Cloud Computing: SOA Design Patterns, Part II

Vijay Dialani, PhD
Boise State University
vijaydialani@boisestate.edu

Scalability Design Patterns

We'll discuss the following patterns:

Decoupled Invocation—Handle normal request loads, peak request loads, and continuous periods of time at high load without failing

Parallel Pipelines —Build services that maintain state and high throughput

Gridable Service—Build services to handle computationally intense tasks in a scalable manner

Service Instance—Build services that are scalable in a simple and cost-effective way

Virtual Endpoint—Provide services with location transparency that gracefully recover from failure without affecting service consumers

Service Watchdog—Increase availability and identify and resolve problems and failures that are service-specific

Decoupled Invocation – Use case

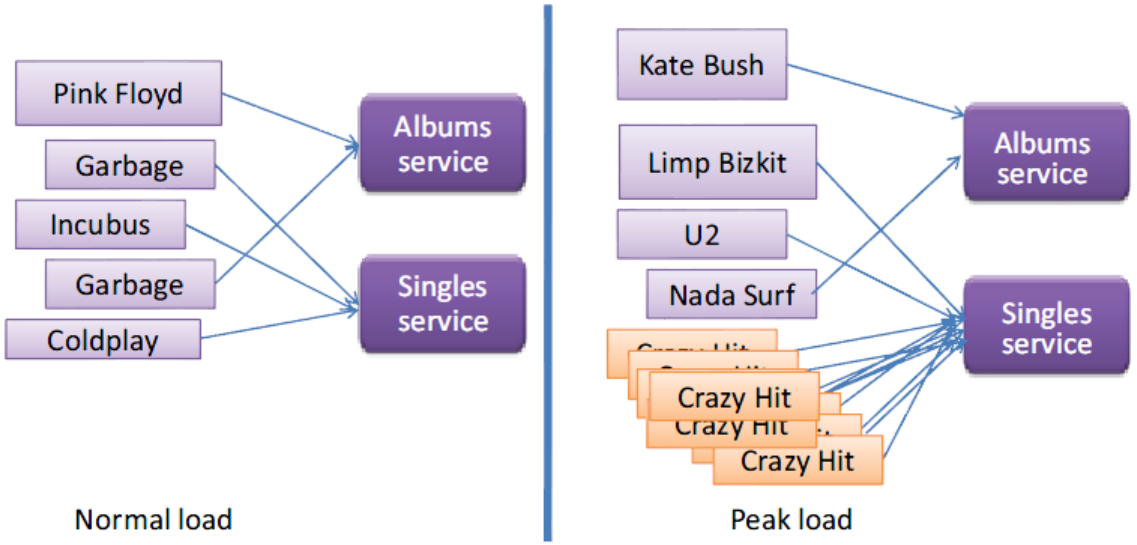


Figure 3.2 **A music store's service-request loads under normal conditions versus peak loads when a popular song is released.**

Decoupled Invocation

Problem:
How can a service handle normal request loads, peak request loads, and a continuous period of high load without failing?

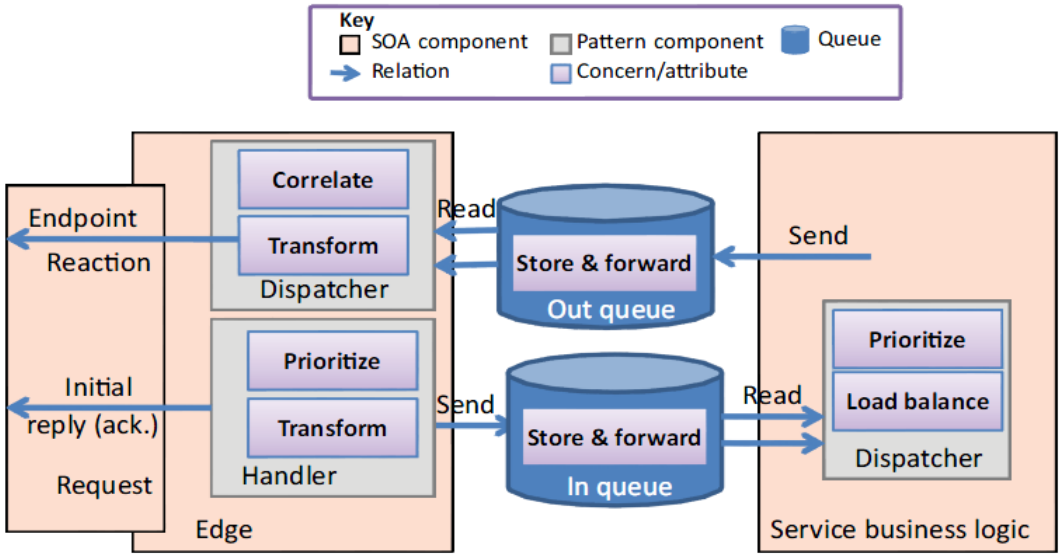


Figure 3.3 The edge component contains a handler that accepts incoming messages, acknowledges them, and queues them. The business logic then reads the queued messages at its own pace. The queue is also used for the responses.

