Preliminary version. Final version appears in *Concurrency and Computation: Practice and Experience*, 22: 2313–2322.

PUBLICATION AND CONSUMPTION OF CABIG DATA SERVICES USING .NET*

MARTY HUMPHREY, JIE LI, AND NORM BEEKWILDER

Department of Computer Science, University of Virginia, Charlottesville, VA 22904

ABSTRACT

The cancer Biomedical Informatics Grid (caBIG) is revolutionizing the way medical researchers share information and collaborate. A key to caBIG's continued success will be interoperability. However, to date, only a single code base (in Java) has been used to create a set of tools and run-time services for caBIG. This paper presents the first significant exploration into the use of Microsoft's .NET Framework and Visual Studio for caBIG. Given its substantial existing community, a .NET-based set of tools for caBIG can significantly increase the pool of qualified software designers and developers for caBIG. Arguably more importantly, a second development foundation could facilitate revisiting a broad set of design decisions made to date in caBIG that have perhaps been unduly based directly or indirectly on a single underlying software technology. We begin by describing issues we have encountered in building relatively simple .NET-based clients to existing caBIG services. Next, we describe how we leverage Microsoft ADO.NET Data Services as the foundation for caBIG Data Services, in particular for the caBIO data set. We find ADO.NET Data Services has a uniquely strong potential to facilitate rapid development and deployment. We conclude with a discussion of the roadmap of our project's future activities.

1. INTRODUCTION

The cancer Biomedical Informatics Grid (caBIG) [1][2] is revolutionizing the way medical practitioners will be able to share information. As the underlying service-oriented infrastructure of caBIG, caGrid [3][4] is the foundation upon which to connect *cancer researchers* with the information they need. It is through caGrid that institutions can share cancer research information securely and efficiently.

A key to caGrid's continued success will be interoperability. In general, interoperability in a large-scale software system can be challenging for a number of reasons, including differing/competing standards, imprecise interpretation of a particular standard, lack of standards, and general political/economic concerns. For example, while the broad community has made excellent progress toward interoperable Web services, it should not be assumed that a client based on one software base can readily and trivially interact with a service layered upon a different set of technologies and tooling. For example, clients and services often do not

* The research and development described in this paper is in part by the National Cancer Institute under a subcontract from Booz Allen Hamilton.

precisely agree on the format of XML-based messages, causing errors to be thrown in the act of serializing and deserializing messages from the *other* technology.

Prior to the research reported in this paper, while a comprehensive set of tools and run-time services have been designed and implemented (based on Java), there had been little investigation into the use of Microsoft's .NET Framework and Visual Studio for caGrid. In addition, the potential use of SQL Server, Windows Workflow Foundation, Sharepoint, etc., has not been comprehensively considered, in particular with regard to what degree the resulting technology is interoperable with the existing set of clients and services. We have recently completed an exploratory project supported by the US National Cancer Institute (NCI) Center for Bioinformatics (which itself is supported by the US National Institutes of Health) to provide alternative platforms, tools, and development environments for caBIG. The overall goal of this project is not merely to provide a "second source" of existing technology/functionality, with identical functionality, but rather to potentially revisit a broad set of design decisions made to date in caBIG that have perhaps been unduly based directly or indirectly on a single underlying software technology. For example, the mission for this project included determining if there are unique capabilities that can be facilitated in caBIG via the wide range of technologies based on .NET.

In this paper, we first give an overview of caBIG, emphasizing its goal of semantic interoperability via a model-driven architecture. We then give an overview of the issues we have encountered in building relatively simple .NET-based clients to existing caBIG services. Next, we focus on our support for a particular, crucial type of caBIG service called *caBIG Data Services*, in which particular sets of data are exposed via a canonical service interface, with a particular query language called the caBIG Query Language (CQL), using a single underlying meta-model with common data elements. We describe how we leverage Microsoft ADO.NET Data Services as the foundation for caBIG Data Services, in particular for the caBIO data set, which is a repository of molecular biology data [5]. We find ADO.NET Data Services has a strong potential to facilitate rapid development and deployment. We conclude with a discussion of the roadmap of our project's future activities.

