Section 1: Week 2: Plan for Distributed Software Architecture

Nate Bachmeier

TIM-8120: Distributed Systems

September 22nd, 2019

North Central University

# Part I: Expand the Conceptual Diagram

Contoso operates a distributed supply chain of clothing manufacturing and retailer outlets. Within their systems, multiple autonomous agents exist and need to interact with the same shared resources. These requirements create the need to consider the operations and service management of the network. Without these capabilities, a single node failure could cascade and compromise the reliability of the topology. As the reliability and availability of the environment degrade, it causes a loss of productivity and consumer confidence. It is, therefore, advantageous to minimize these scenarios.

## Sharing Resources Between Agents

Autonomous agents include humans, machinery, mobile devices, office equipment, and web jobs (Martinez, Vazquez, Estada, Santillan, & Zavala, 2017). As these agents perform work, they persist modifications to centralized data services. These update operations can and will occur in parallel, which leads to the need for protocols to handle these concurrent and potentially conflicting operations (Celar, Mudnic, & Seremet, 2016).

Consider the scenario where three employees attempt to access a garment within the clothing catalog. (1) The first wants to get a scarf, the second updates its price, and the third wants to discontinue (delete) the item. Each of these requests arrives at the application load balancer and becomes routed to healthy front-end web service instances. (2) Within the front-end instance, the operation passes through a collection of middleware layers; these are responsible for decoupling the business logic from the transport configuration. (3) Next, the message moves across another service load balancer into a ring of stateless data storage providers. (4) The stateless data store provider instance interacts with the stateful data store provider. (5) In the case of the update or delete, the caller might provide a revision number or similar entity tag so that the caller can be alerted to changes since they retrieved the remote object. If the revision number has not changed, then the modification can be safely committed. (6) To ensure that the commit is highly durable, the stateful data store would need to wait for a sufficient number of nodes within the cluster to acknowledge the store operation. (7) Only after confirming the successful handoff can the system return to the caller, the process was successful (Baudisch & Schneider, 2013).

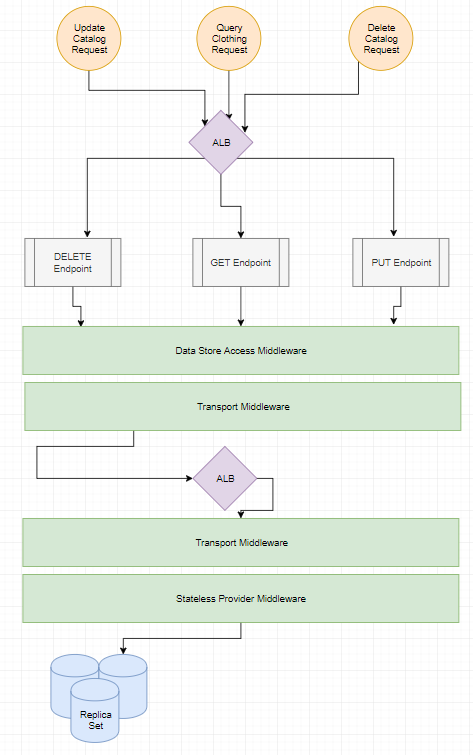


Figure 1: Single System View

## Reducing the Blast Radius

The Clothing Catalog reference architecture applies to many scenarios. First, brokering the network traffic across an Application Load Balancer (ALB) spreads the traffic evenly across a target group. These target groups continuously join and evict service instances by monitoring their performance metrics. Second, middleware within the client is responsible for managing the retry policy as operations fail with transient issues. For example, a service host's physical power state cycle, resulting in arbitrary failures (Celar, Mudnic, & Seremet, 2016). The transport middleware could then detect this state and reissue the request for the ALB to route to an alternative instance. Third, to ensure that the modification is not lost, the caller needs to acknowledge the callees they completed the work (Venkatesan & Sridhar, 2017). Suppose the completion cannot be acknowledged or is erroneous. In that case, that failure needs to be propagated to the client for compensation.

## Management and Operational Services

External management systems need to exist to provide operational capabilities for the broader topology. These capabilities include directory and domain services (e.g., authentication and name resolution); operational monitoring (e.g., system health); provisioning services (e.g., machine replacement); and desired state configuration (e.g., policy management). These systems are self-contained with dedicated target groups and private data stores to ensure high availability.

