Application Architecture

Related terms:

Internet of Things, Intranets, Enterprise Architecture, Data Architecture, Application Component, Enterprise Application, Reference Architecture

View all Topics

Installing Software

Kelly C. Bourne, in Application Administrators Handbook, 2014

8.2.3 Clustered or load-balanced environments

Application architectures include either clustering or load balancing for two specific purposes:

- To maximize availability of the application by having multiple servers available. If one server hangs up or dies, then the other server(s) can continue processing and the application remains available to the users.
- To increase processing throughput by distributing the load across multiple servers. The way the load is distributed between servers can be simple or complex. The most basic method is called round-robin. In it the first user request is assigned to server number one. The second request is assigned to server number two, etc. More complex algorithms take into account the complexity of each request. In this arrangement, one server might be handling a single complex request while another server is handling multiple simple requests.

Figure 8.3 displays an environment that includes clustered servers. This example shows a cluster of three servers supporting the application and a cluster of two servers supporting the reporting function. The number of hardware servers comprising the cluster can vary, but the concept is the same: multiple servers being used to enhance availability or performance.

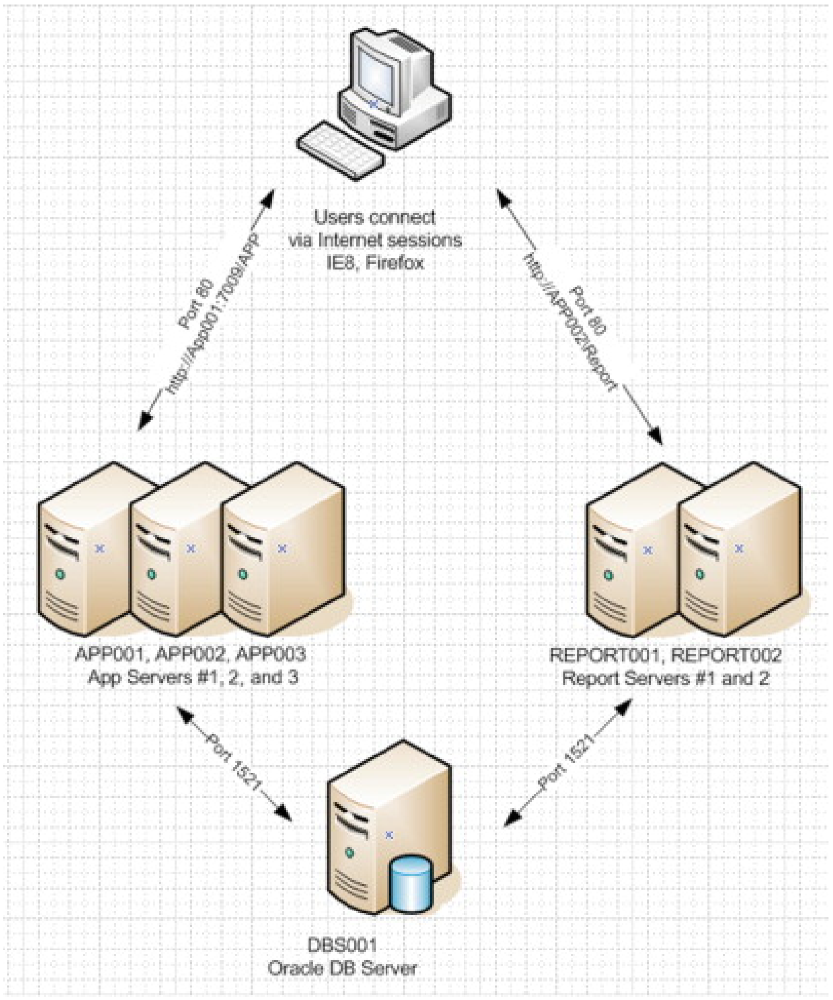


Figure 8.3. Diagram of an environment with clustered servers.

If your application is employing either clustering or load balancing, then you need to understand exactly what benefits you're getting out of it. The objective might be to enhance system availability. It might be to enhance throughput. Depending on the application's design, it might or might not be possible to be getting both benefits. Read the manual or consult with the vendor to make sure you understand what the architecture is providing and more importantly what it isn't providing.

If your environment has multiple servers for either clustering or load balancing, then the installation process will very likely be more complicated than installation is for a single server environment. You will almost certainly have to install the software separately on every server. There will likely be a very specific order in which the servers need to be built. If a "primary" server exists, then it will likely need to be the first one that is built. The "secondary" server(s) probably will have the software installed on them once the primary server is completely operational. Be certain that you know what the process is and include all the necessary steps and sequences in your project plan.

> Read full chapter

Federal Agency Case Study

E.d. Seidewitz, William Ulrich, in Information Systems Transformation, 2010

Application Architecture Assessment

The application architecture assessment expanded on the enterprise assessment work in several ways. We provided a narrative description for each Core System module or application and grouped that module by major functional category. We provided a summary of the module or application as well as a description of the functionality supported by that module or application.

We additionally provided a separate data and execution flow for the main capabilities supported by the Core System. These capabilities included cost allocation, asset management, billing, and accounts receivable. Our analysis of module functionality was based on prior documentation that we discovered, system flows and verification of the correctness through subject matter expert reviews, and the integrity of the information we had collected.

We included an analysis of the functional redundancy across the Core System and the other systems we analyzed. In addition, we highlighted where functional redundancy and fragmentation existed within the Core System. Finally, we created summary level descriptions for all interfacing systems and systems that were directly related to these interfacing systems. The intent was to provide a guide to implementation teams on both sides of the interface regarding the impact on systems that directly or indirectly interfaced with the Core System.

> Read full chapter

Focusing XML and DM on Enterprise Application Integration (EAI)

Peter Aiken, David Allen, in XML in Data Management, 2004

EAI Components

One of the most common factors that determines if IT projects succeed or fail is whether or not the project takes into account all of the different facets of what needs to be done. For example, some projects take a strictly technical approach, ignoring the data-architecture portion of the project, and end up with a corresponding set of drawbacks. In other situations, project teams become enamored of specific technical solutions and miss the larger context that the solution needs to operate within.

