# Lesson 19 – Software engineering aspects

Service Oriented Architectures Security

Module 4 - Architectures

Unit 1 – Architectural features

**Ernesto Damiani**

Università di Milano

# SOA is HAD

- HAD is an old concept in distributed information systems
  - H = Heterogeneous
  - A = Autonomous
  - D = Distributed
- HAD is
  - the essence of and the reason for SOA
  - the problem SOA tries to solve
- HAD is where the OO paradigm has failed
  - CORBA
  - Object Oriented Databases
  - Reuse

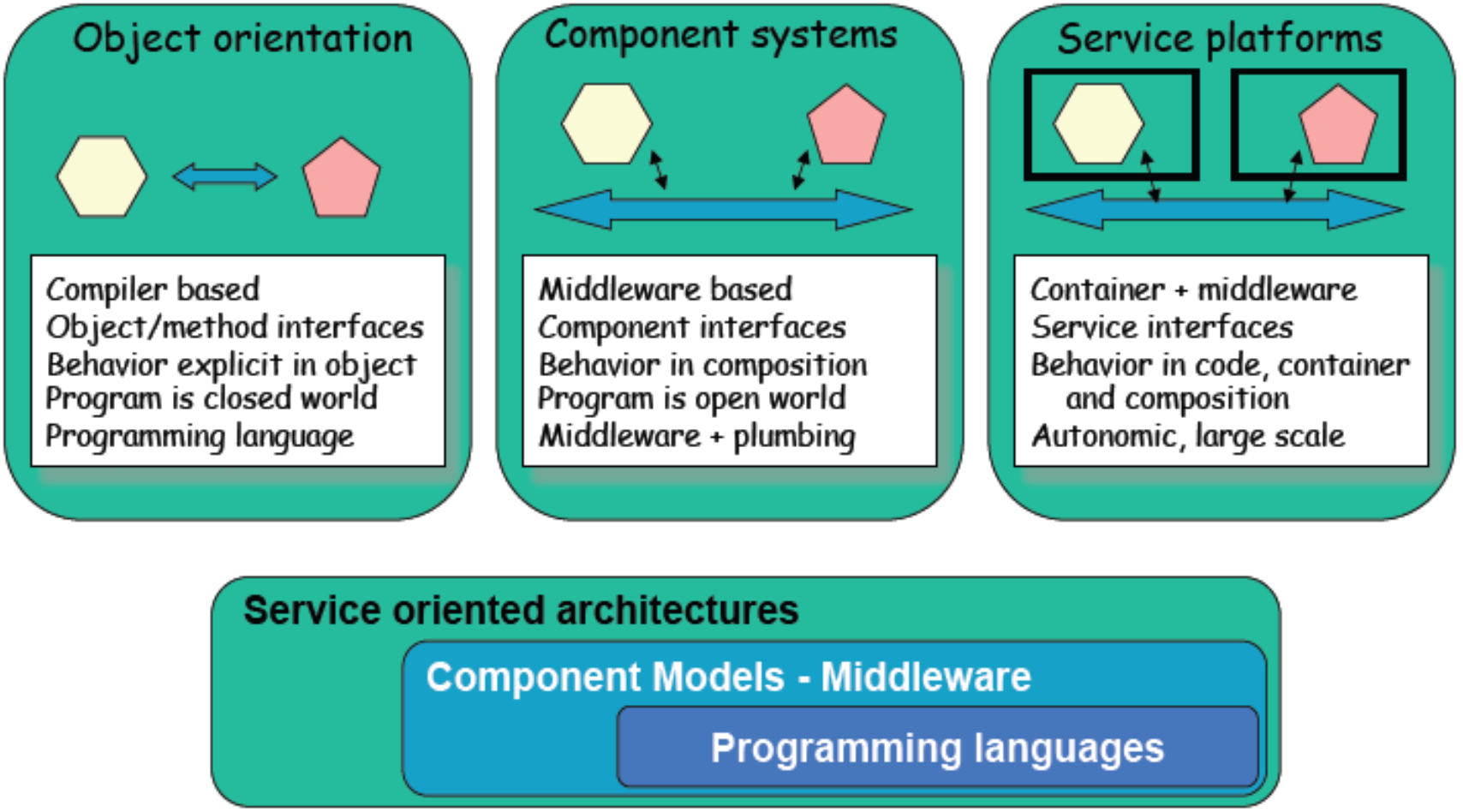
# No HAD in OO: OO Detours (Steve Cook, ECOOP 06)

