Experiences with scenarios and Goal-Oriented RE

Chris Delnooz, Laurens Vrijnsen [1]

This paper explores how scenarios and Goal-Oriented Requirements Engineering (GORE) can be combined to achieve a stable set of requirements. In a case study we explore how requirements for a framework are derived from goals expressed in a number of scenarios, focussing on maintainability.
Scenarios have proven to be a strong tool in visualizing a future product and thus the requirements that describe its characteristics. Especially the mission-critical characteristics are emphasized in scenarios, since they focus on the essentials. Although GORE poses serious concerns on the topic of scalability, it pays off in terms of better requirements and more satisfaction from stakeholders.

1 Introduction

"A good start is half the work;" in software engineering, a good start is often interpreted as a stable set of requirements that define the product to be realized. In this paper, we explore the two problems enclosed in this statement:

1) How to find requirements; there is a big gap between the stakeholders and the architect, especially when new types of products have to be made.
2) How to achieve to a *stable* set of requirements; many projects suffer from requirements that keep changing.

We believe that a shared vision on the product is essential to obtain good insight in the requirements. Therefore we take one step backwards and start with scenarios that describe the product from various viewpoints. In this paper we explore how Goal-Oriented Requirements Engineering (GORE) techniques can be applied to derive requirements from these scenarios.

This paper is a deliverable in the context of the MOOSE project ([MOOSE]). The MOOSE project aims at to improving the quality and productivity of software development through the development of a framework of methodologies and supporting tools for the embedded domain. To achieve this goal, the MOOSE project (1) develops new methodologies or improves on existing ones, and integrates these methodologies in the framework, and (2) validates the framework by means of case studies. In this context, we do not aim at developing a new method, but focus on combining existing methods.

1.1 Terminology

"Scenario" and "goal" are concepts used in many different contexts (e.g., requirements engineering [Jarke et al, 1998], and architecture evaluation [Ionita et al, 2002]). Therefore we start with a domain model to describe what we consider to be a scenario, and how it interacts with other important concepts in our approach. Figure 1 serves as a graphical companion to our explanation.

[1] The authors are members of the Software Technology program at Technische Universiteit Eindhoven ([OOTI]). They can be reached at c.delnooz@tue.nl and l.vrijnsen@tue.nl respectively.

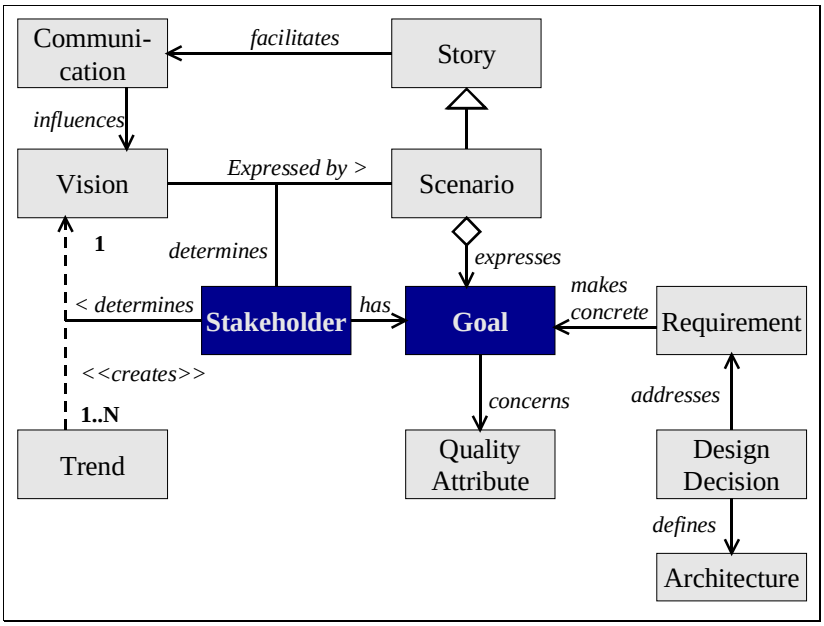

Figure 1: Domain model for concepts in the chosen approach.

For the construction of new products, it is important to look at future developments which we call trends. A *trend* is (the development of) one characteristic element of a possible future, e.g., "communication will become wireless" or "paper will be used as little as possible, since it is considered to be irresponsible to use paper". Notice that not only technological trends exist, but also organizational and social trends: "people will feel comfortable talking to a device".

Obviously, a company that produces medical equipment is interested in different trends than a company that offers solutions in document management. The *stakeholders* determine which trends are captured in a vision on a future product (family). Example stakeholders are the customers, and maintenance engineers. A *vision* is a personal image different for each of the stakeholders, a mental picture that combines the selected trends into a solution.

However, the vision must be shared within the team that will be responsible for the realization of that vision. This is where scenarios can help ([Jarke et al, 1998]). In this paper, a *scenario* is a *story* (i.e. a textual description) that (partially) describes the future system and its environment from the viewpoint of one stakeholder. Please note that, by definition, a textual description is ambiguous, scenarios are not intended to replace requirements, but act as a carrier for creating a shared vision. However, a textual description helps in *communication* of the vision and thus in generating feedback and improving that vision.

