Exactly the Information your Subcontractor Needs: 
**DeSyRe — Decomposing System Requirements**

Birgit Penzenstadler  
Fakultät für Informatik  
Technische Universität München  
Munich, Germany  
penzenst@in.tum.de

**Abstract**—In software systems development, the increasing size and complexity of systems is handled by decomposition. Companies additionally sign up different subcontractors for subsystems. For distributed development and smooth integration, a major challenge is to deduce subsystem specifications from system specifications in order to deliver them to the subcontractors.

Missing information within the subsystem requirements is the pitfall for successful distributed development, so that either the subsystem requirements do not fulfill the overall system requirements completely, or there is a mismatch between subsystems during integration due to inconsistencies between the specifications for the respective subsystems. The objective is to find a good way for the requirements engineer to deduce subsystem requirements from system requirements.

The contribution of this paper is an approach to systematically derive subsystem requirements from system requirements by use of assumption / guarantee reasoning and decomposition patterns. Thereby, the approach ensures completeness of the subsystem requirements with respect to the given system requirements. The approach is evaluated by means of case studies concerning applicability and usefulness.

**Keywords**—Requirements, Subsystem, Decomposition, Refinement, Subcontractor

I. **INTRODUCTION**

In systems development, which includes hardware and software, systems are growing, not only in terms of lines of code, but also in complexity, degree of heterogeneity, number of peripheral devices etc. To handle the increasing complexity of systems, counter measures are required.

One central counter measure is the decomposition into subsystems. Thereby, companies try to master the increasing size and costs of their systems by concentrating on their core competences and signing up different subcontractors for the development of subsystems. First, a requirements specification is elaborated for the system, second, an initial decomposition is decided on in order to be able to develop the subsystems at different sites and, third, respective subsystem requirements specifications have to be deduced. For distributed development and smooth integration, a major challenge is the appropriate deduction of subsystem requirements specifications from system requirements specifications in order to deliver them to the subcontractors. Figure 1 illustrates how requirements (right hand side) shall be decomposed according to a given system decomposition (left hand side). One obstacle for successful distributed development is missing information within the subsystem requirements. Consequently, either the subsystem requirements do not fulfill the overall system requirements completely, or there is a mismatch between subsystems during integration due to inconsistencies in the specifications. Furthermore, systematic derivation of subsystem requirements and their traceability are especially relevant when the system requirements change, in order to identify the respective subsystem requirements for adaptation.

Currently, there is no encompassing approach in the literature that provides guidance to the systematic decomposition and refinement of the requirements to avoid such loss of information or inconsistency.

Therefore, this paper presents an approach how a requirements engineer can systematically derive subsystem requirements specifications from system requirements specifications for a given system decomposition.\(^1\) For systematic

---

\(^1\) We assume that the system decomposition is developed by the system’s architect, and the requirements engineer is then in charge of decomposing the requirements in the appropriate way.
deduction of subsystem requirements, we investigate which cases have to be differentiated as well as which rules and decomposition patterns can be identified. The patterns serve as guiding reference for a requirements engineer in order to complete his task of deducing subsystem requirements, i.e. they support the requirements engineer but cannot perform the task automatically.

Outline: Section II gives an overview of related work, Sec. III presents the approach, Sec. IV describes the evaluation, and Sec. V summarises and gives an outlook on future work.

II. PREVIOUS WORK

Several works exist in the areas of requirements patterns, requirements decomposition, and assumption / guarantee specifications.

A. REQUIREMENTS PATTERNS

The most recognized work on requirements patterns is the pattern system by Dwyer et al. [1] as these patterns have been found in an empirical evaluation of over 550 specifications. The patterns are classified according to the system behaviors they describe in terms of occurrence and order of actions. Smith et al. [2] used these patterns in a “disciplined natural language representation” (by means of templates with lists of alternative phrases) to specify commonly-occurring properties with the aim of eliciting “precise and rigorous requirements from people who are unlikely to be fluent in temporal logics or other specification formalisms” [2, p. 12].

The four basic patterns are: Response, Precedence, Existence, and Absence [2, p. 14]. These patterns are helpful to understand the content of the requirements and proceed in the direction of formalization, but they do not provide a structure to decompose requirements for given subsystems.

