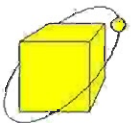

eGrok, Inc!

Inspired Solutions Through Understanding

Distributed Java Applications

An evaluation of the next stage of distributed applications using Rich Thin Client technologies

Prepared By:
Richard Conway
eGrok, Inc!

Date of Publication: 14 December 2004

2970 Day Avenue
Miami, FL 33133
USA

305.479.GROK (4765) :T
305.446.9633 :F
http://www.egrok.com

Evaluating Rich Thin Client Platforms

Prediction - Rich Client Technologies will first supplement and then replace current HTML based web technologies.

Current Situation

The current space for internet enabled applications consists primarily of dynamic web applications powered either by scripting languages (PHP, Python) or standard programming languages paired with tag libraries that are dynamically converted to HTML (ASP/Visual Basic, JSP/Java). While the capabilities of these development tools have increased tremendously over the past few years, they still fall short of providing the rich user interface/experience and functionality that even the simplest desktop application provides. The primary advantages of the web platform are as follows:

1) Zero Client Install: No software needs to be installed on the client
2) Ease of Distribution: Any client with a web browser can access the application without modifying the firewall.
3) Centralized application and data management.

While these technologies do not require the installation of any software on the client (beyond the omnipresent browser), they do require complex interactions between multiple technologies (HTML, JavaScript, Perl, XML, ASP/JSP, Visual Basic/Java) and at best provide an interface that while visually pleasing, is not very responsive and which is limited in its functionality to the user due to the technologies involved (i.e., little or no desktop access/integration and most functionality must be done server side). In addition, the developer(s) must be knowledgeable in multiple disparate technologies to effectively integrate them.

The advent of ASP.NET and Java Server Faces has improved the user (and developer) experience somewhat, but the applications developed with these technologies are still hobbled by complexity and limited functionality. Add integration with framework such as Apache STRUTS, Tiles, Spring or other and you have an even more complex system to manage. Furthermore, some sites use ActiveX controls or Java Applets to selectively add richer functionality but at the cost of additional complexity and, in the case of ActiveX controls, security concerns.

Limiting as these technologies are, they are perhaps the best solution for the development of corporate websites and sites that need to provide some limited functionality to a broad spectrum of unregistered users. But what if you need a more capable platform? What if you need to provide extremely functional applications to

remote employees and customers? Wouldn't it be great to provide the functionality of a desktop application to them over the Internet?

What The Market Needs

What is really needed is a way to provide users the rich, responsive feel of a desktop application while keeping the data and some or all of the application code safely ensconced on remote servers. Furthermore, these applications should be simple to deploy, update automatically, and operate equally well over an intranet or over the Internet. These applications should use industry standard technologies and make efficient use of network resources.

For these reasons and more, the next hot space for distributed client development will be in the area of so called "Rich Thin Clients" since these can provide dramatically better usability and functionality vis-à-vis web applications while simplifying development by reducing the number of technologies required from five or more to just one or two. By using just one or two technologies, such as Java and XML, instead of trying to integrate an alphabet soup of widely differing languages and frameworks (HTML, XML, JavaScript, Java, Perl, JSP, Java Server Faces, STRUTS, SPRING, etc.) development will be quicker and easier – and maintenance will be dramatically simplified as well.

RICH THIN CLIENT GOALS

- **Better User Experience:** Provide a rich, responsive user interface, equivalent to that of a regular desktop application
- **Easy To Deploy:** Deploy and update from a central server/repository.
- **No Changes To Firewall Required:** Communicate over HTTP using XML or similar dialect
- **Simple:** Use standard Java for nearly all programming. No JSP's, JavaScript, JSF, JSTL, etc.
- **Efficient:** Once the GUI is transferred to the client, only data is transferred thereafter. This is far more efficient than serving web pages since the screen only needs to be sent once.

New Options Emerge

Realizing the need for a more robust solution, a slew of companies have recently announced platforms that claim to provide a Rich User Interface for client applications. These frameworks promise the following benefits:

- Lightweight Deployment
- User Interface Is Only Sent To The User Once
- Updates Only Involve Data
- Data Can Be Updated Without A Page Refresh

- Richer User Interface
- Decreased Data Transfer Requirements
- Additional Functionality On Client End

As a rule, these tools can be broken into two types:

- GUI Projectors
- True Clients

In addition, many of these tools/platforms provide additional benefits that simplify and speed up development.

