Service-orientation and Object-orientation

SO Principle	Related Object-Orientation Principles
service reusability	Much of **object-orientation** is geared toward the creation of **reusable classes**. The object-orientation **principle of modularity** standardized decomposition as a means of application design. Related principles, such as **abstraction and encapsulation**, further **support reuse** by requiring a distinct separation of interface and implementation logic. **Service reusability** is therefore a **continuation** of this goal.

Service-orientation and Object-orientation

SO Principle	Related Object-Orientation Principles
service contract	The requirement for a service contract is **very comparable** to the use of interfaces when building object-oriented applications. Much like **WSDL definitions**, **interfaces** provide a means of abstracting the description of a class. And, much like the **"WSDL first"** approach encouraged within **SOA**, the **"interface first"** approach also is considered an **object-orientation** best practice.

Service-orientation and Object-orientation

SO Principle	Related Object-Orientation Principles
service loose coupling	Although the creation of interfaces somewhat decouples a class from its consumers, **coupling** in general is one of the **primary qualities of service-orientation** that **deviates from object-orientation**. The use of **inheritance** and other object-orientation principles encourages a much **more tightly coupled relationship** between units of processing logic when compared to the service-oriented design approach.

Service-orientation and Object-orientation

SO Principle	Related Object-Orientation Principles
service abstraction	The object-orientation principle of abstraction requires that a **class provide an interface** to the external world and that it be accessible via this interface. **Encapsulation** supports this by establishing the concept of **information hiding.** **Service abstraction** accomplishes **much of the same** as object abstraction and encapsulation. Its purpose is to **hide the underlying details of the service** so that only the service contract is available and of concern to service requestors.

Service-orientation and Object-orientation

SO Principle	Related Object-Orientation Principles
service composability	Object-orientation supports association concepts, such as **aggregation** and **composition**. These, within a loosely coupled context, also are supported by service-orientation. For example, the same way a hierarchy of objects can be composed, a **hierarchy of services can be assembled** through service composability.

Service-orientation and Object-orientation

SO Principle	Related Object-Orientation Principles
service autonomy	The quality of autonomy is **more emphasized in service-oriented design** than it has been with object-oriented approaches. Achieving a level of independence between units of processing logic is possible through service-orientation, by leveraging the loosely coupled relationship between services. **Cross-object references** and **inheritance-related dependencies** within object-oriented designs support a **lower degree of object-level autonomy**.

Service-orientation and Object-orientation

SO Principle	Related Object-Orientation Principles
service statelessness	Objects consist of a combination of class and data and are **naturally stateful**. Promoting statelessness within services therefore tends to **deviate** from typical object-oriented design. The principle of statelessness is generally **more emphasized with service-orientation**.

Service-orientation and Object-orientation

SO Principle	Related Object-Orientation Principles
service discoverability	Designing **class interfaces** to be consistent and **self-descriptive** is another **object-orientation best practice.** Discoverability is another principle **more emphasized by the service-orientation** paradigm. It is encouraged that service contracts be as communicative as possible to support discoverability at design time and runtime.

So far,

- Several principles of service-orientation are related to and derived from object-orientation principles.
- Some object-orientation principles, such as inheritance, do not fit into the service-oriented world.
- Some service-orientation principles, such as loose coupling and autonomy, are not directly promoted by object-orientation.

Native Web service support for Service-orientation principles

SO Principle	Web Service Support
service reusability	Web services are **<u>not automatically reusable</u>**. This quality is related to the nature of the logic encapsulated and exposed via the Web service.
service contract	Web services require the use of service descriptions, making service contracts a **<u>fundamental part</u>** of Web services communication.
service loose coupling	Web services are naturally loosely coupled through the use of **<u>service descriptions</u>**.

Native Web service support for Service-orientation principles

SO Principle	Web Service Support
service abstraction	Web services automatically emulate the **black box model** within the Web services communications framework, hiding all of the details of their underlying logic.
service composability	Web services are **naturally composable**. The extent to which a service can be composed, though, generally is determined by the service design and the reusability of represented logic.
service autonomy	To ensure an autonomous processing environment requires design effort. Autonomy is therefore **not automatically provided** by a Web service.

Native Web service support for Service-orientation principles

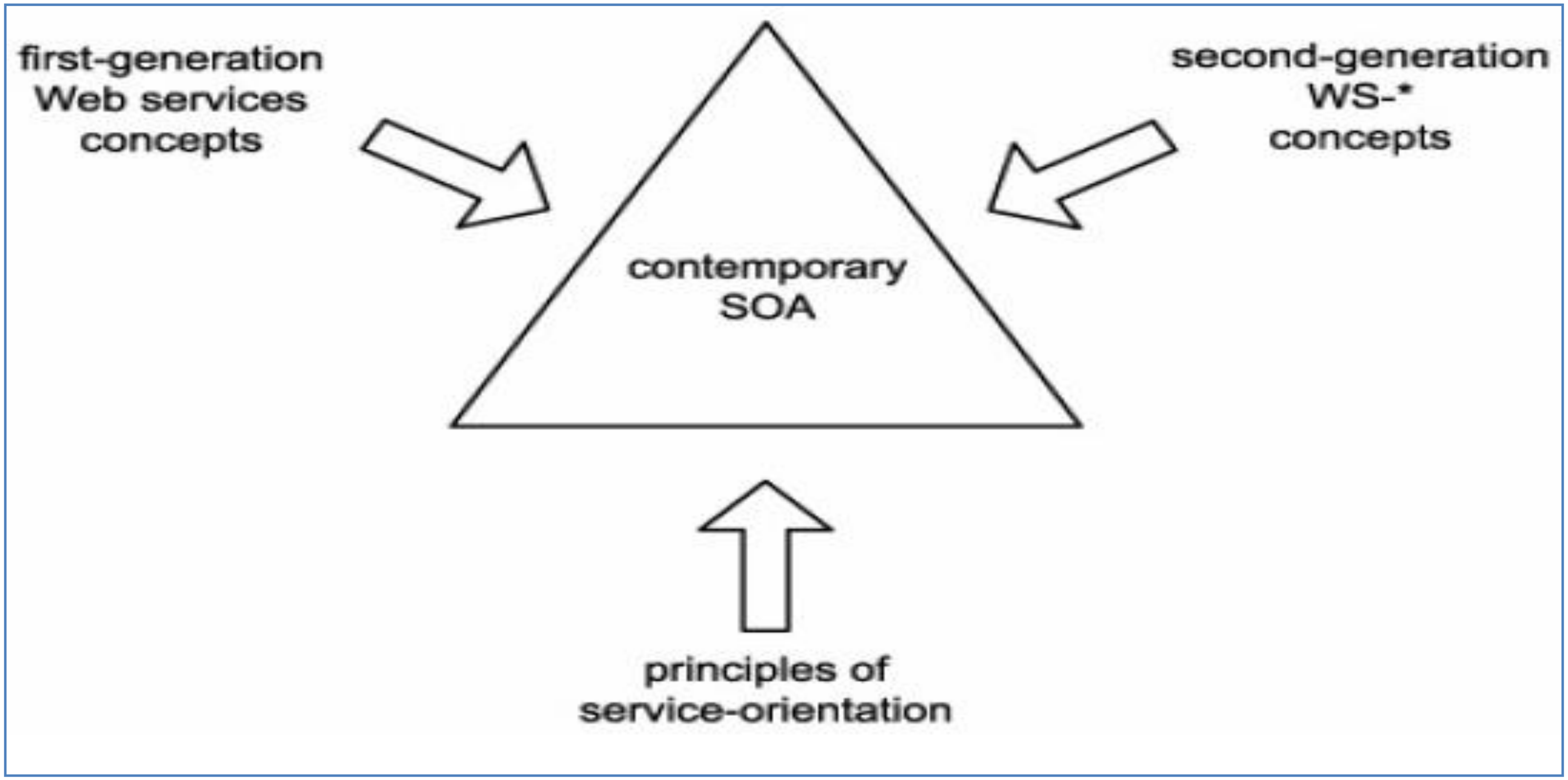
SO Principle	Web Service Support
service statelessness	Statelessness is a **<u>preferred condition</u>** for Web services, strongly supported by many WS-* specifications and the document-style SOAP messaging model.
service discoverability	Discoverability must be **<u>implemented by the architecture</u>** and even can be considered an extension to IT infrastructure. It is therefore **<u>not natively supported</u>** by Web services.

