A Transformation of Controlled Natural Language Behavioural Requirements into Modal Sequence Diagram Simulation Models for Requirement Conflict Detection

Master’s thesis in Software Engineering

ALEXANDER STYRE

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2017

A Transformation of Controlled Natural Language Behavioural Requirements into Modal Sequence Diagram Simulation Models for Requirement Conflict Detection
ALEXANDER STYRE

Supervisor: Grischa Liebel, Department of Computer Science and Engineering
Examiner: Eric Knauss, Department of Computer Science and Engineering

Master's Thesis 2017:NN
Department of Computer Science and Engineering
Software Engineering Division
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Gothenburg, Sweden 2017

ABSTRACT

PURPOSE: Specifying requirements in a semi-formal notation, such as a controlled natural language (CNL), allows reduction of ambiguity and underspecification in requirement specifications, as the notation uses well defined semantics and enforces consistency and conformance to syntactical rules. Contradicting requirements can be problematic to detect in practice depending on the size and complexity of the requirement specification. Requirement simulation is an opportunity to ameliorate the process of detecting inconsistency in requirement specifications. A formal requirement notation, unlike a semi-formal one, comes with the ability to perform requirement simulation. A formal notation, however, requires training and familiarity with formal methods in order to be understood, and this is not something that is suitable for every organisation. Yet, identifying and resolving conflicts between requirements early will help organisations reduce rework, i.e. nonessential efforts. If we can translate a set of requirements in a semi-formal notation into a formal notation, we facilitate the adoption of a useful practice in organisations that would not otherwise like to, or be able to, adopt formal methods.

METHOD: The study adopts the methodology of design science research. Design science research addresses a specific problem that exists in at least one setting and proposes a product such as a model, a principle, a tool or a technique to solve this problem. In our case, the problem can be stated as translating semi-formal requirements into a formal notation for strengthening the scope of validation to include the detection of contradicting requirements in specifications. We use freely and publicly available requirements from the Economic Council of Europe and Daimler-Chrysler to show the application of our translation. These requirements come from a safety-critical requirements domain (the automotive industry) and describe behaviour of vehicular systems. As safety-critical systems have high safety requirements, we propose a consistent translation into simulation models, i.e. a mapping between one source model element into a target model element in a consistent way. We analyse the simulation models created by our transformation and discuss the feasibility of our approach.

RESULTS: The results show that it is problematic to perform an accurate translation of semi-formal behavioural requirements specified on a higher level of abstraction with lower attention to specificity and detail comparatively into a formal notation describing precise details on a more concrete level for the purposes of simulation. Consequentially, a CNL describing behaviour with one specification approach can not fully capture all the information that is required by a fully automatic translation into a formal notation with a different specification approach, without first making essential improvements and necessary adjustments to account for the differences between the two specification approaches and to mimic numerous semantic elements from the formal notation onto the semi-formal notation.

CONCLUSION: We propose that specifying requirements in a semi-formal notation to reduce ambiguity and underspecification in specifications, and then translating the requirements into a formal notation for inconsistency detection, is feasible. This can be applied in the automotive industry and elsewhere where it is considered useful to improve the ability of testing procedures to detect inconsistency in requirement specifications for the purpose of streamlining efforts. It is particularly important for safety-critical systems, where there could be serious consequences of an anomalous specification. Furthermore, we propose features of a semi-formal notation that is susceptible to translation into a formal notation for simulation purposes, which could be used as a starting point for adopting the tool suite that we introduce in this study.

KEYWORDS, ACRONYMS, CONCEPTS AND CATEGORY DESCRIPTORS

Behaviour requirements: In this study, we used two sets of **behaviour requirements**, the Daimler-Chrysler instrument cluster requirement set and the Worldwide Harmonized Light Vehicle Test Procedures (WLTP) gear shift requirement set.

Controlled natural languages (**CNLs**) are limited subsets of the natural language.

Modal sequence diagrams (**MSDs**) are a variant of UML sequence diagrams (**SDs**), implementing play-out semantics. MSDs are inspired by Live Sequence Charts (**LSCs**), enabling specification of mandatory and optional behaviour.

Play-out scenarios are specified in ScenarioTools as modal sequence diagrams (**MSDs**). Scenario Markup Language (**SML**) is the (equivalent) textual format of MSD. We will treat play-out scenarios, SML and MSDs as synonyms and refer to them interchangeably in this report.

ScenarioTools is an Eclipse Modeling Framework-based simulation environment allowing for specifying MSD scenarios and then using play-out simulation to identify anomalies in the specification.

Categories and Subject Descriptors: D.2.1 [**Software Engineering**]: Requirements/Specifications—Languages, Tools

CCS Concepts: •**Software and its engineering** $\rightarrow$ **Model-driven software engineering; Requirements analysis; Model checking; Domain specific languages; System description languages;**

Additional Key Words and Phrases: model-driven engineering (MDE), requirements engineering (RE), model-driven requirements engineering (MDRE), ScenarioTools, scenario markup language (SML), scenario description language (SDL), modal sequence diagram (MSD), message sequence chart (MSC), live sequence chart (LSC), controlled natural language (CNL), deterministic transformation

Contents

1. INTRODUCTION

Specifying requirements in a semi-formal notation such as a controlled natural language (CNL) is the preferred way of specifying requirements for safety-critical systems [2015]. A CNL can, depending on the language, reduce underspecification, because it stipulates details that must be explicitly specified in order for the requirement to conform to the language. This enforced presence of details and an explicit ordering of components in the language help requirement engineers to specify requirements in a consistent format. There is tool support for specifying requirements in CNL by using a language workbench called Xtext, which is part of the Eclipse Modeling Framework (EMF).

While CNL is able to reduce underspecification and enforces conformance to a specific format, determining if a CNL requirement specification is consistent, i.e. non-contradicting could be problematic in practice, depending on the size and complexity of the specification. If there are many, interrelated requirements that, while being correct and completely satisfiable on their own, clash and collide when combined as a unit, these requirements describe erratic and unsound behaviour which cannot be satisfied. Proceeding with development of such a system could lead to profound consequences in a safety-critical domain. The process of finding inconsistency can be ameliorated by using requirement simulation. Requirement simulation allows to identify contradictions, i.e. conflicting requirements, by verifying that the behaviour of the system specified by a requirement specification, in totality, is correct and consistent. If this can be done in an early stage of a project, there can be a reduction in waste, i.e. nonessential efforts, because it strengthens the ability of a requirement engineer to detect anomalies in requirements specifications.

Specifying requirements for the purposes of simulation and detection of contradiction, requires a formal notation. In our study, we have selected a model language called modal sequence diagram (MSD), a language based on sequence diagrams in the Unified Modeling Language (UML). MSD is available in a platform called ScenarioTools, a freely available plugin project in Eclipse Modeling Framework. MSD implements the executional semantics of the Live Sequence Chart (LSC) language, adding the ability of specifying a message in universal mode, i.e. a message which must always occur. The main use case for LSC and MSD is to model behaviour and the ability to make a distinction between what must happen and what could happen (the distinction between universal and existential mode) allows the requirement simulation to identify violations (inconsistent specification). Finding a violation during a requirement simulation implies that the specification is inconsistent, i.e. describes incompatible behaviour which cannot be satisfied.

MSD, being a formal language, requires familiarity and training in formal methods. However, considering that requirements are meant to be understood by not only a select number of requirement engineers, but a vast majority of people for communication purposes in a project, it is not a suitable choice of language for most organisations. Marko et al. [2015], who conducted a research study as part of the CRitical sYSTem Engineering AcceLeration (CRYSTAL) research program, argue that using semi-formal specification is the preferred way of specifying requirements for safety-critical systems in e.g. the automotive industry. The authors reason that natural language is ambiguous. Likewise, the authors argue that it is inappropriate to specify requirements in a formal language because (i) it is problematic to specify the requirements depending on the background of the requirements engineer, (ii) it is unlikely that the customer can understand requirements when they are specified in a formal language and (iii) there is unsatisfactory tool support for formal languages in applications which enable the requirements engineer to specify software requirements.

It would be useful if one can specify requirements in a semi-formal notation for communication purposes and then translate the requirements into a formal notation for simulation purposes. This is useful because we reduce the human error; if we specify requirements in both CNL and MSD we might not end up with semantically equivalent representations and furthermore, there is a reduction in effort if we can reuse information from the CNL requirement specification. CNL requirements, being a semi-formal notation with well defined semantics, are suitable input to model transformations. Therefore, we would like to investigate the possibility of transforming requirements specified in semi-formal notation into formal notation.

The recent trend towards a greater continuity in software development processes is staggering. Processes such as continuous integration [Beck 2000], continuous delivery [Humble and Farley 2010] and continuous deployment [Holmström Olsson and Bosch 2014] have been suggested. Continuous integration means that integration is done frequently on the main branch, continuous delivery means that the product is *always ready to be released* to the customer whenever a successful integration has been finished and continuous deployment means that the product is actually released to the customer with every single build. This opens up a whole new world in which researchers can contribute new knowledge and innovation to software engineering. We concur that greater continuity in software engineering is important, and advise that the validation and simulation of requirements be done as continuously as possible.

For instance, reducing testing feedback time in continuous integration is a challenge which several researchers have studied, e.g. Nilsson et al. have proposed a technique for identifying the frequency of testing activities on different levels; it describes the activities according to their level of automation and the coverage for four different kinds of requirements [2014]. The idea is to bring awareness of the testing activities in a given company, which helps to identify possible opportunities for reducing the testing feedback time in continuous integration. A study which focused on the optimal selection of test cases in a system was performed by Knauss et al. [2015].

Research questions

Specifically, this thesis addresses the following research questions:

RQ1: To what extent can behaviour requirements be expressed in selected controlled natural languages?

RQ2: To what extent can behaviour requirements be expressed in modal sequence diagrams?

RQ3: To what extent can a deterministic transformation facilitate transformation of the behaviour requirements expressed in controlled natural language into modal sequence diagrams?

Motivation for the research questions are the following. Firstly, we don't know to what extent the selected controlled natural languages are able to express the behaviour requirements used in this study. The fact is that the CNLs used in this study are relatively new and have been tried on some requirements. But, we don't know if they are applicable to the requirements in our study.

Secondly, we don't know to what extent we are able to express the behaviour requirements in SML. There have been research studies where some behaviour requirements have been listed and successfully tried out. But we can't generalize and say that this will be the case for the behaviour requirements in this study.

Thirdly, we don't know if a deterministic transformation approach is appropriate for transforming the requirements in controlled natural language into modal sequence diagrams. Answering this question will help us establish whether a deterministic transformation approach is feasible in practice.

Outline

The rest of the report is divided into the following sections: Section 2 is the background, which will describe model-driven engineering, transformation approaches and message sequence charts. Section 3 will describe the design science-based research methodology, the method of evaluation and threats to validity. Section 4 will outline the design process, the rationale for the selection of controlled natural languages (CNLs) and the behaviour requirements and expressing the requirements in SML. Section 5 is an additional service to the reader, which will describe Eclipse Modeling Framework, Xtext and ScenarioTools. Section 6 is the description of the artifact(s), where we will describe a small set of requirements and a conceptual overview of the transformation. Section 7 is the evaluation. Section 8 is the discussion. Section 9 is the conclusion, where we will answer the research questions and provide suggestions for future studies. We end with appendices. Appendix A is the behaviour requirements. Appendix B is the controlled natural languages. Appendix C is the controlled natural languages expressed in Xtext. Appendix D is the requirements expressed in CNL. Appendix E is the requirements expressed in SML. Appendix F is the transformation expressed in Xtend.

2. BACKGROUND

2.1 Coping with the uncertainty of requirements engineering through play-out simulation

Brooks, Jr. has stated that "The hardest single part of building a software system is deciding precisely what to build." [1987, p. 17]. This is in line with the findings of Procaccino et al. [2002] and Ghazi et al. [2014]. Procaccino et al. found that inadequately specified requirements are a key factor for failing to meet the project objectives. Similarly, Ghazi et al. found that clearly stated requirements are a key factor for being able to achieve the project objectives.

Because late changes to a project can be cost-prohibitive and technically difficult to realize, it would be useful to minimize the amount of late rework as much as possible. One way in which we can achieve this is to simulate the requirements — early, continuously, frequently.

Somé presents a use case-based approach to requirements engineering in which the requirements are specified as use cases, then they generate a state machine and simulate the interaction of requirements [2006]. In this study, we look at a specific variant of requirement simulation — play-out simulation. There are two main parts of this approach, play-in and play-out [Harel and Marelly 2003]. Play-in allows stakeholders to specify scenarios describing the behaviour of a system. Play-out is then used to simulate the scenarios and allows stakeholders to identify problems with the software requirements specification, such as incomplete, inconsistent and misunderstood requirements.

ScenarioTools is a tool which allows requirements engineers to specify (and simulate) play-out scenarios [Greenyer 2011]. This tool is described in Section 2.5. Greenyer has shown that it is useful to specify scenarios and identify

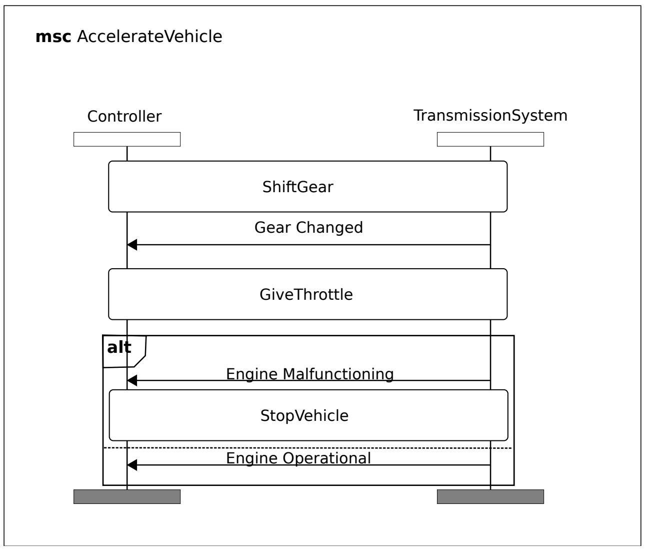


Fig. 2.1. Example of an MSC.

anomalies in the requirements specification by using simulation. We will describe modal sequence diagrams, which are used to specify play-out scenarios, in Section 2.2. Modal sequence diagrams are a specific type of message sequence charts (MSCs), which we will describe in the next section.

2.2 Modal sequence diagrams

A message sequence chart (MSC) is a graphical language that describes the interaction between actors in a system [Telecommunication Standardization Sector of ITU 2011]. They consist of scenarios which mainly describe actors, messages and parameters. A popular variant of MSCs are sequence diagrams in the UML. Figure 2.1 shows an example of an MSC. A description follows:

Two actors, Controller and TransmissionSystem, transmit and receive messages between each other. ShiftGear is a reference to another MSC, where presumably more messages are sent between Controller and TransmissionSystem (and error handling et cetera takes place). In this scenario, we consider only the success story, i.e. the gear has changed. Then, another MSC, GiveThrottle is triggered. This MSC could involve more actors i.e. the TransmissionSystem could communicate with Engine. Finally, an alternative region where we imagine that (i) the acceleration damages the engine and we stop the car *unless* (ii) the engine actually does not suffer any damage and remains in operation. Here we end the MSC.

The main criticism of message sequence charts is that MSCs merely specify what could happen in a system [Damm and Harel 2001]. That is, they are semantically weak because there is no prescription of what must happen. For this reason, MSCs are usually referred to as modelling only existential behaviour, i.e., *at least one* instantiation is required to successfully satisfy the chart. Live sequence charts (LSCs), on the other hand, is a specific variant of MSCs that introduces liveness properties, i.e., they distinguish between what must happen and what could happen. Therefore, LSCs are referred to as multi-modal sequence charts, i.e., modelling both existential and universal behaviour — where universal behaviour means that *every* instantiation of a scenario is required to successfully satisfy the chart. Because of the distinction between mandatory and optional behaviour, LSCs can be used to formally verify the behaviour of e.g. distributed environments such as driver-less railway cars [Damm and Westphal 2005]. An example of an LSC is

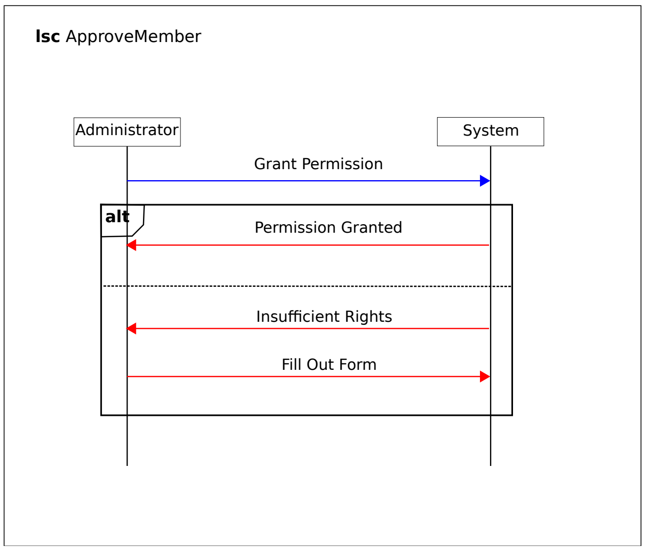


Fig. 2.2. An LSC showing an interaction between A and B in a system.

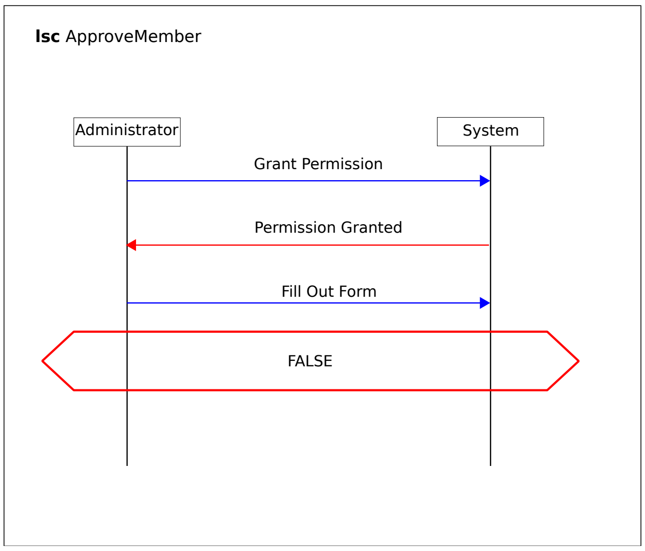


Fig. 2.3. An LSC showing forbidden behaviour.

shown in Figure 2.2. Note that the lines are colored red and blue. A common analogy in LSCs is temperature. Mandatory behaviour is associated with being 'hot' and optional behaviour is associated with 'cold'. Hence, LSCs use red and blue colors to distinguish between these two kinds of behaviour. A description follows:

Two actors, Administrator and System, communicate with each other. An Administrator could (but not necessarily) grant permission to some unprivileged user. We imagine the system must not allow any banned users to be granted any permission by a single administrator, so we permit the administrator to fill out the request for permission form (which will be delivered to other administrators in the interface, who will then be able to reject or accept the request for permission). Alternatively, a non-banned user may be granted permission by a single administrator. Either one of these paths must be triggered because they are both strictly required, i.e. they must occur.

LSCs allow engineers to express behaviour which cannot happen. This is referred to as forbidden behaviour and modelled with one or more hot conditions, i.e., conditions in universal mode, at the end of a chart; if the condition is matched, the chart is violated [Harel and Marelly 2003]. An example of this is shown in Figure 2.3. This scenario places an invariant on the interaction in Figure 2.2. Basically, after the Administrator got the permission request granted, it is *forbidden* to fill out the form to ask administrators to reject or accept the request for permission (that is, it must not happen — Ever). The message Fill Out Form is modelled as being possible, i.e. it *might* occur. If it does, the condition at the end of scenario will trigger a violation. If, however, the event labelled Fill Out Form is specified as strict, i.e. it must occur, then the scenario would be unsatisfiable once it has been activated for the following reason. After Grant Permission is received, Permission Granted is strictly required and must occur, otherwise the LSC scenario is violated. Fill Out Form likewise is strictly required and must occur, otherwise the LSC scenario is violated. Because the LSC scenario will progress to the (strict) FALSE condition at the end of said scenario, the entire scenario will be considered to have been violated. LSCs, like MSCs, can be expressed in a formal model, i.e. a Büchi automaton, but the difference is that LSCs are expressed in a Co-Büchi automaton. An example of a Co-Büchi automaton is shown in Figure 2.4. A description follows:

State 1. *Requirement Scenario Deactivated.* The state is final, because a scenario does not need to become active. Self-transition M \ {grant permission} is omitted from this figure.

States 2, 3 and 5. The *First Message* is received. State 2 is not final, because the transition can either go into State 3 (*Success*) or State 5 (*Violation*). If permission granted is not received next, the transition M \ {permission granted} will occur. State 3 is final — accepting and State 5 is a non-final, i.e. non-accepting, rejecting dead state (i.e. there is no sequence of transitions which will lead to a final, accepting state). The transition M \ {fill out form} from State 3 to State 1 is omitted.

