Business Process Digitalization and Cloud Computing

10. Composing Services

Andrea Morichetta, Phd

November 29, 2017

Computer Science Division

Table of contents

Service composition

Composing Service

- Return of investment?
  - Provide **business value** and **solve real-world problems**.
  - Services are **reusable components** and are meant to be combined to meet business needs for enterprise applications.
- In this part **we focus on**:
  - **service layer interaction**, choreography, orchestration
  - Business process execution language (BPEL)
  - **Strategies** in service composition

Understanding composition

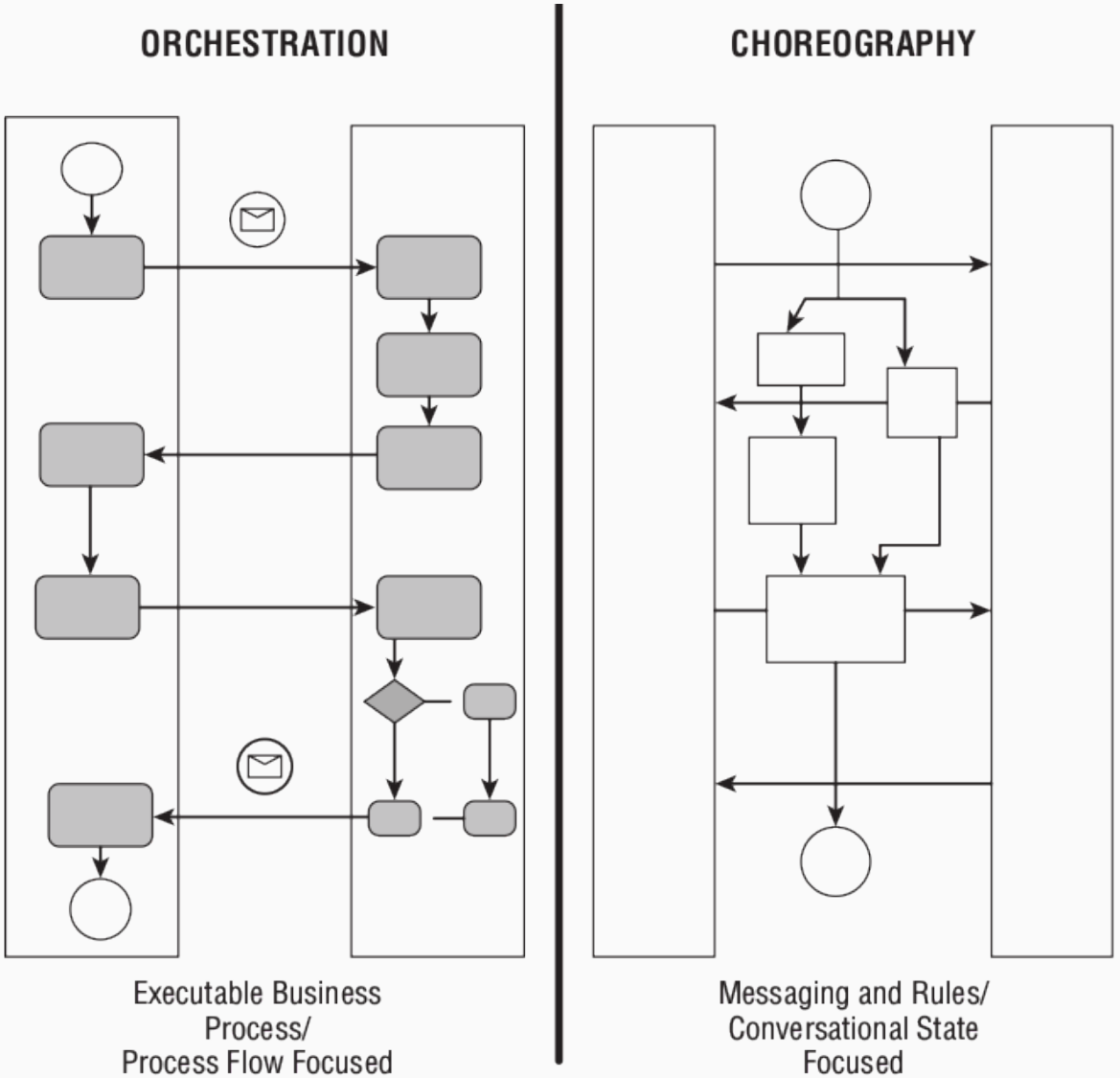
Services are able to **interact** by means of **collaboration dependencies** defined in conversation **rules**.

