COMP 9322 Software Service Design and Engineering

Lecture 1 – Introduction to Service Oriented Computing

References

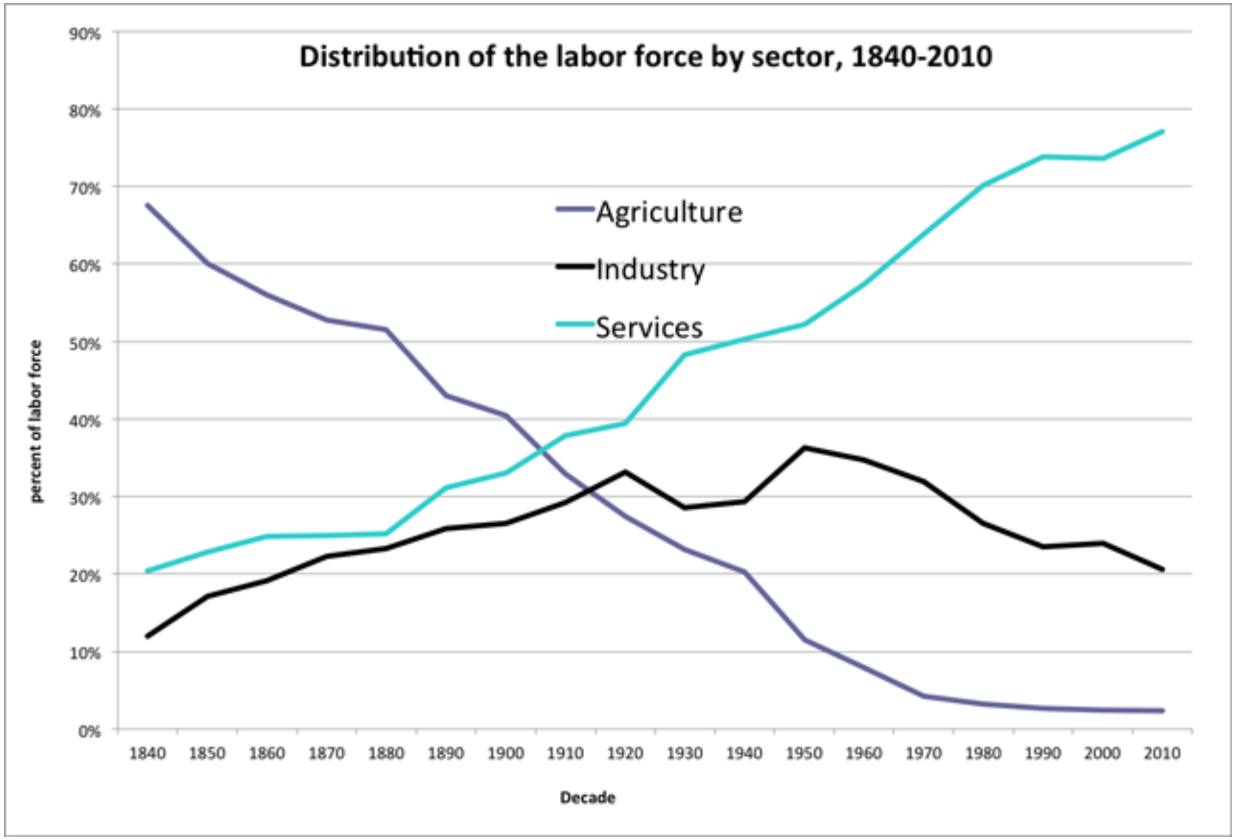
- Thomas Erl, Service-Oriented Architecture: Concepts, Technology, and Design, 2005, Prentice Hall.
- Thomas Erl, SOA: Principles of Service Design, 2008, Prentice Hall.
- http://www.soa-manifesto.org/
- http://www.soa-manifesto.com/annotated.htm
- Peter F. Drucker, Post-Capitalist Society,1993

Outline

- Service Orientation
  - The roots, Services
  - Service Oriented Architecture
  - Service Design Principles
  - State of the Art: Web Services
- Challenges of Service Orientation
- SOA Manifesto as a Summary

Service

- Service is the application of specialized competences (knowledge and skills), through deeds, processes, and performances for the benefit of another entity or the entity itself. LUSCH & VARGO, "The Service-Dominant Logic of Marketing". (Armonk, NY: ME Sharpe. 2006).



Organization's of Yesterday

- Vertical Integration was the mode of operation:
  - □ The whole supply chain was owned by a single company.
  - □ Ford owned and produced everything in The Rouge
- The traditional factors of production was:
  - □ Land, labor and capital

The Ford River Rouge Complex:
2.4 km x 1.6 km
93 buildings, 1.5 km2 floor space,
100,000 workers

https://en.wikipedia.org/wiki/Ford_River_Rouge_Complex

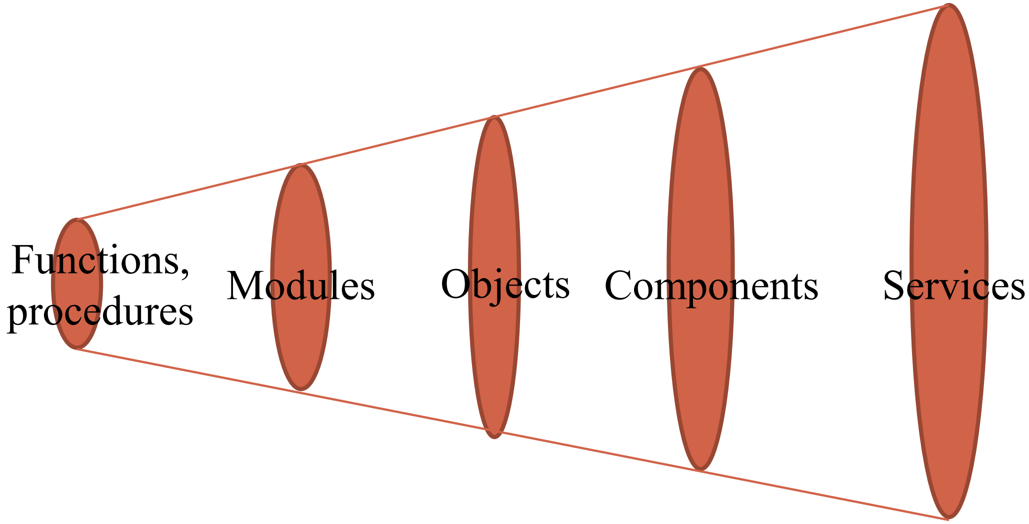
Organizations of Today

- The most critical factor for production is:
  - Knowledge – It is always specialized
- We need organizations to put specialized knowledge into production
- Today’s organizations need to be structured around new principles:
  - Be able to change quickly
  - Be able to specialize – concentrate on a single task
  - Be able to work in closely coupled teams
    - like football/tennis team instead of a baseball team.
  - Be able to innovate systematically
- Pluralization of services
  - Knowledge organizations are necessarily decentralized
  - Command and control does not work
- Service orientation is IT`s response for these challenges

‘Service Orientation’

- Separation of concerns:
  - □ To solve a large problem decompose it into smaller related pieces.
  - □ Each of these pieces addresses a concern or a specific part of the problem.
- How SO achieves this separation?
  - □ It is like different companies producing specialized goods and services as oppose to a large vertically integrated company, like General Motors producing everything.

Service as an Abstraction



- Programming is decomposing into different abstractions:
  - □ Procedures
  - □ Modules
  - □ Objects
  - □ Components
  - □ Services

SDLC Before Service Orientation

- The common popular approach:
  1. Identify the business tasks to be automated

     (Keeping inventory of items)
  2. Define the software requirements

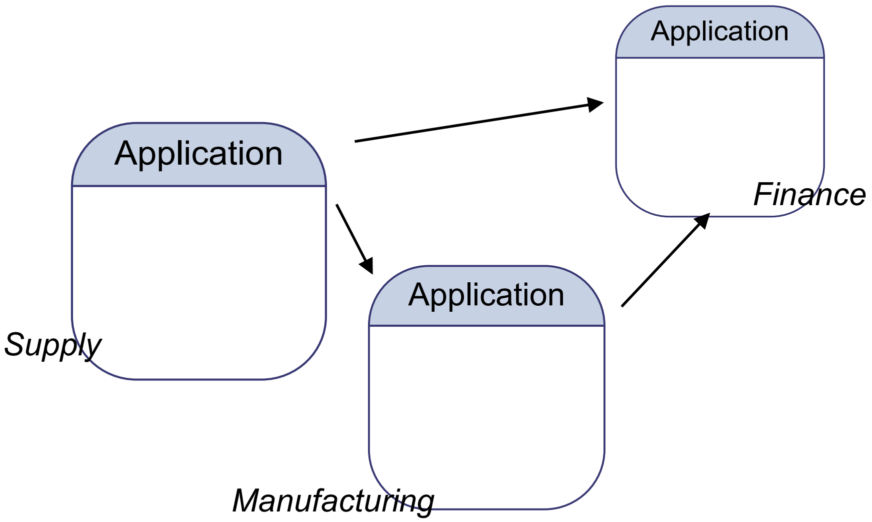
     (The system shall …. Or For the user to be able to …)
  3. Build a corresponding solution logic

     (Decompose into classes including attributes and methods …)
- The benefits of the approach:
  - □ Solutions can be built efficiently -they are specialized
  - □ The business analysis effort is straightforward – well defined
  - □ The project management is relatively easy
  - □ Can take advantage of the latest technology – independent solutions

The Problems

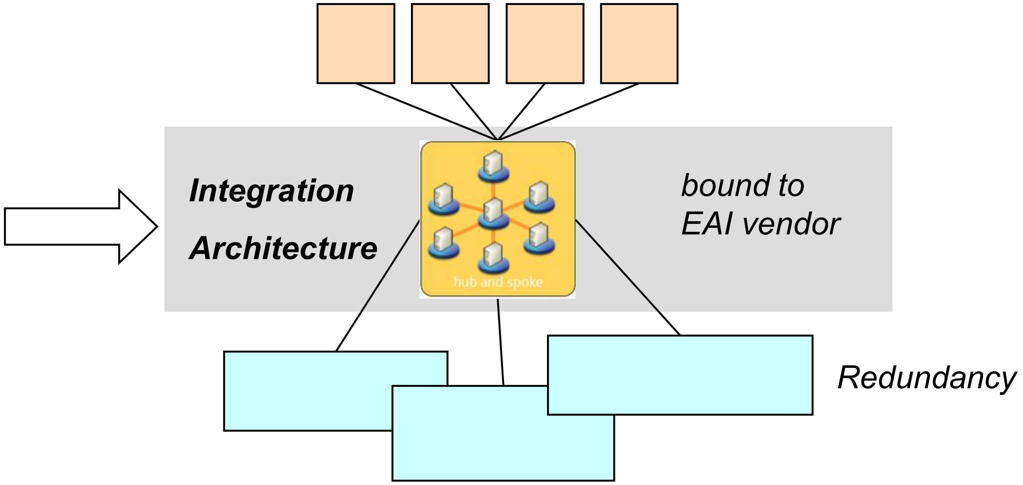
- Significant amount of redundant functionality
  - The effort to create this functionality is also redundant.
- Significant amount of maintenance and administration effort
- Integration is a constant challenge
  - Applications not designed to accommodate interoperability requirements.
- Result in complex Infrastructures
  - Different technology platforms require different architectural requirements
  - Siloed applications lead to counter-federation
  - Evolution is a great challenge