B. REQUIREMENTS DECOMPOSITION

Lamsweerde [3] uses goal refinement to make the transition from abstract or high level goals to concrete system goals and analyze their interrelation. Apart from that, there are principles for system decomposition, but no specific guidance is given as to how to decompose system requirements into subsystem requirements according to a given system decomposition. Our industrial partners report that they do not have methodical background with respect to the topic, especially no systematic way for decomposition.

C. ASSUMPTION/GUARANTEE

When decomposing and refining a requirement, it is necessary to ensure that the resulting requirements for subsystems do indeed guarantee the overall system requirement. Systems are designed for environments that satisfy certain assumptions. If those assumptions are fulfilled, the systems commit themselves to certain guarantees. The same idea can be applied to requirements by using assumption/guarantee (A/G) specifications with logical implications. The first formal method based on A/G specifications that received wide attention was the pre/postcondition style of Floyd/Hoare logic [4], which provided the foundation for further methods. Assumption/guarantee specifications consist of two parts: an assumption and a guarantee. The assumption describes properties of the environment in which the specified component is supposed to run. The guarantee describes what the component has to fulfill in case the assumption is satisfied by the environment [5, Chap. 12]. Denoted as formula in Eq. 1, the assumption implies the guarantee.

\[ A(input, output) \Rightarrow G(input, output) \]  

A/G specifications are used by Abadi and Lamport [6] for conjoining specifications in their composition theorem to prove that a lower-level specification implies a higher-level one. Henzinger et al. [7] use assume-guarantee reasoning and refinement mappings on a formal system description language of reactive modules for verification purposes. In [8], the authors add an assume-guarantee rule for checking simulation. The specification language is defined formally (the system description language of reactive modules), which is not the case for the requirements specification of this approach. However, the idea of assume-guarantee reasoning relies on plain logic, which is also applicable to statements in natural language.

The challenge when refining the system requirements for a subsystem is to ensure that all relevant information is present but no overhead information is dragged along. Therefore, when a requirement is represented as guarantee and the constraints under which the requirement needs to hold are represented as assumption, exactly the constraints which are relevant to fulfill the requirements are present within the assumptions [5, p. 213]. Consequently, a requirements specification in A/G style allows to extract the relevant requirements for a specific subsystem. In comparison to other specification styles, it is at the same time the least restricting one considering natural language input.

III. REQUIREMENTS DECOMPOSITION

In this section, we first give an overview of the steps that are taken for decomposition, then present the patterns that are used, and discuss the alternatives for non-functional requirements. Illustration is provided by a running example from the automotive domain.

Running example: The driver assistance systems (DAS) are a real-life example from automotive industry and served to evaluate the applicability of the approach. From the DAS, three example subsystems are detailed, namely the adaptive cruise control (ACC), the radio frequency warning (RFW) system, and the navigation system. The ACC is a speed control system that automatically maintains a pre-defined speed taking into account a minimum distance to the car.
in front. The RFW supports the driver in coping with the information input from the surrounding environment by use of radio frequency signals.

Overview: We describe how to decompose and refine system requirements into subsystem requirements with a stepwise approach (Sec. III-A) that uses decomposition patterns as reference (Sec. III-B). Its application for different types of requirements is discussed in Sec. III-C.

A. Steps to Subsystem Requirements

Figure 2. Steps for Requirements Decomposition and Refinement.

The steps to take for refining and decomposing a system requirements specification are depicted in Fig. 2. Refinement means enriching the requirement with more detailed information, while decomposition means disassembling the requirement into two or more separate requirements. The process input are the [System requirements] specified in A/G style, which may be informal, e.g., natural language. The transition from system requirements to subsystem requirements usually includes both a decomposition and a refinement of the system requirements. For each system requirement, the following steps (referenced by [name] in Fig. 2) are performed:

[Analyse Type of Requirement] In order to be able to decompose requirements, they need to contain a sufficient amount of detail. The analysis starts with deciding whether sufficient granularity is already provided in the requirement for the subsequent decision of whether it can be decomposed.

[Can be Decomposed?] Now the requirements engineer decides whether a requirement provides sufficient information to be decomposed straight away\(^2\). Thereby, decomposition requires composability, which can be guaranteed in case of a functional system specification [9]. The answer [no\_decomp] leads to the second decision.