- loosely coupled services involves loosely coupled interaction processes between services.

Service Interaction

- Orchestration: The point of reference for orchestration is a **single controller**
  - **how** service interact
  - business logic
  - **order** of interactions
  - BPEL is an orchestrator script, can be executed by an orchestrator based on **rules** and **sequence**.
- Choreography:
  - describe the **sequence of messages** between services (public exchange of messages and conversational state)
  - focused on **exchange of messages** from the **perspective of a third party** observer
  - **WS-CDL** describe the peer-to-peer collaborations.
    - is used when an appropriate path of a composition cannot be determined without an additional input from a service consumer.

Orchestration versus choreography



Orchestration and choreography recap

- Orchestration is based on an **executable business process** from the perspective of **one controller**
- Choreography is based on the **messaging interactions**, from the perspective of a third party (multi-party collaboration)
- Orchestration takes place with a **central engine** controlling an execution flow
- Choreography allows for **multiple parties**, permitting a more peer-to-peer approach.

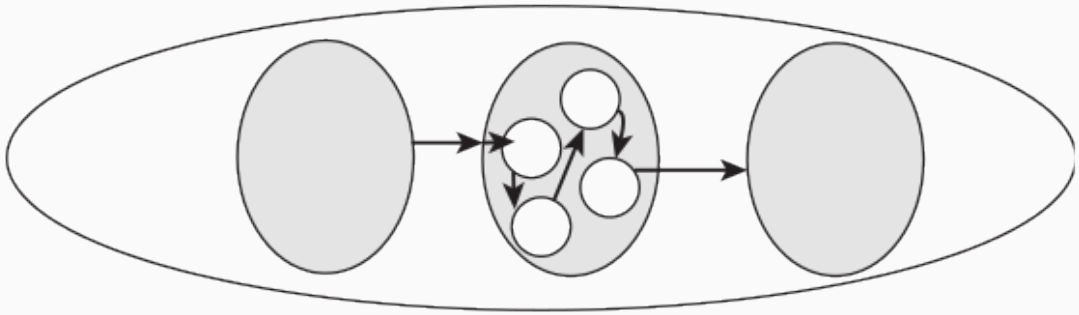
Business process and composition

- **Business process** and **rules** for combining them should be implemented separately
- **Decoupled composition** can change configuration as the business process change.
- **Hard-coded rules** and **business process** logic into the logic of services that aggregate other services, require code changes if requirements are modified.

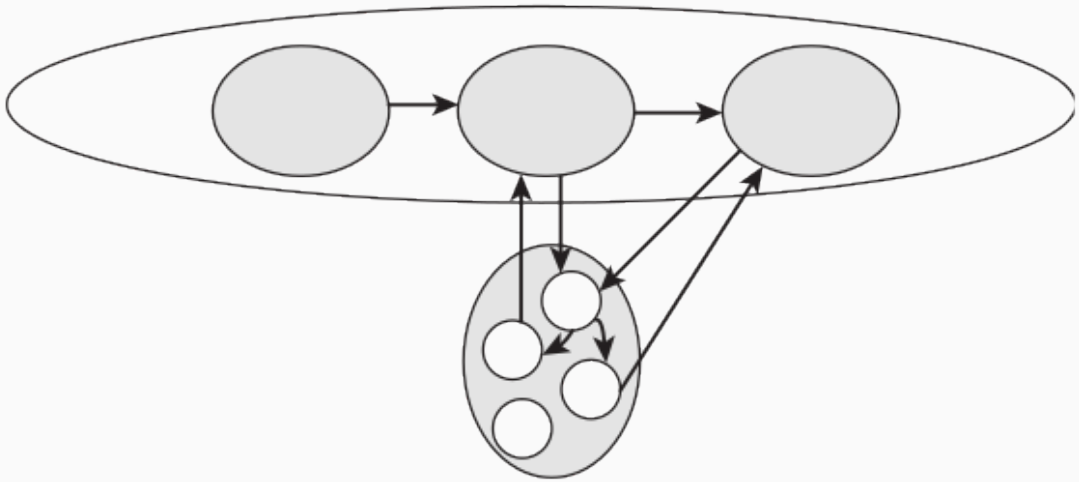
Hierarchical and conversational composition

