Licenciaturas em Engenharia Informática

Modelação e Design

AULAS LABORATORIAIS – 2017 / 2018

FICHA 6 – ARCHITECTURE

Example – Part I

Consider that the system sequence diagram (SSD) in Figure 1 and the domain model (DM) in Figure 2 were obtained for a "Register Sale" use case.

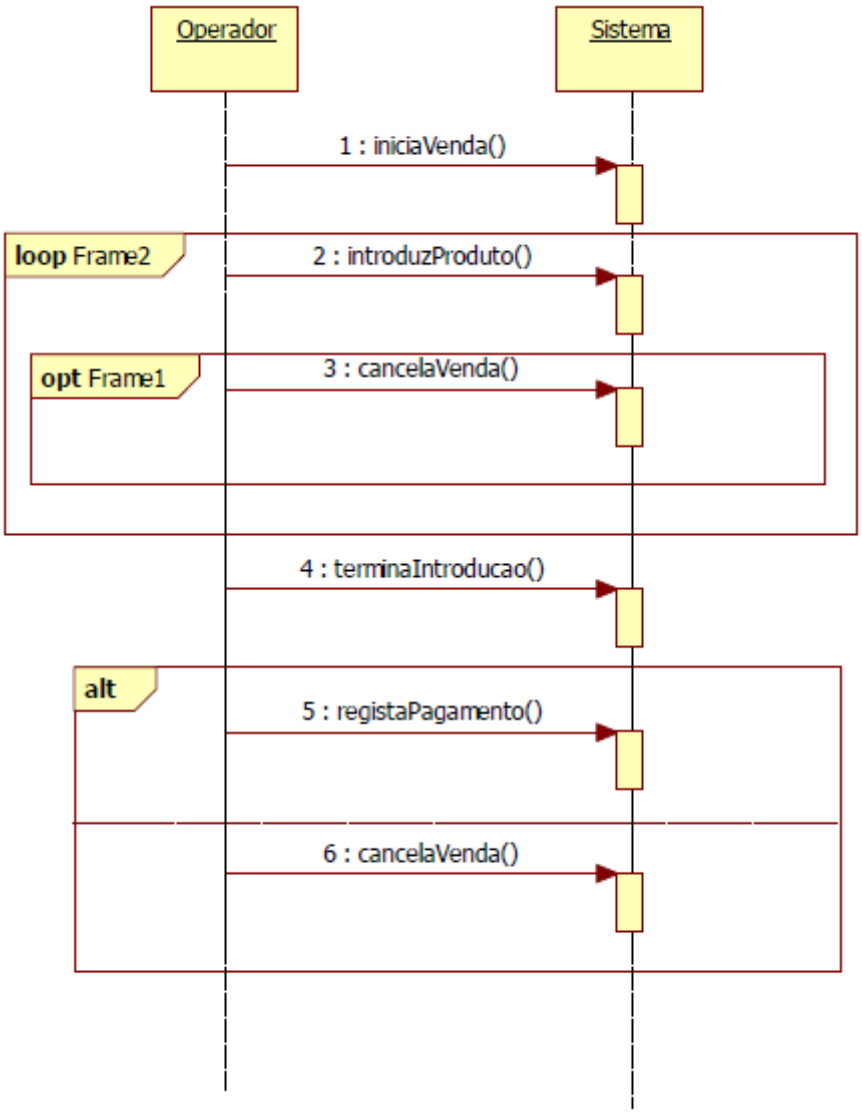

Figure 1) System sequence diagram - "Register Sale" use case

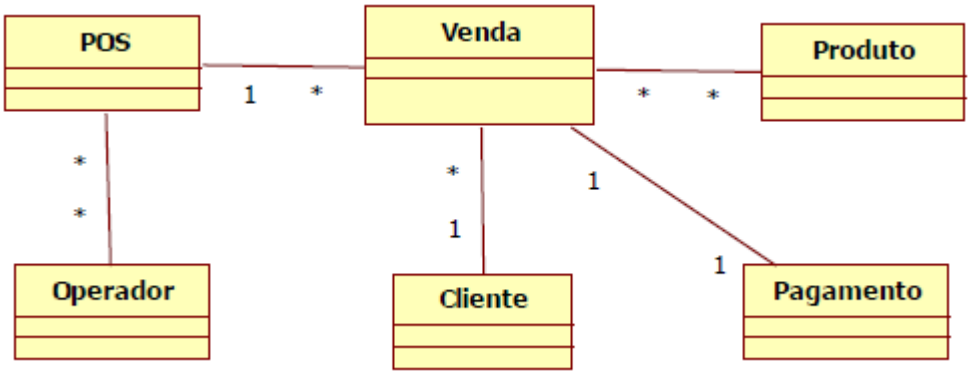

Figure 2) Domain model - "Register Sale" use case

How should development continue?

Having established the messages that circulate between the actor and the system and having identified the most important concepts, it becomes necessary to design the software. The purpose of this activity is to get a description of the structure and behaviour of the software, usually by creating a set of adequate diagrams (in particular, class diagrams and sequence diagrams).

How to design the system?

The general strategy for designing a system can be described as follows:

1. The messages identified in the SSD represent stimuli that elicit a response from the system. Therefore, the response to this message should be considered as a responsibility to distribute by a set of appropriate classes and methods.
2. The assignment of responsibilities is done by identifying the methods that implement them, what the relevant data is, and in what classes these methods should be placed.
3. The quality of the solution must be assessed, considering the basic principles object-oriented analysis:
   a) The data should be close to the methods that manipulate it.
   b) Classes should have high cohesion. That is, they should not accumulate (many) very different responsibilities.
   c) There should be low coupling among the various classes. That is, a class should not depend on many other classes.

These criteria sometimes conflict, and it is sometimes necessary to accept certain compromises.

The design classes should be, wherever possible, inspired by the classes identified in the domain model. However, other classes that are not in the domain model can be created. Typically, it is also necessary to create the data model (to ensure the persistence of the relevant data in a database).

