

Overview



- **User interface-level EAI** is similar to application interface-level EAI in that both **make available business processes and data through an interface exposed by the packaged or custom application**
- **API** can be exploited as well-defined mechanisms that are built to **connect to some sort of resource**, such as an application server, middleware layer or database
- There are **three types of services** available to these interfaces: business service, data service and objects
- **Types of Interfaces:** full-service, limited-service and controlled interfaces

Method-Level EAI

Putu Wuri Handayani (putu.wuri@cs.ui.ac.id)



Learning Objectives

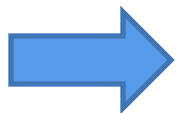


- Understand the basic concepts of method level EAI
- Identify when a company should use method level EAI
- Understand the concepts of framework
- Namely enabling technologies that can be used in method level EAI

Introduction

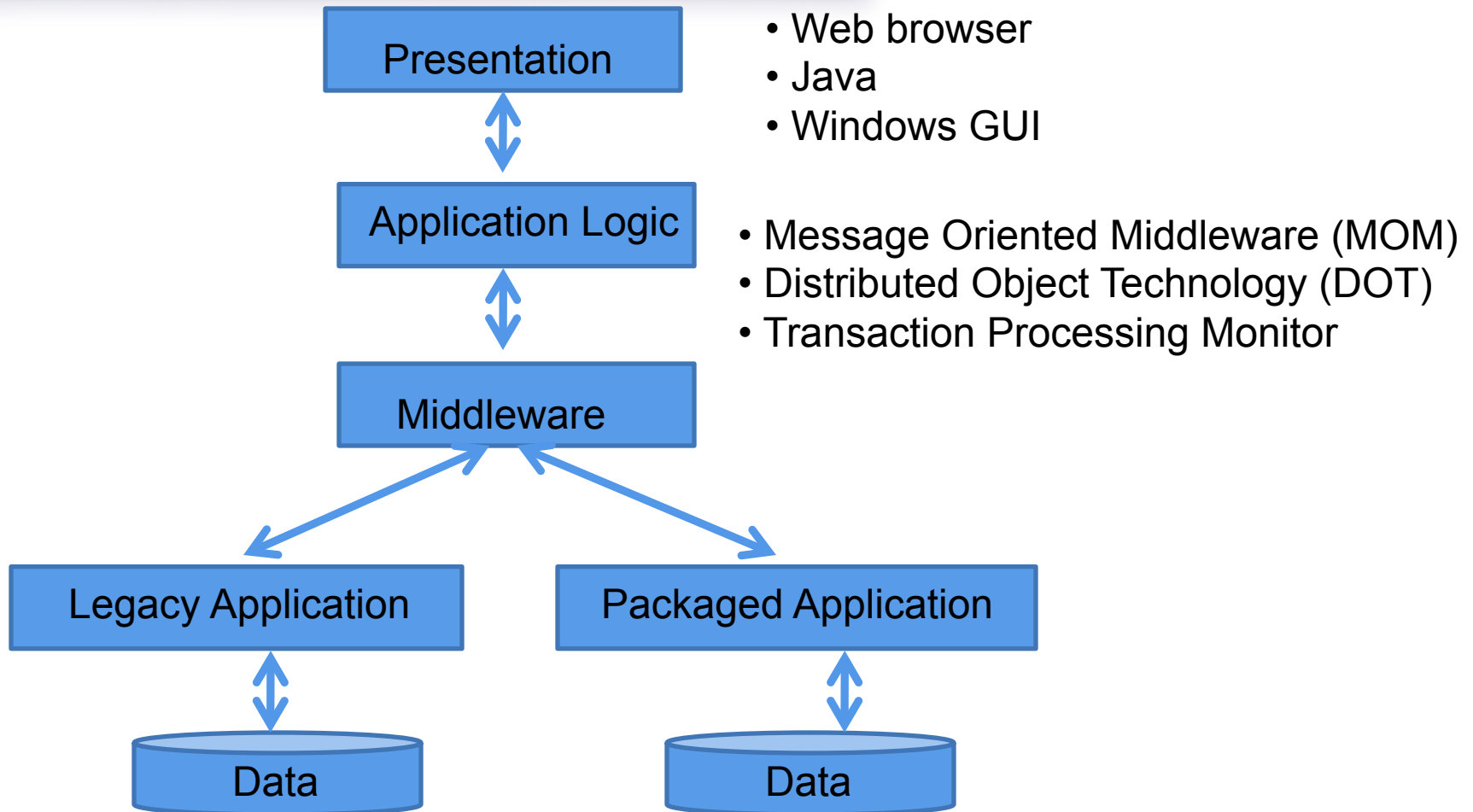


- Method-level EAI allows the enterprise to be integrated through the sharing common business logic or methods
 - This is accomplished either by defining methods that can be shared or by providing the infrastructure for such method sharing
- Methods may be shared either by hosting them on a central server or by accessing them between applications (e.g. distributed objects)



“Reuse” is an important concept in this context

Method Integration Model



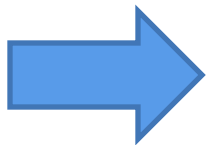
Method-Level EAI



- Requires that many, if not all, enterprise applications be changed in order to take advantage of the paradigm
 - Enterprises must clearly understand both its opportunities and its risks in assessing its value
- The goal of method-level EAI: composite applications
 - Applications bound together by business logic or methods
- Disadvantages:
 - Expensive to implement
 - There is the need to change application logic (test, integrate, and redeploy the application within the enterprise) – a process that often causes costs to spiral upwards


Method-Level Example

- Let's say two applications exist within an enterprise. One application is a C++-based application running on a Linux machine. The other is a NT-based client/server application written in Java on the front-end, with Sybase serving as the back-end database.
- If we want to add a new customer to those applications, then they will invoke different methods, for example:
 Add_Cust() on the Linux/C++ system, and
 AddNewCustomer() on the NT/Java system



How can we integrate those applications?

Problem Solving



- Goal: create a composite application
- Techniques:
 - Move much of the business logic to a shared server, such as an application server
 - Rebuild each application using a method-sharing mechanism, such as distributed object technology, to create a tightly coupled application that allows easy cross-access of methods
 - Expose each method using either a distributed object standard or a custom programming solution

