Contemporary SOA, Part I

605.702 Service Oriented Architecture
Johns-Hopkins University

Lecture 3 Goals

- Part one of two to become familiar with the concepts behind Contemporary SOA
  - More then Web Services
  - Message Exchange Patterns (MEP’s)
  - Linking business processes to services and SOA
  - Control Services: Coordination, Orchestration, Choreography
  - Developing a ….. (Practice session)

Session #3 Today’s Agenda

- Required Reading (to be read before class)
  - Erl Chapter 6: Web Services and Contemporary SOA I
    - Activity Management and Composition
- Today’s Lecture
  - ***Questions about Lecture 2, review sections 5.3 and 5.4***
  - Ch 6: Web Services, Activity Management and Composition

Review Last Week's Material

- An opportunity to ask questions
  - Chapter 4: The Evolution of SOA
  - Chapter 5:Web Services and Primitive SOA

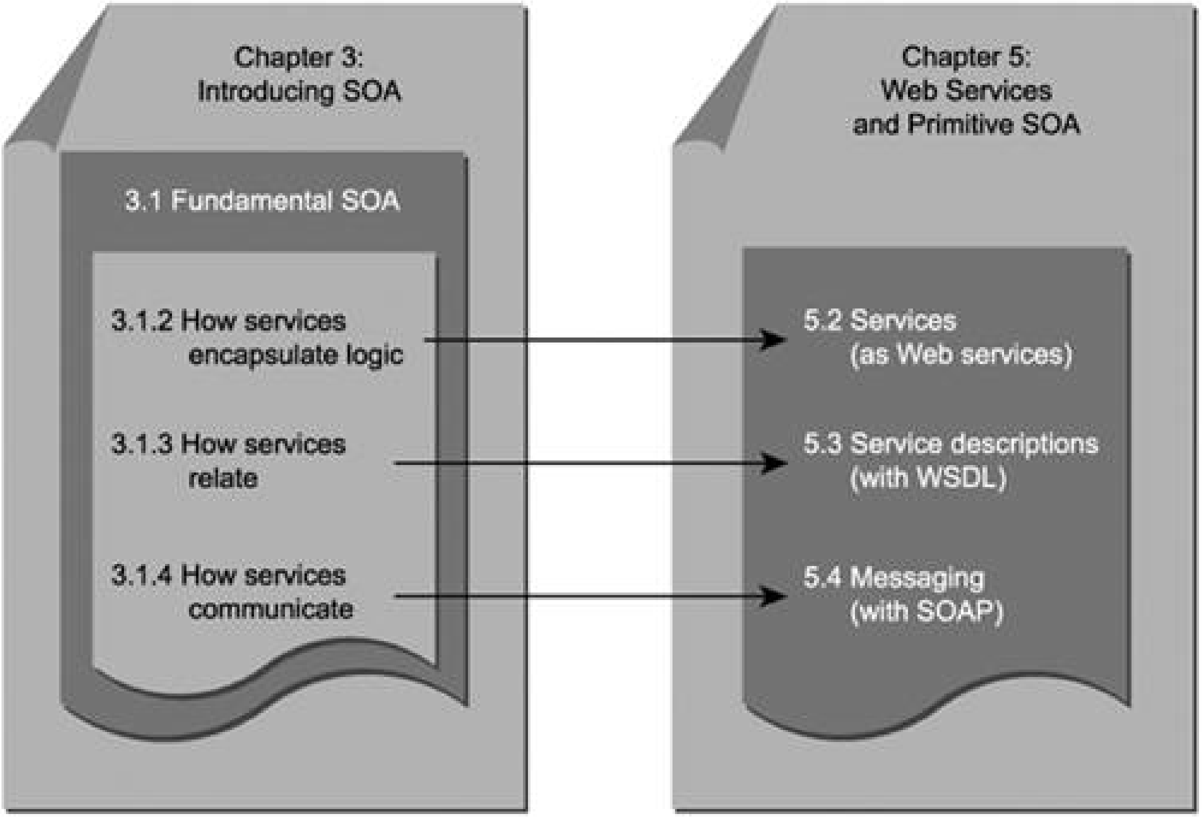
Review Last Week's Material

- Primitive SOA (simplistic definition)
  - SOAP, WSDL, UDDI
- Roles { Service Provider, Requestor, Intermediator}
  - Initial Sender and Utimate Receiver
- Models
  - Business Service Model, Utility Service Model, Controller Service Model

The Web services framework

Figure 5.1. The structural relationship between sections in Chapters 3 and 5.

