

Experiences with Scenario Based Architecting

Laurens Vrijnsen¹, Chris Delnooz¹, Lou Somers^{1,2}, Dieter Hammer¹

¹ Eindhoven University of Technology, Dept. Math. & Comp.Sc., P.O. Box 513,
5600 MB Eindhoven, The Netherlands
{l.vrijnsen, c.delnooz, l.j.a.m.somers, d.k.hammer}@win.tue.nl

² Océ Technologies BV, P.O. Box 101,
5900 MA Venlo, The Netherlands
lsom@oce.nl

Abstract

This paper presents the results of a project conducted in the Research Department of Océ Technologies. Here, many ideas are waiting to be explored, preferably by means of a prototype on a real product. To facilitate such prototyping activities, a framework was constructed. This framework faced the classical problem of software engineering: the many stakeholders have very different ideas on what the framework should be, and no one is able to write down proper and stable requirements. If we apply traditional software engineering techniques and write down requirements to be implemented, little guarantee can be given about the end result: it is indeed what was specified, but now that people can play with it, they find out that those specifications are not quite right.

We explore an alternative approach. First, we want to *achieve stable understanding of what the stakeholders expect from the system*. Usually, there is no alignment between the various stakeholders about the problem, let alone about the solution. As a result, requirements keep changing during the realization. Secondly, we want to *transform this knowledge into a solution that meets all expectations (a “good architecture”)*. Often, a big gap exists between the formulated requirements and the resulting system. Partially, this gap can be explained by incomplete requirements, but there is a need for methods that help to manage the complexity of a software design as well.

The steps we take are as follows. Together with the stakeholders, we construct *scenarios*: textual descriptions (stories) of what the system will look like, and what its characteristic features are. This helps to create a shared vision among the stakeholders: every stakeholder finds at least one scenario that describes the system from his point of view. Subsequent analysis of the set scenarios reveals the *goals* that the various stakeholders pursue with the system. Here we apply ideas from Goal Oriented Requirements Engineering (GORE): we define goals from the scenarios and derive requirements from them. Finally, we create a software *architecture* that meets the goals (and related requirements and qualities) by using Quality driven Architecture Design and Analysis (QADA).

Keywords: requirements engineering, scenario, goal, quality attribute, architecture

1 Introduction

”Scenario” and “goal” are concepts used in many different contexts like requirements engineering [10] and architecture evaluation [8]. Therefore we start with a model (Figure 1) to describe the notions in our approach.

For the creation of new products it is important to look at future developments. A *trend* is (the evolution of) one characteristic element of a possible future, e.g. “communication will be wireless”. Trends are technological as well as organizational and social: “people will feel comfortable talking to a device”.

The *stakeholders* (customers, developers, etc) determine which trends are captured in a vision on a future product (family). The *vision* combines the selected trends into a solution. It is shared between the stakeholders by means of scenarios. In this paper a *scenario* is a *story* (a textual description) that (partially) describes the future system and its environment from the viewpoint of one stakeholder. A textual description is ambiguous, and scenarios are not intended to replace requirements. They help in *communication* of the vision, and thus in generating feedback and improving the vision.

Each scenario visualizes a number of *goals*, the main objectives the stakeholders have in mind for the system. For example, a stakeholder may aim at a system that “always supports the latest version of the PDF standard”. The corresponding scenario explains that the system will be extensible with respect to new versions of this standard. Here, extensibility is a *quality attribute*: a system property by which stakeholders will judge its quality. Typically, one goal results in a number of *requirements*, SMART (Specific, Measurable, Accurate, Realistic and Time bound) demands on the realization of the system. *Design decisions* establish the link between requirements and architecture. A design decision is a (well founded) selection of one alternative for the technical realization of the system. The sum of all design decisions defines the *architecture*, usually described in a number of views [7].

The above definitions fit in the model in [7] for the description of an architecture: The vision can be interpreted as a high level architectural description: it describes the system, its mission and environment. The scenarios provide the different views on the architecture. Each scenario covers a number of viewpoints that belong to some stakeholders. Their goals are the concerns addressed by the viewpoints.

Figure 2 positions related methods in the System Requirements (SR), Architectural Design (AD) and Detailed Design (DD) life cycle phases.