What's a Process?



- A business process: any rule or piece of logic that exists within the enterprise that has an effect on how information is processed
- Thousand of business processes reside in the applications of most organizations
- The fact that those processes depend on a chaotic mix of enabling technologies

Scenario



- In order to implement method-level EAI, it is necessary to understand all the processes that exist within the enterprise
- How best to proceed?
 - Break the processes down into their scenarios or types
 - Those types are rules, logic, data and objects

Rules



- A rule is a set of conditions that have been agreed upon
 - Ex. A rule may state that employees may not fly first class on flights of less than 5000 miles or that all transactions over one million dollars must be reported to the government
- The rules that exist within a given enterprise are built into the applications in order to control the flow of information

Logic



- Logic is a sequence of instructions in a program, for example, **if** this button is pushed **then** the screen pops up - while rules define business restraints
- Three classes of logic:
 - Sequential processing
 - Series of the steps in the actual data processing
 - Input, output, calculation and move (copy) are instructions used in sequential processing

Logic [2]



- Selection
 - Decision-making within the program, performed by comparing two sets of data and, depending on the results, branching to different parts of the program
- Iteration
 - A repetition of a series of steps
 - It is accomplished with DO-LOOPS and FOR-LOOPS in high-level languages

Objects



- Bundles of data encapsulates inside an object and surrounded by methods that act upon the data
- Objects in systems generally use object-oriented technology such as C++ or Java

Method Warehousing



- Process of aggregating all business processes within an organization and then hosting them on a centralized server (the method warehouse)
- Advantage:
 - Power to integrate all enterprise applications through the sharing of methods (e.g. composite applications)
 - The ability to maintain and reuse application logic from a centralized server
 - Ex. If tax law changes, it would be unnecessary to locate all applications within the enterprise that use tax law in order to incorporate the changes. The changes would automatically propagate to all link applications simply by a change in the logic, or rules, within the method warehouse.

Method Warehousing [2]

- Disadvantage:
 - Requires the conversion of the significant methods of every source and target application into a common enabling technology, one that all enterprise applications will be able to access
 - Ex. If the decision is made to implement a method warehouse project using an application server or a distributed object technology infrastructure, then, to create the method warehouse itself, all applications will have to be rewritten in order to invoke these methods remotely.