Application Centric



Functionality is duplicated in each application – generating a report

Limited Consumers
Limited Business Processes



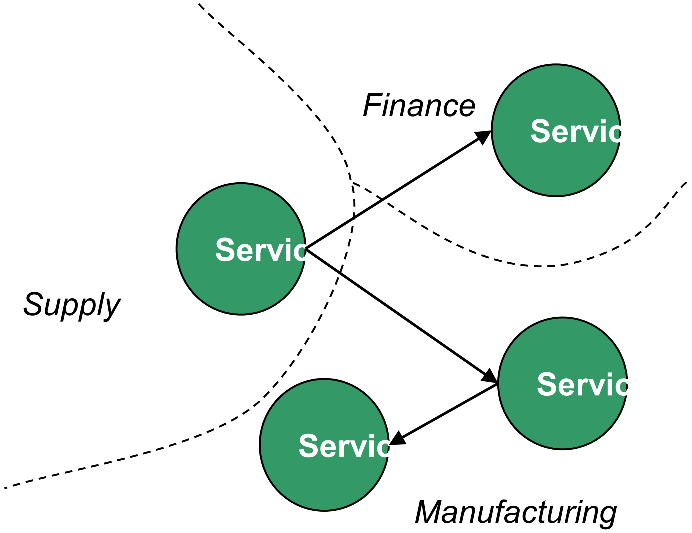
Overlapped resources
Overlapped providers

EAI 'leverage' application silos with the drawback of data and function redundancy.

Service Oriented Architecture

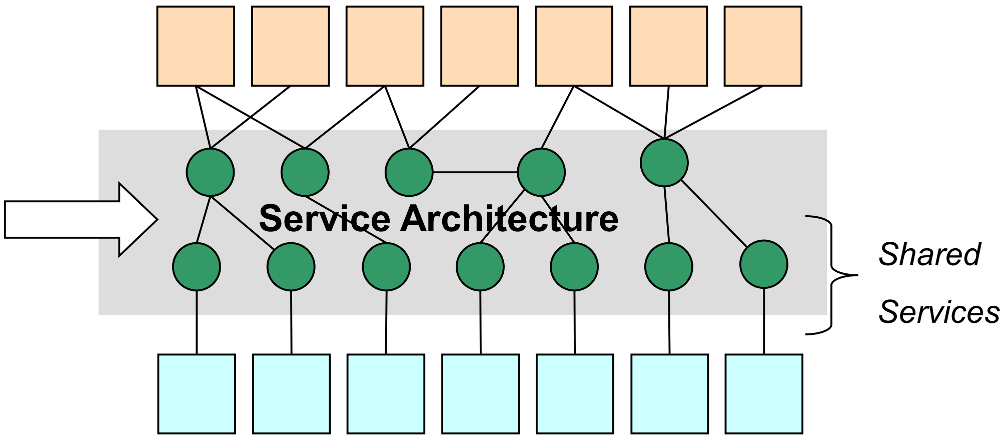
- Is a model in which automation logic is decomposed into smaller, distinct units of logic called services.
- Collectively, services establish a larger piece of business automation.
- Individually, services can
  - □ exist autonomously
  - □ evolve independently
- yet
  - □ conform to set of principles
  - □ maintain a degree of commonality and standardization

Service Centric



SO structures the business and its systems as a set of capabilities that are offered as Services

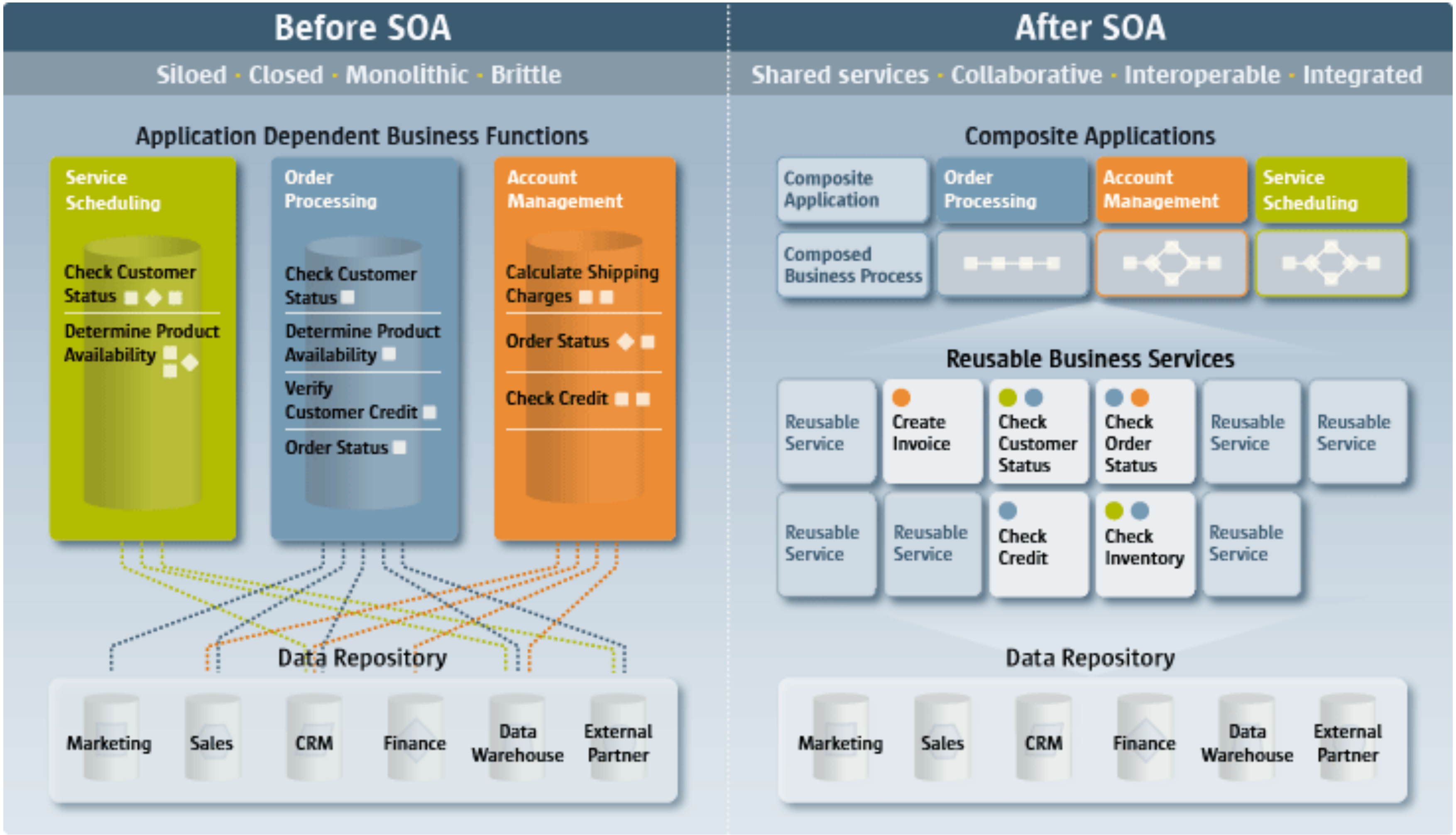
Multiple Service Consumers
Multiple Business Processes



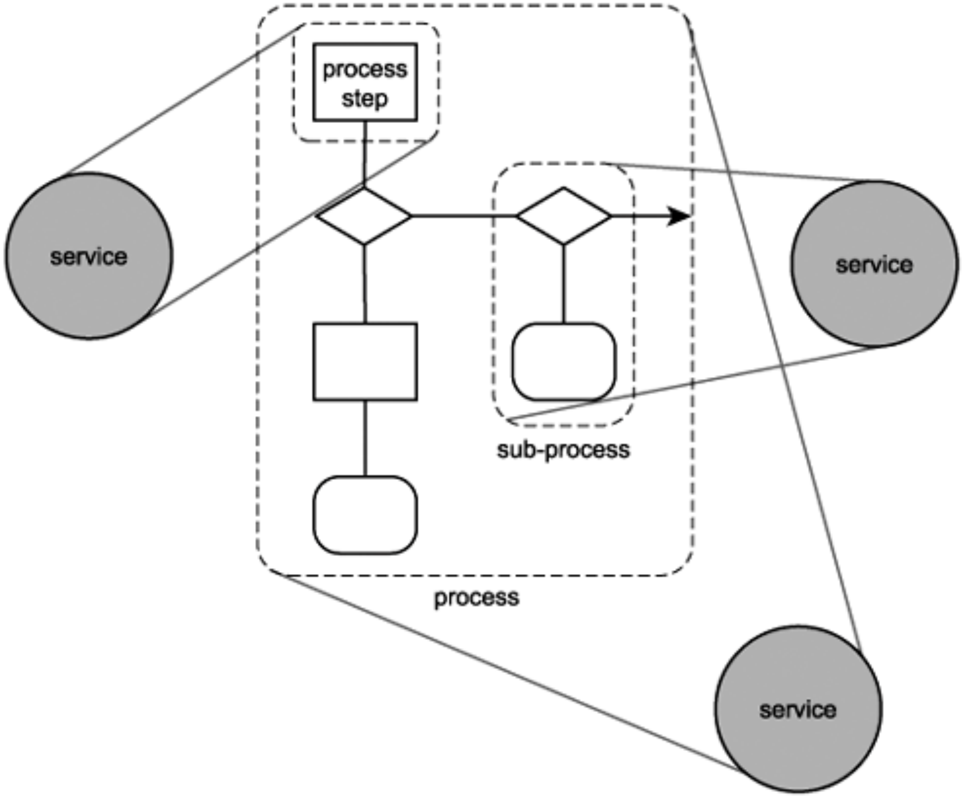
Multiple Discrete Resources
Multiple Service Providers

Service abstracts the details of how ..
Enables multiple providers and consumers to participate together in shared business activities.