Every scenario visualizes a number of *goals*, concerns of the stakeholders. These goals are the main objectives that the stakeholders have in mind for the system. Note that these objectives typically concern functionality, but also quality attributes of the system to be realized. As an example, one of the stakeholders may aim at a system that "supports at any time the newest version of the PDF standard". The corresponding scenario will state that the system has to be extensible with respect to new versions of this standard. In this example, extensibility is a *quality attribute*: a property of the system by which its quality will be judged by some stakeholders.

Typically, one goal results in a number of requirements. A *requirement* is a SMART[2] demand on the realization of the system.

In order to establish the link between requirement and architecture, we provide the reader with our definition of a *design decision*: a (well-founded) selection of one alternative for the technical realization of the system. Design decisions capture what designing is really about: making choices between alternatives. The sum of all design decisions defines the *architecture*. The architecture is usually captured in a number of views (see [IEEE1471]).

[2] SMART – Specific, Measurable, Accurate, Realistic and Time-bound.

Notice that these definitions fit in the model proposed in [IEEE1471] for the description of an architecture. The vision can be interpreted as a high-level architectural description, since it describes the system, its mission and its environment. The scenarios provide the different views on the same architecture. Each of the scenarios covers a number of viewpoints that belong to one (or more) of the system's stakeholders. Their goals are the concerns that are addressed in these viewpoints.

1.2 Related work on scenarios

From the above domain model follows that we want to follow a scenario-driven approach. Therefore we focus at techniques for scenario-based requirements engineering, and take a look at use-cases, use-case maps, and Goal-oriented Requirements Engineering techniques.

Scenarios are currently also used for evaluation of an architecture, to determine how certain quality attributes are met. We briefly describe this technique here to see how it can help us to steer the design upfront and prevent us from having to re-design parts.

1.2.1 Use-cases and use-case maps

Our definition of a scenario resembles the notion of "use-cases" from [UML]. However, a use-case focuses more on *how* the system behaves in response to a request from a stakeholder, whereas a scenario focuses more on *why* the system behaves that way (i.e., the stakeholders' goals that are to be met). Use-cases can be derived from a scenario.

Use-case maps (UCM, see [UCM]) present causal paths as sets of wiggly lines that enable a person to visualize scenarios threading through a system without the scenarios actually being specified in any detailed way (e.g. with messages). Figure 2 depicts how a UCM fits in the UML language.

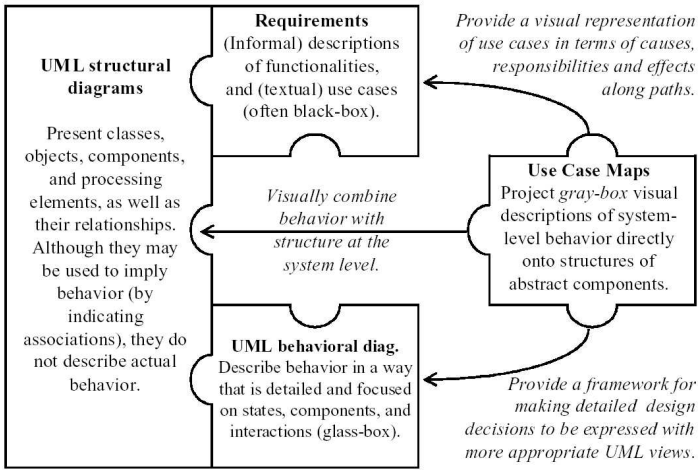

Figure 2: UCMs as missing piece in the UML puzzle (source: [Amyot et al, 2001])

A UCM has clearly a strong relation with scenarios since it is crosscutting the system (more than a use-case) and focuses on one event. However, UCM is a visual language, which leaves more to the imagination of the reader than a textual representation. Worse, it implies a strong focus on the technical realization of the system, which may be less appropriate for communication with (most of) the stakeholders. We feel that this focus on design should be postponed until requirements are known, and different design alternatives can be identified.

1.2.2 Scenario-based evaluation of a software architecture

[Ionita et al, 2002] presents an overview of some scenario-based software architecture evaluation methods: SAAM, ATAM, CBAM, ALMA, and FAAM. In the context of evaluation, a scenario describes a possible modification to the system architecture; the different techniques pursue different goals with scenarios though, e.g., exploring modifiability (SAAM, ATAM, ALMA), exploring costs, benefits and schedule implications (CBAM), or interoperability and extensibility (FAAM). Both

ATAM and the derived CBAM method use the notion of a so-called “utility tree” to make clear which quality attributes are of importance and their relation to scenarios.

As the authors of the paper put it, the classical approach in evaluating software quality is “conformance to requirements”; the scenario-based evaluation methods are shifting the focus of the analysis towards estimating risks and uncertainty associated with the systems’ requirements, architectural decisions and strategies.

