# Architecture

The document describes the Web Services Architecture. It classifies the functional components and defines

the relationships among those components to affect the desired properties of the overall architecture.

Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically transported using HTTP with an XML serialization .

Web services provide a standard means of interoperating between altered software applications, running on a variety of platforms and/or frameworks. This document is intended to provide a common definition of a Web service, and define its place within a larger Web services framework to guide the developers. The architecture does not attempt to specify how Web services are implemented, and imposes no restriction on how Web services might be combined. The Web services architecture is an interoperability architecture: it identifies those global elements of the global Web services network that are required in order to ensure interoperability between Web services.

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network

Basic over view of our Approach

Service

Agent call

Data base

Logger

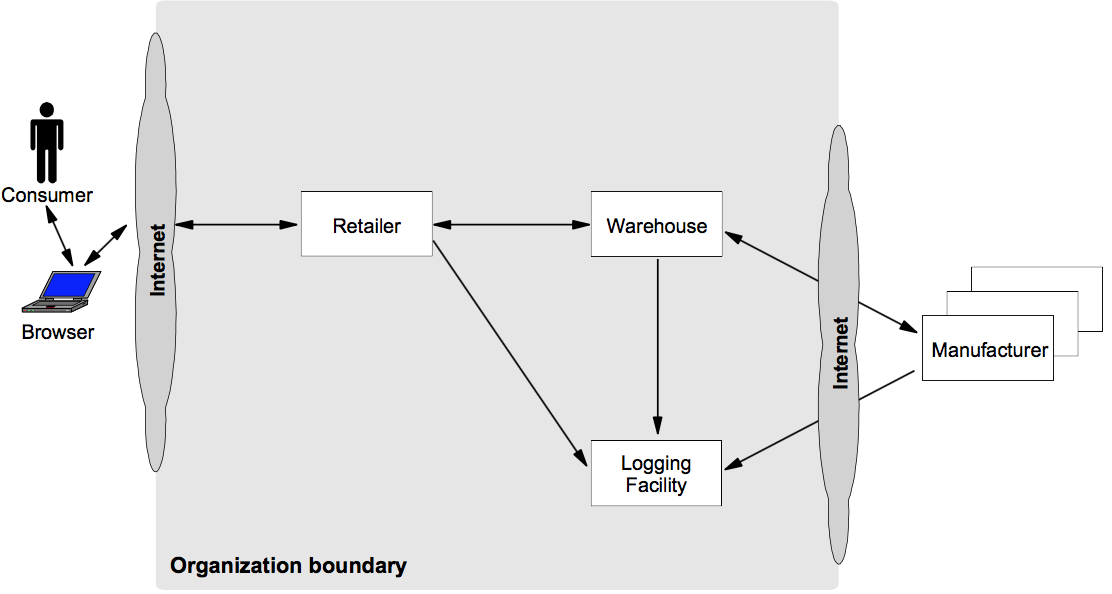
Organizational call

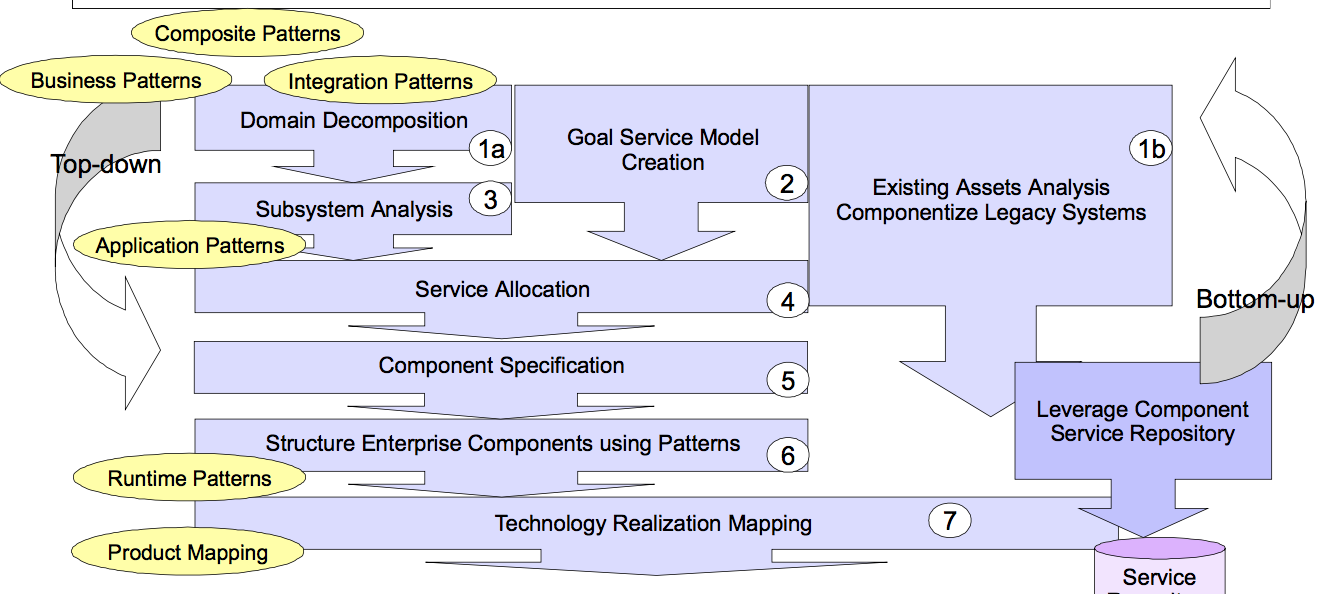
Control

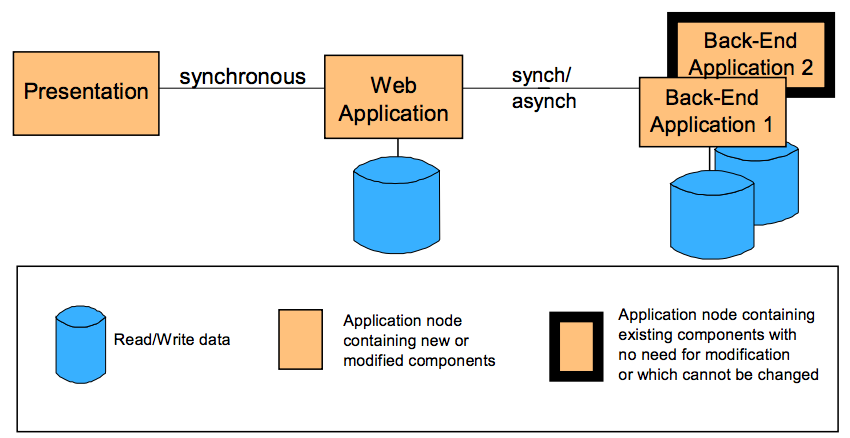
Request

Describe the data

Log calls

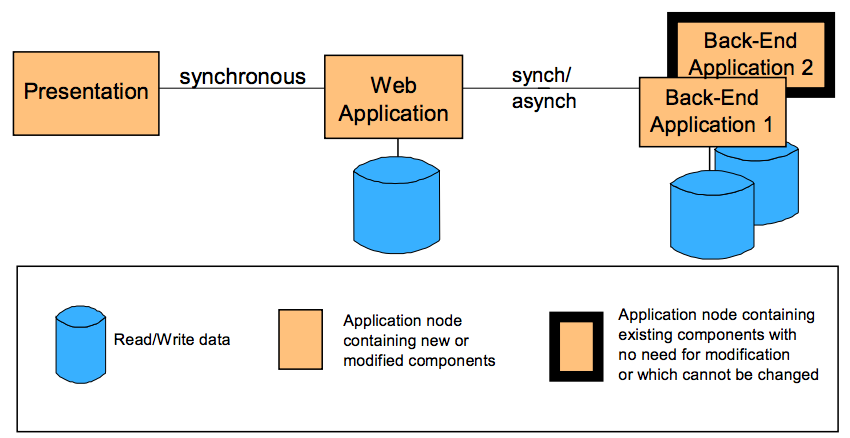






Process flow:

Outside World Demilitarized Zone Internal Network



Protocol Firewall

Existing Applications and Data

Directory and Security Services

Web Application Server

Domain Firewall

User

Domain Name Server

Public Key Infrastructure

Web Server:

Windows 2008 + SP3  WebSphere Application Server V5.0 HTTP

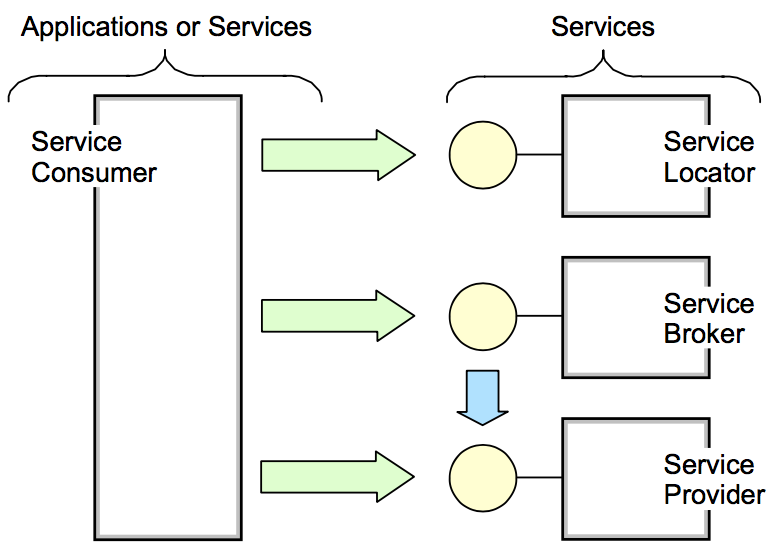
Application Server

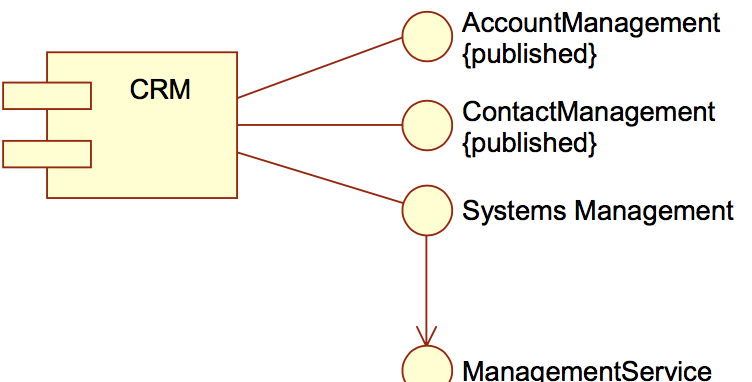
Windows 2008 + SP3  WebSphere Application Server V5.0