The remainder of this paper is organized as follows. In Section 2, we give an overview of caBIG and caGrid. In Section 3, we describe the creation of .NET clients for existing caGrid services. In Section 4, we describe the issues and approach of creating a caGrid Data Service based on .NET. In Section 5, we conclude.

2. caBIG AND caGRID

Increasingly, a cancer researcher needs to discover and use existing data related to his/her research from across the US (and world). The problem is that often this data is stored in different formats and accessed via different mechanisms. In other words, even if a cancer researcher were to know the site(s)/lab(s) at which there exists data relevant to his/her investigations, it can be a challenge to securely get the data and then to interpret its meaning/relevance. The immediate concern is that such challenges cause the replication of experiments locally. More importantly, the inability to recognize and synthesize relevant existing data sources can greatly hinder scientific exploration and discovery.

Started in 2004, the cancer Bioinformatics Grid (caBIG) is a virtual network of interconnected data, individuals, and organizations designed to enhance collaboration of cancer researchers. Overseen by the NIH National Cancer Institute (NCI), caBIG is redefining how research is conducted, care is provided, and patients/participants interact with the biomedical research enterprise. The core principles of caBIG are open access, open development, open source, and federation. To date, forty-nine (of sixty-three) US Cancer Centers in the U.S. are active in caBIG. Based on its early successes, plans are in place to expand the scope of caBIG beyond cancer and into other types of data, including general patient health records.

The NCI caCORE (Cancer Common Ontologic Representation Environment) [6] and infrastructure are the building blocks and tools for caBIG. There is an SDK specifically for creating caCORE software systems, built on the principles of Model Driven Architecture (MDA), n-tier architecture & common API for data access. Currently, the caCORE relies on UML modeling, although UML has not been mandated (the requirement is *model-driven*, not specifically UML). The caCORE SDK generated system features controlled vocabularies and registered metadata and is thus "semantically integrated", whereby a service has runtime-accessible metadata that defines the meaning of the elements using controlled terminology.

caGrid [3][4] is the underlying Grid-based software infrastructure of caBIG and consists of services, toolkits, APIs, and applications. The caGrid production environment includes:

- Community-provided services, such as Data Services and Analytical Services
- Web Application, such as the caGrid Portal (http://cagrid-portal.nci.nih.gov/web/guest/home)
- Metadata Services, including EVS (the Enterprise Vocabulary Services), the caDSR (the Cancer Data Standards Repository, used to store data models as common data elements); GME (the Global Model Exchange service, used to store XML-based representations of concepts); sand the Index Service (to register and search for specific services or service types),
- Client Applications, such as a workflow service, security admin UI, and Introduce (an "authoring toolkit" for Grid services)
- Security Services, such as Authentication Services, Dorian (for provisioning and federation of Grid user identities and credentials), GTS (the Grid Trust Service, which maintains a federated trust fabric of all the trusted credential providers in caGrid), and Grid Grouper (a service that maintains and asserts group membership)

Given its ambitious goal and vision, caBIG and caGrid face a huge number of technical and economic challenges. The software underlying must be user-friendly, scalable, secure, evolvable and evolving, able to find and process the relevant information necessary to the computation at hand, interoperable with other platforms, cost-effective, etc. Based on early successes, cancer researchers as well as the software developers using the underlying common caBIG tools are expecting and requiring more. Continuing to delivering on these requirements has the potential to be truly transformative, revolutionizing cancer research and transforming patient health care into a highly-personalized model.

3. .NET-BASED CLIENTS FOR EXISTING CAGRID SERVICES

From the beginning, a foundation of the design of caBIG and caGrid has been a focus on interoperability. However, to date, only a single code base (in Java) has been used to create a comprehensive set of tools and run-time services for caBIG. By definition, more than one software base is needed to deliver upon this goal of interoperability. Because caBIG's goal is to be a comprehensive, inclusive system for biomedical research, .NET was chosen to explore as a second significant platform. .NET and Visual Studio are an extremely popular platform by which to write software, so a .NET-based set of tools has the potential to greatly expand the number of existing developers capable of contribution to the caBIG and caGrid vision.