- Hierarchical composition: the implementation of the composition is **completely hidden** from its consumer (black box).
  - is optimum for implementing solutions that do not require human or any other interaction from the solution invoker.
- Conversational composition: the implementation of the composition is **hidden** from the service consumer, but selected **intermediate execution results are exposed** (gray box).
  - is used for executing composition that that cannot be determined without an additional input from a service consumer, based on intermediate execution results

Architectural model in service compositions



"Black Box" Composite Service



Conversational Composite Service

Conductor-based and Peer-to-peer composition

- Conductor-based: consists in a specialized service (mediator) that interacts with a consumer and **controls the execution of other component** services participating to the orchestration.
  - The mediator implements a **sequence of service invocations** to reach the final goal.
  - The transitions undertaken are based on the **input received** by the coordinator.
- Peer-to-peer: each participant is responsible for partial orchestration, based on its **individual rules** without a **central coordinator**.
  - The final behavior is specified as a family of permitted **message exchange sequence**
  - Typically this implementation lead to **hierarchical solutions**

Programmatic composition

Simplest way to implement a service composition is to use **general-purpose programming language.**

- The logic for combining services is **statically written** and compiled in the programming language.

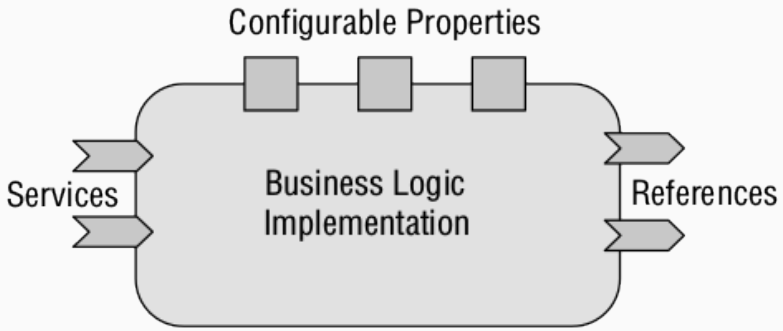
Main drawbacks:

1. **Hard-coding of composition logic**, which makes it harder to modify and maintain
2. Implementation requires some form of **transactional support** to ensure correct behavior in case of failures.
3. Potential introduction of a significant amount of **infrastructure** code required to manage synchronous and asynchronous interactions (DB).