[View full size image]



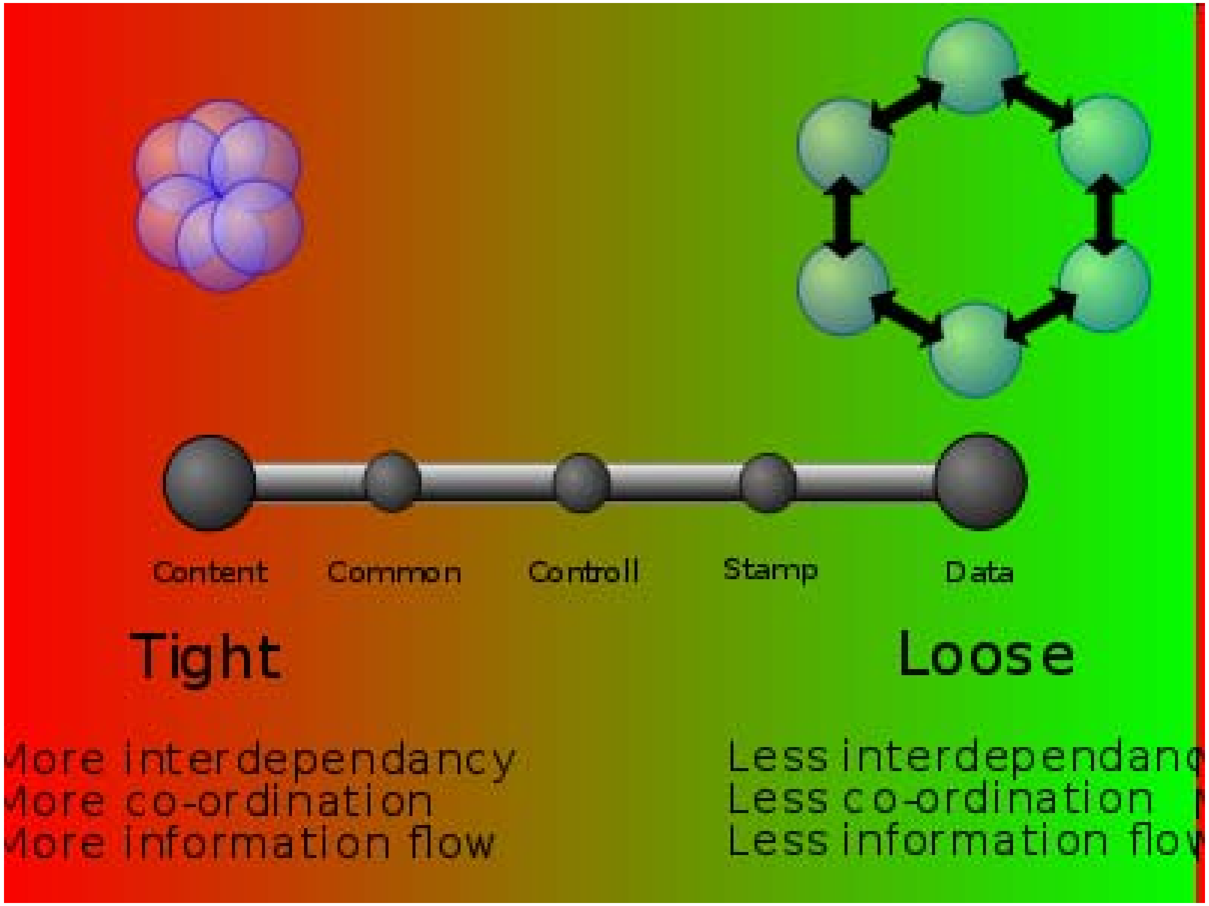
The Web services framework

- an abstract (vendor-neutral) existence defined by standards organizations and implemented by (proprietary) technology platforms
- core building blocks that include Web services, service descriptions, and messages
- a communications agreement centered around service descriptions based on WSDL
- a messaging framework comprised of SOAP technology and concepts
- a service description registration and discovery architecture sometimes realized through UDDI
- a well-defined architecture that supports messaging patterns and compositions (covered in Chapter 6)
- a second generation of Web services extensions (also known as the WS-* specifications) continually broadening its underlying feature-set (covered in Chapters 6 and 7)

Coupling

- http://en.wikipedia.org/wiki/Coupling_%28computer_science%29

Types of coupling



Types of coupling

- Conceptual model of coupling
- Coupling can be "low" (also "loose" and "weak") or "high" (also "tight" and "strong"). Some types of coupling, in order of highest to lowest coupling, are as follows:
- Content coupling (high)
  - Content coupling is when one module modifies or relies on the internal workings of another module (e.g., accessing local data of another module).
  - Therefore changing the way the second module produces data (location, type, timing) will lead to changing the dependent module.
- Common coupling
  - Common coupling is when two modules share the same global data (e.g., a global variable).
  - Changing the shared resource implies changing all the modules using it.
- External coupling
  - External coupling occurs when two modules share an externally imposed data format, communication protocol, or device interface.

Types of coupling

- Control coupling
  - Control coupling is one module controlling the flow of another, by passing it information on what to do (e.g., passing a what-to-do flag).
- Stamp coupling (Data-structured coupling)
  - Stamp coupling is when modules share a composite data structure and use only a part of it, possibly a different part (e.g., passing a whole record to a function that only needs one field of it).
  - This may lead to changing the way a module reads a record because a field that the module doesn't need has been modified.
- Data coupling
  - Data coupling is when modules share data through, for example, parameters. Each datum is an elementary piece, and these are the only data shared (e.g., passing an integer to a function that computes a square root).
- Message coupling (low)
  - This is the loosest type of coupling. It can be achieved by state decentralization (as in objects) and component communication is done via parameters or message passing No coupling
  - Modules do not communicate at all with one another.