Roadmapping [16] studies *trends* that are important for the future of a system. Use cases from UML [17] and use case maps, UCM [20], come close to our definition of *scenarios*, but we see them as more low level. Our scenarios are high level descriptions of the total system in its context, whereas a use case explores only one piece of functionality.

Various papers ([2],[11],[21]) advocate using *goals*, since goals are more stable than requirements. A goal oriented approach like KAOS [11] or GRL [6] starts with goals and derives requirements from them. URN [1] combines UCM (operational and functional requirements) and GRL (non-functional requirements).

CAFCR [15] presents a similar approach to requirements as part of a bigger architecting method. In addition to goal orientation by means of key drivers, scenarios are suggested to create a shared vision. Here, goals are not pursued for their stability.

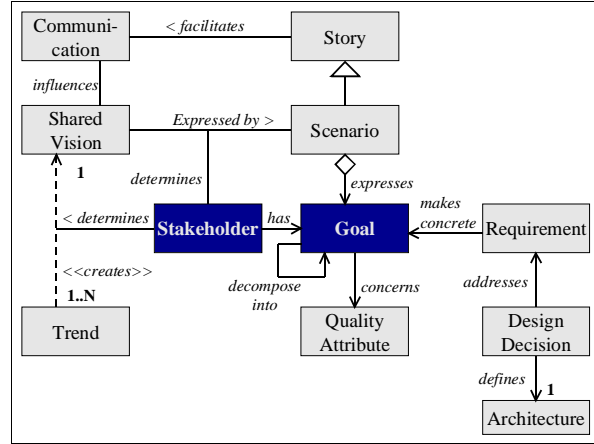


Figure 1: Relations between concepts

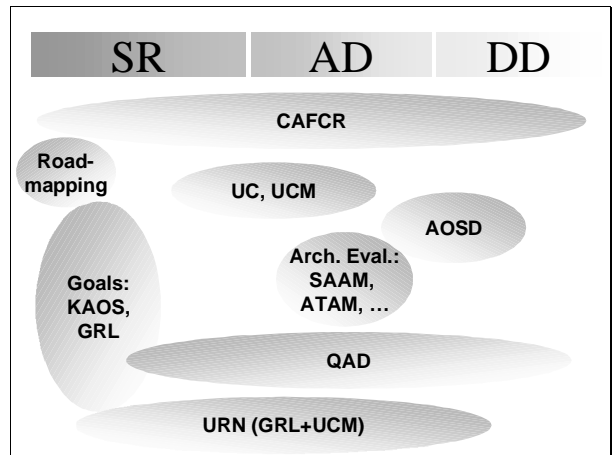


Figure 2: Related work vs. life cycle phases

In the context of architecture evaluation [8] (SAAM, ATAM), many practitioners use the word “scenario” for change cases. A change case describes possible modifications to an architecture with the goal of evaluating it.

[19] states that “*quality attribute requirements ... have a significant influence on the software architecture*”. Thus, we improve our chances to achieve a good architecture by adopting a design method that takes quality attributes into account from the start. Literature provides a number of such methods. [14] proposes to use quality attributes as “needles” piercing through the architectural views, thereby integrating the different views. AOSD [3] represents a collection of methods that consider quality attributes as system aspects. Each aspect is developed separately, and eventually specific tools combine the aspects. [4] focuses on the relation of quality attributes and architectural mechanisms. Each mechanism implements a strategy to achieve some quality attributes. A fairly recent method is Quality driven Architecture Design and Analysis [12]. QADA emphasizes “utilizing architectural styles and patterns as a means of designing high quality architectures”. It has a strong focus on quality attributes, analysis methods and design rationale, together with a reasonably well defined process.

2 Case study: a prototyping framework

Our case study considers the construction of a prototyping framework on top of a controller for a family of multi-functionals (printer, scanner and copier combination). Researchers have many ideas for new applications on this controller, but need to familiarize with many concepts before they can use it for their purposes. Additionally, prototypes typically need a considerable amount of engineering before they are turned into a real product. The framework had to capture the essentials of the controller to allow building good prototypes.

A framework is a set of semi-complete applications that can be specialized to produce custom applications. It has features similar to product line architectures [12] (basis for a wide range of systems, reusable software components, low development cost/time, low maintenance cost). It has a positive influence on productivity and software quality [13], but requires thorough understanding of the domain and how it will be used.

