that allow easy addition or removal of TCP/IP connections, thus allowing the system's topology to be adapted at runtime. GLIDE's connector also implements event filtering such that only the requesting client receives responses from the server.

Finally, to support interoperability of GLIDE with other grid technologies, we provide two additional utility classes: **XMLProfileReader** and **XMLQueryObjReader** parse an XML representation of a resource profile and query object data structure, respectively. Each string is parsed into a GLIDE data object. Similarly, resource profiles and query objects can be serialized into XML.

The high degree of decoupling among GLIDE components directly aids easy dynamic adaptation, the lack of which is a key limitation in current grid systems. Ability

to easily adapt a system's software architecture is an important property missing in OODT that can be leveraged to improve the system's functionality, scalability, availability, latency, and so on. For example, our recent studies [17] have shown that the availability and latency of software systems in DREAM environments can be improved significantly via dynamic adaptation.

3.2 Sample Application Using GLIDE

In order to evaluate the feasibility of GLIDE, we designed and implemented a *M*obile *M*edia *S*haring application (MMS), shown in Figure 3. 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 asychronously 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.

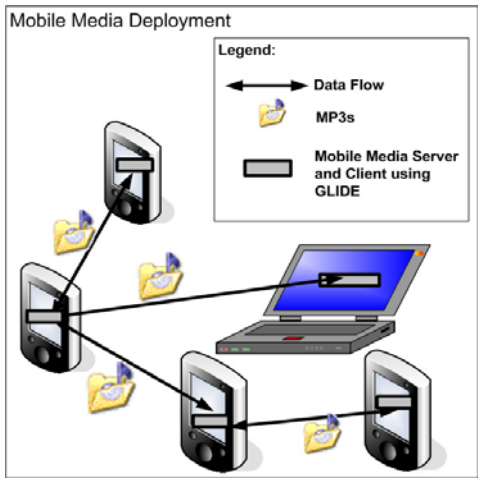

Figure 3. Mobile Media Sharing Application

MobileMediaClient contains a single component called *MediaQueryGUI*, which provides a graphical front-end for creating MP3 queries. MP3 queries use two query parameters, *MP3.Genre* (e.g., rock, classical) 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 the messaging layer connector, and has been deployed on DREAM devices. In the next section we evaluate GLIDE using MMS as an example.