As follows from the fact that they are meant for evaluating architectures, these scenarios are very technical and very different from the type of scenarios we have in mind. However, the clear link between scenarios and quality attributes is very helpful, and is incorporated in our vision on scenarios.

1.2.3 Goal oriented Requirements Engineering (GORE): KAOS and GRL

Various papers (e.g., [Lamsweerde, 2001], [Antón et al, 2000], [Yu et al, 1998]) suggest that a goal-oriented approach to requirements is a good one, since goals are more stable than requirements. A goal-oriented approach starts with goals, and derives the requirements from them. The definition of goal matches ours: a goal is the intention of a stakeholder. [Lamsweerde, 2001] lists a number of benefits for a goal-oriented approach, notably:

- Goals provide a precise criterion for sufficient completeness of a requirements specification: in order to cover all concerns of the stakeholders, every goal must have at least one requirement associated.
- Goals provide the rationale for requirements, since they are derived from them.
- Goals provide indirect information on their stability: the higher-level a goal is, the more stable it will be
- Goals drive the identification of requirements to support them.

We would like to add that goals could describe a whole family of products, rather than one specific product for one specific customer. Also, often customers give (their view on) solutions rather than real requirements.

Notice how well the above definition of goals fits in our definition of the relationship between scenarios and requirements. The author defines this relation as follows: scenarios and goals have complementary characteristics; the former are concrete, narrative, procedural, and leave intended properties implicit; the latter are abstract, declarative, and make intended properties explicit.

KAOS (see [Lamsweerde, 2001] and [KAOS]) is one sibling in the family of GORE-methods. It uses a semi-formal notation (or formal, using temporal logic) to define goals that the future system has to achieve. [Lamsweerde, 2001] claims that the KAOS method has been applied with success in 11 industrial projects.

However, in spite of its formal methods, KAOS seems to allow refinements to be made in many ways, probably with very different results. We did not find any guidelines on how to perform the best derivation. In combination with the amount of effort necessary to learn formal methods, we do not find the formal approach helpful.

Another goal-oriented approach is provided by GRL ([GRL]). It adds correlation between goals to the model: some activities influence multiple goals. It is interesting to notice that [Amyot et al, 2002] combines GRL and UCM into a proposal for a new standard for representation of requirements for future telecommunication systems and services, called User Requirements Notation (URN). The goal-oriented techniques from GRL are used to capture non-functional requirements, and the UCMs capture operational and functional requirements.

1.2.4 The Customer Objectives View in the CAFCR method

[Muller, 2003] presents a similar approach to requirements as part of a bigger architecting method called “CAFCR”. The author states that a better understanding of the customer’s needs enables the design of a better product (notice that goals are here not pursued for their stability!). Ideally, the trade-offs in the customer domain become clear. For instance what is the trade-off between performance and cost, or size and performance, or size and cost? The key driver method articulates the essence of the customer needs in a limited set of drivers.

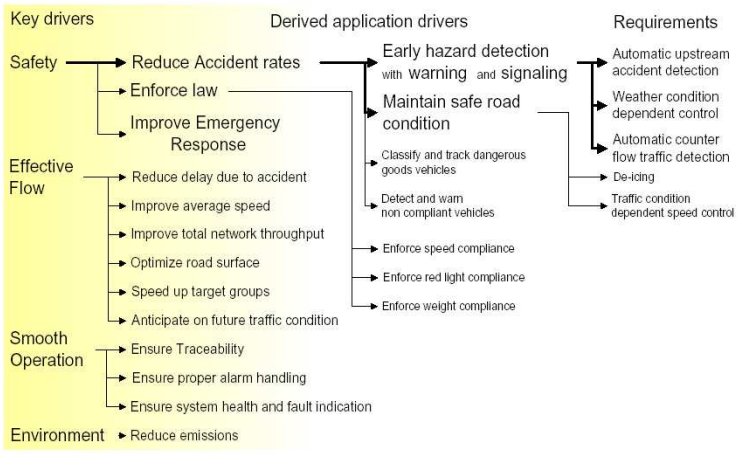

Figure 3: Example motorway management system key drivers (source: [Muller, 2003]).

The author formulates requirements in a number of steps (see Figure 3):

1) Stakeholders have so-called key-drivers (or: goals) that reflect the essence of their objectives. Examples are safety and care for the environment.
2) The key-drivers can be decomposed into high-level requirements to the system that describe which activities are undertaken to enable the key-driver.
3) Finally, the high-level requirements are decomposed into requirements to the product.

In addition to the goal-oriented approach by means of key-drivers, the author suggests to use scenarios for the creation of a shared vision on the product and thus a product definition.

1.3 Approach

As pointed out in Section 1.2.3, GORE promises to deliver stable requirements, good traceability by means of the derivation and a clear rationale for every requirement. In our case study, presented in Section 2, we explore how a goal-oriented approach can aid in the design of a framework. We focus on the quality aspect maintainability, since this is harder to establish than e.g., performance.

