3

A Functional Architecture

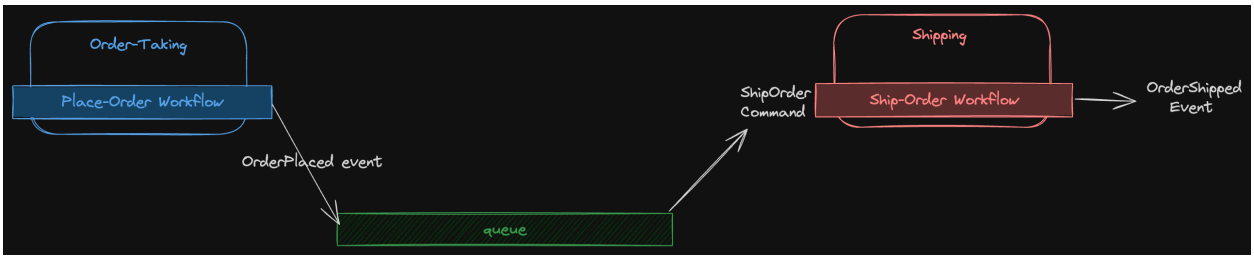
Translating domain understanding into a functional software architecture involves creating a rough plan for implementation, even before fully understanding the system. This often includes creating a crude prototype for early feedback. The architecture consists of four levels: system context, containers, components, and classes (or modules in a functional architecture).

Relationship between “bounded contexts” and “software components”

- A "bounded context" in software architecture is an autonomous subsystem with a well-defined boundary.
- Bounded contexts can be implemented as separate modules, distinct components, or standalone deployable containers.
- The exact implementation approach isn't critical at the initial stage, as long as the bounded contexts remain decoupled and autonomous.
- Correctly defining the boundaries is essential, but these boundaries may evolve as more is understood about the domain.

- Monsieur Scott thinks it’s best to start with a monolithic structure and refactor into decoupled containers as necessary.
- A premature jump to a microservice architecture should be avoided unless the benefits clearly outweigh the drawbacks.