Solution:
Utilize the Decoupled Invocation pattern and separate replies from requests: acknowledge receipt at the service edge, put the request on a reliable queue, and then load-balance and prioritize the handler components that read from the queue.

Assignment-2 Discussions

- How will we adapt the decoupled invocation pattern to the Social Network developed as a part of assignment-1?
- Which queues will we create for request and response?
- Which serialized classes do we need?
- How will we monitor queue performance?

Parallel Pipelines – Use case

Problem:
How can you build services that maintain state and high throughput?

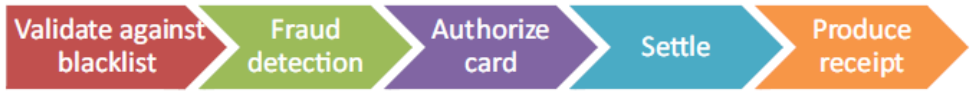


Figure 3.4 Nominal flow for credit card processing in a credit card clearinghouse

Parallel Pipelines

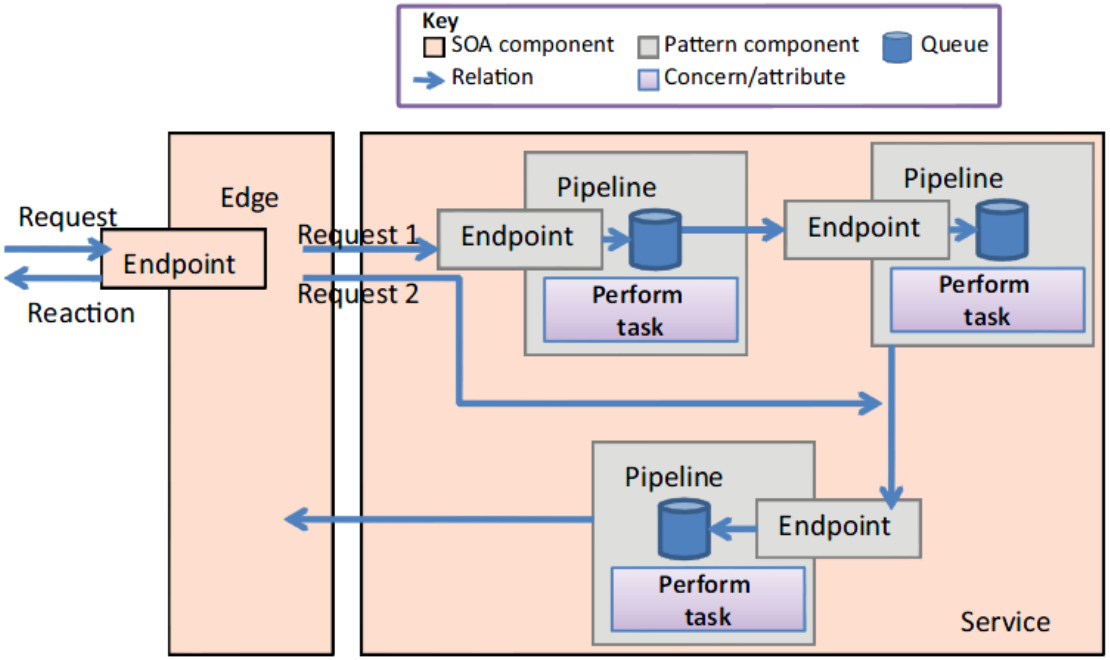


Figure 3.5 With the Parallel Pipelines pattern, the processing is broken into subtasks that are connected by queues to form a processing pipeline. Note that different requests can have different flows of tasks.