PLATFORM ADDITIONAL BENEFITS
• GUI Development Tools
• Wizards/Tools to simplify and speed up development
• Automatically handle client/server communications
• Operate over HTTP to avoid need to open ports in firewalls
• Data is streamed to client as needed
• Client should be able to display data as it is received without waiting for receipt of the entire data set
• Supports simultaneous update of multiple rows and fields within a grid of data
• Allows operation on handheld devices such as PocketPC and Palm

GUI Projectors

The primary focus of these tools is on providing a richer user interface than is possible using existing web technologies. These tools generally run within a java applet that is deployed in a web browser, although some of them can be deployed using Java Web Start. They use XML or a proprietary language for defining the user interface and no code is executed on the client other than the presentation applet.

Pros:

- Very Lightweight
- Provide A Richer UI Than Is Possible With Web Technologies
- Custom Presentation Widgets May Provide Some Enhanced Functionality
- May Eliminate Some Development Complexity

Cons:

- Typically Define The UI In a Non-Industry Standard Way
- Must Be Connected Via The Internet To Function
- Often Must Be Run In A Browser

True Clients

These are applications that can be distributed over the Internet, but which run on the client and can function (to some degree) without an Internet connection.

Pros:

- Provide A Richer UI Than Is Possible With Web Technologies or GUI Projector Equivalent To A Desktop Application
- Allow For Processing On Client Or Server
- May Allow Offline Processing
- Reduce or Eliminate Some Development Complexity

Cons:

- Not Necessarily Lightweight
- May Require An Internet Connection To Function

Historically these types of applications have been developed using the following technologies:

- Sockets – A low level mechanism for passing data between machines.
- CORBA – A language independent protocol for executing code on remote machines.
- Remote Method Invocation (RMI) – A Java specific method of passing objects between machines as well as executing object methods.

These require significant development effort to pass data between the client and server machines, are typically heavy (meaning lots of data is passed back and forth between the client and the server) and more importantly require ports to be opened in the firewall. More recently XML-RPC has emerged as a simpler way to communicate between machines without a lot of overhead and via HTTP.

- XML-RPC – A lightweight, language agnostic method for executing code on remote machines.

XML-RPC solves the firewall configuration problem by operating over HTTP, but is primarily designed for performing procedure calls on remote machines and requires a significant amount of work to pass complex data between the client and server.

While Sockets, CORBA, RMI and XML-RPC provide solutions to the Rich Client problem, recently new options that require less effort have emerged.

A Survey Of Rich Client Frameworks:

eGrok, Inc. recently surveyed the Rich Thin Client space looking for a platform for use in an upcoming project. After meeting with the customer a number of times to gain an understanding of their business and goals, we developed the following list of requirements for their solution:

SOLUTION REQUIREMENTS

- **Must be platform independent**
  - Purpose: Allows the application to select best of breed hardware and software for deployment.
  - Result: Java was selected as the language of choice for development for achieving platform independence
- **Must use standard development languages**
  - Purpose: Ensure easy access to resources with the knowledge to develop and support the application.
  - Result: Java was selected as the language of choice for development since it is the industry standard for enterprise application development
- **Must be easy**
  - Purpose: To simplify development and maintenance
  - Result: This driver led the following requirements
    - No EJB – EJB is complex and requires lots of horsepower
    - Try to minimize the number of languages and technologies involved
- **Must be scalable**
  - Purpose: Achieve the best performance possible with the least resources.
  - Result: To best achieve this goal, we determined that data transfer should be kept to a minimum and that some processing should be offloaded from the server to the client.
  - **Server Processing**
    - This is where much of the heavy processing and data access will take place
  - **Client Processing**
    - This allows some of the load to be removed from the server, increasing scalability of the system and providing a more responsive interface for the user. The ability to offload processing to the client also opens up the possibility for the system to work disconnected from the server which is useful when the server or network is not accessible or when working in remote locations.
- **Must provide a rich and responsive user interface on the client**
  - Purpose: Enhanced usability over standard web application interfaces
  - Result: Use a real desktop GUI rather than a web GUI. Offload some processing to the client
- **Client must be able to interact with the user's computer/workstation**
  - Purpose: Provide the same experience as a desktop application and enable local caching of data for better performance and perhaps for offline use.
- **Must Support Internationalization**
- **Must be secure**
  - Purpose: Data stored on the client and server and all communications should be secure.
  - Result: System should provide communications over SSL to protect data
- **Must support transfer of large data files between the client and server**
  - Purpose: Allows user to offload work from the server, work locally/offline, and use local applications to view/edit data before uploading the changes to the server.

