Facets of Concerns

Crenguța Bogdan
"Ovidius" University, Constanta, Romania
cbogdan@univ-ovidius.ro

Abstract

The concern facet concept is introduced. We claim that in a world of "concerns" a cluster of facets is associated with each and every concern. The facets in the cluster may structure the concern specification. In order to demonstrate this statement, we define a group of concern facets for specifying the concerns of the software process and software product quality and use them in developing software systems.

1. Introduction

Yet there are debates on the exact definition (meaning, as much as possible close to the intuitive meaning) of the "concern" notion. Nevertheless, the scientific world more and more accepts the concern as "any matter of interest in a software system"[8, p.128].

A great deal of the set of concerns comes from the software process model as shown in [10, 9, 7, 2]. These concerns guide the development of software systems and structure the software product.

Our claim is that the concerns can be structured, too. The structuring element is the facet concept, as presented in [1]. In order to highlight the use of facets in structuring of the software process and product, we will consider RUP (Rational Unified Process) as a model [6] of the software process. Some of the RUP concerns are identified.

2. Facets of concerns

In our opinion facets are that kind of questions one might want to ask about a concern. A facet instance is a possible answer to such question. Usually, a stakeholder raises such questions. Answering to them, the stakeholder specifies new facet instances. If the identified facets completely describe the whole concern, the stakeholder will be able to better control the concern. There are also cases when even a concern may query other concerns for information.

In order to explain better the role of facets in concern specification, let us consider a concern C and some typical questions about it:

- Definition: Which is the definition of C?
- Example: What are the things that satisfy C's definition?
- Address: When can we say that C is addressed?
- Resolve: When can we say that C is resolved?
- Generalization: Which other concerns have less restrictive definitions than C?
- Specialization: Which concerns have C's definition plus some additional constraints?
- Life Cycle Phase: Which are those concerns that are addressed or resolved in the same life cycle phase with C?
- Workflow: Which are those concerns that are addressed or resolved in the same workflow of iteration with C?
- Artifact: Which are the other concerns that help together with C in realizing the same artifact?
- Role: If C is relevant to a role, which other concerns are relevant to the same role?
- Dependency: Which are concerns that C is depending on?

In addition, each facet F of a concern C has subfacets that answer at the following questions:

- Check: How can elements of C.F be checked?
- Evaluate: How can elements of C.F be evaluated?

where C.F represents the facet F of a concern C.

The facets of a concern break into three categories:

- basic facets which contain intensive information on concern C itself: Definition, Address, Resolve;
- context facets which are linking the concern to other concern(s): Example, Generalization, Specialization, Life Cycle Phase, Iteration, Artifact, Role, Dependency;
- other facets which can be tacked into any facets from above: Check, Evaluate.

The paper defines some basic and context facets of concerns and introduces several examples of facets.

3. Basic facets

In this section we present three basic facets of concerns, namely: Definition, Address and Resolve.

3.1. Definition

To work with a concern, first of all we have to define it. The layout of this facet is a list of elements, where each element, subfacet of the facet, consists of the following attributes:

- Identification:
  - type: global/local,
  - visibility: opaque/transparent,
  - life cycle: software process/life-cycle phase/iteration,
  - inclusion: aggregate/atomic,
  - behavior: initiator/active/passive,
  - locality: crosscutting/not-crosscutting.
- Responsible: who is responsible with the concern resolving (most likely, a stakeholder).
- Audience: stakeholders that are interested in the concern.
- Collaborators: Using definition of other concern X, Reducing to the definition of Y, Same definition as Y except…, Specialized version of Z, etc.
- Predicate: a test that verifies if a given thing is an example of the concern.

Let consider the attribute values of the Identification subfacet.

A *global concern* is a concern of the entire software project and will be checked during the major milestones. Instead, a *local concern* is a concern of one or few iterations and will be checked during the minor milestone of the iterations. By example, *system_objective* concern is checked at every major milestone; instead, during the minor milestones, the *evaluating-criterion* concern is checked.