Communicating between bounded contexts

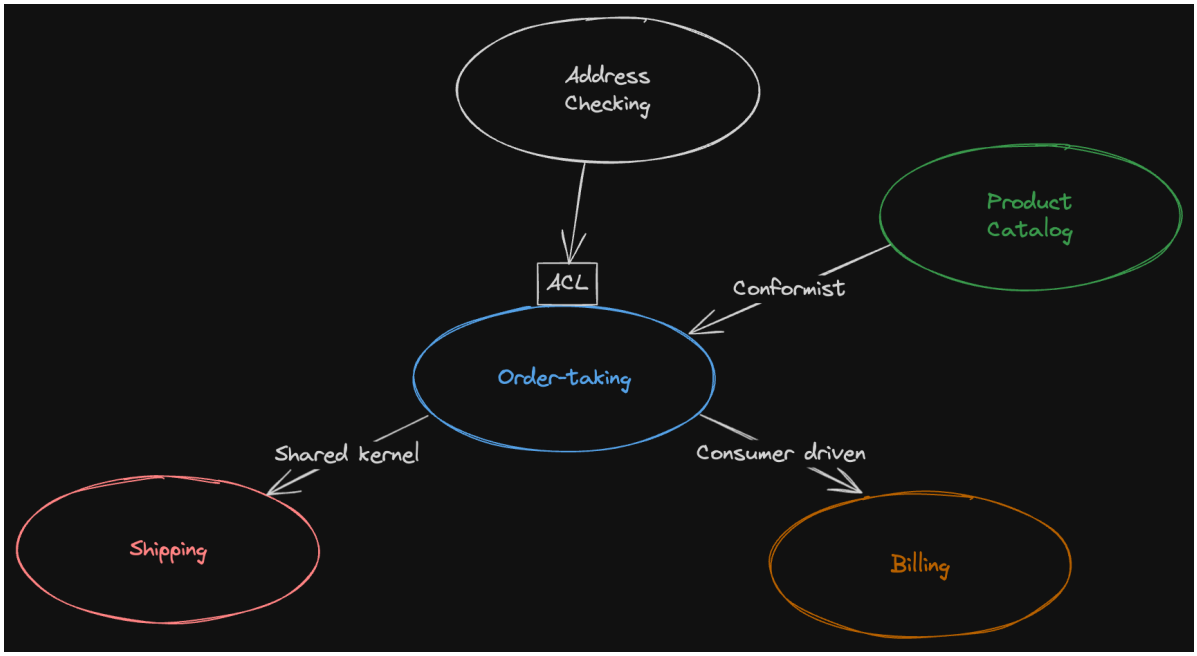


- Bounded contexts in a functional architecture communicate through events.
- This communication maintains a decoupled design for autonomy.
- An event, like 'OrderPlaced', triggers a command, such as 'ShipOrder', in another context.
- The method of transmitting events depends on the chosen architecture.
- Events often contain necessary data for downstream components, which is transferred as Data Transfer Objects (DTOs).
- The boundaries of a context act as 'trust boundaries'.
- Trust boundaries validate input and ensure that private information doesn't leak out.

Contracts between bounded contexts

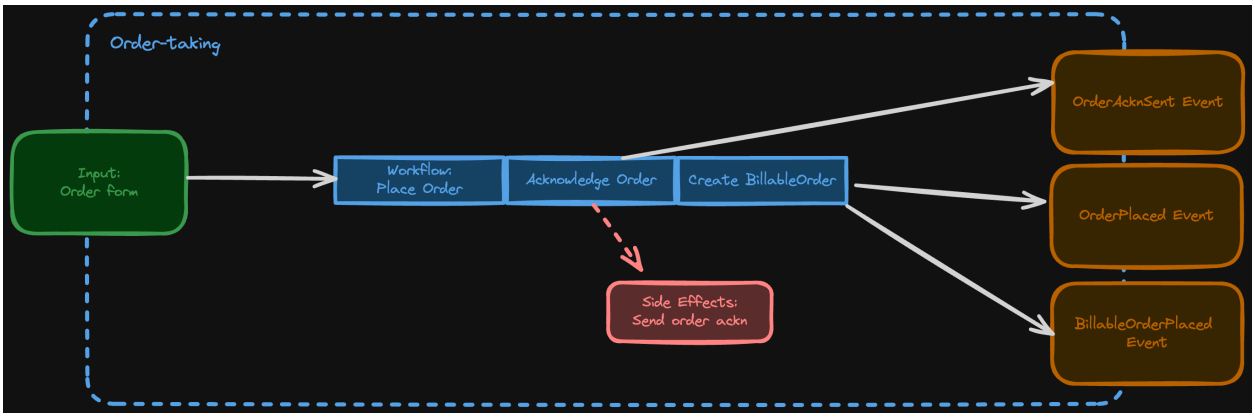
- In Domain Driven Design (DDD), three main relationships exist between bounded contexts:
  - Shared Kernel: Two contexts share a common design, and any changes must be mutually agreed upon.
  - Customer/Supplier or Consumer-Driven Contract: The downstream context outlines the contract the upstream context must fulfil.

    - Conformist: The downstream context adapts its model to match the contract provided by the upstream context.
- Additionally, an Anti-Corruption Layer (ACL) is used when communicating with an external system to prevent the internal model from being influenced by the external model. It's not a context relationship but a protective mechanism.