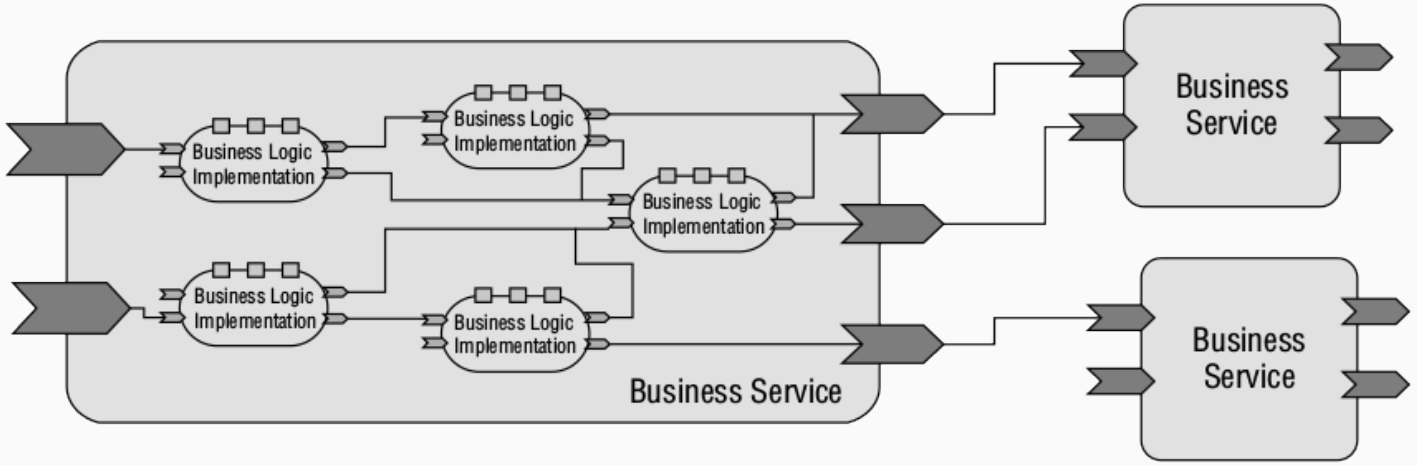
SCA

Service Component Architecture (SCA)

- **Language-neutral**, technology-neutral set of specifications aimed at **simplifying the composition of services** by hiding many of the infrastructure elements of the service invocation.
- SCA specifies how to **create** components, **combine** them, and **expose** the component assembly as a service
- SCA-defined programming models, components can be built with Java or another programming language
- Communication itself is actually technology-neutral (SCA, JMS, REST)



