DECLATIVE METHOD FOR CAPABILITIES DEFINITIONS

Dr. Viktor Sirotin

January 2018

In RPSE (Reification as Paradigm of Software Engineering)[1] we use the term “capabilities” instead of “requirements”. We're doing this for pragmatic reasons. The term "capability" can be used at all stages of the product life cycle. Even consumers of the product understand what is its capability. A smartphone owner can answer much more easily to the question of which capabilities his device has than to the question of which requirements his device meets.

Capabilities can be described declaratively (what needs to be done) or imperatively (how it can be done).

This article briefly describes the rules for optimal declarative capabilities definitions and methods for quality control of these definitions.

Ingredients for declarative capabilities definitions

For optimal[2] definition of capabilities, one needs:

- List of domain objects
- List of operations
- Template

You will find a detailed introduction to the theme Domain Objects in the classic book [2]. In one of my next articles, I plan to describe a pragmatic method for searching and defining domain objects and operations.

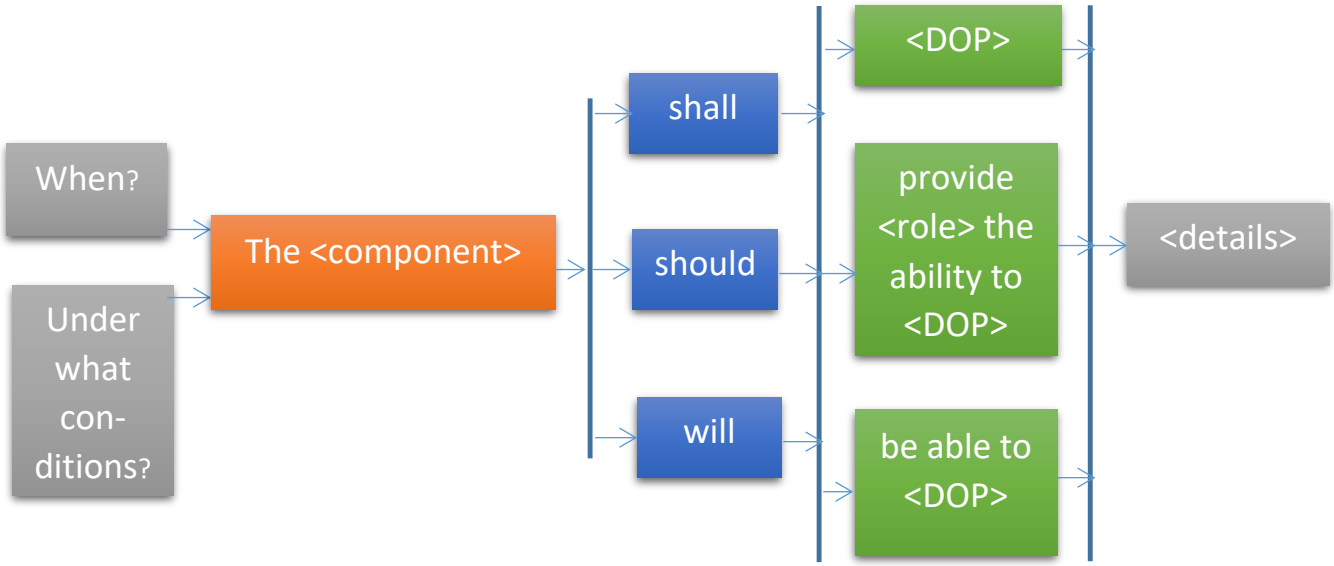
You can define the list of domain objects and operations in the form of a simple table (glossary). This table usually contains two columns: term name and its meaning within the domain under consideration.

Domain Objects (DOB) and Domain Operations (DOP) are very important parts of Capabilities Definition. From my experience, however, I can say that the most difficult mistakes happen with formulations of capabilities. That is why we will focus on this problem first.

[1] See [1].

[2] I assess method described in this article for capabilities definition as optimal based on my personal experience and comparing with other methods. In my opinion, this method is less time consuming than others and gives very good results. Of course, my assessment is subjective.