Cohesion

- http://en.wikipedia.org/wiki/Cohesion_%28computer_science%29
- The types of cohesion, in order of the worst to the best type, are as follows:
- Coincidental cohesion (worst)
  - Coincidental cohesion is when parts of a module are grouped arbitrarily; the only relationship between the parts is that they have been grouped together (e.g. a "Utilities" class).
- Logical cohesion
  - Logical cohesion is when parts of a module are grouped because they logically are categorized to do the same thing, even if they are different by nature (e.g. grouping all mouse and keyboard input handling routines).
- Temporal cohesion
  - Temporal cohesion is when parts of a module are grouped by when they are processed - the parts are processed at a particular time in program execution (e.g. a function which is called after catching an exception which closes open files, creates an error log, and notifies the user).

Cohesion

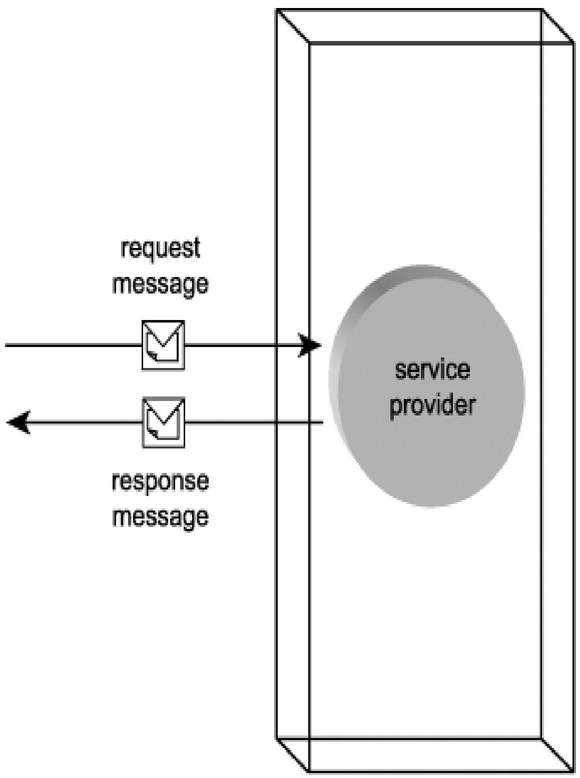
- Procedural cohesion
  - Procedural cohesion is when parts of a module are grouped because they always follow a certain sequence of execution (e.g. a function which checks file permissions and then opens the file).
- Communicational cohesion
  - Communicational cohesion is when parts of a module are grouped because they operate on the same data (e.g. a module which operates on the same record of information).
- Sequential cohesion
  - Sequential cohesion is when parts of a module are grouped because the output from one part is the input to another part like an assembly line (e.g. a function which reads data from a file and processes the data).
- Functional cohesion (best)
  - Functional cohesion is when parts of a module are grouped because they all contribute to a single well-defined task of the module (e.g. tokenizing a string of XML).

Services (as Web services)

- every Web service can be associated with:
  - a temporary classification based on the roles it assumes during the runtime processing of a message
  - a permanent classification based on the application logic it provides and the roles it assumes within a solution environment
- We explore both of these design classifications in the following two sections:
  - service roles (temporary classifications)
  - service models (permanent classifications)

Service roles

Figure 5.2. As the recipient of a request message, the Web service is classified as a service provider.

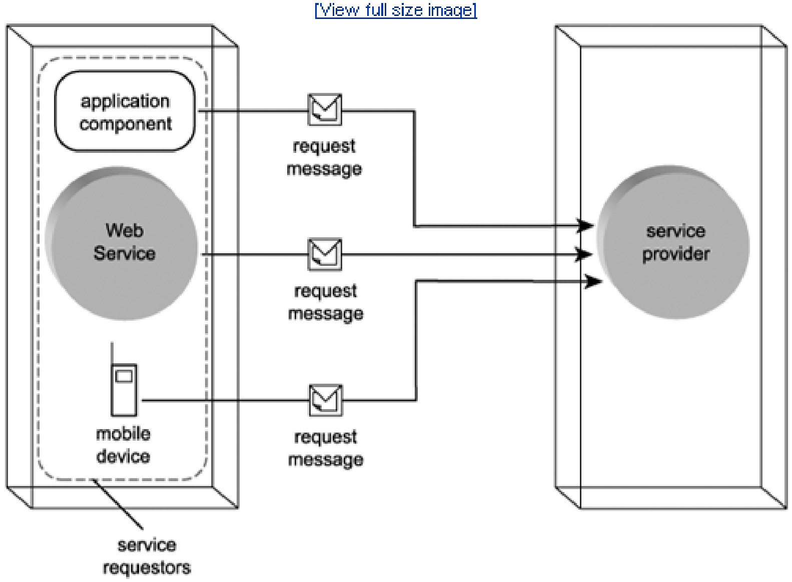


- The Web service is invoked via an external source, such as a service requestor (Figure 5.2).
- The Web service provides a published service description offering information about its features and behavior. (Service descriptions are explained later in this chapter.)

