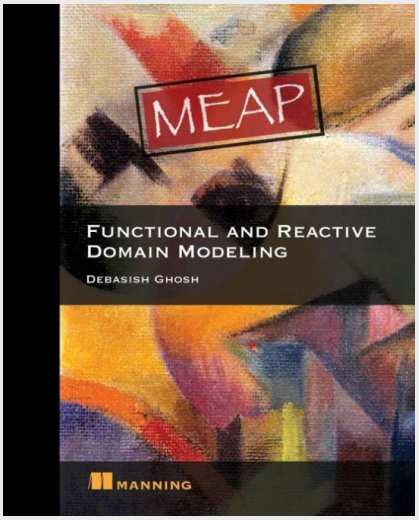

What is a domain model?

By Debasish Ghosh, Functional and Reactive Domain Modeling

The word domain means the area of interest in the business. The abstractions that you design, the behaviors that you implement, the UI interactions that you build, all reflect the business – together they constitute the model of the domain. In this article, we'll talk abaout the domain model and the challenges of its complexity.

This article is an excerpt from Functional and Reactive Domain Modeling by Debasish Ghosh.

When was it the last time you withdrew cash from an ATM? Or deposited cash into your bank account? Or used internet banking to check up if your monthly pay has been credited to your checking account? Or you asked for a portfolio statement from your bank? All the italicized terms relate to the business of personal banking. We call this the domain of personal banking. The word domain here means the area of interest in the business. When you are developing a system to automate banking activities, you are actually modeling the business of personal banking. The abstractions that you design, the behaviors that you implement, the UI interactions that you build, all reflect the business of personal banking – together they constitute the model of the domain.

More formally, a domain model is a blueprint of the relationships between the various entities of the problem domain and sketches out other important details, such as:

- Objects that belong to the domain. For the banking domain, you have objects like bank, account, transactions etc.
- Behaviors that those objects demonstrate in interacting amongst themselves. In a banking system you debit from an account, you issue a statement to your client. These are some typical interactions that occur between the objects of your domain.
- The language that the domain speaks. When you are modeling the domain of personal banking, terms like debit, credit, portfolio etc. or phrases like "transfer 100

USD from account1 to account2" occur quite ubiquitously and form the vocabulary of the domain.

- The context within which the model operates, which includes the set of assumptions and constraints which are relevant to the problem domain and are automatically applicable for the software model that you develop. A new bank account can be opened for a living person or entity only – this can be one of the assumptions that define a context of our domain model for personal banking.

Like any other modeling exercise, the most challenging aspect of implementing a domain model is to manage its complexity. Some of these complexities are inherent to the problem, you really can't avoid them. These are called the essential complexities of the system. For example, when you apply for a personal loan in your bank, determining the eligibility of amount depending on your profile has a fixed complexity that is determined by the core business rules of the domain.

This is an essential complexity that you cannot avoid in your solution model. But some complexities are often introduced by the solution itself, e.g., we implement a new banking solution that introduces extraneous load on operations in the form of additional batch processing. These are known as the incidental complexities of the model.

One of the essential ideas of an efficient model implementation is to reduce the amount of incidental complexity. And more often than not, we find that you can reduce the incidental complexities of a model by adopting techniques that help you manage complexities better. For example, if your technique leads to better modularization of your model, then your final implementation is not a single monolithic unmanageable piece of software. It's actually decomposed into multiple smaller components, each of which functions within its own context and assumptions. Figure 1 depicts such a system – we have shown 2 components for brevity. But you get the idea that with a modular system each component is self-contained in functionality. And interacts with other components only through explicitly defined contracts, which we call the context boundaries. This helps manage complexity better than a monolithic system.

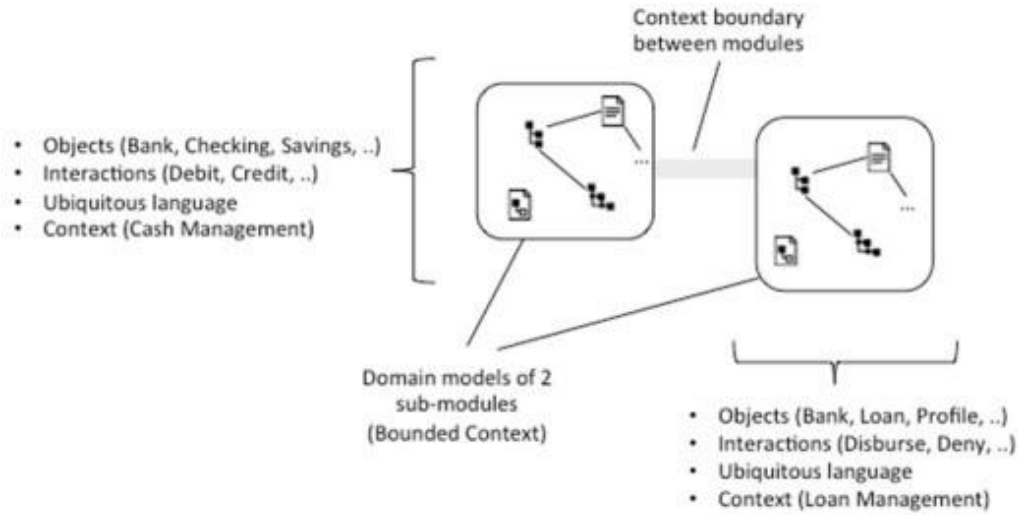

Figure 1 – Overview of domain model and its external context with terminologies from the domain of personal banking. Smaller modules each having its own set of assumptions and business rules are easier to manage than a large monolithic system. However need to keep the communication between them at a minimum and as very explicitly defined protocols.