State 4: Additionally if, after having reached State 3, the message submit form occurs, the transition to State 4 (*Forbidden Message*) will be triggered. State 4 is non-final and triggers the ϵ-transition to State 5.

Sequence diagrams, which traditionally have been heavily inspired by MSCs, were revised in UML 2.0 and included assert and negate statements. Harel and Maoz argued that these constructs were meant to model mandatory and forbidden behaviour in a sequence diagram, with inspiration of LSC [2007]. However, said authors also heavily criticized the definition of assert and negate statements in the UML 2.0 standard as expressively weak, contradictory and ambiguous. Instead, the authors suggested it would be useful to express interaction in terms of modality, i.e. having a mode, something which the UML 2.0 standard had omitted, thereby giving birth to a new variant of LSCs, modal sequence diagrams (MSDs). MSDs are implemented as a UML 2.0 profile and allow the specification of mandatory and forbidden behaviour in UML sequence diagrams. MSDs as defined by Harel and Maoz allow a short-hand notation for describing mandatory and forbidden behaviour — *assert* and *negate*. An asserted sequence has a hot, i.e. mandatory mode. This is shown in Figure 2.5. A description follows:

Two actors, Controller and Device (let's say a Remote Control sending signals to a Blu-ray player for instance) communicate with each other. The signal PlayVideo Button Pressed could be sent (but not necessarily, because there could be other signals being received in other scenarios e.g. Mute Button Pressed). If the signal PlayVideo Button Pressed is triggered, the scenario PlayPauseVideo becomes

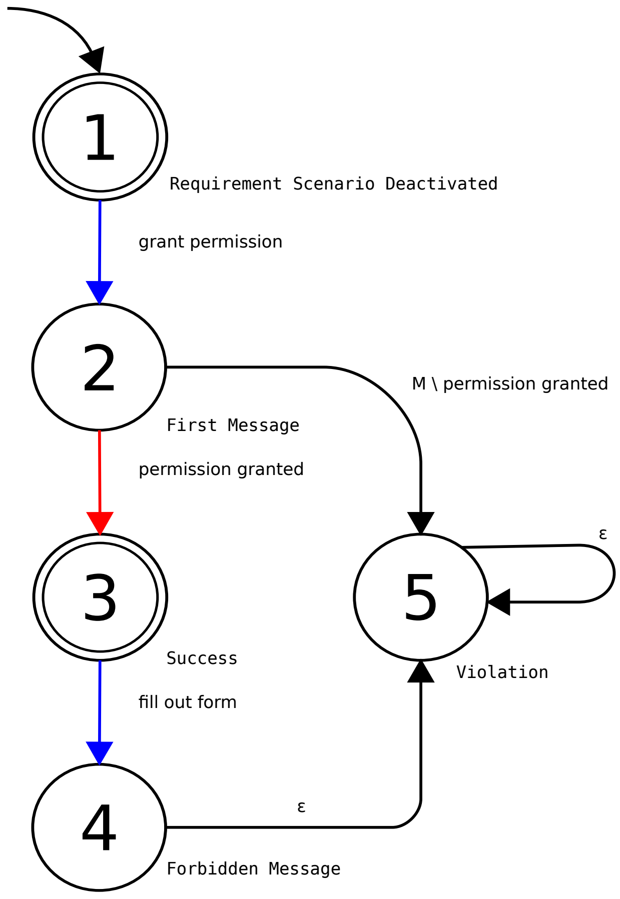


Fig. 2.4. An automaton showing forbidden behaviour.

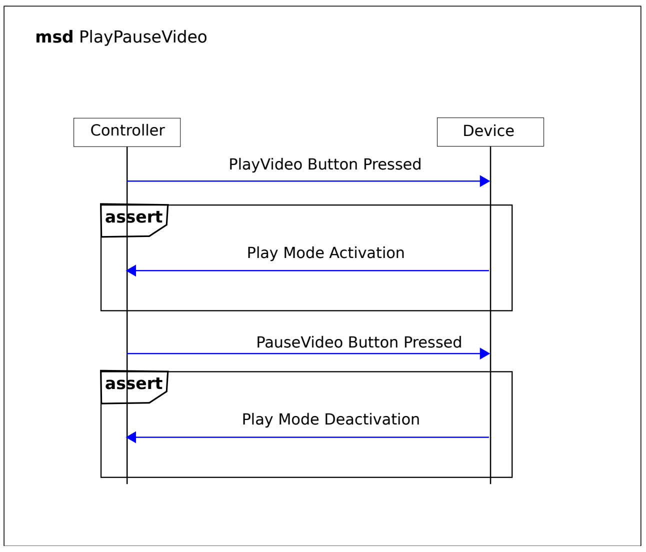


Fig. 2.5. An example of an asserted sequence.

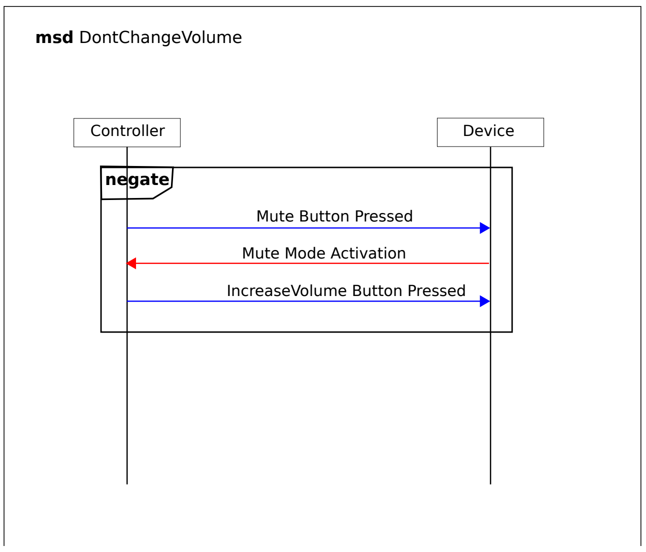


Fig. 2.6. An example of a negated sequence.

activated. The assert region has a hot mode which applies to all messages (in this case, the message Play Mode Activation becomes hot, despite being represented as blue, i.e. being in cold mode). The rest of the scenario states that if PauseVideo Button Pressed signal is received, then Play Mode Deactivation must occur.

A negated sequence has a hot mode and a condition at the end, which evaluates to false. This is shown in Figure 2.6. A description follows:

We continue with the two actors in Figure 2.5. This scenario specifies a sequence where a signal to increase volume level is forbidden when the disc player has been put in muted mode. For clarity, we do not specify Mute Mode Activation inside an assert region, because that would make the diagram a bit more complicated than needed so we simply color it with red color as to indicate that this message is asserted.

Greenyer was inspired by the research on LSC, and their UML 2.0 profile MSD, and suggested that this could be used to identify anomalies and deficiencies in requirement specifications [Greenyer 2011]. His main contributions are:

—The definition of modal sequence diagrams (MSDs). The MSD language is a variant of SD and includes several additions, including semantics of LSC and the MSD UML 2.0 profile proposed by Harel and Maoz [2007]. MSD makes the difference between a system and its environment explicit, the notion that messages are always coming from said environment and being able to specify universal and existential modality based on LSC semantics

—An Eclipse Modeling Framework plugin named ScenarioTools for specifying and simulating requirements in MSD. The tool is described in Section 2.5.

—A synthesis algorithm, which is used to generate a strategy for simulating the requirements successfully with the play-out algorithm, or if that is not possible, a strategy for violating the specification.

It should be noted that while we credit Greenyer to have initiated the research, he is certainly not alone in making the research contributions and certainly not the only one to have made efforts in the research program. In fact, he has acknowledged the help of dozens and dozens people, several of which we refer to in our thesis such as Fockel and Holtmann, Brenner et al. and Liebel and Tichy. Specifically, we note that the research into ScenarioTools was published in a research article by Brenner et al. [Brenner et al. 2013] and later additions were made to said platform by a large host of researchers [Brenner et al. 2014]. We will now describe what Brenner et al. has to say about ScenarioTools and the MSD language [2013]:

Table I. Summary of the languages capable of modeling scenarios.

Language	Graphical	Textual	Degree of formality	Mode
MSC	Yes.	Yes.	Formal.	Existential
SD	Yes.	Yes.	Semi-formal.	Existential
LSC	Yes.	Yes.	Formal.	Existential+Universal
MSD (UML 2.0 profile)	Yes.	Yes.	Formal.	Existential+Universal
MSD (ScenarioTools)	Yes.	Yes.	Formal.	Existential+Universal

Summary of the languages capable of modeling scenarios.

```
requirement scenario R1 {
message env -> gs . selectGear(1)
message requested env -> gs . setAccPhase (true)
violation if [ gs.clutchEngaged ]
}
```

Fig. 2.7. How we can model interaction between two actors, env and gs, in SML.

In regard to mandatory behaviour, the authors make a clear distinction between safety and liveness violations. Safety violations mean that there is a defined order of events, i.e. a safety violation would indicate that a message was triggered at an unbidden time. Liveness violations mean that there are expectancies of events, i.e. a liveness violation would indicate that a message *has not been* triggered at all. The authors also implemented ScenarioTools, a tool for executing play-out simulation using the revised modal sequence diagrams. These MSDs can be expressed either in a graphical language or in a textual language. We use mainly the textual language in this study. This language is known both as Scenario Markup Language (SML) and Scenario Design Language (SDL). In this study, we use SML to refer to the textual language of MSDs. An example of a play-out scenario expressed in SML is shown in Figure 2.7. The scenario is read as: Two actors, Environment env and GearSelector gs communicate with each other. The scenario specifies a sequence where if the first gear is selected and the vehicle accelerates, a violation is triggered should the clutch be engaged.

Time conditions were proposed in LSCs by Harel and Marelly [2002] and later implemented in MSDs by Brenner et al. [2014]. An example of a time condition in an MSD is shown in Figure 2.8. A description follows:

If the disc player is given the sleep signal, then it must at some point be put back in operation. The signal Wake Up must occur after a cooldown period of at least 5 seconds, otherwise the scenario is violated. The scenario is read as: If the wake up occurs after 5 seconds, the time condition clock < 5 is evaluated to false and the scenario is *interrupted* because the condition is in cold mode. Otherwise, the scenario is violated because there is a strict FALSE condition at the end of scenario (see Figure 2.3).

Table I presents a summary of the modeling languages capable of modeling scenarios. Figure 2.9 show the relationship between MSD and related modeling languages. MSD was selected because it is a formal model language with simulation capabilities, it has good and freely accessible tool support, it is available in Xtext language workbench, which the author is familiar with and based on the widely used UML sequence diagram language, which the author also is familiar with. Furthermore, the distinction between mandatory and optional behaviour provides an unambiguous specification of mandatory and optional behaviour and is useful for the validation of requirements in safety-critical domains [Damm and Westphal 2005].

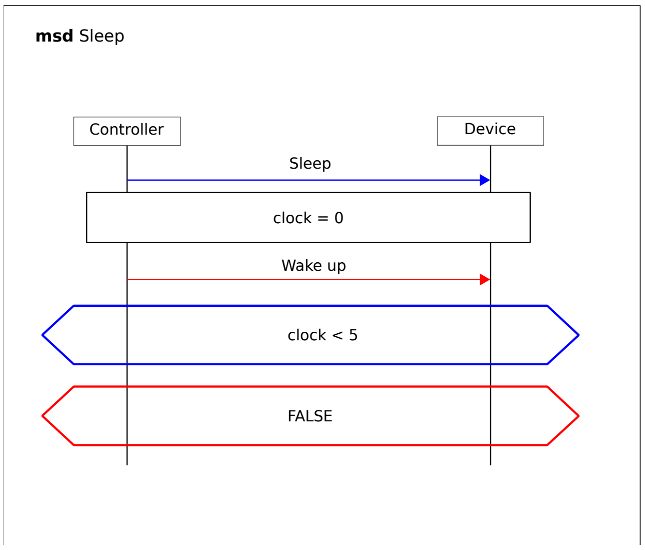


Fig. 2.8. An example of a time condition in MSD.

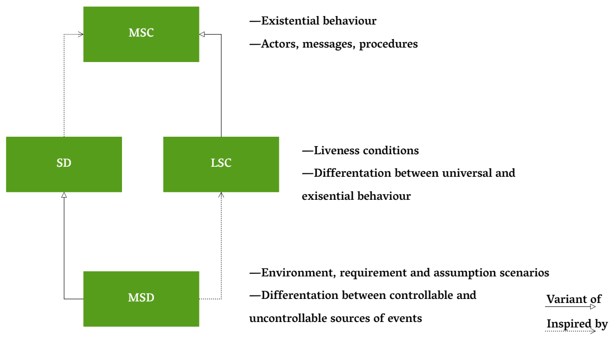

MSC
—Existential behaviour
—Actors, messages, procedures
SD
LSC
—Liveness conditions
—Differentation between universal and exisential behaviour
MSD
—Environment, requirement and assumption scenarios
—Differentation between controllable and uncontrollable sources of events
Variant of
Inspired by

Fig. 2.9. The relationship between MSD and related modeling languages.

2.3 Model-driven requirements engineering with controlled natural language specifications

We have stated that specifying requirements in MSD allows us to simulate the requirements and see in what way they conflict with each other, if there are any contradictions in the requirement specification. However, the quality of simulation and identified anomalies depend on the completeness of the specification. There is no room for ambiguity in MSD, because it requires you to be very certain about, and to specify exactly, what the system can and can not do in a given situation. The interaction must be clearly defined and must state the mode of this interaction, as well as the forbidden and interruption messages. In addition, all of these concepts must be connected to a domain model actors' attributes and procedures to describe exactly which actor initiates any given communication with another actor from the same domain model and what exactly is being communicated. MSD demands a precise and exact specification of all the details necessary to perform simulation on the requirement specification. For some projects, we argue that this may be ideal and fully possible to do. But, if we need to specify underspecified requirements or requirements with uncertainty, we would not be able to specify these requirements in MSD, because we have to be concerned with regards to the low-level interaction between specific actors. There is another problem with specifying requirements in MSD, and that is not related to their attention to details. Mainly, for contractual negotiations and documentation, natural language is by far the most favored approach. We have for instance seen the research by Fockel and Holtmann [2014], where said authors suggest a round-trip approach to facilitate conversion between controlled natural language (CNLs) and SysML block diagrams, was motivated by the demands amongst signatories to be able to have natural or controlled natural language requirements. Therefore, we want to be able to specify requirements in natural or controlled natural language and then transform them into MSD. In this section, we will introduce model-driven requirements engineering.

Model-driven requirements engineering (MDRE) is the application of model-driven engineering to requirements engineering [Berenbach 2012]. An example of this methodology could for instance be seen in the aforementioned study by Fockel and Holtmann [2014]. Model-driven engineering (MDE) is a discipline which cultivates the idea that platform-independent models (PIMs), i.e. models on a higher abstraction level which are independent of technology, can be transformed to other PIMs or to platform-specific models (PSMs), which are models on a lower abstraction level and depend on a technology. MDE integrates activities of a typical software engineering process such as requirements engineering, architecture, design, implementation and testing [France and Rumpe 2007]. Brambilla et al. state that "MDE goes beyond of the pure development activities and encompasses other model-based tasks of a complete software engineering process." [2012, p. 9]. Requirements engineering is a continuous process in which requirements are elicited, specified, prioritised and released [Lauesen 2002].

A central principle of model-driven engineering is the development of domain-specific languages (DSLs). DSLs are "languages that are devised to address the needs of a specific application domain." [Brambilla et al. 2012, p. 70]. Domain-specificity is a desirable property of a language because it promotes the use of familiar constructs. An example of a domain could be a transmission system and a DSL could be defined to specify the behaviour of the transmission system. For instance, the behaviour of the transmission system could be to change the active gear, disengage the clutch and accelerate the vehicle. Controlled natural languages (CNLs) are a specific kind of domain-specific languages. IEEE Std. 830-1998 states that "these languages tend to be better at expressing certain types of requirements and addressing certain types of systems." [IEEE-SA Standards Board 1998, p. 5].

Marko et al. argue that using CNL is the preferred way of specifying semi-formal requirements for safety-critical systems [2015]. This is in line with de Almeida Ferreira and Rodrigues da Silva [2009] and Fockel and Holtmann [2014], who reasoned that CNL is suitable for communication with stakeholders. Additionally, Lauesen [2002] claims that it is inappropriate that requirements engineers specify natural language requirements without being able to establish their correctness or unambiguity.

There are many benefits of using controlled natural languages to express requirements. Firstly, it restricts the expressiveness and subsequent ambiguity in the requirements by forcing the requirements engineers to use mainly a specific and predetermined set of key words and operators to specify requirements [IEEE-SA Standards Board 1998]. Secondly, it provides detection and resolution of problems in requirements specifications [Marko et al. 2015]. Thirdly, requirements engineers can specify requirements in certain controlled natural languages and still be able to negotiate the details of these requirements with customers [Fockel and Holtmann 2014].

2.4 Transformation from requirements expressed in CNL to MSC

We explained that controlled natural languages are a specific kind of DSL. Furthermore, we stated that DSLs are used in model-driven engineering and established that models are transformed in MDE. In model-driven requirements engineering (MDRE), we transform requirements expressed in CNL into another model, in our case, the modal sequence diagrams (a variant of sequence diagrams with LSC semantics). In this section, we describe related research on transformation approaches.

Yue et al. have reported that model transformation approaches for transforming requirements into platform-independent models are unsatisfactory based on the significant amount of effort required to successfully use them [2011]. The authors acknowledge that manual effort can be justified in some cases, for instance to add domain-specific information. The authors assert that user intervention should be minimized in order to provide automation in model transformations. Furthermore, the authors identify the lack of consistent transformation of requirements into models. Specifically, the authors reason that models have to be correct, consistent and complete. Finally, the authors identify the lack of scientific evaluation of model transformations.

In this research, we have a manual intervention step in the transformation for the purpose of adding domain-specific information, which is not specified by the requirements themselves. We thus consider this step to be justified, according to what Yue et al. have stated. The advantage is that there is no longer a need to re-specify, in MSD scenarios, the details which are successfully converted in the transformation. With regard to specification of already specified details in the requirement specification, we avoid re-doing the requirement specification in MSD from scratch if we use the transformation. Furthermore, we have transformed the requirements in a consistent way by using a deterministic approach.

There are many studies of transformations involving the transformation of CNL requirements into models. We have for instance seen such research studies in Fockel and Holtmann [2014] and Somé [2006].

Fockel and Holtmann suggested a round-trip approach where platform-independent models are transformed from a SysML block definition diagram (BDD) into CNL requirements and from CNL into SysML BDD diagrams [2014]. They argued that stakeholders would be much more comfortable reading and negotiating the requirements in CNL rather than reviewing the SysML block definition diagrams. In their study, the authors have demonstrated the feasibility of using Xtext to express requirements in CNL and then apply a transformation on these CNL requirements to derive a complete platform-independent model.

Baudry et al. demonstrated an approach to transform requirements in CNL into use case scenarios for simulation purposes [2007]. The authors reported on the benefits of using simulation is to reveal problems with the specification. There is a fairly limited amount of information available on the transformation they used and the CNL grammar which they used was unrealistic for expressing real requirements, as they concluded. In this study, however, the CNL grammars are realistic and have been used in practice, as we describe in Section 4.3.

Somé [2006] has a similar approach as our study. They use a controlled natural language to specify use cases and then transform them into a state machine using a domain model. The requirements are then simulated. Their simulation approach is similar to play-out simulation. But it does not support liveness conditions (the difference between what *may* happen and *must* happen). Also, the author suggests that requirements are specified as use cases, but states that there are different approaches for this. In his study, the author demonstrated the feasibility of a deterministic transformation of use cases into MSC.

Feijs [2000] created a controlled natural language which he transformed into message sequence charts (MSCs). The author uses mainly three types of sentences, *information* (equivalent to message in a use case), *action* (operation) and *status* (condition). In his study, he investigated different ways to specify requirements and transform into message sequence charts. The author states that the controlled natural language and the message sequence charts are realistic.

Fockel and Holtmann excogitated a controlled natural language which suited the needs of transforming SysML into CNL and backwards. Specifically, they designed a language in which all the information was available, thus making the implementation of model transformation a straightforward task. Somé [2006] used use cases containing all the information needed to transform into MSC.

Unlike the study conducted by Fockel and Holtmann [2014] and the study by Somé [2006], the target language in this study is already out there and can not be easily adapted to fit the needs of the transformation approach. Particularly, requirements by themselves do not contain the information which is necessary to successfully derive such a complete, correct and consistent simulation model. This is a major challenge to tackle in this study and indeed not something which is addressed by for instance, Fockel and Holtmann. As such, the selection of controlled natural language is critically important to this study as it has material consequences on the ability to express the behaviour requirements in this study. The information which is available to the transformation depends on what information we extract from the requirements expressed in controlled natural language.

We did not find any research addressing the same problem as our study. But research suggests that deterministic transformation of controlled natural language requirements to message sequence charts is feasible. Therefore, we think that it is important to execute this thesis.