Service requestor

Figure 5.3. The sender of the request message is classified as a service request

[View full size image]

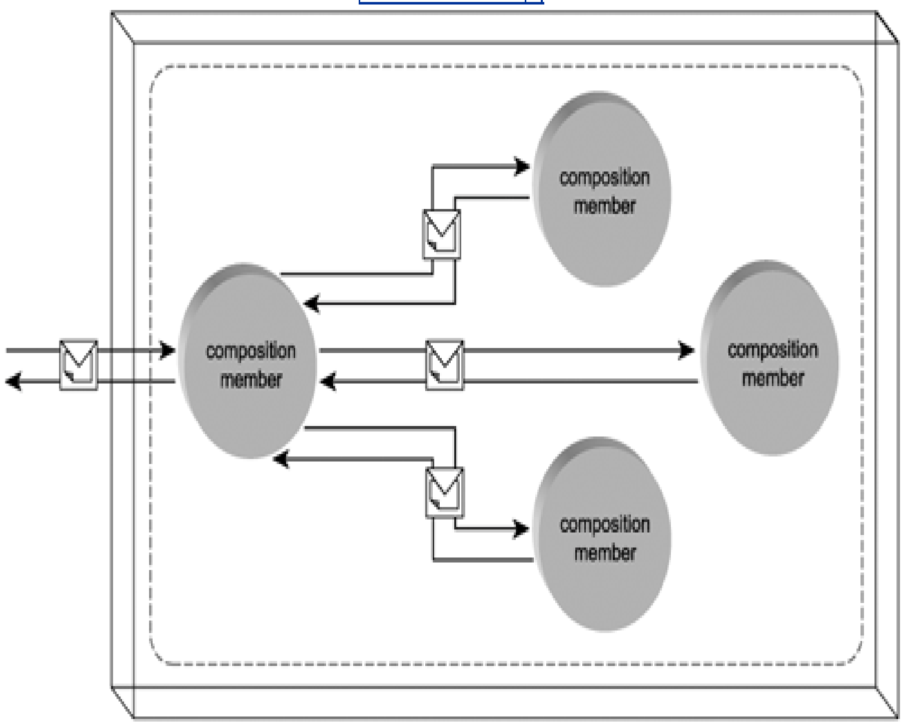


- The Web service invokes a service provider by sending it a message (Figure 5.3).
- The Web service searches for and assesses the most suitable service provider by studying available service descriptions. (Service descriptions and service registries are covered in the Service descriptions (with WSDL) section.)
- service requestor entity (the organization or individual requesting the Web service)
- service requestor agent (the Web service itself, acting as an agent on behalf of its owner)

Service compositions

Figure 5.10. A service composition consisting of four members.

[View full size image]



- Service-orientation principles place an emphasis on composability,
- allowing some Web services to be designed in such a manner that they can be pulled into future service compositions without a foreknowledge of how they will be utilized.
- The concept of service composability is very important to service-oriented environments
- In fact, service composition is frequently governed by WS-* composition extensions, such as WS-BPEL and WS-CDL, which introduce the related concepts of orchestration and choreography, respectively

Service models

- Business services are used within SOAs as follows:
  - as fundamental building blocks for the representation of business logic
  - to represent a corporate entity or information set
  - to represent business process logic
  - as service composition members

