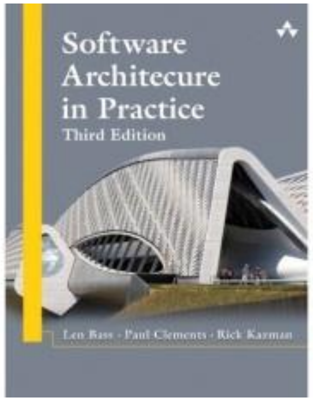
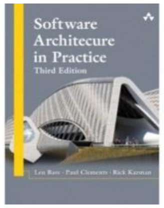

Software Architecure in Practice
Third Edition
Len Bass · Paul Clements · Rick Kazman

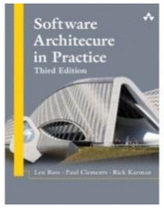
Patterns and Tactics

3-2
Kalıplar ve Taktikler


Software Architecure in Practice
Third Edition
Len Bass · Paul Clements · Rick Kazman

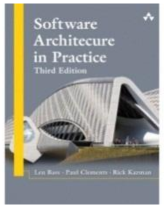
Service Oriented Architecture Pattern

- **Context**: A number of services are offered (and described) by service providers and consumed by service consumers. Service consumers need to be able to understand and use these services without any detailed knowledge of their implementation.
- **Problem**: How can we support interoperability of distributed components running on different platforms and written in different implementation languages, provided by different organizations, and distributed across the Internet?
- How can we locate services and combine (and dynamically recombine) them into meaningful coalitions while achieving reasonable performance, security, and availability?
- **Solution**: $\longrightarrow$



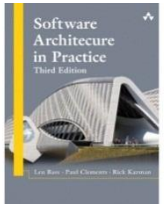
Service Oriented Architecture Pattern

- **Solution**: The service-oriented architecture (SOA) pattern describes a collection of distributed components that provide and/or consume services.
- In an SOA, *service provider* components and *service consumer* components can use different implementation languages and platforms.
- Services are largely standalone: service providers and service consumers are usually deployed independently, and often belong to different systems or even different organizations.
- Components have interfaces that describe the services they request from other components and the services they provide.
- A service's quality attributes can be specified and guaranteed with a service-level agreement (SLA). In some cases, these are legally binding. Components achieve their computation by requesting services from one another.



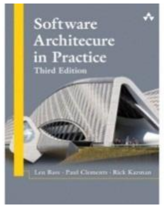
Service Oriented Architecture Pattern

- The elements in this pattern include service providers and service consumers, which in practice can take different forms, from JavaScript running on a web browser to CICS transactions running on a mainframe.
- In addition to the service provider and service consumer components, an SOA application may use specialized components that act as intermediaries and provide infrastructure services:



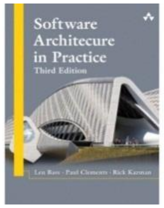
Service Oriented Architecture Pattern

- ….intermediaries and provide infrastructure services:
- Service invocation can be mediated by an *enterprise service bus* (ESB). An ESB routes messages between service consumers and service providers.
- In addition, an ESB can convert messages from one protocol or technology to another, perform various data transformations (e.g., format, content, splitting, merging), perform security checks, and manage transactions.
- Using an ESB promotes interoperability, security, and modifiability. Of course, communicating through an ESB adds overhead thereby lowering performance, and introduces an additional point of failure. When an ESB is not in place, service providers and consumers communicate with each other in a point-to-point fashion.



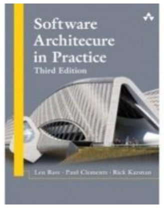
Service Oriented Architecture Pattern

- …. infrastructure services:
- To improve the independence of service providers, a *service registry* can be used in SOA architectures.
  - The registry is a component that allows services to be registered at runtime.
  - This enables runtime discovery of services, which increases system modifiability by hiding the location and identity of the service provider.
  - A registry can even permit multiple live versions of the same service.