Before and After SOA

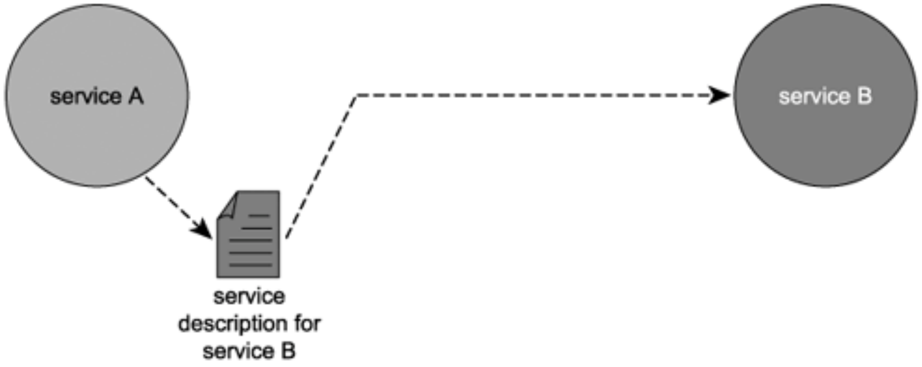


Service Context



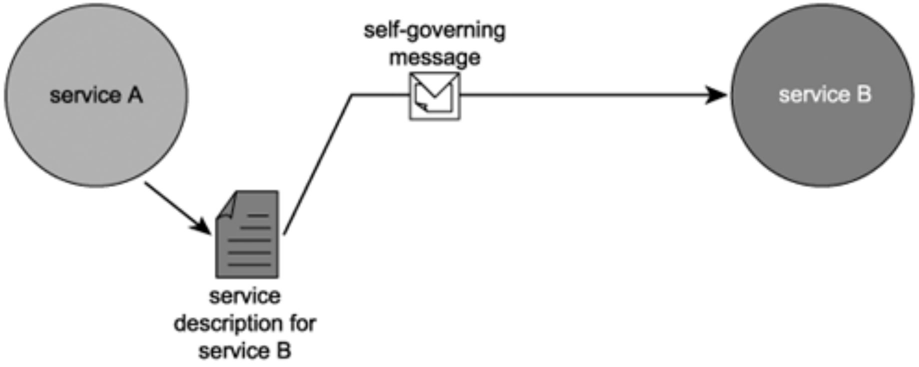
- Each service work in a distinct context:
  - Size might vary
  - Might require coordinated aggregation – service composition
  - To work together they should be related and communicate with each other

Service Interface Description



- At the minimum:
  - □ the name,
  - □ the data expected and
  - □ The data returned
- If Service A knows the Service B's description Service A can communicate with Service B.

Services Communicate



- Messages are independents units of communication
- Once the message is sent the service has no control over the message

Service orientation is a design paradigm

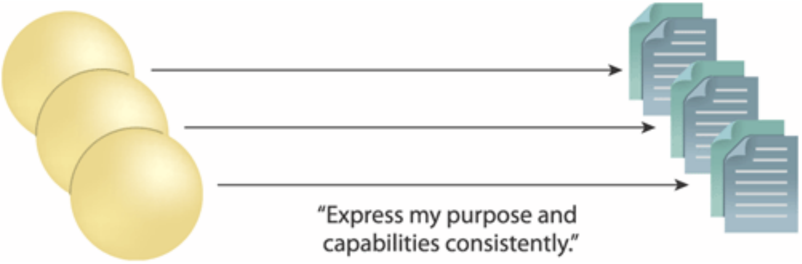
- Design Paradigm
  - bring together ideas on how to decompose and integrate components of programs.
  - a model to define how to solve a class of problems that share a set of common characteristics.
- Expresses in terms of
  - Design Principles / Patterns
  - Components
  - Software Architecture
- Different design paradigms:
  - Object Orientation is the most frequently known
  - State based, Functional, Event based

Service Design Principles

- Standardized Service Contract
- Service Loose Coupling
- Service Abstraction
- Service Reusability
- Service Autonomy
- Service Statelessness
- Service Discoverability
- Service Composability

Standardized Service Contracts

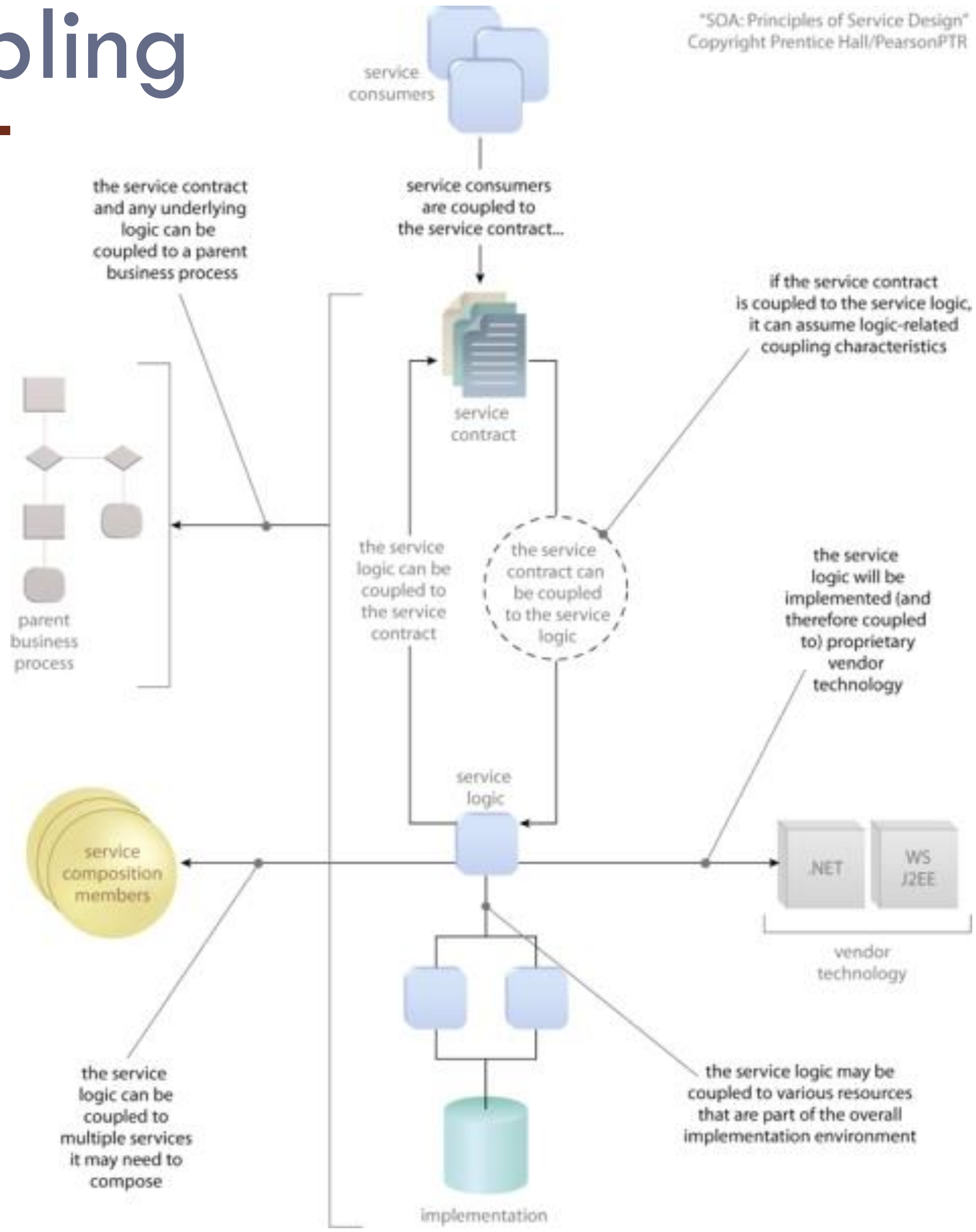
- Services express their purpose and capabilities via a service contract.
- Contract design emphasize:
  - □ How services express functionality
  - □ How data types and models are defined
  - □ How policies are attached
- It is the most fundamental principle.
  - □ Contract standard determines a service's public technical interface.



Source: Thomas Erl

Service Loose Coupling

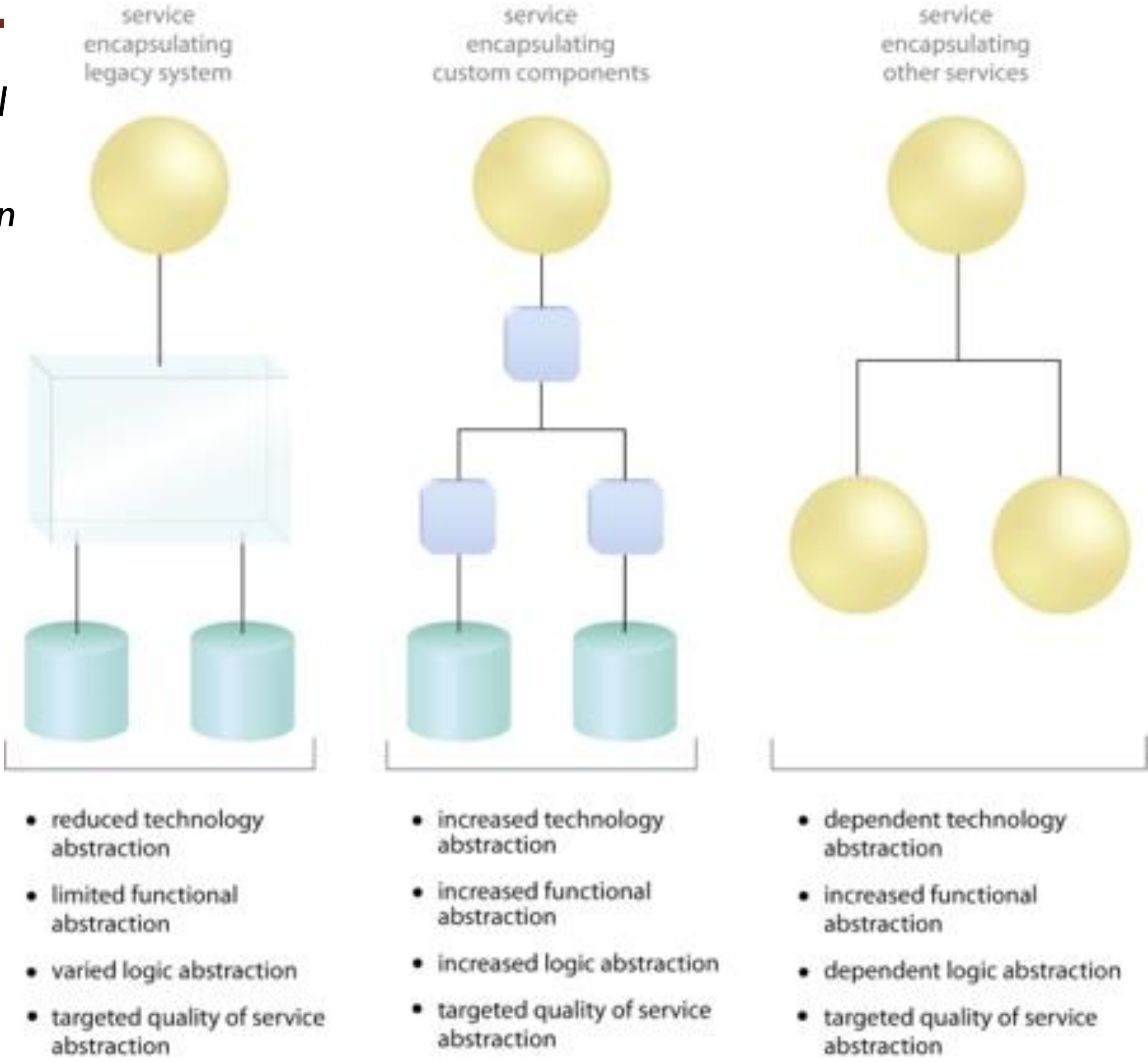
- Coupling:
  - □ Relationship between two components
  - □ Dependency increase with increased/tight coupling
- Loose coupling enables:
  - □ Independent design, evolution


service consumers
service consumers are coupled to the service contract...
the service contract and any underlying logic can be coupled to a parent business process
service contract
if the service contract is coupled to the service logic, it can assume logic-related coupling characteristics
parent business process
the service logic can be coupled to the service contract
the service contract can be coupled to the service logic
the service logic will be implemented (and therefore coupled to) proprietary vendor technology
service logic
service composition members
.NET
WS J2EE
vendor technology
the service logic can be coupled to multiple services it may need to compose
the service logic may be coupled to various resources that are part of the overall implementation environment
implementation