Connecting SCA components

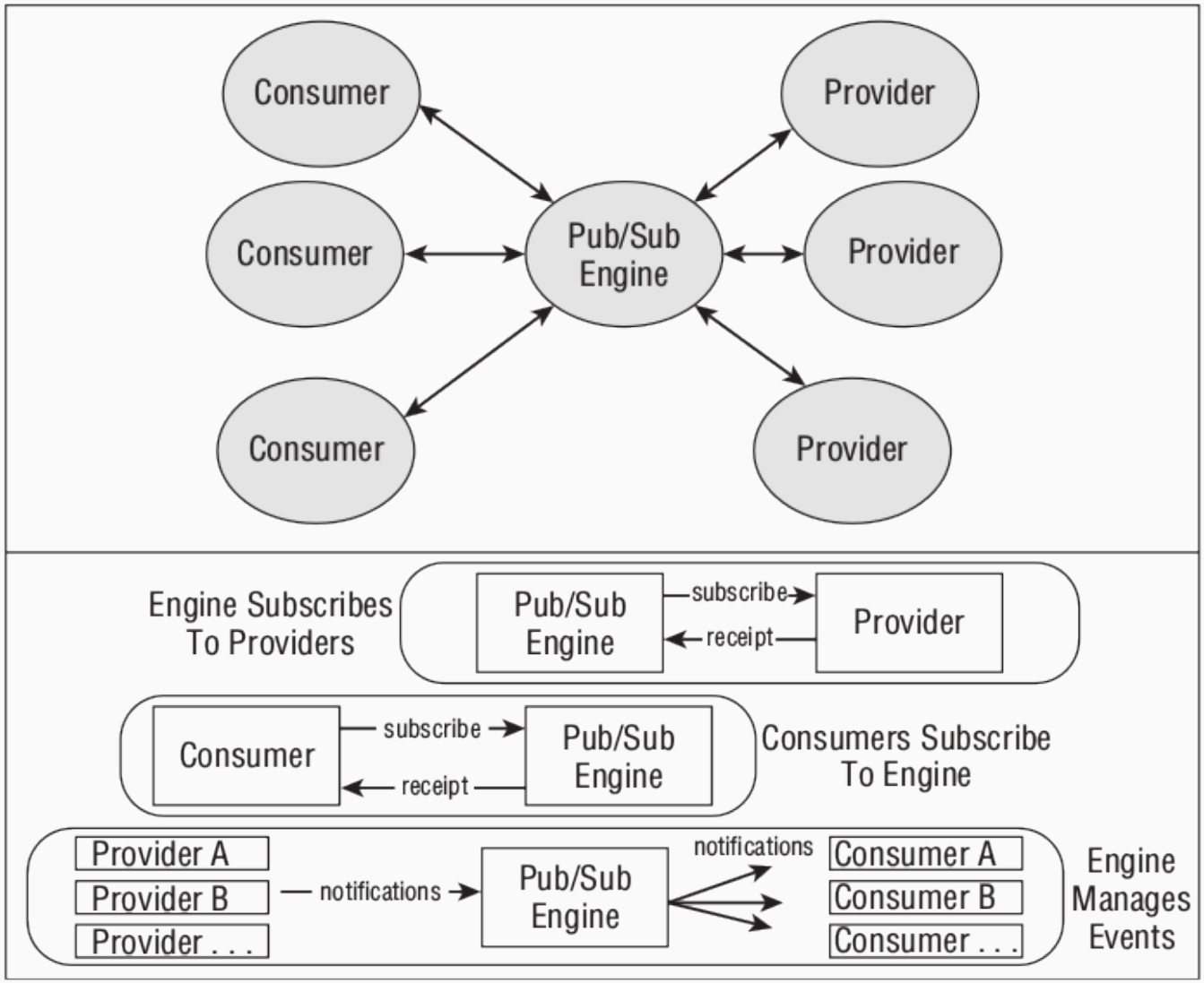


- Components are connected to each other using *wires*.
- **Wire** is an abstract representation of the relationship between a reference and some service that meets the needs of that reference.
- Used for bottom-up composition: selecting a set of deployed components (services), configuring them, connecting them, and deploying the resulting composite service.

Event-base composition

- Service consumers **publish events** to a publish/subscribe intermediary, which delivers them to the actual service providers.
- Event-based composition **decouples layers** between service consumers and the service provider.
- Extremely **flexible implementation** of composite solutions.
- The sequence of events effectively creates a composite solution
- By changing a set of services subscribed o a particular topic, it is possible to completely change an implementation.
- **Drawbacks**:
  - Not provide the notion of service composite solution **instance**, which makes very difficult to coordinate events
  - It is very difficult to **ensure corrective behaviour** if the service fail

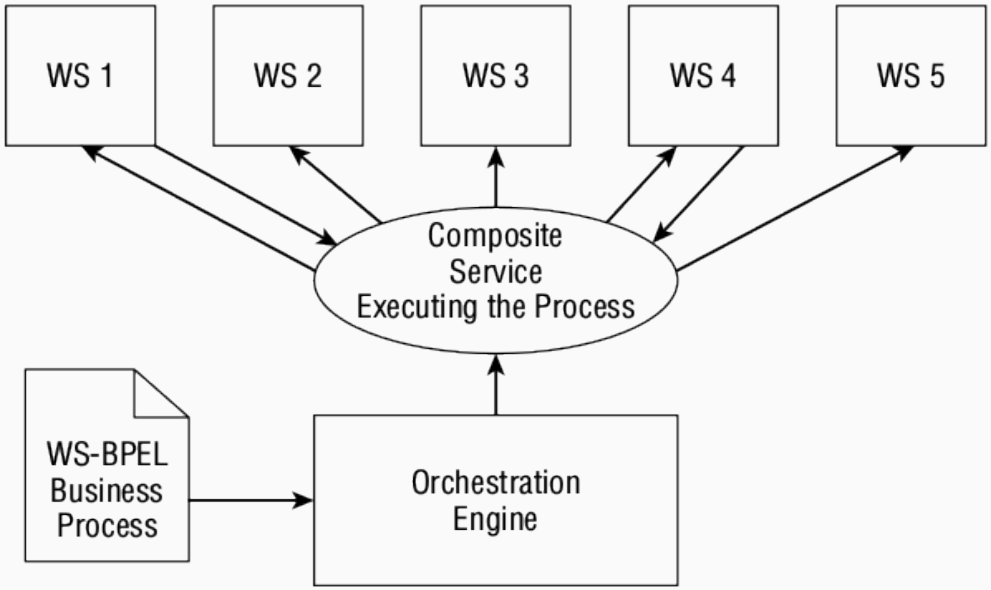
Event-base composition



Orchestration-based Composition

- Use an **orchestration engine** to control the execution flow of a process (WS-BPEL).
- Orchestration uses **centralized process implementation and execution**, this lead to a simpler process **maintainability**.
- The executable process specifies the details and rules of the business process, abstracting the details from the services involved.
- Orchestration provide **recursive aggregation**: composite service can be created to compose new processes involving interactions with services

Orchestration-base composition



Orchestration engine

Features:

- **Asynchronous service** invocation and the use of correlation tokens for matching between messages
- Management of **concurrent execution** of process instances.
- Management of the **execution context** containing the information that determines the state of the business process
- Management of the **data flow**, including data flowing into services
- Support for **manual activities**
- Collection and processing of **business events** and key performance indicators
- Support **scalability** and **availability**.

