SOC

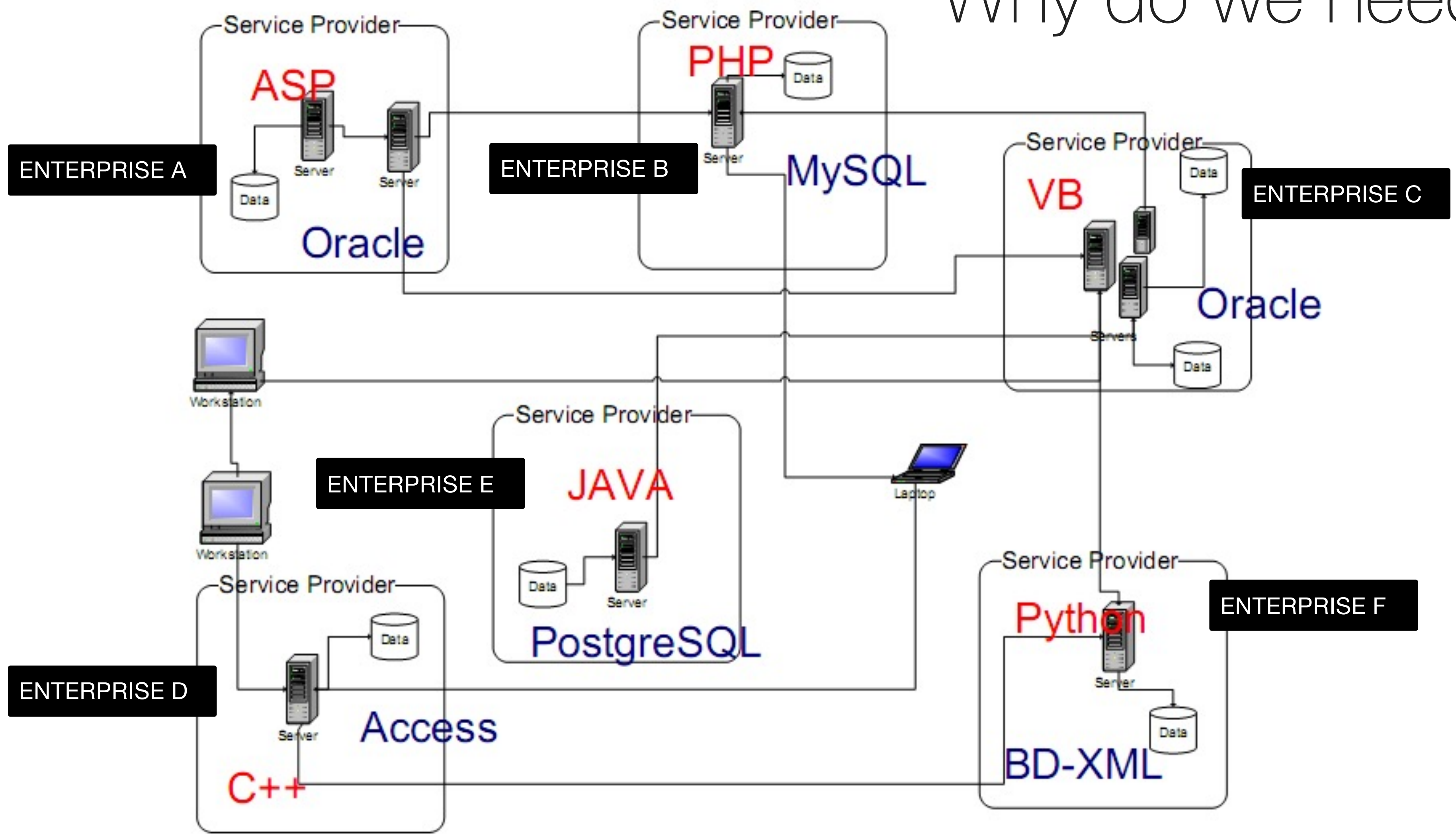
LECTURE NOTES

Service Oriented computing
Computing

By, **Dr. Neila Ben Lakhal**
Ph.D from Tokyo Institute of Technology
Assistant Professor @ ENICARTHAGE

Chapter 2. Service oriented architecture (SOA)

Why do we need SOA?