Example – Part II

Consider again the diagrams in Figure 1 and 3. The message iniciaVenda() is triggered when the operator wants to start recording a new sale. For now, the emphasis is placed on the design of the part of the system that is responsible for responding to this message.

A relevant extract from the description of the use case, is:

1. The operator chooses the option that starts a new sale.
2. The system displays a welcome message.

In a supplementary specification (e.g. a glossary) it is known that each sale has a unique identifier and is associated with the information of the products sold and the type of payment.

Therefore, when the user intends to start a new sale, it is necessary to initialize a new sale, empty, with a unique identifier and present an appropriate message. Thus, one must identify the data involved in this operation and what is (or are) the class(es) that contains it. In this case, it is necessary to store the information about the sale that is initiated.

As such, it is necessary to identify the class that is be responsible for this. How do you choose? The following process will be used:

1. If a suitable class already exists in the design model, this class is used.
2. If not, look at the DM and check if any of the classes, represented there, are a suitable candidate to include the new methods and data.
3. If no class of the DM seems appropriate, ... a new class can be defined!

In the present example, since this is the first message that is being implemented, there is no "candidate class" that already defines this operation. For this reason, it will be necessary to introduce a new class.

Looking at the domain model, the POS class can be considered a suitable candidate, since, semantically, it seems to make sense to "start a sale on POS". We will create an initial version of the design, responsible for giving the answer. This is done by constructing the appropriate sequence diagram and class diagrams.