We found that GORE indeed does offer good traceability and helps through the stepwise refinement. However, we have our doubts about its scalability since even a small number of goals “explodes” into a tree with dozens of nodes; the graphical notation has both advantages and disadvantages. Worse, we did not find many guidelines on how to conduct the refinement steps. This makes the stability of requirements questionable, since stability is dependent on the definition of the requirements. In spite of these shortcomings, we found GORE, in combination with scenarios, a good approach to understanding the requirements that define the mission-critical characteristics of the system to be realized.

2 Case study: a framework for prototyping

In our case study we consider the construction of a prototyping framework on one of Océ’s products: a controller for a family multi-functionals (a combination of printer, scanner and copier). Researchers from Océ have many ideas for new applications on this controller, but are confronted with the need to become familiar with many concepts before they can use the controller for their purposes. Additionally, prototypes typically need a considerable amount of engineering before they are turned into a real product.

The case study is a 1.5 man-year equivalent project that aims at creating a prototyping framework for the controller. This framework must capture the essentials of the controller that allow building good prototypes.

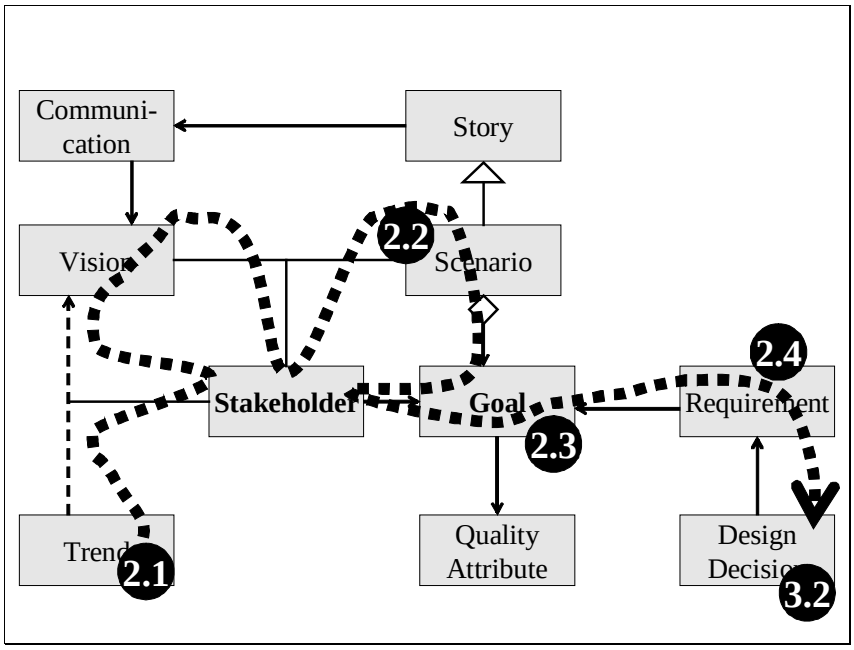

Figure 4: Mapping of steps on the domain model (see also Figure 1); numbers indicate Sections below.

A framework is a set of semi-complete applications that can be specialized to produce custom applications. As suggested in [Morisio et al, 2002], a framework has a positive influence on both productivity and quality of the software constructed from it. However, building a good framework requires a thorough understanding of the domain and how the framework will be used. We explore those by means of scenarios.

As depicted in Figure 4, we start by exploring relevant trends that are likely to show up in prototypes. Together with the stakeholders, we selected the relevant ones, created a vision and demonstrated that vision in a number of scenarios. Analysis of the scenarios yields a number of goals that are validated with the stakeholders. They are the input for the definition of requirements and subsequent design decisions.

Please note the frequent involvement of stakeholders in the trajectory that leads to goals. Following Sections elaborate on the steps taken.

2.1 Observe relevant trends

The first step in the architecting process is determining, together with the stakeholders, what trends are relevant for the controller. In the case of our framework, we observed a number of technical trends that will be interesting to Océ and are thus likely to be prototyped. Prototyping will have to be fast in order to test many alternative ideas in a short time, and will be achieved through various tools.

2.2 Create a shared vision by means of scenarios

The resulting vision on the framework is described in a number of scenarios, at least one for each of the stakeholders. One scenario describes how a researcher realizes his ideas about a networked application with the help of our framework. It describes how he glues pieces of functionality – both from the framework and from other sources – together with 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 underlying product has changed its interface.

Apart from their role as source for the next steps in the architecting method, the scenarios have proven to be very effective in introducing our case study to new people, i.e. in creating a shared vision. All reviewers of our documents were supplied with a copy to provide them with the background information they need to judge on the decisions taken.

2.3 Determine the stakeholders' goals

In the analysis of the vision, we explore the added value of our product: it explains the importance of prototyping for creating new products and for generating feedback on the current architecture. This helps in building commitment amongst the stakeholders.

Further, we analyzed the main challenges, and explored enabling technology as a first draft of a feasibility study.