[Can be Refined?] A second [no\_refine] leads to [Adopt Requirement (Constraint)]. In case of requirements that apply to only one subsystem, the system requirement is adopted for the subsystem specification as no refinement is possible. The same applies when the system requirement is, in fact, a constraint. Straight-forward adoption of a requirement causes redundancy, which is usually not wanted within requirements documentation. However, in this case the redundancy is expedient as a subcontractor does not receive the whole system specification but only one particular subsystem specification with just enough information to meet his needs. The answer [yes\_refine] leads to [Refine Requirement]. The requirements engineer enriches the requirement with more information, explicitly only for the purpose of enabling later decomposition\(^3\). Therefore, the criterion is to add bits of information that the subsystems can be related to.

For sufficiently detailed requirements, [yes\_decomp] leads to the step [Match to Requirement Patterns]. In this step, the requirements engineer decides which decomposition is applicable for the requirement at hand using the given System Decomposition and the Pattern Descriptions as reference (details in Sec. III-B).

The final step is to [Decompose and Refine using A/G] according to the closest match within the reference Pattern Descriptions, thereby deducing the [Subsystem Requirements]. The process steps of decomposition in combination with using assumptions and guarantees of the subsystem requirements imply compliance of the overall system requirement. Therefore, an appropriate refinement covers and implies compliance of the system requirement.

A specific system requirement \(c\) that belongs to a requirements specification \(S_C\) is in many cases realized by more than one subsystem, let these subsystems be \(A\) and \(B\) (as in Fig. 1). Consequently, a specific system requirement \(c\) shall be decomposed into subsystem requirements \(a \in S_A\) and \(b \in S_B\) such that, in the case of compliance of the subsystem requirements, the compliance of the overall system requirement is ensured.

\[
\bigwedge_{a \in S_A} a \land \bigwedge_{b \in S_B} b \Rightarrow \bigwedge_{c \in S_C} c
\]  

(2)

This is expressed by Eq. 2: if all subsystem requirements \(a \in S_A\) for subsystem \(A\) and \(b \in S_B\) for subsystem \(B\) are met, all system requirements for \(c \in S_C\) for system \(C\) are fulfilled. This characteristic is shown for the patterns presented in Sec. III-B.

\(^2\)If this is not the case, the information is first enriched in a subsequent refinement step.

\(^3\)to avoid over-refinement in the sense of over-engineering
The assumption and guarantee for an overall system requirement are denoted as in Eq. 3, using \( i \) as input and \( o \) as output (cf. Eq. 1). In the following, assumptions will be denoted as predicates \( A_X \) with the index indicating the respective (sub-)system specification the assumption is taken for and guarantees will be denoted as predicates \( G_X \) with the index indicating the respective (sub-)system specification the guarantee is given by.

\[
A_C(i, o) \Rightarrow G_C(i, o) 
\]  

(3)

B. Patterns

A single requirement can in many cases be decomposed by using one of the two decomposition patterns subservice and pipeline (bottom of Fig. 3). These terms have been chosen because they resemble principles from the architectural patterns of “main program and subroutines” (here called subservice) and “pipeline” that are described, inter alia, by Vlissides et al. [10].

The two decomposition patterns do already cover a majority of requirements with either a decomposition where the input and the output are produced by different subsystems (e.g. \( i_1 \) and \( o_2 \) in Fig. 3) or a decomposition where input and output are produced by the same subsystem with the subsystem using a second subsystem as service (e.g. \( i_1 \) and \( o_1 \) in Fig. 3).

For the requirements that do not apply to either of the special cases, the generic (third) pattern is used. Decomposition into more than two subsystems is based on a composition of these three patterns. Consequently, these three patterns are all the guidance the requirements engineer needs — two that already cover the majority, and one that is generic enough to cover the rest.

In the following, we exemplarily present the pipeline pattern in more detail, while the subservice pattern and the generic pattern are presented in a reduced manner due to limitations of space.\(^4\)

1) Special Case Pipeline: The pipeline pattern is used in the case when both subsystems provide part of the interface for fulfilling the requirement.

Example: To illustrate the pipeline decomposition pattern, an example requirement from the radio frequency warning system is used. An overall system requirement is: *In case of speeding, the driver shall be warned by the system.* [12]

The interface is described by an input channel \( \text{speed info} \) and an output channel \( \text{display info} \). The system receives the signal from the wheel sensors periodically and detects whether the driver is speeding. In case the vehicle is faster than the allowed speed stored by the system, the driver is warned by use of the display. The assumption and guarantee on the system level are:

\[
\begin{align*}
A_{\text{Sys}} \text{(speed info)}: & \quad \text{The current speed information is available.} \\
G_{\text{Sys}} \text{(speed info, display info)}: & \quad \text{The system displays a warning to the driver if the speed is too high.}
\end{align*}
\]

