agap2IT

TRAINING CENTER

Domain Driven Design
Module 4

Architecture and Event-Driven Modeling

Domain Driven Design

Module Overview

Domain Events

Event Storming

Extreme Modeling

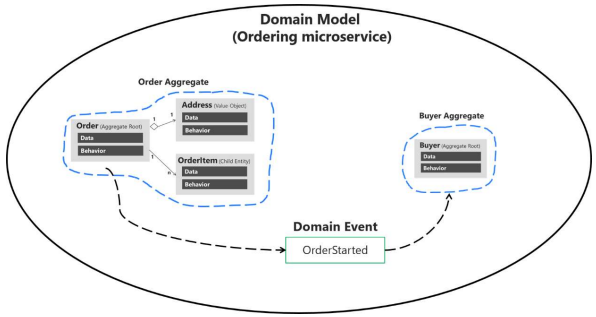
Domain Driven Design

Domain Events

Eric Evans
DDD Author

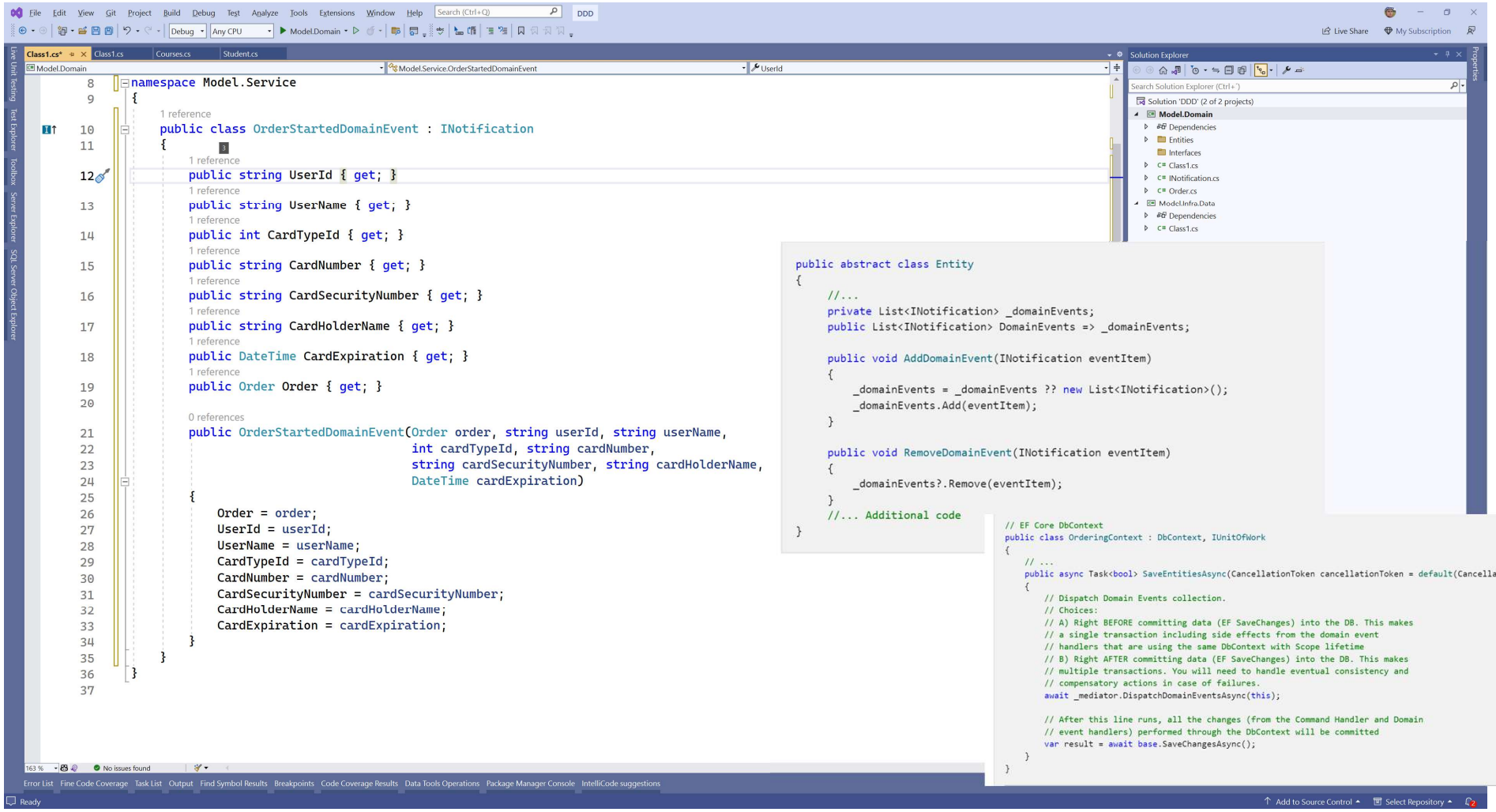
"An event is something that has happened in the past. A domain event is, something that happened in the domain that you want other parts of the same domain (in-process) to be aware of. The notified parts usually react somehow to the events.

For example, if you're just using Entity Framework and there must be a reaction to some event, you would probably code whatever you need close to what triggers the event. So, the rule gets coupled, implicitly, to the code, and you must investigate the code to, hopefully, realize the rule is implemented there."