The scenarios that describe the use of the framework provide a good starting point for definition of use-cases as defined by [UML].

The most important part of the analysis aims at identifying the stakeholders' goals. In this analysis, we answer the question *why* the scenarios are the way they are by making the underlying objectives explicit. In our case-study, we identify each of the three stakeholders by a role name and discuss their goals and associated quality attributes (from [ISO9126-1]):

1) Researcher is the researcher who builds prototypes; he wants prototyping on the controller to be easier. This brings new functionality and ideas closer to development. The framework must enable researchers to focus on functionality instead of how this functionality is incorporated in the controller's workflow. Following quality attributes are applicable:

    a) Functionality and more specifically:
        i) Suitability: provide appropriate functions for researcher's needs / objectives
        ii) Interoperability: how does it interact with the controller and applications?

    b) Usability and more specifically:
        i) Understandability: enable user to understand how the framework can be used
        ii) Learnability: enable user to learn bridge's application

2) Maintainer is a researcher who has to maintain the framework (for new prototypes); he wants maintenance effort to be minimal. Nobody has time for extensive maintenance activities; each researcher building a prototype will probably do a little maintenance. The framework must have maximal de-coupling from the controller, concise interfaces, and good documentation. Following quality attributes are applicable:

    a) Maintainability, and more specifically:
        i) Analyzability: capability of the framework to be diagnosed for deficiencies or for parts to be modified to be identified
        ii) Changeability: capability to enable a specified modification to be implemented
        iii) Stability: capability to avoid unexpected effects of modifications

    b) Portability, and more specifically:
        i) Adaptability: capability to be adapted for different environment
        ii) Installability: capability to be installed in certain environment

3) Engineer is someone from the Engineering department; he wants prototypes to stay close to the controller architecture. This facilitates easier transition of ideas to development. The framework must adopt the main architectural concepts from the controller.

2.4 Decompose goals into requirements

Now that we have identified goals, we can derive subgoals[3] and requirements from them.

If we look for example at the goal for Maintainer, we can split it into the two main causes for maintenance: maintenance because of adding new functionality that is already present in the current version of the controller (the "maintainability" quality), or maintenance because a new version of the controller is released (the "portability" quality if we consider the controller to be a platform).

Then we can zoom in on the analyzability aspect of maintainability and we come to requirements on logging events inside the framework. The changeability aspect of maintenance and the adaptability aspect of portability are made concrete by means of change-cases to be realized in a certain amount of effort, e.g. "Offer proper support for moving a job (e.g., a print job) from one controller to another,

[3] Subgoals are goals that are more specific than the (sub)goals they are derived from, but still not precise enough to be requirements (see Section 1.1).

independent of the (difference in) releases." Notice that these change cases are similar to the scenarios that are employed for evaluation of an architecture (see Section 1.2.2).

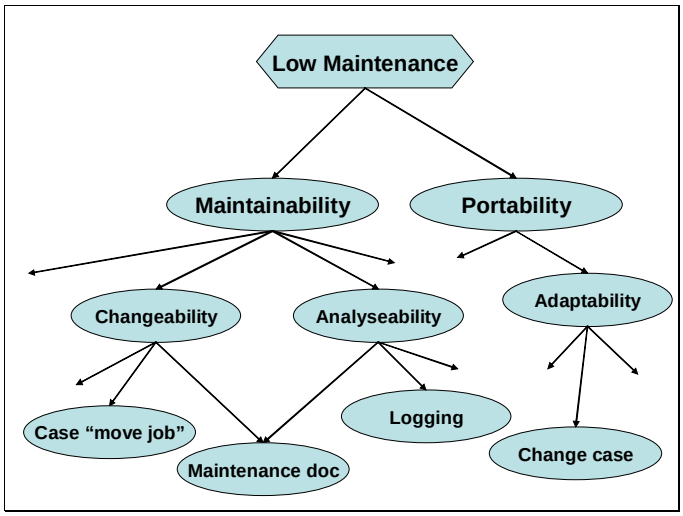

Figure 5: Graphical representation of the decomposition of goal (in hexagon); this fragment reflects the decomposition of the goal "Low Maintenance".

As in the KAOS and GRL methods, we tried a graphical notation, see Figure 5. This provides a bird's eye overview of the relationships between goals, subgoals and requirements: deriving, sharing (the Maintenance doc), or conflicting (not shown). Each of these relationships must be documented with a rationale, e.g. "a maintenance document helps in determining where changes have to be made."

The decomposition stops when we have reached requirements: measurable and testable demands on the realization of the framework.

2.5 Dealing with imperfection: an iterative approach

In spite of a thorough analysis, goals may not be formulated perfectly. During the analysis of a design, new information arises on the requirements: some may be infeasible, others are simply missing. The above examples are taken from the second iteration.