The relevant subsystems are the RFW Control and the Display Control, depicted in Fig. 4.

\[\text{Figure 3. Overview of Requirements Decomposition Patterns.}\]

\[\text{Figure 4. Pipeline Decomposition of an Example of the RFW System.}\]

Compliance to the overall requirement by the subsystem requirements is assured by the following assumptions and guarantees:

\[
\begin{align*}
A_{\text{RFW}} \text{(speed info)}: & \quad \text{The current speed information is available.} \\
G_{\text{RFW}} \text{(speed info, RFW info)}: & \quad \text{The system sends information about legality of the current speed.} \\
A_{\text{Display}} \text{(RFW info)}: & \quad \text{The information about legality of the current speed is available.} \\
G_{\text{Display}} \text{(RFW info, display info)}: & \quad \text{The system displays a warning to the driver in case of speeding.}
\end{align*}
\]

If both guarantees hold, the requirement is satisfied. As one guarantee is the assumption for the next subsystem,\(^4\) For extensive coverage of all three patterns, please refer to [11].
respectively, the satisfaction of the requirement does not depend on further external influences. Hence, the example decomposition for the pipeline pattern works.

General Description Pipeline Pattern: In case of the pipeline decomposition pattern, both subsystems provide part of the interface for fulfilling the requirement, as depicted in Fig. 5.

![Figure 5. Pipeline Decomposition Pattern.](image)

For the overall system $C$ with communication channels $i$ and $o$ (Fig. 5), the guarantee $G_C$ with respect to output $o$ relies on assumption $A_C$ for input $i$, see Eq. 3. The pipeline is formed by the subsystems $A$ and $B$. Subsystem $A$ receives input $i$, produces output $x$, which again is taken as input from subsystem $B$ to produce output $o$. For the pipeline pattern, the decomposition results in the following:

- Subsystem $A$: $A_A(i, x) \Rightarrow G_A(i, x)$
- Subsystem $B$: $A_B(x, o) \Rightarrow G_B(x, o)$
- The guarantee of $A$ implies the assumption of subsystem $B$.

Hence, the composition is a behavioral refinement\(^5\) of the original specification:

$$(A_A(i, x) \Rightarrow G_A(i, x)) \land (A_B(x, o) \Rightarrow G_B(x, o)) \Rightarrow (A_C(i, o) \Rightarrow G_C(i, o))$$

(4)

In other words, the subsystem requirements for $A$ and $B$ comply with the overall system requirement for $C$.

2) Special Case Subservice: The subservice pattern is used in the case when there is only one subsystem that provides the interface of the system and additionally uses other subsystems to fulfill the requirement.

Example: To illustrate the subservice decomposition pattern, consider an example from the navigation system, as depicted in Fig. 6. One system requirement on the usage service level is:

“The system proposes a route from the point of departure to the chosen destination.”

The interface is described by an input channel \textit{query} and an output channel \textit{route proposal}. The navigation system receives a query for a route proposal issued by the driver and proposes an adequate route from the point of departure to the destination.

![Figure 6. Subservice Decomposition of an Example of the Navigation System.](image)

Internally, the service is realized by the \textit{Routing Calculator} who uses the \textit{Data Base} as subservice which delivers up-to-date information about the possible routes between point of departure and destination. If there are valid inputs for the point of departure and the destination, the routing calculator can pose a \textit{request}. If the assumption of the database about request is fulfilled (e.g., has valid parameters), the database can adhere to its guarantee for \textit{data}. This again satisfies the assumption of the routing calculator, so it can give the guarantee to make an appropriate \textit{route proposal}. Therefore the guarantee of the routing calculator is met, which means that compliance to the overall system requirement is assured.

General Description Subservice Pattern: One subsystem interacts at the interface to provide the requested system service while the second subsystem is used as subservice provider by the first one. For the subservice pattern, the decomposition results in the following (depicted bottom right in Fig. 3, internal channels are named $x_1$ and $x_2$):

- Subsystem $A$: $A_A(i, x_2, x_1, o) \Rightarrow G_A(i, x_2, x_1, o)$
- Subsystem $B$: $A_B(x_1, x_2) \Rightarrow G_B(x_1, x_2)$
- The subsystems’ guarantees mutually satisfy each other’s assumption about the internal channels $x_1$ and $x_2$.