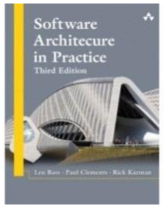
Service Oriented Architecture Pattern

- …. infrastructure services:
- • An *orchestration server* (or orchestration engine) orchestrates the interaction among various service consumers and providers in an SOA system.
- It executes scripts upon the occurrence of a specific event (e.g., a purchase order request arrived). Applications with well-defined business processes or workflows that involve interactions with distributed components or systems gain in modifiability, interoperability, and reliability by using an orchestration server.
- Many commercially available orchestration servers support various workflow or business process language standards.



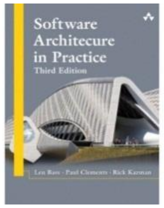
Service Oriented Architecture Pattern

- The basic types of connectors used in SOA are these:
- *SOAP.* The standard protocol for communication in the web services technology. Service consumers and providers interact by exchanging request/reply XML messages typically on top of HTTP.
- *Representational State Transfer (REST).* A service consumer sends nonblocking HTTP requests. These requests rely on the four basic HTTP commands (POST, GET, PUT, DELETE) to tell the service provider to create, retrieve, update, or delete a resource.
- *Asynchronous messaging*, a "fire-and-forget" information exchange. Participants do not have to wait for an acknowledgment of receipt, because the infrastructure is assumed to have delivered the message successfully. The messaging connector can be point-to-point or publish-subscribe.



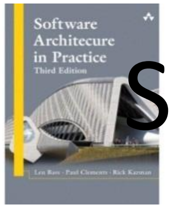
Service Oriented Architecture Pattern

- In practice, SOA environments may involve a
  - mix of the three connectors (SOAP, REST, asynchronous messagging) just listed,
  - along with legacy protocols and other communication alternatives (e.g., SMTP, Simple Mail Transfer Protocol).
  - Commercial products such as IBM's WebSphere MQ, Microsoft's MSMQ (Microsoft Message Queuing), or Apache's ActiveMQ are infrastructure components that provide asynchronous messaging.
-



Service Oriented Architecture Pattern

- As you can see, the SOA pattern can be quite complex to design and implement (due to dynamic binding and the concomitant use of metadata).
- Other potential problems with this pattern include the performance overhead of the middleware that is interposed between services and clients and the lack of performance guarantees (because services are shared and, in general, not under control of the requester).