So far,

- Service abstraction, composability, loose coupling, and the need for service contracts are native characteristics of Web services that are in full alignment with the corresponding principles of service-orientation.
- Service reusability, autonomy, statelessness, and discoverability are not automatically provided by Web services. Realizing these qualities requires a conscious modeling and design effort.

Service Orientation and Contemporary SOA

- External Influences that form and Support Contemporary SOA


first-generation Web services concepts
second-generation WS-* concepts
contemporary SOA
principles of service-orientation

Service Orientation and Contemporary SOA

- Most of the contemporary SOA characteristics are supported by external influences, except the following
  - Enterprise wide loose coupling
  - Service oriented business modeling
  - Organizational agility
  - Layers of abstraction
- To support these remaining characteristics, **modeling and design effort is required**

Service Layer Abstraction

- In the enterprise model, Service interface layer is placed between business process and application layers.
- To implement the characteristics in Service interface layers, following questions need to be addressed:
  - What logic should be represented by services?
  - How should services relate to existing application logic?
  - How can services best represent business logic?
  - How can services be built and positioned to promote agility?
- These questions are answered during service oriented analysis phase

What logic should be represented by services?

- To achieve enterprise wide loose coupling
  - Physically **separate layers** of services are needed
  - One set represents business logic and the other represents technology specific application logic
  - Hence each domain can **evolve independently**

How should services relate to existing application logic?

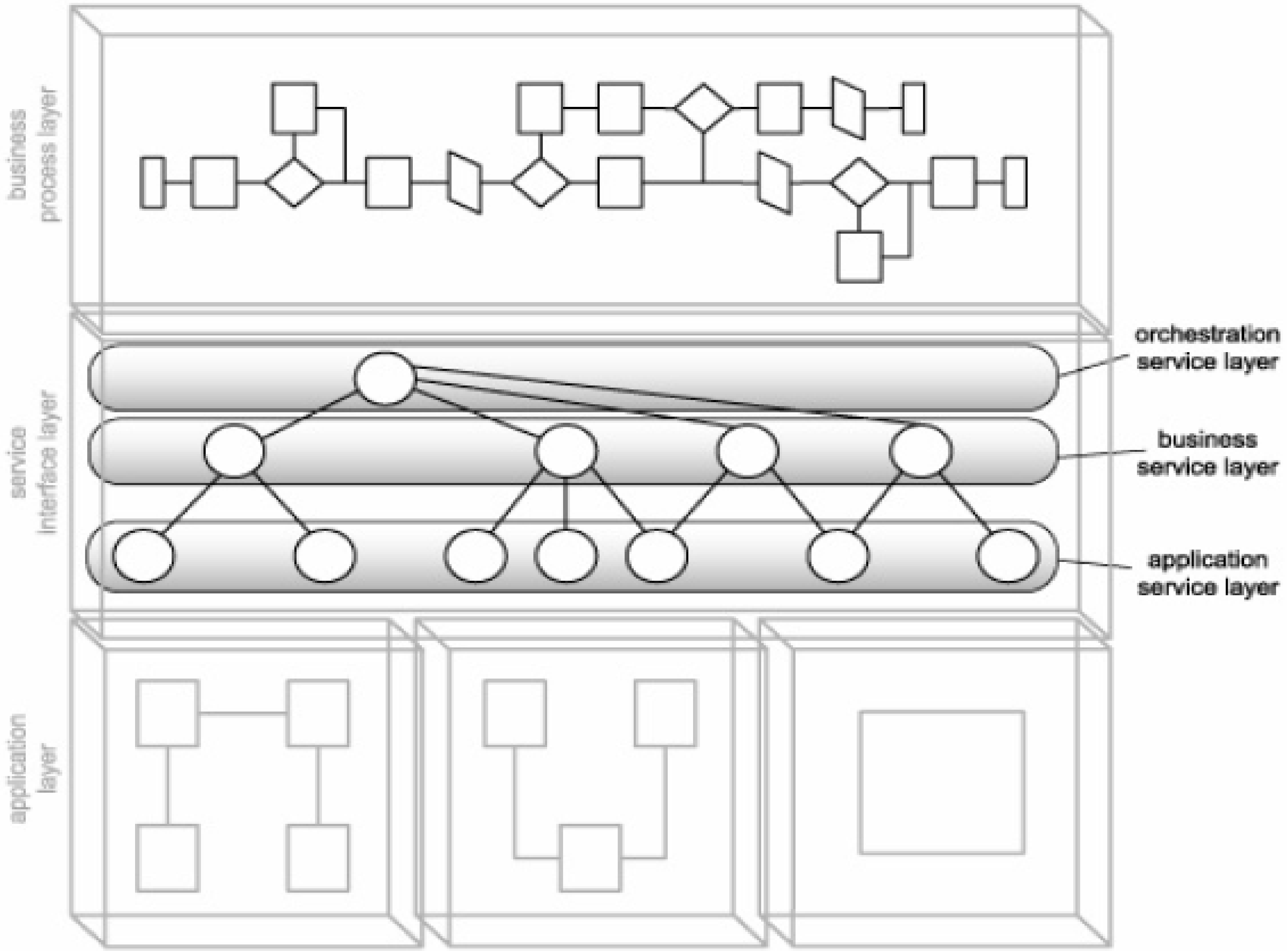
- Depends on
  - **Legacy logic** being exposed
  - **New logic** is being developed
- Applying service layer on **top of legacy** applications may lead to compromising some service orientation principles
- Applying service layer to **new logic** can incorporate service orientation directly
- A separate layer of services – **Application Service Layer** is needed

How can services best represent business logic?