"SOA: Principles of Service Design"
Copyright Prentice Hall/PearsonPTR

Service Abstraction

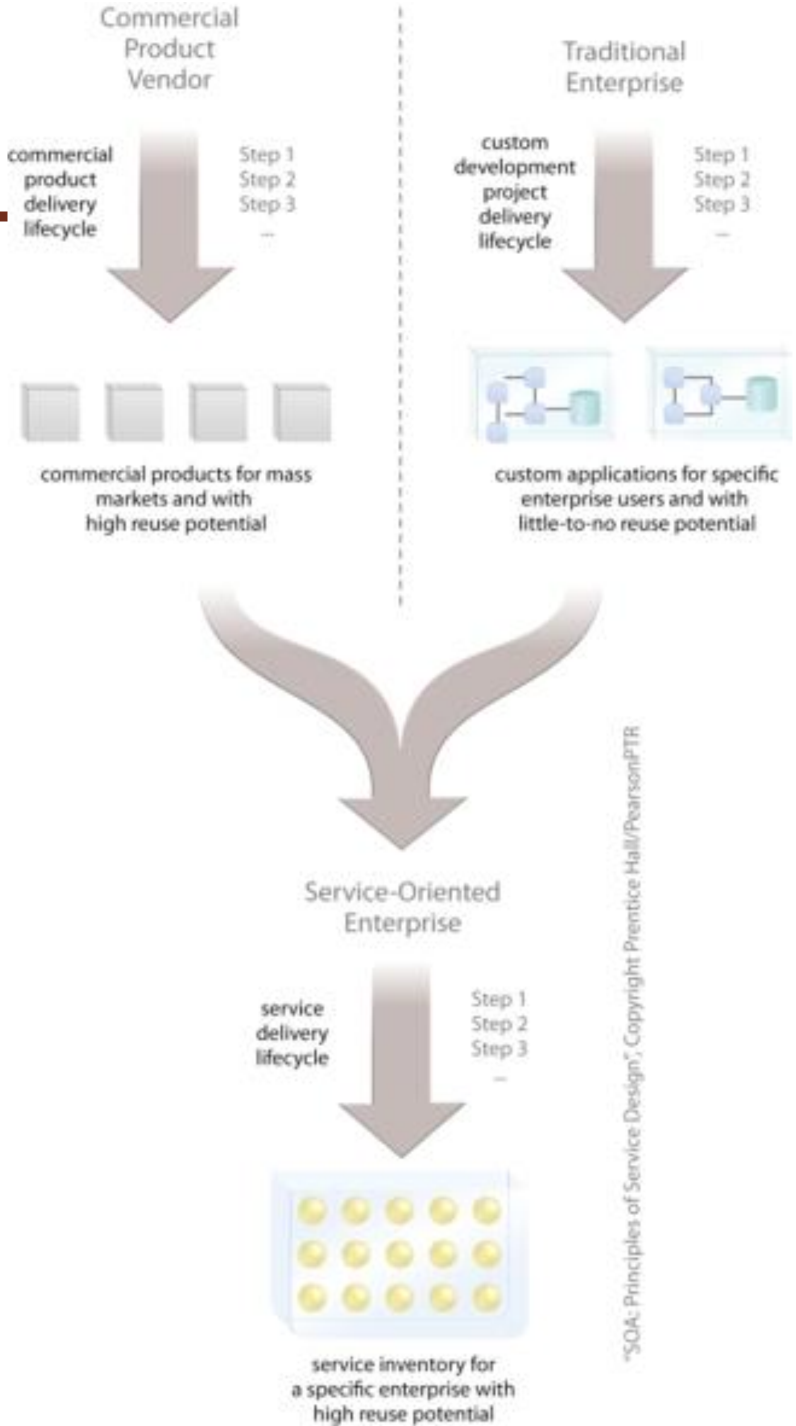
- *“Service contracts only contain essential information and information about services is limited to what is published in service contracts”*
- Avoid the proliferation of unnecessary service information, meta-data.
- Hide as much of the underlying details of a service as possible.
  - □ Enables and preserves the loosely coupled relationships
  - □ Plays a significant role in the positioning and design of service compositions


service encapsulating legacy system
service encapsulating custom components
service encapsulating other services
reduced technology abstraction
limited functional abstraction
varied logic abstraction
targeted quality of service abstraction
increased technology abstraction
increased functional abstraction
increased logic abstraction
targeted quality of service abstraction
dependent technology abstraction
increased functional abstraction
dependent logic abstraction
targeted quality of service abstraction

“SOA: Principles of Service Design”, Copyright Prentice Hall/PearsonPTR

Service Reusability

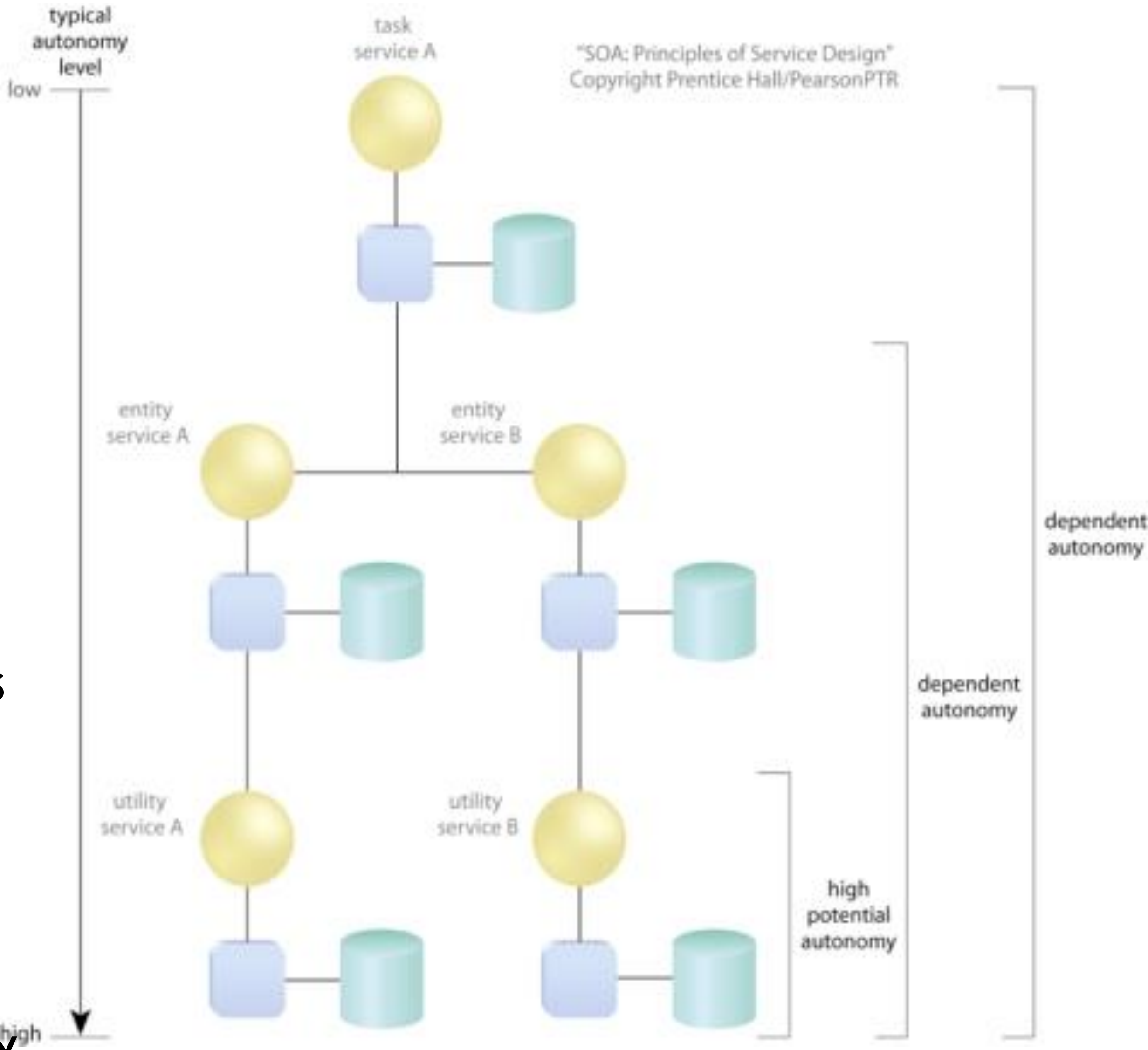
- *"Services contain and express agnostic logic and can be positioned as reusable enterprise resources."*
- Reusable services have the following characteristics:
  - □ Defined by an agnostic functional context
  - □ Logic is highly generic
  - □ Has a generic and extensible contract
  - □ Can be accessed concurrently


Commercial Product Vendor
commercial product delivery lifecycle
Step 1
Step 2
Step 3
...
commercial products for mass markets and with high reuse potential
Traditional Enterprise
custom development project delivery lifecycle
Step 1
Step 2
Step 3
...
custom applications for specific enterprise users and with little-to-no reuse potential
Service-Oriented Enterprise
service delivery lifecycle
Step 1
Step 2
Step 3
...
service inventory for a specific enterprise with high reuse potential

"SOA: Principles of Service Design", Copyright Prentice Hall/PearsonPTR

Service Autonomy

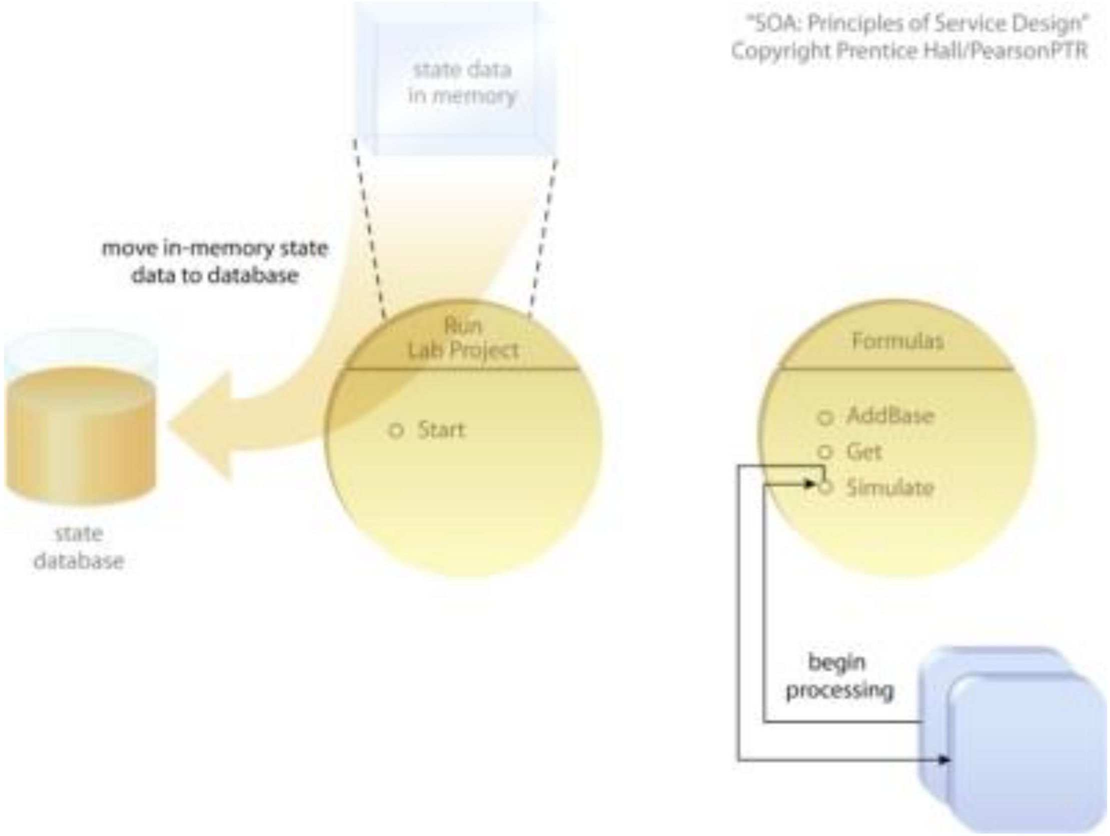
- *"Services exercise a high level of control over their underlying runtime execution environment."*
- Autonomy:
  - □ the ability of a service to carry out its logic independently of outside influences
- To achieve this, services must be more isolated
- Primary benefits
  - □ Increased reliability
  - □ Behavioral predictability



