**Systems Integration and Architecture**

**Summary of Chapter 05: Solving Integration Problems using Patterns**

**Tight Coupling**

A great example of tight coupling is a local method invocation.

Invoking a local method inside an application is based on a lot of

assumptions between the called and the calling routine..

The call is immediate, i.e. the called method starts processing immediately after the calling method makes the call.

Meanwhile, the calling method will only resume processing when the called method completes (meaning the invocation is synchronous).

**Tight Coupling Example Problem**

* Assume we are building an on-line banking system that allows customers to deposit money into their account from another bank.
* Let’s assume that the remote function that deposits money into a person’s account takes only the person’s name and the Dollar amount as arguments.
* To perform this function, the front-end Web application has to be integrated with the back-end financial system that manages fund transfers.



This minimalist integration solution is fast and cheap, but it results in a very brittle solution because the two participating parties make the following assumptions about each other:

– Platform Technology – internal representations of numbers and objects

* 32 bit vs 64 bit internal representation
* A system using 64 bits would be inclined to read 8 bytes off the network and would end up interpreting the whole message (including the customer name) as a single number.

– Location – hard-coded machine addresses

* Remote functions and machine location are hard-coded. Any change in anything, we would have to change code. If we use a lot of remote functions this could become very tedious.

– Time – all components have to be available at the same time

* Establishing a TCP connection requires that both machines and the network are all available at the same time.

– Data Format – the list of parameters and their types must match

* If we want to insert a third parameter, e.g. the name of the currency, we would have to modify both sender and receiver to use the new data format.

**Loosely coupled solution**

Instead, we can develop loosely coupled solution by using a message-oriented middleware (MOM) infrastructure

MOM mechanisms provides services such as

– a common data format and transformers

* Removes platform and data format dependencies as the sender no longer has to depend on the receiver's internal data format

– queuing channels

* Removes location and time dependencies as we do not have to pay attention to computer’s identity & location or whether the other computer is ready to accept requests or not.

**Things that make up Middleware**

Middleware – the things that sit between applications.

In order to connect two systems via an integration solution, a number of things have to happen.

**Messages**

* Message: a snippet of data that has an agreed-upon meaning to both applications that are to be integrated.
* This piece of data can be very small, such as the phone number of a single customer that has changed, or very large, such as the complete list of all customers and their associated addresses.



Message Construction

When two applications wish to exchange a piece of data, they do so by wrapping it in a message.

• Message intent

* Command Message: invoking a function or method on the receiver
* Document Message: enabling the sender to transmit one of its data structures
* Event Message: notifying the receiver of a change in the sender.

• Returning a response

* Request-Reply: The request is usually a Command Message, and the reply is a Document Message containing a result value or an exception.
* Return Address: The requestor specifies what channel to use to transmit the reply.
* Correlation Identifier: Responder specifies which request this reply corresponds to.

• Huge amounts of data

* Message Sequence: break the data into more manageable chunks and send them as a sequence of messages, so that the receiver can reconstruct the original data structure.

• Slow messages

* Message Expiration: the sender can specify an expiration date. If the messaging system cannot deliver a message by its expiration, it should discard the message. If a receiver gets a message after its expiration, it should discard the message.

**Channel**

Data needs to be transported, usually across a network.

We need a communications channel that can move information from one application to the other. This channel could be a series of TCP/IP connections, a shared file, or a shared database.

* A channel is a logical address that both sender and receiver can agree on the same channel without being aware of each other’s identity.



Messaging Channels

* Point-to-Point Channel: To send the data to a single application (1-to-1 interaction)
* Publish-Subscribe Channel: When you send a piece of data this way, the channel effectively copies the data for each of the receivers (One to many interaction)
* Datatype Channel: all of the data on a channel has to be of the same type (many to one interaction)
* Invalid Message Channel: When receiver receives a message that doesn’t meet these expectations
* Dead Letter Channel: for messages which are successfully sent but ultimately cannot be successfully delivered.
* Guaranteed Delivery: makes channels persistent so that their messages are stored on disk
* Channel Adapter: can be used to connect a channel (or set of channels) to the application without having to modify the application
* Messaging Bridge: connecting two message systems, effectively connecting them into one composite messaging system.
* Message Bus: a backbone providing access to all of the enterprise’s various and ever-changing applications and functionality.

