Simplified Modeling Language

Yarco Wang

2016/05

Revision History

Revision	Date	Author(s)	Description
0.2	2016/06/02	Yarco Wang	word correction, add "*" grammar
0.1	2016/05	Yarco Wang	first edition

Abstract

SML, the short name of Simplified Modeling Language, which is also named as Simple Modeling Language or Small Modeling Language, is a modeling language used in modern development process. Actually, it is not just a modeling language, it is a part of my XP(Extreme Programming) experiment.

It contains only *three* types of diagram: *Class-ER Like Diagram*, *Use Case Diagram* and *Activity Diagram*.

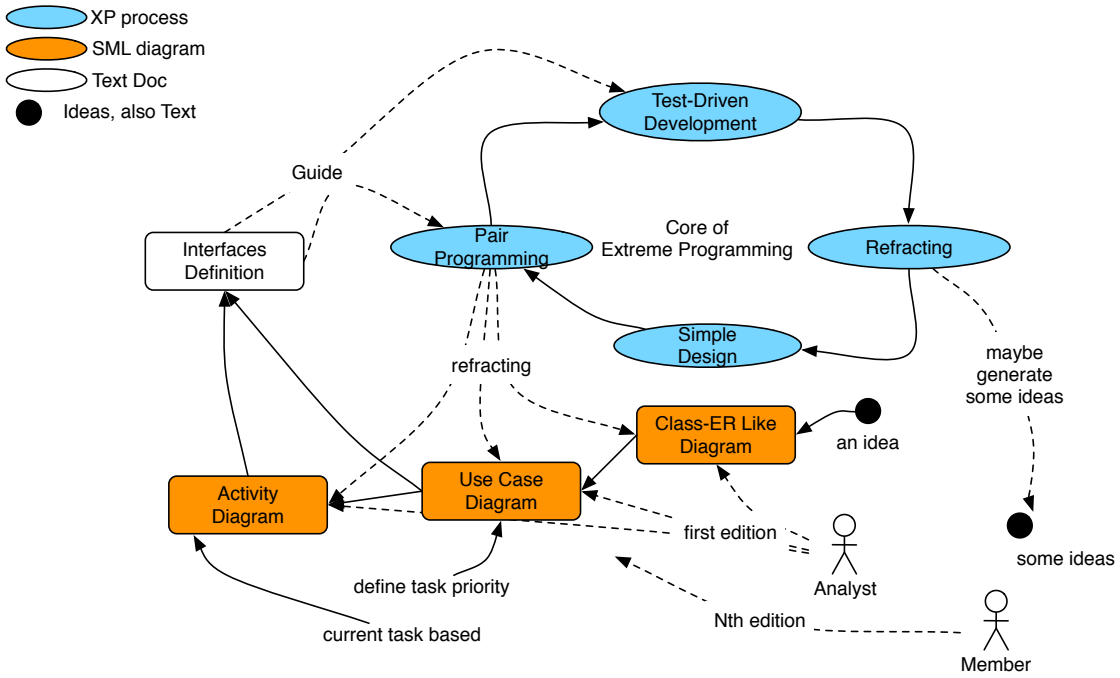
1 Workflow

For better understanding the SML, let's first take a look at the core of XP (colored in blue), in the chart below.

A project begins from a great idea. In XP, it is named as "Simple Design". Normally, in our case, it should be a file with an explanation of the idea, in which it may only take less than 100 lines.

And then, the analyst defines the initial version diagrams. After finished writing the doc "Interface Definition", the pair of the developing group begin to work: one for coding, and the other for writing tests.

When during the coding or discussing the requirements, they may find some information lost in the doc, it should also be refracted in the doc – then the doc is not just the analyst's task, it becomes something in continuously improving.



Simplified Modeling Language

2 Diagrams

2.1 Basic Ideas

Normally when a project started from an idea, it could be more or less than 20 lines text. We accept it, and consider it as just a process to be more clear. The modeling is not just for building the models at once before the coding start. It is a way continuously responding to the current software design.

So when at the beginning, it is a design; then, it turns to be a doc and communication tool; and at the end, it becomes a part of the doc.

Why we are lazy on this? Because sometimes, we can not restrain ourself: as an analyst, he may lost in his imagination, then the doc becomes something large but not really useful. So there are some rules on this when considering drawing our SML:

- If unnecessary , we are not going to do that;
- If something unclear, we are not going to do that (just leave "..");
- If the cost of dealing the thing than the profit we could get, we are not going to do that;
- Try to make the thing re-usable and add more functions on it;
- When drawing, we should be an artist, but not the machine which creates similar products, each diagram should be unique when at first glance;
- We draw things in "tree" like pattern, not "graph" like pattern ("tree", "graph" here is the something in data structure);
- We prefer curve lines, and even you can put flower on the diagram to make it special, So everything good for remembering the diagram are encouraged;
- Colored thing is welcomed to figure out current tasks (or priority).

2.2 Class-ER Like Diagram

The *Class-ER Like Diagram* is a diagram which merges class diagram(UML), Entity Relationship Diagram, mind map and web CRUD OPs' thinkings together.