Service Oriented Architecture Example

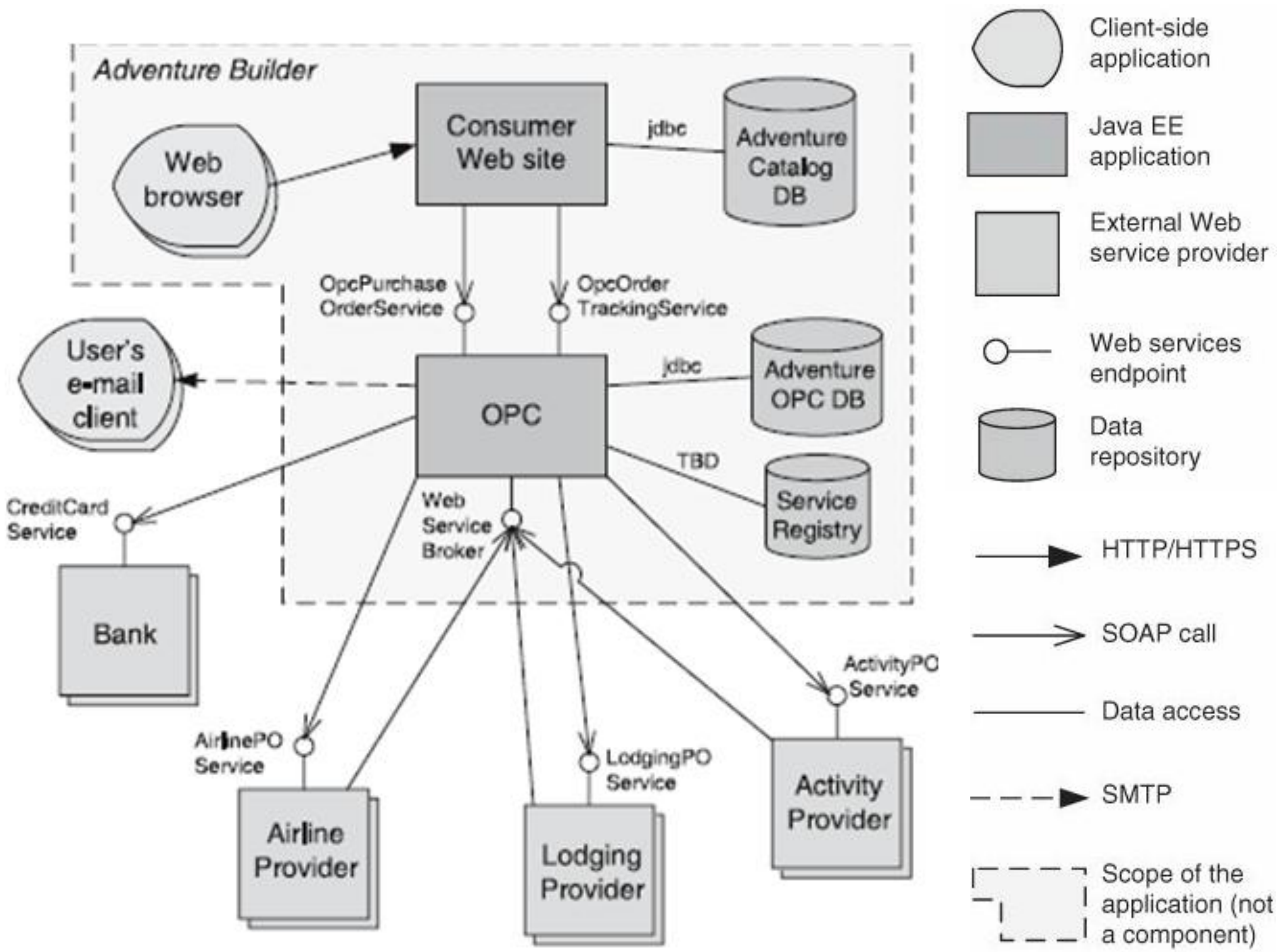
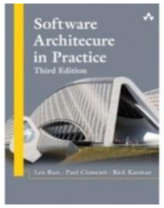
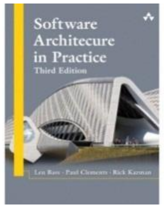


Figure 13.11. SOA view for the Adventure Builder system. OPC stands for "Order Processing Center."



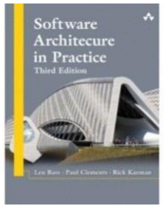
Service Oriented Architecture

- Figure 13.11 hows the SOA view of a system called Adventure Builder.
- Adventure Builder allows a customer on the web
  - to assemble a vacation
    - by choosing an activity and lodging at and transportation to a destination.
- The Adventure Builder system interacts with external service providers
  - to construct the vacation, and
  - with bank services to process payment.



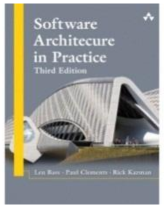
Service Oriented Architecture

- Figure 13.11
- Adventure Builder allows a customer on the web
  - to assemble a vacation
    - by choosing an activity and lodging at and transportation to a destination.
- The Adventure Builder system interacts with external service providers
  - to construct the vacation, and
  - with bank services to process payment.
- The central OPC (Order Processing Center) component coordinates the interaction
  - with internal and external service consumers and providers. (Note that the external providers can be legacy mainframe systems, Java systems, .NET systems, and so on).
- The nature of these external components is transparent because SOAP provides the necessary interoperability.



Service Oriented Architecture Solution - 1

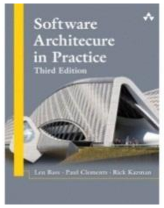
- Overview: Computation is achieved by a set of cooperating components that provide and/or consume services over a network.
- Elements:
- Components:
    - *Service providers,* which provide one or more services through published interfaces.
    - *Service consumers*, which invoke services directly or through an intermediary.
        - *Service providers* may also be service consumers.
  - *ESB*, which is an intermediary element that can route and transform messages between service providers and consumers.
  - *Registry of services,* which may be used by providers to register their services and by consumers to discover services at runtime.
  - *Orchestration server,* which coordinates the interactions between service consumers and providers based on languages for business processes and workflows.



