Robert Stockwell

West Texas A&M University

CIDM 6330 Spring 2022

May 1, 2022

# Overview

Review of concepts to continue development of the architecture development process. Program events are an area that needs to be considered outside of the domain model. If events are addressed here, they can cause very messy code and start to break down the architecture that has been created. According to Percival and Gregory, "it is the goop around the edges that makes a mess of the codebase" (Gregory, 2020).  In the case of Architecture Patterns with Python, the messaging outside the system is addressed by looking at the notification of an out-of-stock SKU. Initially, it was an email. The business analyst proposes it, but what happens if the notice vehicle changes, say to an SMS notification. Doing this in the program presents challenges, such as whether the code base needs to change for each message and whether the notice is turned on and off easily?

Initial thoughts on locations for adding the notifications include the Domain model, which quickly becomes an issue. Events like this do not fall inside the business problem trying to be solved. Keeping the model up to date becomes cluttered with the unneeded process and makes maintenance difficult for future developers and business users. The single responsibility and interface segregation principles also take a back seat in the design, violating two SOLID guidelines. The authors propose new models to handle these situations: the Domain Event and the Message Bus.

# Events

As *Architecture Patterns with Python* describes, Events are data classes called when specific criteria are met. This could include a product out of stock, a request to send a bookmark, or calculating a match total for a particular shooter. When these criteria are met in the domain model, it will raise an event. As events are raised, they are stored and sent to an outside handler for processing. This way, the models stay clean. The process also allows for explicitly ignoring an out-of-stock exception and removing it from the application.

# Message Bus

Using a simple publish-subscribe system, the outside handler, the Message Bus, maps events to handlers. This Python application is enabled using a dictionary data type and its key-value pairs. These events can be called from any piece of the architecture, but which one should be decided upon and consistency should be observed. The authors explore the options at the domain, service, and unit of work patterns raising the events. Using the Service layer does allow for the direct calling of the “add()” and “get()” functions, foregoing messy “\_add()” and “\_get()” calls in the Unit of Work layer.

As the message bus continues to be populated with the event calls, we see the repository items start to be replaced and can be moved into the handlers section of the application. This is demonstrated in the following diagram.

Diagram

Description automatically generated

# Commands

Commands are also a type of message contained in the application. Commands are usually dumb data structures representing instructions to be carried out between different pieces of a software system.

The difference between events and commands is subtle and essential. An Event captures the facts about the system, the on-hand inventory of an SKU, for instance. This is then broadcast out to the system to that a listener can grab on to the information and put it to use. Events are usually named with past-tense verb phrases.

As stated earlier, the command is a targeted indication of something to do. When the commands are sent, an expected result is looked for; if the command is not executed correctly, it should fail nosily with an exception raised and sent back to the command. Commands are usually named with “imperative mood verb phrases” (Gregory, 2020) helping indicate an action.

# References

Gregory, H. J. (2020). *Architecture Patterns with Python.* Sebastopol: O'Reilly Media, Inc.