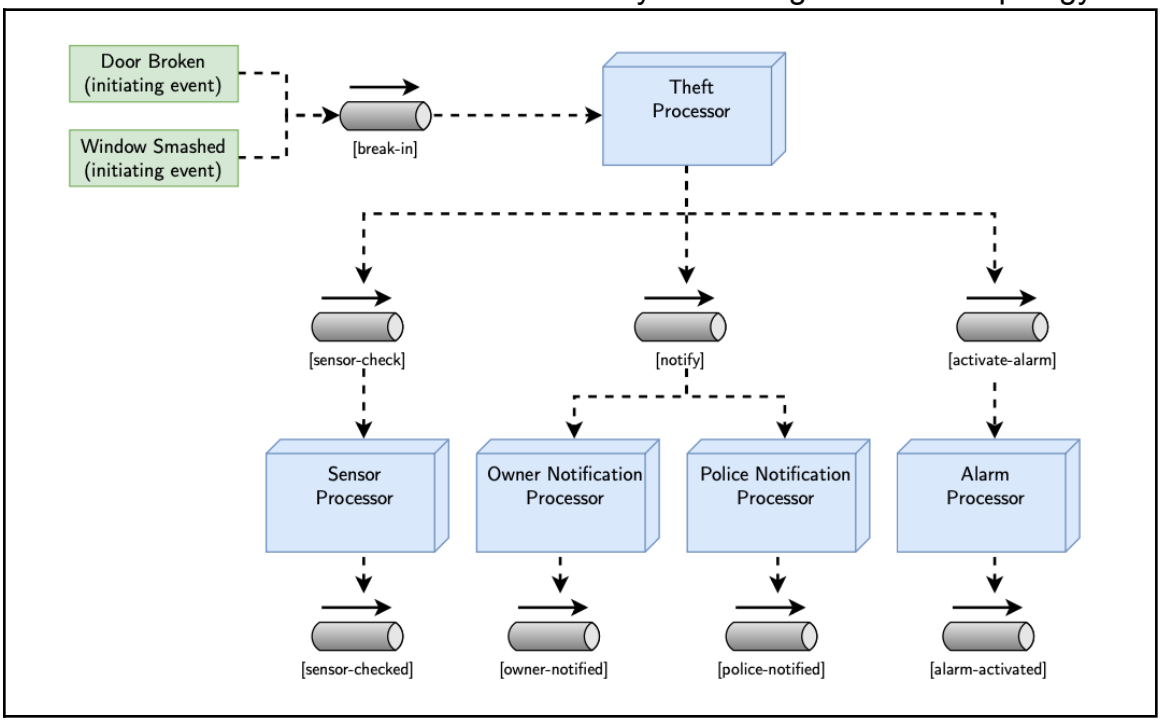
Lab 5: Distributed Architectural Styles

1. The Broker topology is best suited when you have a simple event stream that does not require event orchestration. The topology can follow either a queueing model, a lightweight message topics model, or a combination of both. The only core components involved are the broker and the processors. The broker component is federated and contains event channels used within the event flow. There is no event queue or mediator in broker topology; it depends upon the event processors to obtain each event, publish them, and process another event that announces the end of processing. Preserving the loosely-coupled nature of event-driven architecture, the tasks all remain asynchronous. When a task ends, a callback is triggered. This published message is not consumed by any other event, which may leave the application open for future updates.

   Suppose you are building software for a burglar alarm system for a small jewelry store called Diamonds and Stuff. Diamonds and Stuff has a general retail space and a back room. To protect the store, the owners of Diamonds and Stuff have placed sensors on the doors and windows. In the event of a robbery, the very first event is either the door being broken open or a window being smashed. Either of these events should set off the burglar alarm. The software system should have a simple one-to-one relationship between the check of the sensor and the specific alarm that is activated. When an alarm is activated both the Diamonds and Stuff owners and the police need to be notified.