Directory and Security Services

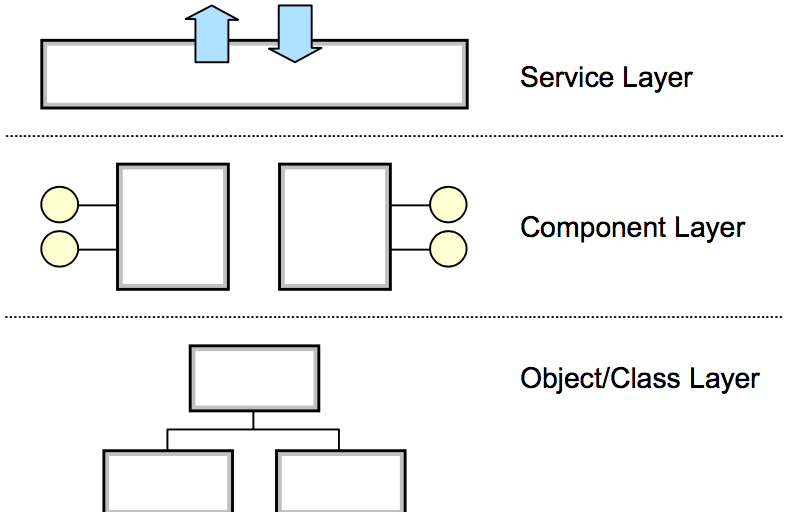
LDAP

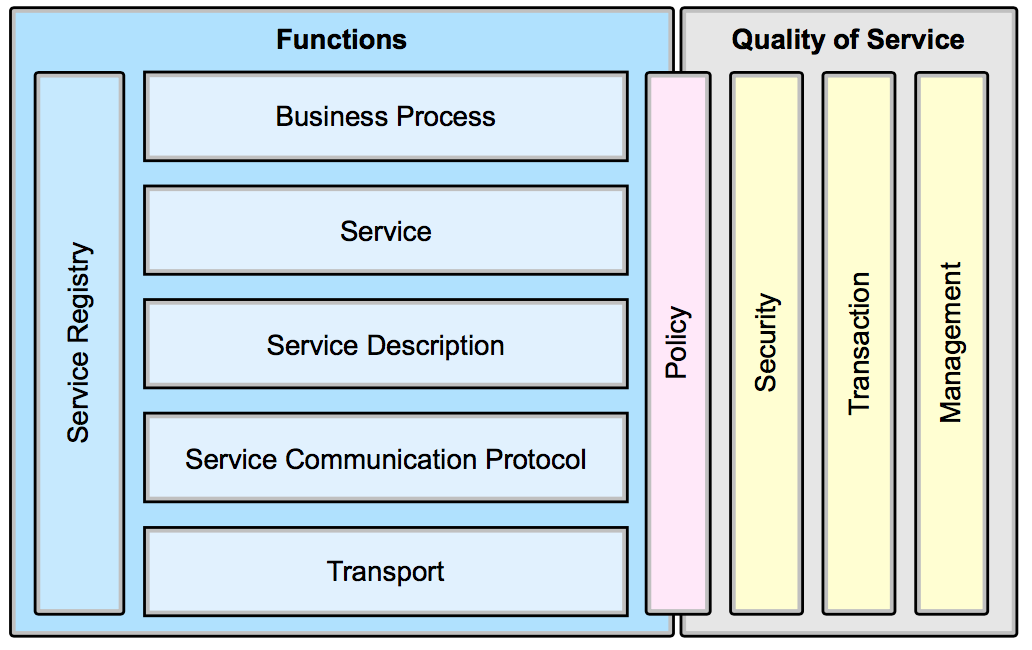
SecureWay Directory





**Layered application architectures**





Explanation of terms :

The architectural stack is divided into two halves, with the left half addressing the functional aspects of the architecture and the right half addressing the quality of service aspects. Refer my previous photo graph about the core architecture concepts. These elements above are described in detail as follows:

Functional aspects include:

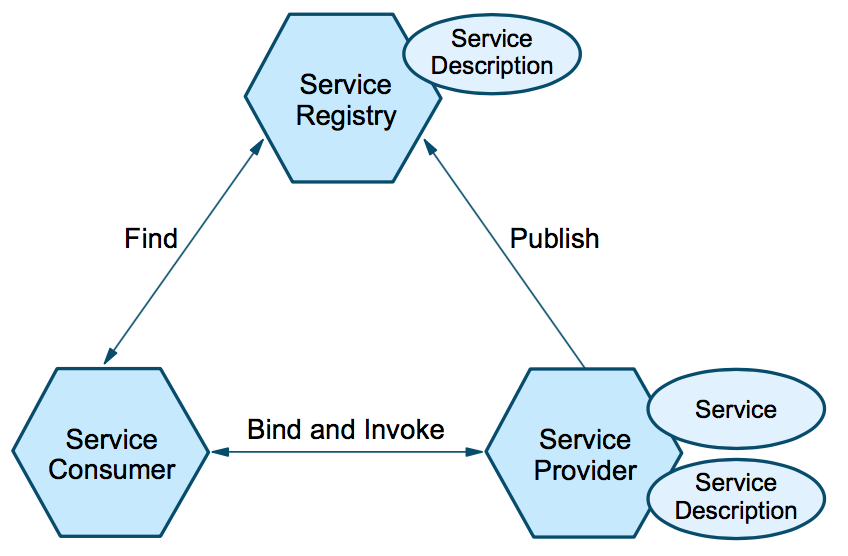
* –  *Transport* is the mechanism used to move service requests from the service consumer to the service provider, and service responses from the service provider to the service consumer.
* –  *Service Communication Protocol* is an agreed mechanism that the service provider and the service consumer use to communicate what is being requested and what is being returned.
* –  *Service Description* is an agreed schema for describing what the service is, how it should be invoked, and what data is required to invoke the service successfully.
* –  *Service* describes an actual service that is made available for use.
* –  *Business Process* is a collection of services, invoked in a particular sequence with a particular set of rules, to meet a business requirement. Note that a business process could be considered a service in its own right, which leads to the idea that business processes may be composed of services of different granularities.
* –  The *Service Registry* is a repository of service and data descriptions which may be used by service providers to publish their services, and service

consumers to discover or find available services. The service registry may provide other functions to services that require a centralized repository.

Quality of service aspects include:

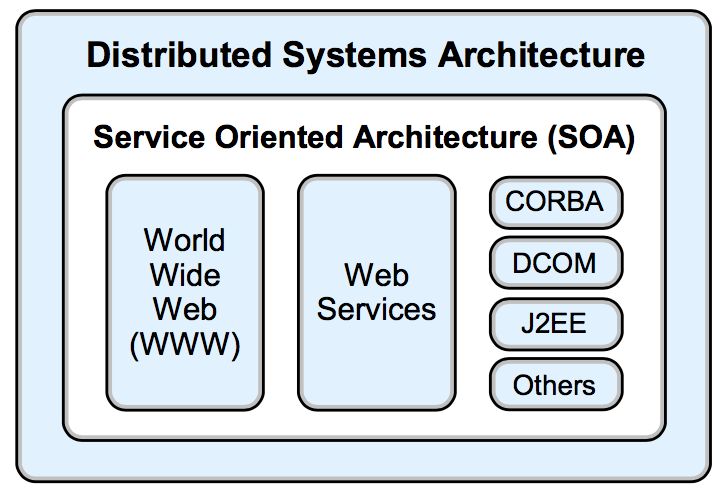
* –  *Policy* is a set of conditions or rules under which a service provider makes the service available to consumers. There are aspects of policy which are functional, and aspects which relate to quality of service; therefore we have the policy function in both functional and quality of service areas.
* –  *Security* is the set of rules that might be applied to the identification, authorization, and access control of service consumers invoking services.
* –  *Transaction* is the set of attributes that might be applied to a group of services to deliver a consistent result. For example, if a group of three services are to be used to complete a business function, all must complete or none must complete.
* –  *Management* is the set of attributes that might be applied to managing the services provided or consumed.

For ease of understanding we have the below figure:



The roles in a service-oriented architecture are:

* Service consumer: The service consumer is an application, a software module or another service that requires a service. It initiates the enquiry of the service in the registry, binds to the service over a transport, and executes the service function. The service consumer executes the service according to the interface contract.
* Service provider: The service provider is a network-addressable entity that accepts and executes requests from consumers. It publishes its services and interface contract to the service registry so that the service consumer can discover and access the service.
* Service registry: A service registry is the enabler for service discovery. It contains a repository of available services and allows for the lookup of service provider interfaces to interested service consumers.  Each entity in the service-oriented architecture can play one (or more) of the three roles of service provider, consumer and registry.  The operations in a service-oriented architecture are:
* Publish: To be accessible, a service description must be published so that it can be discovered and invoked by a service consumer.
* Find: A service requestor locates a service by querying the service registry for a service that meets its criteria.
* Bind and invoke: After retrieving the service description, the service consumer proceeds to invoke the service according to the information in the service description.  The artifacts in a service-oriented architecture are:
* Service: A service that is made available for use through a published interface that allows it to be invoked by the service consumer.
* Service description: A service description specifies the way a service consumer will interact with the service provider. It specifies the format of the request and response from the service. This description may specify a set of preconditions, post conditions and/or quality of service (QoS) levels.  In addition to dynamic service discovery and definition of a service interface contract, a service-oriented architecture has the following characteristics:
* Services are self-contained and modular.
* Services support interoperability.
* Services are loosely coupled.
* Services are location-transparent.



We need to create the DAO as below:

AccessType

+ getOrderNum ( )

+ setOrderNum ( )

+ getRef ( )

+ setRef ( )

ReferenceType

+getValue ( )

+ setValue ( )

Order

+getOrderID( )

+ setOrderID ( )

+getOrderQty( )

+setOrderQty( )

+ getOrderPrice ( )

+ setOrderPrice ( )

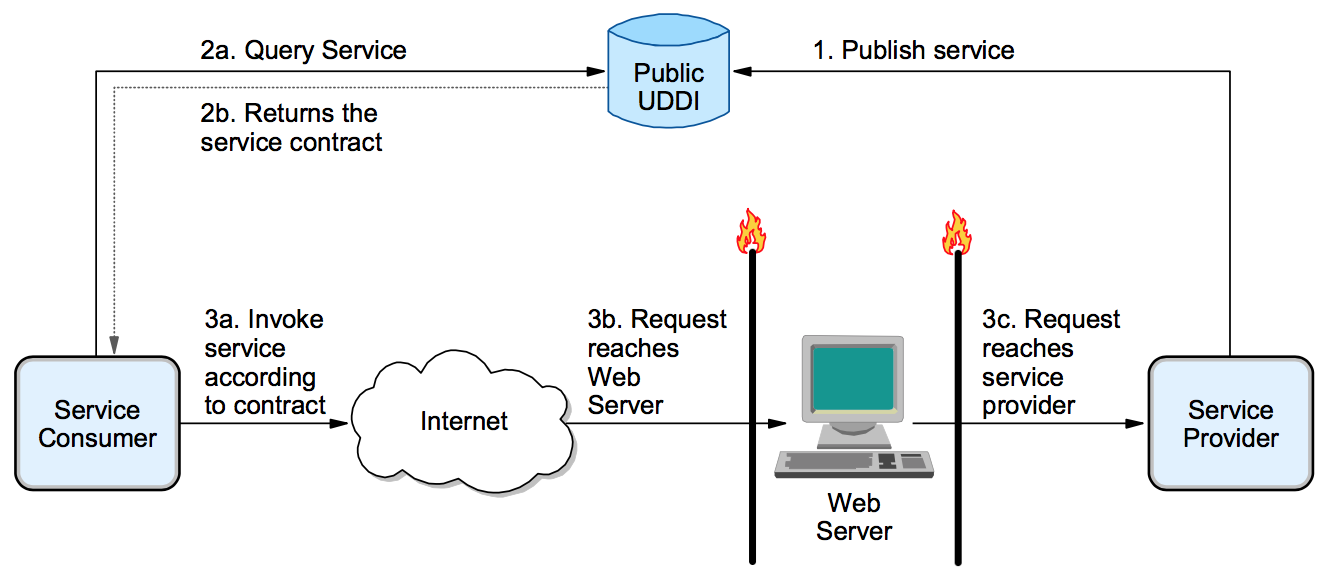
information about the list of ads ordered and the total amount of the ads; the access component provides functions to obtain information about the quantity and price of the ads needed. The implementation of each component is encapsulated behind the interface thus making the API calls easy. So, a user of the Ad ordering component does not know the schema of the Ad Order table and the algorithm for calculating price, rebates and/or discounts on the total amount of the order are all unknown at the given point.