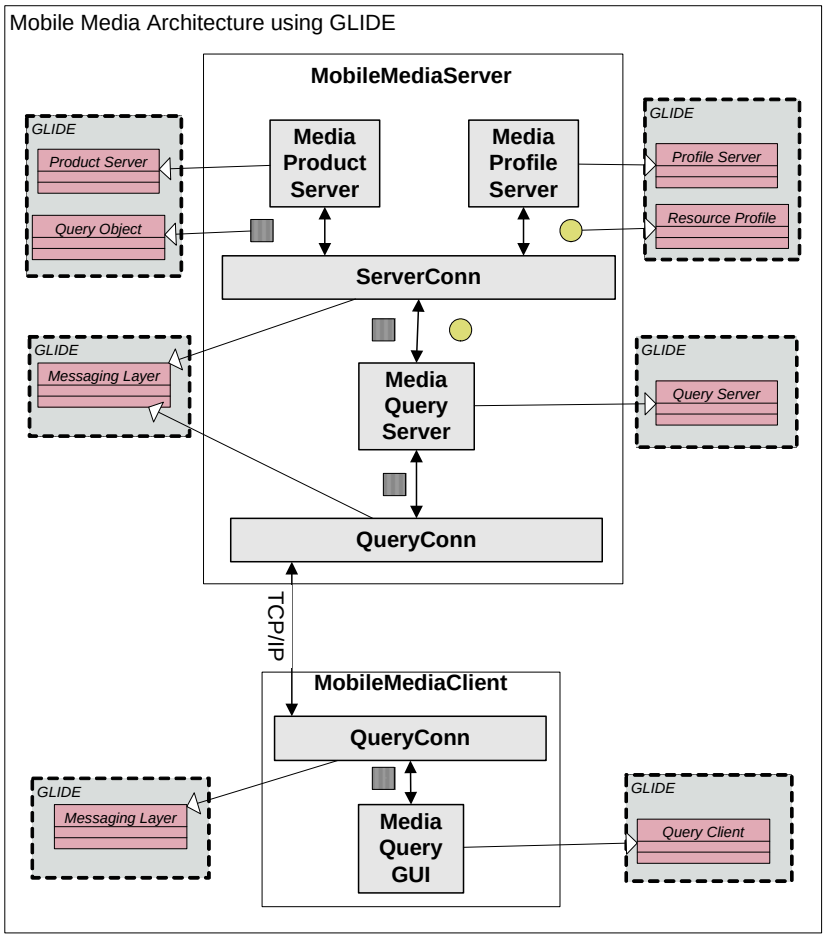

Figure 4. Mobile Media Application Architecture

3.3 Evaluation

In this section we evaluate GLIDE along the three dimensions outlined in the Introduction: (1) its support for architecture-based development, (2) its support for deployment and evolution of the software system, and (3) its suitability for DREAM environments.

3.3.1 Architecture-Based Development Support

GLIDE inherits architecture-based development capabilities, including style awareness, from Prism-MW [18]. 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. The variations of MMS leveraged existing

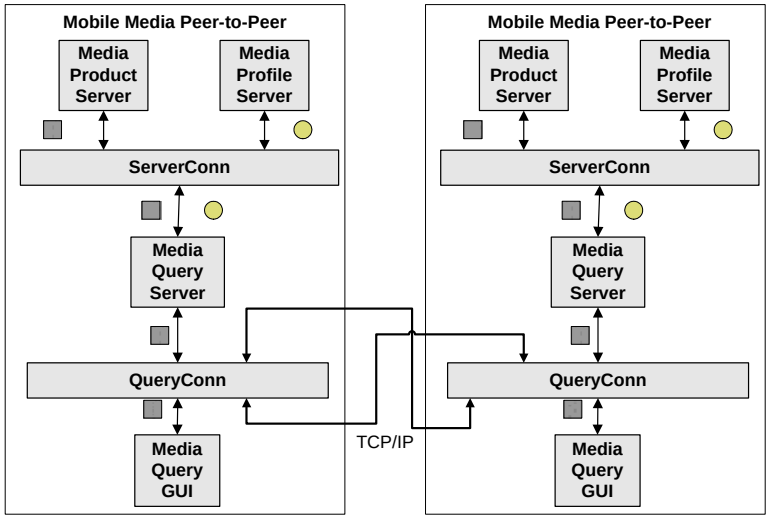

Figure 5. Peer-to-peer variation of the Mobile Media Sharing application.

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 client side, and one component on the server side. Figure 5 shows the peer-to-peer variant of MMS.

3.3.2 Deployment and Evolution Support

Drawing from its basis on Prism-MW, applications developed using GLIDE inherit key deployment and evolution capabilities that were not present in OODT. These capabilities include a graphical component deployment environment [16] and automated runtime architectural reconfiguration [17]. These capabilities are necessary in DREAM environments, where bandwidth, connectivity, memory, and the like hinder the utility of conventional deployment techniques (e.g., build scripts and configuration files) and conventional evolution techniques (e.g., bringing down, modifying, recompiling, and restarting the system). In addition to preventing runtime system modifications, existing grid technologies require users to be familiar with the above conventional deployment and evolution techniques. Considering that many grid users are scientists with little or no formal software engineering training, they are ill-equipped to deploy and evolve their grid systems. In Prism-MW this type of support is a core capability; GLIDE's basis in Prism-MW therefore directly enables it to effectively address this limitation.

3.3.3 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 [18] along with the following techniques to improve GLIDE's performance and minimize the effect of the computing environment's heterogeneity:

- MinML [19], a lightweight XML parser, to parse the resource profiles and query object data structures.
- W3C's Jigsaw Web Server Base64 Encoding Library [11] to compress (at the product server end) and uncompress (at the product client end) the exchanged data.
- 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.
- 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.

Table 1: Memory footprint of MobileMediaServer and MobileMediaClient in GLIDE

MobileMediaServer	Java Packages	# Live Objects	Total Size (bytes)
Java	java.lang	36	2016
GLIDE's Implementation of OODT components	glide.product	1	24
	glide.profile	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***

MobileMediaClient

Java	java.lang	28	1568
GLIDE's implementation of OODT 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***

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.

4 Conclusions and Future Work

This paper has presented the motivation for and prototype implementation of a grid middleware for decentralized, resource constrained, embedded, autonomic, and mobile (DREAM) environments. Although the results of our work to date are very promising, a number of pertinent issues remain unexplored.