Our research group was contacted by NCI in the summer of 2008 to pursue this project, based on our past experiences using .NET (and the Web Services Resource Framework, WSRF [7]) as the basis of Grid services [8][9][10][11][12]. We decided that it would be most effective to focus our efforts on a specific, important use case: A researcher is studying human BRCA1 gene and wants to find information available in public resources on protein encoded by this gene. (We chose this use-case from the earlier (Java-based) training materials from caBIG annual meetings). More specifically, this use-case involved the following activities:

1. Discover multiple caGrid Data Services providing Protein information
    - Use caGrid Discovery Client
2. Find how to combine the information from these Data Services
    - Find semantically equivalent data elements (Common Data Elements) from different data services
3. Identify/query the Protein corresponding to BRCA1 gene
    - Run caBIG Query Language (CQL) queries using caGrid Data Service Client
4. Collect information on the same protein from different resources
    - Run multiple or federated CQL queries against different Data Services leveraging Common Data Elements

We focused our efforts on the third step, in particular, in this use-case by designing and implementing a simple client to an existing caGrid Data Service. Microsoft Visual Studio has an excellent mechanism by which to auto-generate client-side "proxy code" to an existing Web service via Visual Studio's "Add Service Reference" (formerly "Add Web Reference"). That is, our plan was to generate the proxy code via the "Add Service Reference" wizard within Visual Studio, and then fill-in the necessary client code by which to interact with the service and search for proteins corresponding to the BRCA1 gene. Our goal was not to create a comprehensive, graphical application but rather to investigate the issues involved in, essentially, interoperability between the .NET platform and existing (Java-based) caGrid services.

We were able to successfully create a .NET client for the caGrid Data Service. The entire code we had to write within the client is shown in Figure 1. Line 1 instantiates the client-side proxy to the service. Lines 2-10 form the query (in caBIG CQL). Line 11 interacts with the service and gets the results. Lines 12-25 deserialize the results back from the service, and lines 28-32

print the results to the screen (note that of course lines 12-25 and lines 28-32 could be combined into a single loop. Figure 2 shows the output of the client when executed.

```
1.      CaBIOSvcPortTypeClient proxy = new CaBIOSvcPortTypeClient();

2.      QueryRequestCqlQuery arg = new QueryRequestCqlQuery();
3.      arg.CQLQuery = new CQLQuery();
4.      arg.CQLQuery.Target = new Object();
5.      arg.CQLQuery.Target.name = "gov.nih.nci.cabio.domain.Gene";
6.      Attribute aa = new Attribute();
7.      aa.name = "symbol";
8.      aa.predicate = Predicate.LIKE;
9.      aa.value = "BRCA%";
10.     arg.CQLQuery.Target.Item = aa;

11.     CQLQueryResults result = proxy.query(arg);

12.     Console.WriteLine("\nXML representation of 'Genes like BRCA':\n");
13.     List<Gene> genes = new List<Gene>();
14.     for (int i = 0; i < result.Items.Length; ++i)
15.     {
16.       CQLObjectResult objRes = (CQLObjectResult)result.Items[i];
17.       Console.WriteLine(objRes.Any.OuterXml);

18.       XmlDocument doc = new XmlDocument();
19.       doc.LoadXml(objRes.Any.OuterXml);
20.       XmlElement ele = doc.DocumentElement;
21.       XmlNodeReader xnr = new XmlNodeReader(ele);

22.       XmlSerializer xs = new XmlSerializer(typeof(Gene));

23.       genes.Add((Gene)xs.Deserialize(xnr));
24.       xnr.Close();
25.     }
26.     Console.WriteLine("\nHit any key to deserialize the XML");
27.     Console.ReadKey();

28.     Console.WriteLine("\nID and Fullname of each Gene:\n");
29.     foreach (Gene gg in genes)
30.     {
31.       Console.WriteLine(gg.id + "  " + gg.fullName);
32.     }
```

Figure 1: Client code to ask caBIO service for "Genes like BRCA"