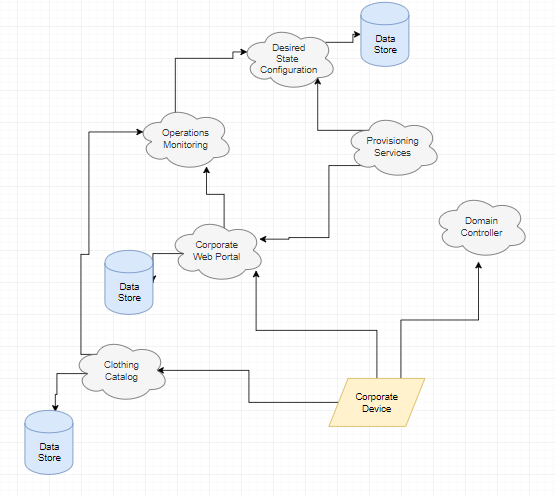


Figure 2: Management View

## Modifications to the Headquarters Diagrams

The headquarters system diagram was missing routers and switches and was updated. There were additional considerations to the global network topology of Contoso and those requirements.

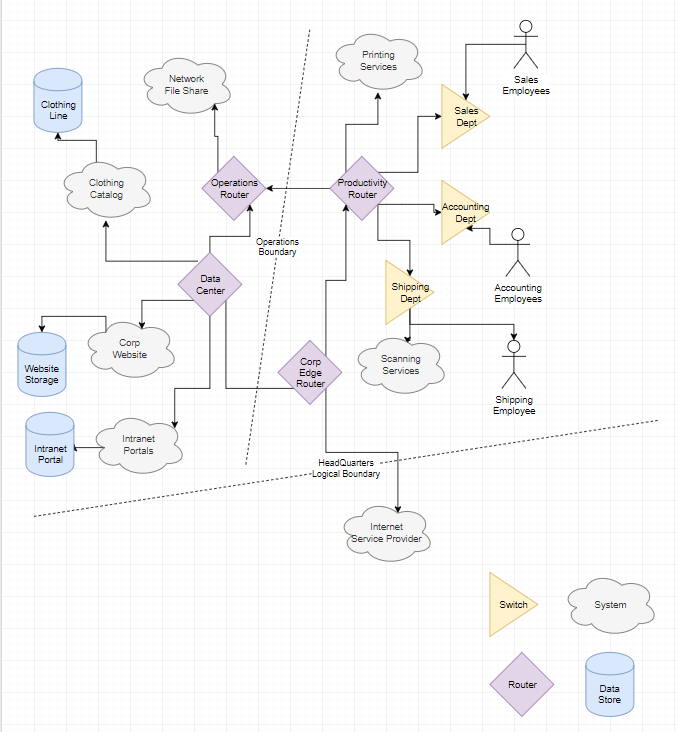


Figure 3 Headquarters Topology

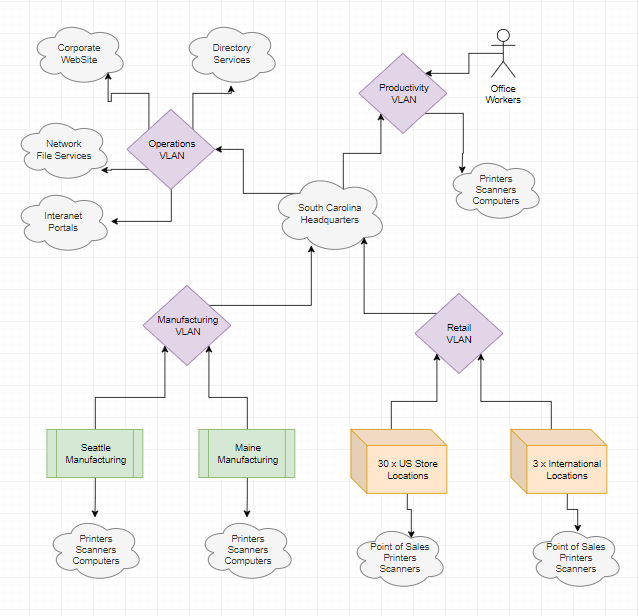


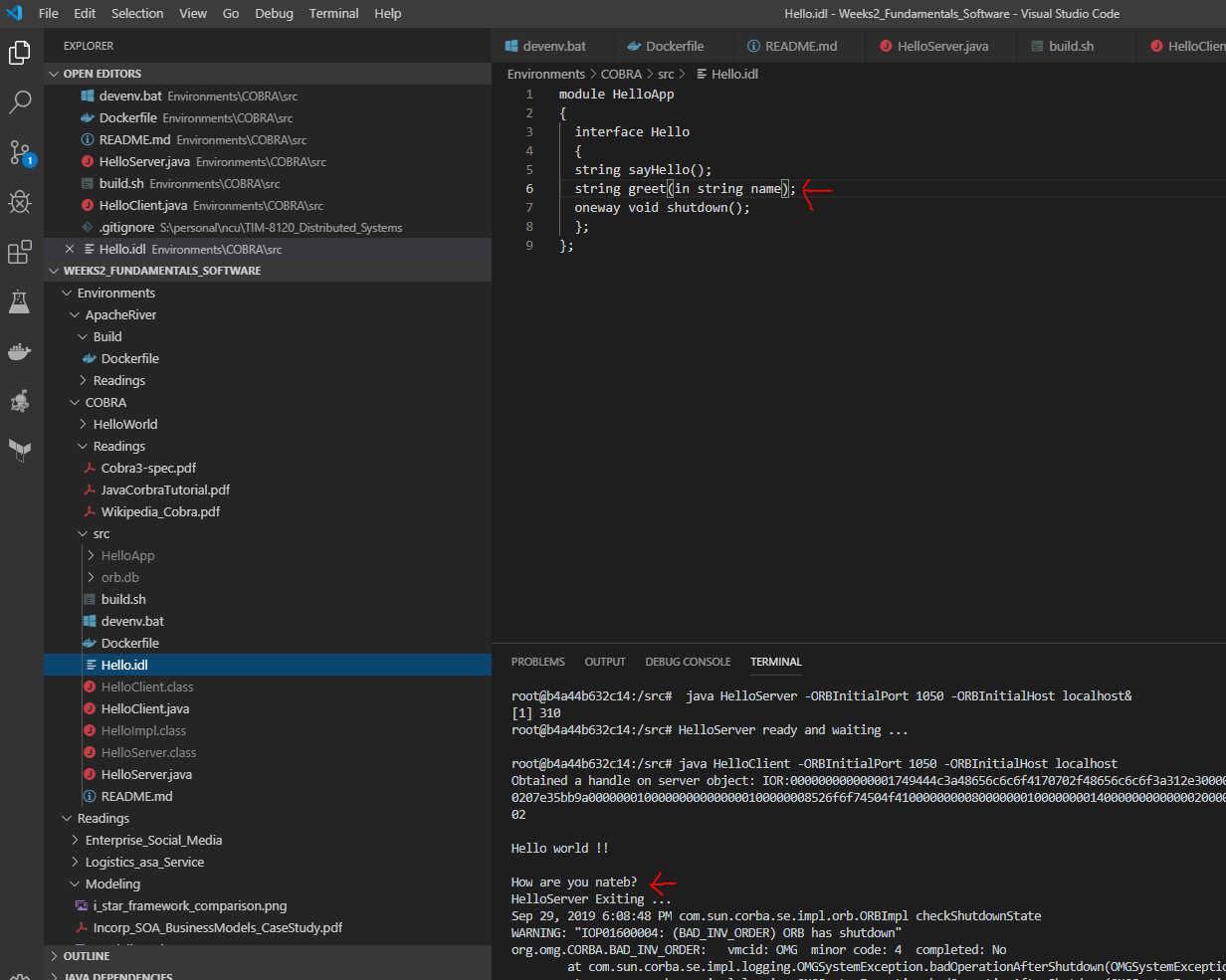
Figure 4 Global Topology

# Part II: COBRA and Apache River

The Common Object Request Broker Architecture (COBRA) and the Apache River Project are two frameworks for creating distributed architectures that leverage remote object access patterns. They are similar to Microsoft's Distributed Component Object Model (DCOM) and Dotnet Remoting technologies. Each of these technologies follows the same pattern of hosting class definitions within a server process. Authenticated callers can then create instances of those classes within the remote process space and invoke methods.