Our immediate work will focus on (1) devising and evaluating suitable architectural styles for grid computing, using GLIDE's existing support for user-specified styles; (2) extending GLIDE to provide a set of meta-level services, including monitoring of data and metadata within a GLIDE system; and (3) addressing the issues of trust in grid applications. These issues represent but a small subset of related concerns that are emerging in this domain. We believe that GLIDE will afford us an effective platform for investigating this rich research area.

5 References

[1] Alchemi .NET Grid Computing Framework. http://www.alchemi.net/doc/0_6_1/index.html

[2] D. Booth et al. Web Services Architecture, http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/

[3] A. Chrevenak et al. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets. *Journal of Network and Computer Applications*, 2000.

[4] D. Crichton et al. A Component Framework Supporting Peer Services for Space Data Management. *IEEE Aerospace Conference*, Big Sky, Montana, 2002.

[5] D. J. Crichton, J. S. Hughes, and S. Kelly. A Science Data System Architecture for Information Retrieval. in Clustering and Information Retrieval. W. Wu, H. Xiong, and S. Shekhar, Eds.: Kluwer Academic Publishers, 2003, pp. 261-298.

[6] N. Davies, A. Friday and O. Storz. Exploring the Grid's potential for ubiquitous computing. *IEEE Pervasive Computing*, Vol 3. No. 2, April-June, 2004, pp.74-75.

[7] M. Gudgin et al. Simple Object Access Protocol Version 1.2, 2003, http://www.w3.org/TR/soap/.

[8] I. Foster et al. Grid Services for Distributed System Integration. *IEEE Computer*, pp. 37-46, 2002.

[9] I. Foster et al. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. *Globus Research*, Work-in-Progress 2002.

[10] I. Foster. What is the Grid? A Three Point Checklist. *GRIDToday*, Vol 1. No. 6, July 22, 2002.

[11] Jigsaw Overview. http://www.w3.org/Jigsaw/.

[12] 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.

[13] N. Maibaum, T. Mundt. JXTA: A Technology Facilitating Mobile Peer-to-Peer Networks. *MobiWac 2002.* Fort Worth, TX, October 2002.

[14] C. Mattmann et al. Software Architecture for Large-scale, Distributed, Data-Intensive Systems. 4th *IEEE/IFIP Working Conference on Software Architecture (WICSA-4)*, Oslo, Norway, 2004.

[15] N. Medvidovic et al. Software Architectural Support for Handheld Computing, *IEEE Computer (Cover Feature)*, vol. 36, pp. 60-67, 2003.

[16] M. Mikic-Rakic and N. Medvidovic. Architecture-Level Support for Software Component Deployment in Resource Constrained Environments, *1st International IFIP/ACM Working Conference on Component Deployment (CD'02)*, Berlin, Germany, 2002.

[17] M. Mikic-Rakic and N. Medvidovic. Software Architectural Support for Disconnected Operation in Highly Distributed Environments. *7th International Symposium on Component Based Software Engineering (CBSE-7)*, Edinburgh, UK, 2004.

[18] M. Mikic-Rakic and N. Medvidovic. Adaptable Architectural Middleware for Programming-in-the-Small-and-Many. *ACM/IFIP/USENIX International Middleware Conference (Middleware 2003)*, Rio De Janeiro, Brazil, 2003.

[19] MinML A Minimal XML parser. http://www.wilson.co.uk/xml/minml.htm.

[20] Object-Management-Group. The Common Object Request Broker: Architecture and Specification, 1998.

[21] D. E. Perry and A. L. Wolf. Foundations for the Study of Software Architecture, *ACM SIGSOFT Software Engineering Notes (SEN)*, vol. 17, 1992.

[22] A. Rajasekar, M. Wan, R. Moore. MySRB and SRB - Components of a Data Grid. High Performance Distributed Computing (HPDC-11). Edinburgh, UK, July 2002.

[23] G. Singh et al. A Metadata Catalog Service for Data-Intensive Applications. *IEEE International Conference on Supercomputing*, 2003.

[24] J. P. Sousa and D. Garlan. Aura: an Architectural Framework for User Mobility in Ubiquitous Computing Environments. *3rd Working IEEE/IFIP Conference on Software Architecture (WICSA-2002)*, Montreal, Canada, 2002, pp. 29-43.

[25] Sun-Microsystems, Java Remote Method Invocation (RMI), http://java.sun.com/products/jdk/rmi/index.jsp, 2004.

[26] 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.

[27] S. Zachariadis et. al. XMIDDLE: Information Sharing Middleware for a Mobile Environment. ICSE 2002, Orlando, FL, May 2002.