Data Sharing III: Tools and Techniques to Advance Interoperability

Adrian Robert, Michael Abato, and Daniel Gardner

Laboratory of Neuroinformatics, Dept. of Physiology & Biophysics, Weill Cornell Medical College, NY, NY

Internals of Neuroinformatic Resources

In a parallel poster (Gardner et al, Data Sharing II, these meetings), we offered standards for *interfaces* promoting interoperability between neuroinformatic resources. In this poster, we present a complementary view of the *internals*—models, architectures, and development strategies—of such neuroinformatic resources.

The Human Brain Project Program Announcement (PAR-03-035) calls for "extensible, scalable and interoperable" neuroinformatic resources to "... present a plan for continued updating and maintenance..." Toward these goals, we propose design choices to aid development and implementation of efficient, maintainable, persistent, and interoperable resources to support data sharing.

The rapid growth of networked enterprise management and customer service applications in the commercial world has produced a rich set of commodity server software. Much of this—including database management systems, languages and application development platforms for scalable deployment, and methods for sharing source code—can be utilized for neuroinformatics projects.

We provide an overview of these major technologies, distilling primary options and making recommendations.

Models for Data Storage

Open Source Databases

Database	URL
PostgreSQL	http://postgresql.org
Robust, wide range of data types, good transaction/locking support, stored procedures and triggers.	
MySQL	http://mysql.com
Most popular open-source database. Excellent performance, at cost of limited transaction support, no stored procedures or triggers.	
SAP DB	http://sapdb.org
Commercial database now open-source. Not as widely used as other two but powerful, robust, and well-supported.	

Commercial Databases

Database	URL
Oracle	http://oracle.com
The gold standard, but expensive and complex to administer.	
DB2	http://ibm.com/db2
Same level as Oracle; decision should be based on company and tool support.	
SQL Server	http://microsoft.com/sql
Most other DBs can run on MS, but this can ONLY run on MS. Bought from Sybase a few years ago. Considered second-tier by many, but very well-integrated with other MS components.	

1. Open Source vs. Commercial Databases

When does a project need a commercial database? To serve a large user community, or rely on management features offered by vendors. Otherwise, performance of free, open-source databases can be very good.

Alternative	URL
Object-Relational Databases	http://objectivity.com
Attempt to apply object-oriented design/development to relational databases. Generally considered a failure due to lack of adoption, awkwardness of implementation, and lack of standards. Most relational databases have object-oriented options that go unused.	
XML Databases	http://www.rpbourret.com/xml/XMLDatabaseProds.htm
The newest family of databases, now just maturing. May be desirable when XML is a fundamental component of the overall server architecture, but many relational databases can process/return XML as well.	
Flat files	
Flat files may serve better than databases for relatively uniform data broken into large units with complex structure, or when highly distributed storage is desired. Databases can still be used for metadata and search in such cases.	

2. Alternatives to Relational Databases

When should a project consider alternatives to a relational database? This depends on the structure and quantity of data and the types and frequencies of search and retrieval the system will support.

Server Architectures

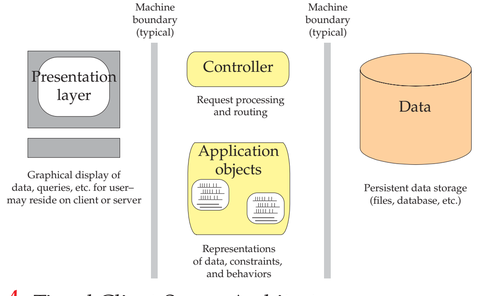
Name	Description	Pros	Cons	Technology Options
Client/Server (includes tiered server architecture)	Centralized data storage and application processing.	Straight-forward implementation with mature technology, flexible, maintainable.	Prone to storage and/or computation bottleneck	Java (J2EE), Microsoft (.NET and earlier technologies PHP, Perl, etc..
Grid/Peer-to-peer	Distributed servers/clients communicating via crafted protocols.	Unbeatable storage, computation, and bandwidth capacities.	Challenging implementation with immature technology, difficult to maintain	Globus, JXTA, web services, roll-your-own.
Hybrid	Client/server with mirroring, or peer-to-peer based on established technology (e.g., www).	Depends on specific combination.	Depends on specific combination.	Replication technologies; HTML/XML piggy-back approaches.