After surveying the industry, we realized that one of the new Rich Client Frameworks would provide the best opportunity for addressing all of our requirements.

The following matrix lists a number of Rich Client Frameworks that purport to meet many of the requirements outlined above.

Platforms/Frameworks	Server Language	Client Language	GUI Definition Language	Browser Based	Java Web Start	Client Side Execution	Operate Off-Line	Cache Data on Client	Desktop Interraction	Secure Communications	Not EJB Based	GUI Development Tools	Wizards	Requires Firewall Ports Opene	Handheld Compatible
Altio Live	Java	XML	XML	Yes	No	No	No	Xpath	Limited		Yes			No	
Asperon AppProjector	Java	XML	XML	Yes	No	No	No	No	No			Yes		No	
Bright Side Framework	Java	Java	Java	No	Yes	Yes	Yes	Yes	Full	Yes		No	No	No	
Canoo ULC	Java	XML	XML	Yes	Yes	No	No	No	Limited	Yes	Yes	Yes	No	No	
Casabac GUI Server	Java	HTML	HTML	Yes	No	No	No	No	Limited	Yes	Yes	Yes	Yes	No	No
ClearNova ThinkCAP	Java	XML	HTML/JSP		Yes	No	No	No	Limited						
Droplets	Java	XML	XML	Yes	No	No	No	No	Limited	Yes	Yes	Yes	No	No	
InsiTech XTT	**Java**	**Java**	**Java**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Full**	**Yes**	**Yes**	**Yes**	**Yes**	**No**	**Limited**
Isomorphic	Java	DHTML	XML	Yes	No	No	No	No	No	Yes					
JTwister	Java	Java	Java	No	Yes	No	No	No	Limited	Yes					
JNDC	Java	Java/XML	Java/XML	Yes	Yes	Yes	Yes	Yes	Full	Yes	Yes	Yes	No	No	
Laszlo Systems	Java	Flash	XML	Yes	No	Yes	No	No	Limited	Yes					
Macromedia Flex	Java	Flash	MXML	Yes	No	Yes	No	No	Limited	Yes					
NexaWeb	Java	XML	XUL - XML	Yes	Yes	Yes	Yes	Yes	Limited	Yes	Yes	Yes	No	No	Yes
Wi.Ser	Java	Java	XML	Yes	Yes	Yes	Yes	Yes	Full	Yes	Yes	Yes	Yes	No	Limited

eGrok, Inc. examined these frameworks in light of the requirements identified for our project. Many of the above frameworks are new (i.e., have not yet released version 1.0) and are not ready for use in development of commercial applications. Others are based on proprietary technologies or are EJB based, which is something we feel is best to avoid – at least until the next version of EJB is released which promises to be lighter, more performant, and less complex. Based on our requirements, InsiTech's XTT emerged as the best solution for realizing our vision. XTT provides the following benefits:

Benefits Of InsiTech's XTT

- Pure Java Solution: Uses standard Java on both the server and the client reducing the number of areas of expertise required to develop an application. This also allows the developer to use any Java packages and tools to develop and enhance the product.
- Extremely efficient communication: Once a form has been downloaded only data is transferred between the client and the server.
- Provides many of the benefits of RMI with the simplicity of XML-RPC and as a bonus provides wizards to support RAD.
- Executes Java methods remotely, works over HTTP, provides server tools to interface with the database and return the desired datasets to the client in an immediately usable format, with widgets that are data aware to present the data.
- Can execute as much or as little code on the client as desired. This means data manipulation can be done on the client, reducing the server load, reducing network traffic, and improving the responsiveness of the application for the user. Data can be persisted locally to allow off-line use of the application for data gathering and/or updates at remote locations.
- Server code automatically streams data to the client as needed and the client can display data as it arrives, further improving the responsiveness of the application.
- Use of Java Web Start as a deployment tool provides the following benefits:
    - Application is always up to date as new code is automatically distributed to the client the next time they access the system.
    - Once transferred to the client, the code runs from the client, reducing network traffic.
- Provides data aware visual widgets for directly manipulating object properties. The objects can be obtained from persistence layers (Hibernate, JDO, etc.), from webservice calls, or from remote method invocation. The client also keeps track of whether an object has been inserted, updated, or deleted for eventual synchronization with the server.
- Allows use of web services on the client without the need for any additional libraries, plus provides a very intuitive way of making webservice calls from forms.
- Encourages use and integration of 3rd party Java widgets into your application. Allows developer to either use the widgets directly or to create their own data enabled widgets by extending the original and implementing a data-enabling interface.