In our case study, the boundaries of the problem were unknown: researchers have lots of wishes, but no one knows what is really possible. Instead of spending a lot of time on the requirements, we decided to start out with very thin scenarios that merely sketched the goals. In order to get more information on the feasibility of certain directions, we used one week to explore the main characteristics of different architectures based on this data. Upon presenting several alternatives to the stakeholders, additional requirements became visible: certain solutions had a clear preference because of reasons that were not mentioned before.

One thing that became obvious was that the framework should reflect the product's architecture to enable easier transfer from the research teams to the engineering teams. We added this stakeholder to our analysis and captured his goals in a new scenario that describes how the framework helps the engineer to transform a prototype into a product. One of the goals identified originally became obsolete since our analysis showed that it was not feasible to realize this. We also added a list of research ideas that should be feasible with the framework.

Eventually, we reformulated all goals into the three high-level goals shown in Section 2.3 and derived a new tree from them. Since we had more information on relevant quality attributes, splitting could be steered on the identified quality attributes. They were subdivided according to [ISO9126-1] which identifies subcategories for the six main quality attributes. The standard helped as a checklist[4] in covering the relevant areas, but unfortunately it does not provide much help in defining metrics.

We include Figure 6 to give you an impression of the difference between the information in the first and second iterations. The difference between the two trees can be attributed to a number of factors:

[4] For more checklists of quality attributes, see e.g. [Muller, 2003].

- We had new insight in the goals of our stakeholders, reflected in newly defined goals on a higher level. Changing the roots of the graph obviously has quite some impact.
- We re-used 12 out of 39 nodes from the first iteration (mainly on functional requirements), but some of them were decomposed according to new criteria; this gives the false impression that the second graph presents a whole different set of requirements.

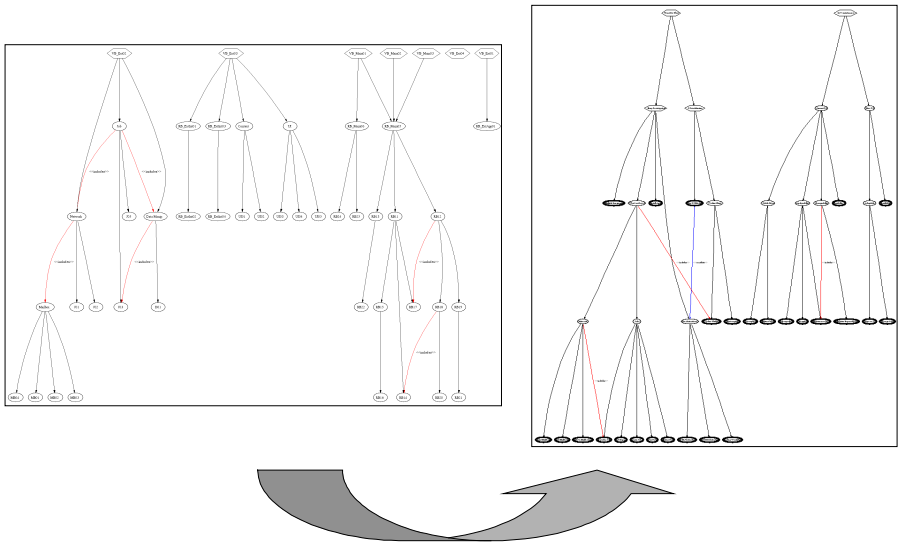

Figure 6: Decomposition graphs in the first and second iterations.

2.6 Heuristics applied

As already hinted by the previous sections, we do not employ any formal rules in the decomposition of goals since none could be found in literature. We realize that this makes the end-result very dependent on (the knowledge of) the person performing the GORE. Figure 6 serves as a tangible proof of this statement. We also realize that it is very hard to provide a set of rules that is applicable in every application domain.

Instead, we applied a number of heuristics. Most gaps were filled by asking a question that starts with "how" ("How to make changing easy?") or "what/which" ("Which elements must be re-usable?"). Sometimes we made an abstraction triggered by a "why" question, as shown in the first step of the case study (Section 2.4). This corresponds to heuristics provided in [KAOS] and [Muller, 2003].

We tried using checklists like [ISO9126-1] to split quality attributes into smaller concerns, but found that they still leave a big gap to be filled. However, in the second iteration it helped us ensuring that all relevant quality attributes were covered by requirements.

On the topic of maintainability, we found that change cases help narrowing the scope: "the system is maintainable enough if we can perform change-case 'port to new version' within two days".

2.7 Tool support

In order to facilitate the method, we have an XML-file that acts as a database containing a description of the (sub)goals, requirements and their relationships. When applicable, relationships are attributed with an explanation on how the refinement was made (e.g., by noticing that there is an asymmetric relationship, which causes two options to be specified independently).

Requirements and goals are specified according to the template presented in [Volere, 2002], which includes a rationale for every requirement, a reference to related quality attributes, and tests to verify that the requirement is met. We identify different types of relationships: most relationships indicate a

refinement, some are inclusions (when a subgoal/requirement also contributes to another subgoal), and some goals may be conflicting. The latter category should be carefully examined to avoid ending up with an infeasible set of requirements.