In a service-oriented design, services are not designed based on business entities and invoice. Instead, each service is a holistic unit that manages agnostic operations across a set of business entities. For example, a customer service will respond to any request from any other system or service that needs to access customer information. The customer service can process a request to update customer information; add, update, delete ad portfolios; and enquire about the customer’s order history. The customer service owns all the data related to the customers it is managing and is capable of making other service inquiries on behalf of the calling party in order to provide a unified customer service view. This means a service is a *manager* object that creates and manages its set components. We will disallow creating the objects of interlinked access objects by using Interface Design pattern.

Our aim will be to create :

Web services = XML + transport protocol (such as HTTP)

As below:



We will use Spring as MVC and connect to basic MYSQL at first with the table structure of Access, Users, ETC ( refer photo sent on the discussion)

And host the Web Server on the Apache using WAS or liberty profile using the Parallel Process application pattern, extending the basic serial process orchestration capability provided by the Serial Process application pattern by supporting parallel (concurrent) execution of the sub-processes. Thus making sure that our server remains fault tolerant

**Application Server/Services**

The Application Server/Services node provides the infrastructure for application logic and can be part of a Web application server. It is capable of running both presentation and business logic, but generally does not serve HTTP requests. When used with a Web server redirector, the Application Server/Services node can run both presentation and business logic. In other situations, it can be used for business logic only. The Application Server/Services node supports hosting of Web services applications.

Applications may also rely on services provided by their hosting server to interact with other applications. Examples of services provided by the Application Server/Services node include:

* A TCP/IP pipe established using the hosting operating system
* A servlet or EJB invoked by WebSphere Application Server
* The JMS or J2EE Connector APIs provided by WebSphere
* **Connector**
* Connectors provide the connectivity between two components. A connector is always present to facilitate interaction between two components.  Depending on the required level of detail, a connector can be:

The rules repository contains the rules generally used to control the mode of operation of an interaction, depending on external factors. Examples of such rules are:

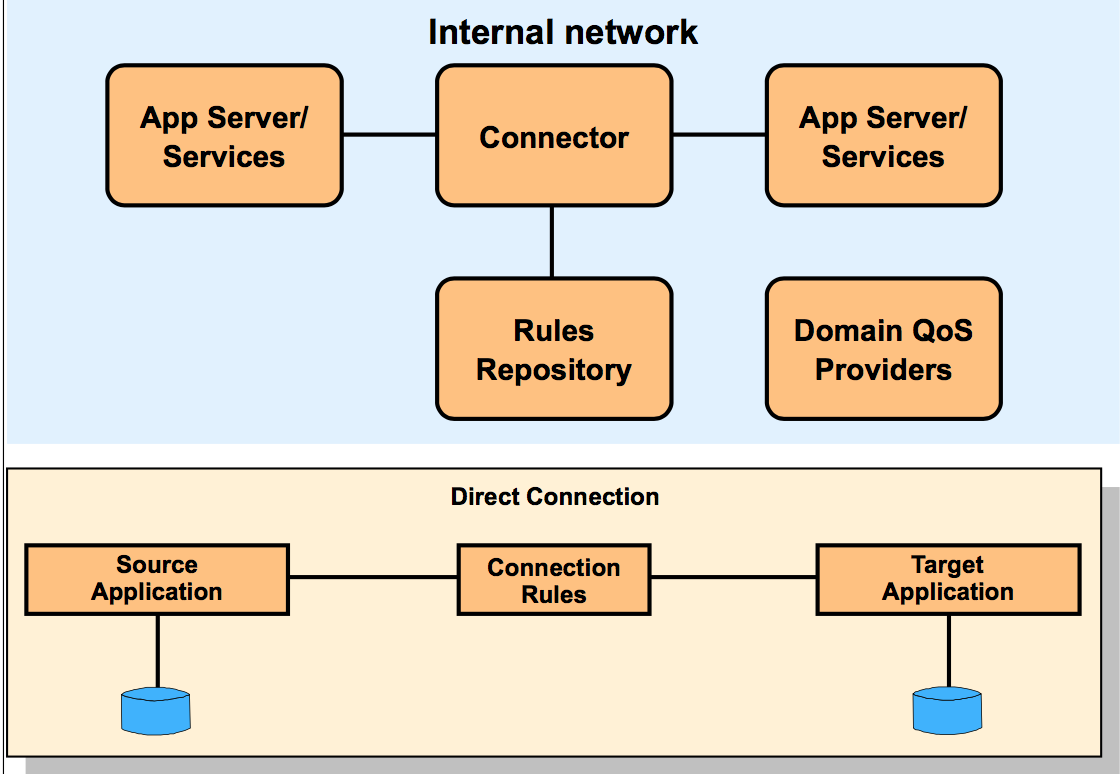
Business data mapping rules (for adapter connectors)