Parallel Pipelines

The following are advantages of the Parallel Pipelines approach:

1. The pipelines pattern is relatively simple to implement.

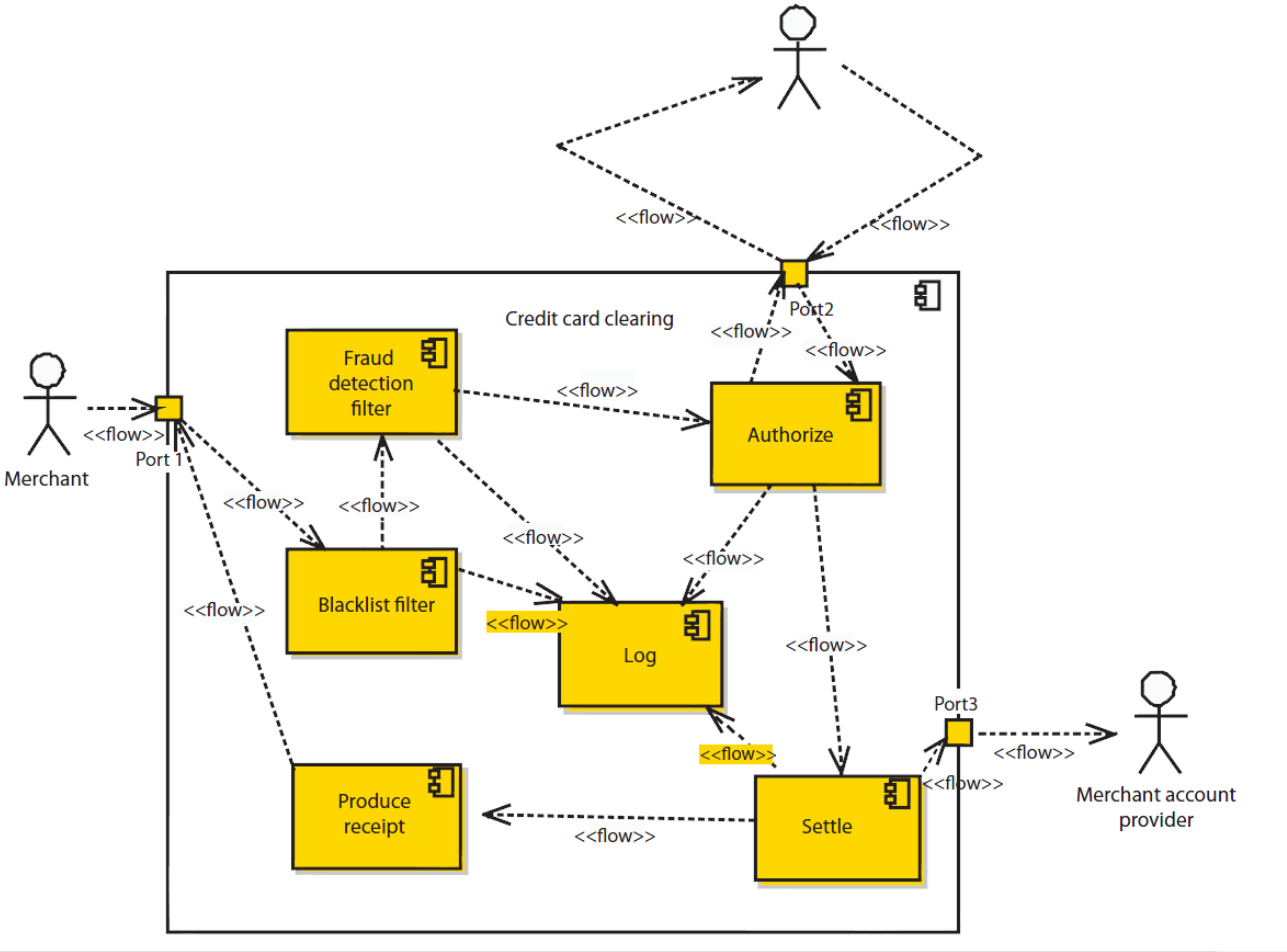
2. Pipelines are easy to test because they operate independently (you can test them with the same technologies and principles you use to test the services that include the pipelines).

3. Because the overall problem is broken into subtasks, each pipeline component tends to be simpler.

4. To scale the solution, you can distribute the pipeline across as many servers as needed.

5. When you need to scale the solution, the simplest option is to put each pipeline on its own server.

Parallel Pipelines



Gridable Service

Problem

How can you build services to handle computationally intense tasks in a scalable manner?