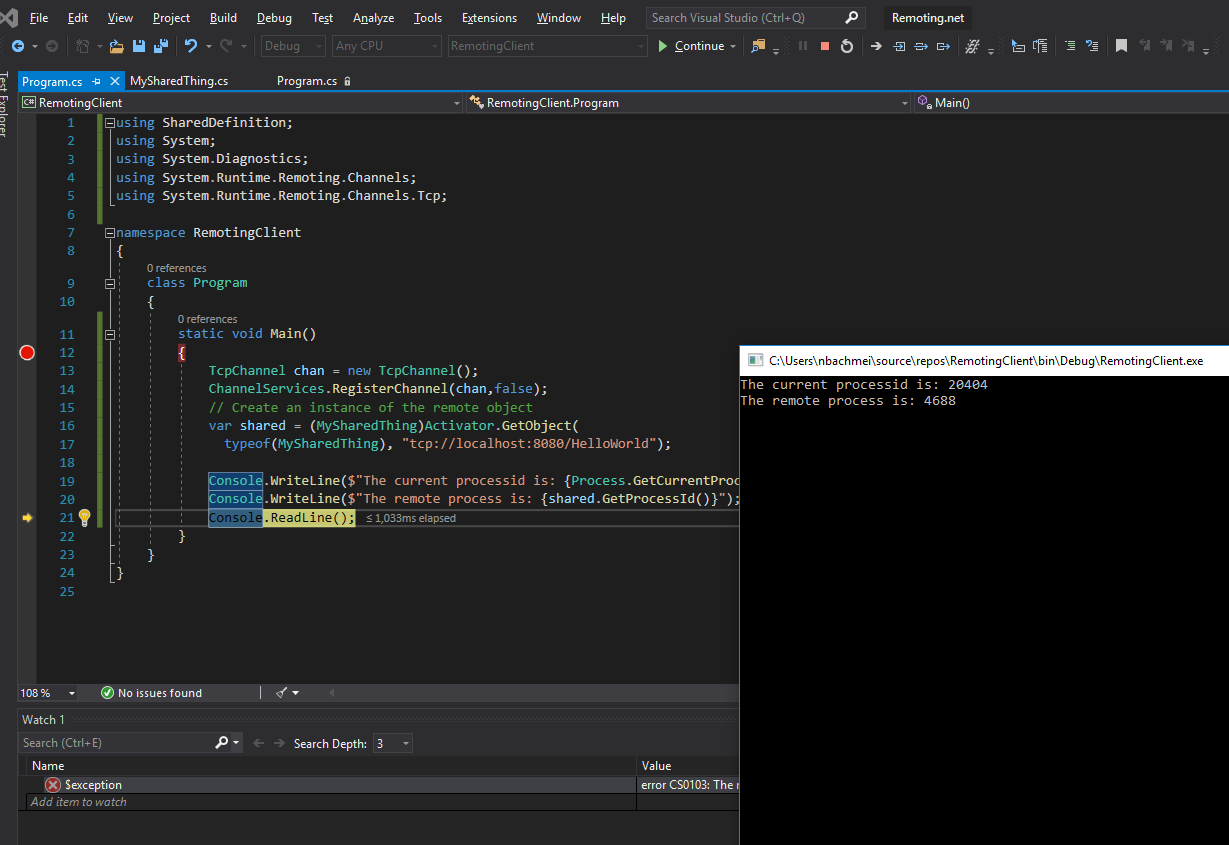
## COBRA: Hello World

Creating a COBRA compatible application is relatively painless, as the author needs to define a service contract and then run the 'idlj compiler (Oracle, 2019).' This compiler emits the boilerplate code required for linking the client and server.



## Remoting.net Hello World

After too much time trying to set up the HelloWorld in Apache River, it became Hello World in Dotnet Remoting. This platform follows a very similar design pattern but without the additional baggage.



## Risks to Market Share

The various products have different strengths and weaknesses, such as River and Remoting only support homogeneous platforms, while COBRA and DCOM leverage an external service contract for heterogeneous deployments. Each of these products is highly suspectable for memory leaks as remote systems manage the server objects' lifecycle. Additional challenges arise over extended periods, as either party needs to apply rejuvenation strategies (Yang, Min, Yang, & Li, 2013).

Many applications avoid these challenges by using message passing or Service Oriented Architecture (SOA). One of the critical advantages of these patterns is that it becomes much easier to write stateless code. Another is that communication can become less chatty as the caller states their entire request upfront instead of piecemealing it. There are specific scenarios, such as monitoring job status, that a long-lived object can be preferable. However, in these scenarios, more modern protocols such as Web Sockets are a more efficient mechanism. Despite the increase in market share to these new architectures, there are decades worth of legacy code already written. While SOA has simplified the transition process to modern designs, the old patterns will continue to exist for the foreseeable future (Xu, Nageshwaraniyer, & Son, 2016).

# References

Baudisch & Schneider. (2013). Evaluation of Speculation in Out-of-Order Execution.

Celar, Mudnic, & Seremet. (2016). STATE-OF-THE-ART OF MESSAGING FOR DISTRIBUTED COMPUTING SYSTEMS.

Martinez, A., Vazquez, B., Estada, H., Santillan, L., & Zavala, C. (2017). Incorporating technology in service-oriented i\* business models: a case study.

Oracle. (2019). *Java IDL: The "Hello World" Example*. Retrieved from Java TechNotes: https://docs.oracle.com/javase/8/docs/technotes/guides/idl/jidlExample.html

Venkatesan & Sridhar. (2017). A novel programming framework for architecting next-generation enterprise-scale information systems.

Xu, D., Nageshwaraniyer, S., & Son, Y. (2016). A service-oriented simulation integration platform for hierarchical manufacturing planning and control.

Yang, Min, Yang, & Li. (2013). Software rejuvenation in cluster computing systems with dependency between nodes.