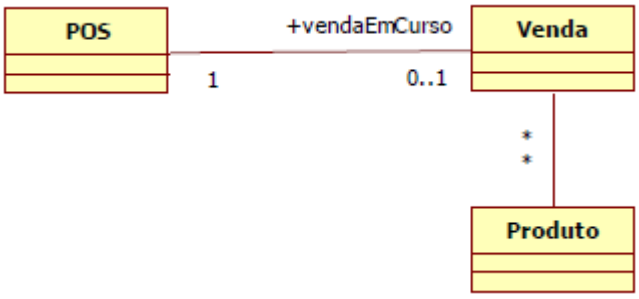

Figure 3) Class Diagram – initial version

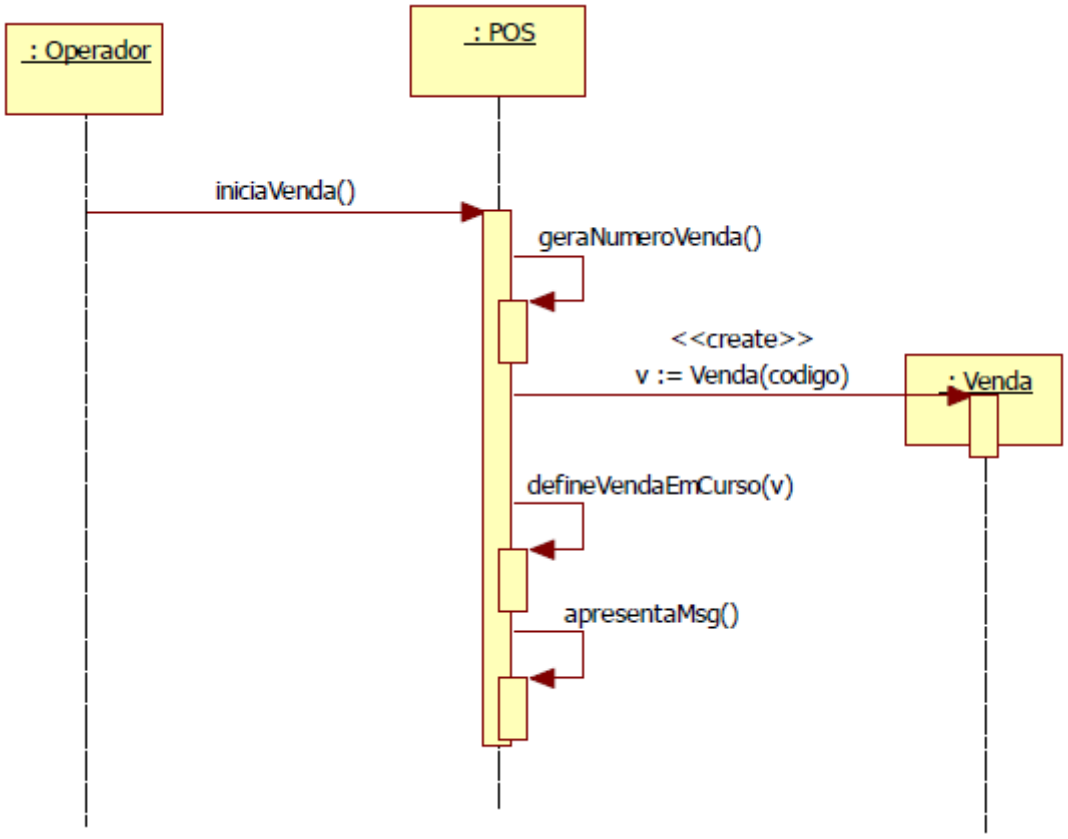

Figure 4) Sequence Diagram – initial version

Together, the two diagrams, give a concrete idea of the classes that are needed and the collaboration between them. At the same time, it is also possible to critically analyze the solution, taking into account the above criteria indicated (cohesion, etc.). In view of these criteria, it can be seen that:

POS class is responsible for creating the new sale, for generating the unique code, and for displaying the message.

It can thus be concluded that the POS class has too many responsibilities. This is not advised because it makes it more difficult to maintain (makes it more difficult to understand, modify, etc.). To solve this problem, we will reorganize the structure, eliminating this disadvantage. This is achieved by moving some of the responsibilities to other classes (and possibly to new classes).

