What is the most correct method of updating an Aggregate through an Aggregate Root?

Asked 2 years, 4 months ago Active 2 years, 4 months ago Viewed 331 times

0

Following the good practices of DDD, Aggregate and Aggregate Root. I have the following scenario:

- `User` (Aggregate Root)
- A collection of `UserEmail` (inside `User`)

Imagining that I have registered a User with 10 Emails, what would be the most correct and perfomable way of updating one of these emails?

Method 1

```
static void UpdateEmailForExistingUserMethod1()
{
    var userId = new Guid("f0cd6e3e-b95b-4dab-bb0b-7e6c6e1b0855");
    var emailId = new Guid("804aff75-8e48-4f53-b55d-8d3ca76a2df9");

    using(var repository = new UserRepository())
    {
        // I'm going to return the user with all their emails?
```

```
        // I will not have performance problems for bringing all emails from this user?
        var user = repository.GetUserById(userId);

        if (user == null)
        {
            Console.WriteLine("User not found");
            return;
        }

        // Updating Email in Aggregate Root
        user.UpdateEmail(emailId, "updated1@email.com");

        // Commit in repository
        if (repository.Commit() > 0)
        {
            Console.WriteLine("E-mail updated with method 1!");
        };
    }
}
```

Method 2:

```
static void UpdateEmailForExistingUserMethod2()
{
    var usuarioId = new Guid("f0cd6e3e-b95b-4dab-bb0b-7e6c6e1b0855");
    var emailId = new Guid("3b9c2f36-659e-41e8-a1c6-d879ab58352c");

    using(var usuarioRepository = new UserRepository())
    {
        if (!usuarioRepository.UserExists(usuarioId))
        {
            Console.WriteLine("User not found");
            return;
        }

        if (!usuarioRepository.EmailExists(emailId))
        {
            Console.WriteLine("E-mail not found");
            return;
        }

        // Grab only the email that I will update from the repository,
        // optimizing performance
        var usuarioEmail = usuarioRepository.GetEmailById(emailId);

        // Updates the e-mail through a method of the e-mail entity itself
        usuarioEmail.Update("updated2@email.com");
```

```
        // Commit in repository
        if (usuarioRepository.Commit() > 0)
        {
            Console.WriteLine("E-mail updated with method 2!");
        };
    }
}
```

domain-driven-design aggregate aggregateroot

asked Dec 12 '17 at 18:20

Tiago Santos

7 ● 5

1 Answer

Active Oldest Votes

0

If `User` is the root of the aggregate, then all modifications to the aggregate should be made by invoking a method on the root; so your "Method 1" is the correct pattern.

Specifically -- access to other entities within the aggregate is achieved by invoking a method on the root, and allowing the root to delegate the work to the internal entity if necessary.

The point is that the aggregate root(s) define the boundary between the domain model and the application.

Now, in some cases, this constraint doesn't seem to make much sense. When that happens, challenge your assumptions: are you sure that email is an entity? are you sure that entity needs to be transactionally consistent with the user entity?

For something like an email address, I would expect that the email address is going to be a value object, which can be added to a collection internal to user. So I wouldn't expect to see EmailId as an abstraction.

```
user.FixTypoInEmailAddress("updated@email.com", "updated1@email.com")
```

> Do not multiply entities beyond necessity.

answered Dec 12 '17 at 18:40

VoiceOfUnreason
30.1k ● 3 ● 25 ● 56