Why do we need SOA?

- Most companies today have a **large**, **heterogeneous** IT infrastructure that:
  - **Keep changing**
  - **Needs to evolve to adopt new technology**
  - **Needs to be connected of that of commercial partners**
  - **Needs to support an increasing number of purposes and goals**

- This was the field of *Enterprise Application Integration* using systems like CORBA.
- However, solutions until now suffered from:
  - **Tightly integrated systems**
  - **Vendor lock-in - Technology lock-in (e.g., CORBA)**
  - **Lack of flexibility and limitations when new technology arises**

SOA definition [computer Dictionary]

"(Service-Oriented Architecture)
The modularization of business functions for greater flexibility and **reusability**. Instead of building monolithic applications for each department, an SOA organizes business software in a granular fashion so that **common** functions can be **used interchangeably** by different departments internally and by external business partners as well."

[http://computer.yourdictionary.com 2007]

SOA Definition [IBM]

“SOA defines a way to make software components **reusable** via **service** interfaces…

Each **service** in an SOA embodies the code and data integrations required to execute a complete business function (e.g., calculating a monthly loan payment).

The **service** interfaces provide **loose coupling**, meaning they can be called with little or no knowledge of how the integration is implemented underneath.

The **services** are **exposed** using standard network protocols—such as **SOAP**/HTTP or JSON/HTTP—to send requests to read or change data.

The services are **published** in a way that enables developers to quickly find them and **reuse** them to assemble new applications.

These services can be built from scratch but are often created by exposing functions from legacy systems of record as **service** interfaces.”

By: IBM Cloud Education- 2019

Service ?

“Services represent self-contained business functionalities that can be part of one or more processes, and on the other hand, can be implemented by any technology on any platform.”

Service ?

- Is a logical representation of a repeatable business activity that has a specified outcome (e.g., check customer credit; provide weather data …)
- Is self-contained,
- May be composed of other services,
- Is a **black box** to consumers of the service,

[http://www3.opengroup.org/]

Services ?

- are the key building blocks in an SOA.
- provide access to well described functionality in an easy-to-use and transparent fashion for service consumers.

- are composed of three parts:
  - **Interface**,
  - **Contract** and
  - **Implementation**

Service contract?

- Contracts specify the conditions under which services can be consumed and what consumers can expect from service providers.
- This can include :
    - Guarantees on response times and availability,
    - applied security measures,
    - cost, and so on.

Service implementation?

- Implementation of a service is the actual realization of the business logic of the service.

- There are 3 ways to realize services:
  - Use an existing software.
  - Build the logic or implementation for the service you need from scratch.
  - Combine existing services into a new one.

Service interface?

- **Interface** is what is available to the service consumers.
- There are 2 types of interface:
  - **Proprietary** interfaces: specific to a programming language or system
  - **Standard** interfaces: based on standards and protocols.