**Translation**

Middleware needs to provide some mechanism to convert one application’s data format in the other’s.

* Internal data format of an application can often not be changed
* For example, one data format may store the customer name in two fields, called FIRST\_NAME and LAST\_NAME, while the other system may use a single field called Customer\_Name.



Message Transformation

* Envelope Wrapper: wrap message payload data into an envelope that is compliant with the requirements of the messaging infrastructure.
* Content Enricher: when the target system requires data fields that the originating system cannot supply. It has the ability to look up missing information or compute it from the available data.
* Content Filter: removes unwanted data from a message.
* Claim Check: removes data from a message but stores it for later retrieval.
* Normalizer: translates messages arriving in many different formats into a common format.
* Canonical Data Model: Design an independent data model from any specific application. Require each application to produce and consume messages in this common format

**Routing**

Middleware needs to take care of sending messages to multiple systems

* As the number of systems increases it becomes very tedious and requires the sending system to have knowledge about all other systems.
* For example, if the customer address changes in the customer care system we could make that system responsible for sending the data to all other systems that store copies of the customer address.
* We could expect each application to specify the target system(s) for the data it is sending over the channel.



**Systems management**

Integration solutions can quickly become complex because they deal with multiple applications, data formats, channels, routing and transformation.

* All these elements may be spread across multiple operating platforms and geographic locations.

In order to have any idea what is going on inside the system we need a systems management function.

* This subsystem monitors the flow of data, makes sure that all applications and components are available and reports error conditions to a central location.



* Control Bus: provides a single point of control to manage and monitor a distributed solution
* Detour: route messages through additional steps, such as validation or logging – with ability to switch on or off these additional steps
* Wire Tap: inspect the contents of a message without affecting the primary message flow.
* Message History: great aid to know where a specific message has been
* Message Store: can provide a complete account of every message that traveled through the system
* Smart Proxy: track messages sent to request-reply services
* Test Message: actively verifying that the running messaging system is functioning properly
* Channel Purger: remove all remaining messages from a channel so that the components under test do not receive 'leftover' messages.

**Endpoint**

Most packaged and legacy applications and many custom applications are not prepared to participate in an integration solution.

* As they were designed to perform specific functionality with specific input/output

• We need a message endpoint to connect the system explicitly to the integration solution.

* The endpoint can be a special piece of code or a Adapter provided by an integration software vendor.



Messaging Endpoints

• Messaging Gateway: a class than wraps messaging-specific method calls and exposes domain-specific methods to the application.

• Messaging Mapper: creates a mapping logic between the messaging infrastructure and the domain objects.

• Transactional Client: make the client’s session with the messaging system transactional so that the client can specify transaction boundaries.

• Polling Consumer: explicitly makes a call when it wants to receive a message.

• Event-Driven Consumer: automatically handles messages as they’re delivered on the channel.

• Competing Consumers: Create multiple consumers on a single channel so that multiple messages can be processed concurrently.

• Message Dispatcher: consume messages from a channel and distribute them to performers.

• Selective Consumer: filters messages delivered to a channel so that it only receives the ones that match its criteria.

• Durable Subscriber: to make the messaging system save messages published while the subscriber is disconnected.

• Idempotent Receiver: can safely receive the same message multiple times.

• Service Activator: connects the messages on the channel to the service being accessed.

**Summary of Chapter 06: XML and Application Integration**

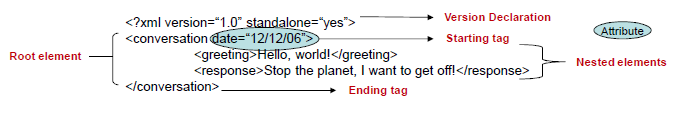
**The Value of XML**

* XML provides a common-data exchange format, encapsulating both data and metadata
* Promotes a self-defining message structure
  + A standard format of information exchange on the Internet
  + An infrastructure for information exchange and management in the world of application integration
* XML provides a robust, human-readable information exchange standard
  + Supports the exchange of application semantics and information content
  + Provides an application-level mechanism for producing business information