Two aspects of the client warrant further discussion. First, in our research experience with WSRF.NET, interoperability frequently was non-trivial. More particularly, we often found that proxy-code generation was problematic, and we found this to be true as well working with caBIG (probably because caBIG uses the Globus Toolkit v4 [13], which was the same Java-based platform we attempted to build clients and services around in WSRF.NET). We could not use the Visual Studio wizard to directly generate proxies; instead, we have to use the command prompt to acquire the XSD and WSDL files, hand-edit 6 lines of a particular WSDL file retrieved, and then again use the command prompt to generate the proxy code. We are currently designing and implementing our own caBIG-specific "wizard plug-in" for Visual Studio to eliminate the need to perform these operations. Second, once we were able to generate (and compile) the proxy code, the client-side code was remarkably easy to write. Utilizing Visual Studio's "intellisense" (in which, after typing a few characters, a menu appears of possible completions), we were able to follow the standard Web service client-side pattern of

instantiating the proxy, constructing the necessary arguments to a particular service operation, invoking the service operation, and then parsing the results. Note that proxy-generation created definitions for CQL (lines 2-10) in addition to the definition of the service and operations; we were able to determine the appropriate strings to fill the CQL structures by interacting with the service through the existing caGrid portal. The representation for Gene (first appearing on line 13) was generated via XSD.exe using XSDs retrieved from the caBIG GME service (separately, we wrote a .NET-based client for this service). Overall, after resolving issues regarding client-side proxy generation, the creation of a client was easy and straightforward.

```
<ns2:Gene id="9188" fullName="Breast cancer 1, early onset" clusterId="194143" symbol="BRCA1"
        xmlns:ns2="gme://caCORE.caCORE/3.1/gov.nih.nci.cabio.domain" />
<ns3:Gene id="137079" fullName="Breast cancer 1" clusterId="244975" symbol="Brca1"
        xmlns:ns3="gme://caCORE.caCORE/3.1/gov.nih.nci.cabio.domain" />
<ns4:Gene id="1685" fullName="Breast cancer 2, early onset" clusterId="34012" symbol="BRCA2"
        xmlns:ns4="gme://caCORE.caCORE/3.1/gov.nih.nci.cabio.domain" />
<ns5:Gene id="136510" fullName="Breast cancer 2" clusterId="236256" symbol="Brca2"
        xmlns:ns5="gme://caCORE.caCORE/3.1/gov.nih.nci.cabio.domain" />
```

Figure 2: Output of execution of client code

4. .NET-BASED CAGRID DATA SERVICES

Having successfully generated a .NET-based client for an existing caGrid service, we then turned our attention to the use of .NET within a service, which we believed would be more challenging, and again focusing on the use-case from earlier. That is, we set out with the specific goal of creating a .NET-based *service* that held the caBIO data. The fundamental challenge we faced was how to design and implement an externally-facing Web service that would understand CQL and retrieve the appropriate caBIO data from a back-end data store (SQL Server). An important determination of success would be the ability to aim our existing caBIO .NET client at the resulting .NET-based caBIO service and achieve the same results (Genes).

The architectural components of our .NET-based caGrid Data Service are shown in Figure 3. SQL Server 2005 is used to store the data, in this case the caBIO data. One approach considered but not chosen was to embed the appropriate SQL invocations into the externally-facing Web service. Instead, we chose to use a new technology from Microsoft called *ADO.NET Data Services* [14] , which exposes a REST [15] interface and internally relies on LINQ [16] to retrieve data from SQL Server. In other words, instead of embedding potentially complex SQL operations into our externally-facing Web service, the challenge became how to construct the appropriate URI syntax to access the back-end data. We used the Windows Communication Foundation (WCF) as the basis for the externally-facing Web service. The remaining challenges within this service were to: implement the canonical interface of the caBIG Data Service, “understand” CQL in order to map from CQL queries into REST URIs, and how to implement the necessary methods in order to register with the caGrid “Index Service”.