Therefore, the composition of $A$ and $B$ is:

$$(A_A(i, x_2, x_1, o) \Rightarrow G_A(i, x_2, x_1, o)) \land (A_B(x_1, x_2) \Rightarrow G_B(x_1, x_2))$$

(5)

Equation 5 is a behavioral refinement of Eq. 3, therefore Eq. 5 implies Eq. 3. By this refinement, the system requirement is appropriately decomposed and refined.

3) Generic Case: The most generic case for a subsystem decomposition is that the service a system offers is provided by both subsystems with both of them providing part of the interface.

Example: For the generic decomposition case, we use an example from the adaptive cruise control (ACC) system. The \textit{ACCSystem} allows the driver to set a specific speed

\(^\text{5}\) As defined in [5, p. 241], a behavioral refinement relates specifications of the same syntactic interface, where the refined specification may impose further requirements.
that the vehicle automatically maintains. The given guarantee $G_{Sys}$ is that the output display delivers feedback for the driver according to the request set speed and the output adapt speed sends commands to the motor for adapting the speed. The system is depicted in Fig. 7.

The subsystems are the Motor ECU and the ACC ECU. The Motor ECU calculates the current speed from the revolutions and the excess information is checked whether it is necessary to adapt speed is provided. The information about excess speed is delivered by ACC after comparing the current speed to the set speed and the feedback is delivered to the driver via display. This complies with the overall ACC System guarantee $G_{Sys}$.

General Description Generic Pattern: Both subsystems interact with the surrounding system and provide a part of the interface, as depicted in the white box view of the system C in the middle of Fig. 3. For the decomposition, part of the input for system $C$ goes to subsystem $A$ via communication channel $i_1$ and part of the input goes to subsystem $B$ via communication channel $i_2$. The output is provided in part by subsystem $A$ via communication channel $o_1$ and in part by subsystem $B$ via communication channel $o_2$. The subsystems interact via communication channels $x_1$ and $x_2$. The assumptions and guarantees for the subsystems $A$ and $B$ are:

- **Subsystem $A$**:
  - Assumptions: $A_A(i_1, x_1, x_2, o_1)$
  - Guarantees: $G_A(i_1, x_1, x_2, o_1)$

- **Subsystem $B$**:
  - Assumptions: $A_B(i_2, x_1, x_2, o_2)$
  - Guarantees: $G_B(i_2, x_1, x_2, o_2)$

Furthermore, the subsystems satisfy each other’s system-intern assumptions about the channels $x_1$ and $x_2$.

The composition is a behavioral refinement of Eq. 3:

$$
(A_A(i_1, x_1, x_2, o_1) \Rightarrow G_A(x_1, x_1, i_1, o_1)) \wedge
(A_B(i_2, x_1, x_2, o_2) \Rightarrow G_B(x_1, x_2, i_2, o_2)) \Rightarrow
(A_C(i_1, o_1, i_2, o_2) \Rightarrow G_C(i_1, o_1, i_2, o_2))
$$

(6)

In other words, the decomposition according to the general patterns complies with the overall system specification.

C. Non-functional Requirements

In general, all types of system requirements are treated equally within DeSyRe, no matter whether they are functional or not. The preconditions are that the requirements are refined to a sufficient degree and that composability is given.

Specific quality requirements are often difficult to decompose and refine because it requires the expertise of the requirements engineer to find appropriate test criteria or measures for their validation. However, without sufficient validation criteria provided by the requirements engineer, the system designer cannot check whether his specification of the (sub-)system meets the requirements. If a requirement cannot be decomposed, this is an indicator for a necessary refinement before intending a decomposition for the subsystems. To differentiate between different types of requirements, we refer to Glinz [13], who distinguishes functional requirements, performance requirements, process requirements, and specific quality requirements. Furthermore, we reference the classification by Robertson and Robertson [14] to support intuitive understanding.

1) System Attributes Referenced: Many specific quality requirements [13] are expected to be decomposable and refineable as soon as the responsible system characteristics are identified. For look and feel requirements, cultural and political requirements as well as many usability requirements [14], the rule for decomposition is:

i) Refine the overall system requirement as precisely as possible according to applicable/responsible properties of the system vision.

ii) Decompose the requirement and refine it for the subsystems that exhibit any of these properties.

The DAS example in Tab. I shows, if all affected subsystems refine the requirement accordingly, compliance with the overall system requirement is guaranteed.