I found the best solution for this problem in the book [3]. That was three templates for different types of requirements. I merged these templates into one template for capabilities definition:



The term "component" refers to the system, application or app to be developed or its parts.

We recommend to use modal verbs as follows:

Shall - This behavior must be implemented (K.O. criterion).

Should - This behavior should be implemented if the implementation cost is not disproportionately high. It is mostly an application of technologies, processes, algorithms, etc. that have been little tried and tested.

Will - It would be nice if this behavior is implemented, but this development does not have to cause large implementation costs.

It is amazing that with this template up to 95% of system capabilities can be defined. This means that almost for every capability from practical projects there is a way on the diagram above that allows define it correctly.

A correct definition must use exactly one **orange**, one **blue** and one **green** box. The use of grey boxes is optional.

Some examples of capabilities definitions with coloring of mandatory definition parts:

By achieving of 90% of capacity, **the system** **shall** **notificate** the monitoring system about this event.

The resource management sub-system **shall** **provide the system administrator the ability to remove** user data for each registered user.

The system **will** **be able to recommend** the user next possible operation in current context.

It is useful to present the definition together with relevant information in tabular form:

Element	Meaning
ID	Id or number inside of document or model.
Main part	Definition according template above.
Source	Where this capability was notices or occurs (document, meeting, interview).
List of used domain objects.	Here are listed the domain objects, operations and roles used in main part. Although these elements can be found there, it is better to list them in these separate lines to facilitate comparison with the glossary.
List of used domain operations	
List of used roles	
Structure information	Normal (default), abstract, parent, child, depends on, etc.
Supplementary explanation	Supplementary text information, that explains somewhat, bat is not mandatory for implementation and test.

Normally the first five first rows are mandatory in this tabular form.

How to check the quality of declarative capacity definition

If you have a set of declarative capability definitions[3], you can check their quality analogously to the quality of a set of axioms in mathematics. Namely, you need to evaluate the following aspects:

- Completeness[4]
- Consistency
- Correctness

Below I propose a checklist for quality assurance review of declarative definitions of capabilities.

Completeness

1. Together, the capability definitions cover the entire behavior of component that shall be implemented[5].
2. Mandatory rows in the table must be filled.
3. For each abstract capability there exist one or more derived capabilities.
4. For each parent capability there exist one or more child capabilities.
5. For each child capability there is exactly one parent capability.

Consistency

6. No capability definition is a contradiction of another capability definition.

Correctness

7. Template: Each capability definition uses template and “three color rule” is fulfilled.

[3] Of course, if you have a mix of declarative and imperative capability definitions, you must apply quality check for both definition types.
[4] You are able to check completeness only if your specification document or model claim to be complete.
[5] This check is easier than you think when it comes to migrating or refactoring an existing system. In this case, the "old" behavior is known.

8. DOBs: Used in main part object belong to list of objects defined in glossary or has unique and clear meaning.
9. DOPs: Used in main part verbs (<DOP>) belong to list of operation defined in glossary or has unique and clear semantics.
10. Supplementary explanation: Supplementary explanation contains only a few sentences. They explain some additional details, but not define essential additional behavior.
11. Implementability: Developers do not need any additional information to understand what needs to be implemented[6]. (Capabilities definitions and glossary contains enough information for these professionals.)
12. Testability: Testers do not need any additional information to understand how to test implemented behavior. (Capabilities definitions and glossary contains enough information for these professionals.)

References

1. Viktor Sirotin. RPSE – Reification as Paradigm of Software Engineering. https://arxiv.org/abs/1810.01904
2. Eric J. Evans. Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison Wesley. (20. August 2003). ISBN-13: 978-0321125217.
3. Chris Rupp, die Sophisten. Requirements-Engineering und -Management: Professionelle, iterative Anforderungsanalyse für die Praxis. Carl Hanser Verlag GmbH & Co. KG. ISBN-13: 978-3446405097.

[6] This allows a developer to understand what is expected, but not immediately to know how to implement it. In this case, a developer may need additional information (e.g. description of algorithms or external components) or help from consultants.