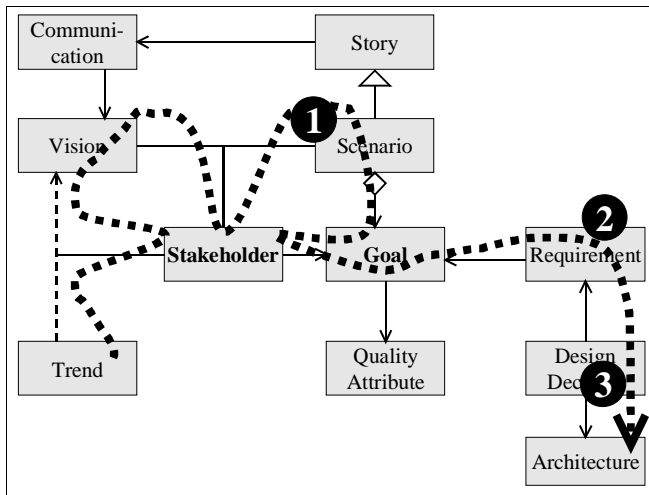


Figure 3: Mapping steps on the domain model of Figure 1

Figure 3 shows the logical ordering of our steps: we started to explore trends likely to show up in prototypes. Together with the stakeholders we selected the relevant ones, created a vision and shared that vision by means of scenarios (1). Analysis of the scenarios (2) yields a number of goals that are validated with the stakeholders. The goals are input for the definition of requirements and subsequent design decisions (3) that define the architecture. Step 2 involved GORE, step 3 QADA.

The process involved many iterations over communication, vision, scenarios and goals to achieve a stable shared vision. Also, the stakeholders were frequently involved in the trajectory leading to goals. The next sections elaborate the steps by an example focusing on the impact of maintainability.

2.1 Step 1: Share your vision on the future

To begin with, we had to determine the stakeholders. We identified a.o. the Researcher who wants to prototype, the Maintainer to maintain the framework, and the Engineer to translate a prototype into a real product. Together with (representatives of) the stakeholders, we decided on relevant trends for prototypes. We observed a number of technical trends likely to be prototyped. Also, prototyping should be fast in order to test many alternative ideas in a short time, and will be achieved through combining various tools.

The resulting vision is described in a number of scenarios, at least one for each stakeholder. One scenario describes how a researcher realizes his ideas about a networked application with the help of our framework. It tells how he glues together pieces of functionality – from the framework and other sources – using a scripting language. Another scenario looks at the framework from the viewpoint of a maintainer who has to ensure that research prototypes continue to function even if the interface of the underlying controller changes.

“(…) Ralph is curious, so he takes a quick glance at the source code of the ImageManipulator. As it turns out, it was quite easy to build: it receives data from the scanner, makes some transitions (using the library of image manipulation algorithms that is already available for years) and puts back the new images”

“(…) When Ralph runs his demonstration on the new version of [the framework], he is happy to find out that he does not have to change his code”

Figure 4: Scenario fragments

Figure 4 shows two snippets from the scenarios. They only address the *characteristic features* of the framework like easy integration with existing tools, little overhead for prototypes, and a number of prototype ideas. The scenarios are close to what our stakeholders expect from the product, at the same level of detail and in their language. They provide little barriers to reviewing (as opposed to long lists of requirements) and leave some room for discussion. In discussing them, the goals became clearer to all stakeholders. However, in this stage it is hard to foresee the technical consequences of the features described in the scenarios.

2.2 Step 2: Analyze the scenarios

The scenario analysis helps to build commitment among the stakeholders. During the analysis, we also explored enabling technology as a first draft of a feasibility study. The scenarios describing the use of the framework are a good starting point for the definition of use cases. For example, the first quote in Figure 4 leads to a use case “Data Processing” (Figure 5).

1. The prototype is informed about new data
 2. It processes the data and gives the result back to the framework; in turn the framework ensures that the result is placed in the correct place
 3. Generating a result automatically creates a progress event for the next worker
- (…)

Figure 5: Use case “Data Processing”

The most important part of the analysis aims to identify the stakeholder goals: we answer the question *why* the scenarios are the way they are by making the underlying goals explicit. We identified each stakeholder by a role name and stated his goals and associated ISO [9] quality attributes (Figure 6). The goals are by no means concrete enough to be requirements, but describe the problem to be solved. This provides a stable starting point for requirements.