Solution

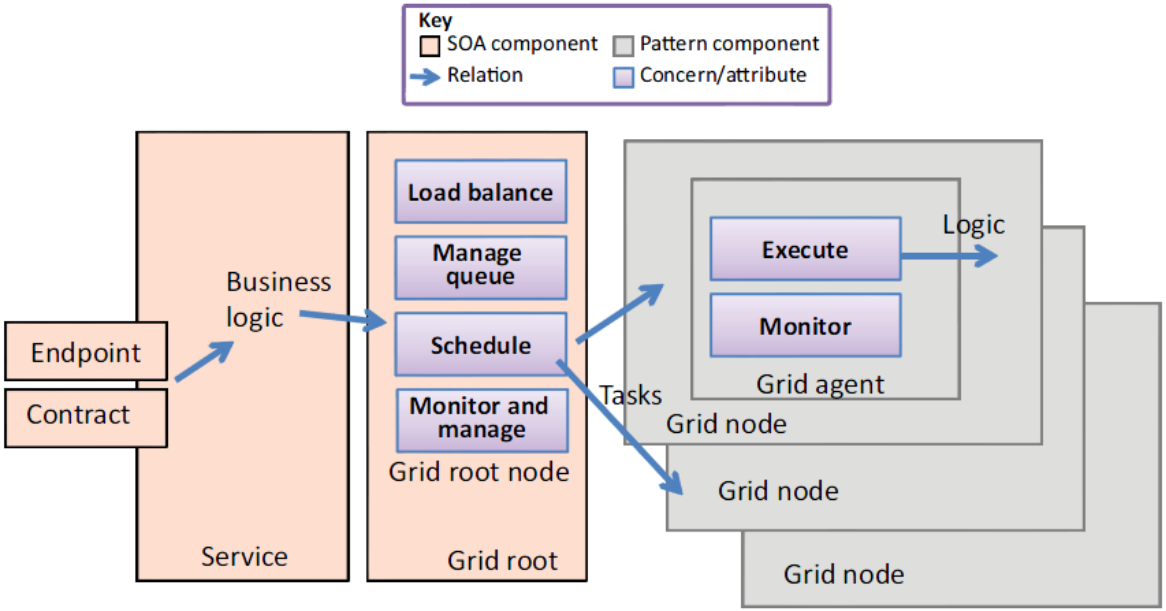


Figure 3.7 When the business logic within the service has to invoke a computationally intensive task, it creates a job on the grid root. The grid root manages all the resources within the grid or compute cluster and executes the task efficiently.

Service Instance Pattern

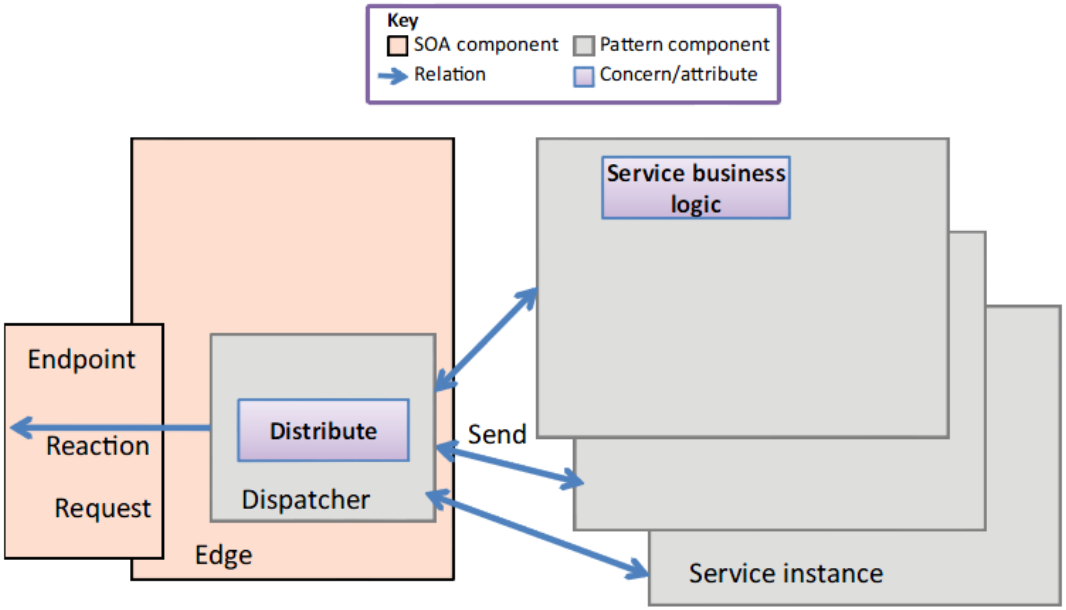


Figure 3.11 **In the Service Instance pattern a dispatcher (usually deployed on the edge) routes messages to one of the instances of the service business.**