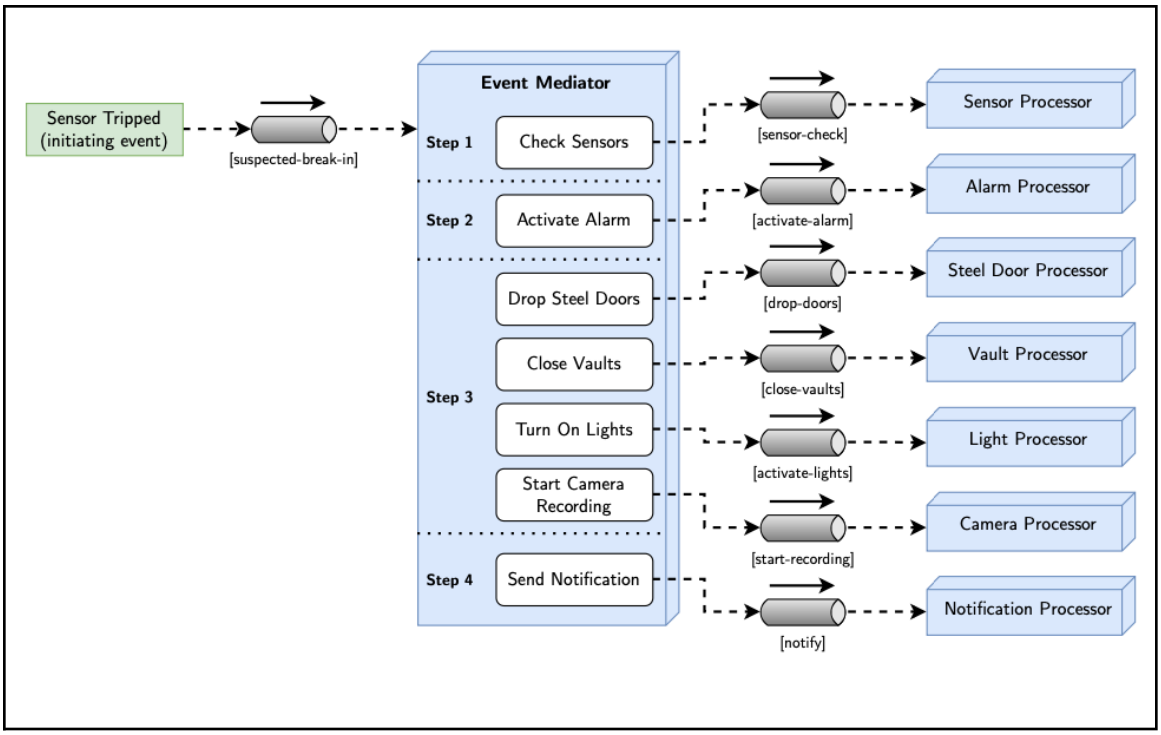
   Sketch an event-driven architecture for this system using the Broker topology.



2. The Mediator topology is best suited for complex situations in which multiple steps are required to process an event, thus requiring event processing coordination or orchestration. An event sends a message through a pre-processing event channel to a single event queue, which will transport the event to the mediator—thus initiating an event stream that routes events to relevant event processors. Orchestration is performed by the mediator, sending asynchronous events to various event channels that will perform each individual step of the process. The event processors, meanwhile, are listening to the channels and receive the event from the mediator. They then perform the business logic required to process it. The mediator transforms the initial event it receives by generating a processing event that can be received by the processors. This is not to say that the mediator itself performs any business logic. It simply provides processing instructions to the processor. All business logic processing for every single event is performed by the event processor containing the required application logic associated with that specific task. No other processors will be involved.