Utility service model

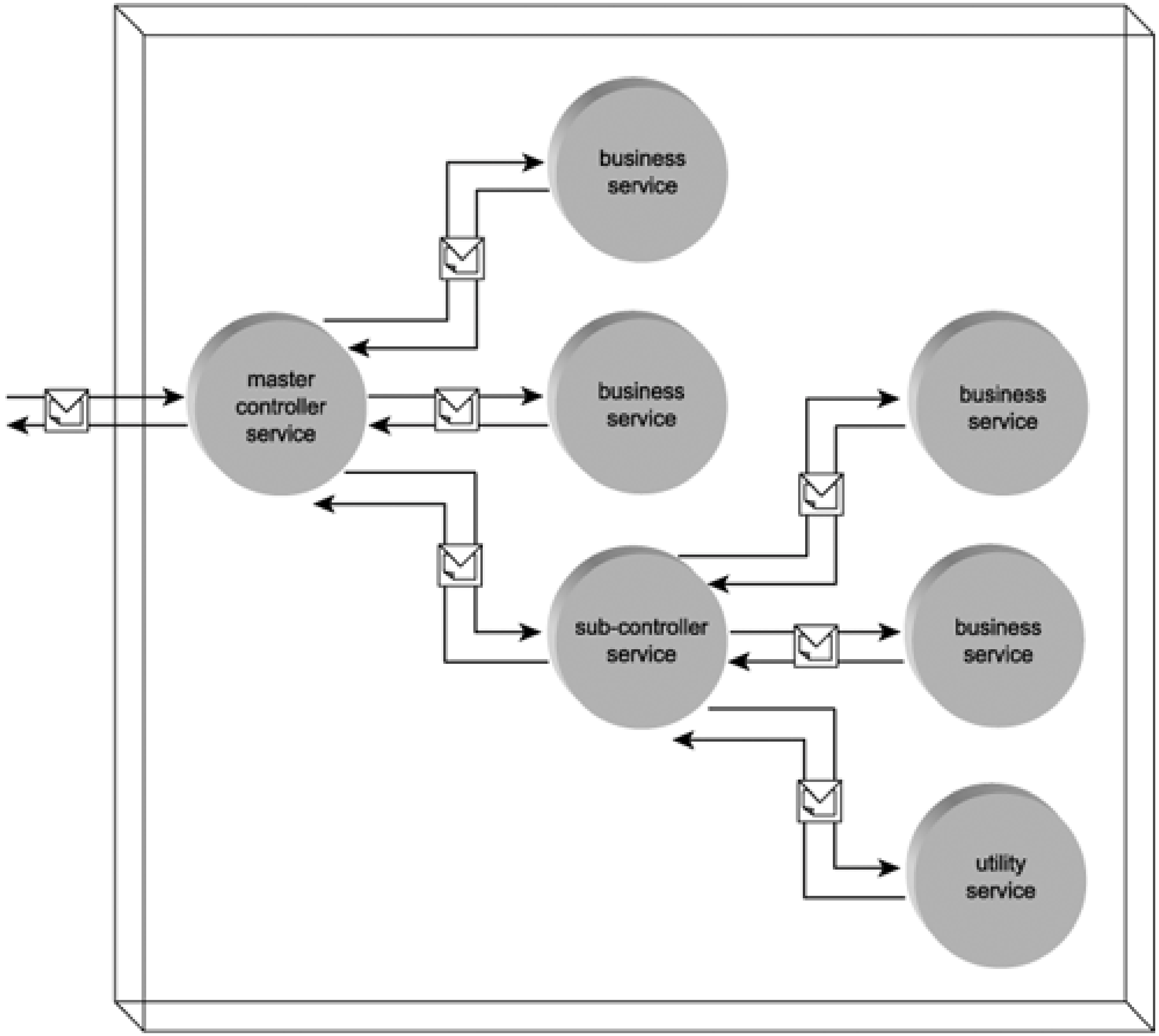
- Utility services are used within SOAs as follows:
  - as services that enable the characteristic of reuse within SOA
  - as solution-agnostic intermediary services
  - as services that promote the intrinsic interoperability characteristic of SOA
  - as the services with the highest degree of autonomy

Controller service model

- Controller services are used within SOAs as follows:
  - to support and implement the principle of composability
  - to leverage reuse opportunities
  - to support autonomy in other services

Figure 5.12. A service composition consisting of a master controller, a sub-controller, four business services, and one utility service.

[View full size image]



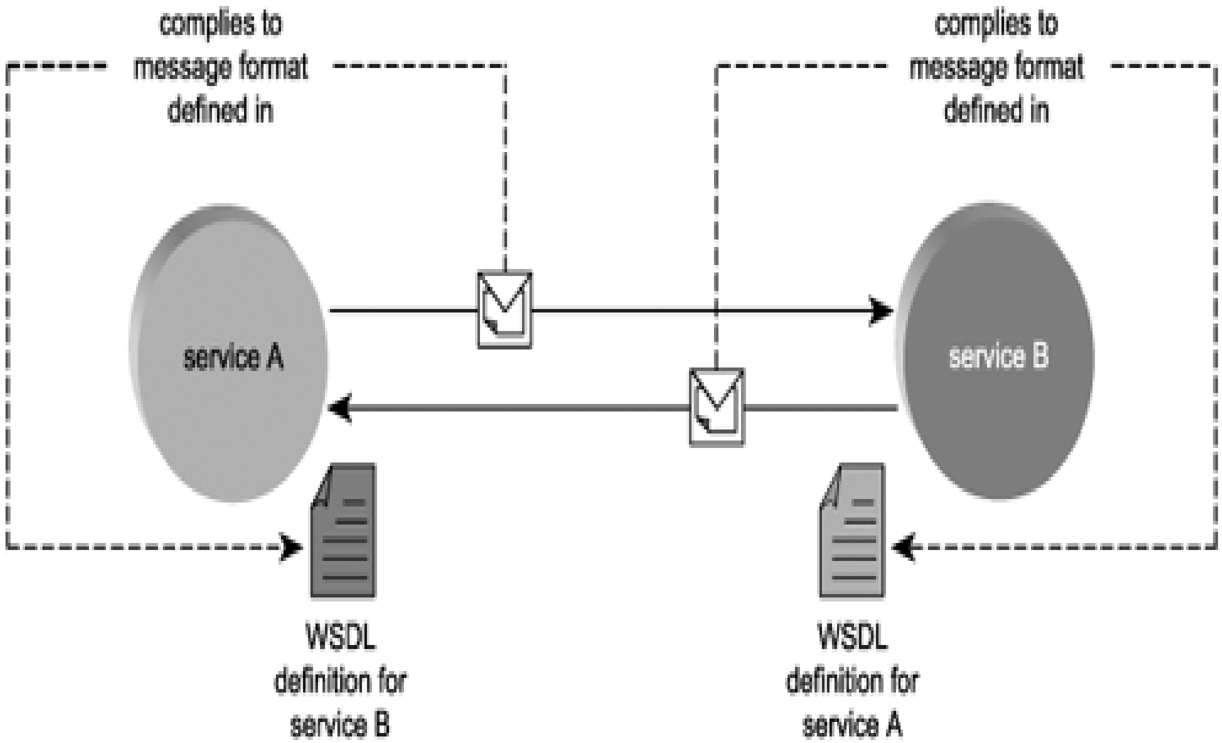
Sect. 5.3 Service Descriptions (with WSDL)