Virtual Endpoint Pattern

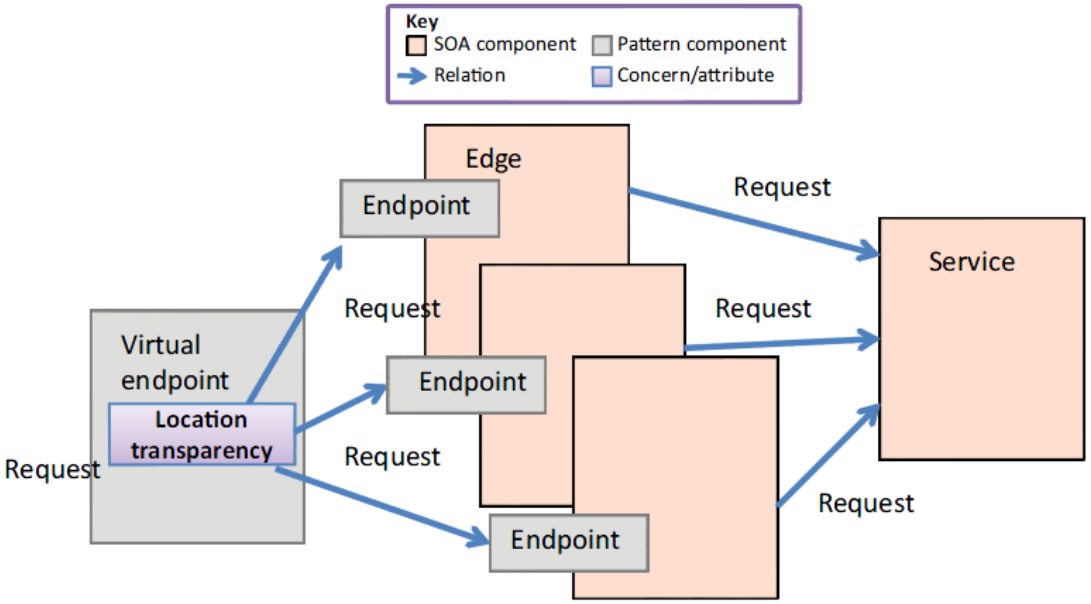


Figure 3.13 **The virtual endpoint exists as a known address, but the requests are actually handled by edge components that exist on other, internal, addresses.**

Assignment-2 Discussions

- How will you implement the Service Instance Pattern for the Social Network Services developed in Assignment-1?
- What will be the HA-Proxy Configuration?

Watchdog Pattern

How can you increase availability by identifying and resolving problems and failures that are service-specific?

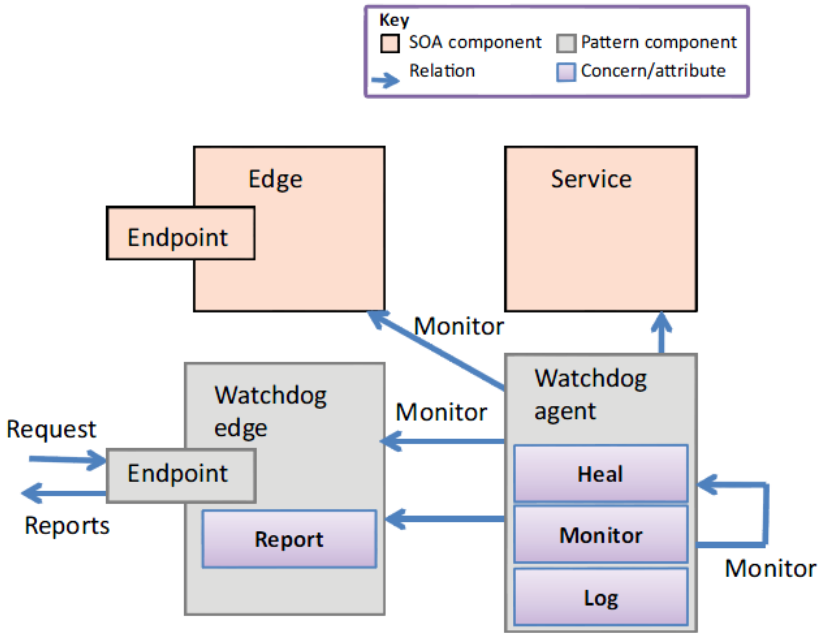


Figure 3.14 **With the Service Watchdog pattern, the watchdog edge component sends health reports and listens for requests. The watchdog agent component receives these reports and tries to heal itself before the problem gets worse.**