- A separate layer is required to represent business logic – **Business Service Layer**
- To support "Service oriented business modeling"

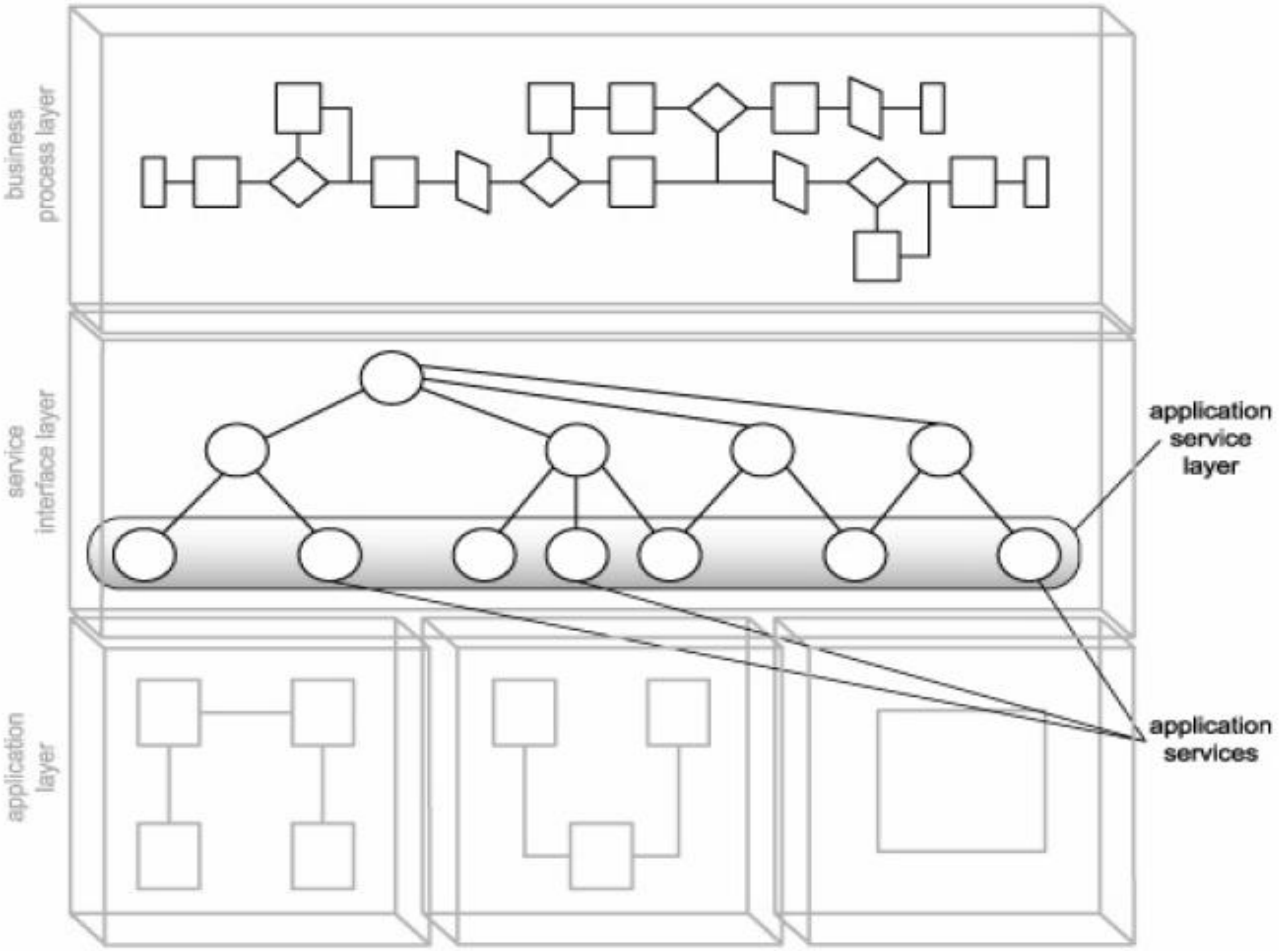
How can service be built and positioned to promote agility?

- Parent controller layer is needed
  - To provide business rules and composition rules
- Orchestration is used for this purpose
- New layer is called **Orchestration Service Layer**


business process layer
service interface layer
application layer
orchestration service layer
business service layer
application service layer

Application Service Layer

- Services in this layer are called application services
- **Characteristics:**
  - Represent specific processing context
  - Use underlying resources
  - Solution agnostic
  - Generic and Reusable
  - Inconsistent in exposing interface granularity


business process layer
service interface layer
application layer
application service layer
application services

Application Service Layer – Service Models

- Utility Service
- Wrapper Service
- Hybrid Service
- Application Integration Service

Application Service Layer – Service Models

- **When a separate business service layer exists**
  - All application services are turned into generic utility services

- **When the business logic does not reside in a separate layer**
  - Hybrid application services are used which contain both application logic and business rules

Application Service Layer – Service Models

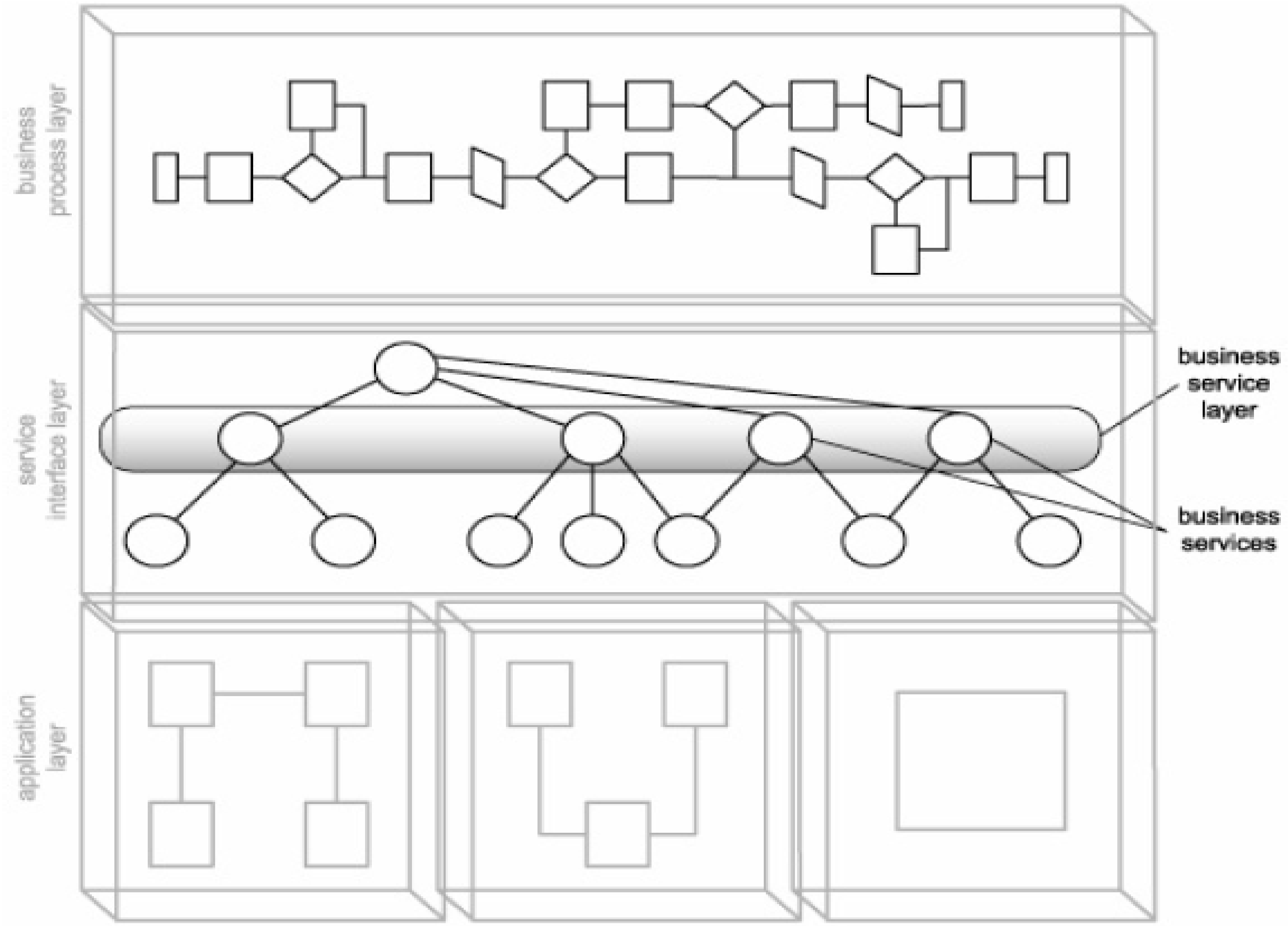
- **When an application service composes others to make coarse grain logic**
  - It is called application integration service