Service Oriented Architecture Solution - 2

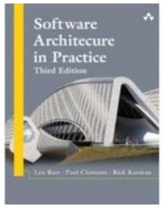
– Connectors:

- *SOAP (Simple Object Access Protocol) connector*, which uses the SOAP protocol for synchronous communication between web services, typically over HTTP.
- *REST (Representation State Transfer) connector,* which relies on the basic request/reply operations of the HTTP protocol.
- *Asynchronous messaging connector,* which uses a messaging system to offer point-to-point or publish-subscribe asynchronous message exchanges.



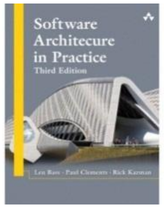
Service Oriented Architecture Solution - 3

- Relations: Attachment of the different kinds of components available to the respective connectors
- Constraints: Service consumers are connected to service providers, but intermediary components (e.g., ESB, registry, orchestration server) may be used.
- Weaknesses:
  - SOA-based systems are typically complex to build.
  - You don't control the evolution of independent services.
  - There is a performance overhead associated with the middleware, and services may be performance bottlenecks, and typically do not provide performance guarantees.

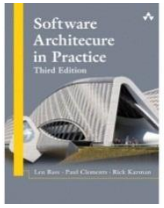


Service Oriented Architecture Solution - 3

- The main benefit and the major driver of SOA is interoperability.
- Because service providers and service consumers may run on different platforms, service-oriented architectures often integrate a variety of systems, including legacy systems.
- SOA also offers the necessary elements to interact with external services available over the Internet.
- Special SOA components such as the registry or the ESB also allow dynamic reconfiguration, which is useful when there's a need to replace or add versions of components with no system interruption.

