DDD and MVC architecture

What is DDD Architecture?

Domain-Driven Design (DDD) is an architectural approach and methodology for software development that emphasizes focusing on the core domain and its logic. It helps in creating software systems that closely reflect the real-world problems they are designed to solve.

Key principles of DDD:

1. **Ubiquitous Language**: A common language shared by technical and non-technical stakeholders.
2. **Bounded Contexts**: Clear boundaries for different parts of the system to maintain separation of concerns.
3. **Entities and Value Objects**: Modeling real-world concepts with entities (objects with a distinct identity) and value objects (immutable and defined by their attributes).
4. **Aggregates**: Clusters of domain objects treated as a single unit for consistency.
5. **Repositories**: Abstract data stores to handle persistence.
6. **Services**: Operations that don't naturally belong to entities or value objects.

DDD aligns software design with business needs and focuses heavily on the **domain layer**, which represents the core business logic.

MVC Architecture

Model-View-Controller (MVC) is a design pattern for organizing the structure of software systems, commonly used in web and application development. It separates the application into three interconnected components:

1. **Model**: Represents the data and business logic of the application.
2. **View**: Handles the presentation layer, rendering UI components.
3. **Controller**: Mediates user input, processes it, and updates the Model or View.

Comparison Between DDD and MVC

Feature	**DDD (Domain-Driven Design)**	**MVC (Model-View-Controller)**
Focus	Focused on the **domain layer** and business logic.	Focused on separating UI, logic, and data.
Complexity	Designed for **complex systems** with rich domains.	Suitable for simpler systems or straightforward UI-driven applications.
Separation of Concerns	Emphasizes separating **domain logic** via entities, value objects, and bounded contexts.	Emphasizes separating **UI logic** from business logic and data.
Business Logic Placement	Encapsulated in the **domain model**.	Typically placed in the **Model**.
Scalability	Scales well for large, evolving business domains.	Scales for smaller, simpler applications but can become unwieldy for complex systems.

Feature	DDD (Domain-Driven Design)	MVC (Model-View-Controller)
User Interaction	User interaction is not a central concern; focuses on domain integrity.	Explicitly manages user interaction and UI changes.
Repositories	Uses repositories as abstractions for persistence.	Often integrates data access directly into the Model.
Suitability	Best for enterprise-grade applications or systems with **rich, evolving business rules**.	Best for web applications with **clear separation between UI and logic**.
Flexibility	Highly flexible; requires significant design upfront.	Straightforward; often tightly coupled with frameworks (e.g., Rails, Django).

Use Case Examples

- **DDD**: A complex e-commerce platform where pricing, inventory, and order systems have rich business rules and domain logic.
- **MVC**: A blog or content management system where the focus is on rendering views and CRUD operations.

Integration

In practice, you can use **DDD principles** within the **Model** of an MVC framework. For example, the domain layer in DDD can act as the Model in MVC, while Views and Controllers manage presentation and user interaction separately.

Unique solution ID: #1159
Author: n/a
Last update: 2025-05-21 13:23