2.5 Eclipse Modeling Framework, Xtext, and ScenarioTools simulation framework

Eclipse Modeling Framework (EMF) is a framework for creating models and writing transformations. In this study, we used this framework to create a transformation in Xtend, express the requirements in CNL, express the requirements in SML and to simulate the scenarios in ScenarioTools. In this section, we will describe Xtext and its relation to Xtend, and we see that Xtext is the language workbench that is used to express Xtend. We also describe ScenarioTools and how a SML specification could be expressed in Xtext.

In this study, we want to accomplish the transformation of behaviour requirements expressed in CNL into PIMs for simulation purposes. This is realized by using Xtext. Xtext is a language workbench and comes with the Eclipse Modeling Framework (EMF) platform. A language workbench is a concept to describe an integrated platform with the means to create DSLs, including support for deriving (and customizing) an editor for creating concrete representations in a given DSL [Fowler 2010]. Such an editor could include e.g. error detection, highlighting etc. Some language workbenches also support specifying how the execution of the concrete representations is carried out.

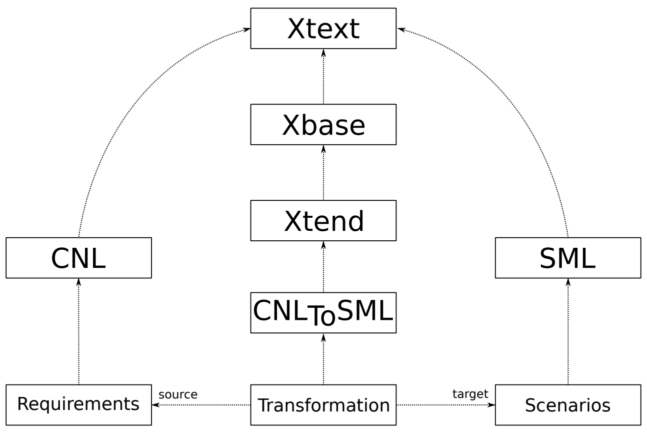


Fig. 2.10. Overview of a model transformation. On the left, CNL. On the right, SML. A connection between them is the model transformation in the middle. This picture has been inspired based on illustrations by Brambilla et al. [2012].

An overview of how the transformation relates to Xtext is shown in Figure 2.10. The source of the model transformation is the requirements expressed in a CNL grammar. The process of getting these requirements expressed in CNL is described in Section 4.5 and the process of implementing the grammars of the CNLs in Xtext is described in Section 4.4. The behaviour requirements are described in sections A.1 and A.2 and the CNL grammars used in this study are described in Section 4.3. The target of the model transformation is the play-out scenarios. The play-out scenarios are expressed in MSDs, which we describe in Section 2.2. We describe the process of getting the requirements expressed in MSDs for evaluation purposes in Section 5.3. SML is the textual version of MSDs, which we describe in Section 2.6. Because SML contains quite a few constructs, we think it is important to describe the structure of an SML specification in this thesis. An SML specification is expressed in Xtext. The transformation operates on and uses constructs from the CNL and the SML grammars (not shown by the figure). As shown in the figure, the transformation is defined in Xtend. Xtend is defined by using Xbase, a Java mapping, which is defined in Xtext.

ScenarioTools is a simulation framework, which is capable of identifying irregularities in scenarios according to the synthesis algorithm described by Greenyer [2011]. As a simple example, Greenyer gives a scenario where something must happen, but in a different scenario, the same thing is strictly not allowed to happen. Therefore, a scenario is violated because there is no way we can legally proceed in the scenario. ScenarioTools can specify scenarios like this and allows simulation, meaning to try different things and see if violations occur. If one wants to know more about ScenarioTools, we refer to Greenyer's doctoral thesis. We will use ScenarioTools to specify requirements in Scenario Markup Language (SML), which we will describe next.

2.6 Structure of SML

Lastly, we will go over the structure of an SML specification. The ScenarioTools configuration consists of a domain model, an instance model, an SML specification and a runtime configuration model. Figure 2.11 shows how the structure of an SML specification looks like. A description follows:

Domain model: an Ecore model defining the context, i.e. the surrounding domain, the actors both inside and outside of the environment and the operations and the attributes they may have.

Instance model: a model is instantiated from the Ecore model with the root class instance containing a reference to each instance of the domain concepts (e.g. the instance of root class Transmission contains one instance each of Environment, GearSelector and GearController).

SML specification: a model responsible for (i) importing the domain model, (ii) defining the actors and the role they play in the scenarios (environment or system), (iii) describing a collaboration (see Collaboration) and (iv) mapping the actors in the SML specification with the concepts in the domain model. A specification has a name.

Collaboration: A model responsible for (i) describing a set of scenarios (see Scenario) and (ii) defining the actors and the role they play in the scenarios (static or dynamic). A collaboration also has a name.

Scenario: A scenario consists of e.g. messages, constraints and violations. A scenario has a name.

Message: Two actors, sender and recipient, send and receive messages. An actor (or more precisely, a corresponding class of the actor) must exist in the Ecore domain model. Each message is defined as a reference to an operation of the corresponding class of the actor on the receiving end, i.e. the operation must exist on beforehand.

Constraint: A constraint is an invariant, which could be cold (interruption, not to be confused with what is said about interruptions below, which does not apply to every message in the MSD). A constraint describes what must not happen when the MSD is active and thus, it makes sense to view constraints as a type of precondition.

Interruption: A cold invariant which happens after a sequence of messages. Unlike interruptions in a constraint, this interruption does not apply to every message in the MSD, just the one it is placed after. What happens is that a cold violation occurs, i.e. the MSD becomes deactivated, when the post conditions are not met.

Violation: Identical to an interruption, the only difference being that the condition is in hot mode, i.e. will lead to a termination of the simulation if the post conditions are not met.

In addition, in order to execute the simulation, you need a runtime configuration.

Runtime configuration: a model instructing ScenarioTools to (i) import the SML specification, (ii) refer to the instance model and (iii) map the actors in the SML specification with the actors in the instance model.

At this stage in this design, we only consider the transformation into an SML specification. Particularly, the domain model, the instance model and the runtime configuration are created manually, as they will likely not change. The only realistic possibilities for changing any of these models is when one of the following occurs: There is an actor, or one of its attributes or operations, which is added, renamed or removed.

3. METHOD

This thesis adopts the methodology of design science research. In this chapter, we introduce the methodology of design science research, the contextualization of design science research and the threats to validity. We chose design science research as we focus on the creation of a product, in our case a transformation that can ameliorate the process of detecting contradictions in requirement specifications, and for establishing and evaluating the usability and feasibility of this approach. Since we propose a method which is not used in industry, we can not use a case study approach, for instance.

```
import "MetaModel.ecore"
system specification Specification {
domain Domain
define Environment as uncontrollable
define Controller as controllable
define Device as controllable

collaboration Collaboration{
static role Environment env
static role Controller c
static role Device d

requirement scenario R1 {
message env -> c.setMuteMode(true)
message strict requested c -> d.setMuteMode(true)
}
}
}
```

Fig. 2.11. The structure of an SML specification containing one collaboration with one requirement scenario.

3.1 Design science research

We will describe the concrete choices of our method, including the method of evaluation, in this section.

Design science research studies use tools and techniques to solve a problem while focusing on the evaluation of these artifacts [Johannesson and Perjons 2014]. In our case, the problem could be stated as "How do we transform controlled natural language requirements into modal sequence diagrams?" and the answer could be a transformation approach (a principle) or a transformation (an artifact).

A natural step in the design phase is to do an evaluation, because we need to somehow establish the value of the answer. The type of evaluation depends on the research questions, so it would not make sense for instance to conduct interviews with practitioners or to do a questionnaire or a survey of their attitudes towards model transformation of requirements if they do not have the answers to our research questions. Therefore, we do not consider a case study to be applicable in our study, because the requirements engineers would need to be familiar with, or to first adopt, the principle which we propose to fully be able to evaluate and assess accurately its efficacy. Furthermore, an experiment could be done e.g. in order to measure the efficiency in time units and the degree of correctness in percentage of the requirements engineers when specifying requirements in semi-formal notation and a formal notation. The experiment could be combined with qualitative interviews enquiring the opinions of the requirement engineering regarding which language they prefer. There would be a number of issues in setting up an experiment like this. One problem would be the experience of the requirement engineers. MSD is a formal notation and requires training in order to be used. Likewise, if the CNLs are new to the requirement engineers, this would also pose a problem in that the data would be skewed and give us an understanding of how a CNL is received among participants with a limited background in using these CNLs. It is likely that the experiment would yield limited results for this reason. A second problem would be to properly acquire a random sample amenable to statistical inference, as including requirement engineers from one organisation is not going to be reflective for the entire industry. Indeed, if an organisation is to be involved, it is likely that the participants would be asked by management to participate, posing a number of ethical issues as well as skewing the results. On the other hand, if we allow self-selection, it would lead to bias. The environment in which the requirement engineers work could be stressful and monitored by management, making participants feel uncomfortable and therefore negatively impact on the results. Another, related problem is how feasible it is to acquire a sufficient number of participants to properly draw a sound conclusion from the data set. A third problem would be to avoid bias while conducting the interview, as it is extremely important not to influence the requirement engineers to give a particular answer. For instance, the researcher should not indicate what he expects as an answer from the interviewee. A fourth problem would be to keep the validity intact. For instance, if a manager acting as a gatekeeper asks to review the answers of each participant, this should be known and agreed on by the interviewees. It would still be questionable to share raw results without first coding away details of the participant for confidentiality purposes, even if agreed, because the participant may be pressured into accepting by management. Furthermore, if requests are made to alter the raw results or the conclusions drawn from the experiment, because a gatekeeper stakes his approval on publishing the thesis on this, it will negatively influence the research results.

Hevner et al. have described methods of evaluation in design science research [2004]. They organize these methods into a classification with observational, analytical, experimental, testing and descriptive. Scenarios and informed argument fall under descriptive methods. Descriptive methods are used for instance to exemplify how a specific artifact is used. We will use scenarios and informed argument as a way to describe the limitations and benefits of this transformation. This is the only option that is left to us, because we want to give the requirements and show the limitations and benefits of this transformation. This is why we want to use freely accessible requirements.

But the usefulness of the results in this study is still applicable to real business. We aim to select a set of requirements from the automotive industry. And of course this does not mean that the requirements are going to be identical to a specific project's requirements. It means that the idea — transformation of requirements into scenarios which are used for play-out simulation — can be used in industry. As such, we want to be able to demonstrate the artifact, this is a key concern of this study. Furthermore, because the purpose of this study is also to answer *how reasonable is this approach?*, we might not be able to create a transformation which is suitable to facilitate transformation of the requirements into modal sequence diagrams. If we find that it will be too limited to be used in practice, then this is also a useful outcome.

By examining the types of CNL requirements that the transformation can handle, one can establish a notion of quality in this transformation approach. That is why it is appropriate to describe the applicable requirements, and the limitations of the transformation approach.

To summarize, our evaluation method consists of the following approach:

—Step 1: Generate a simulation model.

—Step 2: Execute a simulation model in ScenarioTools.

—Step 3: Check the correctness of the simulation model when executed in ScenarioTools and in comparison to the handmade simulation models.

—Step 4: Describe the scenarios, how well they are handled by the transformation, and identify where the problems (if any) are.

Step 1: The transformation should be able to produce a simulation model.

Step 2: The model is tried in the simulation environment ScenarioTools. This will identify (all) syntax and (some) semantic problems.

Step 3: The model is evaluated according to the execution in ScenarioTools and in comparison to the handmade simulation models. A scenario can be simulated if (and only if) it can be specified in ScenarioTools. A scenario may be specified in ScenarioTools only if it adheres to play-out semantics, i.e. it has a finite sequence of messages, it includes actors etc. First and foremost, we must compare the generated simulation model to the handmade one, to see if we are actually able to specify a requirement at all. If we are unable to do so, it is obviously impossible for the transformation to generate a simulation model and thus the evaluation becomes irrelevant. Second, we can say that we are evaluating how the handmade model is handled by ScenarioTools and then comparing the generated model to this expectation. A generated model may however look different than a handmade model, yet it might still produce the correct behaviour. Therefore, we also endorse and carry out execution with ScenarioTools as this is relatively easy to do. Naturally, the scenarios will have to be described in detail so the reader can assess the conceptualization of the problem.

Step 4: This is reported on in Section 6.

In this study, the main artifact is the model transformation into modal sequence diagrams. This artifact is a "situated implementation of artifact" [Gregor and Hevner 2013, p. 342]. However, if we can somehow generalize the guiding principle or provide comparisons to other works, the artifact may be a "nascent design theory" [Gregor and Hevner 2013, p. 342]. There are other artifacts which are required to answer the research questions such as the behaviour requirements expressed in CNL (RQ1), the behaviour requirements expressed in SML (RQ2) and the controlled natural language requirements transformed into SML (RQ3).

The construction of the transformation will follow Jones' model [Wynn and Clarkson 2005]. This process model is shown in Figure 3.1. The analysis step is about determining the problem and the constraints on the solution. In order to describe our constraints, we first go into the activities of design science research. We will do this in the next paragraphs. The synthesis step is about picking one solution among a set of alternatives. For this study, this means that we considered what solution would be most suitable based on the constraints that we had. This also means that we considered different alternative solutions to a problem. For instance, in order to use a deterministic transformation approach, we decided to specify requirements in controlled natural language in order to limit the expressiveness. However, since this restrictive syntax does not have all the information we need, we decided to create a separate transformation and define additional information which requirements themselves do not have. There were also different ways to transform requirements, and we had to decide on which options we thought was appropriate. The evaluation step is about evaluating the quality of the solution which we have described previously in this section.

The methodology has been introduced and the method of evaluation and construction have been outlined. We want to provide more insights into the constraints and the choices we

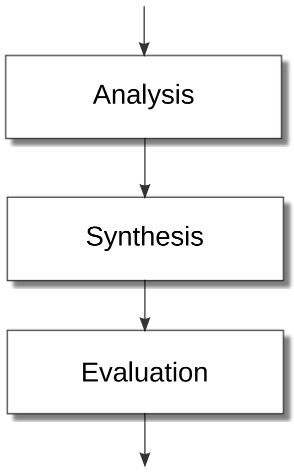


Fig. 3.1. Jones' design process model [Wynn and Clarkson 2005, p. 38].

have made in our study. Therefore, we describe the activities of design science research; these activities are grouped into three cycles — relevance, design and rigor [Hevner and Chatterjee 2010, p. 17].

The relevance cycle is about establishing the context, identifying constraints on the solution and justifying the construction of the design artifact. In our study, we have chosen the automotive industry, which is especially sensitive with regards to safety and where a mistake can result in huge fines, life-and-death situations and loss of revenues in a business there is a significant amount of competition. Requirements, as we described in Section 2 may be poorly communicated and misunderstood. If this is discovered early in the process, we said this is going to be a huge improvement as this will reduce waste. And, we stated that it will be harder to apply changes the further into the process one is in. Therefore, continuously being able to verify requirements in this domain is important. However, a manual specification of SML requirements is not a good approach in our opinion. Then they can be put down in the wrong way and give false positives. Such an error could evaporate the benefits of using play-out simulation. Similarly, a probability-based transformation approach would infer details of the SML specification which are not provided by the requirements. The uncertainty is that we could be in one of those special *unlikely* events which don't happen very often, and then we will get the wrong SML specification. This kind of uncertainty is inappropriate in the domain of safety-critical engineering. Especially if you are to verify a requirement specification which is large and could include hundreds of requirements, maybe even thousands. Then, you would be hard-pressed to identify a single mistake in the specification of the requirement simulation scenarios.

Constraints are therefore:

—behaviour requirements should be from the automotive industry

—a deterministic transformation (should convert any number of requirements in the same way)

—requirements should be expressed in controlled natural languages

The rigor cycle is about drawing ideas from existing knowledge, establishing appropriate selection of methods for evaluating the artifact and aligning the research contribution with existing research. First, we read up on the existing literature about ScenarioTools, model transformation approaches, model-driven requirements engineering with CNL, message/live sequence charts and the general challenges of model-driven engineering. We also read up on existing literature about design science research and decided on a method of evaluation of the transformation.

The design cycle is about the construction of the design artifact, using the knowledge gained from the rigor cycle and the constraints placed on the design artifact by the relevance cycle. Based on the constraints and the knowledge we had, we began by expressing the requirements in SML, to establish whether we were able to express this. Then, we expressed the requirements in CNL. And finally we began writing the transformation in Xtend. A detailed description of the design, including the selection of behaviour requirements, controlled natural languages and writing the model transformation and the evaluation is further described in Section 4.

3.2 Threats to validity

Internal validity: We have defined a mapping definition which helped us convert some of the requirements into complete play-out scenarios. The transformation relies on two things to be correct: the requirements are expressed in a specific way, with a certain format and the mapping definition has to be correct and properly identify the target actors etc, otherwise it will not produce correct output models. This is mentioned in Section 4.

Construct validity: There is a formal theory behind MSDs and we integrate this into the evaluation of the research contribution, as we describe in sections 4.6 and 5.3. It is thus believed that the method of evaluation makes sense for establishing what the transformation can and can not do.

External validity: It is not possible to draw any inferential conclusions based on this study because the population of requirements is difficult to measure and cope with. The sheer amount of requirements that exists today and the continuous growth of requirements make such a generalization of results unattainable. However, the principles could theoretically be applied in different settings and projects, but we can't know and make generalized assumptions that it will work for every type of project. If we did that, the results would not be scientific nor would we be fair in our assessment. Rather, we can only say what the transformation can and can not do with the requirements in this study, which is pointed out in Section 8.2. Since we publish the transformation online [1], there is a possibility for independent researchers to acquire and to use the transformation on a different set of behavioural requirements to research how well it works for these behavioural requirements. Furthermore, we have selected real requirements from the automotive industry in order to strengthen the external validity of our test.

Conclusion validity: The results of this study depend on the ability to express the requirements in CNL, SML and the ability to create a transformation in Xtend all depends on the background of the researcher. It could be that there is a better way to convert some of the requirements, but this is not achieved because of lack of experience. Therefore the conclusion validity is low. The selection of requirements is also skewed and a reason that the study is biased. Convenience sampling was applied. There are thus many different requirements excluded from this study. Our behaviour requirements come from one domain, which can not account for behaviour in different domains. Furthermore, our behaviour requirements do not account for the behaviour in all vehicular systems. Our interpretation and understanding of

[1] https://github.com/alexanderstyre/ConstraintDslToSml

the requirements are threats to conclusion validity, because it could have a negative impact on the results of this study. Furthermore, our understanding of the CNLs and MSD is also a threat to conclusion validity, because the results from the transformation depend on the manual conversion of natural language behavioural requirements into CNL. Lastly, because the transformation can be found online, there is a possibility for researchers to get a copy of the transformation and review the results of this study by running the transformation on our behavioural requirements.

Ethical issues and impact of the research: An increase in automation could lead to a reduction of manual labour and increase the effectiveness of the field, and eventually obsolete the need for requirement engineers specializing in requirement validation. This could lead to a loss of job opportunities, which burdens employment agencies, state politicians and society at large. Another consequence of this study is to raise the awareness of simulation of requirements for validation purposes, which could lead to more organisations adopting this methodology and thereby increase the correctness of requirement specifications in industry. Finally, another consequence of this study is to bridge the gap for organisations to adopt formal methods in software engineering. This could raise the bar for requirement engineers and software engineers both in education and in industry to learn more formal methods.

4. DESIGN

In this section, we introduce a more thorough conceptual overview of the process of designing the solution. Firstly, expressing the requirements in CNL. Secondly, the changes made to the CNL grammars needed in order to provide for such a specification and justifications for these changes. Thirdly, for conducting the evaluation of the transformation, we describe the process of expressing the requirements in SML.

4.1 Conceptual overview of the design process

A conceptual overview of realizing the model transformation of this study is demonstrated in Figure 4.1. The first step is to document the requirements in a CNL (RQ1). The second step is to use a model transformation approach to derive an incomplete model based on the information which is stored by the requirements. The third step is to add domain-specific information to derive a complete model. Finally, the complete model is evaluated to establish the quality of the model transformation (RQ3).

4.2 Selecting behaviour requirements

In this study, we have used behaviour requirements from the automotive industry to demonstrate the results of our study. The behaviour requirements were selected based on availability, having been published in an online, freely accessible medium. They were furthermore also published by reputable agents in the field, one being a manufacturer of cars (Daimler-Chrysler) and the other being a council of security experts (organised under the UN).

The requirements are found in Appendix A. In this study, requirements R1-R7 refer to requirements a-g in Figure A.1 while requirements R8-R16 refer to features 1-7 and use cases (UCs) 1-2 in Section A.2.