Domain Driven Design

Domain Events



```
namespace Model.Service
{
    1 reference
    public class OrderStartedDomainEvent : INotification
    {
        1 reference
        public string UserId { get; }
        1 reference
        public string UserName { get; }
        1 reference
        public int CardTypeId { get; }
        1 reference
        public string CardNumber { get; }
        1 reference
        public string CardSecurityNumber { get; }
        1 reference
        public string CardHolderName { get; }
        1 reference
        public DateTime CardExpiration { get; }
        1 reference
        public Order Order { get; }

        0 references
        public OrderStartedDomainEvent(Order order, string userId, string userName,
                                       int cardTypeId, string cardNumber,
                                       string cardSecurityNumber, string cardHolderName,
                                       DateTime cardExpiration)
        {
            Order = order;
            UserId = userId;
            UserName = userName;
            CardTypeId = cardTypeId;
            CardNumber = cardNumber;
            CardSecurityNumber = cardSecurityNumber;
            CardHolderName = cardHolderName;
            CardExpiration = cardExpiration;
        }
    }
}
```

```
public abstract class Entity
{
    //...
    private List<INotification> _domainEvents;
    public List<INotification> DomainEvents => _domainEvents;

    public void AddDomainEvent(INotification eventItem)
    {
        _domainEvents = _domainEvents ?? new List<INotification>();
        _domainEvents.Add(eventItem);
    }

    public void RemoveDomainEvent(INotification eventItem)
    {
        _domainEvents?.Remove(eventItem);
    }
    //... Additional code
}
```

```
// EF Core DbContext
public class OrderingContext : DbContext, IUnitOfWork
{
    // ...
    public async Task<bool> SaveEntitiesAsync(CancellationToken cancellationToken = default(CancellationToken
    {
        // Dispatch Domain Events collection.
        // Choices:
        // A) Right BEFORE committing data (EF SaveChanges) into the DB. This makes
        // a single transaction including side effects from the domain event
        // handlers that are using the same DbContext with Scope lifetime
        // B) Right AFTER committing data (EF SaveChanges) into the DB. This makes
        // multiple transactions. You will need to handle eventual consistency and
        // compensatory actions in case of failures.
        await _mediator.DispatchDomainEventsAsync(this);

        // After this line runs, all the changes (from the Command Handler and Domain
        // event handlers) performed through the DbContext will be committed
        var result = await base.SaveChangesAsync();
    }
}
```

Event Storming

“Event storming is a rapid, lightweight, and underappreciated group modeling technique that is intense, fun, and useful for accelerating development teams. The brainchild of Alberto Brandolini, it's a synthesis of facilitated group learning practices from Gamestorming and the principles of domain-driven design (DDD). The technique isn't limited to software development. You can apply it to practically any technical or business domain, especially those that are large, complex, or both.

Event storming catalyzes and accelerates group learning, often achieving in a few hours or days what more traditional modeling techniques never do—a common understanding of the domain in which the software must operate.

To understand event storming you first need to understand two key terms. A domain event is anything that happens that is of interest to a domain expert. The domain expert is not interested in databases, web sockets, or design patterns, but in the business domain of the things that must happen. Domain events capture those facts in a way that doesn't specify a particular implementation.”

Domain Driven Design

Extreme Modeling

“Domain-Driven Design Extreme Modeling (D3X) is an advanced and intensive technique that takes Domain-Driven Design (DDD) principles to the extreme. It aims to push the boundaries of modeling and understanding complex domains by leveraging collaborative, immersive, and iterative practices. D3X is a proactive approach that seeks to uncover deep domain insights and drive the design of highly effective software solutions.

D3X focuses on intense modeling sessions that involve domain experts, development teams, and stakeholders. These sessions are typically time-boxed and structured to encourage active participation, creativity, and knowledge sharing. The goal is to create a rich and precise model of the domain, capturing its intricacies, rules, and behavior in a holistic manner.”

Key elements and practices of D3X

Immersive Collaboration: D3X fosters a high level of collaboration between domain experts and development teams. By bringing stakeholders together in a focused and interactive environment, D3X promotes real-time exchange of knowledge, ideas, and insights. This collaborative effort helps bridge the gap between domain understanding and software design, ensuring a shared vision of the solution

Visual Modeling Techniques: D3X utilizes visual modeling techniques to express the domain concepts, relationships, and behaviors. These techniques include event storming, process modeling, behavior-driven development (BDD), and user story mapping. Visual models help stakeholders and team members better understand the domain, align their mental models, and identify potential areas of improvement.

Rapid Prototyping and Feedback: D3X leverages rapid prototyping and feedback loops to validate and refine the model. By quickly translating the domain model into working prototypes or proofs of concept, the team can gather feedback from stakeholders and iterate on the design. This iterative feedback loop helps uncover potential issues, validate assumptions, and refine the model further

Model Exploration: D3X encourages exploration and experimentation through iterative modeling. It starts with a core domain model and progressively refines and expands it based on evolving insights and requirements. By continuously iterating and refining the model, D3X ensures that it accurately represents the complexities and nuances of the domain.

Ubiquitous Language Refinement: D3X places a strong emphasis on refining and evolving the ubiquitous language—the shared language that bridges the gap between domain experts and development teams. Through continuous collaboration, the team refines the language, ensuring that it accurately captures the nuances and vocabulary of the domain. The ubiquitous language becomes a powerful communication tool, enabling precise discussions and reducing misunderstandings.

Conclusion

“The D3X approach requires a high level of commitment, engagement, and openness from all participants. It challenges traditional modeling practices by promoting intense collaboration, visual modeling techniques, and iterative refinement. By embracing D3X, teams can gain a deep understanding of complex domains, leading to more accurate and effective software designs that closely align with the business goals.”

Who else needs some coffee ?

Domain Driven Design

Technical Training

Let’s do it