"SOA: Principles of Service Design"
Copyright Prentice Hall/PearsonPTR

Service Statelessness

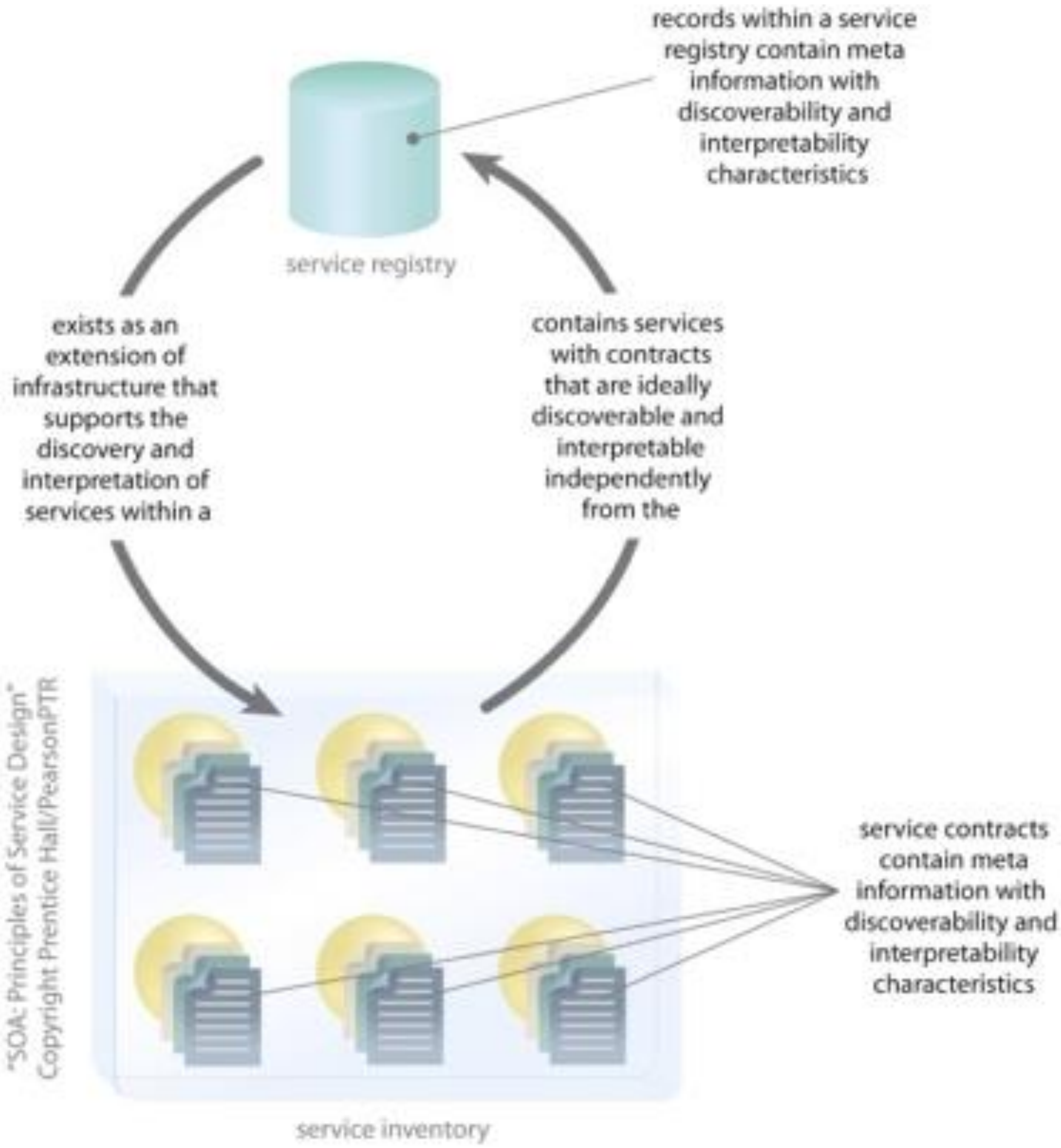
- *"Services minimize resource consumption by deferring the management of state information when necessary."*
- Incorporate state management deferral extensions within a service design
- Goals:
  - □ Increase service scalability
  - □ Support design of agnostic logic and improve service reuse



"SOA: Principles of Service Design"
Copyright Prentice Hall/PearsonPTR

Discoverability

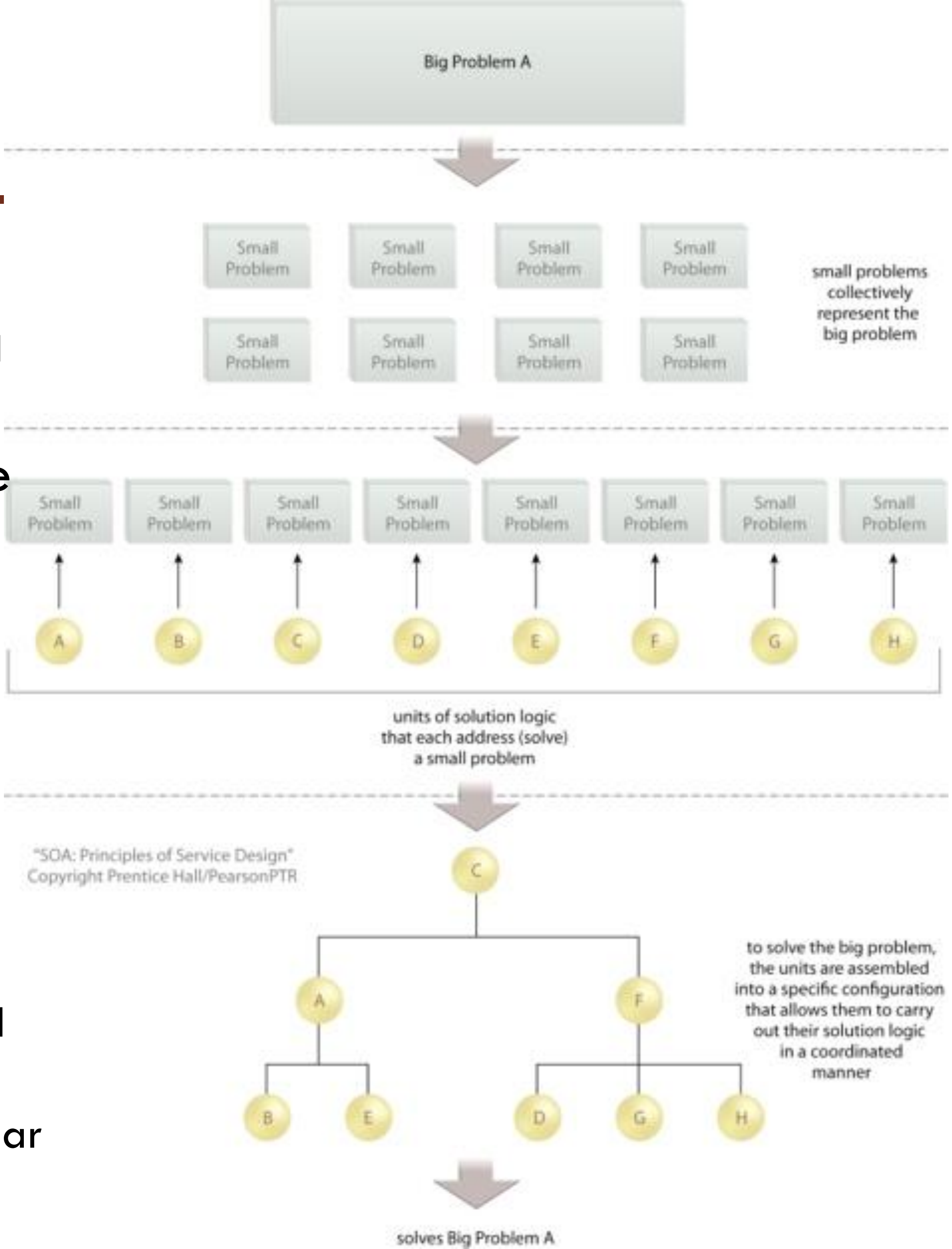
- *"Services are supplemented with communicative meta data by which they can be effectively discovered and interpreted."*
- Services need to be easily:
  - □ Identified
  - □ Understood
- Service design needs to take the "communications quality» of the service into account


records within a service registry contain meta information with discoverability and interpretability characteristics
service registry
exists as an extension of infrastructure that supports the discovery and interpretation of services within a
contains services with contracts that are ideally discoverable and interpretable independently from the
service contracts contain meta information with discoverability and interpretability characteristics
service inventory