4.3 Selecting CNLs

In this study, the following controlled natural languages were used: the state machine language defined by

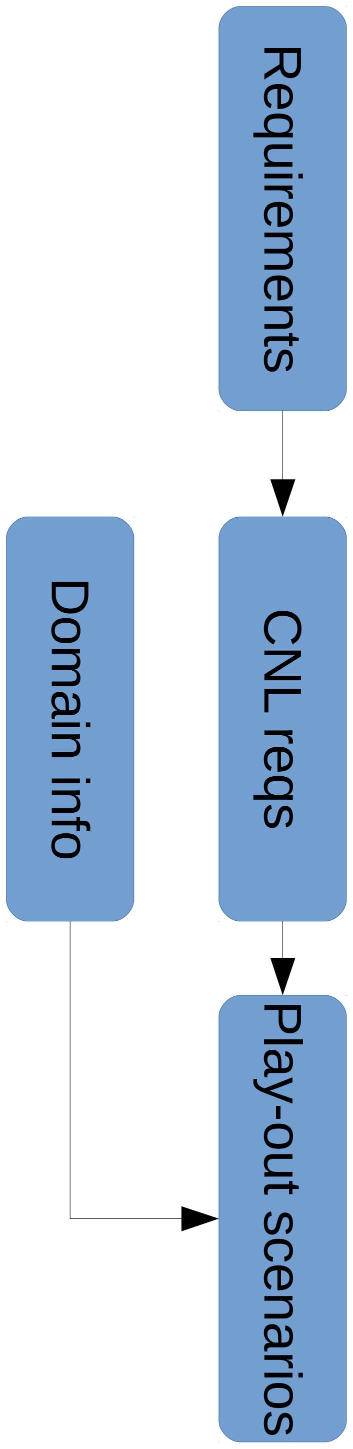


Fig. 4.1. Overview of how to apply a model transformation approach to transform a CNL requirements specification into a complete platform-independent model.

ID R01: While state: waiting, the system: system shall action: select parameter: digit quantity: 1 within time: 5 s before system: system starts state: dialling.

Fig. 4.2. A requirement expressed in CNL Marko.

trigger = if event "waiting" with key ("dial_initiated", "value") == 0 occurs condition = event "dialling" with key ("data", "value") == key ("dial", "digit") occurs within 5 seconds.

Fig. 4.3. A requirement expressed in CNL Vierhauser.

Marko et al. [2015] and the constraint language defined by Vierhauser et al. [2015].

The CNL grammars were selected based on availability and having been published scientifically. Furthermore, they seemed to support the expression of the requirements in this study.

Marko: Marko et al. used a state-based language. This language was used in order to specify conditions on what must happen depending on what state a system is in. An example of a requirement expressed in the grammar used by Marko et al., henceforth referred to as CNL Marko, is seen in Figure 4.2.

Vierhauser: Vierhauser et al. used a constraint-based language. This language was used in order to specify temporal and events-with-data constraints to express what must happen in a system. An example of a requirement expressed in the grammar used by Vierhauser et al., henceforth referred to as CNL Vierhauser, is seen in Figure 4.3.

CNL Marko and CNL Vierhauser are rather different in terms of what they can express. While they both can express requirements for safety-critical systems, their approach in doing so differs significantly in terms of the concepts they have used. One is the state-based approach adopted in CNL Marko, i.e. the system is modelled as a set of states with a set of transitions between them and the other is the event-based approach adopted in CNL Vierhauser, i.e. the system is modelled as a set of trigger events with a set of adjacent future and past occurrences of events which may contain data attachments and invariants on data. Please remember, however, from Section 2, that the distinction between a state-based and an event-based specification approach is not relevant for this study. Particularly, we said that MSDs are a form of MSCs, which can also be expressed in automaton models. There is thus no real barrier to express MSDs with either approach. This is in line with the research study conducted by Liebel and Tichy, who performed an experiment with a sample of requirements which they found to be possible to model with either a state-based or an event-based approach [2015]. Of course, these results are only an indication and not necessarily generalizable to other settings. However, we are of the opinion that this study gives us a rational and logical explanation as to why we should not prefer one approach over the other—that is, the two CNLs in our study are judged solely based on their ability to express the requirements.

The essence of play-out simulation is that it depends on being able to exactly specify the expected behaviour of a system. It is thus a formal language. A CNL, on the other hand, is a language which mainly contains a set of keywords and operators which does suit the need of expressing requirements in order to reduce ambiguity in specifications, but does not have a framework that defines the executable semantics of a requirement.

The CNLs are found in Appendix B.

ID R01_acceleration: While state: standstill and if parameter: clutch is constant: disengaged, the system: transmission shall action: select parameter: gear quantity: 1 within time: 1 s before system: transmission starts state: acceleration.

Fig. 4.4. Requirement R1 expressed in CNL Marko.

trigger = if event "clutch_disengaged" with key ("transmission", "clutch_disengaged") == true occurs condition = future events "gear_selected" with key ("transmission", "active_gear") == 1, "acceleration_phase" with key ("transmission", "acceleration_phase") == true do occur within 1 seconds.

Fig. 4.5. Requirement R1 is expressed in CNL Vierhauser.

4.4 Realizing the CNL grammars in Xtext

There is a conceptual overview of the platform in Section 2.5. In this section, we describe how we went about realizing the CNLs in Xtext. For CNL Marko, we had access to a copy of their Xtext project, so no realization process was performed. For CNL Vierhauser, however, we did not have access to such a project and we therefore decide to realize the grammar of CNL Vierhauser in Xtext based on their description of their grammar in the report by Vierhauser et al. [2015]. CNL Vierhauser was realized as follows: First, we created an Xtext grammar project in EMF (having installed the Xtext SDK plugin from the Eclipse Marketplace) and then a default (unrelated) grammar was given. We then progressively defined (and refined) until the grammar was fully realized in accordance with the paper by Vierhauser et al. Furthermore, we attempted to specify the example requirements that were given in that paper with success. A revision of the original CNL Vierhauser was made, which we describe in the next section, and that we give the name of "Revised CNL Vierhauser".

4.5 Expressing the requirements in CNL

Since the behaviour requirements were expressed in natural language, we needed to express the requirements in CNL. This is done by first expressing the grammars of a CNL in Xtext, as we described in Section 4.4 and then generating a model editor in EMF. Then, we start an instance of Eclipse and load the model. Then, we are able to create a new file where we can express requirements in the given CNL.

Based on our understanding of the controlled natural language and the requirement, we specified the requirement how we thought the requirement should be expressed in CNL. Based on this approach, we found that the first requirement from the WLTP gear shift requirements should look like that in Figures 4.4 and 4.5.

The requirements in CNL are described in Section 5.2.

A couple of changes were needed in order to fully express the WLTP gear shift requirements in CNL Vierhauser. The revised grammar can be found in Section C.3. These changes were:

Ability to express conditions. Examples:
future events A, B do *not* occur
future events A with a *or* b, B do occur
future events A with a *and* b, B do occur

Ability to express multiple occurrences of events with data. Example: future events event1 with dataitem1, event2 with dataitem2 do occur

Ability to express events with no restriction on time. Example: future events A, B do occur

Ability to express addition and subtraction of data items.
Example: future events A with a + 1, B with b - 1 do occur

In addition, changes were performed to make the distinction between past and future occurrences more clear.

Examples:
future event A occurs
future events A, B do occur
past event B has occurred
past events A, B, C have occurred

Changes were also made to make the expression of cold and hot events possible.

Examples:
future event A may sometimes occur (cold)
future events A, B may sometimes not occur
future event B must always occur
future events A, B, C must sometimes not occur

The interpretation of "may sometimes" is that it is entirely possible to occur, but not enforced, i.e. it does not necessarily follow from a sequence of precedent events that this must occur *every time*. However, it is required that it happens at least once. Note that trigger events are always considered to be in cold mode.

Multiple trigger events were also added to make the distinction between precedent events and antecedent events, something which was needed in order to properly handle the *timing of events* in the transformation.

Examples:
if A AND B occurs then future event B may occur
if A AND B AND C occurs future events D, E must occur

Currently, only 'AND' is supported. It is of course not possible to transform past to future constraints with multiple trigger events. Instead a past constraint will always be transformed into a future constraint with one (and only one) trigger event.

4.6 Expressing the requirements in SML

ScenarioTools comes with an editor for Scenario Markup Language (SML). In this study, this editor was used to specify play-out scenarios. To execute the scenarios, you need to define a number of models, which we explain in Section 2.6.

Like we saw in Figure 2.4, MSDs can be expressed in automatons. A requirement can be expressed in an MSD if (and only if) it can be expressed in a deterministic Co-Büchi automaton. Figure 4.6 shows the first WLTP gear shift requirement expressed in an automaton model. A description follows:

State 1. *Requirement Scenario Deactivated.* The state is final, because a scenario does not need to become active. Self-transition M \ {disengage_clutch} is omitted from this figure.

State 2. *First Message.* The state is final, i.e. if nothing else happens, the scenario is fulfilled, because the next message is optional, i.e. in cold mode. The transition M \ {select_gear(1)} from State 2 to State 1 is omitted in the figure.

State 3. *Second Message + Start Clock.* The state is final, because the driver does not need to accelerate after having selected the first gear. For instance, the driver might want to select neutral gear. In that case, the scenario is deactivated (the transition M \ {accelerate} from State 3 to State 1 is omitted in the figure).

States 4-6. The *Third Message* is received. The driver accelerates. The fourth state is not final, because there is a

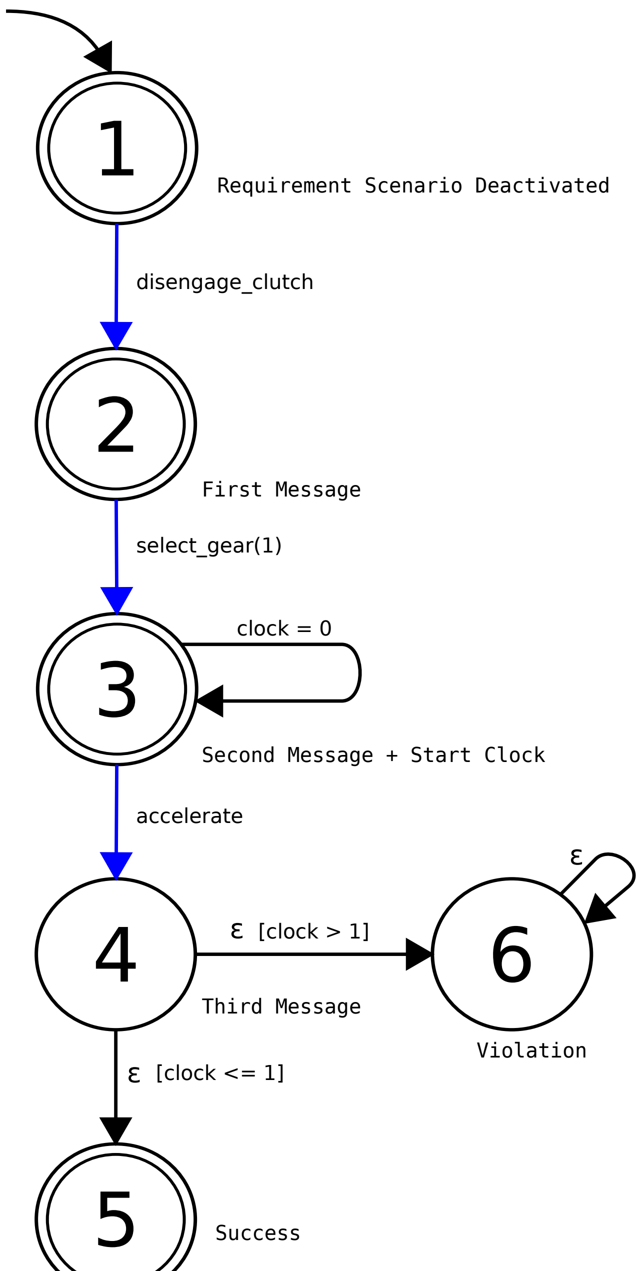


Fig. 4.6. An automaton showing the first requirement from the WLTP.

time condition involved. Either the transition to State 5 (*Success*) or State 6 (*Violation*) must be triggered. State 5 is final, i.e. accepting and State 6 is non-accepting. To enter State 5, the acceleration must have occurred within one second after shifting to the first gear, i.e. after having entered State 3. From State 5, the scenario is deactivated (the transition ϵ to State 1 is omitted in the figure).

Based on the automaton depicted in Figure 4.6, we can derive how the scenario should be expressed. That is, it follows that the MSD (in SML) should look like that of Figure 4.7.

Firstly, the scenario assumes the existence of the environment and system actors. However, additional mapping and/or further breakdown might be needed to properly reflect the actual simulation model.

Secondly, environment must be uncontrollable for the messages to be activated in the console.

```
requirement scenario R1 {
message environment -> system . setClutchDisengaged (true)
message requested environment -> system . selectGear (1)
var EInt clock = 0
message requested environment -> system . setAccPhase (true)
violation if [ clock > 1 ]
}
```

Fig. 4.7. Requirement R1 expressed in SML.

Thirdly, all messages are requested (but not strictly, i.e. in hot mode). The first message is not allowed to specified as requested, however it is still viewed as being requested; a scenario can only become activated when its first message has occurred, hence the first message is *requested*. ScenarioTools is clever enough to know that a requested message should be enabled in the console, that is, there is no requirement to have one more scenario with the same messages *without* being marked as requested.

The requirements in SML are described in Section 5.3.

4.7 Writing the model transformation

Due to time limitation, we decided to go with the transformation of requirements expressed in revised CNL Vierhauser to an SML specification. In this section, we describe the conceptual overview of the model transformation.

Figure 4.8 shows that the CNL requirements are transformed in two steps into a simulation model. The first step is to derive the basic sequence of interaction between two quasi-actors, one being named environment and the other being named system. What happens is that the requirements are transformed into an incomplete SML specification where these quasi-actors communicate with each other, specifying what is communicated and when the interaction may occur, as well as the constraints on the interaction, i.e. the post conditions (what must hold after a given message has been communicated). However, because everything is derived from the requirements, it is not possible (yet) to simulate the output model. That is why we need to create a mapping between the quasi-actors and actual actors in the complete transformation. In the second step, we iterate through each construct in the incomplete model and link it to an existing actor, domain, operation or concept if present in the mapping definition.

Initial transformation: The initial transformation tries to achieve one thing and is guided by one question: *What information can we derive from the requirements themselves?*

Complete transformation: Because the initial transformation is unable to derive everything needed in order to simulate the requirements with ScenarioTools, we write another transformation, which is guided by one question: *What information do we need to add to the incomplete model?*. This leads to *a mapping definition*.

Mapping definition: The mapping definition consists of the information which is required to facilitate simulation in ScenarioTools.

The initial and the complete transformations are described with more detail in Section 5.4.

5. ARTIFACT

5.1 CNL grammars expressed in Xtext

Marko: The grammar of CNL Marko is implemented in Xtext by Marko et al [2015]. To acquire an official copy of this language, please contact the authors of that study. We do not include it in the report as to encourage readers to acquire an official copy of the language.

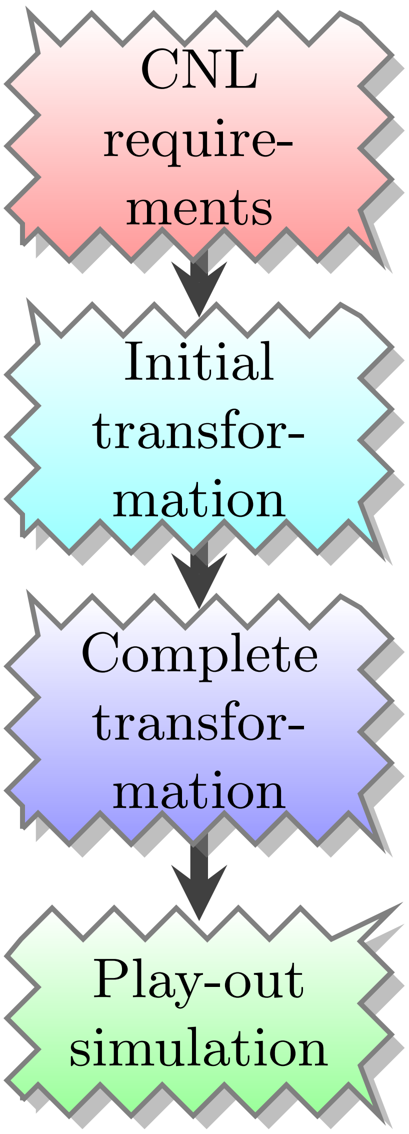


Fig. 4.8. How the behaviour requirements expressed in CNL get transformed into play-out scenarios.

Vierhauser: CNL Vierhauser was implemented in Xtext and can be found in Section C.

5.2 Requirements expressed in CNL

Figure 4.4 shows the first requirement from the WLTP gear shift requirements expressed in CNL Marko.

Figure 4.5 shows the first requirement from the WLTP gear shift requirements expressed in revised CNL Vierhauser.

The remaining WLTP gear shift and DC instrument cluster requirements expressed in CNL can be found in Section D.

5.3 Requirements expressed in SML

Figure 4.7 shows the first requirement from the WLTP gear shift requirements expressed in SML.

```
import "MMA.ecore"
system specification Specification {
domain Domain
define System as controllable
define Environment as uncontrollable
collaboration Collaboration {
static role Environment environment
static role Environment system
requirement scenario R1 {
message environment -> system.clutch_disengaged ( true )
violation if [ clutch_disengaged != true ]
message environment -> system.gear_selected ( 1 )
violation if [ active_gear != 1 ]
var EInt clock = 0
message strict requested system -> system.acceleration_phase (
true )
violation if [ acceleration_phase != true ]
violation if [ clock >= 1 ]
}
}
}
```

Fig. 5.1. Initial play-out scenario for R1.

The remaining WLTP gear shift and DC instrument cluster requirements expressed in SML can be found in Section E.

5.4 Model transformation expressed in Xtend

For the evaluation of the transformation, we refer to Section 6.

Initial transformation: The transformation works mainly in terms of future constraints, i.e. constraints with future occurrences. A past constraint is transformed into a future constraint. For instance, the constraint A then B, C, D, E have occurred is semantically equal to the constraint B then C, D, E, A occur (We group B, C, D, E as an ordered sequence of events, take B as our first event in the future constraint and lastly append A as the last occurring event).

The future constraints are transformed in the following way:

One constraint corresponds to exactly one scenario of type REQUIREMENT, i.e. a requirement scenario, in *the initial transformation*. The name of a scenario is automatically generated by the transformation, e.g. a scenario with at least one constraint will have at least one requirement scenario with the name "R1". We will have messages between two quasi-actors, as we describe in Section 4.7. ScenarioTools can not use this information to simulate requirements. We must provide domain-specific information, such as actors, operations and parameter types. Therefore, we suggest another transformation can provide domain-specific information.

The outcome of the intial transformation is an initial play-out scenario. This is incomplete and can not be simulated by ScenarioTools. The initial play-out scenario for R1 is shown in Figure 5.1.

Complete transformation: The complete transformation takes the incomplete play-out scenario given by the initial transformation and a mapping definition to derive a complete play-out scenario.

Mapping definition: Figure 5.2 shows the mapping definition for the first requirement from the WLTP gear shift requirements. The mapping definition is defined once and used for all requirements.

The implementation of the model transformation can be found in Section F.

```
domain transmission {
import_uri = "Transmission.ecore"
roles {
role GearSelector gs as controllable,
role Environment env as uncontrollable,
role GearController gc as controllable
}
}
mapping{
messages {
message 'clutch_disengaged' from 'environment' to env ->
gs.setClutchDisengaged
message 'clutch_disengaged' from 'system' to gs ->
gs.setClutchDisengaged
message 'gear_selected' from 'environment' to env ->
gc.setActiveGear
message 'gear_selected' from 'system' to gs -> gc.setActiveGear
message 'gearshift' from 'environment' to env -> gc.setActiveGear
message 'gearshift' from 'system' to gs -> gc.setActiveGear
message 'acceleration_phase' from 'environment' to env ->
gs.setAccPhase
message 'acceleration_phase' from 'system' to gs ->
gs.setAccPhase
}
parameters {
parameter 'clutch_disengaged' to gs.clutchDisengaged
parameter 'active_gear' to gc.activeGear
parameter 'acceleration_phase' to gs.accPhase
}
kinds {
kind EBoolean 'selected_acceleration_phase'
}
}
```

Fig. 5.2. Mapping definition for R1.

6. EVALUATION

In this chapter, we will report on our evaluation. We first present our self-assessed understanding of the requirements in this study. Then, we discuss the expressiveness of CNL Marko, CNL Vierhauser (standard and revised) and MSD. Finally, we discuss the suitability of our transformation to transform the CNL requirements into MSD.

6.1 Material

The evaluation is based on the material referenced in Section 4 and 5.

We introduced the requirement numbering in Section 4.2. We would also like to clarify that R2 is essentially three requirements in one and for the purpose of keeping track of these three requirements, each sub-requirement is given a suffix a, b, c and explained below. Requirement R2a refers to "Gears shall not be skipped during acceleration phases", R2b refers to "Gears used during accelerations [...] must be used for a period of at least three seconds" and R2c refers to "Gears used during [...] decelerations must be used for a period of at least three seconds" [United Nations Economic and Social Council 2013, p. 72].