2) Supportive Composition Model: For other specific quality requirements or performance requirements [13], the limitation for a guaranteed appropriate decomposition is the composability of the subsystem requirements. For example, for performance requirements, security requirements, some usability requirements, and some legal requirements [14], a decomposition that guarantees compliance with the overall system requirement is only possible with adequate composition models. The hypothesis is that they require additional properties for composition. These additional properties are not necessarily known — for some quality attributes, there may be a model for calculation, for others, the property may include probabilities, for some, it may still remain completely unsolved what the additional property is. The example in Tab. II shows that “losing control of the vehicle” needs to be represented by a complex model that can involve
The user interface has to provide appearance according to the CI standards. Now the requirement can be decomposed and refined for the subsystems, i.e., every subsystem of the decomposition that realizes part of the user interface has to refine the requirement according to its related characteristics.

Example requirement: The product shall conform to the CI standards. This requirement is still high-level and abstract. Before performing a decomposition, it is suggested to refine the requirement for the system characteristics that are relevant for the CI.

Refinement: The constraint can be added to a designated list of constraints for the system. For such specific quality requirements or process requirements [13] that cannot be refined, an explicit constraint list at the beginning of the subsystem requirements specification (including validation means) is recommended. An example for such a constraint is shown in Tab. III.

<table>
<thead>
<tr>
<th>Table III</th>
<th>Example for Constraint Handling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example requirement</td>
<td>The use of micro controllers in battery-monitoring sensors has to include a reference to the patent number by “Mikrochip”. This requirement can only be adapted for the subsystems, but no real decomposition is possible.</td>
</tr>
<tr>
<td>Handling</td>
<td>The constraint can be added to a designated list of constraints for the subsystem.</td>
</tr>
</tbody>
</table>

many parameters while only two of them are used here. It is possible to derive subsystem requirements, but they do not yet fulfill the system requirement completely as “losing control” is still underspecified.

<table>
<thead>
<tr>
<th>Table II</th>
<th>Example for Necessary Composition Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td>The electronic stability control (ESC) interferes when the driver is losing control of the vehicle. This real-time requirement first needs a refinement that defines losing control before it is possible to decompose it for the subsystems.</td>
</tr>
<tr>
<td>Refinement</td>
<td>Losing control is defined by either lateral acceleration greater than x m/s, or slip greater than y % (traction control).6) Thereby, the requirement is turned into a functional one, as losing control can now be measured. This definition allows to decompose the original requirement for the subsystems that are concerned either with lateral acceleration or with slip.</td>
</tr>
</tbody>
</table>
| Decomposition and refinement | • In case of slip greater than y %, traction control has to notify the ESC. 
• In case of lateral acceleration (LA) greater than x m/s, the LA sensor controller has to notify the ESC. 
• The ESC controller gets activated in case of notification by either traction control or LA sensor controller. 

Thereby, all subsystems relevant for the activation of the ESC refine the original requirement and their composition leads to compliance.

composability is still under active research for quality attributes. For example, Pavlich-Mariscal [15] proposes a composable security definition that uses concern-specific modeling languages. For performance requirements, Russell and Zilberstein [16] approach composability by using so-called anytime algorithms that are characterized by a probabilistic description of the quality of results as a function of time.

3) Constraint Handling: Operational and environmental requirements as well as maintainability and support require-

ments, and many legal requirements [14] can often not be refined as they are general constraints for the system. For such specific quality requirements or process requirements [13] that cannot be refined, an explicit constraint list at the beginning of the subsystem requirements specification (including validation means) is recommended. An example for such a constraint is shown in Tab. III.

Table III | Example for Constraint Handling |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Example requirement</td>
<td>The use of micro controllers in battery-monitoring sensors has to include a reference to the patent number by “Mikrochip”. This requirement can only be adapted for the subsystems, but no real decomposition is possible.</td>
</tr>
<tr>
<td>Handling</td>
<td>The constraint can be added to a designated list of constraints for the subsystem.</td>
</tr>
</tbody>
</table>

IV. Evaluation

The evaluation of applicability was performed with a case study in the automotive domain on driver assistance systems. The evaluation of usefulness was performed with a questionnaire filled out by the participants of a tutorial on the method. For the complete documentation of the case studies, please refer to [11].

A. Applicability

The research objective for the case study is defined according to the goal definition template by Wohlin et al. [17]:

1. Analyze the DeSyRe (Decomposition of System Requirements) approach for the purpose of evaluation with respect to the applicability from the point of view of the requirements engineer in the context of driver assistance systems.