Consider the generation of the unique identifier by the following process:

1. Looking at the classes currently in the model, it may seem that the Venda class is a reasonable candidate for generating the unique identifier. While this is not a bad option, it should be noted, however, that the Venda class refers to a single sale only, while the generation of a unique identifier refers to ALL sales (previously) generated. Thus, one can choose to assign this responsibility to a new class, specifically created for that purpose.
2. As before, one should look at the DM to find a suitable class. However, in this case, there does not seem to exist a reasonable alternative to the POS class.
3. It is therefore necessary to create a new "artificial" class, responsible for creating the "unique sales number": GeradorIdentificador.

Using an analogous process, it can be concluded that a class "InterfaceVenda" can be defined, which will be responsible for receiving the input of the user and presenting the message. Making these two modifications, results in the new design shown in Figures 5 and 6.

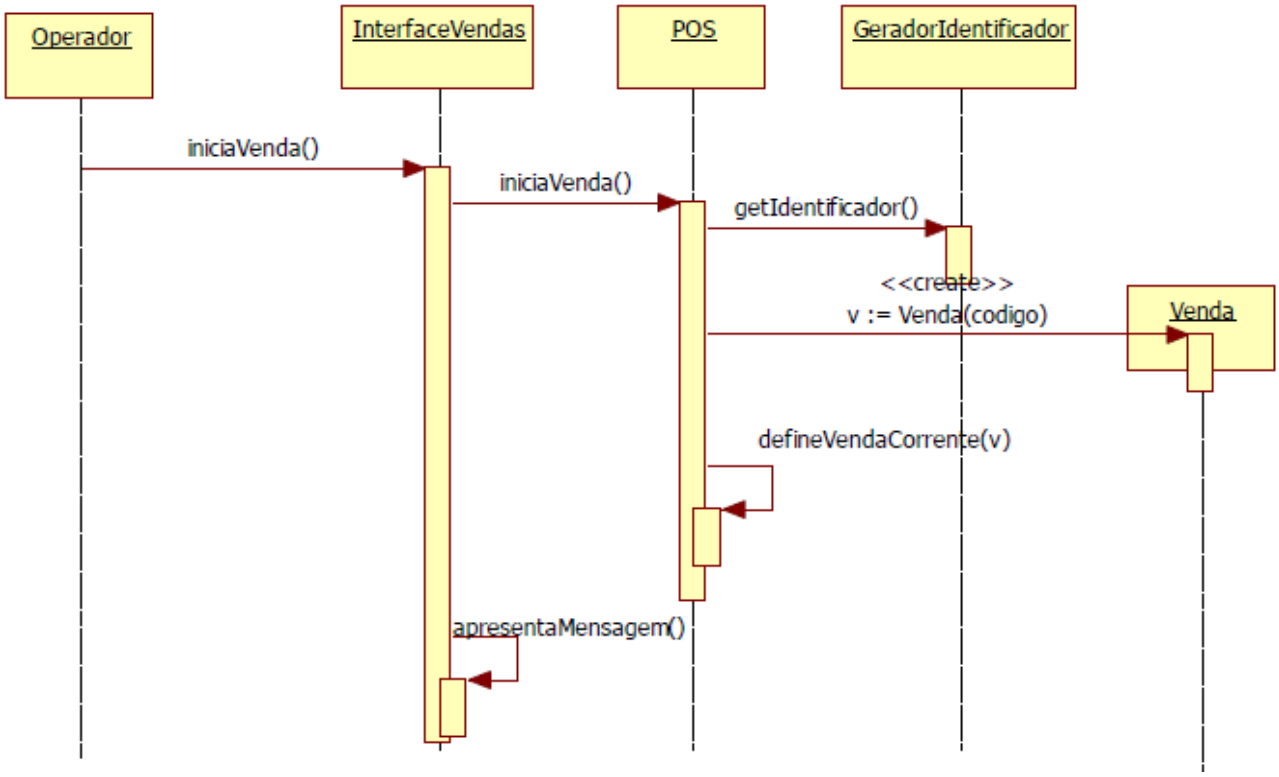

Figure 5) Sequence Diagram – updated version

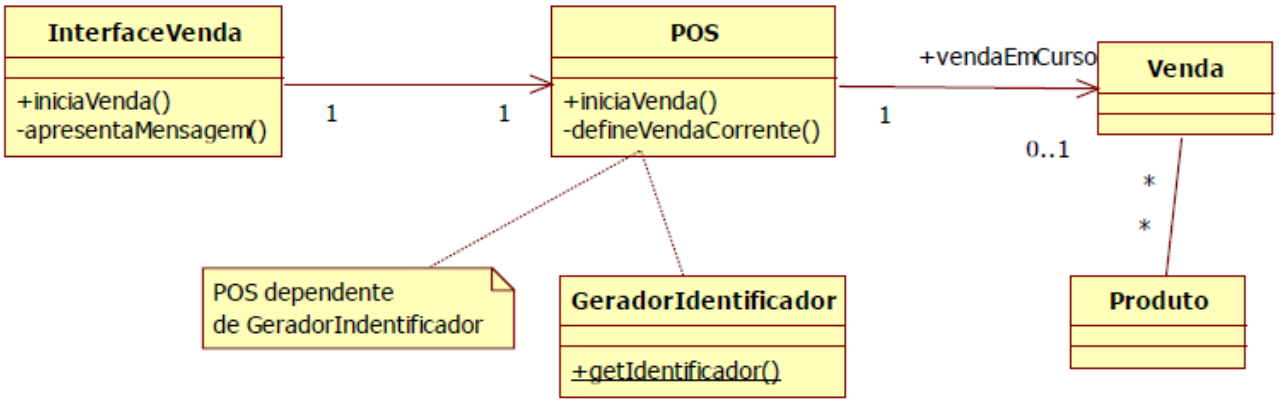

Figure 6) Class Diagram – updated version

Note that the getIdentificador method is static. In addition to including the new classes, the Class diagram also displays the following information:

- The methods are in the appropriate classes.
- According to the behaviour described in the sequence diagram, one can already identify the visibility of the methods as well as the navigability between classes InterfaceVenda, POS and Venda. Naturally, this information can be updated if the design of the other parts of the system reveals that, for example, a given method needs to have its visibility modified.

According to the criteria, the new solution is better than the initial solution. If it is still not suitable, new iterations can be performed until the new solution is satisfactory. The drawing process would continue, for example, by repeating the process with the next message (introduzProduto). The result is the integration of the outcomes obtained in several iterations.