Advantages of an orchestration engine

- Orchestration languages directly support the majority of orchestration concepts
- Equipped of a **visual editor**
- WS-BPEL, are **portable** from any programming language platform, and they can be run on an orchestration server regardless of whether it is J2EE-based,

Centralization and decentralization of Orchestrations

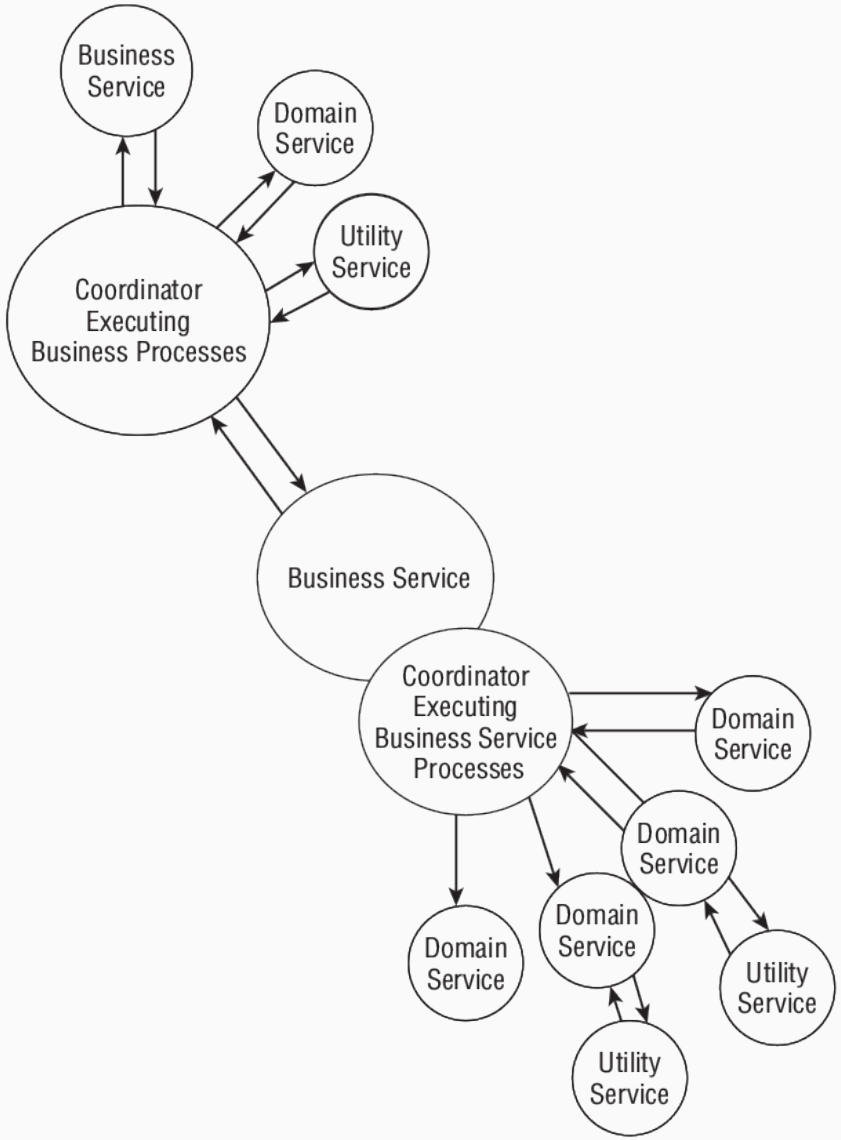
The main advantages of a centralized approach are:

- Business decision are **hard-coded at design time** or at compiling time.
- **Simple** to manage
- **Event** auditing
- **Easy to store** the business process in one place

The main disadvantages are:

- Processing **bottleneck**
- **Performance** and **availability**
- **Single point failure**

Scalability on centralized coordination



Engineers can decide how to split up services giving them

Business Rules vs Business Process

Business Rules

- Business rules: describe the **sequence of invocations** of a particular service participating to a business process.
  - Help the organization to better achieve goals, manage the communication between organization, operate more efficiently, automate operations.
- Business Process: Describe how to achieve a specific goal.