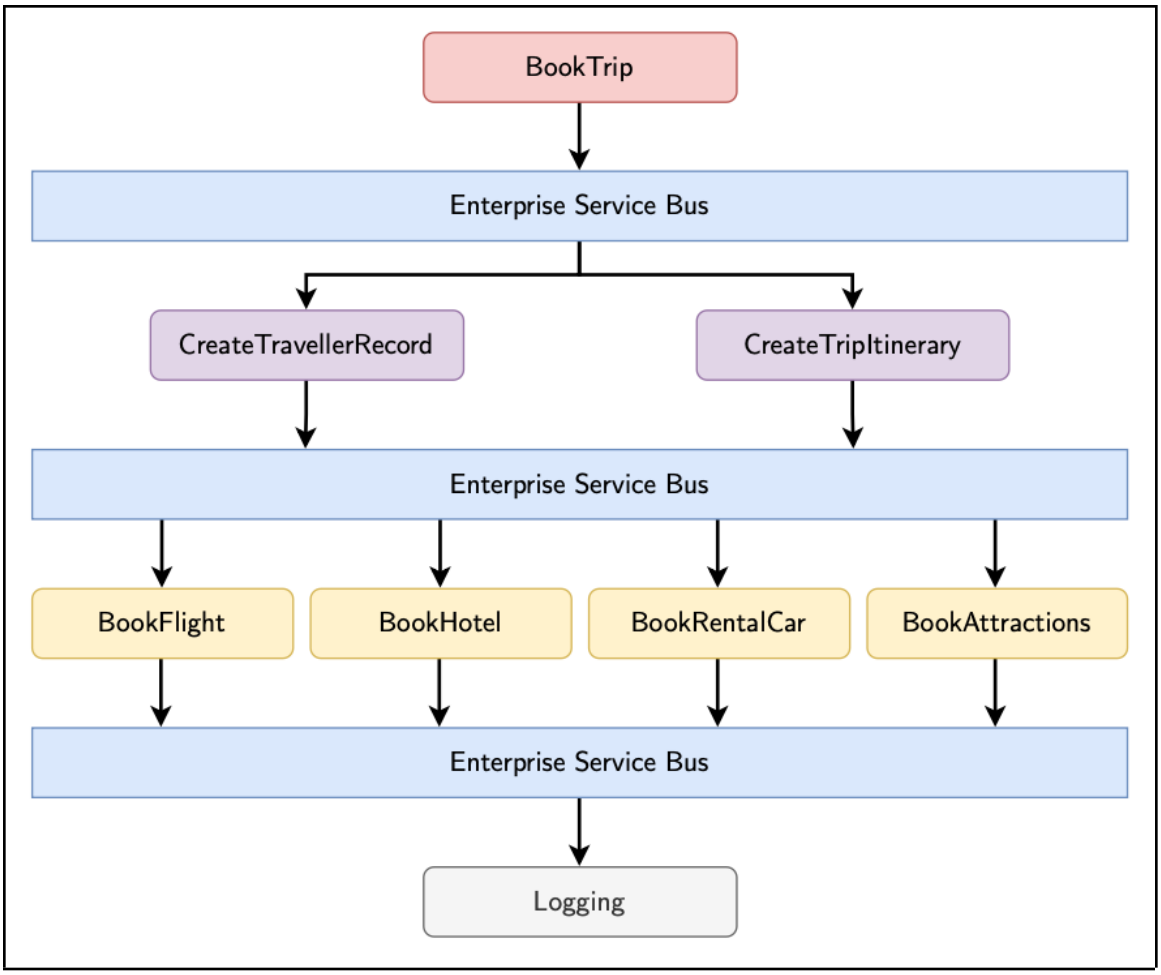
   Suppose you are building software for a burglar alarm system for a high-security facility (like Fort Knox). Such a facility will have many doors, windows, movement sensors, cameras, and other sensors. Only one sensor of them must be tripped to create the event of a suspected break-in. In such a facility activating an alarm is only one part of the response (unlike the jewelry store scenario from Problem 1). In the event of a suspected break-in, the sensors must be checked, then the alarms must be activated, and a lockdown is commenced where large steel doors throughout the complex drop into place, open vaults automatically close, all the lights go on, and all cameras begin recording. Several people are notified directly and alerted.

   Sketch an event-driven architecture for this system using the Mediator topology.