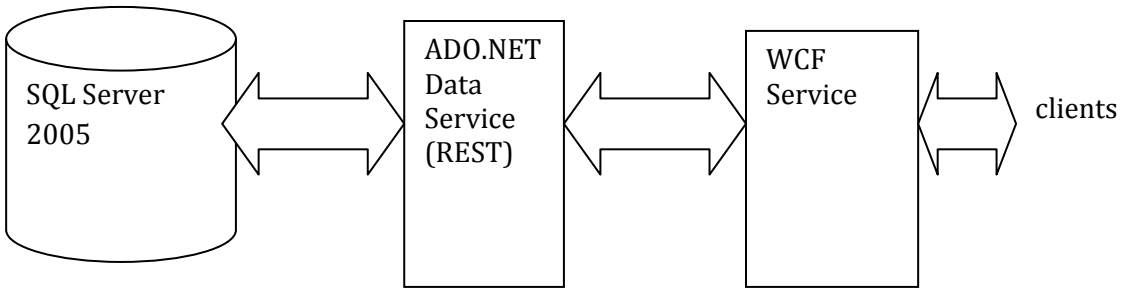

Figure 3: Architecture of .NET-based caGrid Data Service

After ingesting the data into SQL Server, the creation of the ADO.NET Data Service (REST) went very smoothly. Visual Studio has a graphical wizard specifically for creating a ADO.NET Data Service; the wizard interacts with the SQL Server to create a "ADO.NET Entity Data Model", from which all aspects of the REST service are defined. After completion of the wizard, there is only one line of code that needs to be modified before compiling and deploying the service.

Next, we created the externally-facing WCF service. This is the (only) service that the caGrid clients will interact with in order to retrieve caBIO data (the REST service is not exposed to external clients). The conformance to the existing caGrid Data Service WSDL was straightforward, via existing support within Visual Studio. That is, after editing a single file as per the instructions for generating client proxy code, we were able to generate proxy code that conforms to the WSDL (as with the client, this proxy code included a basic "understanding" of CQL). When creating a client, the basic pattern is to instantiate a proxy object to the service and then "perform operations" on this object (as if the object were "local"). In contrast, when creating a service, it is necessary to change a single line of code within the service to specify that the service conforms to the interface of the WSDL-generated proxy.

Next, a Visual Studio wizard is used to "Implement the interface" of this service, when results in the generation of stubs for each of the methods exposed by the service (which conforms to the proxy code generated from the caGrid Data Service). We then needed to generate the classes that the clients want returned (in other words, we have proxy objects via *CABIOEntities.cs* so that the Web service can represent what is returned from the REST Data Service, but this representation is not what clients of the Web service will understand; we now need to generate classes for representations that the clients will recognize). For this, basically, we need to generate classes from the XSD that corresponds to the CABIO UML model.

Finally, we implemented those methods that were necessary to register the service with the caGrid Index Service. That is, registering is a two-phase operation. In the first phase, we inform the caGrid Index Service of the existence (URL) of this new Data Service. In the second phase, the Index Service attempts to obtain certain information (location, owner, type of service, etc.) from the service by attempting to invoke certain well-know operations. Within the service code, we were able to hand-generate this information by special-casing the code to reflect our particular service (this would have to be changed in other services).

Upon completion of the above steps, the service was deployed and ready to be discovered (by clients/people via the Index Service). A fully-operational service would require extensive CQL processing (with the mapping to the REST URIs of the ADO.NET Data Service). Because our