Figure 8.5 displays the three critical components that need to be considered in order to deliver successful EAI solutions. While it is possible to deliver a solution without considering all three aspects, such systems often are delivered either over budget, past deadline, with less than expected functionality, or all of the above.

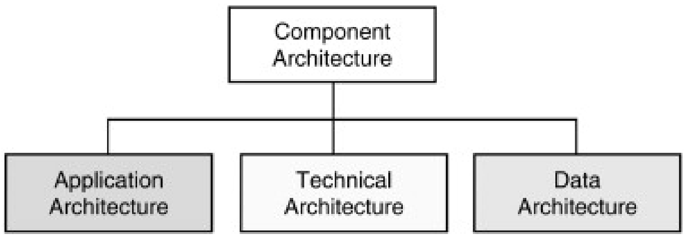


Figure 8.5. Three critical components required to deliver EAI systems.

1. Application Architecture**.** The Application Architecture component of an EAI solution takes into account the "style" and approach of the solution. Will the solution be client-server based? Will it be built in an object-oriented way? With a fourth-generation programming language? How will the application be put together, and how does it logically attack the core issues of the problem it is trying to solve?
2. **The Technical Architecture.** This component is the one that project teams often spend too much time considering. The technical architecture deals with questions of what operating systems, hardware, platform, and network facilities will be used. It also encompasses a host of other non-software, non—data-related system and infrastructure questions. Technical facilities are the bricks that are used to build larger systems. While care should be taken to make sure that the right bricks are selected, the form and characteristics of the brick definitely should not dictate how the finished building or solution will be created.
3. **The Data Architecture.** This component is typically the one that too little time is spent considering. The data architecture component of an EAI solution addresses issues of which data structures will be used, which access methods will be employed, and how data is coordinated throughout the system.

Throughout this chapter, XML is discussed particularly as it applies to the data architecture component of EAI above. Traditionally, when building systems that were focused around the use of toolkits like DCOM and CORBA, the most focus was put on the application architecture component. XML encourages architects to take a hard look at their data architecture through the facilities that XML provides, and to spend the appropriate amount of time in each area. For an EAI project to be a true success, focus is needed in each of the areas, rather than skewing analysis to one particular part of the picture.

The difficulty here is that focusing on three different areas requires three different sets of expertise. IT and systems infrastructure people are best suited to deal with the technical architecture; programmers and program architects are best equipped for the application architecture component; while data managers are in the best position to take care of the data architecture. Given these three areas of expertise, data managers still seem to be the ones who end up leading these efforts, for several reasons. Data managers typically have a higher-level view of the business and its technology needs, since they coordinate data resources in many different places. They also are often better suited to understanding the issues of integration that frequently come down to sticky data architecture questions.

These components come together to form the basis for technically effective EAI—in other words, an EAI system that delivers what was promised in the way of system integration. There are other definitions of a successful EAI system, however, particularly from the business and financial perspectives. In the next section, we will take a look at the motivation for why businesses want to approach EAI in the first place, and what financial incentives and pitfalls surround its use.

> Read full chapter

Software Application Design

Jean-Louis Boulanger, in Certifiable Software Applications 3, 2018

11.1 Introduction

The software application architecture phase (Chapter 9) allowed us to identify components, interfaces between these components and interfaces with the environment. As we have explained, the architecture phase could be built upon at least two levels (SwAD and SwCD), but if needed more decomposition levels should be employed. It is now necessary to define the content of each component. This stage involves the identification of the services performed by these components and the definition of the associated algorithms.

The safety implementation principles (information encoding, data redundancy, etc.) for a software application must be taken into account within the context of the design.

In order to prepare the coding phase, the design phase considers the type of programming language that will be used. For example, when using the C ++ language, an object-oriented-based design must be implemented so as to avoid inconsistencies or issues during coding.

> Read full chapter

Introduction to Architecture-Driven Modernization

William Ulrich, in Information Systems Transformation, 2010

An architectural view of modernization

We have been discussing application, data, and technical architectures from an IT perspective and business architecture from a business perspective, but these architectures have a mutual dependence on one another. Pursuing a modernization strategy, therefore, must recognize these dependencies, otherwise, business and IT can get out of sync. One may be modified or deployed and the business cannot adapt or the systems and data cannot support change. It is important, therefore, to understand where and how these architectures interact.

The technical architecture is an abstract representation of the platforms, languages, and supporting technological structures that provide the "plumbing" that keeps the environment functioning. This includes SOA and related communication structures. The application architecture is an abstract representation of the applications, subsystems, services, and orchestration structures and their relationship to each other and to the data. The data architecture is an abstract representation of the data files, databases, and relationships to the application architecture. These three views create the IT architecture and modernization impacts each in unique ways.

Similarly, the business architecture has its own set of artifacts and abstractions. These include organization units, capabilities, process, semantics, and rules at the most basic level. Business and IT architectures live in separate yet related domains (commonly referred to as the 'enterprise architecture'). Figure 1.1 shows the business and IT domains of an organization. Business architecture describes the business domain while IT architecture describes the IT domain. The business architecture relies on the application and data architecture, which in turn relies on the technical architecture as shown in Figure 1.1.

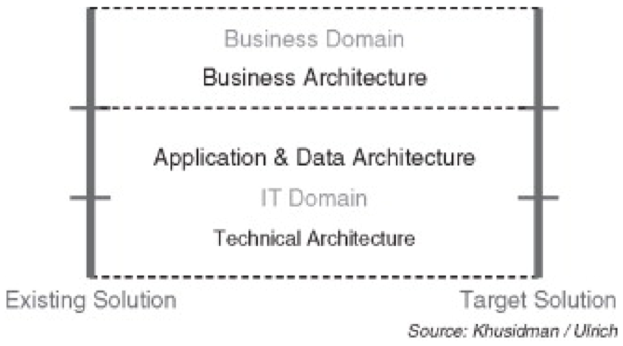


Figure 1.1. Business versus IT architecture domains.25

Figure 1.1 also shows the existing (as-is) solution to the left and the target (to-be) solution to the right. As business and IT architectures evolve, the existing solution morphs into the target solution. The challenge facing most enterprises is that the business environment continues to evolve beyond IT's ability to keep application and data architecture synchronized with business requirements. Modernization disciplines allow IT to evolve the as-is IT architecture in a way that keeps it synchronized with business changes and requirements.