We created a simple tool[5] to transforms the XML specification into various useful representations:

- A PDF file for easier reading
- An HTML file that can be embedded in the hyperlinked architecture documentation (see also Section Beyond requirements)
- A graphical representation of the refinement steps; the number of refinements gives a hint on the level of specificity, the balance in the graph may indicate that some goals have not yet received enough attention.
- Statistics on coverage, number of tests, etc.

We feel that tool support becomes even more important when the set of requirements grows, and (sub)goals may change.

3 Conclusion and discussion

We started this article with two questions: (1) how to get a set of requirements in the first place, and (2) how to make it a stable set. Scenarios have proven to be a strong tool in visualizing the future product and thus the requirements that describe its characteristics. Especially the mission-critical characteristics are emphasized in scenarios, since they focus on the essentials. These are also the characteristics that are likely to hold for a whole product family, rather than only one product.

By means of scenarios, we explored the goals of the system's stakeholders. Deriving requirements from these goals, the relationship between requirements and stakeholders became very apparent. Stability of these requirements depends however not only on the corresponding goals – which indeed are stable – but also on the *definition* of subgoals. If these are changed, the new information ripples though the subtree… with consequences for the requirements. Our second iteration makes this apparent. Therefore we feel that tool-support is necessary to keep track of the – probably large – number of subgoals and requirements.

The introduction of intermediate subgoals contributes to traceability and coverage, but introduces a concern about scalability. Especially the graphical representation becomes unreadable very quickly (cf. Figure 6). The notion of subgoals introduces the option of a hierarchy of requirements: a top-level diagram may stop at the level of subgoals ("requirements") for components. More detailed diagrams can be derived for those subcomponents, with the subgoal from the top-level diagram as a goal in the detailed diagram.

This solution improves the readability of diagrams, but does not alter the number of items to be maintained. Tool support will become necessary to track the consequences of changes and to maintain a consistent set. However, we do believe that the burden posed by subgoals pays off in terms of better requirements and more satisfaction from stakeholders; industrial experiences with KAOS (see Section 1.2.3) strengthen us in this belief.

3.1 Challenges encountered

We realize that the choices made during refinement steps have an impact on the resulting set of requirements. However, we can only provide a small set of *heuristics* (see Section 2.6) on how to decompose, since they depend on the domain and the knowledge of the person applying GORE. We would like to point out that checklist can help in the refinement, e.g., the refinement of quality attributes can be guided by subdivisions as made in [ISO9126-1]. We urge the reader to document every step taken, thus yielding a set of requirements that can be traced to their originating goals and other sources. The use of change-cases for defining maintainability urges to use scenario-based evaluation methods (see Section 1.2.2) as early as possible.

The graphical representation of the requirements by means of the derivation tree gives quick feedback on the level of elaboration. Even if this is only a hint (some goals may need more elaboration than

[5] This can be achieved by an XSLT-parser, a tool to transform an XML specification into a document (be it XML, HTML, or an input file for a graph visualizer). Tools like Telelogic's DOORS also provide information on traceability.

others), it invites to examine the set of requirements as a whole, rather than only one-by-one. One should be careful in attributing too much meaning to the graphical representation though. As was shown in the case study, the two iterations yielded very different trees; a first glance gives the false impression that they describe two completely different sets of requirements.

The set of scenarios provides a powerful communication tool with all stakeholders. It sketches how their concerns will be addressed and helps in focussing on the added value of the project. As mentioned before, the scenarios do not contain enough information to construct the requirements. Therefore we conducted a number of additional interviews to explore the meaning of the goal. This demonstrates how the goals and subgoals were used as driver for the identification of requirements. Completeness is an ideal that will never be achieved, but proper decomposition from the goals ensures that the set of requirements is complete enough.

3.2 Beyond requirements: the link to design

Requirements are choices that pertain to concepts in the problem domain: transaction speed, number of copies, the reaction that follows pressing the green button, etc. Design decisions are also choices, but pertaining to the concepts in the solution domain: network bandwidth, certain library-interfaces, etc. Both types of choices limit the realization of the system by specifying what should be provided and how.

Most refinement steps were made by asking the question "how". If continued long enough, we cross the border between problem domain and solution domain: "How can we support multiple operating systems? Either by programming on a virtual machine (e.g., the Java Virtual Machine), or by creating a portable abstraction layer for the OS, or by …"

Clearly, design decisions address requirements. If collected, these partial solutions may help finding an overall architecture that solves the requirements. This hints that the techniques applied in GORE help in designing as well. However, it is important to realize that combining these partial solutions into one product is far from trivial… and is steered by the higher-level goals. This leads us to shifting our attention from requirements to goals.

3.3 Goals instead of requirements?

In Figure 4 we showed how our stakeholders were involved until their goals were known, and not in the formulation of requirements. Although this seems surprising, we feel that it reflects the current practice. Stakeholders tend to think in terms of scenarios to be realized by the system, not in lists of detailed requirements.

