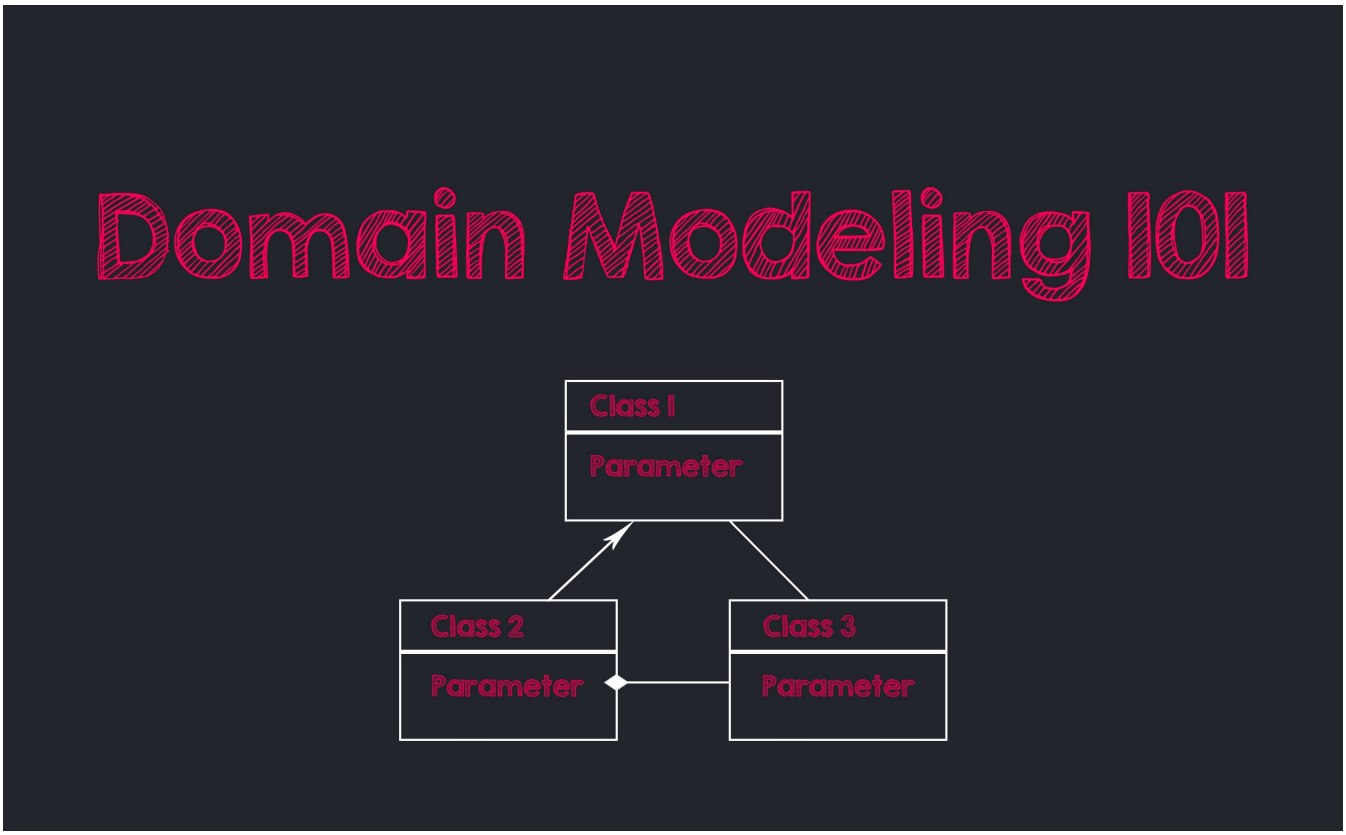
A Brief Introduction to Domain Modeling

Oleg Chursin
Dec 19, 2017 · 4 min read



Coming from a Cognitive Linguistics into the world of software engineering in general and object oriented programming in particular has its own benefits. Both worlds rely on conceptualization constructs which although having different implementation and context, possess common properties. At their core they provide “a way of characterizing the structured encyclopedic knowledge.” [1] And while in cognitive linguistics we deal with frames, mental spaces, idealized cognitive models, and semantic fields to name a few, the focus of this particular piece will be a **domain model** as applied to software engineering.