Furthermore, a few requirements were skipped:

—Part of requirement R1 was skipped, because we argue that the second sentence stating that we should be standing still if the speed is less than 1 KM/h was a clarification and not a requirement. As such, this sentence was skipped from our evaluation.

—We argue that, apart from the condition of being in an acceleration or a deceleration phase, requirement R2c is

Table II. Summary of the evaluation of the expressed and transformed requirements in this study.

Requirement	RQ 1 M	RQ 1 V	RQ1 RV	RQ 2	RQ 3
R1	N	N	Y	Y	Y
R2	Y	N	Y	Y	N
R3	N/A	N/A	N/A	N/A	N/A
R4	Y	N	Y	Y	Y
R5	N	N	Y	Y	Y
R6	N	N	N/A	N/A	N/A
R7	N	N	N	Y	N/A
R8	Y	Y	Y	Y	Y
R9	Y	Y	Y	Y	Y
R10	Y	Y	Y	Y	Y
R11	Y	Y	Y	Y	Y
R12	Y	Y	Y	Y	Y
R13	Y	Y	Y	Y	Y
R14	Y	Y	Y	Y	Y
R15	N/A	N/A	N/A	N/A	N/A
R16	N/A	N/A	N/A	N/A	N/A

A summary of the requirements in this study.

identical to what is already stated in requirement R2b. Therefore, we treat requirement R2c in the same way as requirement R2b.

—Requirement R3 was skipped, because it states that something may happen. We argue this is a clarification of behaviour instead of a requirement. Based on our motivation for expressing R1 and R2 in CNL Marko and CNL Vierhauser, we believe that it is possible to express this requirement in revised CNL Vierhauser. Because we can not distinguish between universal and mandatory behaviour in CNL Vierhauser and CNL Marko, we can not express this requirement in respective CNL. We furthermore believe that it is possible to express this requirement in MSD, for the same reason as in R1 and R2. We also think that our transformation will transform this requirement into a complete play-out scenario correctly.

—We can say that for the second part of requirement R7, we do not need to re-check the sequences because this is covered by other scenarios. If a violation is going to happen, it will be detected by the other scenarios (R1–R6).

—We also present our evaluation of one requirement in the Daimler-Chrysler instrument cluster requirement set. We argue that requirements R9–R14 are variants of requirement R8. They do not give additional requirements, which we do not already handle in the WLTP requirement set. For instance, requirement R9 introduces timing of events. We addressed this concern in the WLTP requirement set.

—We argue that requirement R15 is identical to what is already stated in requirements R9–R10.

—We argue that requirement R16 is identical to what is already stated in requirement R12.

6.2 Self-assessed understanding of the requirements

We think that most of the requirements were unambiguously specified and it was easy to understand the intention and meaning of the requirements in this study. However, there were some exceptions:

R1 says "one second before" [United Nations Economic and Social Council 2013, p. 72] and some may argue that this is ambiguously specified. This could be interpreted in two ways, either as exact comparison i.e. it occurs *exactly* one second before or as similar comparison, i.e. it occurs *within* one second before. We interpreted the requirement as the latter, since we are talking about behaviour and we argue that it is possible to satisfy this requirement if we shift the gear within one second. As for what the requirement really says, we bear in mind that the WLTP requirement set talks about seconds as integers, and thus, it would seem to favour the first and the second interpretation of this requirement. The reasoning behind this requirement is that we have the first gear being selected *at most* one second before entering the acceleration phase.

R7 was not easy to understand because we do not think it was unambiguously specified. Instead of saying "a lower gear is required at a higher vehicle speed" [United Nations Economic and Social Council 2013, p. 73], we think it would have been better if it said "while the speed is increasing". In this case, we can identify that there is a parameter named speed (or vehicle speed) that is increasing, whereas "higher vehicle speed" is possibly misunderstood as a categorization of speed (lower, low, medium, high, higher). Even there is a possibility to interpret this in a different way. Does it mean that a lower gear is required to accelerate and get an even higher vehicle speed? However, by reading the whole requirement, we could understand the intention and think that we understood requirement.

6.3 Summary

A description of how well the requirements were expressed in CNL Marko, CNL Vierhauser, revised CNL Vierhauser and MSD and then transformed into MSD by the transformation is found in Table II. The table uses the following characteristics:

RQ1: No — Not expressed in CNL, because it was not possible, Yes — Expressed and represents the information of the requirement in natural language correctly.

RQ2: No — Not expressed in MSD, because it was not possible, Yes — Expressed, represents the information of the requirement in natural language and interpreted correctly by ScenarioTools.

RQ3: No — Not transformed correctly, Yes — Transformed, represents the information of the requirement in natural language and interpreted correctly by ScenarioTools.

6.4 Expressing the requirements in CNL Marko

We could express some of the requirements of the WLTP requirement set in CNL Marko. We could express all of the requirements of the Daimler-Chrysler instrument cluster requirement set in CNL Marko.

We argue that it is not possible to express requirement R1 in CNL Marko. If you read the requirement, it says that every time we stand still and the first gear has been selected, we have to start acceleration within one second, if the clutch is disengaged. However, this requirement is strictly not true. There may be instances where you would like to engage the clutch or even turn off the engine for example. That is not what the requirement in NL says. It does not follow that, based on disengaging the clutch, that you have to switch to the first gear.

Another way to specify requirement R1 in CNL Marko is "if we enter the acceleration phase, the system shall not switch gear after 1 second". This is semantically equivalent to it is forbidden to switch gear after 1 seconds. While this is true, it does not say anything about what the user must do. The user can simply neglect to switch gear at all and still satisfy the requirement. It captures only part of the information in the required behaviour. If the user selects the first gear after 1 second, then it is a violation, but it does not follow from the CNL requirement that the user needs to select the first gear at all. Simply saying that the user must not do something is not the same as saying that the user must do something.

We argue that requirement R2a is possible to express in CNL Marko. It is slightly harder to take into account this negated behaviour in CNL Marko. We can say that the gear shall not be selected, but we have to introduce a new variable and say that this new variable is unequal (not equal) to i + 1. This is a slightly more complicated specification than for revised CNL Vierhauser, where we could simply state that the active gear is not equal to $i \pm 1$ without introducing a different variable.

We argue that it is possible to express requirement R2b in CNL Marko. In CNL Marko, we say that whenever we go into an acceleration phase and the gear is changed, we can not change gear to another gear for at least 3 seconds, which is true. This was slightly more tricky to express than in revised CNL Vierhauser, because we have to introduce another variable and the possibility to simply state "system: transmission shall action: use parameter: gear for at least time: 3 s" is a serious limitation of CNL Marko. The semantics of this specification is ambiguous. Does this mean that we shall use the same gear for at least three seconds, or does it mean that we shall repeat the action of selecting any gear for three seconds? It is thus less formal than revised CNL Vierhauser or even the "right" CNL Marko in that sense and leaves room for interpretation. There is another concern with CNL Marko. We can express the requirement in a similar way as we expressed the requirement in CNL Vierhauser, but in CNL Marko we talk about states and actions, which do not allow us to talk about timing of events as we can do with CNL Vierhauser. Here we want to talk about not switching a gear until after three seconds. This is both an action (CNL Marko) and event (CNL Vierhauser). In CNL Marko, we have no ability to express an action as a precondition, instead we must use parameters and states. In CNL Vierhauser, we have the ability to express events and events with data as pre-conditions and we argue that this can be viewed as one of the benefits of using revised CNL Vierhauser instead of CNL Marko to express this behaviour requirement.

We argue that it is possible to express requirement R4 in CNL Marko. In CNL Marko, we simply state if there is a transition between an acceleration phase and an deceleration phase (with an intermittent state acceleration_stop), then we can not change gear.

We argue that it is not possible to express requirement R5 in CNL Marko. In CNL Marko, we express this constraint directly on the requirement. We say that a gear is not allowed to change from i to i - 1 and from i - 1 to i within five seconds. This is true. But it is also unhelpful. Consider the sequence (t_i, g_i) where t_i is the time and g_i is the gear value): (1, 1), (2, 2), (3, 3), (5, 1). It is ambiguous if the requirement captured this scenario as a violation correctly. When we change the gear a second time (3, 3), we will not have violated the scenario. The requirement says this is not a violation. But when we change the gear again (5, 1), it is not clear if the requirement in CNL has captured this violation. We reason that (1, 1), (2, 2), (3, 3), (5, 1) is a violation according to the requirement in NL, because $i = 1$ within five seconds, but not according to the CNL requirement, because (2, 2) is the second gear shift, and it is ambiguous whether the subsequent gearshifts (3, 3) and (5, 1) are handled by the requirement with only two variables. An alternative could be to specify multiple requirements, each dealing with a sequence between one and five gearshifts, but this is impractical.

We argue that it is not possible to express requirement R6 in CNL Marko. The first part of the requirement is comparable to requirement R5. This means that we can not express this requirement in CNL Marko.

We argue that it is not possible to express requirement R7 in CNL Marko. In CNL Marko, we can express the constraint that the speed should be higher than the previous speed for at least two seconds. But consider this sequence (t_i, g_i, s_i) where t_i is the time, g_i is the gear value and s_i is the speed: (1, 1, 30), (2, 2, 35), (3, 2, 40), (4, 1, 35). What is the previous speed and speed in this sequence? According to the requirement, it is all of them. But in CNL Marko, we have only one speed and previous speed. Which one do we select? Is it 30? 35? 40? 35? We do not know.

We argue that requirement R8 can be expressed in CNL Marko. As we do not distinguish between hot and cold events, i.e. we assume that all events are in hot mode, the CNLs are able to express this requirement. Furthermore, we think it is trivial to express this requirement in CNL Marko.

In summary, the main limitations of CNL Marko are that:

—it is not possible to identify the person or system who initiates the interaction, e.g. the user performs an action. Rather, the language describes that a parameter has changed or that a state is triggered. In MSD, we use actors to identify who initiates and responds to interaction.
—it is not possible to describe multimodal behaviour, i.e. to distinguish between mandatory and optional behaviour. Instead every action is described as mandatory. In MSD, we can describe multimodal behaviour.
—it is ambiguous with respect to how the binding of multiple variables are to be interpreted. If we take a scenario with two gearshifts during a period of five seconds, we have two variables, one for each gear shift, e.g. in requirement R5. When we first make a gear shift, we can store it in the first variable. We call the value of the first gearshift i. But if the second gear shift is not i - 1, and it happens after two seconds, we satisfy the requirement. The problem is, how do we handle a third gear shift, which could occur between three and five seconds after the first gear shift? Since the requirement is satisfied, it would not be possible to detect a violation if indeed a third gearshift is i - 1. One workaround solution would be to specify five different scenarios to describe the behaviour of two, three, four, five and six gear shifts, respectively. This is impractical.

6.5 Expressing the requirements in CNL Vierhauser

We could not express any requirement of the WLTP requirement set in CNL Vierhauser. We could express all of the requirements of the Daimler-Chrysler instrument cluster requirement set in CNL Vierhauser.

We argue that it is not possible to express requirement R1 in CNL Vierhauser, because it does not support multiple trigger events and the distinction between mandatory and optional behaviour. We need to say that you may sometimes enter an acceleration phase within one second after disengaging the clutch and selecting the first gear, because we do not want to enforce the user to enter acceleration phase when the first gear is selected. Another way we could express requirement R1 in CNL Vierhauser is using past occurrences, e.g. "if acceleration_phase then clutch_disengaged, first_gear_shift must have occurred previously in the last 1 seconds". However, this is also incorrect because requirement R1 says "with the clutch disengaged" [United Nations Economic and Social Council 2013, p. 72], not that the events clutch_disengaged and first_gearshift happen within one second before acceleration_phase. We could disengage the clutch at any time. Also, we do not think that using past occurrences is a good approach to specify requirements if we aim to transform them into MSD. MSD is not capable of expressing that if some events occur, then some other events have occurred previously, as far as we know. Instead, we say that if some events occur, then some other events will occur. Therefore, we would prefer to use only future occurrences for expressing requirements in CNL Vierhauser.

We argue that requirement R2a is not possible to express in CNL Vierhauser because you can not express addition and it also says that whenever you enter acceleration phase, you must change gear.

We argue that it is not possible to express requirement R2b in CNL Vierhauser because it says that whenever you enter acceleration phase, you have to switch gear. This does not capture the requirement.

We argue that it is not possible to express requirement R4 in CNL Vierhauser because we can not negate a sequence.

We argue that it is not possible to express requirement R5 in CNL Vierhauser, because it does not allow the specification of arithmetic expressions and negated sequences. The violation of never changing gear to a gear less than the minimum gear can be expressed in a data constraint which would say that "any gear is changed such that the gear is above the minimum possible gear". We have seen an example of a data constraint in Section D.3, where we say that if the speed is less than 1 km/h, then we are standing still.

We argue that it is not possible to express requirement R6 in CNL Vierhauser. The first part of the requirement is comparable to requirement R5. This means that we can not express this requirement in CNL Vierhauser. The engine speed does not drop below the minimum possible engine speed is possible to express with a data constraint, similar to our discussion of specifying requirement R5.

We argue that it is not possible to express requirement R7 in CNL Vierhauser, because we cannot express multiple events with data. Since this requirement cannot be expressed with a single event, we need to use multiple events.

We argue that requirement R8 can be expressed in CNL Vierhauser. As we do not distinguish between hot and cold events, i.e. we assume that all events are in hot mode, the CNL is able to express the requirement. Furthermore, we think it is trivial to express this requirement in CNL Vierhauser.

In summary, the main limitations of CNL Vierhauser are that:

—it is not possible to identify the person or system who initiates the interaction, e.g. the user performs an action. Rather, the language describes that a parameter has changed or that a state is triggered. In MSD, we use actors to identify who initiates and responds to interaction.

—it is not possible to describe multimodal behaviour, i.e. to distinguish between mandatory and optional behaviour. Instead every action is described as mandatory. In MSD, we can describe multimodal behaviour.

—it is not possible to specify basic arithmetical expressions, e.g. $i \pm 1$ in CNL Vierhauser.

—it is not possible to negate sequences of events.

—it is not possible to distinguish between events and constraints in CNL Vierhauser, which would be useful to model which events must not occur during an event sequence. We could furthermore use constraints to specify which events will interrupt a sequence if they occur in a negated event sequence. However, this would drastically change the semantics of CNL Vierhauser, since such constraints would be a whole new construct of CNL Vierhauser. Furthermore, CNL Vierhauser is a language for modelling constraints on a sequence of events. Constraints on constraints on sequence of events is a whole new language which would not correspond with the semantics of CNL Vierhauser.

6.6 Expressing the requirements in revised CNL Vierhauser

We could express some of the WLTP gear shift requirements in revised CNL Vierhauser. We could express all of the Daimler-Chrysler instrument cluster requirements in revised CNL Vierhauser. Below we have a discussion on each one of the requirements:

We argue that it is possible to express requirement R1 in revised CNL Vierhauser. Because we changed CNL Vierhauser to allow multiple trigger events and distinguish between mandatory and optional behaviour, we can say that you may sometimes enter acceleration phase within one second after disengaging the clutch and selecting the first gear. While this is true, it still leaves room for interpretation of what happens if we decide to first select the first gear and then disengage the clutch. In that case, we have not completely captured the behaviour. However, we still feel that the captured CNL requirement is reasonable and has partially captured the intended behaviour. Since the interaction is existential, i.e. you must have switched gear at most one second before beginning acceleration phase and this must happen at least once, it is believed that revised CNL Vierhauser is able to express this requirement correctly.

We argue that requirement R2a is possible to express in revised CNL Vierhauser. Because revised CNL Vierhauser makes a distinction between mandatory and optional behaviour and supports arithmetic expressions and negated sequences, we can now specify that a gear may sometimes change and when it does, we must never change gear if the selected gear is not equal to the previous gear $i \pm 1$, because this is a violation of the scenario. We read "may sometimes not occur" as a negation in cold mode (existential mode), this is the same as asserting that something does not exist.

We argue that it is possible to express requirement R2b in revised CNL Vierhauser, because we may say that if we enter acceleration phase and the gear is changed, then we must never change to another gear if there is less than 3 seconds since the last time the gear has changed.

We argue that it is possible to express requirement R4 in revised CNL Vierhauser, because we may use a negated sequence in cold mode to specify that we may never stop acceleration, change gear and then go into deceleration.

We argue that it is possible to express requirement R5 in revised CNL Vierhauser, because we may say that if we change gear to i, then change the gear to $i + 1$ and change the gear to i, if this happens in less than five seconds, we have violated the scenario. The violation of never changing gear to a gear less than the minimum gear can be expressed in a data constraint, which is similar to our discussion on specifying requirement R5 for CNL Vierhauser.

We argue that it is possible to express requirement R6 in CNL Vierhauser. The first part of the requirement is comparable to requirement R5. This means that we can express this requirement in revised CNL Vierhauser. The second part of this requirement is a bit more tricky and requires specification of negation of limited recurring behaviour. The CNL does not allow for the expression of such recurring behaviour in a straight forward way. We could theoretically state that if we enter the extra high speed phase, change the gear to i, then i - 1 and change the gear to i, in 3 seconds and repeat this statement three times, then the scenario is violated. However, we would need to specify that for low, medium and high speed cycle as well, making it very inefficient. We argue that it is possible, but impractical to specify requirement R6 in revised CNL Vierhauser. The engine speed does not drop below the minimum possible engine speed is possible to express with a data constraint, similar to our discussion on specifying requirement R6 for CNL Vierhauser.

We argue that it is not possible to express requirement R7 in revised CNL Vierhauser. Since this requirement cannot be expressed with a single event, we need to use multiple events. In revised CNL Vierhauser, we come close to expressing the requirement correctly. Here it is much more clear that we have multiple instances of speed and previous speed, but we cannot say that the entire sequence of first switching gear, then increasing the speed, switching gear to i + 1, increasing the speed, switching gear back to i and increasing the speed takes two seconds. We cannot say that a gear shall be used for at least two seconds in revised CNL Vierhauser and at the same time negate this sequence and expect to get meaningful results.

We argue that requirement R8 can be expressed in revised CNL Vierhauser, because we specify events in hot mode.

In summary, Revised CNL Vierhauser was a great improvement in terms of expressing the WLTP requirements in CNL. It added expressions, the ability to distinguish between hot and cold events and negation. What it does not solve, however, is the ability to identify the origin of events, to distinguish between messages and constraints, to negate individual events in a sequence, which would have been useful to express requirement R7 and to express recurring, inter-event, i.e. recurring sequences within a sequence, behaviour in a meaningful way, which we discussed when specifying requirement R6 in Revised CNL Vierhauser. We motivated why we did not add the ability to make a distinction between messages and constraints in the last section.

6.7 Expressing the requirements in MSD

We could express most of the requirements of the WLTP requirement set in MSD. We could express all of the requirements of the Daimler-Chrysler instrument cluster requirement set in MSD.

We think it is possible to express requirement R1 in MSD. We simply have to make messages between two actors and add a timer to ensure that the interaction is performed in one second after selecting first gear and disengaging the clutch. Just as we stated in the previous paragraph, the order is important, you first disengage the clutch and then select the first gear, not the other way around. We also place a constraint on the scenario, saying that if the clutch is engaged, we abandon the scenario. If we abandon a scenario, there will not be a violation. It is allowed to engage the clutch in the scenario, and this should not be a violation of the requirements.

We argue that requirement R2a is possible to express in MSD. The tricky part is that it is not required that a gear is switched because we enter the acceleration phase. This means that all messages are in cold mode. Another problem is to express that it is only possible to change gear to $i \pm 1$. In MSD, we can only specify behaviour that must always or sometimes happen. We have to use a negated sequence to specify that some message must not happen. We say that gear may be changed after initiating the acceleration phase, and having changed gear again, we expect that the new gear is equal to $i \pm 1$. Otherwise, the scenario will be violated.

We argue that requirement R2b is possible to express in MSD. For this to work properly, you need to use multiple trigger events or pre-conditions. Otherwise, the scenario will be wrong. If we do not use multiple trigger events, the scenario will set the clock to zero after the scenario goes from being deactivated to becoming active, which is after the acceleration phase begins. This is not what the requirement says as it should happen after acceleration has started and the gear has been selected. Furthermore, the interaction is non-mandatory, as we do not have to switch gear at all. But, if we change gear, then there is a violation if the clock is less than 3 seconds.

We argue that it is possible to express requirement R4 in MSD. It is a trivial task to do that. We simply state that if we stop the acceleration phase, switch gear and enter deceleration phase, this is a violation.

We argue that it is possible to express requirement R5 in MSD. We express this scenario as saying if we switch gear to i, then change the gear to j and change the gear to i, we see if this happens in less than five seconds, we have violated the scenario.

We argue that it is possible to express requirement R6 in MSD, because we can state that if we enter extra high speed phase, change the gear to i, then i - 1 and change the gear to i, in 3 seconds and repeat this statement three times, then the scenario is violated. We would need to specify this for low, medium and high speed cycle as well.