Scenarios show how the system's vital features are employed to achieve a stakeholder's goals. Requirements on the other hand, tend to have a technical focus that zooms in on the small parts. This is important for the realization of the system, but introduces a big risk of loosing the context and thus understanding of how each of the requirements contributes to the system.

In our case-study, the derivation of requirements served more as a tool to understanding the stakeholders, than as an approach to construct a contract with them. Understanding the rationale behind requirements helps in taking technical decisions that eventually lead to a product that is accepted. 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 directly linked to goals.

Acknowledgements

We would like to thank following people for their help through discussions, questions and reviews: Marvin Brünner, Bas Graaf, Dieter Hammer, Jan Jacobs, Marco Lormans, Päivi Parviainen, Anu Purhonen, Michiel van Osch, Lou Somers, and Cees-Jan Sonnenberg.

This paper results from an industrial project at Océ Technologies B.V. as part of the MOOSE project ([MOOSE]). It is published in the MOOSE project as deliverable 2.6.3.

References

[Amyot et al, 2001] *Bridging the Requirements/Design Gap in Dynamic Systems with Use Case Maps (UCMs),*

Daniel Amyot, Gunter Mussbacher,
Tutorial in: 23rd International Conference on Software Engineering (ICSE'01),
Toronto, Canada, May 2001.
(also available as http://www.usecasemaps.org/pub/icse01.pdf)

[Amyot et al, 2002] *URN: Towards a New Standard for the Visual Description of Requirements*,
Daniel Amyot, Gunter Mussbacher,
http://www.usecasemaps.org/pub/sam02-URN.pdf

[Antón et al, 2000] *Managing Use Cases During Goal-Driven Requirements Engineering: Challenges Encountered and Lessons Learned*
Annie I. Antón, John H. Dempster, Devon F. Siege
http://www.csc.ncsu.edu/faculty/anton/pubs/icse2000.pdf

[GRL] *Goal Oriented Requiements Language*,
http://www.cs.toronto.edu/km/GRL/

[IEEE1471] *IEEE Recommended Practice for Architectural Description*,
IEEE Computer Society,
IEEE Std 1471 - 2000, October 2000

[Ionita et al, 2002] *Scenario-Based Software Architecture Evaluation Methods: An Overview*,
M. Ionita, D. Hammer, Henk Obbink,
Proc. Int. Conf. on SW Engineering, Orlando (FA), May 2002,
(also available as http://www.win.tue.nl/oas/architecting/aimes/papers/Scenario-Based%20SWA%20Evaluation%20Methods.pdf)

[ISO9126-1] *Software Engineering – Product Quality – Part 1: Quality Model*
ISO Committee,
ISO/IEC 9126-1, 2001

[Jarke et al, 1998] *Dagstuhl Workshop on Scenario Management*,
M. Jarke, X. Tung Bui, J. Carroll,
February 9-13, 1998

[KAOS] http://www.info.ucl.ac.be/research/projects/AVL/ReqEng.html

[Lamsweerde, 2001] *Goal-Oriented Requirements Engineering: A Guided Tour*,
C. van Lamsweerde,
Proceedings RE'01, 5th IEEE International Symposium on Requirements Engineering,
Toronto, August 2001, pp. 249-263

[MOOSE] *Project Moose*
http://www.mooseproject.org

[Morisio et al, 2002] *Quality, Productivity, and Learning in Framework-Based Development: An Exploratory Case Study*,
M. Morisio, D. Romano, I. Stamelos,
IEEE Transactions on Software Engineering, Vol.28, No.9, September 2002
pages 876 – 888

[Muller, 2003] *Architectural reasoning; balancing genericity and Specificity*
Gerrit Muller, Embedded Systems Institute, Eindhoven, 2003
http://www.extra.research.philips.com/natlab/sysarch/ArchitecturalReasoningBook.pdf

[OOTI] *The OOTI Program*
http://wwwooti.win.tue.nl

[UCM] http://www.usecasemaps.org/index.shtml

[UML] *The Unified Modeling Language Reference Manual*,
J. Rumbaugh, I. Jacobson, G. Booch,
Addison-Wesley, 1999

[Volere, 2002] *Volere Requirements Specification Template (Edition 8)*,
J. Robertson, S. Robertson,
The Atlantic Systems Guild, 2002

[VTT, 2002] *Quality-driven architecture design and quality analysis method*
Mari Matinlassi, Eila Niemelä, and Liliana Dobrica
VTT Publications, 2002, ISBN 951-38-5968-1
(available at http://www.inf.vtt.fi/pdf/)

[Yu et al, 1998] *Why Goal-Oriented Requirements Engineering*
E. Yu and J. Mylopoulos,

Proceedings of the 4th International Workshop on Requirements Engineering: Foundations of Software Quality (8-9 June 1998, Pisa, Italy). E. Dubois, A.L. Opdahl, K. Pohl, eds. Presses Universitaires de Namur, 1998. pp. 15-22