2. Design: The driver assistance systems were chosen as case study for the purpose of validating the applicability of the approach presented in the work at hand. As they provide a wide range of different granularities and types of requirements with a sufficient degree of detail, they qualified as suitable for an evaluation of DeSyRe. The initial input for the case study was composed by a number of information sources: General background knowledge about driver assistance systems was gained during research projects with BMW, Daimler, Bosch, and Siemens. The original source document of the requirements specification of the RFW system was elaborated by Daimler AG as illustrative case study within the REMsES project [12]. The original requirements that have been used in the ACC case study are documented in [18].

3. Execution: The case study was performed as action research by the author. I applied the complete DeSyRe method to design and documented the decomposition of driver assistance systems and derived the requirements for
the subsystems radio frequency warner and adaptive cruise control.

Results: The results are a requirements specification for the driver assistance systems and two derived requirements specifications for the subsystems. Examples from the specifications are used throughout the paper as illustrating examples.

Many requirements on the system level (= driver assistance systems) could be decomposed and refined as functional requirements for the subsystems as presented in Sec. III. Others were identified as constraints that could not be decomposed any further but still could be refined for the subsystems, for example as in Tab. IV. The constraint *The driver may not be overly distracted by the DAS* is refined for the ACC by naming potential distractions that may originate from the ACC, and for the RFW the constraint refines and restricts the possible distractions by the RFW.

**Table IV**

<table>
<thead>
<tr>
<th>System Level</th>
<th>The driver may not be overly distracted by the DAS.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subsystem ACC</td>
<td>The driver may not be overly distracted by interference of the ACC. Acceleration or braking shall occur smoothly.</td>
</tr>
<tr>
<td>Subsystem RFW</td>
<td>The warning tone of the RFW system shall be in an adequate volume to be noticed by the driver but not to startle them. The frequency of the displays shall not exceed the cognitive abilities of the driver.</td>
</tr>
</tbody>
</table>

Discussion:

Application of Requirements Refinement: Due to the limited amount of coarse-grained functional requirements, only a limited number of example requirements could be found that can be decomposed and refined to illustrate broader applicability. However, the smaller number of natural language functional requirements (in comparison to traditional natural language requirements specifications as still in practice in the automotive domain) is due to the graphical artifacts used for parts of the case study.

Requirements Decomposition Patterns: The question of which pattern to use depends on the point of view of a specific requirement. For example, for the requirements for the signal queue of the RFW system, the queue can be seen as pipeline between controller and display, or as subservice to either of them. This lies within the preferences of the requirements engineer who performs the decomposition. The important part is that no information gets lost.

Less requirements could be transformed by using the patterns than expected. The application of the patterns requires the possibility to represent them in assumption / guarantee style. This is the case for functional requirements that state details relying on perceivable inputs and outputs.

Feasibility of Requirements: Requirements that cannot be operationalized can only be copied, and therefore are hard to validate. This is the case for high-level requirements like goals that have not been transformed properly into system requirements. These requirements tend to stay vague because if the developer does not know how to satisfy them (e.g., by certain test measures), they cannot be realized. Consequently, they have to be refined sufficiently.

Threat to Validity: The threat to internal validity is an experimenter bias, as the author performed the case study herself. External validity is threatened by specifics of the application domain as the case study was performed only in the embedded systems domain, although the method was successfully applied to a number of small examples from the information systems domain.

Countermeasure for both threats is further validation in a case study performed by industrial software developers. A follow-up case study in a different application domain performed by industrial research partners is also intended, as this would further show domain-independent applicability.

B. Usability

The specific usefulness of the approach developed in the work at hand was evaluated by giving a tutorial for practitioners and, subsequently, asking them to fill out a questionnaire. The research objective was:

Analyze the DeSyRe approach for the purpose of validation with respect to the usefulness from the point of view of the software developer in the context of general software systems development.

Design: The author presented a tutorial on the approach followed by handing out a questionnaire that the audience was asked to fill out straight away. The tutorial was designed as a slide presentation with emphasis on the concepts of system decomposition criteria and requirements refinement and an overview of the guiding process. The questionnaire was designed according to the template by Davis [19]. The following statements could be rated in 6 degrees from *I strongly agree to I strongly disagree*: Using DeSyRe

1) improves the structuredness of requirements engineering for complex systems.
2) improves the completeness of subsystem requirements specifications.
3) improves the traceability of requirements.
4) eases system integration as the approach ensures that all relevant information is captured and processed.
5) improves the reusability of requirements.

