Architectures and Technologies for Enterprise Application Integration

Ian Gorton
National ICT Australia
Ian.gorton@nicta.com.au

Anna Liu
Microsoft Australia
annali@microsoft.com

Abstract

Architects are faced with the problem of building enterprise scale information systems, with streamlined, automated internal business processes and web-enabled business functions, all across multiple legacy applications. The underlying architectures for such systems are embodied in a range of diverse products known as Enterprise Application Integration (EAI) technologies. In this tutorial, we highlight some of the major problems, approaches and issues in designing EAI architectures and selecting appropriate supporting technology. The tutorial presents a range of the common architectural patterns frequently used for EAI applications. It also explains Service Oriented Architectures as the current best practice architectural framework for EAI. It then describes the state-or-the-art in EAI technologies that support these architectural styles, and discusses some of the key design trade-offs involved when selecting an appropriate integration technology (including buy versus build decisions).

1. Introduction

Application integration is concerned with the ease with which a software component can be usefully incorporated in to a broader application context. The value and lifetime of an application or component can frequently be greatly increased if its functionality or data can be used in ways that the designer did not originally anticipate. The most widespread strategies for providing interoperability are through data integration or providing an application programming interface (API).

Data integration involves storing the data an application manipulates in ways that other applications can easily access. This may be as simple as using a standard relational database for data storage, or perhaps implementing mechanisms to extract the data into a known format such as XML or a comma-separated text file that other applications can ingest.

With data integration, the ways in which the data is used (or abused) by other applications is pretty much out of control of original data owner. This is because data integrity and business rules imposed by the application logic are by-passed. The alternative is for interoperability to be achieved through an API. The raw data the application owns is hidden behind a set of functions that facilitate controlled external access to the data. In this manner, business rules and security can be enforced in the API implementation. The only way to access the data and interoperate with the application is by using the supplied API.

Integration is such a common problem that technological solutions for integration have spawned their own segment of the software market, known as Enterprise Application Integration (EAI). Numerous software companies have developed large, complex technologies specifically aimed at easing the problems that are inherent in application integration, especially in large enterprises with 10's or 100's of different legacy applications.

This tutorial is about architectures and technologies for EAI. It aims to convey to attendees the problems solved by EAI technologies, describe common architectural approaches for EAI solutions, and discuss the strengths and weaknesses of a range of EAI technologies.

2. EAI Technologies

Architects must make design decisions early in a project lifecycle. Many of these are difficult, if not impossible, to validate and test until parts of the system are actually built.

Due to the difficulty of validating early design decisions, architects sensibly rely on tried and tested approaches for solving certain classes of problems. This is one of the great values of architectural patterns. They enable architects to reduce risk by leveraging successful designs with known engineering attributes.

Proceedings of the 26th International Conference on Software Engineering (ICSE'04)
0270-5257/04 $20.00 © 2004 **IEEE**

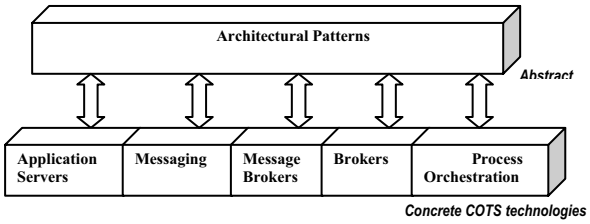

Figure 0 **Mapping between logical architectural patterns and concrete technologies**

Patterns are an abstract representation of an architecture, in the sense that they can be realized in multiple concrete forms. The publish-subscribe architecture pattern describes an abstract mechanism for loosely coupled integration and many-to-many communications between publishers of messages and subscribers who wish to receive messages. It doesn't however specify how publications and subscriptions are managed, what communication protocols are used, what types of messages can be sent, and so on. These are all considered implementation details.

Widely utilized architectural patterns for EAI are supported in a variety of commercial off-the-shelf (COTS) technologies. If a design calls for publish-subscribe messaging, or a message broker, or a three-tier architecture, then the choices of available technology are many and varied indeed. This is an example of software technologies providing reusable, application-independent software infrastructures that implement proven architectural approaches.

As Figure 1 depicts, several classes of COTS technologies are used in practice to provide packaged implementations of architectural patterns for use in EAI. Within each class, competing commercial and open source products exist. Although these products are superficially similar, they will have differing feature sets, be implemented differently and have varying constraints on their use.

Architects are somewhat simultaneously blessed and cursed with this diversity of EAI product choice. Competition between product vendors drives innovation, better feature sets and implementations, and lower prices, but it also places a burden on the architect to select a product that has quality attributes that satisfy the application requirements. All applications are different in some ways, and there is rarely, if ever, a 'one-size-fits-all' product match. Different implementations have different sets of strengths and weaknesses and costs, and consequently will be better suited to some types of applications than others.

The difficulty for architects is in understanding these strengths and weaknesses early in the development cycle for an EAI project, and choosing an appropriate reification of the architectural patterns they need. Unfortunately, this is not an easy task, and the risks and costs associated with selecting an inappropriate technology are high. The industry is littered with poor choices and associated failed projects.

3. Tutorial Outline

The tutorial is divided in to six sections. These are briefly described below:

Introduction: What is EAI? What are the core components of an EAI technology? A classification of EAI technologies based on their various feature sets. Explain briefly some common architectural patterns used in EAI, eg publish-subscribe, real-time data synchronization, n-tier client server, business process automation. For each, explain the key challenges and their typical solutions and trade-offs.

Service Oriented Architectures for EAI: Describe Service Oriented Architectures (SOA), including the responsibilities of the layers in an n-tiered SOA architecture, role of a business process executor and role of Web Services.

Using J2EE for EAI: Show how J2EE fits in to an SOA, including role and integration with Web Services. Explain salient features of the Java Messaging Service, Java Connector Architecture and the J2EE application server, and why they are useful in an EAI context. Briefly mention proprietary value-add features (eg SonicMQ).

Using .NET for EAI: Show how .NET fits in to an SOA, including role integration with Web Services. Explain salient features of BizTalk, MSMQ, and why they are useful in an EAI context.

Web Services Technologies: Describes the BPEL4WS and WS-Transactions specifications, and supporting
technologies. Examples will show how workflows and process ochestrations can be described with BPEL4WS, and how long running transactions are supported with WS-Transactions.

J2EE, .NET, Web Services: Discuss the strengths and weaknesses of these various technologies.