Analysis

XTT provides the experience of a desktop application to users over an intranet or over the Internet. It provides a simple, elegant solution that is based on industry standards, has a shallow learning curve, scales well and is secure.

Since pure Java is used on both the client and the server, fewer areas of expertise are required to develop and maintain applications developed with XTT. Applications can be developed using any functionality provided by the Java language and libraries and can be developed using any Java tools and IDEs.

Most database interactions are vastly simplified by using XTT's data aware widgets. It is supremely easy to create the standard admin tools for an application that normally require quite a bit of development effort. A simple form that allows you to Create, Read, Update, and Delete from a table can take as little as 3 minutes to create. This enables developers to focus on providing functionality for the user – which is the goal of any development project – instead of focusing on management tools and infrastructure.

Finally, the ability for us to offload some processing to the client is a very attractive feature – it opens up enormous possibilities for improving performance by caching data locally, operating offline for data collection, and providing a much more responsive UI to the user.

Conclusion:

For simple dynamic web applications accessed by thousands of users that are not registered and are not paying for a service (or working for the company) – use Java Server Faces.

For desktop functionality with the benefit of access to remote data sources, use XTT.

APPENDIX

A Survey Of Root Technology Options

1. **SOAP**
    a. SOAP has experienced so much scope creep that the W3C has had to split the SOAP 1.2 specification into three separate parts (primer, messaging framework and adjuncts). There is so much XML syntax that programmers need to have extensive knowledge of how XML works to know what parts of the SOAP message to parse for the message's actual contents.
    b. With structured complexity, SOAP can do a lot more things that XML-RPC cannot. For example, it can handle "XML Schemas, enumerations, strange hybrids of structs and arrays, and custom types"
    c. SOAP offers a much easier means of messaging and communication than does CORBA and other remote protocols.
    d. If you code Java, talk to Java, and never need anything but Java, SOAP loses some of its appeal; RMI is much more accepted than SOAP and can be easier for developers to learn and use.
2. **XML-RPC**
    a. XML-RPC can be considered to be the lightweight version of SOAP.
    b. XML-RPC is a web service protocol based on sending and receiving XML messages to perform procedure calls on remote servers. The basic architecture of the XML-RPC model can be broken down into three areas: the data types, request structure and response structure.
    c. Data Types - basic data types include int, boolean, string, double, dateTime.iso8601, base64, and struct
    d. Request Structure - HTTP post message that specifies the method name and list of parameters in XML
    e. Response Structure - HTTP response message that contains the single return value. Unless an error has occurred with the function, a HTTP 200 OK message is always sent with the return value.
3. **RMI**
    a. Java Remote Method Invocation
    b. Natively supported by Java
    c. Normally requires additional ports to be opened.
    d. The major disadvantage is that RMI is proprietary to Java; it is not interoperable with other languages. If there's a homogeneous environment using purely Java, than RMI would be the way to go since the overhead of doing data translation to a common interchange format (such as XML) is removed.

e. It should be noted that notwithstanding the built-in mechanism for overcoming firewalls, RMI suffers a significant performance degradation imposed by HTTP tunneling. There are other disadvantages to using HTTP tunneling too. For instance, your RMI application will no longer be able to multiplex JRMP calls on a single connection, since it would now follow a discrete request/response protocol. Additionally, using the java-rmi.cgi script exposes a fairly large security loophole on your server machine, as the script can now redirect any incoming request to any port, completely bypassing your firewall system. Developers should also note that using HTTP tunneling precludes RMI applications from using callbacks, which in itself could be a major design constraint.

4. **Sockets**
   - **a.** Sockets provide performant low-level communications between devices.
   - **b.** The downside is that programming sockets is complex, requires a good understanding of the technology and requires a lot of work. Also, sockets are typically opened on ports other than the standard HTTP ports and therefore programs that rely on sockets must have the firewall adjusted to allow them to work.

Analysis Of Root Technology Options

Using the above technologies, we could certainly create any application we desired, however a lot of work would be involved in creating the plumbing and communications. The new Rich Thin Client framework space is aimed at simplifying the development of distributed client applications and typically use one or more of the above technologies to achieve this goal.