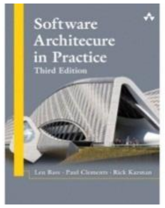


Map-Reduce Pattern

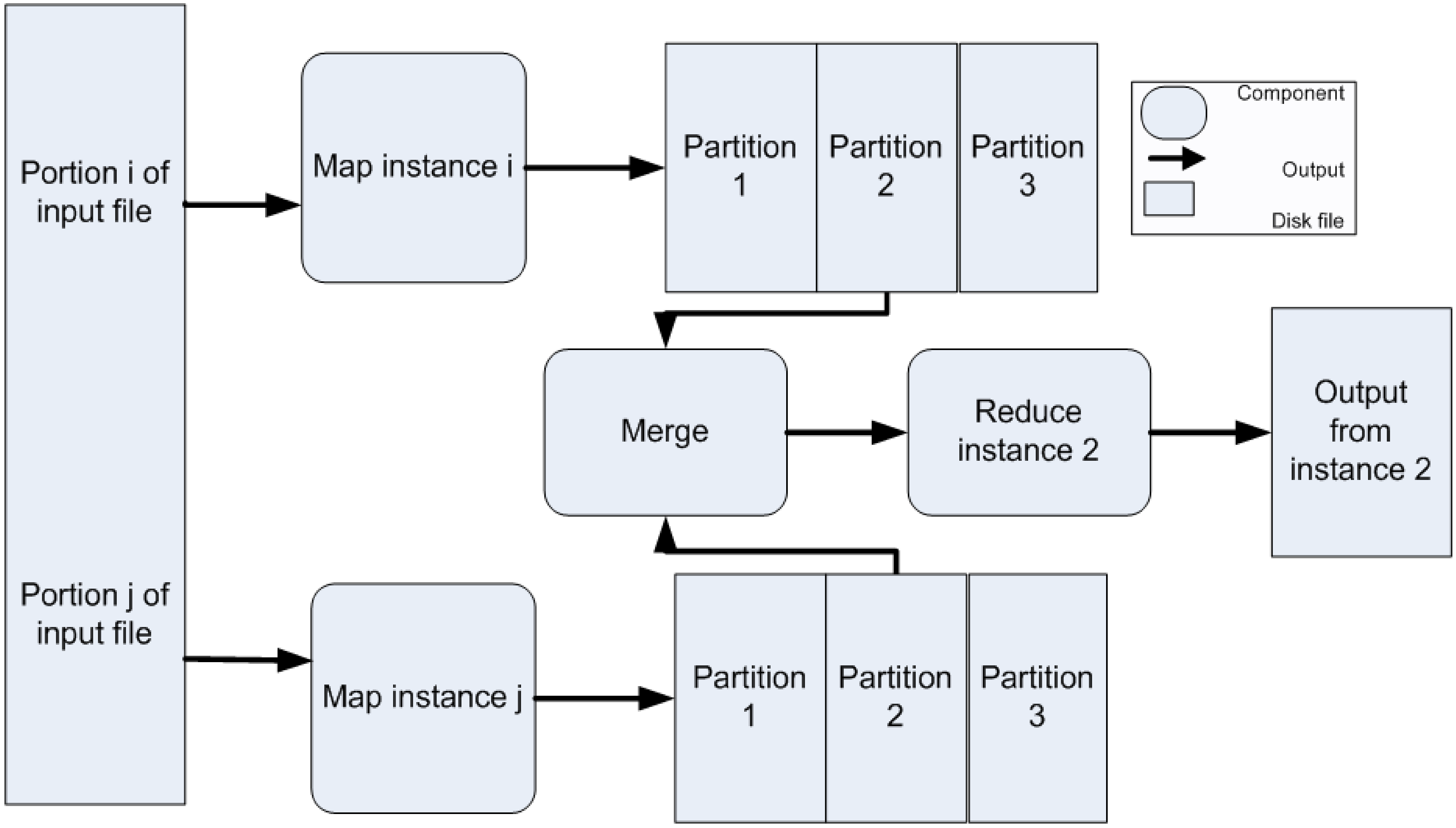

Software Architecure in Practice
Third Edition
Len Bass · Paul Clements · Rick Kazman

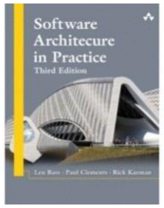
Map-Reduce Pattern

- **Context:** Businesses have a pressing need to quickly analyze enormous volumes of data they generate or access, at petabyte scale.
- **Problem:** For many applications with ultra-large data sets, sorting the data and then analyzing the grouped data is sufficient. The problem the map-reduce pattern solves is to efficiently perform a distributed and parallel sort of a large data set and provide a simple means for the programmer to specify the analysis to be done.
- **Solution:** The map-reduce pattern requires three parts:
  - A *specialized infrastructure* takes care of allocating software to the hardware nodes in a massively parallel computing environment and handles sorting the data as needed.
  - A *programmer specified component* called *the map* which filters the data to retrieve those items to be combined.
  - A *programmer specified component* called *reduce* which combines the results of the map



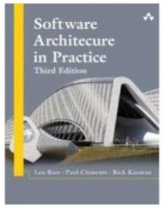
Map-Reduce Example




Software Architecure in Practice
Third Edition
Len Bass · Paul Clements · Rick Kazman

Map-Reduce Solution - 1

- Overview: The map-reduce pattern provides a framework for analyzing a large distributed set of data that will execute in parallel, on a set of processors. This parallelization allows for low latency and high availability. The map performs the extract and transform portions of the analysis and the reduce performs the loading of the results.
- Elements:
  - Map is a function with multiple instances deployed across multiple processors that performs the extract and transformation portions of the analysis.
  - Reduce is a function that may be deployed as a single instance or as multiple instances across processors to perform the load portion of extract-transform-load.
  - The infrastructure is the framework responsible for deploying map and reduce instances, shepherding the data between them, and detecting and recovering from failure.



Map-Reduce Solution - 2

- Relations:
  - Deploy on is the relation between an instance of a map or reduce function and the processor onto which it is installed.
  - Instantiate, monitor, and control is the relation between the infrastructure and the instances of map and reduce.
- Constraints:
  - The data to be analyzed must exist as a set of files.
  - Map functions are stateless and do not communicate with each other.
  - The only communication between map reduce instances is the data emitted from the map instances as <key, value> pairs.
- Weaknesses:
  - If you do not have large data sets, the overhead of map-reduce is not justified.
  - If you cannot divide your data set into similar sized subsets, the advantages of parallelism are lost.
  - Operations that require multiple reduces are complex to orchestrate.