- Reuse Problem: objects ignore the environment where they live
  - Real objects in different systems are autonomous
  - Real objects in different systems are heterogeneous
- Distribution Problem: abstracting away the problem's essence
  - Tight coupling (language, interaction, development, operation)
- Database Problem: impedance mismatch
  - Still present with XML, messages, and events
- Modeling Problem: from OO models to software systems
  - Objects are too low level to model real HAD systems

# Services not components

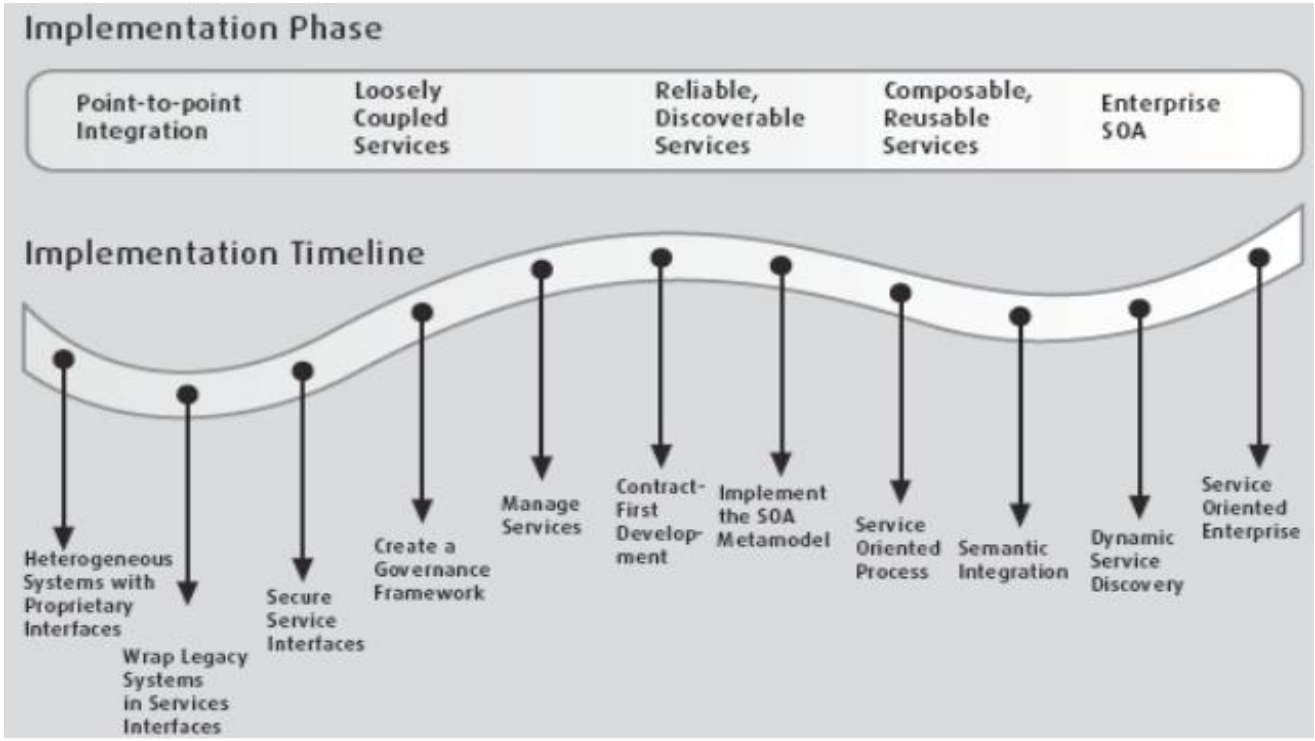
- The two sides:
  - Integration is based on services
  - Programming is based on objects and components
- There are similarities in theory, in practice they are very different
  - Services are not object oriented
  - Services have document based interfaces
  - Services have a behavioral contract with their consumers
  - Services are reusable by definition
  - Services are (should be) loosely coupled
  - Services are autonomous
  - Services have (very) explicit boundaries