goal was to show that this was indeed possible, we chose to only implement the support necessary to answer a query from the client regarding “Genes like BRCA”. This code is shown in Figure 4 (“query” is the name of the operation exposed via the WSDL of the caGrid Data Service). Lines 2-11 perform a rudimentary parse of the CQL query. Lines 12-14 make a connection to the ADO.NET Data Service. Lines 15-17 perform the query (using LINQ), and lines 18-19 count the number of matching Genes. The remaining lines essentially transform the SQL Server “internal” representation (“masterModel.GENE_TV”) into a corresponding canonical representation for the caGrid client (“Gene”) and return the information to the client.

```
1.  public queryResponse query(queryRequest request) {

2.    QueryRequestCqlQuery cql_q = request.cqlQuery;
3.    CQLQuery q = cql_q.CQLQuery;
4.    QueryModifier qm = q.QueryModifier;
5.    Object cql_target = q.Target;

6.    if ((cql_target.name == "gov.nih.nci.cabio.domain.Gene") &&
7.        (cql_target.Item != null))  {
8.      Attribute target_item_attribute = (Attribute) cql_target.Item;

9.      if ((target_item_attribute.predicate != Predicate.LIKE) ||
10.         (target_item_attribute.name != "symbol"))
11.       throw new NotImplementedException();

12.     masterModel.masterEntities svc =
13.       new masterModel.masterEntities(new
14.       Uri("http://localhost:1059/WebDataService1.svc"));

15.     var query1 = from g in svc.GENE_TV where
16.                  g.SYMBOL.Contains(target_item_attribute.value)
17.                  select g;

18.     int count = 0;
19.     foreach (masterModel.GENE_TV gtv in query1) { ++count; }

20.     queryResponse rv = new queryResponse();
21.     rv.CQLQueryResultCollection = new CQLQueryResults();
22.     rv.CQLQueryResultCollection.Items = new CQLResult[count];

23.     count = 0;
24.     foreach (masterModel.GENE_TV gtv in query1)    {
25.       CQLObjectResult item = new CQLObjectResult();
26.       MemoryStream ms = new MemoryStream();
27.       Gene gg = new Gene();
28.       gg.id = (long) gtv.GENE_ID;
29.       gg.idSpecified = true;
30.       gg.fullName = gtv.FULL_NAME;
31.       gg.clusterId = (long) gtv.CLUSTER_ID;
32.       gg.clusterIdSpecified = true;
33.       gg.symbol = gtv.SYMBOL;
34.       XmlSerializer xs = new XmlSerializer(typeof(Gene));
35.       xs.Serialize(ms, gg);
36.       ms.Position = 0;
37.       XmlDocument doc = new XmlDocument();
38.       doc.Load(ms);
39.       ms.Close();
40.       item.Any = doc.DocumentElement;
41.       rv.CQLQueryResultCollection.Items[count] = item;
42.       ++count;
43.     }
44.     return rv;
45.   }
```

Figure 4: Code inside caBIO service to answer query for “Genes like BRCA”

After deployment, we were able to aim our existing .NET client at the URL of this new service and confirm its correct operation. While the service is clearly just a sampling of the full functionality, its correct operation in response to "Genes like BRCA" show that this architectural approach is feasible.

There are a number of aspects of the design and development of the caGrid .NET Data Service to further discuss. First, conforming to the existing WSDL was easy – 6 lines of code in the existing WSDL had to be changed, and then one line of the service had to be changed before being able to deploy the service (albeit with none of the functionality actually implemented). Second, the REST service was very valuable – the connection to LINQ allowed us to get data from SQL Server in an intuitive manner. (We separately have created services that use the REST interface directly). CQL processing is a challenge, as we need to fully understand the language in order to map from CQL to SQL (we have heard estimates that this would require approximately 300 lines of code, although we have not confirmed this ourselves).

5. CONCLUSION

In this project, to date, we have attempted to determine the ease/difficulty by which to produce and consume data in caBIG by using the .NET framework. While we believe that the existing Java-based approach is attractive in many ways, we believe that that opening up caBIG to .NET developers has the potential to greatly expand the developer base, and in general leverage current and future technology trends based on the Microsoft platform. We have found that client-side development has great potential today, requiring only a minor change to downloaded WSDL files before proxy code is generated and compiled. The service-side is only a little more complicated.

The most significant open issue involves security. That is, the research reported in this paper does not consider the challenges of authentication, authorization, confidentiality, integrity, etc. For many clients/services, arguably particularly notably in the medical domain, security will be a major concern. Based on past experiences with WSRF.NET, we believe that we will be able to successfully create secure clients and secure services for caBIG. In addition, we believe that we will be able to leverage existing Windows/.NET-based security infrastructures, further lowering the barrier to entry. For example, we are currently investigating the degree to which an existing Active Directory deployment can be used as the basis by which to "log onto caGrid". Preliminary results have been encouraging, and this research is continuing. Once we believe that we have successfully secured clients and services, we will direct our attention at the second major type of service in caGrid, the Analytical Service. Based on our successes with the Data Service, we believe that we will be successful with the Analytical Service.