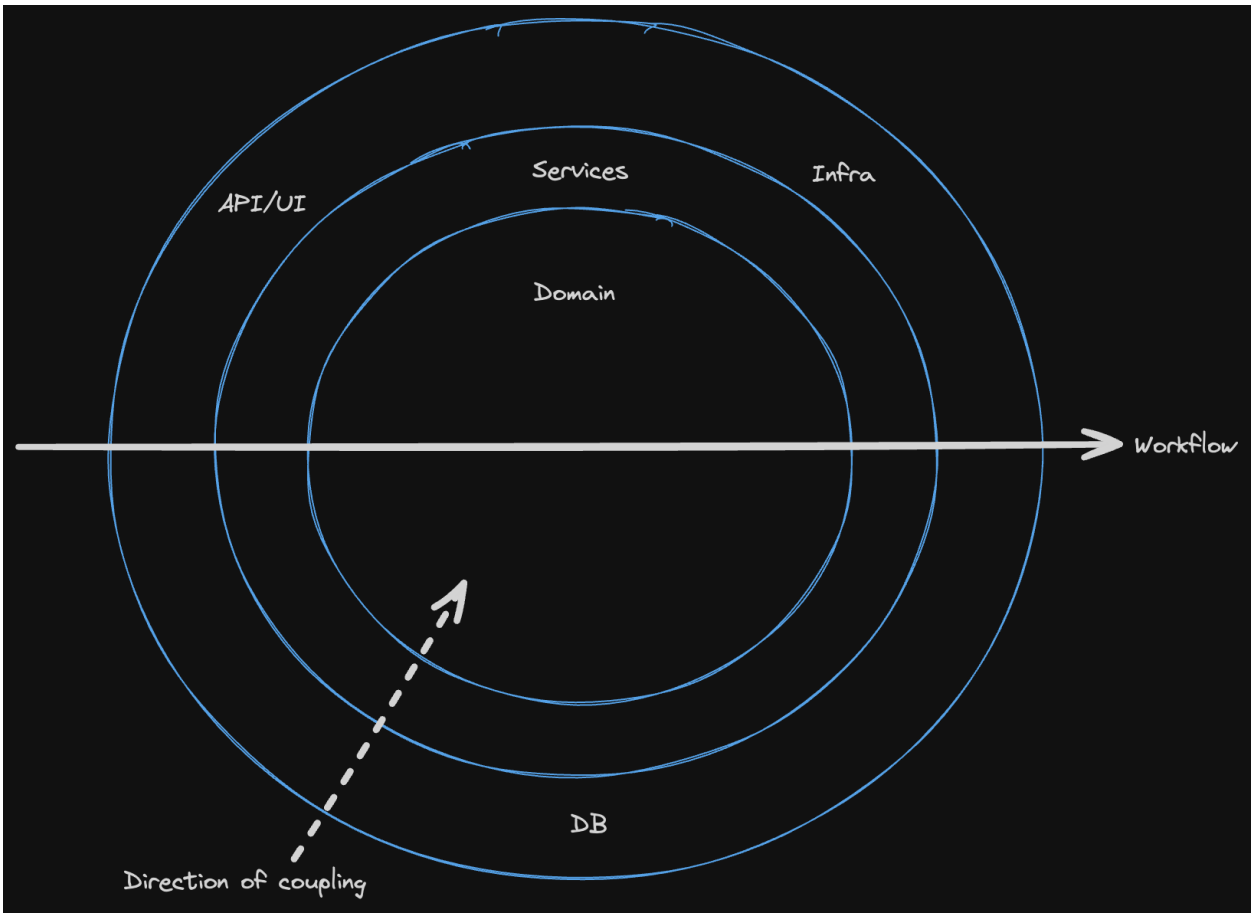
Workflows within a bounded context

- Business workflows in a functional architecture are treated as single functions, initiated by command objects and generating event objects.
- Each workflow is contained within a single bounded context and does not implement scenarios end-to-end through multiple contexts.
- The input to a workflow is always the data associated with a command, and the output is a set of events meant to communicate to other contexts.
- A workflow function does not publish Domain Events; it simply returns them.



Code structure within a bounded context

- Traditional layered approach to code structure within a bounded context breaks the design principle of "code that changes together belongs together."
- A suggested alternative is a vertical slice approach, where each workflow contains all necessary code.
- The ideal structure is the "Onion Architecture," where the domain code is central and all dependencies point inward → ensures predictability and easy reasoning
- Functions should avoid side effects, including I/O, which should be kept at the edges of the architecture.
- This separation of concerns aligns with the concept of persistence ignorance, keeping the core domain model focused on business logic



Summary

- Domain Objects: objects designed for use only within the boundaries of a context, as opposed to a Data Transfer Object.
- Data Transfer Objects (DTOs): objects designed to be serialized and shared between contexts.
- Relationships: Shared Kernel, Customer/Supplier, and Conformist are different kinds of relationships between bounded contexts.
- Anti-Corruption Layer (ACL): a component that translates concepts from one domain to another in order to reduce coupling and allow domains to evolve independently.
- Persistence Ignorance: This concept means that the domain model should be based only on the concepts in the domain itself and should not contain any awareness of databases or other persistence mechanisms.

What’s next

- Implementing workflows using the F# type system and understanding 'types' in functional programming.