Maintainer is a researcher who has to maintain the framework (for new prototypes); his goal is *minimal maintenance effort*. Nobody has time for extensive maintenance activities; each researcher building a prototype will probably do a little maintenance. The framework must provide maximal decoupling from the controller, concise interfaces, and good documentation. The following quality attributes are applicable:

- 1) Maintainability, and more specifically:
 - a) Analyzability: capability of the framework to be diagnosed for deficiencies or for parts to be modified to be identified
 - b) Changeability: capability to enable a specified modification to be implemented
 - c) Stability: capability to avoid unexpected effects of modifications
- 2) Portability, and more specifically:
 - a) Adaptability: capability to be adapted for different environment
 - b) Installability: capability to be installed in certain environment

Figure 6: The stakeholder Maintainer, his goals and associated quality attributes

Using Goal Oriented Requirements Engineering techniques from KAOS, the goals were step wise refined into specific requirements. For example, the goal for Maintainer has two causes (Figure 7): adding functionality already present in the controller (the “maintainability” quality), or moving to a new controller version (the “portability” quality). Zooming in, changeability and adaptability are concretized by change cases like “Offer support for PDF documents”. These change cases fill the gap between the goals and what will be realized in the first version of the system. The stability aspect of maintainability, indicated in the second quote of Figure 4, can be realized by decoupling the prototypes from the implementation of the controller.

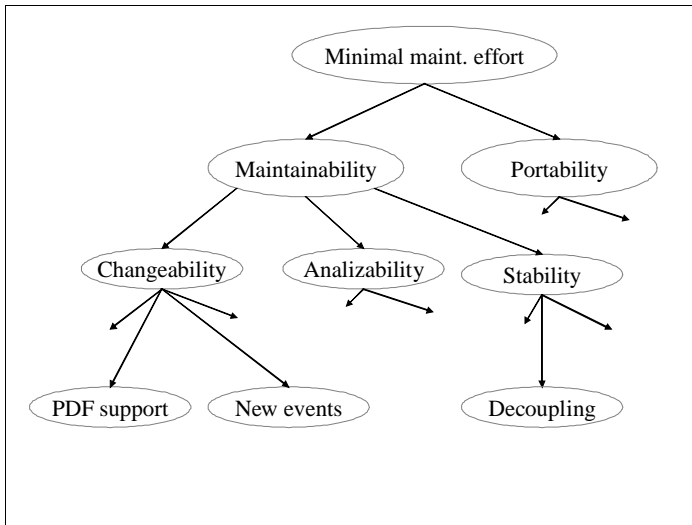


Figure 7: Derivation of requirements

The tree of (sub)goals grows quickly, which requires tool support to keep it consistent. However, this administrative burden pays off in terms of better traceability, better requirements, and more stakeholder satisfaction. Experiences with KAOS [11] support this belief.

The stakeholders were involved until their goals were known, not in the formulation of requirements. Although this seems surprising, it reflects the current practice: stakeholders tend to think in terms of scenarios to be realized, not in lists of detailed requirements. Requirements usually have a technical focus that zooms in on small parts. This is important for the realization, but introduces a big risk of losing the context and not understanding how each requirement contributes to the system.

The derivation of requirements serves more as a tool to understand the stakeholders, than to construct a contract with them. Understanding the rationale behind requirements helps to take technical decisions that eventually lead to an accepted product. Especially in first-of-a-kind products, requirements are very volatile. The stability of goals makes them much more suitable to steer the design, since design decisions should be based on quality attributes, which can be linked directly to goals.

2.3 Step 3: Decide on the architecture

We applied QADA to create a software architecture that meets the goals. We started by exploring the design space, i.e. by quickly scanning a large amount of possible (partial) solutions and combining them into a number of rough architectures. Since each fulfilled all goals, they should be equally favorable to the stakeholders. However, from the feedback – rejecting some alternatives – we had to conclude that we missed some important goal initially. After additional interviews we had a clearer vision on the final product and updated the scenarios.