Figure 1.2 illustrates how certain business and technical factors drive movement from the as-is to the to-be business and IT architectures. Business architecture is driven by the business and this should be the overriding consideration as executives consider how IT architecture must evolve. Business requirements have a direct bearing on the evolution of data and applications. This means that IT must respond to an evolving business ecosystem and drive corresponding changes to data and application architectures. Understanding IT requirements means understanding existing and target business architectures and mapping the impacts of the changing business to the as-is IT architecture. IT can then identify the modernization plan to move from the existing to the target IT architecture.

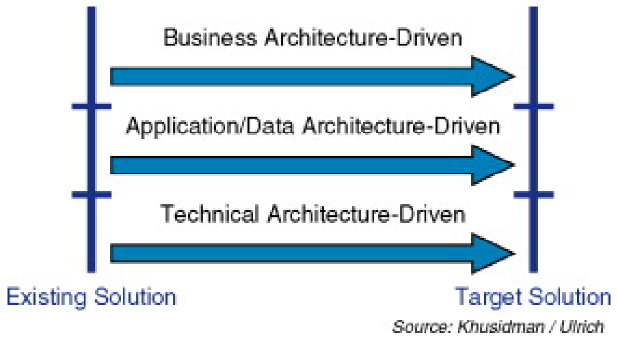
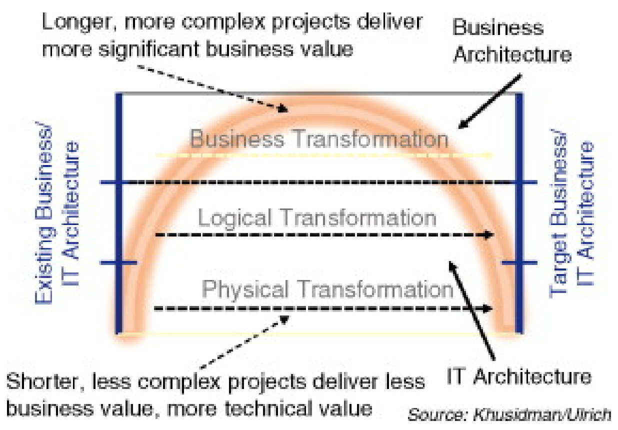

Business Architecture-Driven
Application/Data Architecture-Driven
Technical Architecture-Driven
Existing Solution
Target Solution
Source: Khusidman / Ulrich

Figure 1.2. Modernization drivers and trajectories.25

Figure 1.3 shows that there are many paths to moving from an as-is to the to-be business and IT architectures. One common path followed by IT is to focus on the technical architecture. This physical transformation shown in Figure 1.3 represents a “lift and shift” concept where the technical architecture is transformed and data and application architectures remain largely intact. While the cost of this approach is lower and project duration shorter, there is almost no impact or value to the business.


Longer, more complex projects deliver more significant business value
Business Architecture
Existing Business/ IT Architecture
Business Transformation
Logical Transformation
Physical Transformation
Target Business/ IT Architecture
Shorter, less complex projects deliver less business value, more technical value
IT Architecture
Source: Khusidman/Ulrich
Figure 1.3. Modernization domain model.25

A modernization effort that seeks to provide value to the business, on the other hand, would need to change the application and data architecture, which in turn would rely on an analysis of requirements stemming from shifts to the business architecture. These types of projects are of a longer duration, require more investment, and deliver significantly more value to the business. In Figure 1.3 this means traveling up the "horseshoe" into the application and data (i.e. logical) architecture and into the business architecture.

It is important to understand that IT has been selling technical architecture transformation via concepts like SOA and model-driven architecture (MDA) to executives, and management has invested millions of dollars in pursuit of these architectures. The impact on business, however, has been inconsequential. To really deliver business value, SOA and MDA must be driven by business requirements that drive application and data architecture transformation. Unfortunately, this has not been the case in most IT organizations.

If executives believe that IT will deliver significant business value through a technical or physical route to SOA or MDA, they will be shocked to find out that their investments have been ill-advised and poorly spent. Therefore, when planning a modernization effort, consider the concepts in Figure 1.3 and the impact to the overall architectural environment from a business perspective.

There are a wide variety of disciplines that fall under the modernization umbrella and are used to enable various modernization scenarios. Modernization disciplines, which are a collection of tool-enabled tasks, can be summarized and categorized into three major categories: assessment, refactoring, and transformation.

Disciplines defined within each of these categories are mixed and matched together in a way that creates a specialized solution for a given business and IT situation. These solutions will be generalized into project scenarios in Chapter 4. An overview of assessment, refactoring, and transformation phases of modernization provide the requisite background for planning teams that base their efforts around the scenarios and case studies that are presented in subsequent chapters.

> Read full chapter

Transaction Processing Application Architecture

Philip A. Bernstein, Eric Newcomer, in Principles of Transaction Processing (Second Edition), 2009

Service-Oriented Architecture

In addition to the multitier application architecture, application design methodologies play a role in the structure of TP applications. Service-oriented architecture (SOA) is one such design methodology, which was discussed in Chapter 1. In SOA, the designer identifies a service that a business provides for its customers and partners. The designer maps this business service to a software service, which is an operation. Typically, a set of related operations are grouped together in a service interface. Each operation in a service interface is implemented as a software component that can be invoked over a network by sending it a message. In SOA, operations are intended to be relatively independent of each other, so they can be assembled into applications in different combinations, connected by different message patterns.