**XML Meets Middleware**

* Middleware simply “carries the load”
  + It moves messages that encapsulate or abstract XML
  + Ensures that those message are understood by any source or target applications that need the information
* Middleware manage the interfaces with the source or target application
  + Move information into and out of application
  + Need necessary changes to source and target applications to consume and produce XML
* XML demands huge overhead
* XML document uses more bytes than binary message

**XML: Elements**

****

Elements

* Basic building blocks of an XML document

Defined by tags

* Starting tag and an ending tag

Root element

* Outermost element in the XML document

XML supports nested elements

* Elements within elements
* This ability allows XML to support hierarchical structure

Element names describe the content of the element

Structure describes the relationship between the elements

Attributes of an element

* Describe characteristics of the elements in the beginning tag of an element

**XML Schema**

XML Schema dictates what can and cannot be done with XML data, in same way database schema establish a structural model for data they represent.



XML document is created based on vocabulary of elements

Vocabularies can be defined formally using a schema definition language, XML Schema

* Protect integrity of XML document data by providing structure, validation rules, data type constraints, and inter-element relationships

Plays an important role as an enterprise data transport standard as it greatly refines the quality of XML data

Schema allows authors to establish logical domains to which some or all parts of a schema can be applied

**XML Parsers**

XML parsers reads XML document and extracts the data for access by another program

• Document Object Model (DOM) - Tree-based API

• Simple API for XML (SAX) – event based API

**XML Namespaces**

A namespace is a collection of names that may be used in an XML document as elements or attribute names

* Identify names with a particular domain and avoid redundancy
* Allow use of the same name with two different meanings

Namespaces are identified by a Uniform Resource Indicator (URI)

* Which allows each namespace to be unique

**What is XSLT?**

Extensible Stylesheet Language Transformations (XSLT)

* Language designed to transform one XML document into another changing both its schema and content in the process
* At its most primitive, it is text processing system
* Can generate other standard markup languages

**XSLT Mechanism**

Transforming an XML document using XSLT requires two main steps

1. Structural transformation

* + Data is transformed from the input structure to the output structure
  + Involves selecting data, grouping it, sorting it, or aggregating it

2. Formatting the text to new characteristics

* + Information is placed in a particular type of text structure
  + E.g., XML, HTML, PDF, etc

**XSLT Processors**

XSLT processors apply an XSLT style sheet to an XML source document and thus

create a resulting document

* + While remaining consistent with the way processors handle XML through trees

XSLT must process three tress

* + Input tree
  + Stylesheet tree
  + Output tree

Stylesheet document defines the transformation to occur

* + XSLT processor uses stylesheet tree to transform the input tree to the output tree



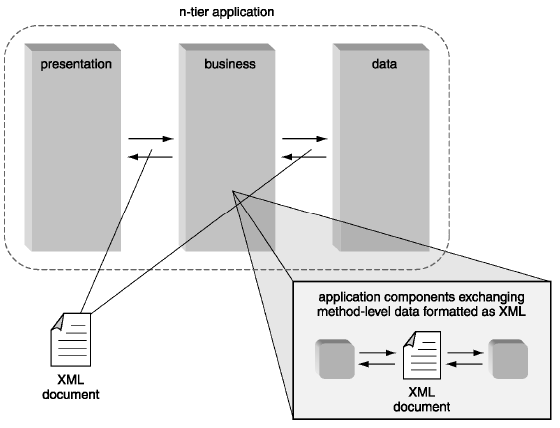
**Using XSLT for B2B Application Integration**

• XSLT provides standard approach to both rules and transformation processing

• Preferred standard mechanism for transforming content and application semantics as information moves from application to application and business to business

• XSLT can also create text-based formats

**XML can establish a standard data transport format within and between application tiers**



**Summary of Chapter 07: Web Services**

**Supporting Interactions**

Purpose of Systems Integration is to support each of the following:

* Human to Human Interactions
* Human to Machine (Applications) Interactions
* Machine (Applications) to Machine interactions

**Constructs Necessary to Support Interactions**

Different things necessary to support interactions

* + Language (English, ASP.Net, Java)
  + Vocabulary (meaning of words, meaning of code)
  + Context (conversation context)
  + Medium (communication channel, ability to speak, and ability to listen)
  + Situation awareness (recognizing where you are, role of the person)

• Web service supports application-to-application communication

* + Supports “loosely coupled” integration
  + Minimizes the amount of effort required to build integrated applications.