"SOA: Principles of Service Design"
Copyright Prentice Hall/PearsonPTR

Composability

- "Services are effective composition participants, regardless of the size and complexity of the composition."
- In the figure we solve a single problem
- Services need to be able to participate in multiple compositions to solve multiple larger problems
  - □ Individual processing should be highly tuned
  - □ Flexible service contracts should allow different types of data exchange requirements for similar functions



small problems collectively represent the big problem

units of solution logic that each address (solve) a small problem

"SOA: Principles of Service Design"
Copyright Prentice Hall/PearsonPTR

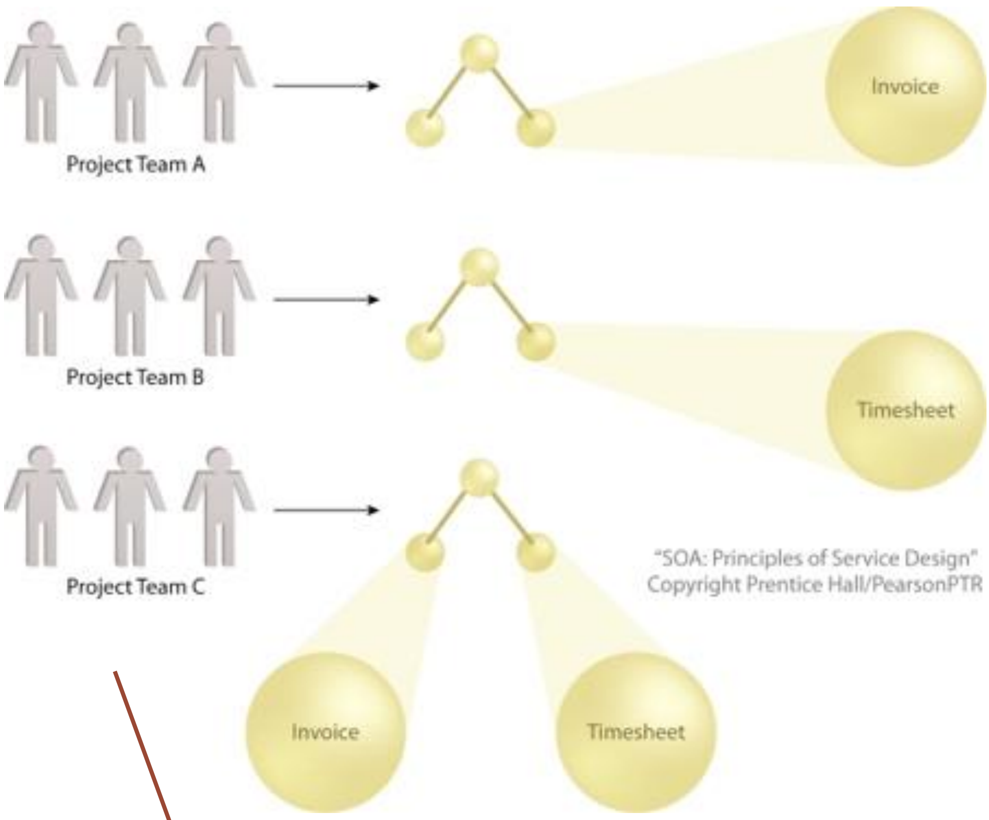
to solve the big problem, the units are assembled into a specific configuration that allows them to carry out their solution logic in a coordinated manner

solves Big Problem A

Benefits of Service-Orientation

- Increasing Intrinsic Interoperability
- Increasing Federation
- Increasing Vendor Diversification Options
- Increasing Business and Technology Domain Alignment
- Increasing ROI
- Increasing Organizational Agility
- Reducing IT Burden

Increased Intrinsic Interoperability

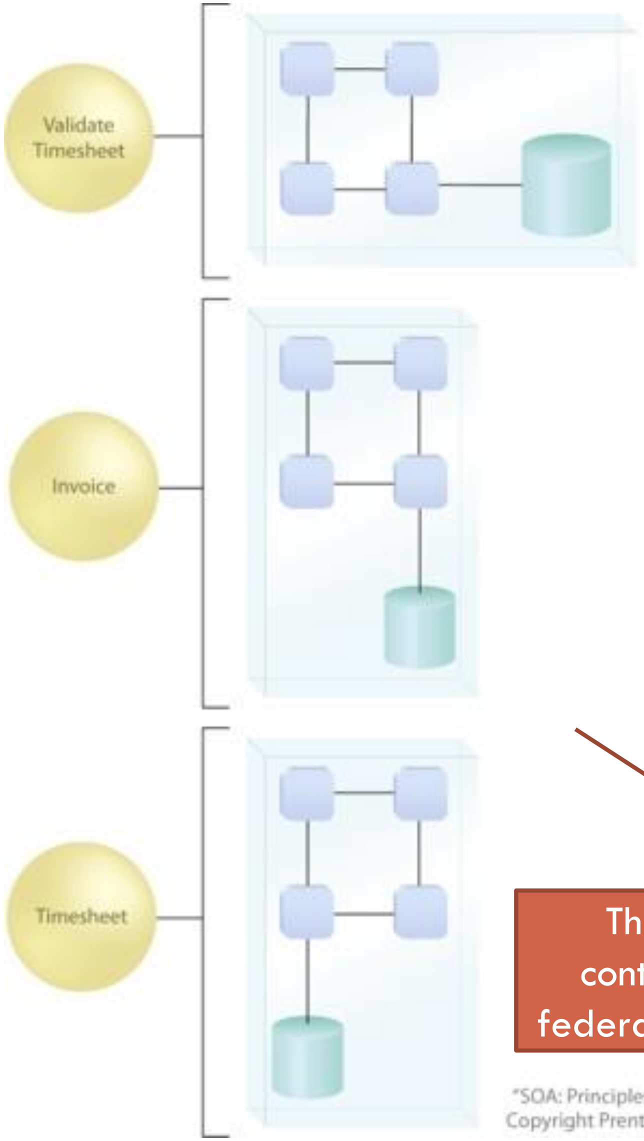


"SOA: Principles of Service Design" Copyright Prentice Hall/PearsonPTR

Project Team C can compose a new application using Invoice and Timesheet

- Interoperability:
  - □ Is the ability to share information is critical.
- SO establish a native mechanism to share information within services.
- Design principles foster interoperability:
  - □ Contract standardization
  - □ Discoverability
  - □ Composability

Increased Federation

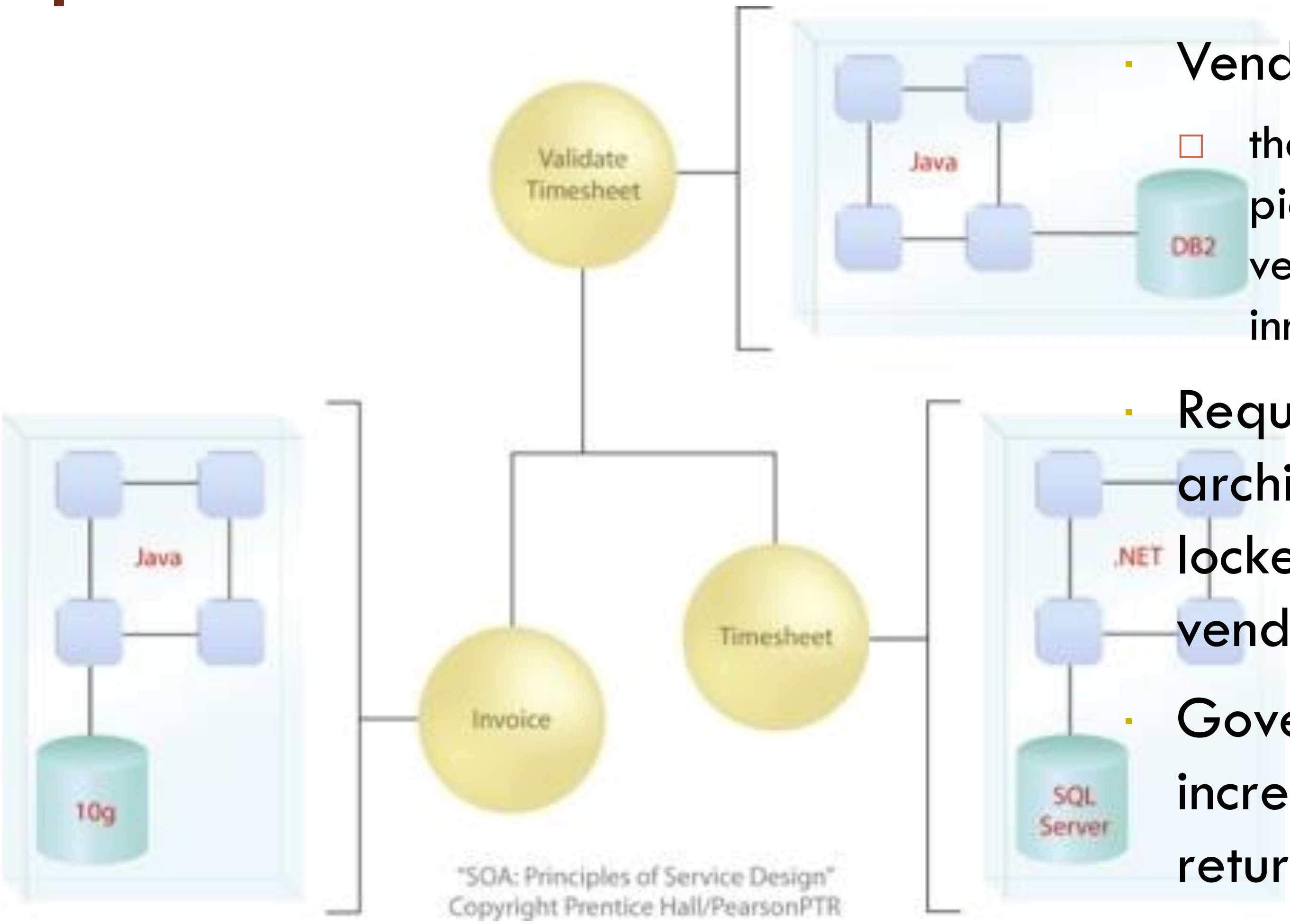


Three service contracts - three federated end points

"SOA: Principles of Service Design"
Copyright Prentice Hall/PearsonPTR

- In a federated environment:
  - □ Resources and applications maintain individual autonomy.
- Service Orientation:
  - □ Wide spread standardized and composable services
  - □ Upfront standardization attention
- Design principles
  - □ Standardized Service Contract,
  - □ Loose Coupling,
  - □ Service Abstraction

Vendor Diversification Options

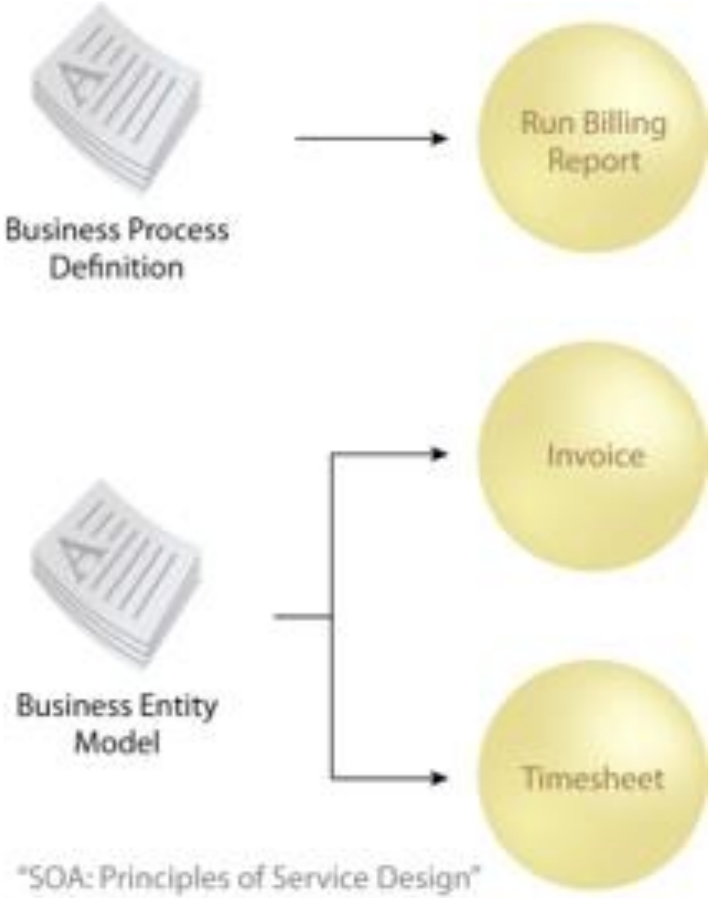


"SOA: Principles of Service Design" Copyright Prentice Hall/PearsonPTR

- Vendor diversification:
  - □ the ability of an organization to pick and choose "best-of-breed" vendor products and technology innovations
- Requires that the technology architecture not be tied or locked into any one specific vendor platform.
- Goverenment autonomy increase life span and financial return of IT.
- Web services framework supports this property.