- Web Service sLanguage (WSDL)
  - Abstract Description:
    - API-like interface definition
  - Concrete Description:
    - Service end points: protocol and physical location of service
- XML Schema Definition (XSD)
  - Data type definitions
- Policy Statement (e.g. Service Level Agreements SLA)
  - More in chapter 7

Service descriptions (with WSDL)

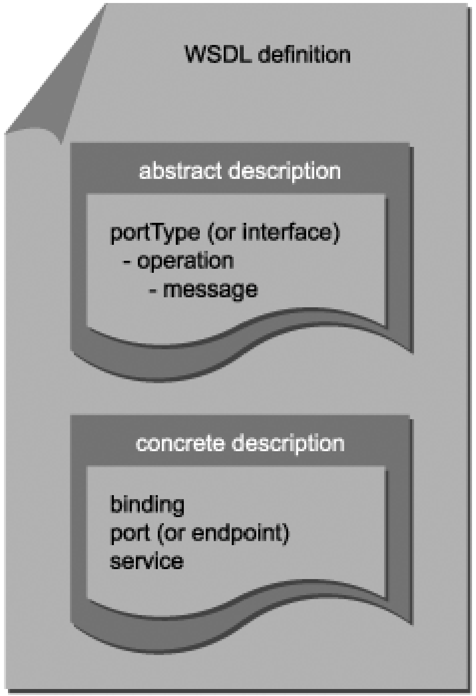
Figure 5.14. WSDL definitions enable loose coupling between services.

[View full size image]



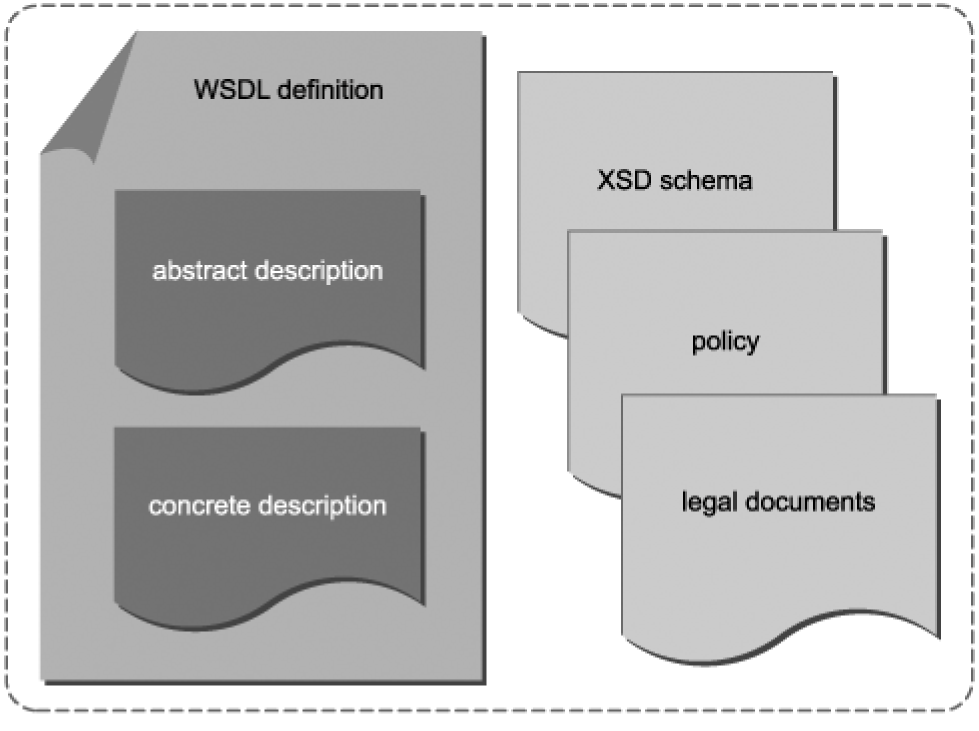
WSDL

Figure 5.16. WSDL document consisting of abstract and concrete parts that collectively describe a service endpoint.