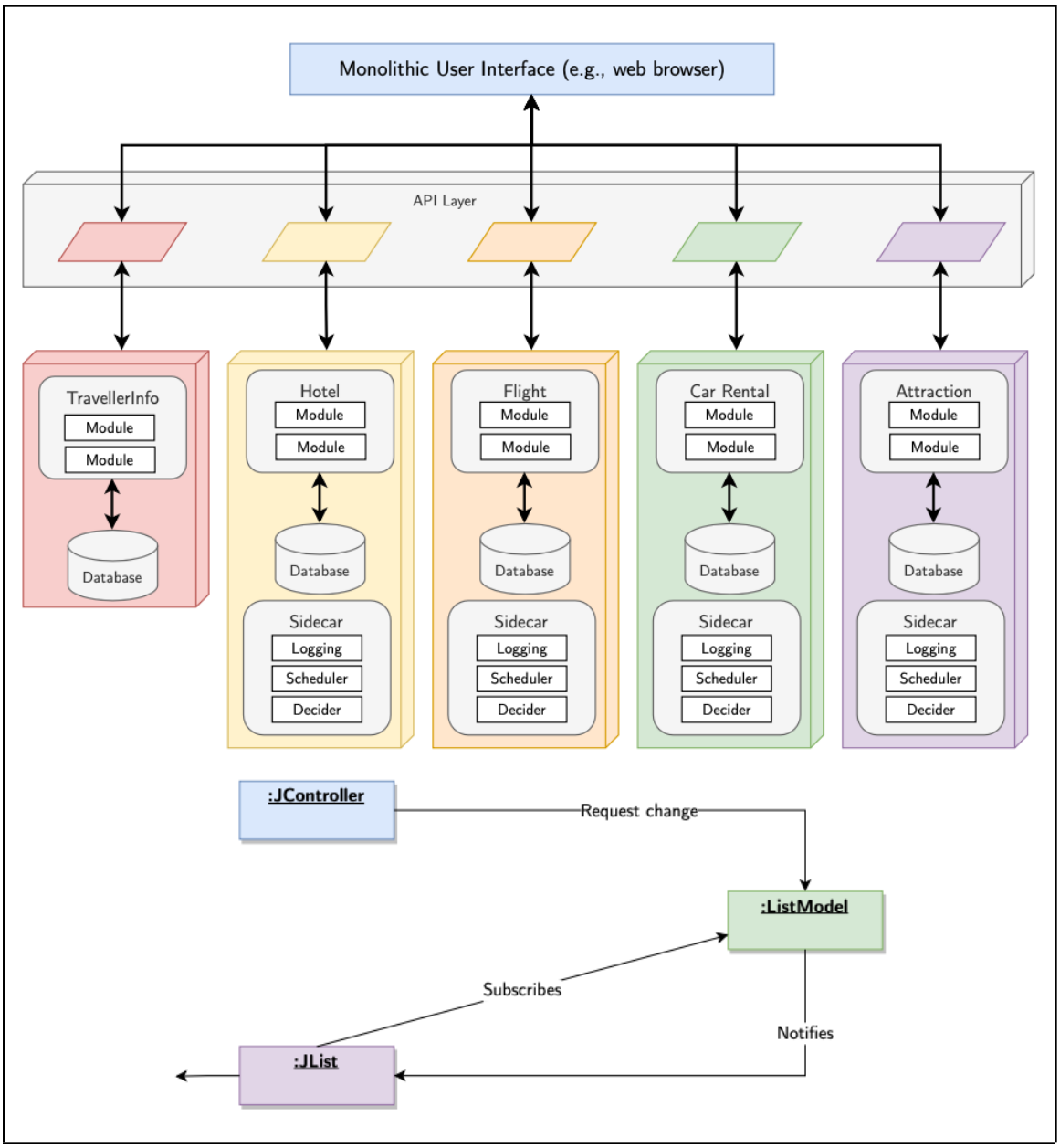


3. SimpleTravel is an online travel agency. The existing SimpleTravel system consists of three web services: airline reservation, car rental, and hotel reservation. The existing system uses a service-oriented architecture (SOA) to manage the built-in business rules and business strategy such as decision making and budget and schedule constraints to make optimal decisions enabling clients to choose from a combination of airlines, hotels, etc. SimpleTravel wants to expand to also provide an attraction service to help travelers book tickets to popular attractions during their trips.

   Consider the workflow for booking a trip that requires creating a traveler information record and a trip itinerary complete with flight hotel, rental car, and attractions using the expanded SimpleTravel system. Assume that there is some logging functionality to provide traceability into the decision making process for creating the trip itinerary. Sketch the message flow for this BookTrip workflow.



4. Suppose that instead of expanding their existing SOA-based system, SimpleTravel wanted to rebuild their system using the microservices architecture style to take advantage of the scalability and elasticity characteristics (among others) that a microservices architecture can provide. Sketch a microservices architecture for the expanded (i.e., with the attraction service capabilities) SimpleTravel system.


Monolithic User Interface (e.g., web browser)
API Layer
TravellerInfo
Module
Module
Database
Hotel
Module
Module
Database
Sidecar
Logging
Scheduler
Decider
Flight
Module
Module
Database
Sidecar
Logging
Scheduler
Decider
Car Rental
Module
Module
Database
Sidecar
Logging
Scheduler
Decider
Attraction
Module
Module
Database
Sidecar
Logging
Scheduler
Decider
:JController
Request change
:ListModel
Subscribes
Notifies
:JList