- **When services are used to encapsulate some or all parts of legacy systems**
  - It is called wrapper service (Adapter to interact with legacy application)

Business Service Layer

- Services in this layer are called business services
  - Represent the business logic

- **Service Model:**
  - These services implement Business service model
  - Business services is further subdivided into
    - **Task centric** business service
    - **Entity centric** business service


business process layer
service interface layer
application layer
business service layer
business services

Business Service Layer – Task Centric Business Service

- Encapsulates business logic specific to a task or business process
- Used when orchestration layer does not contain business logic
- Limited reuse potential

Business Service Layer – Entity Centric Business Service

- Encapsulates specific business entity (employee, timesheet)
- Reusable
- Composed by Orchestration Service Layer or Task centric Business service

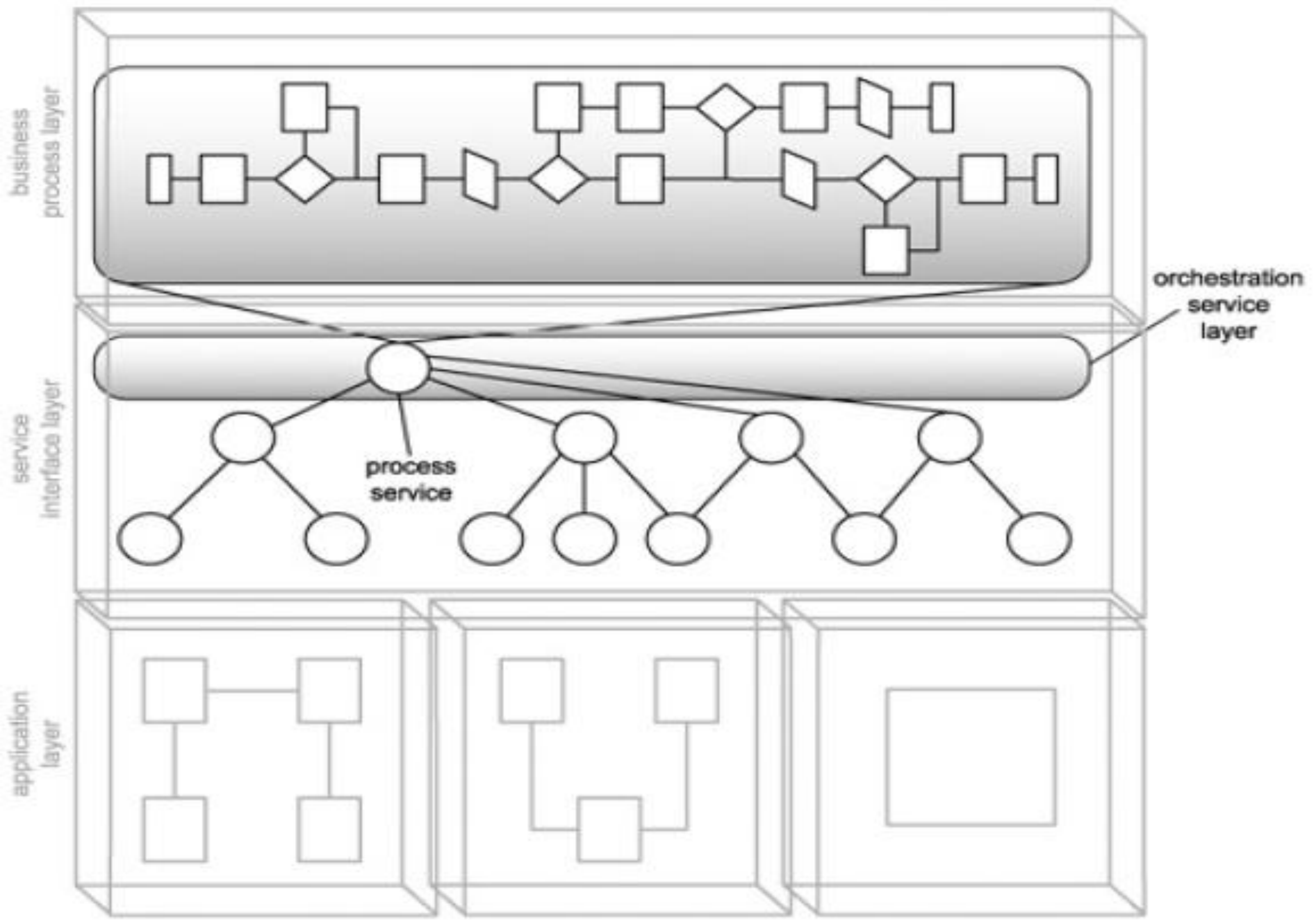
Examples

- Application Service Layer
  - Load Balancer
  - Notification

- Business Service Layer
  - Accounts
  - Purchase Order
  - Ledger

Orchestration Service Layer

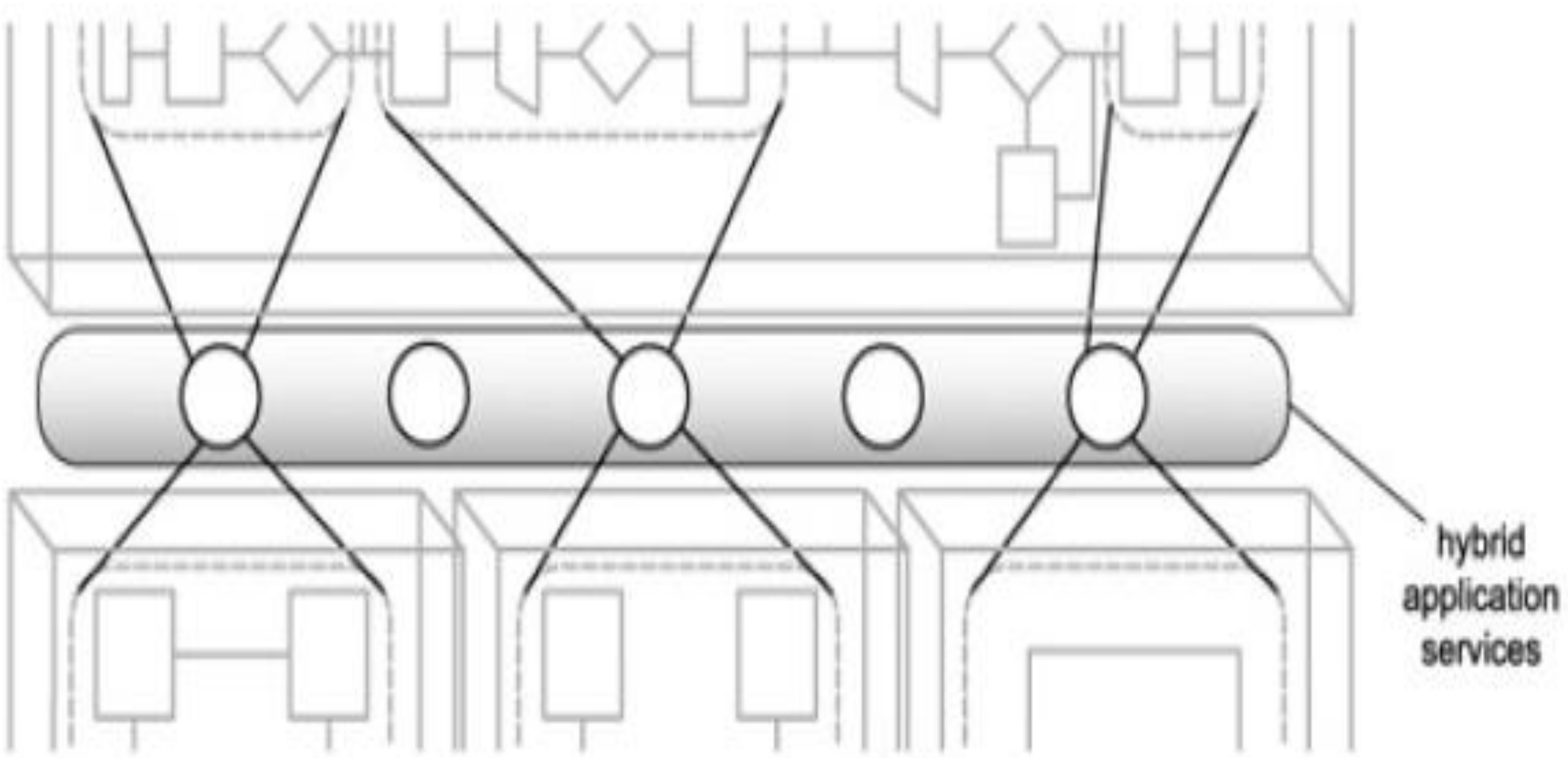
- Parent Level of Abstraction
  - Other services do not need to maintain interaction details / execution sequence
- Service is known as Process service
- **Service Model:**
  - Controller
- "Process service has potential to become utility service to an extent, if the process is entirely reusable"
- Orchestration service imposes significant expense and complexity