Metadata and service contracts

Figure 5.17. A service contract comprised of a collection of service descriptions and possibly additional documents.



Semantic descriptions

- Service semantics include:
  - how a service behaves under certain conditions
  - how a service will respond to a specific condition
  - what specific tasks the service is most suited for

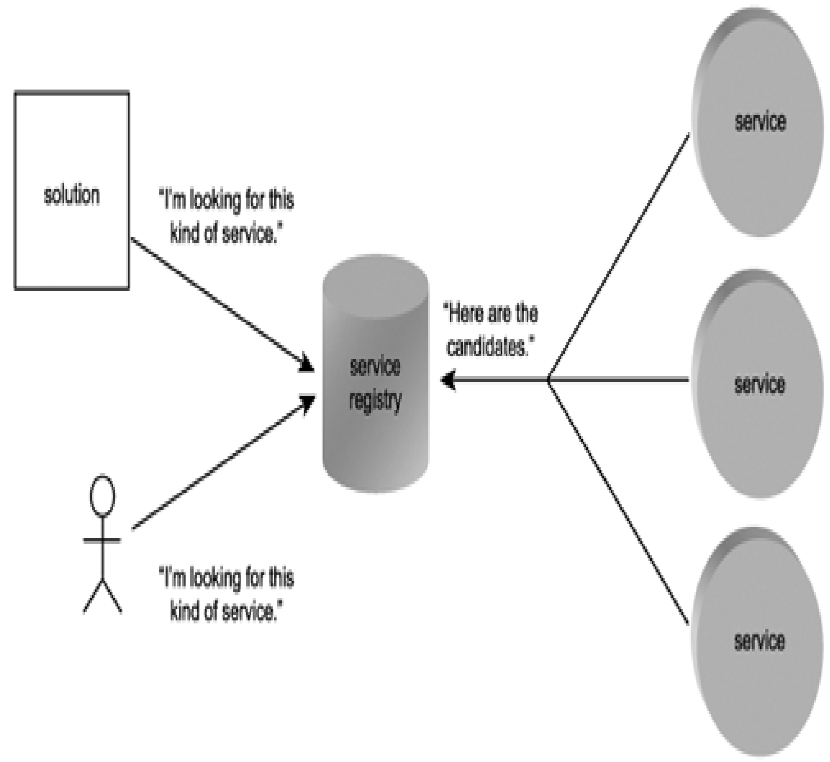
Service description advertisement and discovery

- Central directories and registries become an **option** to keep track of the many service descriptions that become available. These repositories allow humans (and even service requestors) to:
  - locate the latest versions of known service descriptions
  - discover new Web services that meet certain criteria
- UDDI provides us with a registry model

Private and public registries

Figure 5.18. Service description locations centralized in a registry.

[View full size image]



- **Public registries** accept registrations from any organizations, regardless of whether they have Web services to offer. Once signed up, organizations acting as service provider entities can register their services.
- **Private registries** can be implemented within organization boundaries to provide a central repository for descriptions of all services the organization develops, leases, or purchases.

Business entities and business services

- Each public registry record consists of a **business entity** containing basic profile information about the organization (or service provider entity). Included in this record are one or more *business service areas*, each of which provides a description of the services offered by the business entity. Business services may or may not be related to the use of Web services.

Section 5.4 Messaging with SOAP

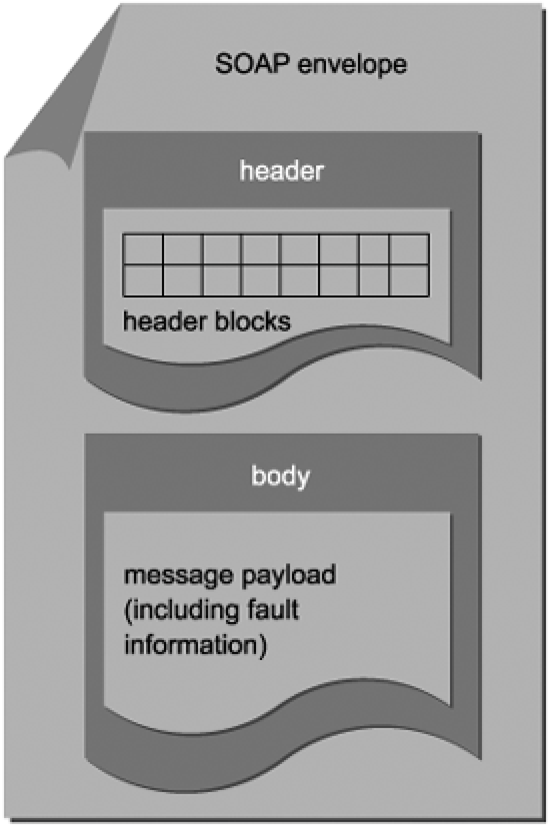
- Messages
- Header Block
- Message Styles (page 146)
  - RPC style
  - Document style
- Attachments
- Message Paths
  - Static
  - Dynamic

Messages

- **Simple Object Access Protocol**, the **SOAP**
  - a standard message format.
  - The structure of this format is quite simple,
  - to be extended and customized
  - driving force behind many of the most significant features of contemporary SOAs.