• With Web services,

* + the applications or programmers can find cooperative programs to accomplish a specific task,
  + allowing programmers to rapidly assemble applications by merely tying together application modules.

• Web services are designed to enable application modules (objects) to communicate with other application modules.

• Once connected, service applications provide transactional or computational services.

**Three main “parts” to Web services**

1. Data is presented & shared using XML

2. Applications find services and share information and data across diverse systems environments using shared, open, emerging technology standards - SOAP, UDDI, WSDL etc.

3. How communications take place over a common network (the Internet) using HTTP protocol as the transport

The above parts are supported by Service-Oriented Architecture

**Service-Oriented Architecture**

• Consider a scenario of finding a plumber to fix a plumbing problem.

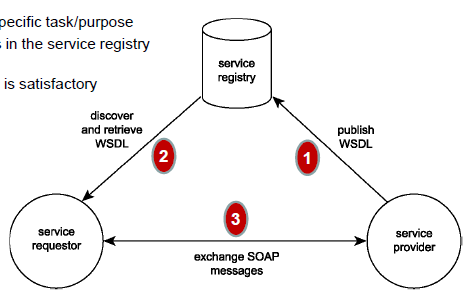
* + Find a plumber in Yellow Pages (Discovery)
  + Plumber should have advertised in the Yellow Pages (Publishing)
  + You call plumber to schedule appointment (Binding)

• Three basic participants

* + Service Provider
  + Service Registry
  + Service Requestor

• Three basic operations

* Publishing services
* Discovering services
* Binding services



1. **Service Provider**

* + Develops an application and converts into a service
  + Creates a WSDL document describing the capabilities of the service and how to access the service
  + Publishes the WSDL document in a service registry (UDDI)

2. **Service Requestor**

* Needs a service for specific task/purpose
* Searches for services in the service registry meeting needs
* Selects a service that is satisfactory

3. **Service Requestor’s**

* application invokes the provider’s service
* Upon acceptance, application and service can exchange data using SOAP

**Key Web Service Standards: SOAP**

Simple Object Access Protocol (SOAP) specification

* Submitted to W3C in 2000
* SOAP 1.1 became standard in July 2003
* SOAP 1.2 become standard in April 2007 (current standard version)
* XML-based messaging format established a transmission framework for inter-application (or inter-service) communication via HTTP
* SOAP specification is vendor neutral technology, therefore, it was an attractive alternative to proprietary protocols such as CORBA and DCOM

**Key Web Service Standards: WSDL**

Web Service Description Language (WSDL) specification

* XML-based language for describing the interface of Web services.
* WSDL 1.1 became standard in March 2001
* WSDL 2.0 became standard in June 2007 (current standard version)

**Key Web Service Standards: UDDI**

Universal Description, Discovery, and Implementation (UDDI) specification

* Mechanism for the dynamic discovery of service descriptions
* UDDI 2 became standard in April 2003
* UDDI 3 became standard in February 2005 (current standard version)

**Relationship between key Web service standards**

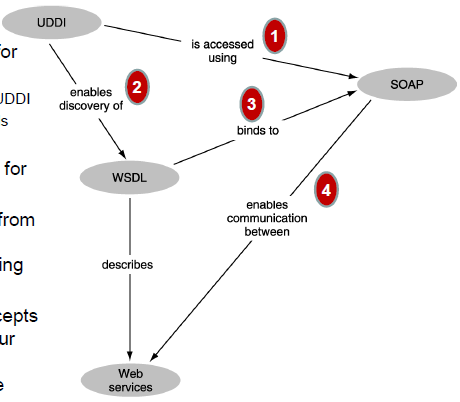
1. Use UDDI to search for services

* Use SOAP to access UDDI
* Search using key words

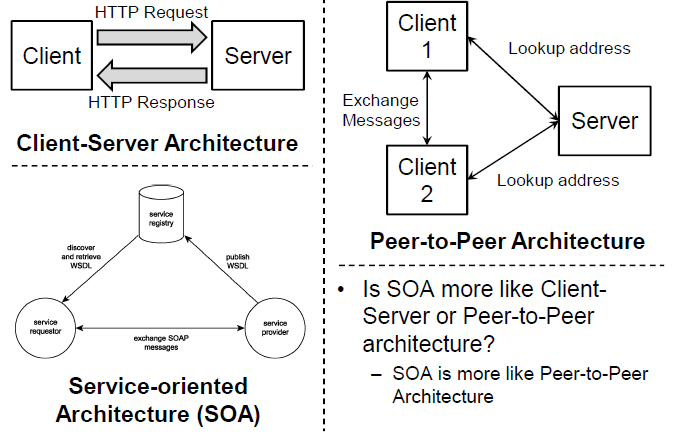
2. Download WSDL document from UDDI for selected service