Service types

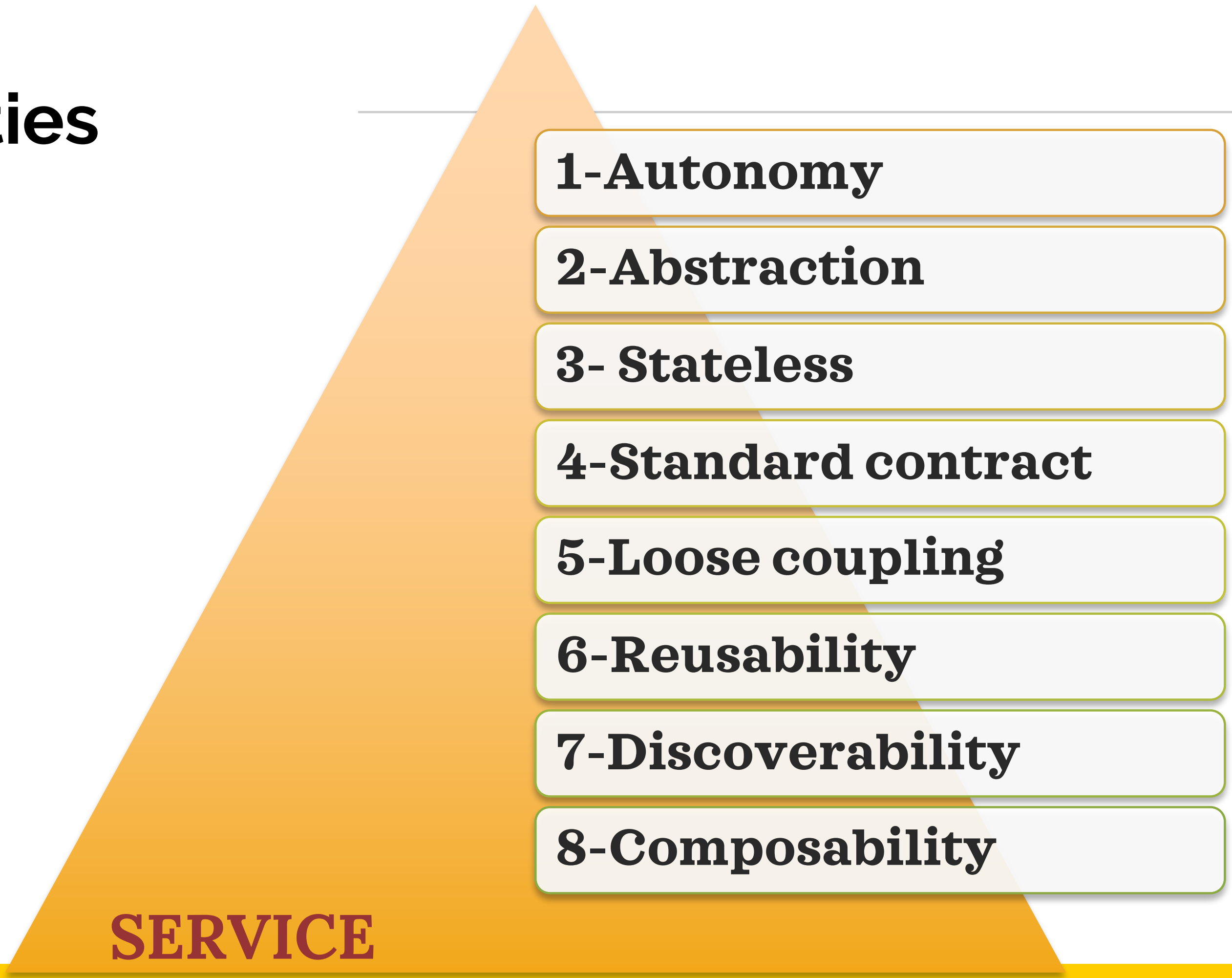
Atomic service

- An atomic service is a well-defined, self-contained function that does not depend on the context or state of other services.
- An atomic service would be seen as **fine-grained** or having a finer **granularity**.

Composite service

- A composite service is an assembly of atomic or other composite services. The ability to assemble services is referred to as composability.
- Composite services are also referred to as compound services.
- A composite service would be seen as **coarse-grained** or having a larger **granularity**.

Service properties



Service key characteristic 1/4

Standardized service contract

◉Each service should have a standardized service contract, consisting of an **interface** and service **description**. Even though we want services to be independent, they have to adhere to a common agreement to enable interoperability and allow the purpose of services to be more easily understood.

◉In order to have **standardized** service contracts, all service contracts should follow a set of **design standards**.

Service loose coupling

◉Services should be loosely coupled and **independent** of each other. Service contracts should be designed to have independence from service consumers and from their implementations.

◉Loosely coupled services can be modified faster and easier.