We argue that requirement R7 is possible to express in MSD. For the sequence of increasing speed and switching the gear, we simply state that the speed must increase after every gear shift and the gears should be changed from i to i + 1 and from i + 1 to i. We start the timer between the second and third gear shift to establish whether the gear is needed for at least 2 seconds. If we consider the sequence (1, 1, 30), (2, 2, 35), (4, 1, 36). This sequence will violate the scenario. However, (1, 1, 30), (2, 2, 35), (3, 2, 40), (4, 1, 41) will not violate this scenario. Because we execute the third gear shift (3, 2, 40), we will conclude the scenario with saying that no violation will occur. When (4, 1, 41) occurs, we enter another scenario. For this to work, we thus need two scenarios, not one. In this case, we need to specify four gear shifts instead of three.

We argue that it is possible to express requirement R8 in MSD. This is trivial. We simply state that if the ignition is activated, we must enable the instrument cluster.

Transformation

We argue that the transformation works correctly for some of the requirements in the WLTP gear shift requirement set. For some requirements, we could not express them in MSD and/or in CNL. We argue that the transformation works correctly for all of the requirements in the Daimler-Chrysler instrument cluster requirement set.

We argue that the transformation works correctly for requirement R1. The generated scenario is identical to the scenario we just described, but we could not derive that a constraint should be placed on the scenario (interrupt if [

gs.clutchDisengaged != true]). We do not think it is reasonable to have a deterministic transformation make an assumption that this should be the case for every event in the CNL. Without distinguishing between messages and constraints in the CNL, we do not think there is any way a deterministic transformation can derive the difference between these types of model elements (constraint and message). We believe this can be addressed by manual intervention.

We argue that the transformation does not transform requirement R2a into a scenario correctly. It says if the gear is changed, we bind the current gear to i and if the gear is changed, we have to check that the new gear equals $i \pm 1$. However, there is no interruption if the gear is changed to $i{\pm}1$, thereby forcing the violation to be triggered. Otherwise, with a small (manual) correction, this is handled correctly.

We argue that the transformation handles requirement R2b correctly, with the interpretation that no gear shift is allowed at all during three seconds after the first time the gear has changed. If we decide to select gear, even if it is the same gear, we will violate this scenario. We do not have an invariant that says selected_gear != selected_gear_2, because we do not think that it makes sense to allow changing the gear to the same gear. However, if we need to do that, then we could place a constraint saying that selected_gear != selected_gear_2 is forbidden. This will be manually added, as we can not derive this constraint based on the information in the requirements.

We argue that the transformation transforms requirement R4 into a play-out scenario correctly. The generated scenario states that if we stop an acceleration phase, switch gear and enter a deceleration phase, this is a violation.

We argue that the transformation transforms requirement R5 into a complete play-out scenario correctly. The generated scenario says that if we change gear to i, then change the gear to i + 1 and change the gear to i, we see if this happens in less than five seconds, we have violated the scenario. We can also change the CNL requirement to say change the gear to j instead of i + 1.

We are not sure if our transformation would allow us to transform requirement R6 correctly. We think that it would produce a correct scenario, but because we argue that the specification of the requirement is impractical, we did not specify this requirement in CNL.

As we could not express requirement R7 in CNL, we do not know if the transformation transforms this requirement correctly. But, based on our understanding, we argue that this requirement would not possible to transform into MSD, because we need to define two scenarios, not one. We do not think there is any way to deterministically transform a requirement into two scenarios based on revised CNL Vierhauser, since we have to do it for every requirement and the former requirements did not require us to create two scenarios. We would also not be able to derive that only a subset of events in the requirement should be performed for at least two seconds, as opposed to the entire set of events, in revised CNL Vierhauser.

We argue that the transformation transforms requirement R8 into a play-out scenario correctly. The generated scenario says that if the ignition is activated, we must enable the instrument cluster.

In summary, we discuss the main limitations of the transformation. Firstly, it requires manual intervention, e.g. in R1 and R2. In R1, we need to specify a restriction on the events which may not occur during the activated scenario and will interrupt the scenario if they do. In R1, we have to interrupt the scenario if the gear is engaged. In R2, we want to specify that we want to interrupt the scenario if the gear is shifted to $i \pm 1$. This is handled by adding interrupt if [true] inside the if clauses in the scenario (which are generated by the transformation) in R2a. Also in R2, we want to specify that the second gearshift is different from the first. We do this by adding a constraint on the scenario stating that "selected_gear != selected_gear_2" is not allowed in the activated scenario. Secondly, it is required to add domain-specific information. This is related to the first limitation, as it is also a form of intervention, but on a much larger scale. We described this in Section 5.

7. DISCUSSION

7.1 RQ1: To what extent can behaviour requirements be expressed in selected controlled natural languages?

Based on our evaluation, we have found that the CNLs allow the expression of some behaviour requirements in this study. In this section, we will discuss the benefits and limitations of the CNLs.

CNL Marko seems to be a very potent language. It is a DSL with many constructs and an amazingly sophisticated array of model elements available. It seems to be applicable to modelling state-based systems very well. Clearly, its applicability does not extend well to finite recurrences and the distinction between universal and existential behaviour. Furthermore, it also does not support the specification of actions in a list of pre-conditions, so we always consider state and parameter input. That means we can not identify who has interacted, e.g. the user. This would have to be added later on if we were to transform the requirements. Since it does not support the difference between universal and existential behaviour, as our evaluation shows, it is not useful to try to convert the requirements expressed in CNL into play-out scenarios.

CNL Vierhauser was a very plain language and it was a surprising finding that it was this basic, because it had been used in practice to model monitoring of events. We found that it was not possible to express any of the WLTP requirements in CNL Vierhauser. We thus had to make changes to CNL Vierhauser and give it a bit more expressiveness to be able to express these requirements. This amount of expressiveness was not available in CNL Marko, e.g. we could not accurately represent the required behaviour of the requirements in this study without a distinction between mandatory and optional behaviour and furthermore, a significant improvement of revised CNL Vierhauser over both CNL Marko and CNL Vierhauser is the ability to specify multiple events with data. For instance, we could specify that we shall switch gear and to begin an acceleration phase in the same scenario. In CNL Marko, multiple actions can only appear in the pre-condition list, not in the part which states what the system shall do if all of the pre-conditions of a requirement are matched. However, we did not change the overall specification approach with revised CNL Vierhauser. For instance, CNL Vierhauser can specify multiple events, without data. The semantics of CNL Vierhauser are largely preserved. We prefer the event-based approach in modelling MSD specifications. This was a natural approach to describe interaction and forces explicit specification of the timing of interaction. This restriction is a very good thing if we want to use it for transformation, as it allows no room for interpretation as to when something must happen, because we specify events that happen in relation to other events. CNL Marko was very expressive and did not impose any restrictions on how to specify timing of events, so there are a number of different ways we could do that.

Based on the number of changes needed in order to specify requirements in CNL Vierhauser, as described in Section 4.4, we might have done better to begin with a slightly better choice of CNL. But we still prefer the overall specification approach of using constraints on sequences of events. DSLs are designed to be changed and evolved in order to fit the needs of a particular domain [Bettini 2013]. This is in line with Vierhauser et al. [2015]. I would stress that the changes are not to the extent that we changed the overall meaning of the language, and its applicability to its intended domain, but we managed to adapt it to our domain as well.

Furthermore, with CNL Marko, we get the perspective that requirements are written from the perspective to specify unambiguous requirements that both the customer and the development company can understand. CNL Marko allows for many different ways of specifying a requirement. Which is good if you are specifying unambiguous requirements. We want to be a bit flexible. We want to be a bit unspecific at times. We want to leave out things which we do not care about. With CNL Marko, we believe that there are many different ways in which we could express a requirement. But it is not good if we need this information for the transformation to transform the requirements correctly. For the transformation to transform requirements correctly, we somehow have to define a mapping between model input element to another model output element. Therefore, some model elements are used for creating variables, for instance. If we do not need to specify this in CNL Marko, there is no way we can derive a complete play-out scenario with our transformation. Therefore, when specifying requirements, we need to make sure that they look like a specific format, with a defined meaning on the model elements. Therefore, CNL Marko allows requirements to be incompletely specified, while still correct, they miss important details and that does not facilitate transformation into platform independent models correctly. We need to be precise about what information we give to our transformation. We can not simply say "we will do this for at least 3 seconds", we need to be much more precise. We need to say that if some thing happens, then another thing happens in three seconds at some times but other times it does not happen. CNL Marko does not support such unambiguity in the timing of events and how often a thing happens.

CNL Marko supports expressing arithmetic expressions and conditions. We think that CNL Marko would be useful to model MSC specifications. We also think that the distinction between actions and conditions would allow us to distinguish between messages and constraints. This is something we could not manage with CNL Vierhauser.

In conclusion, using either CNL is an improvement over natural language requirement specification as a process because we reduce underspecification and ambiguity while we increase the representational consistency of the requirements, since syntactically correct CNL requirements must have certain elements specified, in a certain order, and with a well defined meaning of each element that is specified. Based on our understanding of the CNLs and the behavioural requirements, CNL Marko seems to an apt choice for structure and organization, enforcing each requirement to possess a unique requirement identifier. In CNL Vierhauser, there is no possibility to label a requirement at all. From a requirement engineer's viewpoint, this is a significant advantage of CNL Marko over CNL Vierhauser, especially if there are many requirements to keep track of. For instance, we could imagine that there would be a significant strain of using CNL Vierhauser, for this reason, in a large-scale requirement engineering process. Furthermore, CNL Marko supports arithmetical expressions. We have for instance seen that we want to be able to specify gearshift increments for our behavioural requirements, which CNL Vierhauser does not support without modification. Not all systems need to be specified with arithmetical expressions, e.g. traffic lights are modelled with categorical variables of "red", "yellow" and "green". In vehicular systems, however, we use numerical variables such as speed and gearshift and without the ability to specify arithmetical expressions, CNL Vierhauser is inadequate for modeling behaviour in systems where numerical variables shall change. There are many examples of such systems, e.g. a global positioning system measuring the speed and direction of traffic, a weather forecasting system, a missile defense system etc. CNL Vierhauser is designed for monitoring events, and for that domain, it seems to be apt, e.g. it would be able to describe the behaviour of a traffic light system, including modeling the switching of traffic lights within a specific time and the arrival of vehicles and pedestrians which shall trigger a suitable response from the system. In such a system, we argue that it would be cumbersome to use CNL Marko, which nevertheless would be able to model the same behaviour. CNL Marko would be suitable for describing the decomposition of a system into subsystems, the interface descriptions of each subsystem and the internal behaviour in a subsystem, e.g. it would be possible to describe a requirement specification such as Fockel and Holtmann's illustrative application [2014] with CNL Marko.

7.2 RQ2: To what extent can behaviour requirements be expressed in modal sequence diagrams?

When we selected the behaviour requirements in this study, we did not know whether they were possible to express in MSD. We were interested to know which requirements we could not express in MSD so we could explain why we did not transform some requirements into MSD. It was important to establish this fact to reduce the ambiguity in the analysis step of this thesis. In case a requirement was not transformed into MSD, we would like to find out whether it was because it was not possible to express the requirement in MSD or if the transformation could not transform the requirement into MSD. Therefore, we tried to express every requirement in MSD. We had assumptions that every requirement would be possible to express in MSD, because the scenario language is use case-based and we assume that behaviour is naturally possible to express in a scenario, e.g. you do something and something else must happen. You must not do something until another thing happens. etc. Furthermore, there are many constructs in an MSD, including support for boolean operators and arithmetic expressions. We know that it is possible to express complicated conditions in MSD, as well as trivial arithmetic expressions.

We are pleased that our assumptions hold for most of the requirements. We found that modal sequence diagrams are able to express most of the requirements in this study. Only a few requirements were not possible to express in MSD. The problems of expressing these requirements are that they are not easily externalized into a formula, which we did not know how to exactly specify as a condition that this event must happen. When you make time frequencies, like a set of events must not have occurred in the last five seconds, it is not well documented how to express this in MSD. Our understanding is that MSD cannot explicitly express requirements like this.

We can only say that we argue that they are correct and what cannot be expressed in MSD, based on the interpretation of the requirements. This does not mean we have a right interpretation. However, since we publish our requirements in MSD, such misunderstanding can be identified.

Because the behaviour requirements are not representative, we can not say that a behaviour requirement is always possible to express in MSD. We have seen examples where we argue behaviour requirements can not be expressed in MSD.

7.3 RQ3: To what extent can a deterministic transformation facilitate transformation of the behaviour requirements expressed in controlled natural language into modal sequence diagrams?

In the background, we discussed different transformation approaches. We did not find any other comparable work. But we did find a study which addressed a similar problem. Somé [2006] transformed use cases into MSC. The main difference between his work and this thesis is that the requirements contained enough information by themselves to derive a complete simulation model. Furthermore, MSC is extremely different from MSD. For instance, MSC does not have liveness conditions.

In this study, we were not able to convert all requirements into MSD. However, we are able to convert some requirements. These requirements follow pretty much the same standard form: If some antecedent events occur, some consequent set of events occur within some period of time (or infinite time). We added hot and cold events (difference between may sometimes and must always occur) in order to distinguish between the safety and liveness properties in MSD. But, the antecedent-consequent specification of requirements differs slightly from a use case scenario. A use case scenario lists pre- and postconditions, and each step can have alternative interaction, which means that there is enough information to derive complete sequences of interactions much more easily than what we could achieve with CNL Vierhauser. A scenario also has an actor identifying the origin of an event. We say that because CNL Vierhauser was unable to express this kind of information, we were unable to deterministically transform the requirements based only on the requirements. Instead, we need to use a domain-specific information, saying where the events come from, and what message in the domain model a specific event corresponds to.

Furthermore, we can make no difference between hot and cold without somehow specifying that a constraint is in a *mode*. Without this specification, one can transform all requirements into events in either cold or hot mode. In the transformation, we decided to use events in cold mode by default. We did argue in the beginning that the default would be the hot mode because it takes advantage of the mandatory characteristics of requirement, i.e. a requirement specifies what a system does in certain circumstances. However this is not always correct in MSD, because e.g. as semantically was realized in requirement R1. If some event happens, another could (but not strictly all the time, i.e. in hot mode) happen. The problem with CNL Vierhauser was that it can not express this information, and there is nothing we can do about this in the transformation. Nonetheless, even cold events are required to be satisfied at one point. The difference between hot and cold events has important semantics. We therefore decided to add the ability to express requirements as must always and may sometimes occur. The latter in revised CNL Vierhauser is non-optional, it still must happen, just not every time.

We use a built-in logic in the transformation to distinguish between a variable binding and reference. The first time a data item is encountered in the requirement, a parameter binding is created (i.e. bind to). The second (to n$^{\text{th}}$) time a data item with the same path name is encountered, e.g. selected_gear in requirement R2, a reference to the variable is used instead. This is the logic that you can't define multiple variables without getting the semantics wrong. This is the case for R2, but not necessarily everywhere. The alternative would be to somehow specify this information in the requirements, and it is possible to do so, regardless of the logic built in the transformation. To do so, one might write (the rather ugly) data item + 0 which will produce a reference to variable + 0. The transformation here works because a variable must exist in order to be able to conduct arithmetic expressions, i.e. you can not sum the binding of a variable and e.g. a number in MSDs, so the transformation works under the assumption that the data item is a reference (which of course can be used in arithmetics), that the variable reference is valid, i.e. the variable has been defined and initialized with proper data before and thus references that variable instead of making a binding (which would make the output invalid as a simulation model).

8. CONCLUSION

Based on our discussion, we can conclude that there are a number of desirable features of a DSL that should be possible to specify in order to support the transformation of semi-formal requirements into MSD:

- —Distinguish between universal and existential behaviour.
- —Distinguish between messages and constraints.
- —Use a pre-condition list.
- —Use a future approach. Looking at the events that have happened in the past is not going to be suitable for transforming the requirements into MSD.

We say a list of desirable features of a DSL, but this list is a bare minimum in order to facilitate a deterministic transformation approach into MSD.

RQ1: To what extent can behaviour requirements be expressed in selected controlled natural languages?

Based on the discussion, we found that it is possible to specify most of our requirements in revised CNL Vierhauser. We found that it is possible to express some of the WLTP gear shift requirements in CNL Marko. We found that it is possible to specify all of the DC instrument cluster requirements in CNL Marko and CNL Vierhauser (standard and revised).

RQ2: To what extent can behaviour requirements be expressed in modal sequence diagrams?

Based on the discussion, we found that it is possible to specify most of our requirements in MSD.

RQ3: To what extent can a deterministic transformation facilitate transformation of the behaviour requirements expressed in controlled natural language into modal sequence diagrams?

Based on the discussion, we found that it is possible to transform most of the WLTP gear shift requirements. We found that it is possible to transform all of the DC instrument cluster requirements.

Maturity

Gregor and Hevner have classified the maturity of a research contribution in design science research according to the level of maturity of the (i) solution and (ii) application domain [2013, p. 345]. The authors talk about invention, improvement, exaptation and routine design.

Invention: Inventions are novel and involve studying an area where existing research is rather limited. Problems may not be properly defined and researched in earlier studies etc. Gregor and Hevner state that "a key contribution is the conceptualization of the problem itself." [2013, p. 346]. Often, according to the authors, innovations are on the artifact level. The authors state that no innovations have ever (to their knowledge) started out as a theory, and argue that theories can only develop once we have gathered enough information about the problem: "such theories have to build on an accumulation of knowledge about a range of artifacts all addressing the same problem domain — which means that the problem is known and, thus, not in the invention quadrant." [Gregor and Hevner 2013, p. 346].

Improvement: Improvements are novel in the sense that they improve on existing solutions to a design problem. The problem is defined and well-known. Often, according to the authors, improvements are on the artifact level.

Exaptation: Exaptations are novel in the sense that the application of a solution is known (from one domain), but transferred into a new situation where problems may or may not be prevalent.

Routine design: Routine designs are, as the name implies, familiar solutions to familiar problems. The authors suggest that routine designs may sometimes identify potential research problems, which one can explore further and then it ideally becomes one of invention, improvement and exaptation.

One thing to bear in mind of the above scheme is that it organizes contributions into exactly one out of four categories. On philosophical grounds, we would argue that a research contribution could be in one of the four quadrants (in the 2x2 matrix) and gravitate towards another quadrant. For instance, routine designs are a good example of this. Initially, they may be classified as routine designs, but later on, as the research moves on, depending on their directions, they need to move into one of the other quadrants, i.e. invention, improvement or exaptation. Also, a research contribution may well be within the boundary of two or more quadrants at the same time, because the definition of maturity, i.e. when does novel becomes familiar? and vice versa can become rather difficult to sort out. With this said, we believe the research contribution in this study may be classified as a kind of exaptation, with progression towards innovation, as the conceptualization of the problem defined in the study is by itself a contribution. In addition, since the problem is novel and interesting, searching for relevant literature is naturally difficult. That adds further support to our claim that this study is progressing towards being an innovation — but not necessarily. We therefore make the more lenient and humble claim that the study is in the exaptation quadrant.

General challenges of model-driven engineering

Van Der Straeten et al. have identified challenges of model-driven engineering (MDE) [2009]. There are mainly three research challenges which are relevant to this study:

- Requirements modelling
  - How can we model requirements?
  - How can we bridge the gap between informal (textual) requirement specifications and formal requirement models?
  - How can we integrate the activity of requirement specifications into traditional modelling? [Van Der Straeten et al. 2009, p. 39]

Firstly, "How can we model requirements?" is a challenge which is covered in a related work by Harel and Marelly [2003], Damm and Westphal [2005], Harel and Maoz [2007], Brenner et al. [2014] and outlined in Section 2.1. Secondly, "How can we bridge the gap between informal (textual) requirement specifications and formal requirement models?" is a challenge which is covered in a related work by de Almeida Ferreira and Rodrigues da Silva [2009], Baudry et al. [2007], Fockel and Holtmann [2014] and Marko et al. [2015] and is outlined in Section 2.3. Thirdly, "How can we integrate the activity of requirement specifications into traditional modelling?" is a challenge which is covered in a related work by Fockel and Holtmann [2014] and is outlined in sections 2.3 and 2.4. The main contribution of this study is to derive a platform-independent model according to the definition of a play-out simulation model based on the information which is stored by behaviour requirements. This contribution is central to alleviate the progress towards "integrating the activity of requirement specification into traditional modelling".

8.1 Concluding remarks

By specifying requirements in a semi-formal notation, and then translating them into a formal notation, we can ease the adoption of requirement simulation in organisations that would otherwise not be able to adopt formal methods in software engineering. We propose that the approach can be used in different organisations in industry and adopted to different contexts and ways of working in industry. For instance, one can select a CNL that is apt to model the behaviour of a global positioning system for tracking vehicles on the street for companies specializing in this field. Being able to detect inconsistency in a specification early on allows to minimize waste, i.e. nonessential efforts and also helps to correct the requirement specification. Because we can now learn what scenarios contradict each other and in what way do they end up being unsatisfiable as a unit. It increases the ability of requirement engineers to find inconsistency. With this ability, problems can be detected and remedied. This is especially useful for large-scale requirement engineering, where a manual process of detecting inconsistency is unfeasible. It is particularly important for safety-critical domains, where there could be serious consequences of an anomalous specification, such as a loss of life, a loss of reputation or a class action civil lawsuit.