business process layer
orchestration service layer
process service
service interface layer
application layer

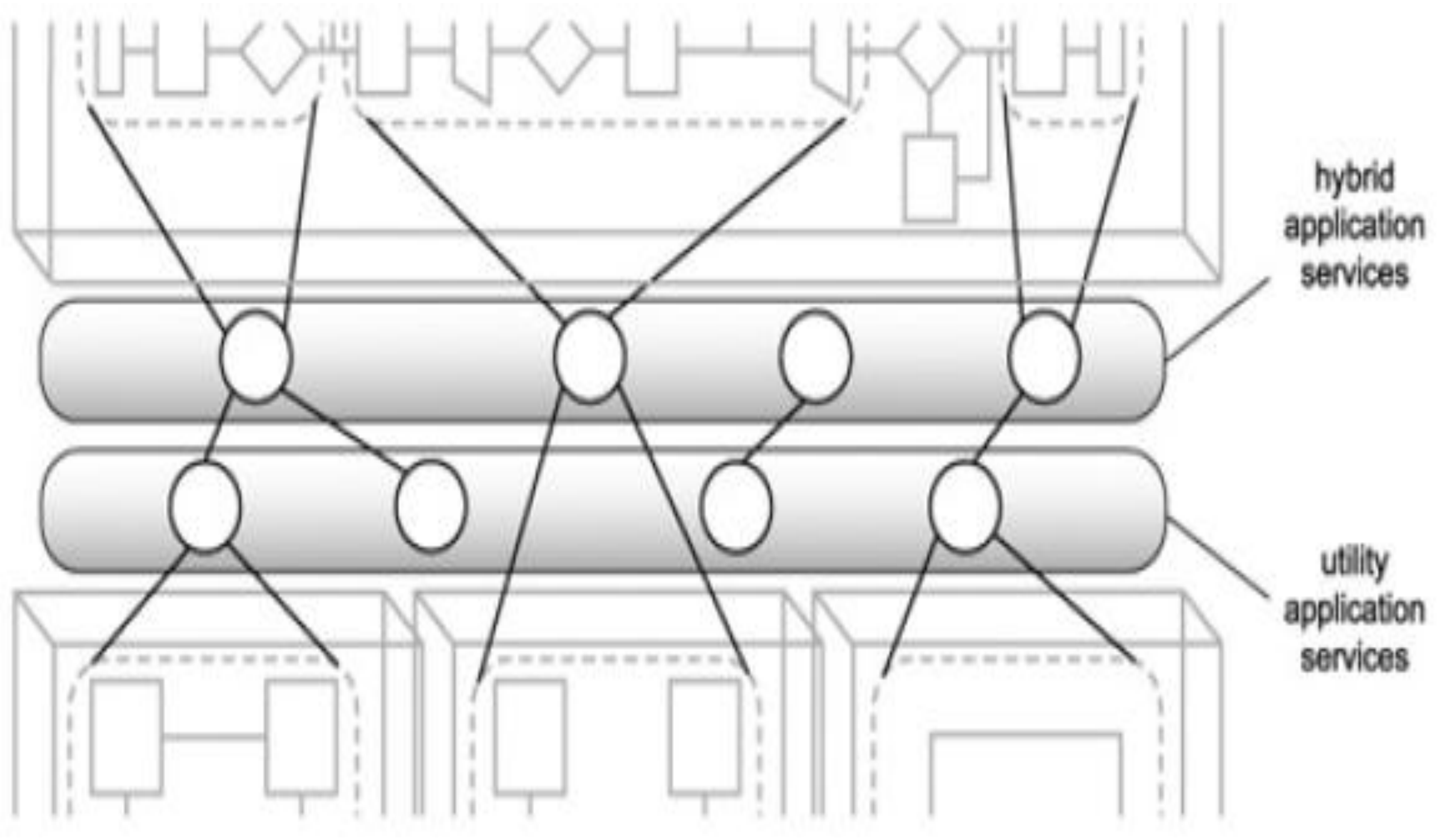
Service Layer Configuration Scenarios

- Many combinations / Scenarios are possible with the use of following services:
  - Hybrid Application Service
  - Utility Application Service
  - Task Centric Business Service
  - Entity Centric Business Service
  - Process Service