◉Decoupling service contracts from their implementations allows service contracts to be modified with minimal impact to service consumers and service implementations.

◉By minimizing the dependencies between services, each service can change and evolve independently while minimizing the effects of those changes on other services.

Service key characteristic 2/4

Abstraction

◉Service contracts should only contain information that it is necessary to reveal, and service implementations should also hide their details.

◉Any information that is not essential to effectively use the service should be abstracted out.

◉Design decisions, such as the technology used for a service, can be abstracted away. If a design decision needs to be changed later, the goal is that it can be made with minimal impact.

Reusability

◉Services should be designed with reusability in mind, with their service logic being independent of a particular technology or business process.

◉Service reusability increases:

- Software development teams productivity, leading to savings in both costs and time.
- Organizational agility because organizations can use existing services to respond to new business automation needs.
- New software development and release speed and may allow a feature or product to reach the market faster.
- software quality because existing services have already been tested. They may already be in production and, if there were any defects with the service, they may have already been exposed and corrected.

Service key characteristic 3/4

Autonomy

◉Services should be designed with more independence from their runtime environments.

- This leads to better performance and reliability of those services at runtime.

Statelessness

◉Service designs should minimize the amount of state management that takes place, and **separate** state data from services.

- This will reduce resource consumption and will allow to handle more requests reliably.
- Service statelessness improves service **scalability** and improves the **reusability** of services.

Service key characteristic 4/4

Discoverability

◉Services need to be discoverable:

- By humans who are searching manually
- By software applications searching programmatically.
- By each other to interact.

◉Service developers are required to provide meaningful **metadata** within a service about the purpose of the service and the functionality it provides.

Composability

◉Services should be designed so that they are **composable**. This is the ability to use a service in any number of other services, and those services may themselves be composed of other services.

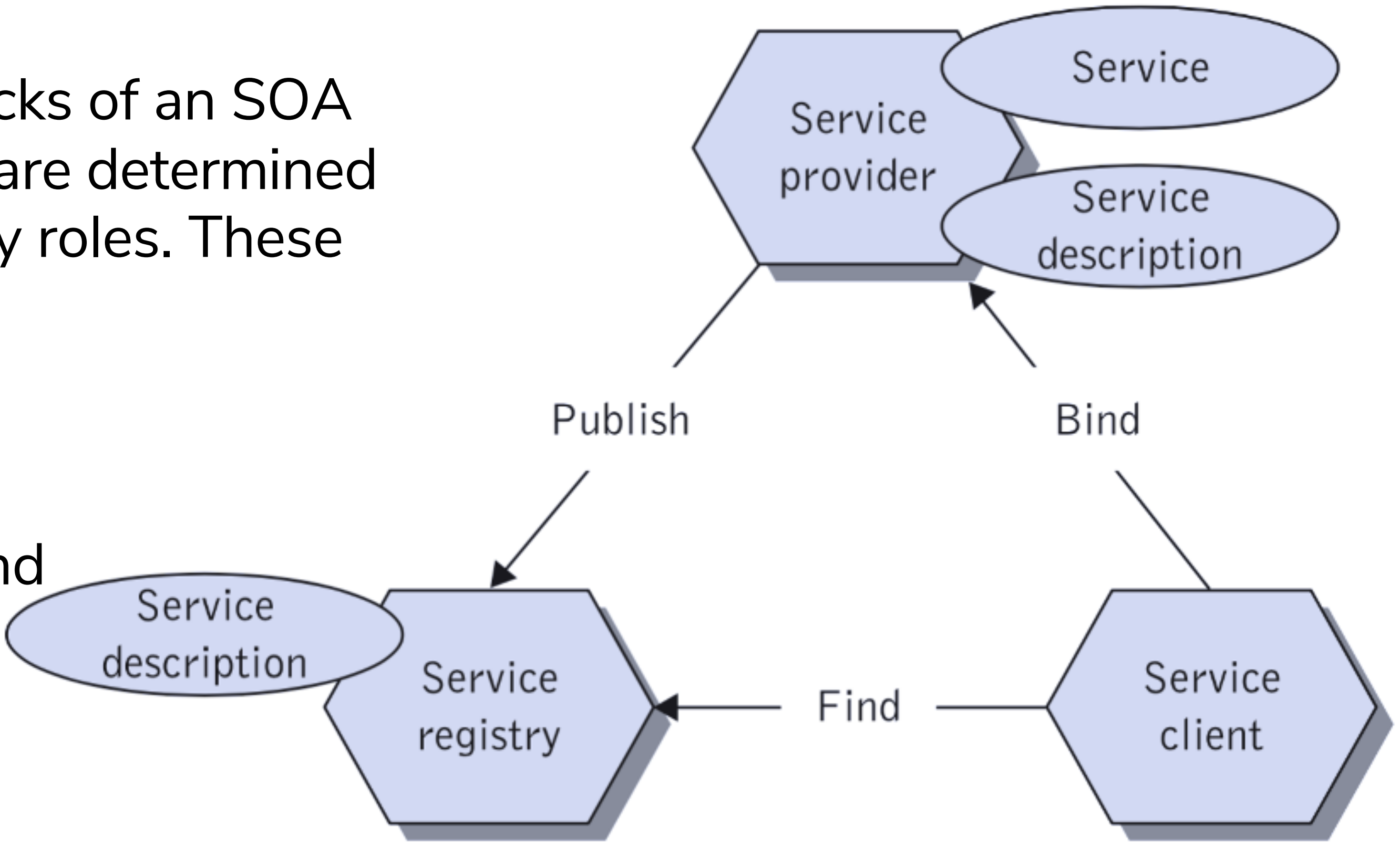
◉Service composition is heavily related to service **reusability**. The ability to create solutions by composing existing services provides organizations with one of the most important SOA benefits: organizational agility.

