Service-Oriented Architecture

The Service Oriented Society

Imagine if we had to do everything
we need to get done by ourselves?

From Craftsmen to Service Providers

- Our society has become what it is today through the forces of
  - Specialization
  - Standardization
  - Scalability
- It is now almost exclusively “service” oriented
  - Transportation
  - Telecommunication
  - Retail
  - Healthcare
  - Financial services
  - …

Attributes of physical services

- Well defined, easy-to-use, somewhat standardized interface
- Self-contained with no visible dependencies to other services
- (almost) Always available but idle until requests come
- “Provision-able”
- Easily accessible and usable readily, no “integration” required
- Coarse grain
- Independent of consumer context,
  - but a service can have a context
- New services can be offered by combining existing services
- Quantifiable quality of service
  - Do not compete on “What” but “How”
  - Performance/Quality
  - Cost
  - …

Context, Composition and State

- Services are most often designed to ignore the context in which they are used
    - It does not mean that services are stateless they are rather context independent !
    - This is precisely the definition of “loosely coupled”
        - Services can be reused in contexts not known at design time
- Value can be created by combining, i.e. “composing” services
    - Book a trip versus book a flight, car, hotel, …

Service Interfaces

- Non proprietary
  - All service providers offer somewhat the same interface
- Highly Polymorphic
  - Intent is enough
- Implementation can be changed in ways that do not break all the service consumers
  - Real world services interact with thousands of consumers
  - Service providers cannot afford to “break” the context of their consumers

Intents and Offers

- Service consumer expresses “intent”
- Service providers define “offers”

- Sometimes a mediator will:
  - Find the best offer matching an intent
  - Advertise an offer in different ways such that it matches different intent
- Request / Response is just a very particular case of an Intent / Offer protocol

Service Orientation and Directories

- Our society could not be “service oriented” without the “Yellow Pages”
- Directories and addressing mechanisms are at the center of our service oriented society
- Imagine
  - Being able to reach a service just by using longitude and latitude coordinates as an addressing mechanism?
  - Only being able to use a service if you can remember its location, phone or fax number?

Service Orientation is scalable

- Consumers can consume and combine a lot of services since they don’t have to know or “learn” how to use a service
- Service providers can offer their services to a lot more consumers because by optimizing
  - The user interface
  - Access (Geographical, Financial, …)
  - They were able to provide the best quality of service and optimize their implementations

So…

- Service orientation allows us
  - Complete freedom to create contexts in which services are uses and combined
  - Express intent rather than specific requests
- Our society should be a great source of inspiration to design modern enterprise systems and architectures or understand what kind of services these systems will consume or provide

The connected (new) world

Over the past 20 years,
everything got connected to everything else

Seamless Connectivity enables “On Demand” Computing

- Use software as you need
- Much lower setup time, forget about
  - Installation
  - Implementation
  - Training
  - Maintenance
- Scalable and effective usage of resources
  - Provision
  - Billed on true usage
  - Parallelism (CPU, Storage, Bandwidth…)

But Seamless Connectivity is also questioning all our beliefs…

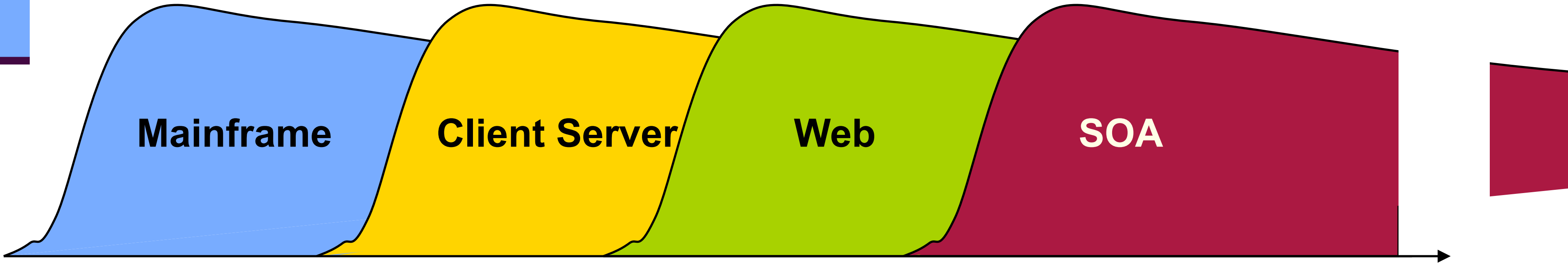
- An application is NOT a single system running on a single device and bounded by a single organization
- Continuum Object … Document
- Messages and Services
  - As opposed « distributed objects »
  - Exchanges becomes peer-to-peer
- Asynchronous communications
- Concurrency becomes the norm while our languages and infrastructures did not plan for it