Defining a Domain Model

What is a domain model? Here are some definitions that are used in articles and research papers on DDD (Domain Driven Design):

1. “The Domain Model is your organized and structured knowledge of the problem. The Domain Model should represent the vocabulary and key concepts of the problem domain and it should identify the relationships among all of the entities within the scope of the domain.” [2]
2. “A domain model is a visual representation of conceptual classes or real-world objects in a domain of interest.” [3]
3. ‘A visual dictionary of abstractions.”[3]
4. “A *Domain Model* creates a web of interconnected objects, where each object represents some meaningful individual, whether as large as a corporation or as small as a single line on an order form.” [4]
5. “… a conceptual model of the domain that incorporates both behavior and data.” [5]

We may even come up with our own mega definition:

Domain model is a structured visual representation of interconnected concepts or real-world objects that incorporates vocabulary, key concepts, behavior, and relationships of all of its entities.

The key word in the above definition for me personally is *visual*. My brain performs at a maximum if a theoretical concept can be visualized in a digestible and somewhat eye-pleasing manner. Most of the time a domain model is illustrated with a set of class diagrams which may show:

- domain objects or conceptual classes
- associations between conceptual classes
- attributes of conceptual classes

Building a Domain Model

Let’s take a look at an example of modeling a particular domain and analyze its components. We will be dealing with a simplified **Vehicle** domain. Here’s a verbal description that will serve as a base for our model:

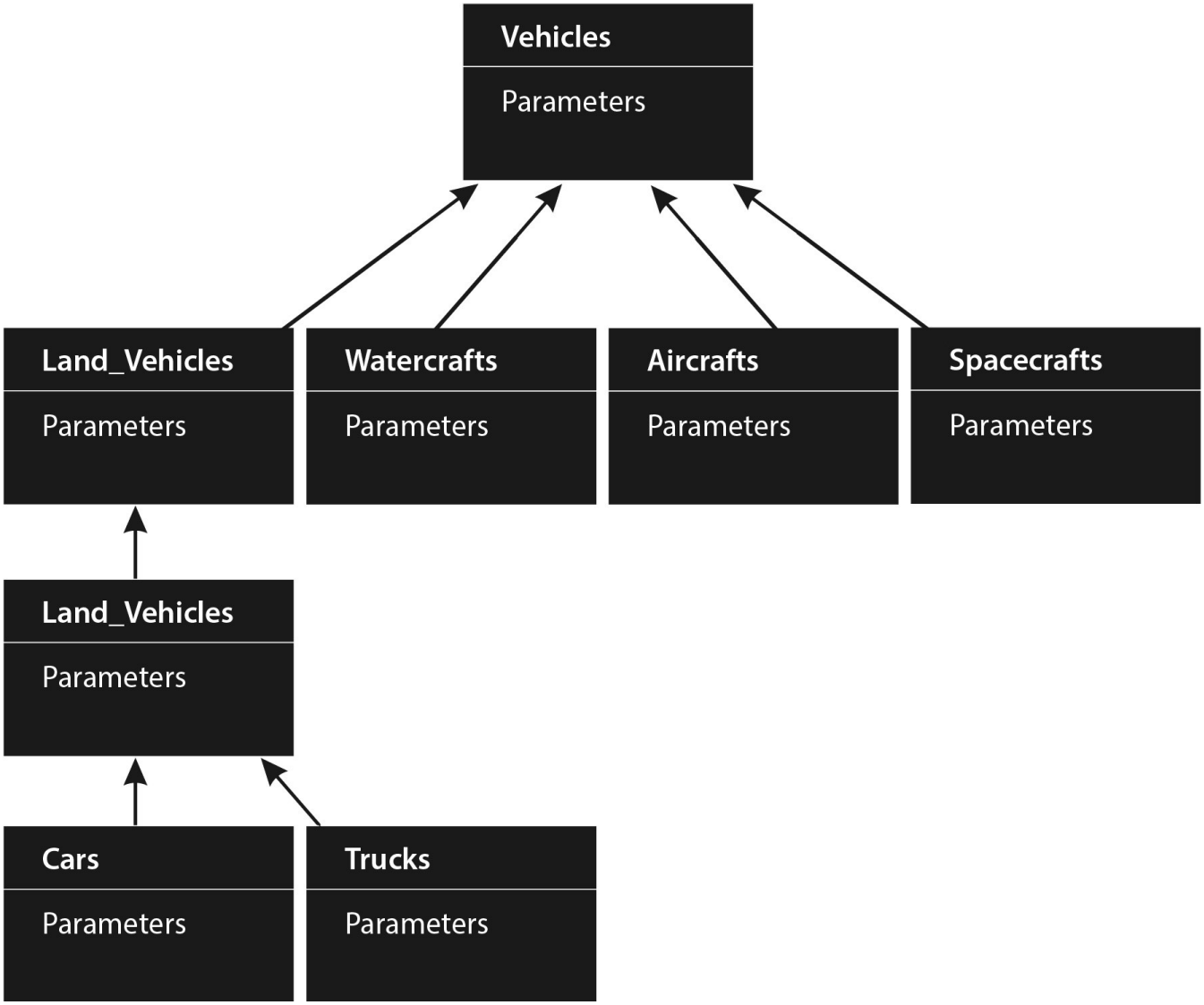
*A vehicle is a mobile **machine** that transports people or cargo. Typical vehicles include **wagons, bicycles, motor** vehicles (**motorcycles, cars, trucks, buses**), **railed** vehicles (**trains, trams**), **watercraft** (**ships, boats**), **aircraft** and **spacecraft**. **Land** vehicles are classified broadly by what is used to apply steering and drive forces against the ground: **wheeled, tracked, railed** or **skied**.*

