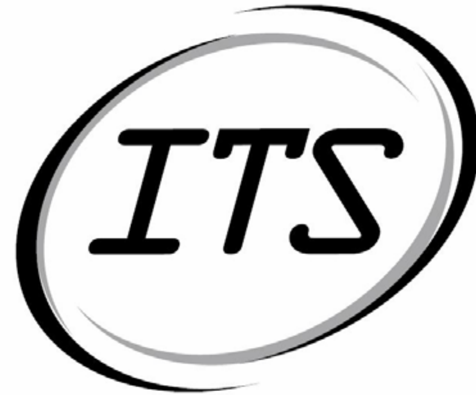
R·I·T



ITS Graphical Report Maker

Technology Comparison for Design

05 April 04

Team JACT Software
RIT Software Engineering Department

Version 1.1.0

Revision History

Revision	Date	Author(s)	Section	Comments/Changes
1.0.0	3 April 04	J. Myers	All	Initial Revision
1.1.0	5 April 04	All	All	Grammar and fixes

Table of Contents

1. INTRODUCTION 1

2. SCOPE 1

3. OVERVIEW 1

4. TRADEOFFS 1

5. RISKS 1

6. CONCLUSIONS 1

1. Introduction

The purpose of this document is to present technical information regarding the architectures investigated when deciding which implementation to utilize on the Graphical Report Maker Project. Each explored architecture has tradeoffs for each listed and a final determination is made from these. There are a total of 6 architectures discussed in this document: Standalone Java Program, Java Thin-Client using Web Services, Java Thin-Client using RMI, Java Smart-Client, JavaServer Pages, and Java Applets.

2. Scope

This document is intended to be used by the team designing and implementing the final system and for the customer as a reference for future projects. It is more than likely that the design considerations investigated for this project will reappear in the future; therefore, documenting these tradeoffs could lower costs in the future.

3. Overview

Six different architectures were chosen for investigation to find the optimal one for this project. A short description of each follows:

1. Standalone Java Program – A program residing solely on the user's computer and handles all interaction with the databases, the web servers, and persists the data on the machine that it is running on. This type of application is ideal if several users do not need to modify the same data and that single computer can be devoted to processing the data and graphs.
2. Java Thin-Client using Web Services – A program running on a client communicates to the server via Web Services. Essentially, the server has no idea what the client implementation is, as Web Services are language independent. All that it knows is that it has gotten a request to perform some function and does so accordingly. To enable this, the communication medium is generally through XML SOAP-based objects. This is the only way to make it portable across all the different types of implementations. This will require the client to do more error-checking and have more logic built into it, as it cannot communicate as often with the server.
3. Java Thin-Client using Remote Method Invocation (RMI) – A program running on a client that communicates to the server via Remote Method Invocation. This type of communication allows for many interactions with the server as the communication method is simply Java Objects; that is, no conversion in data has to occur if the client is also Java.
4. Java Smart-Client – Although not an industry accepted term, our definition of a "Smart-Client" does all but the minimalist interactions with the server. For

our definition, the client performs all of the error-checking and logic for program control. In this scheme, the only functions of the server would be for persistency of the data and execution of the reports. Everything else would be left up to the client.

5. JavaServer Pages – The purpose of JavaServer Pages are to make the transfer of information to and from the client as minimal as possible. One modern way of doing so is through dynamic web pages. These simple HTML pages are created on-the-fly by the JavaServer (typically TomCat) and know how to process the information entered by the user interacting with the system.
6. Java Applet – A Java program distributed and executed within a standard web browser. These programs tend to be very small in size, but also limited in functionality. To prevent security violations, Applets intentionally cannot interact with the file system nor network resources as much as a full Java program can.

4. Tradeoffs

Technology	**Pros**	**Cons**
Standalone Java Program	• No network communication with a server required. • Less complex program means less of a chance of errors.	• Central persistency of data not available. • Standalone system required to execute reports. • Still requires external interaction with the database and web servers.
Java Thin-Client using Web Services	• Server can interact with any client that can utilize Web Services. • Communicates over standard HTTP. • Simple interface. • Data communicated via XML-SOAP, which can be persisted in that form.	• Less interaction between server and client can be performed. • Communication of data must be parsed at both ends into common Java Objects. • Error-Checking will need to reside on the client before communication can take place.

Java Thin-Client using RMI	• Server and client can interact in standard formats. • Interaction can be designed so that error-checking and logic can reside on the server.	• Language dependent communication requires Java client. • Non-Standard communication means persistency likely Java Serialization of Objects.
Java Smart-Client	• Less interaction between server and client for low-bandwidth situations. • Server has minimal functionality and less likely to fault.	• Client likely too large for distribution over the Internet. • Logic and error-checking not resident on server and changes are hard to propagate.
JavaServer Pages	• HTML pages require very low bandwidth. • Interaction is cross-platform requiring only an Internet browser. • No processing on client.	• Limited functionality in terms of graphical user interfaces. • Requires the installation of a separate JavaServer from the Web Server, like Tomcat.
Java Applet	• Small program. • Can be run within any standard web browser that supports Java.	• Limited interface abilities. • File System interaction restrictions that may inhibit the program from functioning correctly.

5. Risks

The main reason for doing an analysis of this type is to allow for our quality attributes, as defined in the Software Requirements Specification, to remain. Some of these quality attributes include: Maintainability, Extensibility, and Availability. Each of the types discussed in the table above contribute in some way to these attributes. It is important to choose an architecture that has the best proportions of each of them before proceeding with the remainder of design and implementation.

Following this decision, a risk could emerge if the architecture chosen does not continue to meet the quality attributes originally defined for the project. Understanding and adapting any design that emerges to ensure that no quality attributes are lost in this process can mitigate this risk.

6. Conclusions

For the ITS Graphical Report Maker System, the Java Thin-Client using RMI Technology was chosen as the predominate client/server architecture. The reasons for this decision are:

1. Allows for smaller client program to be downloaded to the user's computer as the logic and processing for the UI can remain on the GRM server.
2. No parsing of data on either side as Java Objects are the communication medium.
3. Java Objects that are passed back and forth can be serialized directly on the server to persist the Element data.
4. Server execution of reports can be performed through any Java program with RMI.

It should be noted that when deciding on an architecture, it came down to either Java Thin-Client with Web Services and Java Thin-Client with RMI. The main point that moved the decision to Remote Method Invocation was the fact that more logic needed to reside in the client than the group felt comfortable with. More logic in the client meant that the computer would have to do the processing before communicating with the server, and this was a tradeoff against performance. Therefore, RMI was used so that this logic and processing could be passed-off to the server.