…we are reaching the point of maximum confusion

- Federation and Collaboration
  - As opposed to « Integration »
- Language(s)
  - Semantic (not syntactic)
  - Declarative and Model driven (not procedural)
- Licensing and Deployment models
- …

So…

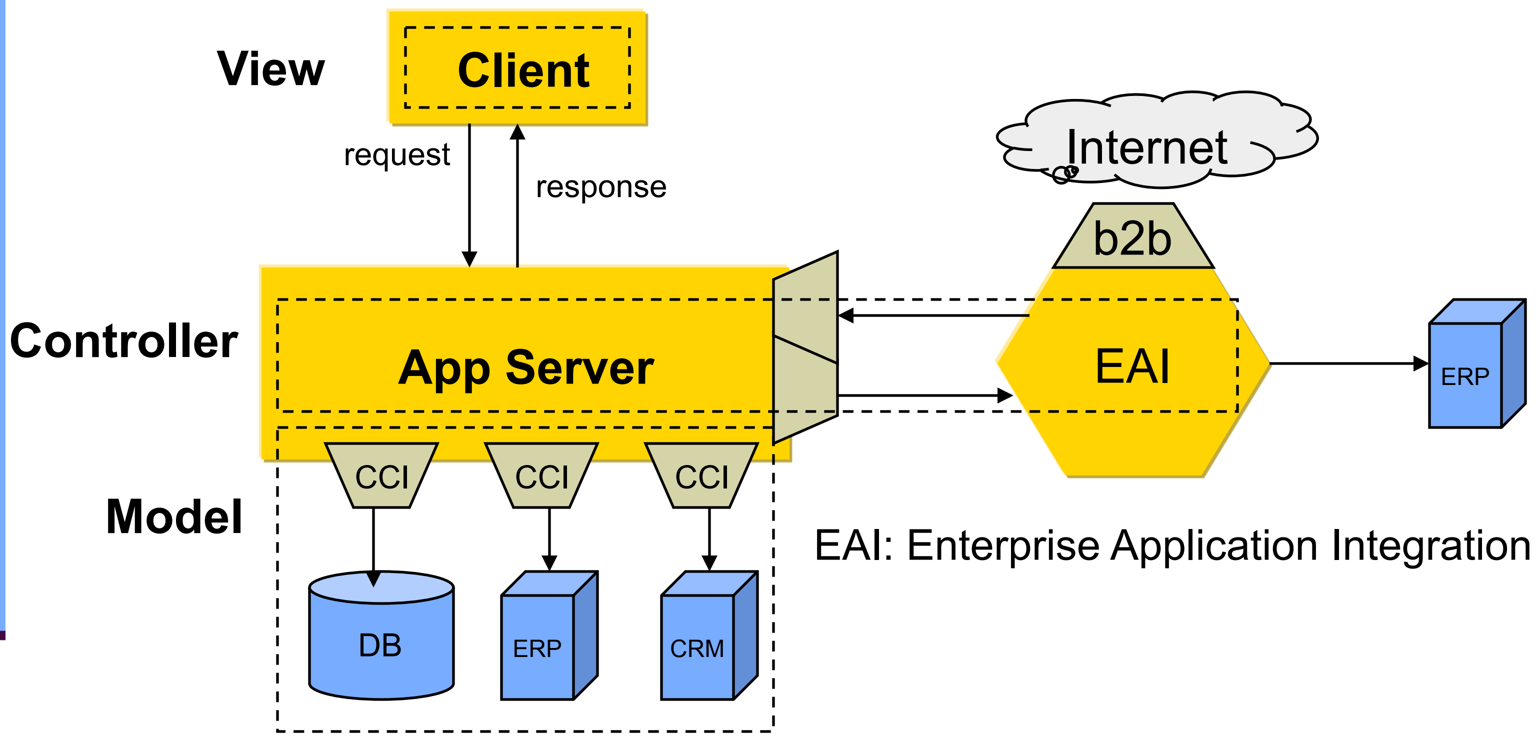
- Today, the value is not defined as much by functionality anymore but by connectivity
  - However, we need a new programming model
- Why SOA today?
  - We are reaching a new threshold of connectivity and computing power