In a TP system, a service can implement a transaction or a step within a transaction. That is, it can play the role of a request controller or transaction server. In either case, it is invoked by sending a message to the service. In this sense, the notion of a service is nicely aligned with multitier TP system architecture.

This alignment between SOA and TP depends only on the fact that SOA decomposes applications into independent services. It does not depend on the particular technology that is used to define service interfaces or to communicate between services, such as RPC or Web Service standards.

> Read full chapter

PowerBuilder/4GL Generator Modernization Pilot*

Philip H. Newcomb, ... Robert Couch, in Information Systems Transformation, 2010

New High-Level Application Architecture

In the new architecture, 4GL generated COBOL and PowerBuilder services will be implemented in automatically generated Java using standardized frameworks including Spring and Hibernate (see Figure 6.5).

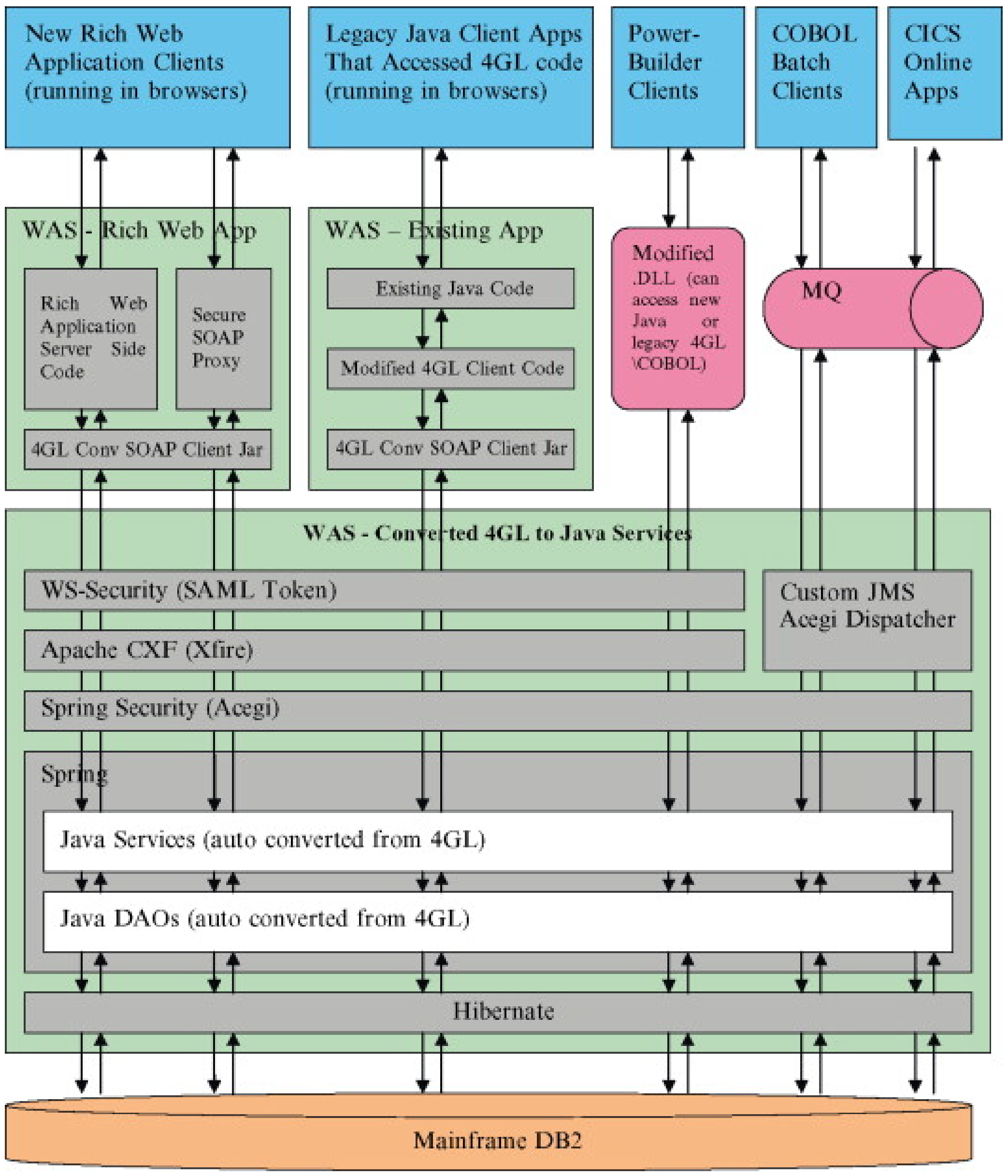

New Rich Web Application Clients (running in browsers)
Legacy Java Client Apps That Accessed 4GL code (running in browsers)
Power-Builder Clients
COBOL Batch Clients
CICS Online Apps
WAS - Rich Web App
Rich Web Application Server Side Code
Secure SOAP Proxy
4GL Conv SOAP Client Jar
WAS – Existing App
Existing Java Code
Modified 4GL Client Code
4GL Conv SOAP Client Jar
Modified .DLL (can access new Java or legacy 4GL \COBOL)
MQ
WAS - Converted 4GL to Java Services
WS-Security (SAML Token)
Custom JMS Acegi Dispatcher
Apache CXF (Xfire)
Spring Security (Acegi)
Spring
Java Services (auto converted from 4GL)
Java DAOs (auto converted from 4GL)
Hibernate
Mainframe DB2

Figure 6.5. New application architecture.

This new architecture functions as follows. For security purposes, the Spring code uses the Acegi framework, which isolates the code from how the authentication and authorization actually occurs. There are at least two types of Acegi interceptors, one for WS-Security and another to establish the credentials from incoming JMS/MQ messages. Finally, the Spring services are remotely enabled as Web services using the Apache CXF framework (which is the new version of XFire).

Note that the converted services are exposed as standard Web services secured by WS-Security using the SAML Token authentication. This means that they are accessible to any Web service client or framework (e.g., Axis1, Axis2, JAXB-WS/Glassfish, etc.). The converted service provides a prepackaged Apache CXF-based Web service client that exposes the Web service client interface as a standard Spring service bean. From the client application code's perspective, the application appears just like any other Spring bean accessible via ApplicationContext.getBean lookups or via any framework that supports Spring dependency injection.

