A Context-Driven Use Case Creation Process for Specifying Automotive Driver Assistance Systems

Hannes Omasreiter, Eduard Metzker

DaimlerChrysler AG

Research Information and Communication

Postfach 23 60

89013 Ulm, Germany

{hannes.omasreiter, eduard.metzker}@daimlerchrysler.com

Abstract

Current use case guidelines typically recommend a goal-driven process for use case creation and, in many cases, this approach indeed works sufficiently well. Yet, in our experience, this goal-driven approach does not seem to offer the best fit for every kind of function, e.g. for highly context-sensitive automotive driver assistance systems. For these functions we suggest selecting a context-driven process for use case creation.

In this paper we first show the effectiveness of goal-driven use case creation for a usual car telephone function. Then we demonstrate the limits of this approach for an adaptive vehicle cruise control function (the Distronic system). Finally we present our basic ideas of a context-driven use case creation process and evidence its advantages as compared to the goal-driven approach using the example of the Distronic function.

1. Introduction

Software-based assistance systems for supporting and unburdening the driver have become an increasingly important component of modern premium passenger cars and commercial vehicles. Examples for such assistance technologies are the automatic emergency breaking assistant or the Distronic system, which autonomously keeps a predefined safety distance to vehicles running ahead. More advanced assistance systems such as lane keeping assistants or systems which recognize and react to traffic signs indicate the future of such technologies.

Early, high-level, abstract specification of such systems helps to draw conclusions about basic system properties such as actors and tasks as well as about advanced issues such as general feasibility, critical issues in user interaction, and potential risks.

Hence, use cases have been successfully applied for this purpose during the development of typical application software systems. However, our experience with specifying automotive driver assistance systems has

indicated some limitations of the classical goal-driven use case approaches for the class of systems described above.

In this paper we give a short overview of existing use case creation approaches and useful guidelines. In section 2.2 we illustrate the strength of these goal-driven approaches using the example of a hands-free car telephone system. In section 3 we describe the general limits of this approach for the specification of highly context-dependent systems. We define the special constraints of this class of systems. Based on the properties of this class of software systems, we propose a context-driven approach to the use case creation process. The special advantages of our context-driven approach are illustrated using the Distronic system example.

Section 4 summarizes the constraints that indicate the preferred deployment of a context-driven approach as compared to a goal-driven approach. From our perspective, a comprehensive compilation of use case guidelines could increase the utility and usability of use cases for software engineering practitioners. Such a compilation of use case guidelines should take both the general properties of the system to be developed and the information needs of the development team into account. It could improve the effectiveness of the use case approach for practitioners and indicate open research issues for the scientific community.

2. Existing Guidelines and Goal-Driven Use Case Creation Processes

A wide range of use case formats and guidelines are available for use case creation. We will address a selection of these in section 2.1. A common denominator of most of the guidelines is the goal-driven approach. In the second section of this chapter, we show the essential steps of the goal-driven use case creation process using the example of a car telephone.

2.1 Existing Use Case Guidelines

Traditionally, use cases have been proposed in the software engineering community to provide a mechanism for determining system boundaries as well as for developing a user-oriented requirement model. There appears to be a great deal of commonality between task models and use cases. Use cases and specifications of tasks both describe work, but in different ways. So there are similarities between the two approaches even though sometimes different terms are employed for the same meaning.

According to Cockburn [1], use cases are not well defined, and many different uses coexist. They differ in purpose, content, plurality, and structure. Cockburn proposes that use cases should be structured hierarchically with respect to goals and sub-goals, which limits the scenario explosion. The goals are structured as a tree containing “summary goals” as high-level goals and “user goals” as atomic goals.

Constantine and Lockwood [2] suggest that use case analysts first prepare lightweight use case model descriptions (essential use cases) that do not contain any implicit user interface or technology decisions. Essential use cases focus on user intention rather than interaction details and on simplification rather than elaboration. Later, the user interface designers can exploit these essential use cases as input to create the user interface without being bound by any implicit decisions. Furthermore, Constantine and Lockwood [2] suggest use case maps to structure use cases similar to hierarchical task models.

Aside from the different flavors of use case formats, a large number of guidelines have been developed to support the guideline creation process. Rolland, for example, put forward a set of general guidelines that apply to use cases and any requirements documents in general [3]. The following guidelines (GL) as proposed by Rolland are examples for general guidelines:

GL: Avoid negated sentences.

GL: Avoid synonyms and homonyms.

Denger [4] compiled a set of guidelines for use cases for embedded systems. They range from very informal guidelines such as

GL: Tasks should pass the coffee test, i.e. the actor has deserved a cup of coffee after finishing a task [5].

to more concrete guidelines such as

GL: Use complex variables. For outputs, show the actor's intention, not the detailed system reaction [6]

Guidelines like the above have been employed to optimize the goal-based guideline creation process for the examples in sections 2.2 and section 3.1.