Constructing Software In a Connected World

From Components to Services

Constructing software in the web era (J2EE, .Net, …)



EAI: Enterprise Application Integration

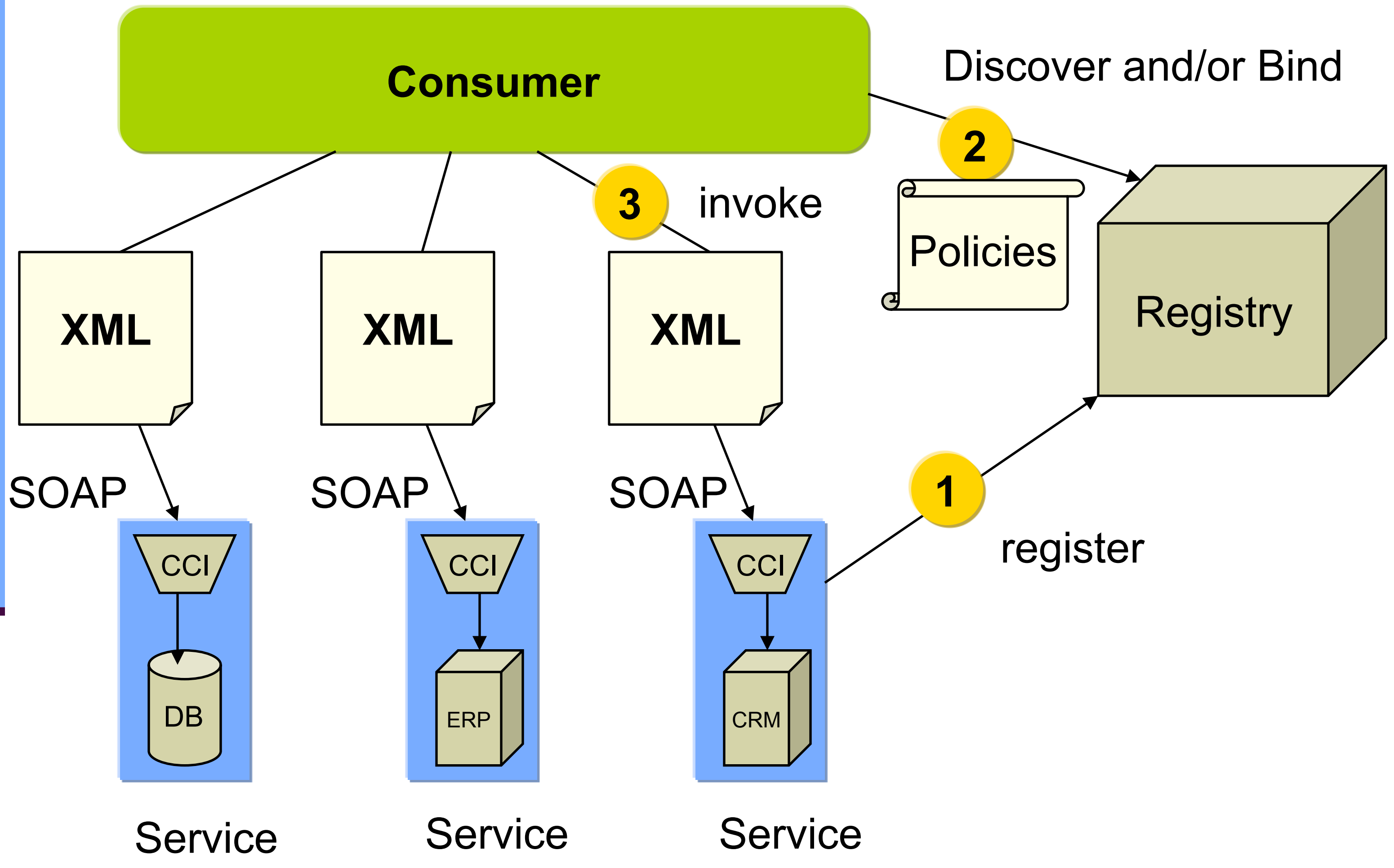
CCI: Client Communication Interface

ERP: Enterprise Resource Planning

Why do we Want to Move to a New Application Model Today?

- We are moving away precisely because of connectivity
  - J2EE, for instance was designed to build 24x7 scalable web-based applications
  - Job well done
- But this is very different from, “I now want my application to execute business logic in many other systems, often dynamically bound to me”
  - JCA (J2EE Connector Architecture) is not enough
  - EAI infrastructures are not enough

A Component now Becomes a Service Running Outside the Consumer Boundaries



From Components to (Web) Services

- Requires a client library
- Client / Server
- Extendable
- Stateless
- Fast
- Small to medium granularity

→

- Loose coupling via
  - Message exchanges
  - Policies
- Peer-to-peer
- Composable
- Context independent
- Some overhead
- Medium to coarse granularity

Web Services: what is changing?

- Loose coupling (of course)
  - Web Services don’t require a CCI (Client side Communication Interface)
    - Very few “gears” needed to consume a service
  - Web Services can accept message content they do not fully understand or support
    - XML, WSDL
  - Web services are very late bound
    - Location is independent of the invocation mechanism
    - Directories

Web Services: What is Changing?

- Peer-to-peer interactions are possible
  - Request / response is an inefficient and very limiting mode of interaction
  - As components coarsen, it is difficult to differentiate a client from a server

What Happens to the Technical Services Typically Provided by an Application Server?