8.2 Research directions

We elaborate on potential research directions we think are beneficial to future research based on the results of this study.

It would be useful to validate this research in industry. For this reason, we argue that future researchers should attempt to use the transformation approach (or write their own based on a similar approach) and validate the results of this study in industry. Figure 8.1 depicts a simple version of a model for industry research collaboration by Gorschek et al. [2006]. We use it to describe where we are at this time in the research.

Right now, we are in a position where we consider a **Candidate solution** of a transformation, which we have tried in academia. We are not yet convinced that the results of this study allow for further exploration into a real, industry-type project. The point here is that the research is not yet validated in a real industrial setting. Having performed an evaluation with a toy example is called **Validation in academia**. This should be done with some research-oriented direction from industry, e.g. by formulating realistic requirements on the research. This we have already described in the section 3. If we were convinced of the suitability of the transformation, the next step would be to try the solution in a real

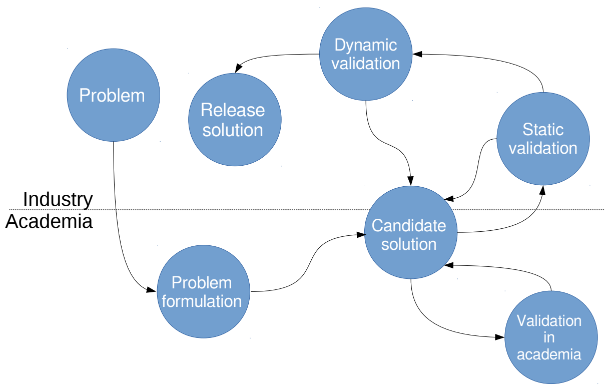


Fig. 8.1. A simplified version of a model for technology transfer advocated by Gorschek et al. [2006, p. 5].

setting, which is called **Static & Dynamic Validation**. This could for instance be interviews with project members and trying the transformation on more requirements (which are representative of an industrial setting). By doing so, we would gain knowledge and insights on the suitability of the approach in real practice. Finally, we would release the approach to the industry, i.e. introduce it as a viable option for requirements engineers in practice. This is what Gorschek et al. mean when they say to *transfer technology* to industry.

ACKNOWLEDGMENTS

Grischa Liebel was the driver of the thesis, coming up with the idea and helped carrying out the execution of the research study. He is an expert in play-out simulation with ScenarioTools in safety-critical requirements engineering and has been a major source of influence on this project since its inception. During the course of the research, he has supervised and offered guidance on research directions, practical problems with the tooling, theories, research papers, proofreading, formulation of the research questions and recommendation of the process methodology.

I also would like to extend my gratitude to those who haven't directly been involved with the research study. Yet, these people have had a profound impact on the outcome of it. First, I would like to acknowledge that Joel Greenyer has published all of his publications online at jgreen.de. Since ScenarioTools is largely undocumented, reviewing several of his co-authored publications helped to acquire a better understanding of his work (which is highly intertwined with the thesis).

REFERENCES

B. Baudry, C. Nebut, and Y. L. Traon. 2007. Model-Driven Engineering for Requirements Analysis. In *Enterprise Distributed Object Computing Conference, 2007. EDOC 2007. 11th IEEE International.* IEEE, Piscataway, New Jersey, USA, 459–466. DOI:http://dx.doi.org/10.1109/EDOC.2007.15

K. Beck. 2000. *Extreme Programming Explained: Embrace Change.* Addison-Wesley, Boston, Massachusetts, USA.

B. Berenbach. 2012. A 25 Year Retrospective on Model-Driven Requirements Engineering. In *Model-Driven Requirements Engineering Workshop (MoDRE), 2012 IEEE.* IEEE, Piscataway, New Jersey, USA, 87–91. DOI:http://dx.doi.org/10.1109/MoDRE.2012.6360078

L. Bettini. 2013. *Implementing Domain-Specific Languages with Xtext and Xtend.* Packt Publishing, Birmingham, United Kingdom.

M. Brambilla, J. Cabot, and M. Wimmer. 2012. *Model-Driven Software Engineering in Practice.* Morgan & Claypool Publishers, San Rafael, California, USA. DOI:http://dx.doi.org/10.2200/S00441ED1V01Y201208SWE001

C. Brenner, J. Greenyer, J. Holtmann, G. Liebel, G. Stieglbauer, and M. Tichy. 2014. ScenarioTools Real-Time Play-Out for Test Sequence Validation in an Automotive Case Study, In Proceedings of the 13th International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2014). *Electronic Communications of the EASST* 67 (2014), 1–14. http://jgreen.de/wp-content/documents/2014/ScenarioTools-Real-Time-Play-Out-for-Test-Sequence-Validation-Automotive-CaseStudy-GT-VMT14.pdf

C. Brenner, J. Greenyer, and V. Panzica La Manna. 2013. The ScenarioTools Play-Out of Modal Sequence Diagram Specifications with Environment Assumptions, In Proceedings of the 12th International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2013). *Electronic Communications of the EASST* 58 (2013), 1–14. http://jgreen.de/wp-content/documents/2013/Brenner-Greenyer-Panzica-scenariotools-playout-w-environment-assumptions-GTVMT2013.pdf

F. P. Brooks, Jr. 1987. No Silver Bullet: Essence and Accidents of Software Engineering. *Computer* 20, 4 (April 1987), 10–19. DOI:http://dx.doi.org/10.1109/MC.1987.1663532

K. Buhr, N. Heumesser, F. Houdek, H. Omasreiter, F. Rothermel, R. Tavakoli, and T. Zink. 2003. DaimlerChrysler Demonstrator: System Specification Instrument Cluster. (2003). http://www4.in.tum.de/lehre/vorlesungen/re/ws10/uebung/instrumentcluster_b.pdf [Online; accessed 2016-06-01].

W. Damm and D. Harel. 2001. LSCs: Breathing Life into Message Sequence Charts. *Formal methods in system design* 19, 1 (2001), 45–80. http://www.wisdom.weizmann.ac.il/~dharel/SCANNED.PAPERS/LSCs.pdf "(Preliminary version in Proc. 3rd IFIP Int. Conf. on Formal Methods for Open Object-Based Distributed Systems (FMOODS'99), (P. Ciancarini, A. Fantechi and R. Gorrieri, eds.), Kluwer Academic Publishers, 1999, pp. 293-312.)".

W. Damm and B. Westphal. 2005. Live and let die: LSC based verification of UML models. *Science of Computer Programming* 55, 1–3 (2005), 117 – 159. DOI:http://dx.doi.org/10.1016/j.scico.2004.05.013 Formal Methods for Components and Objects: Pragmatic aspects and applications.

D. de Almeida Ferreira and A. Rodrigues da Silva. 2009. A Controlled Natural Language Approach for Integrating Requirements and Model-Driven Engineering. In *Software Engineering Advances, 2009. ICSEA '09. Fourth International Conference on.* IEEE, Piscataway, New Jersey, USA, 518–523. DOI:http://dx.doi.org/10.1109/ICSEA.2009.81

L. M. G. Feijs. 2000. Natural language and message sequence chart representation of use cases. *Information and Software Technology* 42, 9 (2000), 633–647. DOI:http://dx.doi.org/10.1016/S0950-5849(00)00107-5

M. Fockel and J. Holtmann. 2014. A Requirements Engineering Methodology Combining Models and Controlled Natural Language. In *Model-Driven Requirements Engineering Workshop (MoDRE),2014 IEEE 4th International.* IEEE, Piscataway, New Jersey, USA, 67–76. DOI:http://dx.doi.org/10.1109/MoDRE.2014.6890827

M. Fowler. 2010. *Domain-Specific Languages.* Pearson Education, Boston, Massachusetts, USA.

R. France and B. Rumpe. 2007. Model-driven Development of Complex Software: A Research Roadmap. In *Future of Software Engineering, 2007. FOSE '07*. IEEE, Piscataway, New Jersey, USA, 37–54. DOI:http://dx.doi.org/10.1109/FOSE.2007.14

P. Ghazi, A. M. Moreno, and L. Peters. 2014. Looking for the Holy Grail of Software Development. *IEEE Software* 31, 1 (Jan 2014), 93–96. DOI:http://dx.doi.org/10.1109/MS.2014.8

T. Gorschek, P. Garre, S. Larsson, and C. Wohlin. 2006. A Model for Technology Transfer in Practice. *IEEE Software* 23, 6 (Nov/Dec 2006), 88–95. DOI:http://dx.doi.org/10.1109/MS.2006.147

J. Greenyer. 2011. *Scenario-based Design of Mechatronic Systems*. Ph.D. Dissertation. University of Paderborn, Paderborn. http://dups.ub.uni-paderborn.de/hs/urn/urn:nbn:de:hbz:466:2-7690

S. Gregor and A. R. Hevner. 2013. Positioning and Presenting Design Science Research for Maximum Impact. *MIS Quarterly* 37, 2 (2013), 337–A6. http://misq.org/positioning-and-presenting-design-science-research-for-maximum-impact.html [Online; accessed 2015-12-04].

D. Harel and S. Maoz. 2007. Assert and negate revisited: Modal semantics for UML sequence diagrams. *Software & Systems Modeling* 7, 2 (2007), 237–252. DOI:http://dx.doi.org/10.1007/s10270-007-0054-z

D. Harel and R. Marelly. 2002. Playing with time: on the specification and execution of time-enriched LSCs. In *Modeling, Analysis and Simulation of Computer and Telecommunications Systems, 2002. MASCOTS 2002. Proceedings. 10th IEEE International Symposium on*. IEEE, Piscataway, New Jersey, USA, 193–202. DOI:http://dx.doi.org/10.1109/MASCOT.2002.1167077

D. Harel and R. Marelly. 2003. Specifying and executing behavioral requirements: the play-in/play-out approach. *Software and Systems Modeling* 2, 2 (07 2003), 82–107. DOI:http://dx.doi.org/10.1007/s10270-002-0015-5

A. Hevner and S. Chatterjee. 2010. Design Science Research in Information Systems. In *Design Research in Information Systems*. Integrated Series in Information Systems, Vol. 22. Springer US, New York, New York, USA, 9–22. DOI:http://dx.doi.org/10.1007/978-1-4419-5653-8_2

A. R. Hevner, S. T. March, J. Park, and S. Ram. 2004. Design Science in Information Systems Research. *MIS Quarterly* 28, 1 (2004), 75–105. http://www.jstor.org/stable/25148625 [Online; accessed 2015-12-04].

H. Holmström Olsson and J. Bosch. 2014. Climbing the "Stairway to Heaven": Evolving From Agile Development to Continuous Deployment of Software. In *Continuous Software Engineering*, J. Bosch (Ed.). Springer International Publishing, Cham, Switzerland, 15–27. DOI:http://dx.doi.org/10.1007/978-3-319-11283-1_2

J. Humble and D. Farley. 2010. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional, Boston, Massachusetts, USA.

IEEE-SA Standards Board. 1998. *IEEE Recommended Practice for Software Requirements Specifications*. Technical Report IEEE Std 830-1998. The Institute of Electrical and Electronics Engineers.

P. Johannesson and E. Perjons. 2014. Introduction. In *An Introduction to Design Science*. Springer International Publishing, Cham, Switzerland. DOI:http://dx.doi.org/10.1007/978-3-319-10632-8_1

E. Knauss, M. Staron, W. Meding, O. Söder, A. Nilsson, and M. Castell. 2015. Supporting Continuous Integration by Code-Churn Based Test Selection. In *Rapid Continuous Software Engineering (RCoSE), 2015 IEEE/ACM 2nd International Workshop on*. IEEE, Piscataway, New Jersey, USA, 19–25. DOI:http://dx.doi.org/10.1109/RCoSE.2015.11

S. Lauesen. 2002. *Software Requirements: Styles and Techniques*. Addison-Wesley Professional, Boston, Massachusetts, USA.

G. Liebel and M. Tichy. 2015. Comparing Comprehensibility of Modelling Languages for Specifying Behavioural Requirements. In *Human Factors in Modeling (HuFaMo 2015), First International Workshop on*. n.p., n.p., 17–24.

N. Marko, A. Leitner, B. Herbst, and A. Wallner. 2015. Combining Xtext and OSLC for Integrated Model-Based Requirements Engineering. In *Software Engineering and Advanced Applications (SEAA), 2015 41st Euromicro Conference on*. IEEE, Piscataway, New Jersey, USA, 143–150. DOI:http://dx.doi.org/10.1109/SEAA.2015.11

A. Nilsson, J. Bosch, and C. Berger. 2014. The CIViT Model in a Nutshell: Visualizing Testing Activities to Support Continuous Integration. In *Continuous Software Engineering*, J. Bosch (Ed.). Springer International Publishing, Cham, Switzerland, 97–106. DOI:http://dx.doi.org/10.1007/978-3-319-11283-1_8

J. D. Procaccino, J. M. Verner, S. P. Overmyer, and M. E. Darter. 2002. Case study: factors for early prediction of software development success. *Information and Software Technology* 44, 1 (2002), 53–62. DOI:http://dx.doi.org/10.1016/S0950-5849(01)00217-8

S. S. Somé. 2006. Supporting use case based requirements engineering. *Information and Software Technology* 48, 1 (2006), 43–58. DOI:http://dx.doi.org/10.1016/j.infsof.2005.02.006

Telecommunication Standardization Sector of ITU. 2011. *Message Sequence Chart (MSCs)*. Technical Report Z.120. International Telecommunication Union. https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-Z.120-201102-I!!PDF-E&type=items

United Nations Economic and Social Council. 2013. Proposal for a new UN Global Technical Regulation on Worldwide harmonized Light vehicles Test Procedures (WLTP). (2013). http://www.unece.org/fileadmin/DAM/trans/doc/2013/wp29grpe/ECE-TRANS-WP29-GRPE-2013-13.pdf [Online; accessed 2016-01-01].

R. Van Der Straeten, T. Mens, and S. Van Baelen. 2009. Challenges in Model-Driven Software Engineering. In *Models in Software Engineering*, M. R. V. Chaudron (Ed.). Lecture Notes in Computer Science, Vol. 5421. Springer Berlin Heidelberg, Berlin, Germany, 35–47. DOI:http://dx.doi.org/10.1007/978-3-642-01648-6_4

M. Vierhauser, R. Rabiser, P. Grünbacher, and A. Egyed. 2015. Developing a DSL-Based Approach for Event-Based Monitoring of Systems of Systems: Experiences and Lessons Learned (E). In *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*. IEEE, Piscataway, New Jersey, USA, 715–725. DOI:http://dx.doi.org/10.1109/ASE.2015.25

D. Wynn and J. Clarkson. 2005. Models of designing. In *Design process improvement*, J. Clarkson and C. Eckert (Eds.). Springer London, London, England, United Kingdom, 34–59. DOI:http://dx.doi.org/10.1007/978-1-84628-061-0_2

T. Yue, L. C. Briand, and Y. Labiche. 2011. A systematic review of transformation approaches between user requirements and analysis models. *Requirements Engineering* 16, 2 (2011), 75–99. DOI:http://dx.doi.org/10.1007/s00766-010-0111-y

(a) First gear shall be selected one second before beginning an acceleration phase from standstill with the clutch disengaged. Vehicle speeds below 1 km/h imply that the vehicle is standing still;
[(b) Gears shall not be skipped during acceleration phases. Gears used during accelerations and decelerations must be used for a period of at least three seconds (e.g. a gear sequence 1, 1, 2, 2, 3, 3, 3, 3, 3 shall be replaced by 1, 1, 1, 2, 2, 2, 3, 3, 3);]
[(c) Gears may be skipped during deceleration phases. For the last phase of a deceleration to a stop, the clutch may be either disengaged or the gear lever placed in neutral and the clutch left engaged;]
(d) There shall be no gearshift during transition from an acceleration phase to a deceleration phase. E.g., if v(j) < v(j+1) > v(j+2) and the gear for the time sequence j and j + 1 is i, gear i is also kept for the time j + 2, even if the initial gear for j + 2 would be i + 1;
(e) If a gear i is used for a time sequence of 1 to 5 s and the gear before this sequence is the same as the gear after this sequence, e.g. i - 1, the gear use for this sequence shall be corrected to i - 1.
Example:
(i) a gear sequence i - 1, i, i - 1 is replaced by i - 1, i - 1, i - 1;
(ii) a gear sequence i - 1, i, i, i - 1 is replaced by i - 1, i - 1, i - 1, i - 1;
(iii) a gear sequence i - 1, i, i, i, i - 1 is replaced by i - 1, i - 1, i - 1, i - 1, i - 1;
(iv) a gear sequence i - 1, i, i, i, i, i - 1 is replaced by i - 1, i - 1, i - 1, i - 1, i - 1, i - 1;
(v) a gear sequence i - 1, i, i, i, i, i, i - 1 is replaced by i - 1, i - 1, i - 1, i - 1, i - 1, i - 1, i - 1.
(f) a gear sequence i, i - 1, i, shall be replaced by i, i, i, if the following conditions are fulfilled:
(i) engine speed does not drop below n(min); and
(ii) the sequence does not occur more often than four times each for the low, medium and high speed cycle phases and not more than three times for the extra high speed phase.
Requirement (ii) is necessary as the available power will drop below the required power when the gear i - 1, is replaced by i;
(g) If, during an acceleration phase, a lower gear is required at a higher vehicle speed, the higher gears before shall be corrected to the lower gear, if the lower gear is required for at least 2 s.
Example: v(j) < v(j + 1) < v(j + 2) < v(j + 3) < v(j + 4) < v(j + 5) < v(j + 6). The originally calculated gear use is 2, 3, 3, 3, 2, 2, 3. In this case the gear use will be corrected to 2, 2, 2, 2, 2, 2, 3.
Since the above modifications may create new gear use sequences which are in conflict with these requirements, the gear sequences shall be checked twice.

Fig. A.1. WLTP gear shift requirements [United Nations Economic and Social Council 2013, p. 72–73].

APPENDIX

A. REQUIREMENTS USED IN THIS STUDY

A.1 UN WLTP gear shift requirements

The Worldwide harmonised Light Vehicle Test Procedures (WLTP) is a UN proposal to create a standard test procedure for different classes of vehicles, mainly American class 1 (e.g. Dodge Dakota), class 2 (e.g. Ford F-150) and 3 (e.g. Ford E-350) [United Nations Economic and Social Council 2013]. In Annex 2 of the WLTP, there are requirements for shifting gears in manual transmission systems which are used in this study. These requirements are found in Figure A.1.

A.2 Daimler-Chrysler instrument cluster requirements

The Daimler-Chrysler Instrument Cluster Requirements, referred to as the DC instrument cluster requirements from this point on, describe the behaviour of the instrument cluster in a vehicle, including the rev meter, the speedometer, the indicator lights and the display [Buhr et al. 2003]. The DC instrument cluster requirements encompass a number of different perspectives, or viewpoints, including business requirements, user requirements and system requirements. Of particular interest to this study are the user requirements as they are documented in use case scenarios. In addition, the requirements documented as features seem to be possible to be included. Because of the vast amount of information expressed in the features and the use case scenarios of the DC instrument cluster requirements, only some of these requirements are used in study, namely the requirements listed under the three subsections in section 1.1.3. Figure A.2 show features 1-7 and UCs 1-2 from the report by Buhr et al. [2003]. For the interested reader, he or she may refer to the remaining requirements in sections 1.2.3, 1.3.3, 1.4.3, 1.5.1.3, 1.5.2.3 and 1.5.3.3 in the technical report by Buhr et al. [2003].

F-1 Permanent activation when ignition on
After the ignition has been switched on the instrument cluster is activated.

F-2 Deactivation by switching off ignition
Half a minute after the ignition has been switched off the instrument cluster is deactivated and all (warning) lights dim out.

F-3 Permanent activation by setting ignition key in position radio
After the ignition key is set in position radio, the instrument cluster is activated.

F-4 Temporal activation by opening driver's door
After the driver's door has been opened the instrument cluster is activated for half a minute.

F-5 Temporal activation by closing driver's door
After the driver's door has been closed the instrument cluster is activated for half a minute.

F-6 Temporal activation by switching on headlights
After the headlights have been switched on the instrument cluster is activated for half a minute.

F-7 Temporal activation with the push-button
After the instrument cluster push button has been applied the instrument cluster is activated for half a minute.

[...]

[**UC1**]

The driver starts the car and the instrument cluster is turned on.

Basic flow:

1. The use case begins when a driver sits in the car and the instrument cluster is deactivated. The ignition is turned off.
2. The driver switches on the car (ignition key in position ignition on).
3. The instrument cluster is turned on and stays active.
4. After the trip the driver switches off the ignition.
5. The instrument cluster stays active for 30 seconds and then turns itself off.
6. The driver leaves the car.