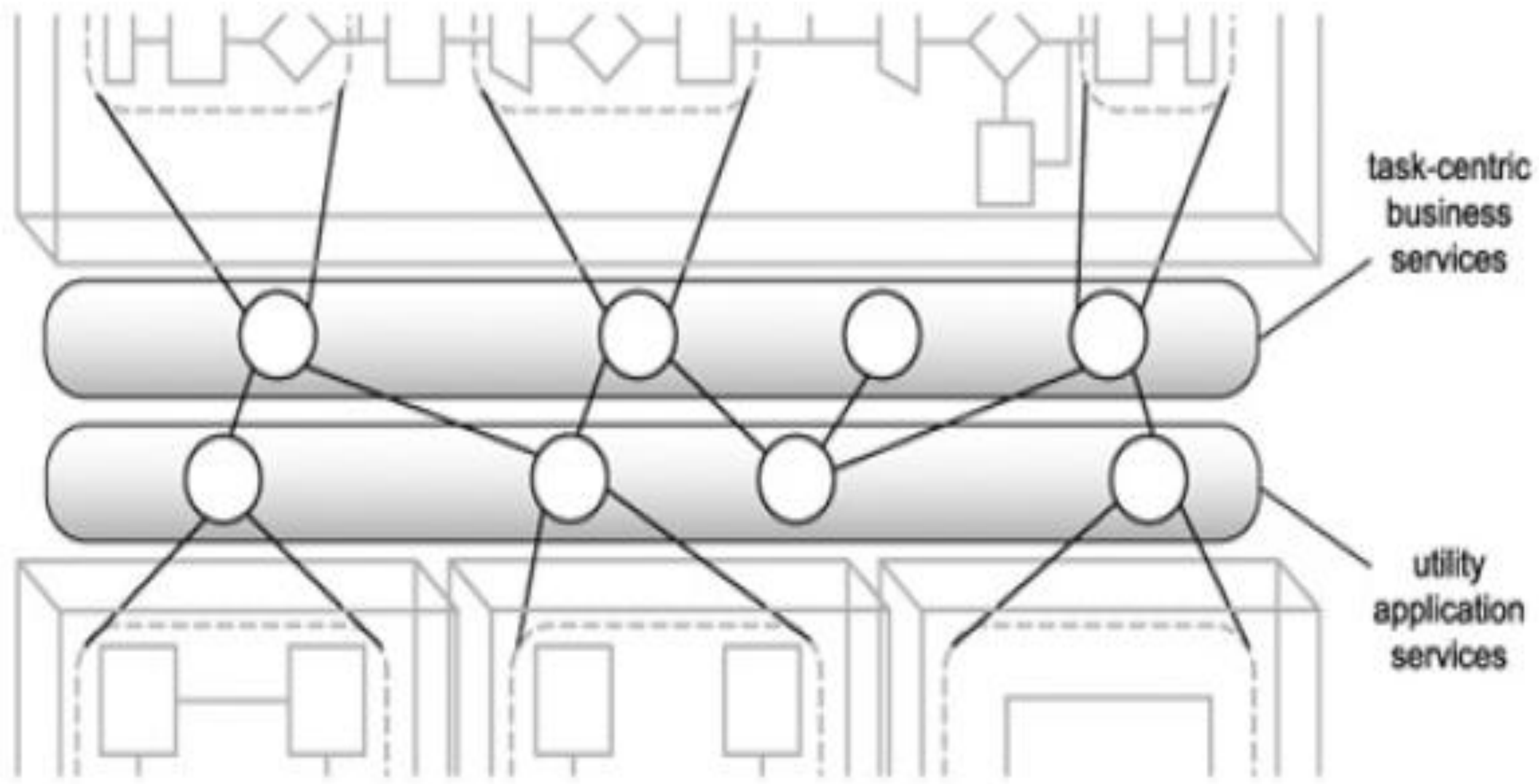
1. Only Hybrid



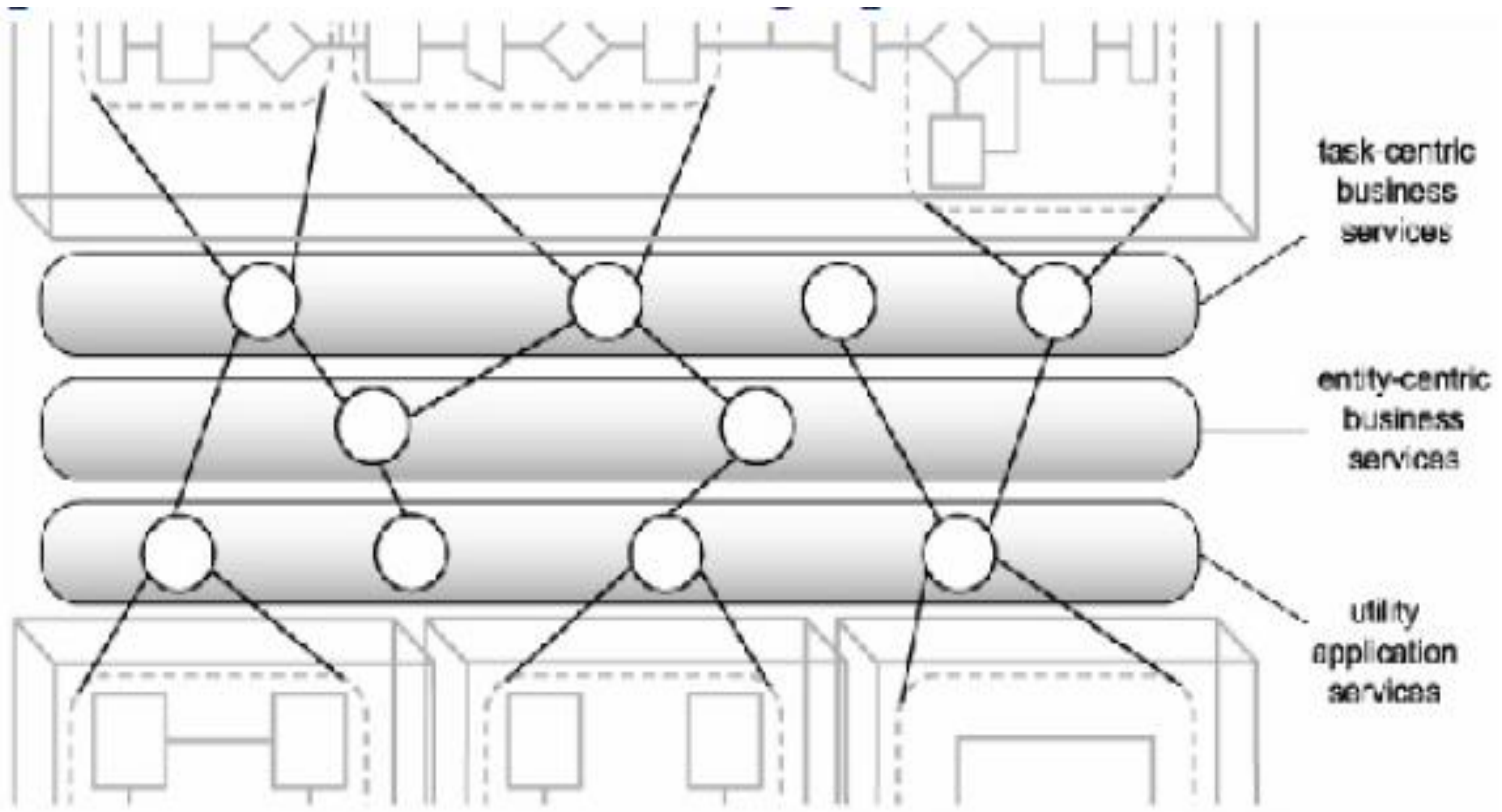
2. Hybrid and Utility



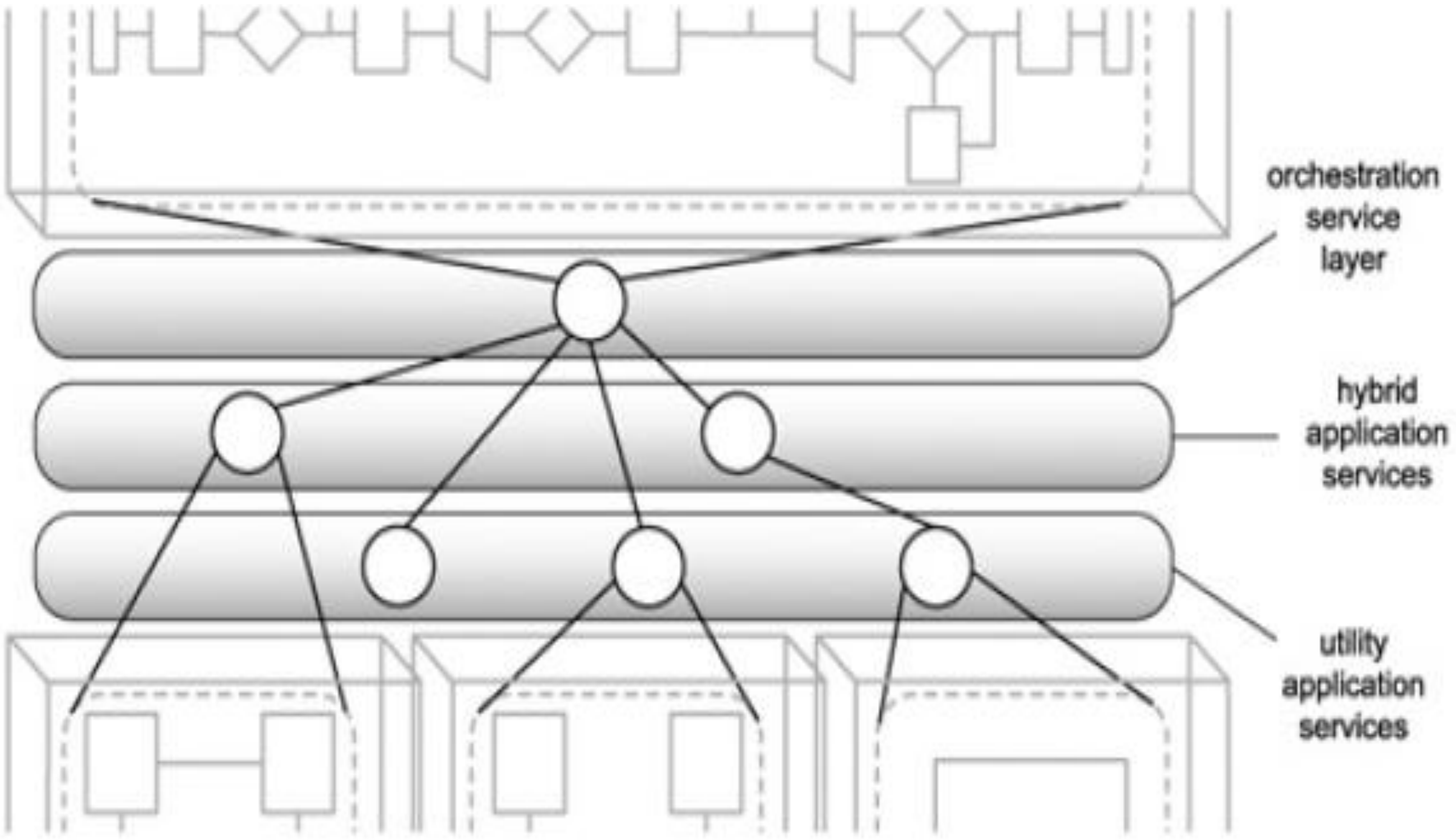
3. Task Centric and Utility