Watchdog Pattern

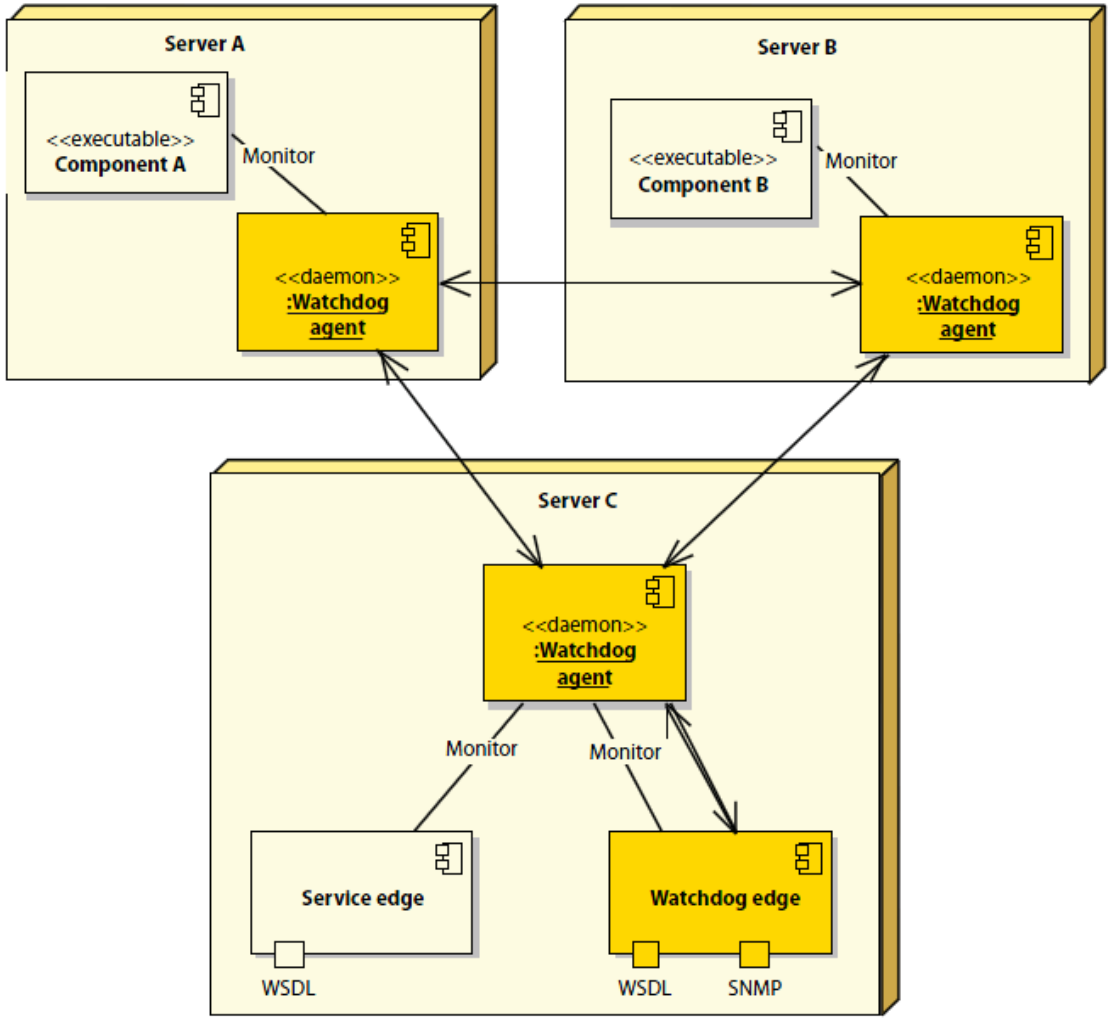


Figure 3.15 The daemon processes on the servers monitor the running components on each server. With the Service Watchdog pattern, the Watchdog edge component exposes the current state through a web-service interface, and as SNMP traps.

Assignment-2 Discussions

- How will you implement the Watchdog pattern for your service?
- How will you expose the Watchdog's Edge component?

SOA Design Patterns