3. Use URI information from WSDL document to invoke the service using SOAP

4. After the services accepts your request, then your application and the service can exchange data using SOAP



**Client Server and Peer-to-Peer Architectures**



**Web-based Application Development**

Diagram

Description automatically generated

**Web Service Description Language (WSDL)**

• Abstract – the description of a Web service interface, independent of implementation details

• Abstract interface definition is made up of

* Interface and Message elements

• Concrete – specifies location and implementation information about a Web service

• Concrete interface definition is made up of

* Binding, Endpoint, and Service elements

**WSDL Definition Elements – Abstract Interface definition**

• Interface – contain a group of logically related operations

• Operations – represents a single action or function performed by an application (i.e., a method in an application)

• Operations consists of group of related input and output messages

* Message exchanges required to support execution of the operation

• Message element can contain one or more input or output parameter that belong to an operation.

* Part elements are used to define parameters

• Part element provides a name, value set, along with an associated data type.

**WSDL Definition Elements – Concrete (Implementation) definition**

• Service – represents one or more endpoints at which the Web service can be accessed

• Endpoint – consist of location (URI) and protocol information

• Binding – defines invocation requirements of each of its operation

* Associates protocol and message format information to operations
* Operations construct within Binding block resembles its counterpart in the interface section

• Each Endpoint can reference to a Binding element, and therefore relates the Endpoint information to the underlying operation

**WSDL Supplementary Constructs**

• Provide additional information about the service

• Types element – provide data type support for Web service definitions

* XSD Schema information is provided

• Documentation element allows supplementary annotations to be added

**Simple Object Access Protocol (SOAP)**

• SOAP specification establishes a standard message format that consists of an XML document capable of hosting RPC and document-centric data

* Document-centric is most commonly used, due to standard endpoint description is provide in WSDL

• SOAP facilitates synchronous (request and response) as well as asynchronous (process-driven) data exchange models

**SOAP Message Structure**

• The root Envelope element frames the message document consists of a mandatory Body element and an optional Header element

• Header element is used for

* Including implementation of SOAP extensions (advanced Web service standards)
* Identification of target SOAP intermediaries
* Processing information for SOAP intermediaries
* Providing supplementary meta information about the SOAP message

• Within the Body element, data being delivered by the SOAP message is included

• Fault element can be used host exception information

* Embedded within Body element

**Universal Description, Discovery, and Implementation (UDDI)**

• Fundamental of SOA is a mechanism for service descriptions to be discovered by potential requestors.

• UDDI is a central directory that hosts service descriptions (including WSDL)

• UDDI specifies a Registry that stores service descriptions

within a directory

• UDDI Registry can be

* Public Registry - a global directory of services
* Private Registry – repositories of services hosted within an organization

**UDDI Registry Elements**

UDDI registries are organized into six primary types of data:

• Business Entities (businessEntity element)

* Provides profile information about the registered business, including its name, a description, and a unique identifier

• Business Services (businessServices element)

* Records of the actual services offered by the registered business are nested within the businessEntitity element

• Specification Pointers (bindingTemplate element)

* Provides address linking the businessService to implementation information
* Developer can learn how and where to physical bind to a Web service

• Service Types (tModel element)

* Points to location of service interface definitions (WSDL), message formats, as well as message and security protocols

• Business Relationships (publisherAssertion element)

* Provides a means of establishing the relationship of the current businessEntity with another

• Subscriptions (subscription element)

* Allows subscribers to be notified when business entity profile information is updated

**Accessing UDDI Registry**

• UDDI provides inquiry and publishing APIs allowing applications to interface programmatically with a registry

• Registries are expected to provide interface for humans as well

Good Luck

Dr. Abdullah Baqasah