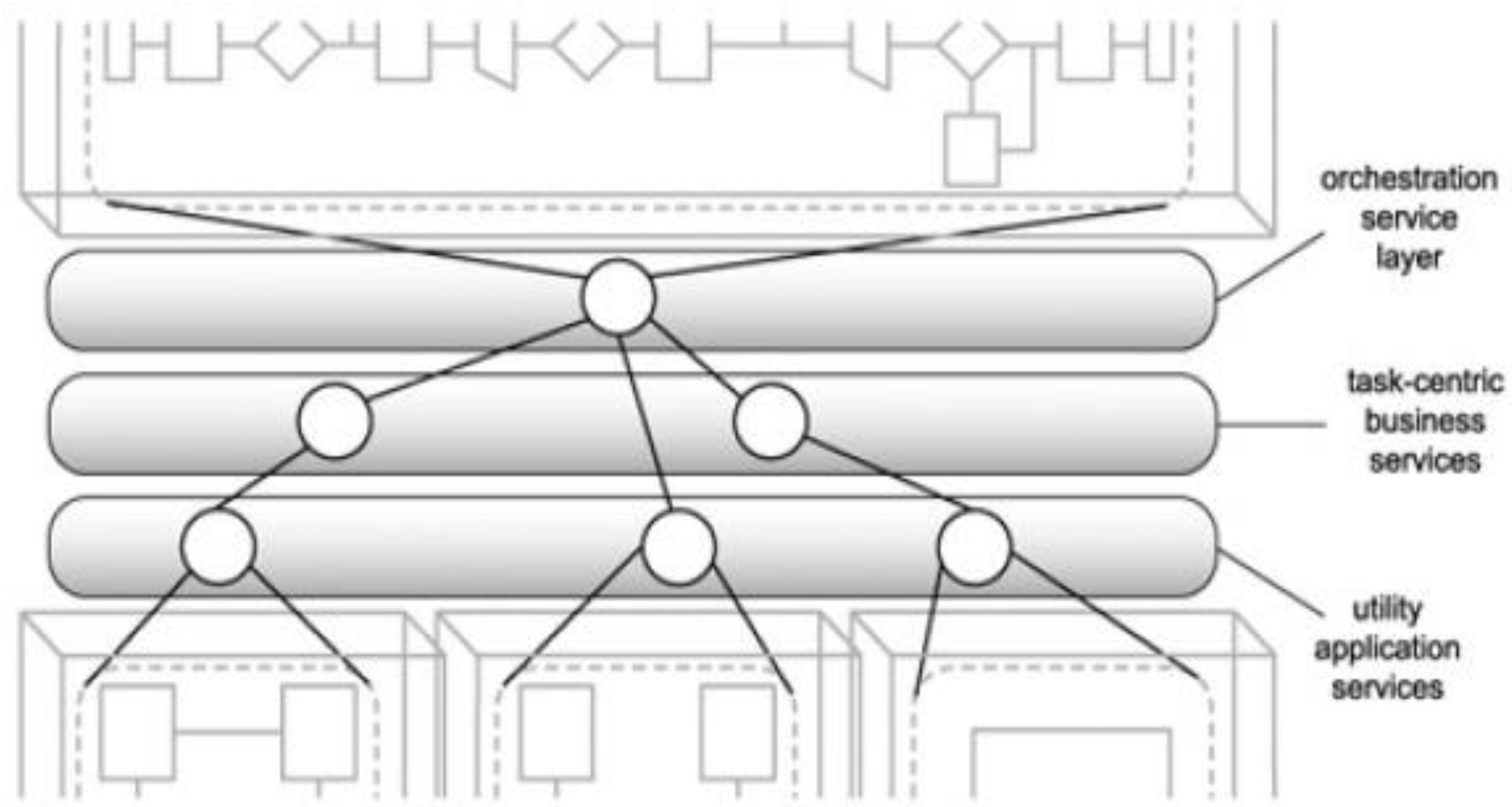
4. Task Centric, Entity Centric and Utility


task-centric business services
entity-centric business services
utility application services

