Hello!

I think it would be best for me to set the scene for a development project, then discuss the architectural designs that I would choose to fulfill the architectural software requirements. For many of my individual projects at CTU, I've worked with an e-commerce bookstore, so I think that will be the domain I'll focus on here. When considering e-commerce, there is a wide range of domain responsibilities required and potential third-party implementations. Some examples of domain responsibilities would include inventory management, payment processing, order fulfillment, shopping cart management, advertisement servicing (depending on the size of your application), product shipping, etc.

With this in mind, I think it would be best to pursue a microservices infrastructure. Using microservices, you can loosely couple these relationships between the different domain responsibilities of your application. Additionally, this loose coupling will allow your development teams to more easily grasp smaller, more focused sections of domain functionality at a time. If you have a large-scale development effort, the modularized nature of microservices would allow for asynchronous development of each service without teams bumping elbows in the same code-base. Implementation of this might look like a containerized set of services using Kubernetes.

Paired with the microservices architecture, I would implement event driven architecture. Considering how an e-commerce system operates, much of it is based off events. When a user puts an item in their shopping cart, that item might need to be reserved temporarily from inventory. When a user purchases the items from their shopping cart, a payment processor will need to handle the user's payment, then inventory will need to be decreased, the order will need to be shipped, certain correspondence may need to be made, and a restocking event might need to occur based on the amount of items left in inventory. To fulfill this need for the system, I might implement a publisher/subscriber event bus of some sort like Apache Kafka, RabbitMQ, or Google's Pub/Sub.

To fulfill the requirements of the discussion board, I'll focus on event driven architecture. I've already discussed how it fulfills the needs of this real-world scenario, but it's important to understand the qualities of event driven architecture and consider the strengths and weaknesses of this architectural solution as well.

Event Driven Architecture (EDA) can be qualified by the following attributes:

1. Events. A state change occurs per user interaction that results in a chain of subsequent domain required actions. An example is the user purchasing the items in their cart.

2. Service/Event Handler(s). A service or application reacts to the user generated event, resulting in data processing, data storage, and/or subsequent event generation. These services in this real-world scenario would be the microservices that handle the events after user interaction, like the payment processing service or inventory management service.
3. Event Loop (Event Bus). An event loop facilitates interactions between events and services. This would be our event bus, like Google Pub/Sub.
4. Event Flow Layers. There are three types of event layers, the event producer, event consumer, and event router. These are the actors within our event driven system.

Using EDA is only useful if your system calls for it. Some systems may not require that level of intricacy in their processes. The major weakness of EDA is complexity. Organizing events between separate services (whether they be services in code or services in infrastructure) can become difficult to maintain if they aren't carefully designed. This complexity also leads to difficulty in debugging issues and monitoring. When including the additional complexity of microservices, this can result in issues with eventual consistency between microservice owned databases, network latency issues, event ordering problems, etc.

In return for the added complexity, EDA provides a lot of flexibility, scalability, and loose coupling. This is also further extended by the microservices architecture. Also, EDA typically is designed using asynchronous, non-blocking processing. Depending on your infrastructure, this could be a large cost-saving measure. So, if your goal is to build a large-scale system that requires complex logic and user driven event handling, EDA + Microservices might be your answer.

Originally, I'd included domain driven design in this response, but my answer got too long. If you're interested in discussing that as well, I'd be happy to in the comments.

Overview of Event-Driven Architecture (EDA). (2021, August 10). GeeksforGeeks. https://www.geeksforgeeks.org/overview-of-event-driven-architecture-eda/