Envelope, header, and body

Figure 5.21. The basic structure of a SOAP message.



Header Blocks

- include:
  - processing instructions that may be executed by service intermediaries or the ultimate receiver
  - routing or workflow information associated with the message
  - security measures implemented in the message
  - reliability rules related to the delivery of the message
  - context and transaction management information
  - correlation information (typically an identifier used to associate a request message with a response message)

Message styles

- The SOAP specification was originally designed to replace proprietary RPC protocols by allowing calls between distributed components to be serialized into XML documents, transported, and then de-serialized into the native component format upon arrival.
- This RPC-style message runs contrary to the emphasis SOA places on independent, intelligence-heavy messages.
- SOA relies on document-style messages to enable larger payloads, coarser interface operations, and reduced message transmission volumes between services.

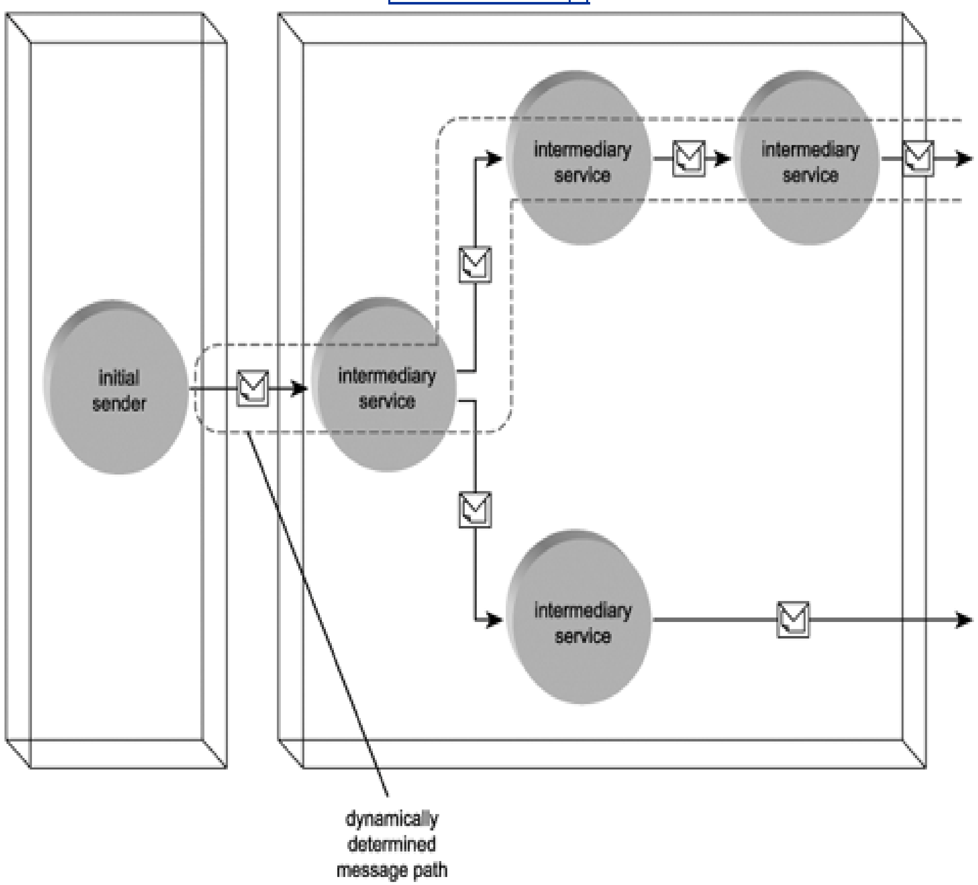
Attachments

- To facilitate requirements for the delivery of data not so easily formatted into an XML document,
- SOAP attachment technologies exist.
- Each provides a different encoding mechanism used to bundle data in its native format with a SOAP message.
- SOAP attachments are commonly employed to transport **binary** files, such as **images**.

Message paths

Figure 5.27. A message path determined at runtime.

[View full size image]



- A message path refers to the route taken by a message from when it is first sent until it arrives at its ultimate destination.
- Therefore, a message path consists of at least one initial sender, one ultimate receiver, and zero or more intermediaries (Figure 5.26).
- Mapping and modeling message paths becomes an increasingly important exercise in SOAs, as the amount of intermediary services tends to grow along with the expansion of a service-oriented solution.
- Design considerations relating to the path a message is required to travel often center around performance, security, context management, and reliable messaging concerns.