We define a *transparent concern* as a concern that can be resolved and checked. Instead, an *opaque concern* can only be addressed, because we don't have all the necessary information to resolve it. All the quality concerns of the product are opaque because only when the software project is closed, we can say if the system is of quality or not. Instead, *architecture-baseline* is a transparent concern in the elaboration phase of the software process: it will be resolved in this phase and will be checked during the architecture life-cycle milestone.

Until a concern is resolved, it evolves during the entire software process or only a part of it. This part may be an entire life-cycle phase or only one or few iterations of it. The life-cycle parameter of the Identification subfacet indicates the area of software process where the concern is relevant. By example, the *architecture* concern evolves through the software process like that:

- in the inception phase, an architecture statement is synthesized by evaluating the design trade-offs, problem space ambiguities and available-space assets (technologies and existing components),
- the elaboration phase builds up an architecture baseline that resolves the critical use cases,
- in the construction phase, the architecture is enriched by integrating remaining components in order to fulfill all the requirements from the vision document,
- during the transition phase, the architecture is modified in order to resolve multiple-component design issues that affect the quality attributes of the system, like reliability, performance and maintainability.

A concern is *aggregate* if its resolution needs the resolution of other concerns. Otherwise, we will call it *atomic* concern. By example, *architecture* is an aggregate concern because for constructing an architecture, we must firstly construct the "4+1" views [4]: *design view*, *process view*, *component view*, *deployment view* and *use case view*, which are concerns themselves. Furthermore, to obtain a *stable architecture* we have to resolve the quality concerns such as *performance*, *reliability*, *adaptability*, *robustness*, *scalability*, *economic-trade-offs*, *technology-constraints*, *comprehensibility* and *ease-to-construct*. These concerns are aggregate concerns, too.

In general, concerns obtained during the software process are aggregate. Atomic concerns are relative to the software system and depend on this.

A concern is *active* if its addressing or resolving implies that the system executes one or more activities. Examples include some features such as *modify-component* or *execute-a-use-case* in order to check the *operational-concept* of the system. Such a concern is a system concern and it is used when the system is viewed as an "open box" by the developers.

By definition, a concern has *initiator behavior* if it comes from the software process of the system, where the system is seen like a "black box" and addressing or resolving of it implies, actually, addressing or resolving of active concerns. By example, *operational-concept* concern of the system is an initiator concern.

All the concerns in the software process are crosscutting concerns, if we consider the system to be a base.

The subfacet Collaborators of a concern indicates all concerns and their facets, which help us to define the concern.

As we mentioned above, the Predicate subfacet denotes a test that verifies if an entity is an example of the concern. The test has to be (*i*) faithfully described by Identification, and (*ii*) related to other concerns belonging to the Collaborators facet.

For example, let consider the Definition facet of the *architecture* concern:

Identification: *Global*, *Opaque*, *Software process*, *Aggregate*, *Initiator*, *Crosscutting*

Responsible: *Software Architecture Team*

Audience: *Software Management Team*, *Software Development Team, Software Assessment Team*

Collaborators:

> **Concerns***:* {*the-system's-structure*, *the-system's-behavior*, *heuristic-rules-for-the-architecture-design*, *heuristic-rules-for-the-architecture-evolution*}
>
> **Facets:** {*scope-of-the-system, objectives-of-the-system*, *requirements-set*, *development-risks*, *quality-attributes-of-the-architecture*, *estimate-scope*, *schedule*}

Predicate:

```
resolve the-system's-structure and
resolve the-system's-behavior and
resolve heuristic-rules-for-the-architecture-design
and
resolve heuristic-rules-for-the-architecture-evolu-
        tion
and
check scope-of-system and
check objectives-of-system and
check requirements-set and
check development-risks and
check quality-attributes-of-architecture and
check estimate-scope and
check schedule
```

If the Definition facet of a concern has a list of elements, that is the concern has more definitions, there should be no conflict among each other. Namely, they must have the same characteristics (indicated in Identification subfacet), the same responsible and audience, and have different collaborators and predicates.