Business and Technology Domain Alignment

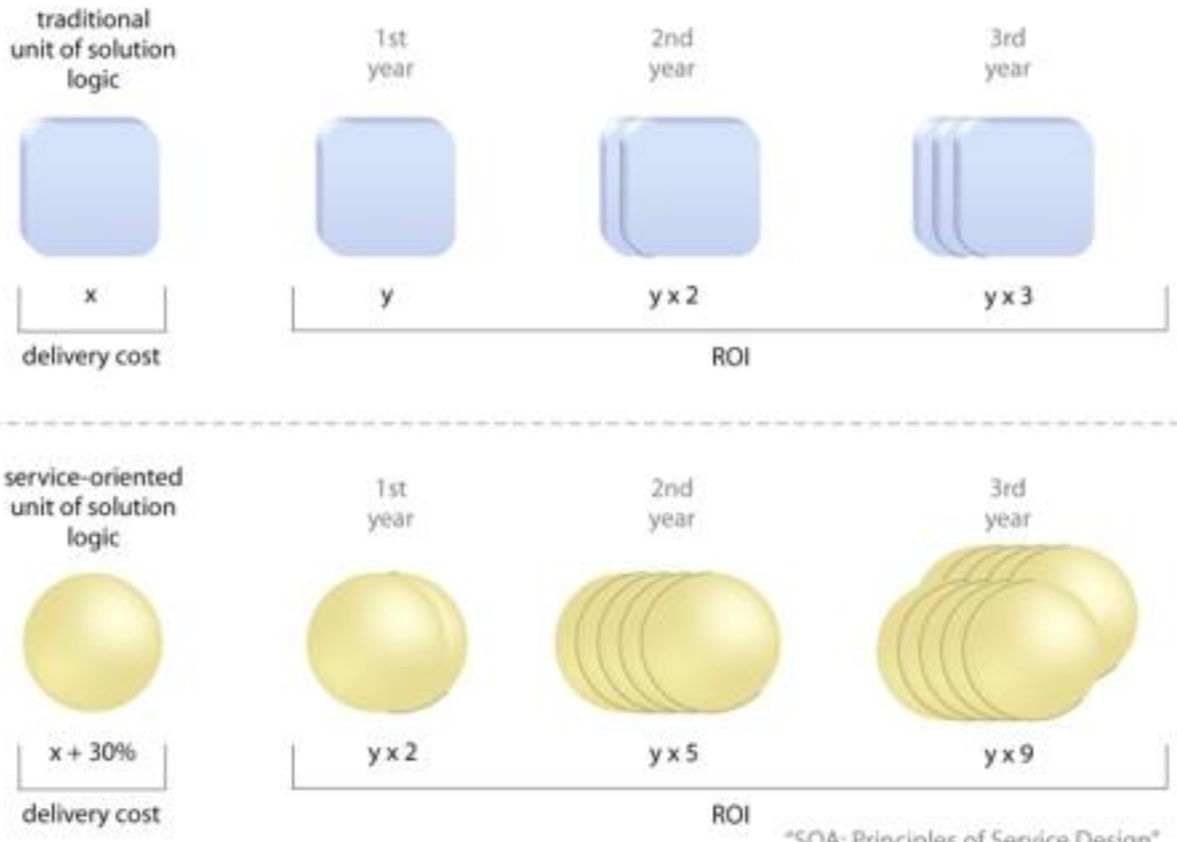
- Services are identified based on the business entities and business processes.
- Service are designed to be interoperable.
- They are capable of aligning new demands by means of new compositions.



"SOA: Principles of Service Design"
Copyright Prentice Hall/PearsonPTR

Return On Investment

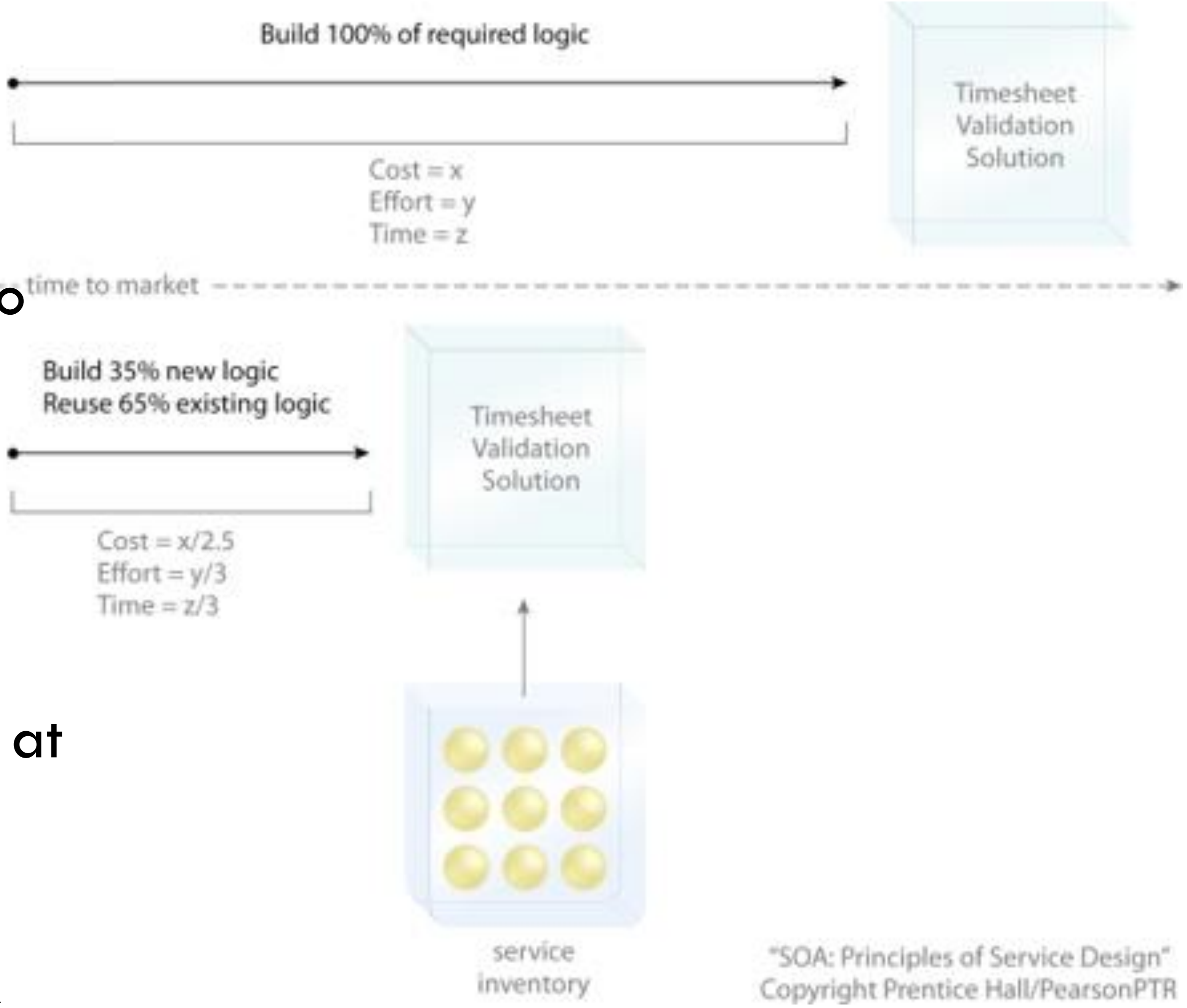
- ROI is a measure to understand how cost effective the solution is
- Reusablity requires investment
  - □ Designing the agnostic solution using service orientation principles requires more upfront effort



"SOA: Principles of Service Design"
Copyright Prentice Hall/PearsonPTR

Organizational Efficiency

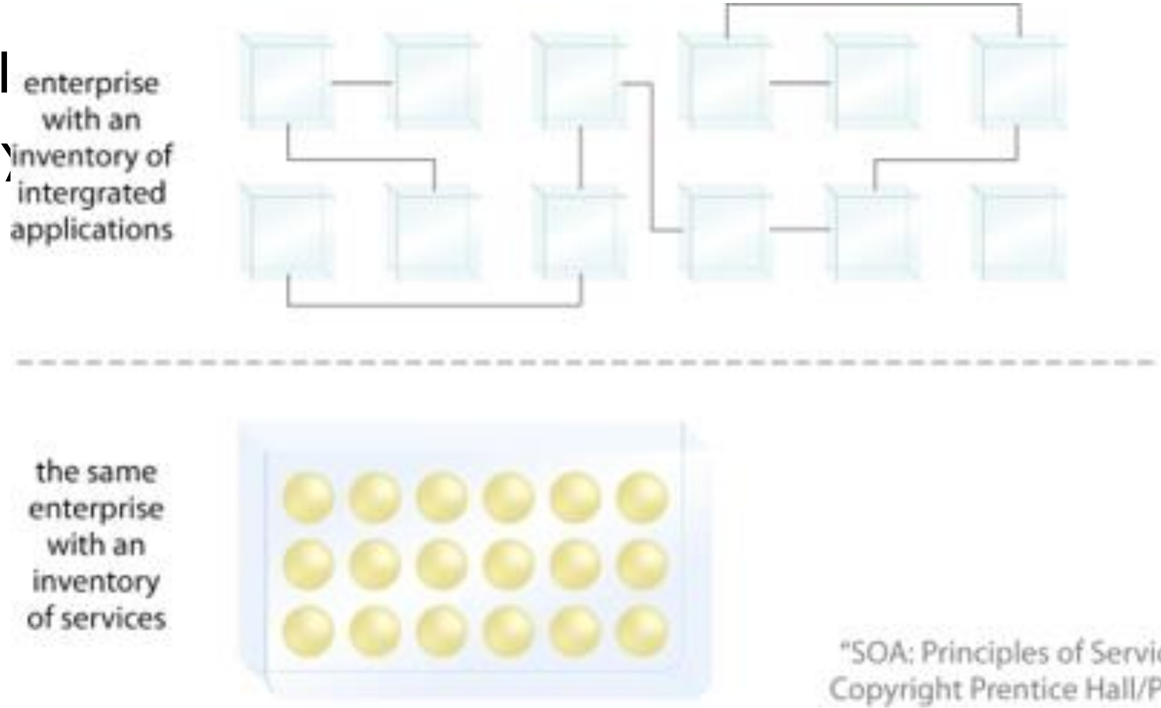
- Efficency:
  - How fast can we deliver?
  - How much we need to spend?
- We have agnostic services
  - reusable assetes reduce time and cost at the same time.
- We increase upfront costs to built services properly