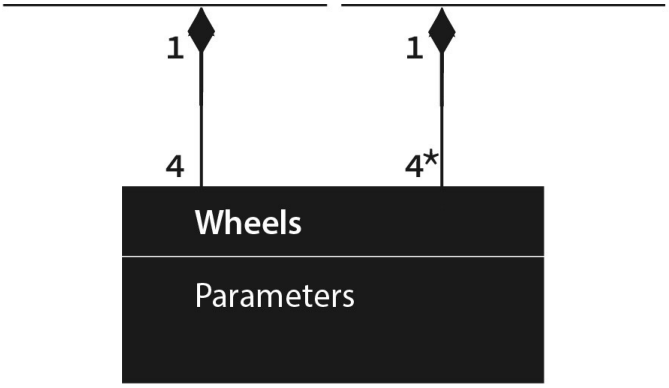
Using Noun Phrase Identification to Single Out Conceptual Classes

We will be applying a useful technique — linguistic analysis: identifying the nouns and noun phrases in textual description of a domain, and considering them as candidate conceptual classes or attributes. Therefore, here's the list of our class candidates:

Machine ,wagons, bicycles, motor vehicles, motorcycles, cars, trucks, buses, railed vehicles, trains, trams, watercraft, ships, boats, aircraft, spacecraft. Land vehicles, wheeled, tracked, railed, skied.

Transforming the above list into classes with the corresponding `Class_names` , this is what our **Vehicle** domain model may look like:





Let's dive into the structural components of the above domain model. We are using UML (**Unified Modeling Language**). A concise structural analysis will give us the following results:

1. When we see a straight line connecting 2 classes together this shows us that these 2 classes are **related**.
2. An arrow implies a **relation of inheritance**. So for the diagram above, its should be obvious that `Land_Vehicles` will inherit some methods from `Vehicles`.
3. The line with a diamond on the end states is that class 'A' **contains** class 'B'. For example, above we say that a `Cars` and `Trucks` contain `Wheels`.
4. Numbers. This is a case of **multiplicity**. It defines how many instances of a class A can be associated with one instance of a class B. In other words, what numbers state is a relation such as one-to-one, or one-to-many. In the example above we have these numbers on the 'contains' relation for `Cars` and `Trucks` and `Wheels`. We may say that one (1) `Car` contains four (4) `Wheels`. And one (1) `Trucks` contains four or more (4*) `Wheels`. It is important to note, that we can only use numbers on plain relation lines and contains relations, but NOT on inheritance relations. [6]

. . .

Sources:

1. Cienki, Alan. (2007) Frames, Idealized Cognitive Models, and Domains. *The Oxford Handbook of Cognitive Linguistics* 170–187. Oxford: Oxford University Press.
2. Brown, Philip. (11.12.2014) What is the Domain Model in Domain Driven Design? *Culttt* Retrieved from: https://goo.gl/cLDiTT

3. Larman, Craig. *Applying UML and Patterns. An Introduction to Object-Oriented Analysis and Design and the Unified Process.* Second edition.

4. Fowler, Martin. (11.11.2009) Framework Design Guidelines: Domain Logic Patterns. Retrieved from: https://goo.gl/mWsL5R

5. Wikipedia article on Domain Model.

6. Badgerati.(11.26.2009) What is Domain Modeling? *Computer Science Source* Retrieved from: https://goo.gl/ti382L

Software Architecture Domain Modeling Conceptual Class Ddd