# Objects – Components - Services



# Services: integration not programming

- The key issue in enterprise computing today is integration
  - Services are the best way to approach integration known so far



# XML and Asynchronous Data Streams: two contemporary examples (1)

- XML
  - Programming language variables
  - Semi-structured Documents
  - Procedural interfaces
  - Document based interfaces
  - Behavioral interfaces
  - Variable assignment
  - Declarative queries
  - Impedance mismatch

# XML and Asynchronous Data Streams: two contemporary examples (2)

- Asynchronous Data Streams
  - Procedural control flow
  - Event based control flow
  - Language based distribution
  - Platform based distribution
  - Behavioral interfaces
  - Sequential programs
  - Highly parallel and concurrent
  - Model mismatch

# Services = run time Software Engineering (1)

- A Service contract involves the interface, the Service Level agreement and QoS
- Contracts are key to be able to develop, debug, optimize and maintain systems developed as a combination of services
- The management of the information about services, and the engineering of systems based on services leads to the notion of SOA Governance

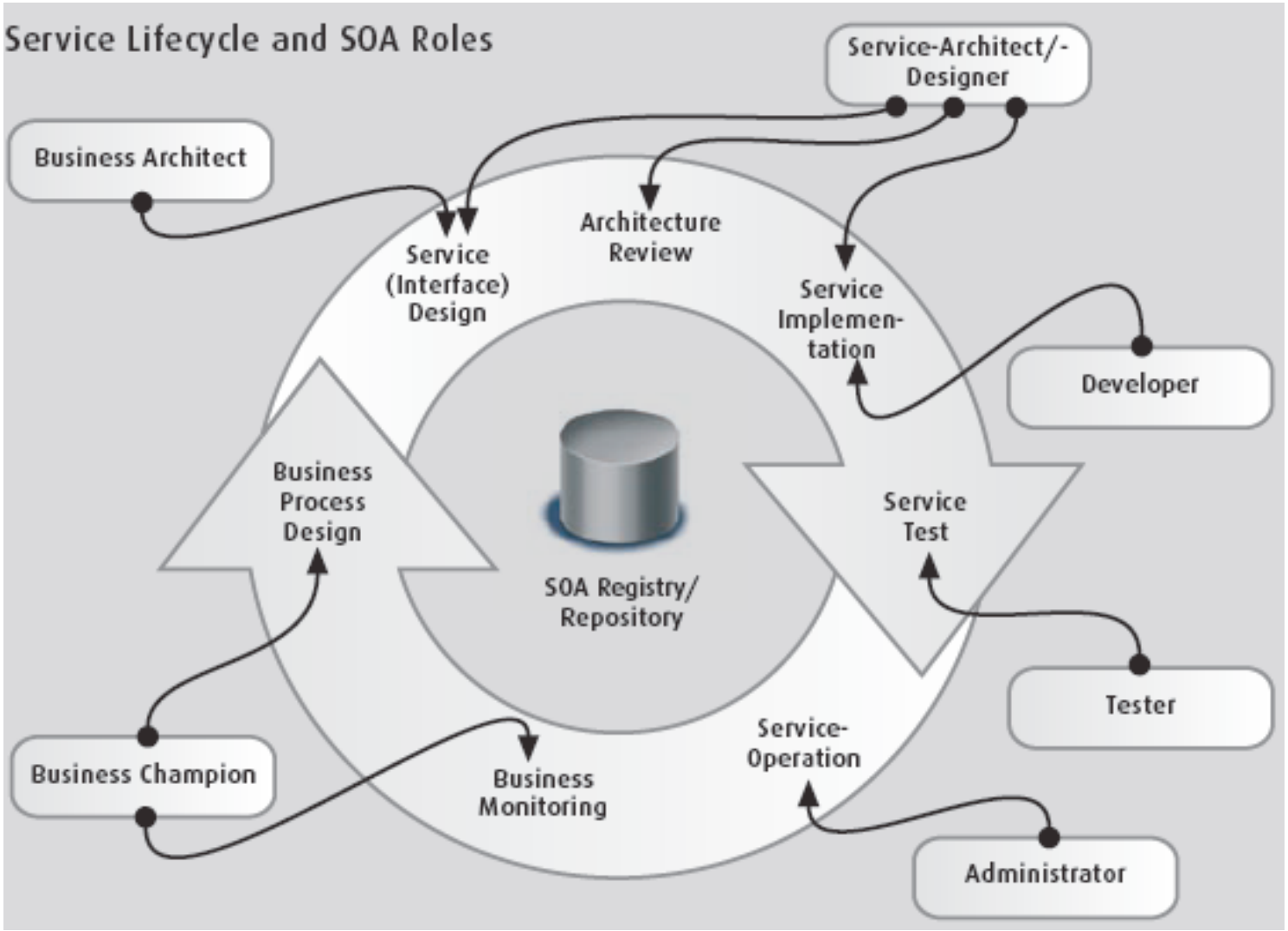
# Services = run time Software Engineering (2)