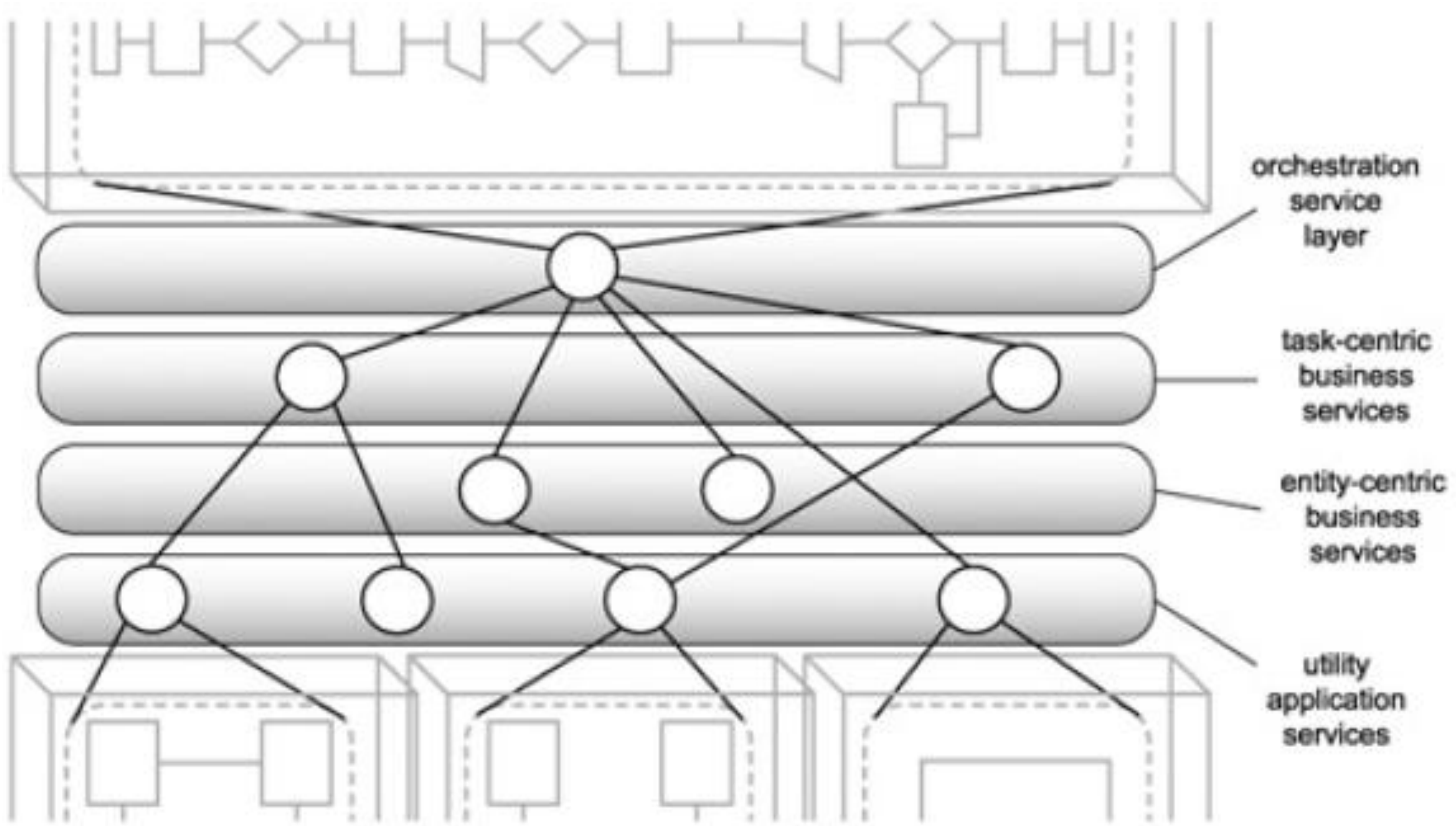
5. Process, Hybrid and Utility


orchestration service layer
hybrid application services
utility application services

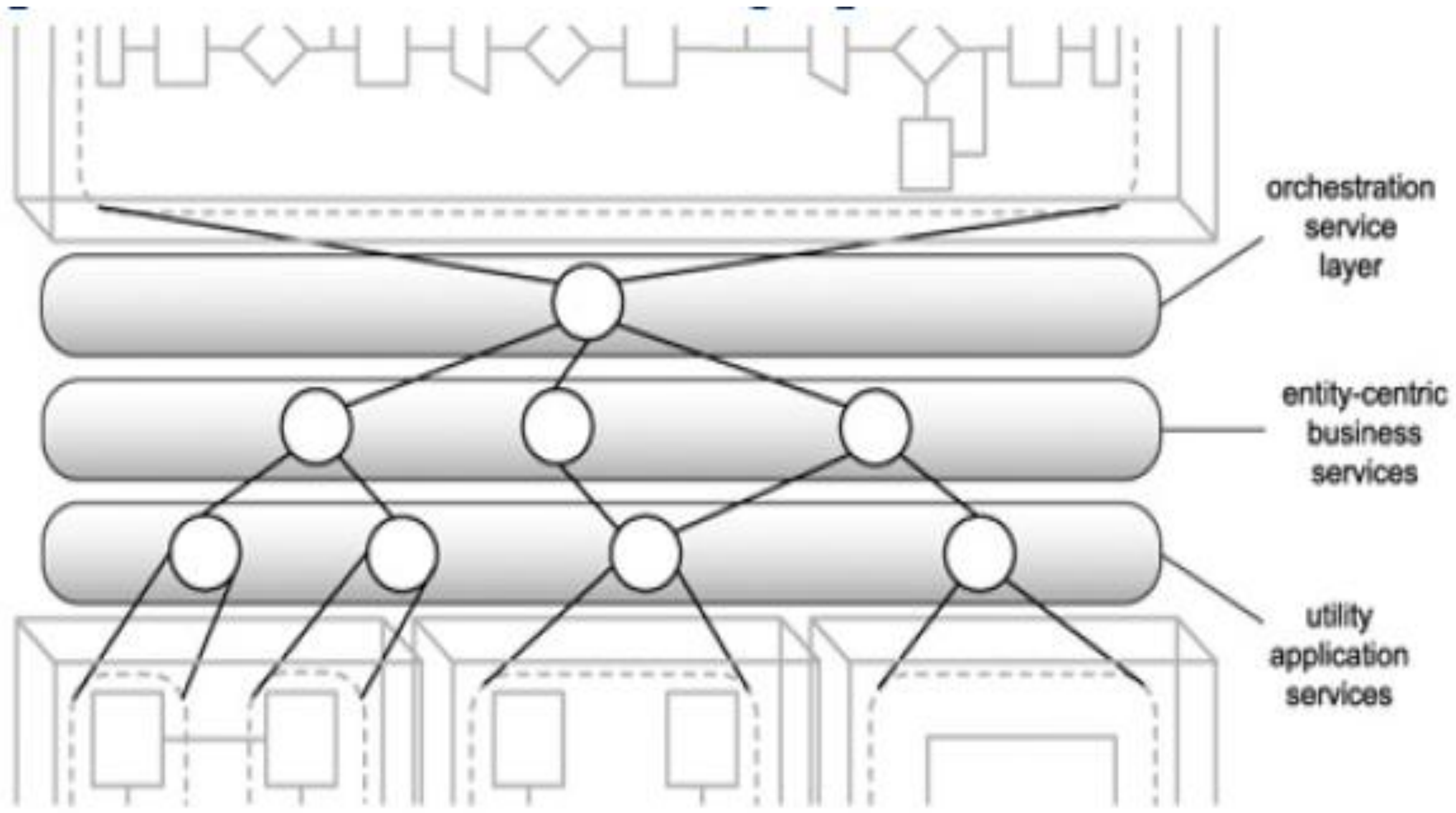
6. Process, Task centric and Utility


orchestration service layer
task-centric business services
utility application services

7. Process, Task Centric, Entity Centric, and Utility


orchestration service layer
task-centric business services
entity-centric business services
utility application services

8. Process, Entity centric and Utility


orchestration service layer
entity-centric business services
utility application services