2.2 Goal-driven Use Case Creation for a Hands-free Car Telephone Function

The following example (car telephone function) illustrates the classical steps in creating use case descriptions. We call this process “goal-driven” as the search for different user goals plays a pivotal role.

Step1: Describe user context of the system.

To obtain a first overview of the system under design it is quite helpful to find out which actors will be in contact with the system and what are the possible input and output variables that are relevant for system operating. We call this the system context from the user's point of view. We concentrate on the main user – the car driver. The context diagram in Figure 1 portrays possible input and output parameters of the telephone function.

Figure 1. Context diagram for car telephone function.

Table 1 sets out the applicable input and output variables for the telephone example.

Table 1. Variables of car telephone function.

Variable	Description
Ignition	Indicates the position of the ignition key
Steering wheel button 1	Indicates whether the button 1 of the steering wheel is pressed
Voice decoder	Recognizes voice commands of the user
Phone display	Optical display of the phone
Acoustic signal 1	A type of an acoustic phone signal that, for example, indicates an incoming call
Optical signal instrument cluster	Optical signal of the phone that appears in the instrument cluster
...	...

Step 2: Describe use case framework.

Based on the system's context, it is worth examining the goals of the actors. As a rule, we can start with a top goal and refine this goal to several sub-goals and so forth, thus deriving a goal tree that is a foundation for the use case descriptions as we can now write down a use case for

every goal. The use cases describe the necessary steps to reach the goal and the goal name can be the use case name.

If we think about user goals for the telephone example function we can start with a top-level goal such as

- support driver in phone communication.

Sub-goals of this top-goal could be as follows:

- Make phone call.
- Indicate incoming call.
- Output voice message.
- Store phone number.

Each of these goals could be the starting point for a use case. The use cases that are derived from the sub-goals described will be sub-use cases of the use case that is derived from the top goal. Figure 2 illustrates the connection of the use cases in an use case diagram (based on the format provided by "scenarioplus" [7]).

Figure 2. Use case diagram telephone.

Step 3: Refine/Complete use case description.

During step 2 it may be sensible to write down sketchy use case descriptions for some goals (e.g. primary path), if this helps to complete the use case diagram. The detailed and complete elaboration of each use case should follow in step 3. For the elaboration of the use case description, one of the popular use case forms should be utilized.

3. Context-driven Use Case Creation Process

Goal-driven use case creation as mentioned above is a viable approach toward creating use cases of systems that can be easily divided into subsystems. We and other researchers have made good experience with this approach when describing the behavior of a wide variety of software systems. However, we have also dealt with functions where it seemed to be quite hard to develop a

use case description by applying the goal-driven use case creation process, especially in a project where our task was to create use cases that were to describe an innovative assistant system for truck drivers. We therefore thought about another use case creation process that would be more systematical, effective, and usable for our concrete task. We call this use case creation process context-driven in contrast to the goal-driven approach previously introduced.

In this chapter we will present the context-driven process and compare it with the goal-driven process by means of a concrete example function, namely the assistant function adaptive cruise control - Distronic - for passenger cars. We have selected this function as the original functions that we created use cases with this approach for are of a proprietary nature and currently held confidential. The Distronic use cases that we will present are neither original use cases of the Distronic function nor sufficient to describe the whole function, but of course they are quite realistic and adequate to show the essential parts of our approach.

Distronic assists you when driving on motorways or high-speed major roads. If a slower-moving vehicle is detected in front, Distronic causes your vehicle to brake and maintain a preset distance from the vehicle in front. If there is no vehicle in front, Distronic operates the same way as a cruise control. Distronic does not react to a stationary obstacle, e.g. to a vehicle which has broken down, or to an oncoming vehicle in the same lane. In the next section we show the limits of the goal-driven use case process for the Distronic function. We then present the fundamental ideas of the context-driven approach.

3.1 Limits of Goal-driven Use Case Creation for the Distronic Function

If done analogous to the telephone example given (chapter 2), the usual way for goal-driven Distronic use case creation would start with a context diagram for this system (Fig. 3). Typically this job would be done in a use case workshop in collaborative working with different experts who have a raw idea of the essential requirements for the new system.

Figure 3. Context diagram for Distronic function.

The context diagram shows the input and output variables of a system that are relevant to the system user

(here: the driver). Table 2 sets out the input and output variables for the DISTRONIC example.

Table 2. Variables of DISTRONIC function.