- Service contracts are not the static, compile time pre- and post-conditions of conventional programming languages
- They are an additional software layer in charge of the dynamic aspects of using services

# SOA governance (1)

- SOA governance introduces different aspects to standard software engineering concepts:
  - Service definition (the scope, interface, and boundaries of a service)
  - Service deployment lifecycle (the lifecycle stages)
  - Service versioning (including compatibility)
  - Service migration (deprecation and sunsetting)
  - Service registries (dependencies)
  - Service message model (canonical data models)
  - Service monitoring (problem determination)
  - Service ownership (corporate organization)

# SOA governance (2)



# Service definition

- The biggest challenge by service definition is to identify service boundaries:
  - Services are seen as having well defined, clear boundaries
  - In practice, not that easy
  - What is a service?
    - Functionality vs data
  - Data cohesion
    - services that use common data are difficult to separate
- Boundaries make sense at the business level not at the implementation level:
  - A single database can expose many services but it remains a single database

# Coupling (1)

- Tighter coupling tends to cost more over time:
  - Synchronization
  - Coordinated deployment and deployment; updates
  - Combinatorial explosion in dependencies
  - Services are not independent (boundaries)
  - Coupling implies more expensive testing

# Coupling (2)

- Looser coupling requires greater investment up front:
  - More design work
    - Generality
    - Message model
  - More implementation work
    - Queues
    - Message management