3. Overview of Server Architectures

In terms of server architectures, while interesting grid and peer-to-peer technologies are developing, the standard client-server model is fully mature. A three-tier server architecture is typical, in which a middle 'logic' layer mediates between the data storage and/or computational components and the application seen by the user.

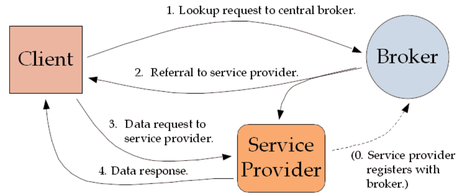
Tiered server architectures are designed to achieve maximal modularity at the most natural boundaries. The benefits are scalability (the ability to add more hardware to seamlessly serve more users) and maintainability (the ability to update or upgrade portions of the system without needing to change the other portions).

Distributed architectures are the future of computing, but how soon that future arrives for any particular application area (and in what form) will be governed by the rate at which varied technical, commercial, and political challenges can be surmounted. Technologically, various currents including file sharing, grid computing, software agents, the semantic web, and web services are all exploring separate but largely parallel paths, drawing and leading forward from various technologies underlying the current internet. While standards are emerging, the problem is more one of too many standards for some things (e.g. inter-node communications), not enough for others (semantic alignment).



4. Tiered Client-Server Architecture

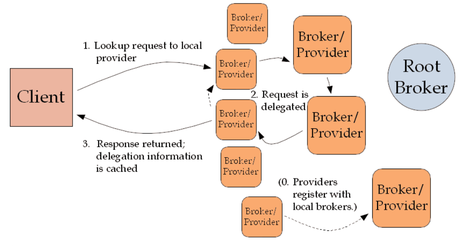
Standard three-tiered client-server architecture seen in many web applications. Scaling may be handled independently at each layer, by utilizing clustering and related techniques. In some cases (particularly in the Java world) implementation (choice of vendor, language, etc.) may be varied across these boundaries as well.



5. Grid or Agent Framework

Architecture characteristic of grid and agent-based frameworks. Clients (which may also be service providers themselves) send requests initially to a central broker. The broker analyzes the request and specifies a node to satisfy it. It may either continue mediating the communication, or pass the connection information directly to the client.

This approach is used in first-generation consumer file-sharing approaches including Napster and Audio Galaxy, as well as many current computational grid technologies and software agent platforms.



6. Distributed Brokering Grid

Variant grid architecture in which service information is cached by providers, which use the information to act as brokers. Avoids scalability problems that can overwhelm a central broker. Again, clients may themselves be brokers and providers.

This approach is used in Gnutella, FreeNet, and other second-generation file-sharing approaches, in some computational grid technologies, and in most 'peer-to-peer' systems. It is also the basis of the internet Domain Name Service (DNS).

Server Architecture Resources:

- `java.sun.com/j2ee`
- `www.apache.org`
- `microsoft.com/net`
- `www.jxta.org`
- `www.openp2p.com`
- `globus.org`

Delivering Client Functionality

Client Type	Description	Pros	Cons	Technologies
Workstation app	Windowing UI	Responsive, powerful interface	Difficult development, deployment, and maintenance	Native OS, Java
Web application	HTML, JavaScript, DHTML for browsers	Easily deployed, less interface work	Impoverished interface, high server load	Java, Microsoft, Perl, PHP, etc.
Hybrid application	Web-based but windowing UI	Easily deployed, with responsive UI	Difficult development	Java applets/webstart, Microsoft ActiveX
Workstation API	Function library	UI work is offloaded, adoption may be boosted	Adoption may be curtailed without providing at least SOME client	Java, COM, Objective C, etc.
Web API	Remote function library	Easily deployed, natural interface to data repository	High server load, immature technology	CORBA, SOAP, JXTA