For example, the *architecture* concern may have another predicate facet:

Predicate:

```
resolve use-case-view
for each Example.use-case-view
    address design-view
if the system is concurrent then
    resolve process-view
if the system is distributed then
    resolve process-view and deployment-view
    resolve component-view
    check scope-of-system
    check objectives-of-system
    check requirements-set
    check development-risks
    check quality-attributes-of-architecture
    check estimate-cost
    check schedule
```

Two facets may overlap, that is their intersection is not empty or equivalent, if they contain one or more common concerns. In other words, the Predicate subfacets of the *architecture* concern are overlapping.

3.2. Address

A concern is *addressed* by a life-cycle phase or iteration of the software process if and only if it uses information from the phase or iteration, but this information is not sufficient for resolving the concern. Furthermore, this information should be checked during the major or minor milestones.

There are cases when addressing a concern needs to address or resolve other concerns. By example, the *architecture* concern is addressed in the inception phase of the process software when the *architecture-statement* is resolved. The same concern is addressed in the elaboration phase where the *architecture-baseline* concern is resolved.

Another example is given by *requirement* concerns that are addressed in the iteration of the elaboration phase, by example, where the set of *evaluating-criterion* concerns is resolved.

The Address facet contains the next subfacets:

- Locality: phase or iteration that addresses the concern,
- Collaborators: name of concerns that are addressing or resolving to address this concern,
- Predicate: a test that tells us if any given thing is an example of this concern.

By example, the Address facet of the architecture concern is the following:

Locality: *Elaboration phase*

Collaborators: *architecture-baseline*

Predicate: resolve *architecture-baseline*

3.3. Resolve

First of all, we are saying that a concern is *implemented* if there is a formalism or artifact that comprises descriptive material that include all information about the concern. A construction from some such formalism might be the unit concept from Hyperspace [5].

A concern is *resolved* if contains all needed information to implement the concern. This information must be evaluated with some metrics and obtain the consensus of all stakeholders who are interested in it. The *architecture-baseline* concern is resolved in the elaboration phase of the software process, when it is evaluated and approved by the manager and client. In this stage, we consider that the concern is closed, that is if we want to modify such concern, it enters in a SCO (Software Change Order) process.

The Resolve facet has the following subfacets:

- Locality: phase or iteration,
- Collaborators: names of the concerns that help to resolve this concern,
- Predicate: a test that tells us if some given entities are examples of this concern.

By example, the *architecture-baseline* concern has the following Resolve facet:

Locality: *Elaboration phase*

Collaborators: {*the-system's-scope*, *the-system's-objectives*, *the-project-vision*, *use-cases-set*, *quality-attributes-of-the-architecture*, *design-risks*, *planning-risks*, *work-and-progress-metric*, *change-traffic-and-stability-metric*, *breakage-and-modularity-metric*, *rework-and-adaptability-metric*}

Predicate:

```
check scope-of-system and objectives-of-system
check the project-vision and use-cases-set
check quality-attributes-of-architecture
resolve design-risks and planning-risks
check work-and-progress-metric and
      change-traffic-and-stability-metric and
      breakage-and-modularity-metric and
      rework-and-adaptability-metric
```

Obviously, Definition, Address and Resolve facets intuitively represent the states of a concern life cycle: defined (created), intermediary (addressed) and closed (resolved).

4. Some Context facets

In this section we present four facets: Life Cycle Phase, Workflow, Artifact, and Role. These facets correspond to points of view, which link through affiliation a concern to others. These facets have the same subfacets:

- Name of the point of view,
- Structure: a set of concern names.

The set of concerns in the Structure subfacet contains the concerns derived from the objectives of the point of view we consider in applying the facet.

4.1. Life Cycle Phase

In case of the Life Cycle Phase facet, the point of view, as we mentioned before, is the phase of software process in which a concern is addressed or resolved. In such phase, a set of concerns are addressed or resolved, namely those concerns that fulfill the objectives of the phase.

For instance, the *architecture-baseline* concern has the next Life Cycle Phase facet:

Name of phase: *Elaboration*

Structure: *vision-baseline*, *plan-baseline-for-construction-phase*, *design-decision* and *demonstrating-architecture*.