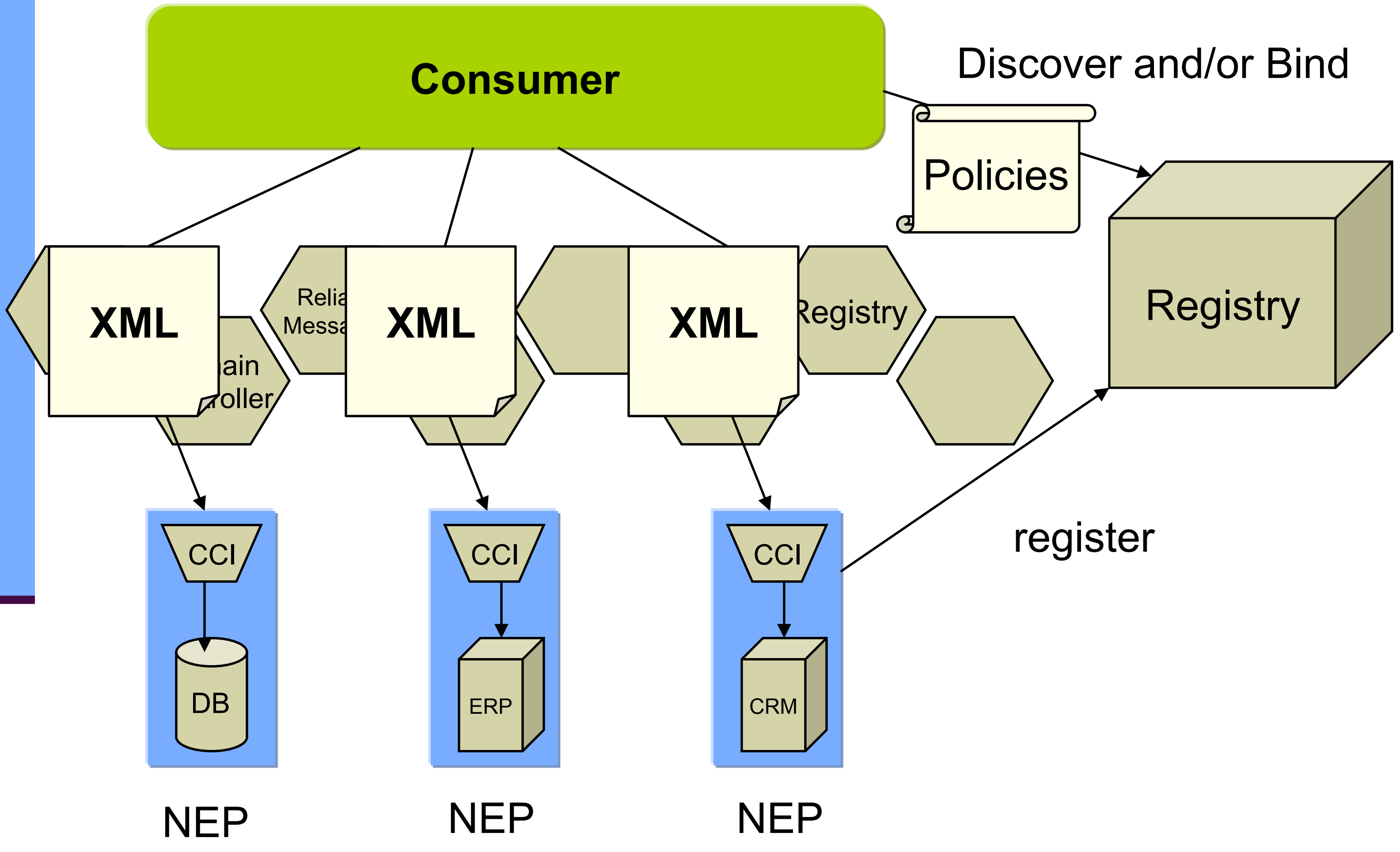
- Transaction
- Security
- Connection pooling
- Naming service
- Scalability and failover
- …

- They become the “Service Fabric”

What about the notion of “Container”? They become Service “Domains”

- The notion of “container” shifts to the notion of “Domain Controller”
  - A domain is a collection of web services which share some commonalities like a “secure domain”
  - A container is a domain with one web service
  - Web Services do not always need a container
- Consumers invoke the domain rather than the service directly
- This concept does not exist in any specification…

A Service Fabric can be more than a Bus with a series of Containers / Adapters



Shift To A Service-Oriented Architecture

From	To
Function oriented	Coordination oriented
Build to last	Build to change
Prolonged development cycles	Incrementally built and deployed
Application silos	Enterprise solutions
Tightly coupled	Loosely coupled
Object oriented	Message oriented
Known implementation	Abstraction

Source: Microsoft (Modified)

So Migrating to SOA

- Would entail
  - Organizing the business logic in a context independent way
    - Typically, model oriented business logic is wrapped behind (web) services
- Re-implementing the controller with “coordination” technologies
- …The view must be completely re-invented

SOA

- A dynamically organized collection of service assets that are composed in different ways to present one or more applications.
- Advantages: Loosely couple, based upon common standards, reuse of existing assets, rapid assembling of new applications
- Weakness: XML verbose, immature