Variable	Description
Control lever	Indicates if DISTRONIC is activated by the driver, for example
Brake	Indicates if brake is pressed or not
Accelerator	Indicates if accelerator is pressed or not
Radar sensor	Sensor to measure distance
Optical warning	Optical warning signal, e.g. to indicate too short distance
...	...

The next step will be to brainstorm about the goals of the driver and to make a use case diagram.

A top-level goal for DISTRONIC could be

- Support driver in maintaining the right distance.

Possible sub-goals of this top goal are as follows:

- (De-)activate DISTRONIC.
- Set distance.
- Maintain defined distance.
- Warn driver in case of sensor defect.

This leads us to the use case diagram in Figure 4.

Figure 4. Use case diagram DISTRONIC (goal-driven).

So far the goal-driven use case creation process seems to be quite adequate for the DISTRONIC function. If we take a closer look at the example use case names, it becomes obvious that most of the use cases will be relatively easy to describe, as they are – from the user’s point of view – quite simple (e.g. (de-)activate DISTRONIC, warn driver). But we also recognize that there is at least one goal, namely “maintain preset distance”, that is the fundamental goal of the overall DISTRONIC function. At first sight it would seem that describing a use case for this goal is not

at all problematical, as the goal should be fulfilled automatically by the DISTRONIC system without any user action. However there is one problem: this goal has to be satisfied in any of the many different situations arising in daily usage (traffic situations, weather conditions, etc.).

In many cases we deal with systems (e.g. telephone) that behave largely independent of their context; yet there are different (and comparatively easy to elicit) goals of the user. DISTRONIC is an example for a function where the user has only one key goal, but this goal should be reached in many, many different system situations that depend on the system’s context. So in this case, the difficult task does not lie in finding out the goals of the user but in finding out the critical situations for the DISTRONIC function from the user’s perspective. For each of these critical situations – which are contingent on the system’s context – a use case should describe the behavior of the system. Even though some of the existing use case guidelines consider the context of use cases and the process of use case discovering e.g. during a workshop [6, 8], we thought about an even more systematic way to find out especially the critical context situations that need to be regarded in use case creation. To obtain these critical context situations and thereby suitable use cases, we suggest a process that we call context-driven.

3.2 Context-driven Use Case Creation for the DISTRONIC Function

Context driven use case creation is recommended to describe the behavior of contextual (sub-)functions that are to fulfill a single key user (sub-)goal (e.g. control loops). We will demonstrate the basic notion of this approach for the sub-goal “Maintain defined distance” of the DISTRONIC function. As we concentrate on this sub-goal, we change the abstraction level and define this goal as our top-level goal. Then we start our use case creation process in the same way as done before: actors are defined (we again concentrate on the driver), a context diagram is created (we can use the existing diagram in Fig. 3), and the top-level goal is written down.

The top goal is

- Maintain defined distance.

The next step is the core of the context-driven process. We want to find out which different critical context situations would require that the system – although the user goal stays the same in principle – behave differently in order to ensure the user goal. When we have captured these critical situations, we can think about how the system should react in each of these situations and

formulate corresponding requirements in the form of use cases.

To discover the critical situations, we create a table: the use case context matrix (Tab. 3). In the first column of this table we write down the input variables of the function and add relevant valuations for every variable. In addition we think about context variables (e.g. weather, traffic conditions) and valuations that should be considered to elicit the use case requirements as these variables may have an effect on the desired behavior of the function.

Table 3. Use case context matrix.

	SG1	SG2	SG3	SG4	...
Context variables and valuations					
Weather					
<i>Icy</i>					
<i>Fog</i>			x		
<i>Uncritical</i>	x	x		x	
...					
Traffic Situation					
<i>Oncoming vehicle in same lane</i>	x				
<i>Driving in motorway exit</i>					
<i>High downhill grade</i>		x			
...					
Input variables and valuations					
Control lever					
<i>Activated</i>	x	x	x	x	
<i>Deactivated</i>					
<i>Store Speed</i>					
...					
Brake					
<i>Activated</i>					
<i>Deactivated</i>	x	x	x	x	
Accelerator					
<i>Minimal or higher speed</i>	x	x	x		
<i>Speed exceeds minimum speed limit</i>				x	
<i>Kick-down</i>					
...					

The rest of the context matrix consists of sub-goal columns. In each of these columns we describe a critical situation where the function's behavior has to be described. If a field in a column is marked ("x") this means that, for the situation considered, the corresponding variable has the valuation that is written in the same line as the marking. The variables and values of unmarked fields do not apply for the respective situations.