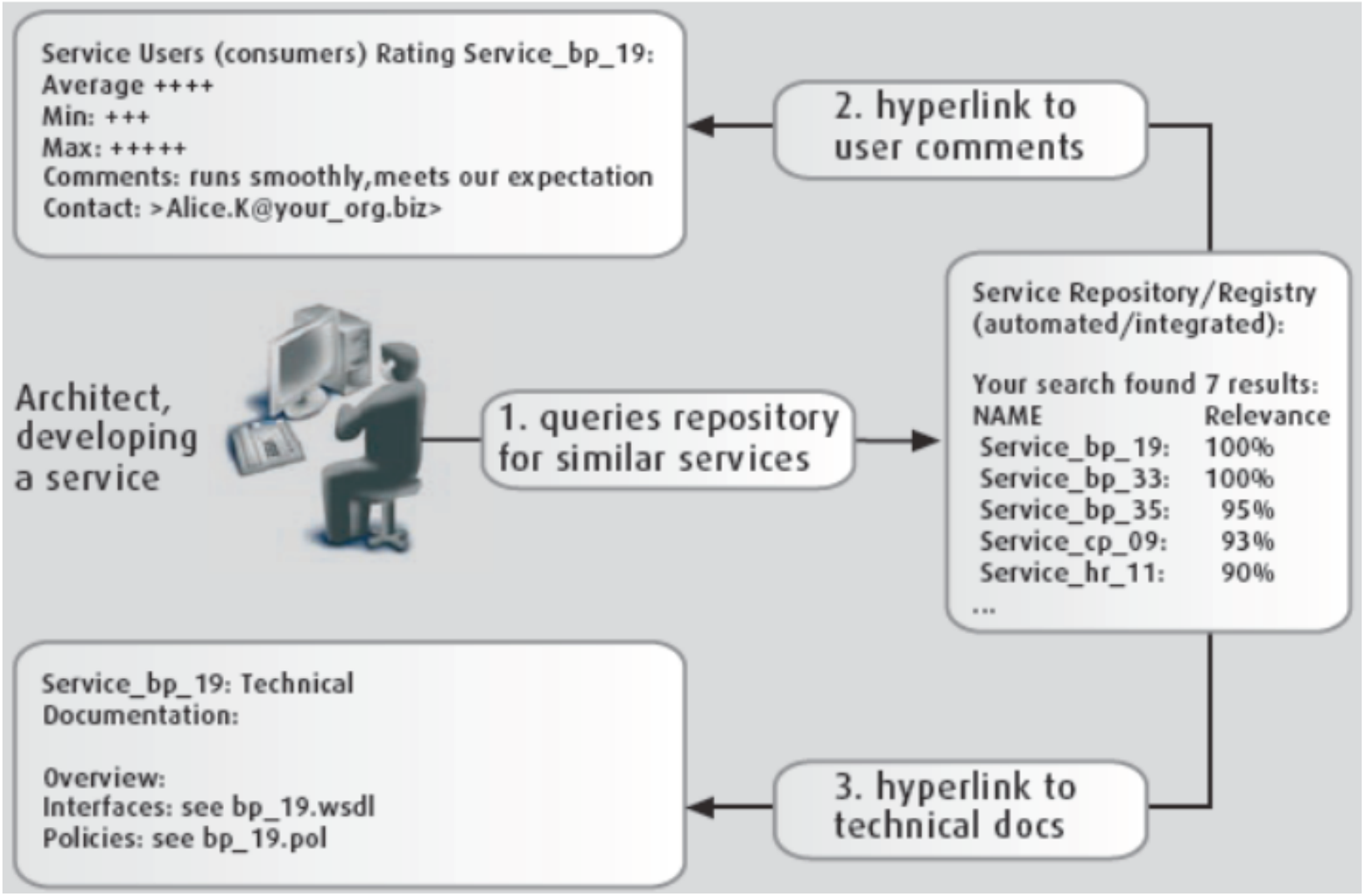
# Life cycle

- Services are autonomous entities used by other services
  - Individual maintenance is not enough
  - Coordination across entities
  - Dependency management
- Service versioning
  - Necessary to maintain compatibility across dependencies (otherwise all connected services need to be upgraded simultaneously)
- Service migration
  - Eventually all consumer services need to be migrated to the newest version
  - Cascading dependencies

# Registries for control and information

- The service registry serves as the name and directory server for services and related information
  - Versions available
  - Services and providers
  - Consumer and dependencies (less common)
- UDDI specification quite useful for this purpose
  - Technical information
  - Documentation

# SOA development



# Message model

- Services are asynchronous and communicate through messages:
  - What if each service defines its own messages
  - What about formats and conventions
- A message model acts as a standard determining
  - The formats and conventions for messages
  - May standardize messages as documents
  - Published as canonical model