Rich Web applications (i.e., currently in PowerBuilder) use the asynchronous JavaScript and XML (AJAX) communication facility. Browser controls with the need for direct calls to 4GL services are routed through a custom "Secure SOAP Proxy" that uses the standard in-house Web application security context to ensure that HTTP sessions are authenticated. SAML token form of WS-Security is used with application-specific certificates (similar to managed ID's today) rather than user-specific certificates.

The 4GL replacement Web services establish a trust relationship with any request signed with one of these application certificates to ensure that the user ID passed by that signed request is valid (if the request is tampered with by changing the user ID, it will invalidate the digital signature).

This results in the following application architecture, which drives the conversion process mapping into target-side components. Note that we are also describing the transition state where applicable.

- Rich Web Applications: Uses a Java-based server-side framework with one or more AJAX-enabled widget sets (e.g., Dojo). For full page renders, the server-side code invokes the backend converted code via a web service client jar. For dynamic AJAX behavior, the browser sends a request to a "Secure SOAP Proxy" that uses the normal HTTP session-based security to ensure the user is authenticated and then wraps the Web service request with SAML signed Web service request to the converted Java services (using the converted 4GL code client jar).
- Preexisting Java Web Applications: These applications use a replacement of their old Java CICS 4GL COBOL client library that makes calls via the new SAML signed Web services instead of the old CICS approach. The existing API uses static methods and exposes some CICS level details that may require some manual migration efforts beyond dropping in a replacement library.
- PowerBuilder Clients: Via a data bridge, these applications use a modified DLL that has both the old capability to invoke the mainframe CICS COBOL 4GL services that interact with DB2 database as well as the capability to invoke the new SAML signed Web services. This can be configured on a call-by-call basis.

(Note: This data bridge is a temporary solution until all of the PowerBuilder applications are migrated.)

- CICS Online Applications: These applications will invoke the converted CICS COBOL 4GL services directly and submit requests via secure JMS/MQ queues. The Java services container listens for requests, dispatches them to the converted Java services, and puts the response on a response queue.
- COBOL Batch Programs: The architecture will include a separately-generated set of COBOL 4GL code that interacts with DB2 on a case-by-case basis. They may be able to use the JMS/MQ-based approach used by the COBOL online applications. In some situations conversion of a COBOL-based batch program to Java will be necessary.

New Application Environment Components Overview

The target Java environment for the converted 4GL framework is based on the company's internal Java starter app platform. This starter app framework is an internally-developed compilation of preferred open source frameworks and ideal techniques for utilizing those frameworks. Some of the key open-source frameworks that make it up are XDoclet, Struts 1.x, Spring, Hibernate, Derby, JUnit, Apache Commons, Log4J, XFire for Web service support, and Acegi for security.

Several internal frameworks are incorporated into the platform as well. The "starter app" platform is the preferred environment for all current and future Java development efforts. As such, the Java code converter makes every effort to generate target code that operates within this starter app environment. The principle components in the internal Java/JEE framework are described briefly in the following list.

- Spring Framework: An open source application framework for the Java platform and the .NET framework.
- Acegi Security System for Spring: Spring Security, using Acegi, provides powerful and flexible security solutions for enterprise applications developed using the Spring Framework.
- Apache Struts: This is an open source Web application for developing Java EE Web applications that uses and extends the Java Servlet API to encourage developers to adopt a model-view-controller (MVC) architecture.
- Apache Commons: This is a set of Java components that have minimal dependencies on other software libraries so that components can be deployed easily. Common components keep their interfaces as stable as possible.
- Dojo Toolkit: This is an open source, modular JavaScript library (i.e., JavaScript toolkit) designed to ease the rapid development of cross platform, JavaScript/AJAX-based applications, and Web sites. AJAX is a group of interrelated Web development techniques used to create interactive Web applications or rich Internet applications. With Ajax, Web applications can retrieve data

from the server asynchronously in the background without interfering with the display and behavior of the existing page. ■

Hibernate: An object-relational mapping (ORM) library for the Java language that provides a framework for mapping an object-oriented domain model to a traditional relational database. Hibernate solves object-relational impedance mismatch problems by replacing direct persistence-related database accesses with high-level object handling functions. ■

JUnit: This is a unit testing framework for the Java programming language. ■

Log4J: This is a Java-based logging utility. ■

Spring Data Access Object (DAO; specifically the Hibernate and JDBC implementations): This access object provides an abstract interface to a *database* or persistence mechanism and provides data access operations, without exposing details of the database. This isolation separates the concerns of which data accesses the application needs. ■

Xdoclet: This is an open source code generation library that enables attribute-oriented programming for Java via insertion of special JavaDoc tags. It comes with a library of predefined tags that simplify coding for various technologies. ■

XFire: A next-generation Java SOAP framework that makes service-oriented development approachable through an easy-to-use API and its support for standards. It also performs well because it is built on a low memory StAX-based model**.**

> Read full chapter

GPU programming

Gerassimos Barlas, in Multicore and GPU Programming, 2015

6.9 Debugging CUDA programs

Nvidia provides two tools for debugging CUDA applications:

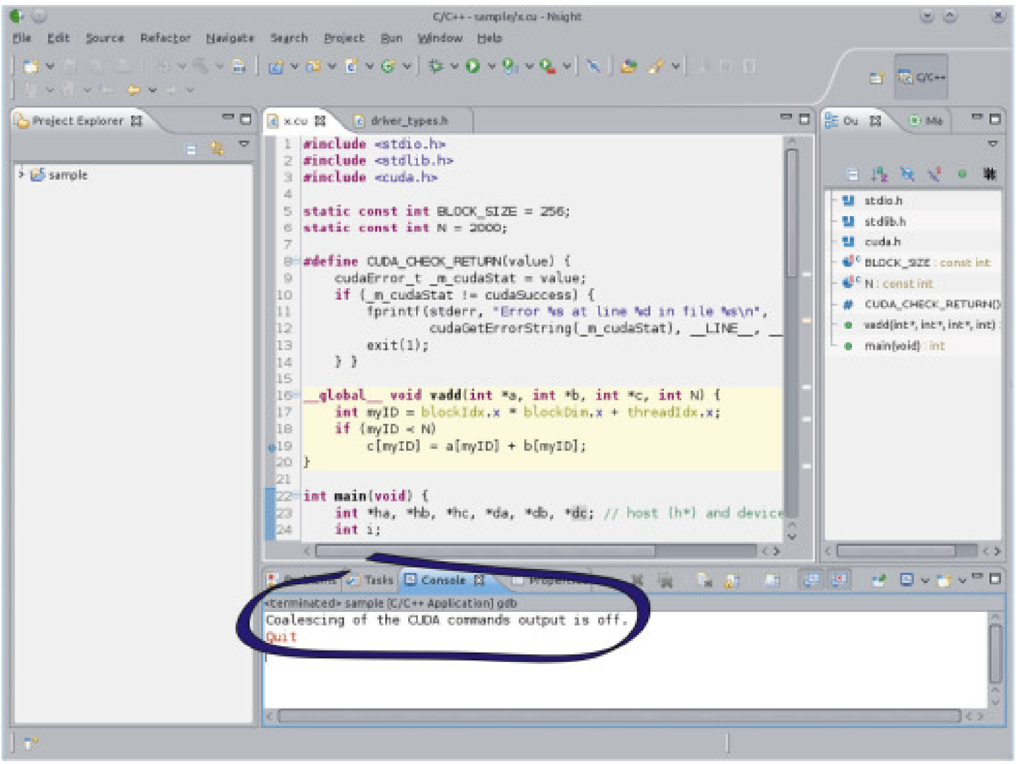
- The Parallel NSight Eclipse-based IDE (nsight) provides integrated GUI-based debugging for Linux and Mac OS X platforms. Under Windows, NSight integrates with Visual Studio.
- CUDA-GDB (cuda-gdb) is a command-line debugger based on GNU’sdebugger (gdb) and it is available for Linux and Mac OS X platforms. Putting aside the awkward interface, CUDA-GDB offers all the conveniences of a modern debugger, such as single-step execution, breakpoints in kernel code,

inspection of individual threads, and so on. Additionally, because it shares the same commands with the GNU debugger, it can be used in tandem with front ends such as DDD, eliminating the need for someone to memorize its commands. For example:$ ddd —debugger cuda – gdb myCudaProgram

Actually, in Linux and Mac OS X platforms, NSight is just a front end for CUDA-GDB.

Debugging CUDA programs suffers from a major drawback that stems from the peculiarity of using a display device for computations: It requires two GPUs, one for running the application under development and one for regular display. The former GPU may be hosted in the same machine or a remote one. Most users would find themselves under the second scenario, i.e., using a remote machine (e.g., a shared server) for debugging purposes. Those unfortunate enough to not have access to a second GPU will have to rely on printf() for probing the state of CUDA threads during execution.

In this section we explore how Parallel NSight can be used for remote debugging, following one of the most probable scenarios that one can encounter. A screenshot of the Eclipse-based Parallel NSight platform is shown in Figure 6.19.

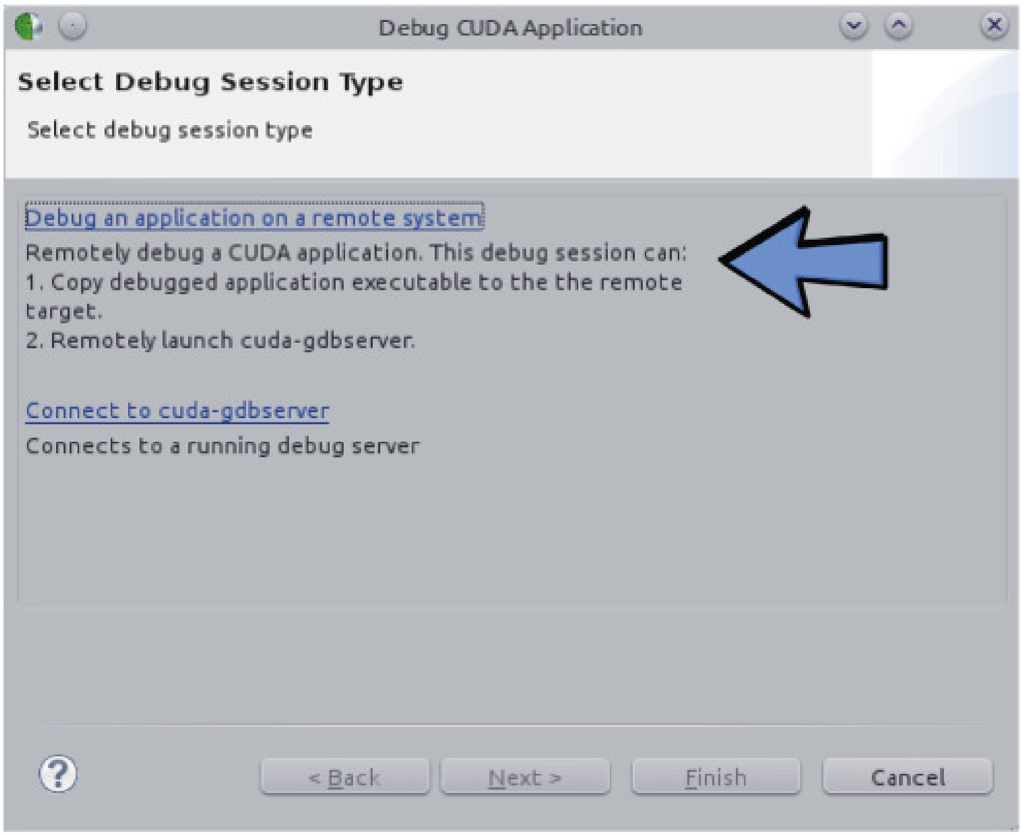


```
#include <stdio.h>
#include <stdlib.h>
#include <cuda.h>

static const int BLOCK_SIZE = 256;
static const int N = 2000;

#define CUDA_CHECK_RETURN(value) {
    cudaError_t _m_cudaStat = value;
    if (_m_cudaStat != cudaSuccess) {
        fprintf(stderr, "Error %s at line %d in file %s\n",
                cudaGetErrorString(_m_cudaStat), __LINE__, _
        exit(1);
    } }

__global__ void vadd(int *a, int *b, int *c, int N) {
    int myID = blockIdx.x * blockDim.x + threadIdx.x;
    if (myID < N)
        c[myID] = a[myID] + b[myID];
}

int main(void) {
    int *ha, *hb, *hc, *da, *db, *dc; // host (h*) and device
    int i;
```

```
<terminated> sample [C/C++ Application] gdb
Coalescing of the CUDA commands output is off.
Quit
```