"SOA: Principles of Service Design"
Copyright Prentice Hall/PearsonPTR

Reduced IT Burden

- As reuse become the norm
  - □ The overall size will reduce considerably
- Together with it the overhead for managing multiple environments will reduce.
- Result:
  - □ Reduced operational costs

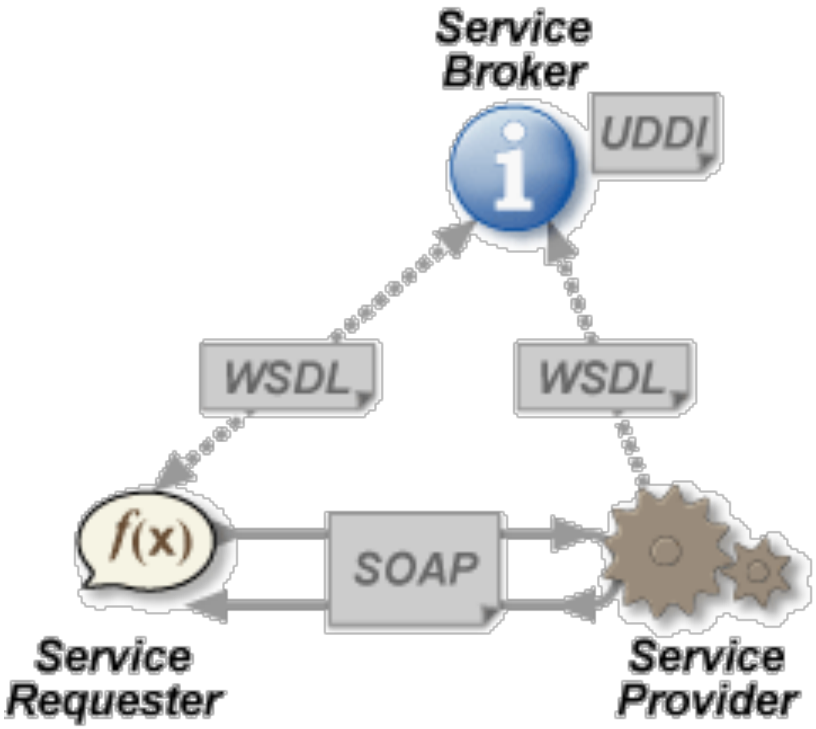

enterprise with an inventory of intergrated applications
the same enterprise with an inventory of services

"SOA: Principles of Service Design"
Copyright Prentice Hall/PearsonPTR

State of SOA - Web Services

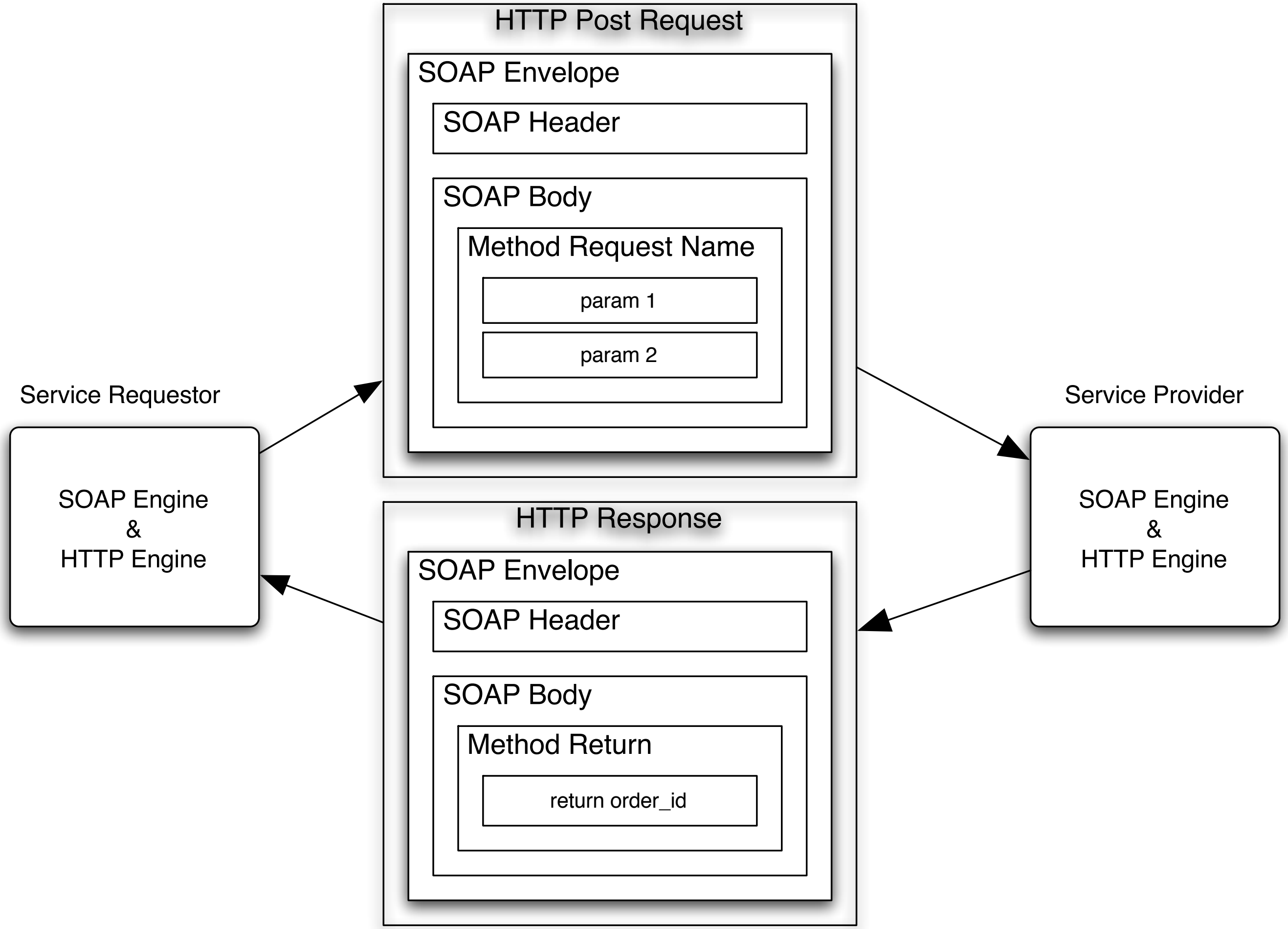
- SOA is agnostic to technology platforms.
  - □ Nevertheless, today's SOA is associated with: Web Services
- First generation web service platform:
  - □ WSDL (Web Service Description Language)- XML based interface definition language, XSD (XML Shema Definition Language) - Specifies how to formally describe elements in XML, SOAP (Simple Object Access Platform) - Protocol specification for exchanging structured information, UDDI (Universal Description, Discovery and Integration) - XML Based registry, BP (WS-I Basic profile) - Interoperability guidance for core services
- Second generation web service platform: 2000 …
  - □ Extensions with quality of service related gaps. Message-level security, cross service transactions, reliable messaging. Labeled as WS-* (such as WS-Policy)
- Light weight alternatives: REST – 2008 …
  - □ JSON instead of XML, OpenAPI 3.0, PP communication, Based on HTTP.
- Reactive Systems: 2015 - …
  - □ Event based architectures:

Web Services Architecture – 2nd Gen



- The service provider sends a WSDL file to UDDI (Universal Description, Discovery, and Integration).
- The service requester contacts UDDI to find out who is the provider and contacts the service provider using the SOAP protocol.
- The service provider validates the service request and sends data using the SOAP protocol.
- This data would be validated again by the service requester using an XSD file.

XML-based APIs …



Early versions of API utilised XML documents → SOAP protocol (W3C standards), or XmlRPC

RESTful Services

- Early XML-based API fell out of favour along with the rise of the number of ‘mobile’ devices (and other ‘non-traditional’ client devices) due to the ‘heavy’ data payload and processing load
- REST is an architectural style of building networked systems - a set of architectural constraints in a protocol built in that style.
- The protocol in REST is HTTP (the core technology that drives the Web)
- Popular form of API … It is popularised as a guide to build modern distributed applications on the Web – let’s work with the components that the Web itself is built in.
- REST itself is not an official standard specification or even a recommendation. It is just a “design guideline” for building a system (or a service in our context) on the Web

Challenges of Service Orientation

- Increased design complexity
- The need for design standards
- The need to identify requirements in advance
- The need for a specific governance structure

Increased Design Complexity

- Emphasis on reuse
  - □ Need services with agnostic logic for different potential customers.

  → Increased level of complexity for services and architectures
- Performance requirements increase
- Reliability issues at peak concurrent usage
- Single point failures – if a reused service fails all reusing services fail
- Increased demands on service hosting
- Versioning issues result in redundant contracts

The Need for Design Standards

- The effective use of services requires standardisation
  - It is healhty for software organizations
  - However it is not a straightforward process
    - Requires a cultural change
    - It is a social problem – most of the time not well understood and undervalued by IT organizations
- Standardization might also create a culture that resists change if you need to change the standard

Requirements First

- It is highly beneficial to create a blueprint for all planned services upfront.
  - Top down - waterfall like - delivery strategy.
  - High level upfront analysis effort is required.
  - Frequently the problems software solves are un-structured
    - We don’t know the formulation before we have the solution
  - Using iterative development approaches might be expensive as major changes can be costly

The Need for A New Governance Structure

- Application Centric development:
  - □ Development by a single project team
  - □ Members know the problem domain well
  - □ Members remains to evolve the application
- Service Centric development:
  - □ Agnostic logic does not belong to a single process
  - □ Domain knowledge is lost
  - □ Team members do not own the service for evolution
  - □ A new governance model is required to maintain services

SOA Manifesto

- Service orientation is a paradigm that frames what you do. Service-oriented architecture (SOA) is a type of architecture that results from applying service orientation.
- We have been applying service orientation to help organizations consistently deliver sustainable business value, with increased agility and cost effectiveness, in line with changing business needs.
- Through our work we have come to prioritize:
  - □ **Business value** over technical strategy
  - □ **Strategic goals** over project-specific benefits
  - □ **Intrinsic interoperability** over custom integration
  - □ **Shared services** over specific-purpose implementations
  - □ **Flexibility** over optimization
  - □ **Evolutionary refinement** over pursuit of initial perfection

We follow these principles:

- Respect the social and power structure of the organization.
- Recognize that SOA ultimately demands change on many levels.
- The scope of SOA adoption can vary. Keep efforts manageable and within meaningful boundaries.
- Products and standards alone will neither give you SOA nor apply the service orientation paradigm for you.
- SOA can be realized through a variety of technologies and standards.
- Establish a uniform set of enterprise standards and policies based on industry, de facto, and community standards.
- Pursue uniformity on the outside while allowing diversity on the inside.
- Identify services through collaboration with business and technology stakeholders.
- Maximize service usage by considering the current and future scope of utilization.
- Verify that services satisfy business requirements and goals.
- Evolve services and their organization in response to real use.
- Separate the different aspects of a system that change at different rates.
- Reduce implicit dependencies and publish all external dependencies to increase robustness and reduce the impact of change.
- At every level of abstraction, organize each service around a cohesive and manageable unit of functionality