Proceeding in this way, every sub-goal column describes a critical context situation of the DISTRONIC function through a combination of critical variables and valuations. For each of these situations we specify a sub-goal of the top-goal "maintain defined distance". Every sub-goal meets the same goal as the top goal, but the sub-goal is restricted to the respective critical situation. Therefore it makes sense for the sub-goal name to start with the top goal name and be supplemented by a term for the critical situation. For the DISTRONIC function we could, for instance, find the following sub-goals:

Maintain defined distance in the following case:

- oncoming vehicle in the same line.
- high downhill grade.
- fog.
- speed exceeds minimum speed limit.

This leads us to the use case diagram in Figure 5.

Figure 5. Use case diagram DISTRONIC (context-driven).

The final step of the context-driven use case process is the detailed description of the single use cases. The critical situation of a use case can be documented as its precondition.

Of course this brief description of the context-driven use case creation process cannot show all the details and possible variations of the process. We used a fairly simple example. In reality, the work could be much more complex, but the basic ideas remain the same. The description of a critical situation, for instance, could be more complicated (e.g. temporal sequence of critical events) or similar critical situations could be merged. We could also consider non-human actors like sensors or other (sub-)systems.

The essential idea of the context-driven process is to focus on the critical context situations of a system to discover the situations for which use case descriptions are necessary. Furthermore, a use case context matrix could support us to organize use case flows. But for an application like the DISTRONIC function the chief problem does not lie in the description of a use case itself but rather in finding the critical context situations. A use case

context matrix can help in this task. Of course this cannot replace other techniques such as brainstorming about critical events. Yet a use case matrix can serve to achieve a better overview and find critical situations more systematically. As all the possibly relevant input and context variables and valuations are listed in a table, it is easier to undeck critical combinations.

The context variables and valuations play a pivotal role. At first sight, it would appear that every critical situation should be describable actually only by a combination of the input values of the relevant function. However, as we want to describe this function, we do not know exactly which input sensors etc. are needed at the beginning. Even if the system will not be able to detect every critical situation in the end, it is necessary to think about what would happen if possibly critical events - such as "driving in motorway exit" or "bendy road" - occurred. This allows us to decide about the important user requirements and also the limits of the system under design. Input and context values of a use case context matrix may be reusable for another context matrix, especially if the corresponding system functions are members of the same domain (e.g. automotive assistant systems).

4. Conclusion

Generally, existing use case guidelines focus implicitly on functions that can be successfully analyzed by breaking them down into a hierarchical set of goals. For these functions, it is the priority in use case creation to find out the important user (sub-)goals and describe the relevant user-system interaction. In this case the use case creation process is goal-driven while the context variables are used to detail the use case descriptions. This approach has been successfully employed to limit the scenario-explosion problem for the class of classical application software systems. However, the goal-driven use case creation process does not provide the best fit for all the function classes.

Another class of systems or functions is more dependent on contextual influences such as "weather conditions", "road conditions", "traffic flow" than on user goals. Examples for this class of systems are advanced driver assistance systems. For this class of functions the goal-driven approach delivers a decomposition of functions that is too coarsely grained to be helpful in requirements engineering. The lack of attention to the context potentially causes missing requirements.

For contextual functions, in particular, we suggest combining the goal-driven process with a context-driven process as it was described in this paper. The essential step of this approach is to undeck the critical context situations that have to be considered for use case creation.

We have made good experiences with the context-driven process when the goal-driven approach lead to only one major goal, that has to be considered in many different circumstances (e.g. control loop functions).

The use case context matrix as suggested in our approach can be helpful in eliciting critical context situations for a software system more efficiently and systematically. Critical situations frequently cannot be detected on the technical level alone but exclusively from the perspective of the user. Discovering the critical context situations is crucial if the quality of software systems is to be improved, particularly in context-driven, safety-critical domains.

10. References

- [1] Cockburn, A., 1997. Structuring Use Cases with Goals, *Journal of Object Oriented programming* (Sept-Oct 97).
- [2] Constantine, L. L., Lockwood, L. A. D., *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley, 1999.
- [3] Rolland, C., Ben Achour, C., Guiding the Construction of Technical Use Case Specifications, *Data & Knowledge Engineering Journal* 25(1-2), 1998., pp. 125-160.
- [4] Denger, C., Paech, B., Benz, S., Guidelines - Creating Use Cases for Embedded Systems, 078.03/E, Fraunhofer Institute for Experimental Software Engineering (IESE), Kaiserslautern, 2003.
- [5] Lauesen, S. Task Descriptions as Functional Requirements, *IEEE Software* 20(2), 2003., pp. 58-65.
- [6] Armour, F., Miller, G., *Advanced Use Case Modeling*, Addison Wesley, 2001.
- [7] Alexander, I., Scenario Plus, www.scenarioplus.org.uk, 2003.
- [8] Bittner, K., Spence, I., *Use Case Modeling*, Addison Wesley, 2003.