4.2. Workflow

Workflow facet focuses on the workflow of iteration and derives the concerns that fulfill the evaluation criteria of iteration.

For instance, the *architecture-baseline* concern has the next Workflow facet:

Name of workflow: *Design*

Structure: *architecture-concept*, *design-components*, *source-component-inventory*, *bill-of-materials* and *executable-components*.

4.3. Artifact

This facet contains those concerns that contribute together with the given concern to the realization of the same artifact. For instance, the *architecture-baseline* concern has the next Artifact facet:

Name of artifact: *Architecture description*

Structure: *architecture-views*, *architectural-interactions*, *quality-attributes-of-architecture*, *architectural-rationale* and *architectural-trade-offs*.

4.3. Role

The Role facet contains the concerns that derive from responsibilities of the stakeholder that addresses or resolves the given concern. For instance, the *architecture-baseline* concern has the next Artifact facet:

Name of role: *Software Architecture Team*

Structure: *architecture-prototyping, make/buy-trade-offs*, *primary-scenario-definition*, *primary-sce-*

nario-demonstration, *make/buy-trade-offs-baseline*, *architecture-maintenance, multiple-component-issue-resolution* and *quality-attributes-of-architecture*.

5. Conclusions

The concern facet concept is introduced by focusing on the software process and the software system quality concerns.

In the paper, examples have been given "from around" the architecture concern. Besides scope, objectives and requirements, the architecture concern is one of the most important concerns.

5.1. Future work

We are working on the specification all other facets we enumerated at the beginning of the paper.

Other possible questions are: how can we relate the facets of concern with Cosmos model [8] and how can we apply them to the Pirol environment [3]?

Finally, our claim is that the concern facet concept could be useful for systems analysis, too. Our further research will also investigate in this direction.

5.2. Acknowledgments

The author thanks to Professor Luca Dan Şerbănaţi for the paper's review. Also, my thanks to Stephan Hermann for helpful comments about concerns.

6. References

[1] R. Davis and D.B. Lenat, *Knowledge-based systems in artificial intelligence*, McGraw-Hill, USA, 1982.

[2] W. Harrison, H. Ossher and P. Tarr, "Software engineering tools and environments: a roadmap", *A. Finkelstein ed., The future of software engineering, 22nd Int. conf. on soft. Eng.,* Limerick, Ireland, ACM, New York, June 2000, pp. 263-277.

[3] S. Hermann, "Views and concerns and interrelationships. Lessons learned from developing the multi-view software engineering environment PIROL", *Phd Thesis*, Berlin, 2002.

[4] P.B. Kruchten, "The 4+1 view model of architecture", *IEEE Software,* 12(6), Nov. 1995, pp. 42-50.

[5] H. Ossher and P.Tarr, "Multi-dimensional of concerns and the Hyperspace approach", M. Aksit editor, Software Architectures and Component Technology, Kluwer Academic Publishers, 2001.

[6] W. Royce, *Software project management: a unified framework*, Addison Wesley, USA, 1998.

[7] S.M. Sutton, Jr., "Multidimensional separation of concerns in testing, *First workshop on Multi-dimensional Separation of Concerns in Object-Oriented Systems, Conference on Object-Oriented Programming*, *Systems, Languages and Applications,* Denver, Colorado, Nov. 1999.

[8] S.M. Sutton, Jr. and I. Rouvellou, "Modeling of software concerns in Cosmos", *Proc. of the 1st International Conference on Aspect-Oriented Software Development*, Enschede, The Netherlands, Apr. 2002, pp. 127-133.

[9] P.Tarr, H. Ossher, W. Harrison. and S.M. Sutton Jr., "N degrees of separation: multi-dimensional separation of concerns", *Proc. of the 21st International Conference on Software* Engineering, Los Angeles, CA, USA, May 1999, pp. 107-119.

[10] E.S. Yu and J. Mylopoulos, "Understanding "why" in software process modeling, analysis and design", *16th International Conference on Software Engineering*, Sorrento, Italy, 1994, pp. 159-168.