Execution: The tutorial was held at a small software development company, the jambit Software Development & Management GmbH, and took about 75 minutes including

7http://www.jambit.com
discussion. The number of participants for the tutorial was 12. The audience took active interest in the tutorial and there was a lively discussion on details of the approach.

The number of filled out questionnaires was 11, as one participant decided he could not make any judgement due to lack of experience. He was a 16-year old apprentice who had just started the job. The other eleven participants were experienced software developers with academic background (diploma or master’s degree in computer science, one in electrical engineering) and between 2 and 10 years of industrial project experience. Of the 11 handed in questionnaires, not all were completely filled out.

Results: The results are given in Tab. V. Thereby, the participants rated between I strongly agree and I strongly disagree with no middle option in order to encourage a concrete decision in favour of or against the approach.

Table V
RESULTS OF THE QUESTIONNAIRE ON PERCEIVED USEFULNESS

<table>
<thead>
<tr>
<th></th>
<th>Strongly agree</th>
<th>Partially agree</th>
<th>Partially disagree</th>
<th>Disagree</th>
<th>Number of Answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Improved Structuredness</td>
<td>1</td>
<td>9</td>
<td>1</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>Improved Completeness</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>Improved Traceability</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>Eased Integration</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>Improved Reusability</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>9</td>
</tr>
</tbody>
</table>

Discussion: Overall, the participants had the impression that the approach provided improvements with respect to all points. However, they estimated the usefulness for their own daily work as rather low because they spend relatively little time on systematic requirements engineering at all. According to their own perception, their projects are developed rather solution-oriented with weak attention to RE. This raises questions with regard to the suitability of the target group.

In contrast, structuredness, completeness, traceability, integration and reusability were all perceived as improving by means of the approach.

With regard to the overall research objective for the approach, the improvement of structuredness is the most emphasized characteristic as the intention was to find a systematic approach for the derivation of subsystem requirements specifications:

How can we systematically derive subsystem requirements specifications from system requirements specifications?

Therefore, the rating of the improvement of structuredness is an indicator for having achieved this objective.

Threats: A threat to internal validity may be the discussions that took place during the tutorial, as the participants may have influenced each other and would have answered differently after participating in the tutorial without sharing their thoughts in between.

The major threat to external validity is that the participants may be plain wrong, since they have not tried the method themselves. The counter measure taken during the tutorial was to use examples from real specifications so the audience could relate them to their experience as most of them have experience in software development for the application domain of the examples, i.e., embedded systems.

Another threat to external validity is that there is no systematic requirements engineering approach established within the company. Therefore, most of the developers have limited knowledge about requirements engineering in general and might not be the most appropriate audience to judge the approach after only a short tutorial. On the other hand, this might actually not harm transferability as requirements engineering in practice and, even more, its education and training, are often neglected in favor of other issues.

Conclusion: The results from the questionnaire on perceived usefulness indicate that the approach achieves the objective of improving the structuredness of results during the process of requirements engineering.

V. SUMMARY AND OUTLOOK

This paper presented an approach how to deduce subsystem requirements from system requirements. It was detailed how requirements decomposition and refinement can be performed using three decomposition patterns, two special cases that already apply for most requirements, and the generic case for the rest of them. Assumption/guarantee style was used in order to be able to show that the composition of the deduced subsystem requirements fulfills the system requirement.

Furthermore, the decomposition and refinement of quality requirements were discussed. These can either be decomposed as soon as the responsible system characteristics are identified, or an additional composition model is needed, or they issue constraints that have to be acknowledged. Thereby, the presented categorization is not strict, there are example requirements that fit into different categories and categories that exhibit example instances for different handling alternatives.

The case studies have shown applicability and perceived usefulness of the approach.

We are currently planning a second case study in the automotive domain, this time performed by industrial developers. Additionally, we are working on the evaluation in a different application domain (health systems). Last but not least, a tool concept is under development for supporting the subsystem requirements deduction within a model-based requirements
engineering tool that will be integrated into the CASE tool AutoFocus [20].

ACKNOWLEDGMENT
The author would like to thank Maximilian Junker and the anonymous reviewers (especially the third reviewer) for helpful feedback on earlier versions of this paper, and Manfred Broy for continued discussion of the topic [21].

REFERENCES