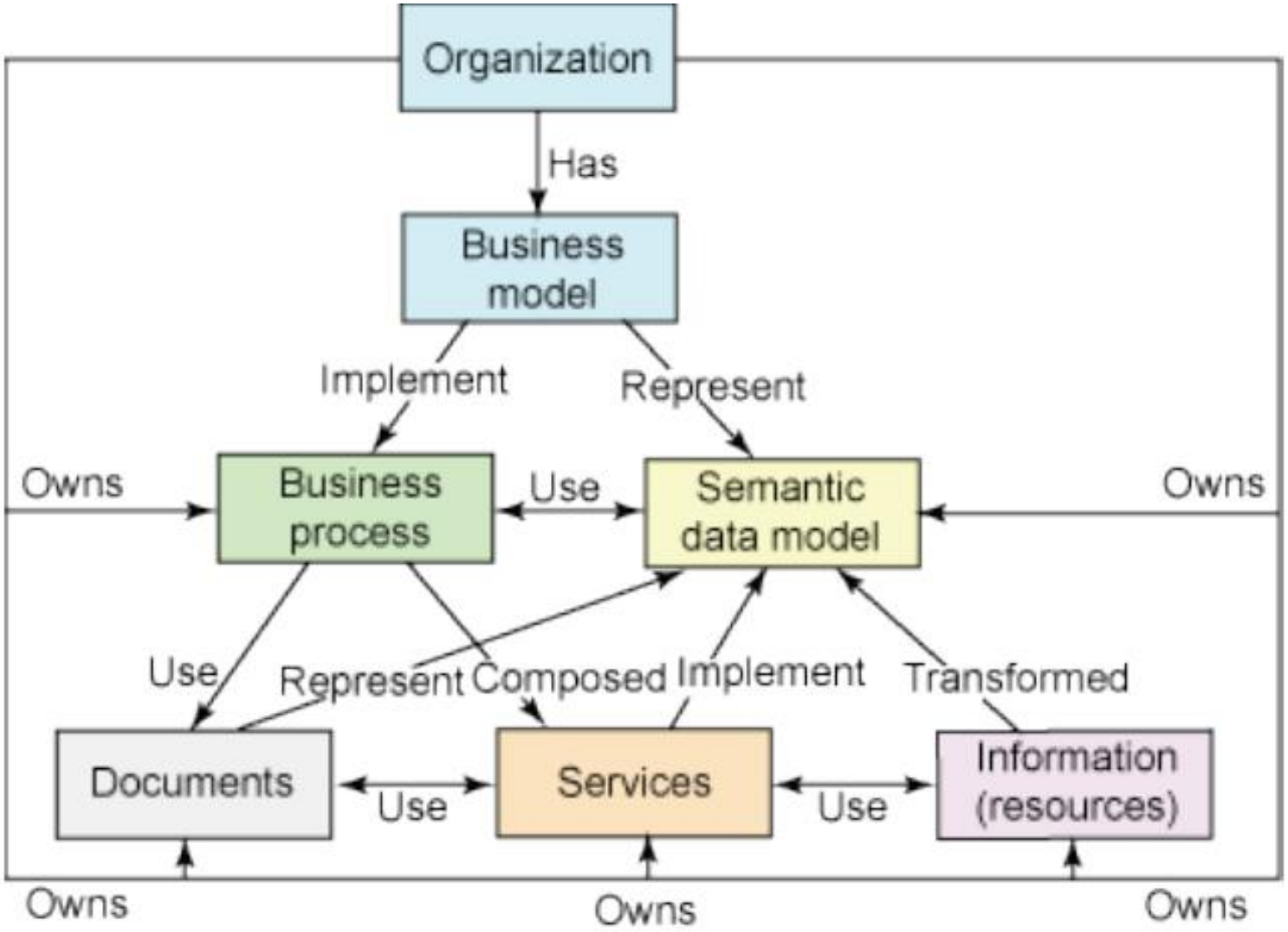
# Monitoring

- Processes and service composition involve the invocation of many remote services
  - What happens if the service is not there
    - Exception, failure, error?
  - Services are autonomous
    - Queues help
    - Eventual manual intervention
  - How to fix the problem?
    - Alternative services
    - Migration
- Several tools in the market

# Ownership

- Arbitrage is necessary when many consumer services use the same provider service:
  - Priorities
  - Life cycle
  - Versioning and migration
  - Load and performance
- Centralized and distributed solutions
  - Both become complex at large scales

# Conceptual model




FINE