7. Client Application Models

Client applications can be categorized into workstation, web-based, and hybrid types. Orthogonal to this is the distinction between applications and APIs (Application Programming Interfaces) or libraries. APIs distribute the burden of developing client applications, and it can be beneficial to publish them even when you are providing applications yourself, to encourage further development and wider adoption. They can be provided at the local workstation level (shared and/or dynamically-loaded libraries), or exposed via CORBA or SOAP, a 'web services' approach.

Workstation applications offer responsive interfaces and minimal server loads but must be manually deployed, and development and maintenance costs are high.

Web applications are easily deployed to diverse clients, but generate high server loads and offer relatively impoverished interfaces. Scripting languages (PERL, Python) can serve simple to moderately complex applications, but get difficult to maintain above a certain level of complexity, and do not scale well. Object-oriented application frameworks such as J2EE deal with these problems.

Hybrid applications (Java applets, MS ActiveX, JavaScript) combine web-based deployment with workstation-like interfaces, inheriting some of both their strengths and weaknesses.

Within any of these models, open rather than proprietary or vendor-restricted solutions are to be preferred.

URL :

- Expanded versions of these and other evaluations may be found at: `datasharing.net`

Acknowledgments:

The authors thank a very large set of open source and other developers in the informatics and computing communities for designing and making available the technologies we review, as well as the documents that describe them.

Human Brain Project Neuroinformatics research funded by NIMH and NINDS via MH/NS57153, and by NIMH SBIR MH60538.

Code Sharing to Complement Data Sharing

Code sharing is good practice, and supports community development via:

- Providing existing free code to promote further modification, extension, and enhancement that aids the provider as well as the wider community (e.g., the Apache web server).
- Advancing a standard (e.g., the Java Development Kit from Sun)

Code sharing brings strong requirements for modular design, good coding and documentation style, and version control. Commonly used supporting infrastructure includes the CVS version control system, and—for large public projects—the SourceForge collaboration website (`sourceforge.net`).

Potential disadvantages include loss of competitive advantage and costs of supporting external developer-users, but these can be mitigated by sharing code under an appropriate open source license, which limits liability and restricts the uses to which recipients can put the code.

'Copyleft' (GPL) licenses are the strongest, stating that any work using the code must also be distributed copyleft (i.e., free, and open source). Perpetually open licenses (LGPL, Mozilla) state that work using the code should include the source for at least the originally-licensed code, even if it has been modified. Initially open licenses (Apache, BSD, MIT) merely require a copyright notice to be distributed in modified or derivative works.

License	Type	1. Can redistribute?	2. Must display copyright?	3. Patents included?	4. Must include source?	5. Must include added?
GPL	Copyleft	Yes	Yes	Yes	Yes	Yes
LGPL	Perpetual	Yes	Yes	Yes	Yes	No
Mozilla, Sun	Perpetual	Yes	Yes	Yes	Yes	No
Apache	Initial	Yes	Yes	No	No	No
BSD, MIT	Initial	Yes	Yes	No	No	No

8. Summary of Open Source Licensing Rights and Requirements

Questions to ask to select the appropriate open source license:

1. Can you redistribute the code, including in a commercial product?
2. Must you include the copyright notice with redistributed code?
3. Are licenses to use associated patents included with the licensed code?
4. Must you include the source to the free portion with redistributed or combined versions?
5. Must you share source code added to it in redistributed or combined versions?

Code Sharing Resources:

- `www.opensource.org` (open-source advocacy)
- `www.fsf.org/licenses/license-list.html` (detailed analyses of open source licenses)