We believe that the successes to date create many further opportunities for the future. We believe that this support described in this paper opens up the capability for rich clients and services based on the .NET framework and Visual Studio. In the future, we will attempt to more closely integrate with and leverage Silverlight, Windows Workflow Foundation, Hyper-V, Cloud Computing, the Microsoft Parallel Computing initiative, Project Oslo, etc. We believe that the caBIG developer would also greatly benefit from a more full-fledged SDK that it integrated with Visual Studio. We are currently in discussions with the NCI to determine the best plan for this future activity.

References

[1] National Cancer Institute cancer Biomedical Informatics Grid (caBIG) Public Information Site. http://cabig.cancer.gov/

[2] National Cancer Institute cancer Biomedical Informatics Grid (caBIG) Community Website. https://cabig.nci.nih.gov/

[3] Saltz J, Oster S, Hastings S, Langella S, Kurc T, Sanchez W, Kher M, Manisundaram A, Shanbhag K and Covitz P. caGrid: design and implementation of the core architecture of the cancer biomedical informatics grid. *Bioinformatics.* Vol. 22 no. 15 2006, pages 1910–1916.

[4] Oster S, Langella S, Hastings, S, Ervin S, Madduri R, Kurc T, Siebenlist FF, Foster I, Shanbhag K, Covitz P, and Saltz J. caGrid 1.0: A Grid Enterprise Architecture for Cancer Research AMIA Annu Symp Proc. 2007; 2007: 573–577.

[5] Cancer Bioinformatics Infrastructure Objects (caBIO). https://wiki.nci.nih.gov/display/ICR/caBIO

[6] Phillips J, Chilukuri R, Fragoso G, Warzel D, Covitz PA. The caCORE Software Development Kit: Streamlining construction of interoperable biomedical information services. *BMC Medical Informatics and Decision Making.* 2006;6(2).

[7] OASIS Web Services Resource Framework (WSRF). http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf

[8] Humphrey M. and Wasson G. Architectural Foundations of WSRF.NET. International Journal of Web Services Research. 2(2), pp. 83-97, April-June 2005.

[9] J.V.S. Watson, Sang-Min Park, and M. Humphrey. Toward GT3 and OGSI.NET Interoperability: GRAM Support on OGSI.NET. *2005 International Conference on Computational Science (ICCS 2005),* May 22-25, 2005. Emory University, Atlanta, GA, USA.

[10] J. Feng, L. Cui, G. Wasson, and M. Humphrey. Toward Seamless Grid Data Access: Design and Implementation of GridFTP on .NET. Proceedings of the *2005 Grid Workshop (Associated with Supercomputing 2005)*. Nov 13-14, 2005. Seattle, WA.

[11] S. Eswaran, D. Del Vecchio, G. Wasson, and M. Humphrey. Adapting and Evaluating Commercial Workflow Engines for e-Science. *2nd IEEE International Conference on e-Science and Grid Computing*. Dec 4-6, 2006, Amsterdam, Netherlands.

[12] M. Humphrey, S.-M. Park, J. Feng, N. Beekwilder, G. Wasson, J. Hogg, B. LaMacchia, and B. Dillaway. Fine-Grained Access Control for GridFTP using SecPAL. *8th IEEE/ACM International Conference on Grid Computing (Grid 2007)*, Austin, TX, Sept 19-21, 2007

[13] I. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pp 2-13, 2006.

[14] Microsoft ADO.NET Data Services. http://msdn.microsoft.com/en-us/data/bb931106.aspx

[15] Fielding R. Architectural Styles and Design of Network-based Software Architectures. PhD Dissertation. University of California, Irvine. 2000.

[16] Microsoft .NET Language-Integrated Query (LINQ). http://msdn.microsoft.com/en-us/vbasic/aa904594.aspx