Autonomic rules (such as priority in a shared environment)

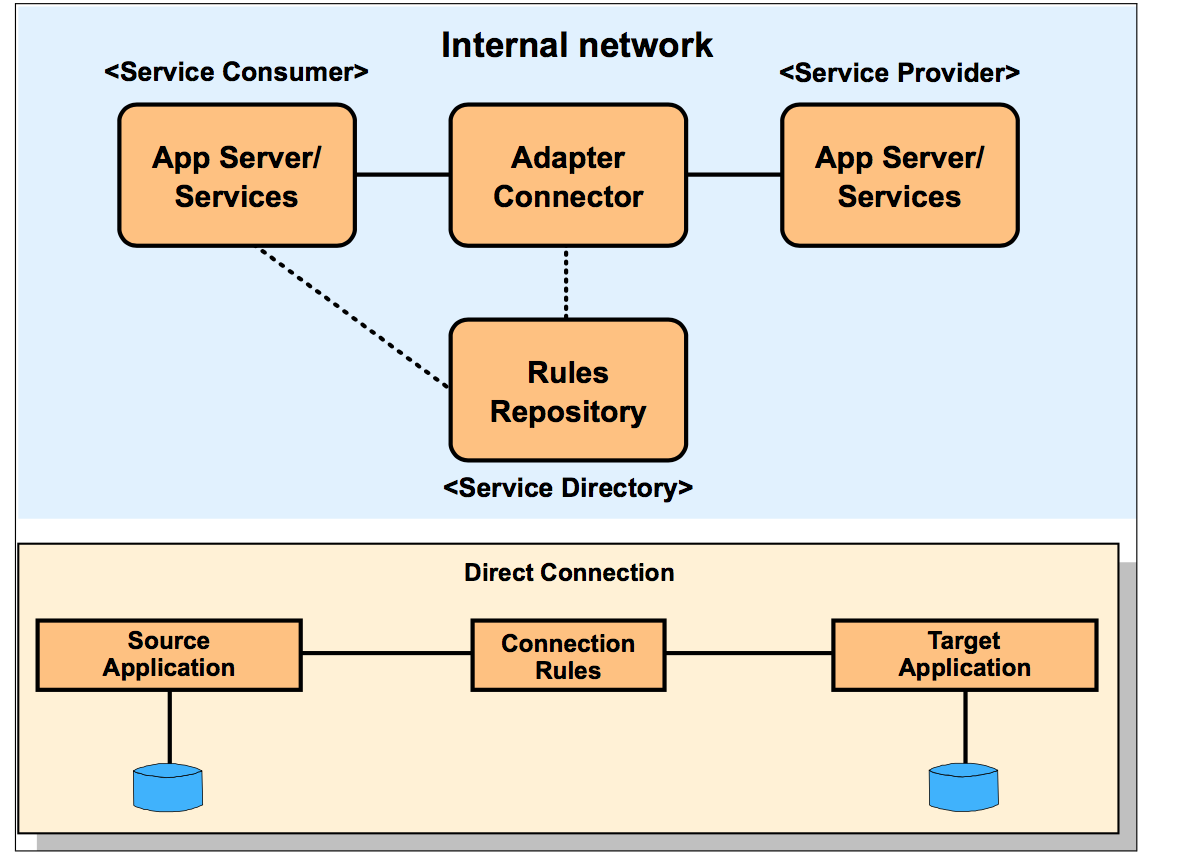
Security rules

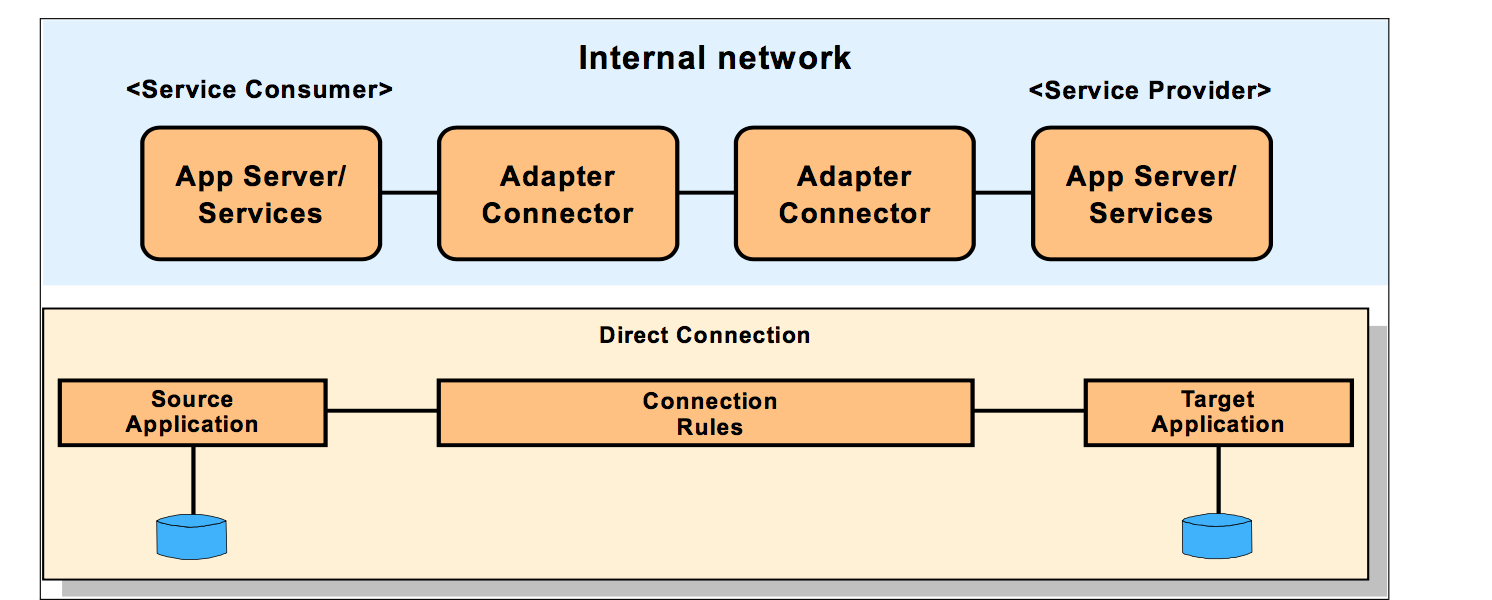
Capacity and availability rules which we define from time to time

The rules repository may or may not exist presently. If it does exist, it could still be left off the Runtime pattern, for example, when analysis determines that interaction rules are not an important part of the solution.



a





We will provide the business rules in the following format to help create the Interfaces and the Business validation cases:

Intially we have the following to be completed

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use case name** | **Description** | **Invoker** | **Implemented by** | **Business or Integration pattern** |
| UC1: Purchase ads | Customer selects and purchases ads from our catalogue | customer | Service team | Self-Service pattern |
| UC2: Source ads | Admin invokes this to source ads from various users and the URL | Admin | Admin team | Application Integration |
| UC3:Create Data base | When ads in a data base falls below a threshold, a resubmit is issued | Admin | Admin team | Application Integration |
| UC4: Supply ready ads | Response to a play ads case | Admin | Admin Team | Extended Enterprise pattern |
| UC6: Configure and run ads | Allows sample application to be run for different technical scenarios | Demo user | GUI and Service Team | Self Service pattern |
| UC7: Log events | Track activities performed by different system users | Admin , Customer, Merchants | Logging facility | Application Integration and Extended Enterprise |
| UC8: View events | View activities recorded in the log | Demo user | Logging facility | Self Service |