Rules vs business process

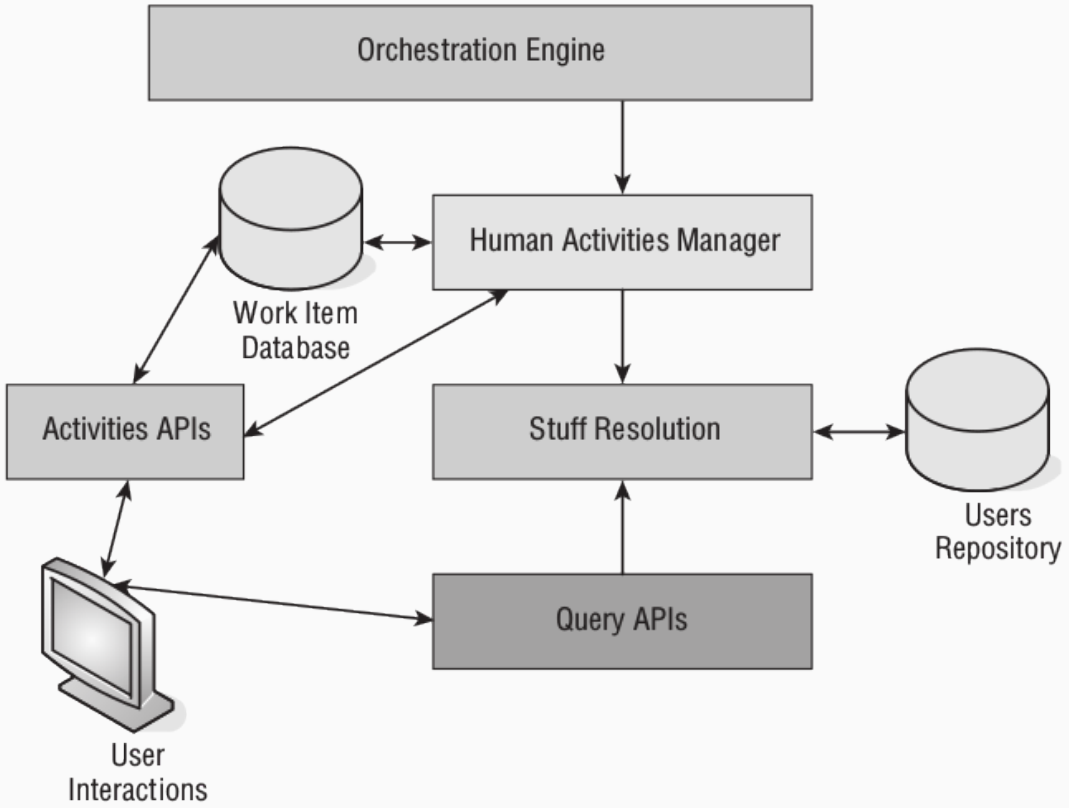
- Synchronicity:
  - Rule engine evaluate rules in a synchronous way.
  - Process engine are asynchronous and invocation are based on request/response
- Statefulness:
  - Rules are stateless, once fired take in input parameters and send back the output
  - Business engine holds the states of each business process
- Determinism
  - Rules are fired simultaneously, however the order is not deterministic
  - Business process are deterministic except with parallel activity
- Granularity:
  - Rules provide a smaller granularity and offer a higher level of flexibility
  - Processes are more stable but less incline to changes

Incorporating Human Activities

- **Human activities** are composed by activities that are **too expensive** (not-cost effective) or **too complex to automate**.
- Main Issues:
  - The **interaction** is based on interface that are different from the ones of software systems
  - The interactions are exclusively **asynchronous**
  - **Slow response time**
  - Low **throughput**
  - Poor **availability**

Human Activity Manager

A typical approach to support human activity is to use the **human activity manager** in collaboration with the orchestration engine.



Questions?