QADA stresses the need to keep a design rationale (the set of all design decisions). This triggers frequent analysis of design decisions with respect to goals and qualities. We used the change cases, which were important because maintainability was a vital quality attribute, to evaluate the architecture, similar to SAAM. Figure 8 shows an example of a design decision. The rationale and consequences demonstrate the influence of the quality attributes in Figure 6 and the derivation in Figure 7.

Description	The framework combines the controller architecture “pattern” with the Interceptor pattern
Alternatives	<ul style="list-style-type: none"> There are no real alternatives to the controller architecture: because of the goal to make transfer to engineering easy, the controller architecture has to be mimicked For the communication with the prototype several options exist: <ul style="list-style-type: none"> – Interceptor pattern [18] that encapsulates only relevant data in Context – Observer pattern [5] that only notifies about changes and leaves data access to the observer (hence is likely to expose all data, unless additional abstractions are provided)
Rationale	The chosen patterns help to mimic the controller architecture, thus facilitating transfer to engineering. The interceptor pattern helps to hide the data models on the controller, thus meeting the goals to decouple and ease of use
Consequences	<p>Close fit with identified goals and quality attributes, but requires some understanding of the controller in order to build correct prototypes. It also results in strict separation between Client, Worker and WFM. The various interceptor, dispatcher, context and synchronizer classes follow from the decision to apply the Interceptor pattern.</p> <p>New functionality from the controller can be added by introducing new dispatchers and interceptors for related events</p>

Figure 8: A design decision

[5] and [18] indicate the consequences of patterns in terms of affected quality attributes. Therefore, we used the quality attributes to pick standard architecture and design patterns. Knowing the priority of quality attributes, we judged whether the trade-offs are acceptable. This is similar to the ATAM that analyzes the trade-offs in qualities that follow from the design decisions.

3 Conclusion and discussion

As shown by our case study, *scenarios* offer a powerful tool to create a shared vision among the stakeholders and to explore their *goals*. We feel that scenarios are an important first step, before elaborating on goals and requirements, in understanding what the stakeholders expect. Usually they tend to think in terms of stories to be realized, not in long lists of detailed requirements. A goal oriented approach helps to discover stakeholder goals and related *quality attributes*. The latter provide guidance during the architecting process by steering trade-offs.

Figure 9 summarizes how scenarios influence the design decisions: they are input for the definition of stakeholder goals and use cases. The goals help to define quality attributes and requirements. If maintainability is an important quality, change cases are also provided. All provide input to the design decisions that determine the architecture.

We used the scenarios to steer the development process, constantly checking whether the provided functionality and quality attributes are enough to realize the scenarios.

During the case study, stakeholders indicated that sometimes the rationale is more important than the requirement itself. This reflects our belief that goals are more important than requirements: in the end it only matters that the stakeholders receive a product that meets their goals, which is not necessarily the product that was specified upfront. In real products, often the end result is (very) different from the requirements specified at the start. Moreover, requirements only show bits of the system, but lose the big picture. Scenarios show the big picture and focus on those (functional and non-functional) elements that determine the success of the product.

The analysis of a scenario to extract goals depends mainly on a number of interview sessions with the stakeholders, steered by the scenarios. The approach would benefit from guidelines for the construction of scenarios to ensure that the necessary information is present.

Traditionally, requirements are also used as a contract with the stakeholders. In our case stakeholders are not much involved in the definition of requirements and such a contract does not exist. The identified goals are not specific enough to serve as a contract either. Future work should look into methods for validation and verification, i.e. to traverse the model of Figure 9 bottom-up.