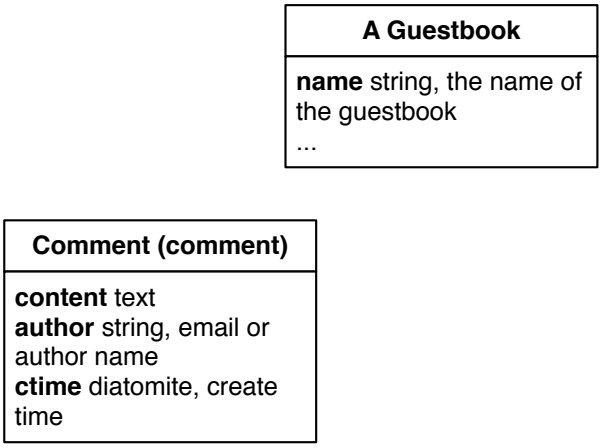
When the project started from 20 lines text, there are different types of items in the description. You can recognize them as "object" or "action". Ex.:

> This is a guestbook where guest could write down comments.

Here, the *guestbook* and *comments* are the "object"; *write down* is the "action". We ignore those actions cause for now the requirements are not very clear, we only care about those objects.

For more complex project, actually, each object should be represented as a diagram when the requirements become more clear; but here, cause it is very simple. We put *guestbook* and *comments* together.

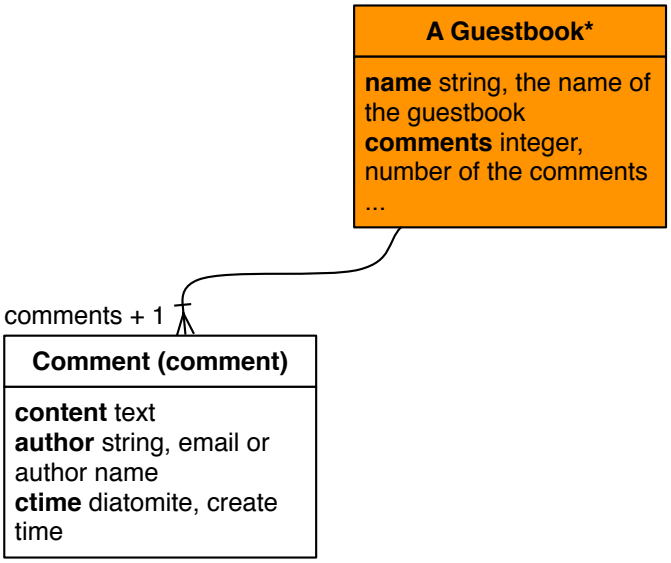
2.2.1 Entities



In this diagram (it is just like the class diagram in UML), but we noticed several differences:

- The title of the object is not very strict (here, "A Guestbook"). You can add extra text in the title between the parenthesis (here, "Comment (comment)""). The meaning between the parenthesis depends on the team. Here we defined it as a *tablename*. You can even add more, ex. "Comment (comment, cache)" means it should also be cached or optimized.
- The format of the attribute contains *a name*, *the type* and *the description*. It is not very strict also. Actually, the type is also a part of the description. You can even add more like "default to null".
- The "A Guestbook" entity also has a field "..." which figure out the requirements are not clear. But it won't stop us to do a prototype.

2.2.2 Connections And Other



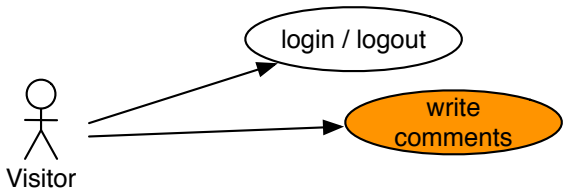
A Guestbook

We are going to add more. The relationship (connection) between the entities normally figure out the inheritance between different classes in UML. And they also have "association" relationship etc. But we use ER relationship to replace it. The reasons are all listed here:

- In modern concepts, we are going to avoid the inheritance because of strong coupling and dependence. We also invent "inject" for that purpose. Then it is in someway reducing the importance of old style relationship. But ER pattern is more simple, clear and easy to remember.
- The connection is represented as curve, but not a direct line.
- The entity "A Guestbook" is colored in orange to figure out it is the root (to make you focus on it).
- The mark "*" after "A Guestbook" means there are details somewhere not in current chart.
- The text "comments + 1" above entity "Comemnts" figure out that when creating (CRUD), it should trigger relevant action (but you don't have to add more for RUD OPs, it is just a remind).

2.3 Use Case Diagram

Use case diagram is the same as in UML, but we add the color to figure out the task current we focus on (or the priority). (The very important thing is you should keep the idea in mind – it is dynamic, not fixed.)



The use case diagram also mean there are several interfaces and unit tests behind each of the use case.

2.4 Activity Diagram

The activity diagram is also the same as in UML, but it is only to explain the details in *Use Case Diagram*, and also for current tasks – that means it changes quite often more than use case diagram.

3 Text Doc

Just like when at beginning, we only have one line text. After we did previous diagrams, we are going to write the interfaces which is very clear in our use case diagram and activity diagram. But of cause, it is in text format (so we are not going to describe it here). As you can view in the main workflow, the pair of the developers' work depends on this. Then, they will also do response to the doc (the diagrams or the text files) if anything require changing.

4 Contact Author

You can contact author through yco_w at me.com about the SML if you do have good idea.

He also accept the donation through paypal using the same email address. Even USD $1 is welcomed .