FOR MORE DETAILS : SOA PRINCIPLES OF SERVICE DESIGN » DE THOMAS ERL.

Roles of interaction in the SOA

- The main building blocks of an SOA are three-fold and they are determined on the basis of 3 primary roles. These are :

- The **service Provider**,
- The **service Registry**, and
- The **service Client**



Service Provider

- From a **business perspective** the **services provider** is the organization that owns the service and implements the business logic that underlies the service.

- From an **architectural perspective** this is the platform that hosts and controls access to the service.

Service Requester

- From a **business perspective** this is the enterprise that requires certain functions to be satisfied.

- From an **architectural perspective**, this is the application that is looking for, and subsequently invoking, the service.

- The **services requestor** searches the **service registry** for the desired services.

Service Registry

- The last important role that can be distinguished in the services architecture is that of the **services registry**, which is a searchable directory where service descriptions can be **published** and **searched**.

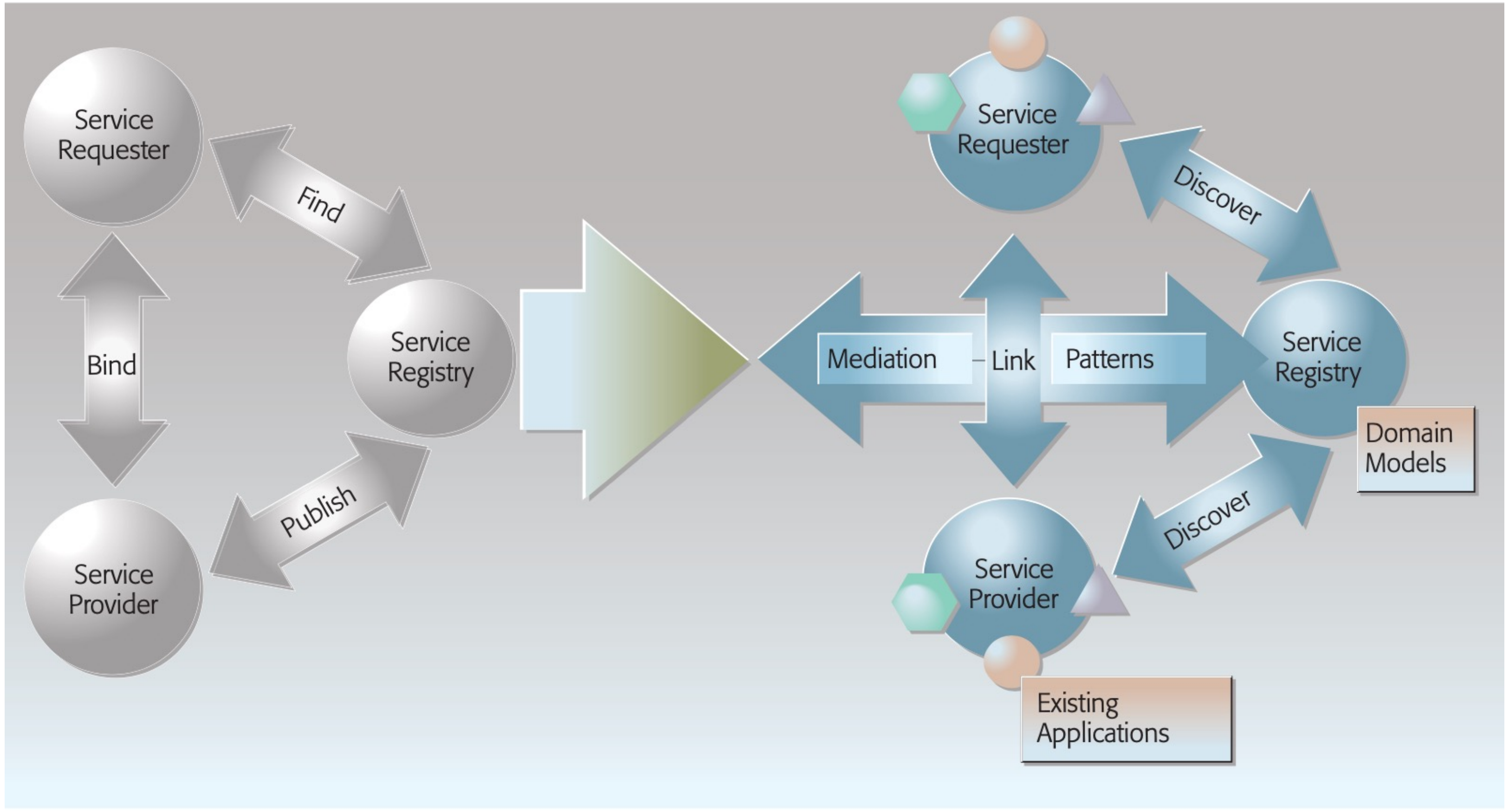
- **Service requestors find** service descriptions in the registry and obtain **binding** information for services.

- This information is sufficient for the **service requestor** to contact, or **bind** to, the **service provider** and thus make use of the services it provides.

Invocation de service

- There are 3 types of service invocation :
  - **Synchronous Invocation** : service requester invokes the service and waits for the service provider's invocation service.
  - **Asynchronous Invocation** : service requester invokes the service and continues processing other tasks. When the service providers finishes the execution it will send a call-back to the service requester.
  - **Invocation via ESB (Entreprise Service Bus)** :
    - ESB provides the tools and runtime infrastructure to realize the promise of SOA formulated in the iconic "publish-find-bind" triangle
    - ESB is a bus-like architecture through which different services are integrated . It enables communication between them.
    - ESB core functionalities: Some core functionalities offered by ESB are: uncoupling, Transport Protocol Conversion, high availability and scalability, Message Transformation, Routing, and Security.

SOA with ESB


Service Requester
Find
Bind
Service Registry
Publish
Service Provider
Service Requester
Discover
Mediation
Link
Patterns
Service Registry
Domain Models
Discover
Service Provider
Existing Applications

Implementing SOA