References

- [1] D. Amyot, G. Mussbacher: *URN: Towards a New Standard for the Visual Description of Requirements*, <http://www.usecasemaps.org/pub/sam02-URN.pdf>
- [2] A. Antón, J. Dempster, D. Siege: *Managing Use Cases During Goal Driven Requirements Engineering: Challenges Encountered and Lessons Learned*, North Carolina State Univ, TR-99-16, December 1999, <http://www.csc.ncsu.edu/faculty/anton/pubs/icse2000.pdf>
- [3] *Aspect Oriented Software Design*, <http://aosd.net>
- [4] L. Bass, M. Klein, F. Bachmann: *Quality Attribute Design Primitives*, CMU/SEI-2000-TN-017, December 2000, <http://www.sei.cmu.edu/publications/documents/00.reports/00tn017.html>
- [5] E. Gamma, R. Helm, R. Johnson, J. Vlissides: *Design Patterns – Elements of Reusable Object Oriented Software*, Addison-Wesley, 1995
- [6] *Goal Oriented Requirements Language*, <http://www.cs.toronto.edu/km/GRL/>
- [7] IEEE Computer Society: *IEEE Recommended Practice for Architectural Description*, IEEE Std 1471, October 2000
- [8] M. Ionita, D. Hammer, H. Obbink: *Scenario Based Software Architecture Evaluation Methods: An Overview*, <http://www.win.tue.nl/oas/architecting/aimes/papers/Scenario-Based%20SWA%20Evaluation%20Methods.pdf>
- [9] ISO Committee: *Software Engineering – Product Quality – Part 1: Quality Model*, ISO/IEC 9126-1, 2001
- [10] M. Jarke, X. Tung Bui, J. Carroll: *Dagstuhl Workshop on Scenario Management*, February 9-13, 1998

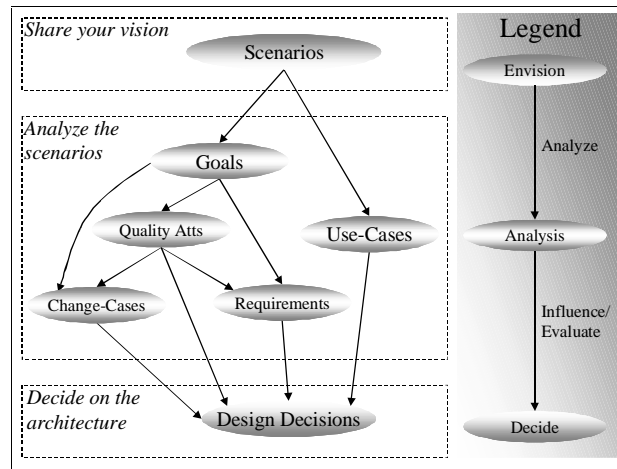


Figure 9: Artifacts in the design process

- [11] C. van Lamsweerde: *Goal Oriented Requirements Engineering: A Guided Tour*, Proc. 5th IEEE Int. Sym. Req. Eng. (RE'01), Toronto, August 2001, pp. 249-263
- [12] M. Matinlassi, E. Niemelä, L. Dobrica: *Quality driven architecture design and quality analysis method*, VTT, 2002, ISBN 951-38-5968-1, <http://www.inf.vtt.fi/pdf/>
- [13] M. Morisio, D. Romano, I. Stamelos: *Quality, Productivity, and Learning in Framework Based Development: An Exploratory Case Study*, IEEE Trans. Software Eng., Vol. 28, No 9, September 2002, pp. 876–888
- [14] G. Muller: *Qualities as integrating needles*, Philips Research, 2002, <http://www.extra.research.philips.com/natlab/sysarch/QualityNeedlesPaper.pdf>
- [15] G. Muller: *Architectural reasoning: balancing genericity and specificity*, Embedded Systems Institute, Eindhoven, 2003, <http://www.extra.research.philips.com/natlab/sysarch/ArchitecturalReasoningBook.pdf>
- [16] G. Muller: *Roadmapping*, Embedded Systems Institute, Eindhoven, 2003, <http://www.extra.research.philips.com/natlab/sysarch/RoadmappingPaper.pdf>
- [17] J. Rumbaugh, I. Jacobson, G. Booch: *The Unified Modeling Language Reference Manual*, Addison-Wesley, 1999
- [18] D. Schmidt, M. Stal, H. Rohnert, F. Buschmann: *Pattern Oriented Software Architecture – Patterns for Concurrent and Networked Objects*, Wiley, 2000
- [19] SEI: *Reasoning About Software Architectures*, April 22, 2003, http://www.sei.cmu.edu/ata/reasoning_about.html
- [20] <http://www.usecasemaps.org/index.shtml>
- [21] E. Yu and J. Mylopoulos: *Why Goal Oriented Requirements Engineering*, Proc. 4th Int. Workshop Req. Eng: Foundations of Software Quality, Pisa, June 8-9, 1998, Presses Univ. de Namur, 1998, pp. 15-22