Leveraging Frameworks for EAI



- Frameworks consist of abstract classes and their object collaboration as well as concrete classes. While object-oriented programming supports software reuse, frameworks support design reuse (Freedman, 1993).
- Frameworks are fully debugged and tested software subsystems, centrally located and accessible by many applications.
- Frameworks provide the infrastructure for sharing methods -- providing objects that are accessible by a number of applications

Advantages of Frameworks

- Advantages:
 - The ability to reuse existing code
 - Provide a standard set of predesigned and pretested application architectures
 - Those architectures provide a common infrastructure for all enterprise applications

Disadvantages of Frameworks



- Disadvantages:
 - Design is an essential part of application development
 - Knowing how the application and framework fit together is essential for success
 - Frameworks are typically bound to a technology
 - There needs to be a commitment to a common set of enabling technologies

Framework Functionality



- Frameworks largely define the architecture for an application architects must build and leverage only those frameworks that are extensible and flexible
- Frameworks can be customized by adding sets of classes or frameworks may change themselves to take advantage of changes in hardware and software as new revision are released
- Components don't provide the same degree of customization as object-oriented frameworks

Framework Types



- Object frameworks
- Service frameworks
- Procedural frameworks
- Component frameworks

Object frameworks



- Are made up of both abstract and concrete classes
- Provide application services through the inheritance mechanisms that most object-oriented languages and tools provide
 - Developers get a copy of the framework source code and have an opportunity to customize the framework

Service Frameworks



- Don't provide functionality to applications through inheritance
 - Ex. distributed object framework

Procedural Frameworks



- Represent a “black box” perspective on frameworks because they do not allow developers to extend or modify their basic set of services
- Enabling technology, which supports procedural frameworks, includes TP monitors (such as Microsoft Transaction Server, etc)

Component Frameworks



- Components deal with the interface on the client, where application services and procedural frameworks provide basic application services on the back-end
- Don't provide the reuse granularity of true object-framework development
- Provide reusable features that eliminate the need-to-know component implementation details
- Usually contain fewer features and functions than applications, but they provide more services than a simple program function or object

Framework Categories



- **Application service frameworks**
 - Encapsulate enterprise application functionality
- **Domain frameworks**
 - Encapsulate expertise for certain problem domains and provide vertical functionality for certain areas of the enterprise (e.g. accounting, marketing or logistics)
 - The idea here is to build common application architectures into a common framework shared across applications
- **Support frameworks**
 - Offer native system-level services such as network support, device access or file access
 - Platform independent

Enabling Technology



- Application or transaction servers
 - TP (Transaction Processing) monitors are industrial-strength middleware products that provide many features that make large-scale distributed transaction-oriented development possible
 - Application servers support the concept of a transactional system
- Message brokers
 - A technology that acts as a broker between one or more target entities (such as network, middleware, applications and systems), allowing them to share information with less difficulty than traditional middleware
 - Middleware for middleware

Enabling Technology [2]



- Message brokers (cont.)
 - Good at moving and translation information between many applications but are poor places to host centralized business processes
 - Their rules engines are designed for message routing and transformation, not application logic
- Distributed objects
 - DCOM (Distributed Component Object Model)
 - CORBA (Common Object Request Broker Architecture)

Summary



- Method-level EAI allows the enterprise to be integrated through the sharing common business logic or methods
 - This is accomplished either by defining methods that can be shared or by providing the infrastructure for such method sharing
- The goal of method-level EAI: composite applications
- Frameworks provide the infrastructure for sharing methods -- providing objects that are accessible by a number of applications
 - Object frameworks, service frameworks, procedural frameworks, component frameworks
- Enabling technology: Application or transaction servers, message brokers, distributed objects

References



- Lithicum, David S. Enterprise Application Integration. Addison-Wesley, 2000
- Ruh, Maginnis, and Brown. Enterprise Application Integration. Wiley, 2001.
- Ruh and Bernstein. Enterprise Integration: the essential guide to integration solutions, Addison-Wesley, 2005.
- Linthicum, David S., Next Generation application integration: from simple information to web services, Addison-Wesley, 2004
- James Fenner, Enterprise Application Integration Techniques.
<http://www.cs.ucl.ac.uk/staff/ucacwxe/lectures/3C05-02-03/aswe21-essay.pdf>. Last Accessed 13rd January 2010