FINAL REMARKS

The solution has an aspect that is common to most applications: the interface is dependent and separated from the logic and the data of the application, but the reverse is not true. Although this was not initially true, one could quickly reach this solution. This type of situation is relatively common and gave rise to the study of software design patterns, which systematize typical problems and respective solutions.

Naturally, it is not mandatory to consider one message at a time. If it is easier, or more logical, to work on a broader scenario, this can be done. However, one way to deal with complexity is to divide a complex problem (the entire system design) into several simpler sub-problems (the design of the elements necessary for the response of a single message).

Note that the role of the domain model is to provide us with a suitable nomenclature for the classes we need to create, while the role of the system sequence diagram is to identify the messages to which the system must be able to develop.

The design of the classes of the system should not be done independently, but in parallel with the design of the collaboration among them. Otherwise, there is the risk of projecting "naive" solutions, usually adapted more or less directly from the domain model, which are inadequate to the problem in question.

Lastly, nothing forces a system message to match only a single message (or method) in the drawing diagram. For example, a system message introduzLogin(username, password) can be decomposed into a two messages (for example, introduzUsername(username) and introduzPassword (password) if convenient).

EXERCISES

1. Do you propose any further modification to the presented drawing? If yes, justify, considering the quality principles set out in the text.

2. Consider the system message introduzProduto(codigoProduto, number). Draw the software design that corresponds to the processing of this message.