Figure 6.19. A screenshot of the Eclipse-based Parallel NSight platform running under Linux. Thehighlighted Console pane error message was generated by trying to debug the application locally, without a second GPU.

A remote debugging session has to be initially configured by selecting the Run → Debug Remote Application... menu option. In response, the dialog box in Figure6.20 will appear. The remote debugging is conducted by having the application run under the supervision of a cuda-gdbserver process. Unless the remote system is already running cuda-gdbserver, the first option should be selected in Figure 6.20.



Select Debug Session Type

Select debug session type

Debug an application on a remote system

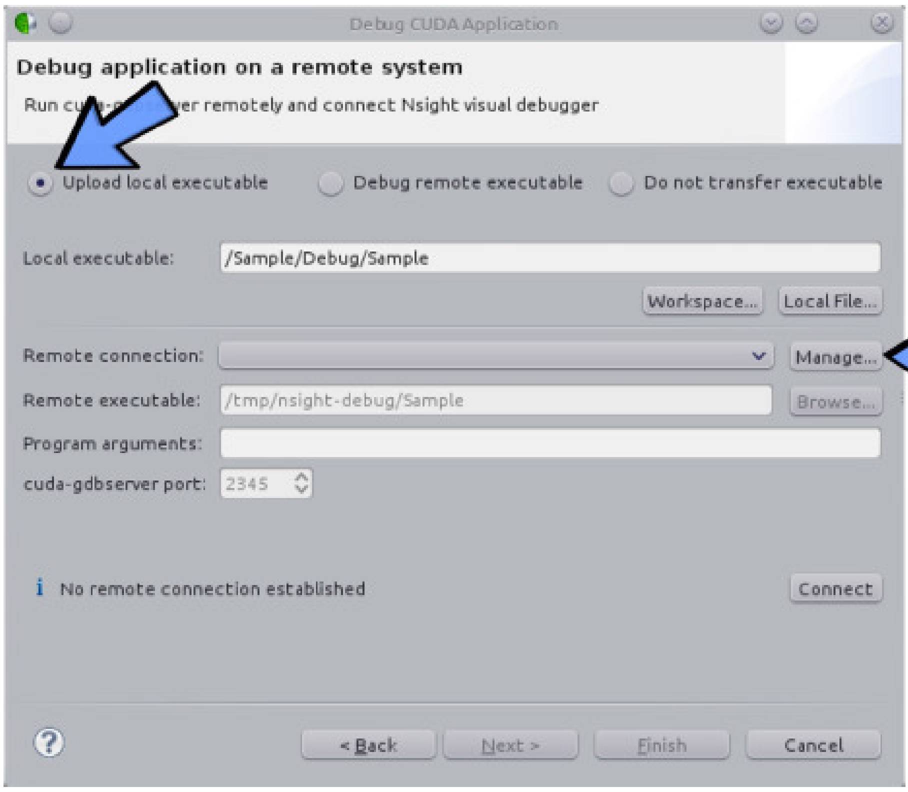
Remotely debug a CUDA application. This debug session can:
1. Copy debugged application executable to the the remote target.
2. Remotely launch cuda-gdbserver.

Connect to cuda-gdbserver

Connects to a running debug server

Figure 6.20. Parallel NSight dialog box for selecting the type of remote debug session.

The next dialog box, which is shown in Figure 6.21, controls how the remote system will get the application. Unless a common filesystem, e.g., an NFS volume, is being used, this has to be done by uploading the new binary. If the remote system is being used for the first time, pressing "Manage" allows the user to specify the connection details.


Debug CUDA Application

Debug application on a remote system

Run cuda-gdbserver remotely and connect Nsight visual debugger

● Upload local executable ○ Debug remote executable ○ Do not transfer executable

Local executable: /Sample/Debug/Sample

Remote connection:

Remote executable: /tmp/nsight-debug/Sample

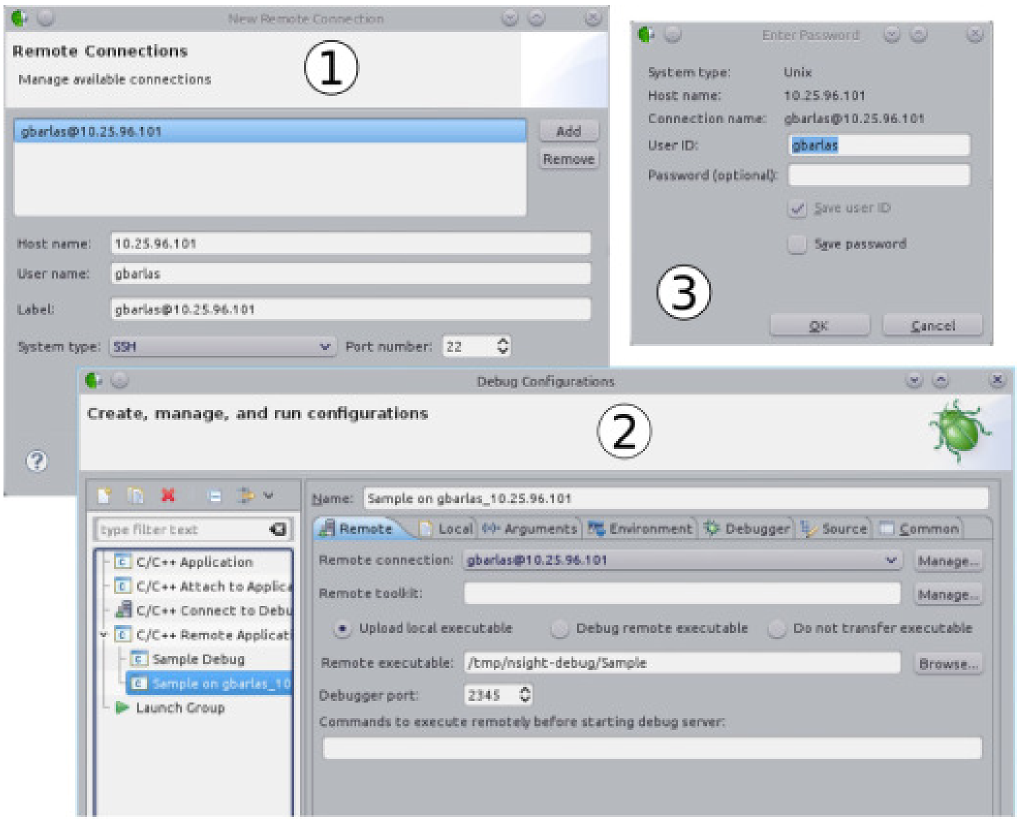
Program arguments:

cuda-gdbserver port: 2345

No remote connection established

Figure 6.21. Parallel NSight dialog box for selecting how the application will reach the remote system (typically by uploading). By pressing &quot;Manage&quot; a user can specify the remote system connection parameters.

The connection details of all the remote systems used for debugging, are collectively managed by the dialog shown in Figure 6.22.


New Remote Connection

Remote Connections

Manage available connections

gbarlas@10.25.96.101

Host name: 10.25.96.101

User name: gbarlas

Label: gbarlas@10.25.96.101

System type: SSH Port number: 22

Enter Password

System type: Unix

Host name: 10.25.96.101

Connection name: gbarlas@10.25.96.101

User ID: gbarlas

Password (optional):

✓ Save user ID

☐ Save password

Debug Configurations

Create, manage, and run configurations

Name: Sample on gbarlas_10.25.96.101

Remote connection: gbarlas@10.25.96.101

Remote toolkit:

● Upload local executable ○ Debug remote executable ○ Do not transfer executable

Remote executable: /tmp/nsight-debug/Sample

Debugger port: 2345

Commands to execute remotely before starting debug server:

Figure 6.22. Parallel NSight dialogs for (1) managing remote connection information and (2) selectingthe configuration for launching a new debugging session. The system may prompt the user for login credentials (3) depending on the selected connection protocol and system setup.

A new debugging session can commence by selecting the just created configuration from the "Run → Debug Configurations..." dialog window (see Figure 6.22).

Caution: It is a good idea to use the same CUDA SDK in both the local and remote hosts in order to avoid errors caused by missing libraries.

> Read full chapter

Models for Phase D

Philippe Desfray, Gilbert Raymond, in Modeling Enterprise Architecture with TOGAF, 2014

Abstract

Technology architecture associates application components from application architecture with technology components representing software and hardware components. Its components are generally acquired in the marketplace and can be assembled and configured to constitute the enterprise's technological infrastructure. Technology architecture provides a more concrete view of the way in which application components will be realized and deployed. It enables the migration problems that can arise between the different steps of the IS evolution path to be studied earlier. It provides a more precise means of evaluating responses to constraints (nonfunctional requirements) concerning the IS, notably by estimating hardware and network sizing needs or by setting up server or storage redundancy. Technology architecture concentrates on logistical and location problems related to hardware location, IS management capabilities, and the sites where the different parts of the IS are used. Technology architecture also ensures the delivered application components work together, confirming that the required business integration is supported.

> Read full chapter

Data Quality and MDM

David Loshin, in Master Data Management, 2009

5.2 Distribution, Diffusion, and Metadata

Because of the ways that diffused application architectures have evolved across different project teams and lines of business, it is likely that although only a relatively small number of core master *objects* (or more likely, object types) are used, there are going to be many different ways that these objects are named, modeled, represented, and stored. For example, any application that must manage contact information for individual customers will rely on a data model that maintains the customer's name, but one application may maintain the individual's full name, whereas others might break up the name into its first, middle, and last parts. Conceptually, these persistent models are storing the same *content*, but the slightest variance in representation prevents most computer systems from recognizing the similarity between record instances.

Even when different models are generally similar in structure, there might still be naming differences that would confuse most applications. For example, the unique identifier assigned to a customer is defined as a 10-character numeric string, padded with zeros out to the left, and that format is used consistently across different applications. Yet even when the format and rules are identical, different names, such as CUST_NUM versus CUSTOMER_NUMBER versus CUST_ID, may still confuse the ability to use the identifier as a foreign key among the different data sets. Alternatively, the same data element concept may be represented with slight variations, manifested in data types and lengths. What may be represented as a numeric field in one table may be alphanumeric in another, or similarly named attributes may be of slightly different lengths.

Ultimately, to consolidate data with a high degree of confidence in its quality, the processes must have an aligned view of what data objects are to be consolidated and what those objects look like in their individual instantiations. Therefore, an important process supporting the data quality objectives of an MDM program involves collecting the information to populate a metadata inventory. Not only is this metadata inventory a resource to be used for identification of key data entities and critical data elements, it also is used to standardize the definitions of data elements and connect those definitions to authoritative sources, harmonizing the variances between data element representations as well as identifying master data objects and sources. Collecting and managing the various kinds of master metadata is discussed in Chapter 6.

> Read full chapter