[...]

[**UC2**]

The instrument cluster is turned on temporarily by the driver.

Basic flow:

1. The use case begins when a driver enters the car. The ignition is turned off.
2. The driver opens the door.
3. The instrument cluster is activated temporarily.
4. The instrument cluster turns itself off after 30 seconds.
5. The driver leaves the car.

[...]

Fig. A.2. Daimler-Chrysler instrument cluster requirements [Buhr et al. 2003, p. 3–4].

B. CNL GRAMMARS USED IN THIS STUDY

B.1 Grammar of CNL Marko

The grammar of CNL Marko is implemented in Xtext by Marko et al [2015]. To acquire an official copy of this language, please contact the authors of that study. We do not include it in the report as to encourage readers to acquire an official copy of the language.

B.2 Grammar of CNL Vierhauser

Figure B.1 shows the grammar of CNL Vierhauser.

C. CNL GRAMMARS EXPRESSED IN XTEXT

C.1 Grammar of CNL Marko expressed in Xtext

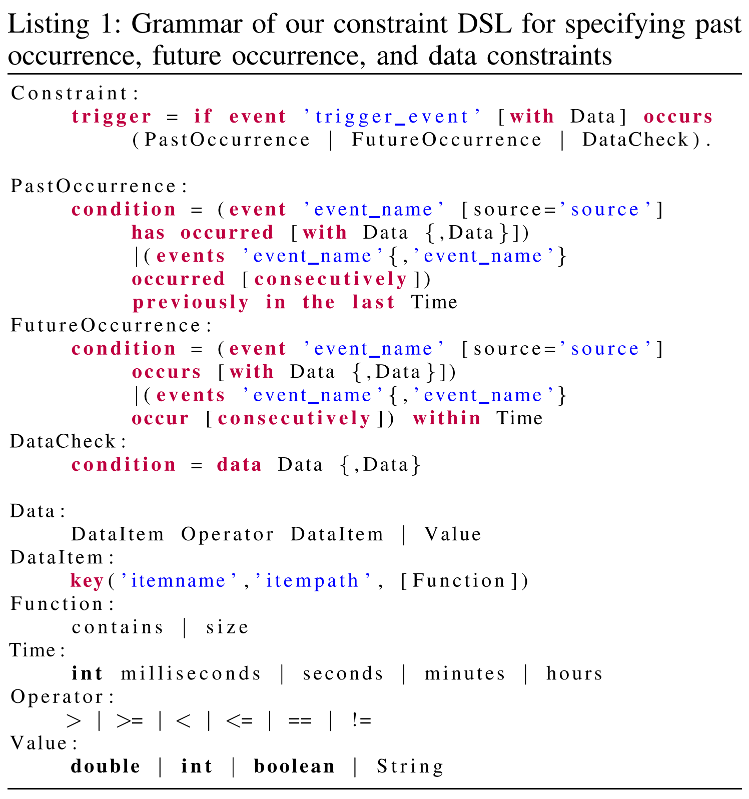
For legal reasons, the grammar of CNL Marko expressed in Xtext could not be included in this report.

C.2 Grammar of CNL Vierhauser expressed in Xtext

Figure C.1 shows CNL Vierhauser expressed in Xtext.

C.3 Grammar of revised CNL Vierhauser expressed in Xtext

Figures C.2 and C.3 show the *revised* CNL Vierhauser expressed in Xtext.



Listing 1: Grammar of our constraint DSL for specifying past occurrence, future occurrence, and data constraints

```
Constraint:
    trigger = if event 'trigger_event' [with Data] occurs
        (PastOccurrence | FutureOccurrence | DataCheck).

PastOccurrence:
    condition = (event 'event_name' [source='source']
        has occurred [with Data {,Data}])
        |(events 'event_name'{,'event_name'}
        occurred [consecutively])
        previously in the last Time
FutureOccurrence:
    condition = (event 'event_name' [source='source']
        occurs [with Data {,Data}])
        |(events 'event_name'{,'event_name'}
        occur [consecutively]) within Time
DataCheck:
    condition = data Data {,Data}

Data:
    DataItem Operator DataItem | Value
DataItem:
    key('itemname','itempath', [Function])
Function:
    contains | size
Time:
    int milliseconds | seconds | minutes | hours
Operator:
    > | >= | < | <= | == | !=
Value:
    double | int | boolean | String
```

Fig. B.1. Grammar of the constraint-based DSL used by Vierhauser et al. [2015, p. 717].

D. REQUIREMENTS EXPRESSED IN CONTROLLED NATURAL LANGUAGE

D.1 WLTP gear shift requirements expressed in CNL Marko

Figures D.1 and D.2 show the WLTP gear shift requirements expressed in CNL Marko.

D.2 DC instrument cluster requirements expressed in CNL Marko

Figures D.3 and D.4 show the DC instrument cluster requirements expressed in CNL Marko.

D.3 WLTP gear shift requirements expressed in CNL Vierhauser

Figures D.5 and D.6 show the WLTP gear shift requirements expressed in CNL Vierhauser.

D.4 WLTP gear shift requirements expressed in revised CNL Vierhauser

Figures D.7, D.8 and D.9 show the WLTP gear shift requirements expressed in revised CNL Vierhauser.

D.5 DC instrument cluster requirements expressed in revised CNL Vierhauser

Figures D.10 and D.11 show the DC instrument cluster requirements expressed in revised CNL Vierhauser.

E. REQUIREMENTS EXPRESSED IN SML

E.1 WLTP gear shift requirements expressed in SML

Figures E.1, E.2 and E.3 show the WLTP gear shift requirements expressed in SML.

```
grammar de.ase.vierhauser.language.ConstraintDSL with org.eclipse.xtext.common.Terminals

generate constraintDSL "http://www.ase.de/vierhauser/language/ConstraintDSL"

Model:
	constraints+=Constraint*;

Constraint:
	'trigger' '=' 'if' 'event' trigger_event=STRING ('with' data=Data)? 'occurs'
occurrence=(PastOccurrence |
	FutureOccurrence | DataCheck) '.';

PastOccurrence:
	'condition' '='
	('event' event_name=STRING ('source' '=' source=STRING)? 'has occurred' ('with'
data+=Data (','
	data+=Data)*)?
	|
	('events' event_names+=STRING (',' event_names+=STRING)* 'occurred'
('consecutively')?))
	'previously in the last' time=Time;

FutureOccurrence:
	'condition' '=' ('event' event_name=STRING ('source' '=' source=STRING)? 'occurs'
('with' data+=Data (','
	data+=Data)*)?
	|
	('events' event_names+=STRING (',' event_names+=STRING)* 'occur'
('consecutively')?))
	'within' time=Time;

DataCheck:
	'condition' '=' 'data' data+=Data (',' data+=Data)*;

Data:
	source=DataItem operator=Operator target=(DataItem | Value);

DataItem:
	'key' '(' itemname=STRING ',' itempath=STRING (',' function=Function)? ')';

Function returns Function:
	{Function}
	('contains' | 'size');

Time:
	value=INT ('milliseconds' | 'seconds' | 'minutes' | 'hours');

Operator returns Operator:
	{Operator}
	'>' | '>=' | '<' | '<=' | '==' | '!=';

Value returns Value:
	{Value}
	DOUBLE | INT | BOOLEAN | STRING;

terminal BOOLEAN:
	'true' | 'false';

/* https://eclipse.org/Xtext/documentation/301_grammarlanguage.html */
terminal DOUBLE:
	INT '.' INT;
```

Fig. C.1. CNL Vierhauser expressed in Xtext.

```
grammar de.ase.vierhauser.language.ConstraintDSL with
org.eclipse.xtext.common.Terminals

generate constraintDSL "http://www.ase.de/vierhauser/language/ConstraintDSL"

Model:
	constraints+=Constraint*;

Constraint:
	'trigger' '=' 'if' 'event' trigger_event=Event 'occurs'
occurrence=Occurrence '.';

Occurrence:
	PastOccurrence | FutureOccurrence | DataCheck;

PastOccurrence:
	'condition' '='
	('past event' events+=Event (negated?='has not occurred' | 'has occurred')
	|
	('past events' events+=Event (',' events+=Event)*
	(negated?='have not occurred' | 'have occurred') ('consecutively')?))
	('previously in the last' time=Time)?;

FutureOccurrence:
	'condition' '='
	('future event' events+=Event (negated?='does not occur' | 'occurs')
	|
	('future events' events+=Event (',' events+=Event)*
	(negated?='do not occur' | 'do occur') ('consecutively')?))
	('within' time=Time)?;

DataCheck:
	'condition' '=' 'data' data+=Data (',' data+=Data)*;

Event:
	name=STRING ('source' '=' source=STRING)? ('with' data+=Data (','
data+=Data)*)?;

Data:
	ComparisonExpression | BinaryExpression;

BinaryExpression:
	'{' source=Data operator=BinaryOperator target=Data '}';

ComparisonExpression:
	source=TargetExpression operator=ComparisonOperator
target=TargetExpression;

TargetExpression:
	ArithmeticExpression | DataItemExpression | ValueExpression;

ArithmeticExpression:
	'(' left=TargetExpression operator=ArithmeticOperator
right=PlainTargetExpression ')';

PlainTargetExpression:
	PlainArithmeticExpression | DataItemExpression | ValueExpression;

PlainArithmeticExpression:
	source=DataItemOrValueExpression operator=ArithmeticOperator
target=PlainTargetExpression;
```

Fig. C.2. Revised CNL Vierhauser expressed in Xtext.

```
DataItemOrValueExpression:
      DataItemExpression | ValueExpression;

DataItemExpression:
      dataitem=DataItem;

ValueExpression:
      value=Value;

DataItem:
      'key' '(' itemname=STRING ',' itempath=STRING (',' function=FunctionType)?
')';

Time:
      value=INT format=TimeFormat;

Value:
      DoubleValue | IntegerValue | BooleanValue | StringValue;

DoubleValue:
      value=DOUBLE;

IntegerValue:
      value=INT;

BooleanValue:
      value=BOOLEAN;

StringValue:
      value=STRING;

enum FunctionType:
      NONE='none' | CONTAINS='contains' | SIZE='size';

enum TimeFormat:
      MILLISECONDS='milliseconds' | SECONDS='seconds' | MINUTES='minutes' |
HOURS='hours';

enum ComparisonOperator:
      GREATER_THAN='>' | GREATER_THAN_OR_EQUALS='>=' | LESS_THAN='<' |
LESS_THAN_OR_EQUALS='<='
      | EQUALS='==' | NOT_EQUALS='!=';

enum ArithmeticOperator:
      ADDITION='+' | SUBTRACTION='-';

enum BinaryOperator:
      AND='and' | OR='or';

terminal BOOLEAN:
      'true' | 'false';

terminal DOUBLE:
      INT '.' INT;
```

Fig. C.3. Revised CNL Vierhauser expressed in Xtext (cont'd).

```
/*
 * (a) First gear shall be selected one second before beginning an
acceleration phase from standstill with the clutch disengaged. Vehicle
speeds below 1 km/h imply that the vehicle is standing still;
 */

// First gear shall be selected one second before beginning an
// acceleration phase from standstill with the clutch disengaged.
ID R01_acceleration: While state: standstill and if parameter: gear is quantity:
1,
the system: transmission shall action: start state: acceleration within time: 1
s
dependent on parameter: clutch equal to constant: disengaged.

//Vehicle speeds below 1 km/h imply that the vehicle is standing still;
ID R01_standstill: if parameter: speed decreases below quantity: 1 unit: KM_H,
the system: transmission shall action: start state: standstill.

/*
 * [(b) Gears shall not be skipped during acceleration phases. Gears used
during accelerations and decelerations must be used for a period of at
least three seconds (e.g. a gear sequence 1, 1, 2, 2, 3, 3, 3, 3, 3 shall be
replaced by 1, 1, 1, 2, 2, 2, 3, 3, 3);]
 */

// Gears shall not be skipped during acceleration phases.
ID R02_not_skipped: While state: acceleration,
the system: transmission shall not action: select parameter: gear
from variable: i to variable: j unequal to variable: i + 1 or
from variable: i to variable: j unequal to variable: i - 1.

// Gears used during accelerations and decelerations must be used for a
// period of at least three seconds (e.g. a gear sequence 1, 1, 2, 2, 3,
// 3, 3, 3, 3 shall be replaced by 1, 1, 1, 2, 2, 2, 3, 3, 3);
ID R02_gear_used : While state: acceleration or While state: deceleration and if
parameter: gear is variable: i, the system: transmission shall not action:
select parameter: gear
to variable: j unequal to variable: i for at least time: 3 s.

/*
 * [(c) Gears may be skipped during deceleration phases. For the last phase
of a deceleration to a stop, the clutch may be either disengaged or the
gear lever placed in neutral and the clutch left engaged;]
 */

// Skipped, because it not a requirement. A requirement is "The system shall..."

/*
 * (d) There shall be no gearshift during transition from an acceleration
phase to a deceleration phase. E.g., if v(j) < v(j+1) > v(j+2) and the gear
for the time sequence j and j + 1 is i, gear i is also kept for the time
 j + 2, even if the initial gear for j + 2 would be i + 1;
 */

// There shall be no gearshift during transition from an acceleration
// phase to a deceleration phase.
ID R04_no_gear_shift: While state: acceleration
and if system: transmission starts state: deceleration,
the system: transmission shall not action: select parameter: gear to variable: i
dependent on parameter: acceleration_stop equal to constant: true.

/*
```

Fig. D.1. The WLTP gear shift requirements expressed in CNL Marko.

```
 * (e) If a gear i is used for a time sequence of 1 to 5 s and the gear before
this sequence is the same as the gear after this sequence, e.g. i - 1, the
gear use for this sequence shall be corrected to i - 1.
 * Example:
 * [...]
 * For all cases (i) to (v), g(min) <= i must be fulfilled;
 */

ID R05_gear_used: the system: transmission shall not action: select
parameter: gear from variable: i to variable: i - 1
and from variable: i - 1 to variable: i
within time: 5 s dependent on parameter: gear
greater or equal to constant: minimum_gear.

/*
 * (f) a gear sequence i, i - 1, i, shall be replaced by i, i, i, if the
following conditions are fulfilled:
 * (i) engine speed does not drop below n(min); and
 * (ii) the sequence does not occur more often than four times each
 * for the low, medium and high speed cycle phases and not more
 * than three times for the extra high speed phase.
 * Requirement (ii) is necessary as the available power will drop below
 * the required power when the gear i - 1, is replaced by i;
 */

ID R06_gear_used: While state: acceleration,
the system: transmission shall not action: select parameter: gear
to variable: i and from variable: i - 1 to variable: i within time: 3 s
dependent on parameter: engine_speed greater or equal to constant:
min_engine_speed.

// the sequence does not occur more often than four times each

/*
 * (g) If, during an acceleration phase, a lower gear is required at a higher
vehicle speed, the higher gears before shall be corrected to the lower
gear, if the lower gear is required for at least 2 s.
 * Example: [...]
 * Since the above modifications may create new gear use sequences
which are in conflict with these requirements, the gear sequences shall
be checked twice
 */

ID R07_gear_used: While state: acceleration,
the system: transmission shall not action: select parameter: gear
from variable: i to variable: i + 1 and from variable: i + 1 to variable: i
dependent on parameter: speed greater than variable: previous_speed for at least
time: 2 s.
```

Fig. D.2. The WLTP gear shift requirements expressed in CNL Marko (cont'd).

```
/*
 * F-1 Permanent activation when ignition on
 * After the ignition has been switched on the instrument cluster is activated.
 */

ID R09_permanent_activation: the system: instrument_cluster shall action: select
parameter: instrument_cluster to constant: on after parameter: ignition reaches
constant: on.

/*
 * F-2 Deactivation by switching off ignition
 * Half a minute after the ignition has been switched off the
 * instrument cluster is deactivated and all (warning) lights dim out.
 */

ID R10_deactivation: the system: instrument_cluster shall action: select
parameter: instrument_cluster to constant: off within time: 30 s
after parameter: ignition reaches constant: off.
/*
 * F-3 Permanent activation by setting ignition key in position radio
 * After the ignition key is set in position radio, the instrument cluster is
activated.
 */

ID R11_permanent_activation: the system: instrument_cluster shall action: select
parameter: instrument_cluster to constant: on after parameter: ignition reaches
constant: radio.

/*
 * F-4 Temporal activation by opening driver's door
 * After the driver's door has been opened the instrument cluster is activated
for half a minute.
 */

ID R12_temporal_activation: the system: instrument_cluster shall action: select
parameter: instrument_cluster to constant: on for time: 30 s
after parameter: driver_door reaches constant: open.

/*
 * F-5 Temporal activation by closing driver's door
 * After the driver's door has been closed the instrument cluster is activated
for half a minute.
 */

ID R13_temporal_activation: the system: instrument_cluster shall action: select
parameter: instrument_cluster to constant: on for time: 30 s
after parameter: driver_door reaches constant: closed.

/*
 * F-6 Temporal activation by switching on headlights
 * After the headlights have been switched on the instrument cluster is
activated for half a minute.
 */

ID R14_temporal_activation: the system: instrument_cluster shall action: select
parameter: instrument_cluster to constant: on for time: 30 s
after parameter: headlights reaches constant: on.

/*
 * F-7 Temporal activation with the push-button
 * After the instrument cluster push button has been applied the instrument
cluster is activated for half a minute.
```

Fig. D.3. The DC instrument cluster requirements expressed in CNL Marko.

```
 */

ID R14_temporal_activation: the system: instrument_cluster shall action: select
parameter: instrument_cluster to constant: on for time: 30 s
after parameter: push_button reaches constant: on.

/*
 * UC1
 * The driver starts the car and the instrument cluster is turned on.
 * Basic flow:
 * 1. The use case begins when a driver sits in the car and the instrument
cluster is deactivated. The ignition is turned off.
 * 2. The driver switches on the car (ignition key in position ignition on).
 * 3. The instrument cluster is turned on and stays active.
 * 4. After the trip the driver switches off the ignition.
 * 5. The instrument cluster stays active for 30 seconds and then turns itself
off.
 * 6. The driver leaves the car.
 */

// Already supported by R9-R10.

/*
 * UC2
 * The instrument cluster is turned on temporarily by the driver.
 * Basic flow:
 * 1. The use case begins when a driver enters the car. The ignition is turned
off.
 * 2. The driver opens the door.
 * 3. The instrument cluster is activated temporarily.
 * 4. The instrument cluster turns itself off after 30 seconds.
 * 5. The driver leaves the car.
 */

// Already supported by R12.
```

Fig. D.4. The DC instrument cluster requirements expressed in CNL Marko (cont'd).

```
/*
 * (a) First gear shall be selected one second before beginning an
acceleration phase from standstill with the clutch disengaged. Vehicle
speeds below 1 km/h imply that the vehicle is standing still;
 */

// First gear shall be selected one second before beginning an
// acceleration phase from standstill with the clutch disengaged.
trigger = if event "gearshift_acceleration" occurs
condition = event "gear selected" has occurred with
key ("transmission", "gear") == 1,
key ("transmission", "clutch") == "disengaged" previously in the last 1 seconds.

// Vehicle speeds below 1 km/h imply that the vehicle is standing still;
trigger = if event "standstill" occurs
condition = data key("transmission", "speed") < 1 .

/*
 * [(b) Gears shall not be skipped during acceleration phases. Gears used
during accelerations and decelerations must be used for a period of at
least three seconds (e.g. a gear sequence 1, 1, 2, 2, 3, 3, 3, 3, 3 shall be
replaced by 1, 1, 1, 2, 2, 2, 3, 3, 3);]
 */

// Gears shall not be skipped during acceleration phases.
// TODO: Would be nice if possible to gear + 1
trigger = if event "gearshift_acceleration" occurs
condition = data key("transmission", "gear") ==
key("transmission", "previous_gear_plus_one").

// Gears shall not be skipped during acceleration phases.
// TODO: Would be nice if possible to gear - 1
trigger = if event "gearshift_acceleration" occurs
condition = data key("transmission", "gear") ==
key("transmission", "previous_gear_minus_one").

// Gears used during accelerations and decelerations must be used for a period
of at
// least three seconds
// TODO: Wrong, should say not occurred
// ... What is gearshift? Should say gearshift_acceleration,
gearshift_deceleration.
trigger = if event "gearshift_acceleration" occurs
condition = events "acceleration", "gearshift" occurred previously in the last 3
seconds.
trigger = if event "gearshift_deceleration" occurs
condition = events "acceleration", "gearshift" occurred previously in the last 3
seconds.

/*
 * [(c) Gears may be skipped during deceleration phases. For the last phase
of a deceleration to a stop, the clutch may be either disengaged or the
gear lever placed in neutral and the clutch left engaged;]
 */

// Skipped, because it not a requirement. A requirement is "The system shall..."

/*
 * (d) There shall be no gearshift during transition from an acceleration
phase to a deceleration phase. E.g., if v(j) < v(j+1) > v(j+2) and the gear
for the time sequence j and j + 1 is i, gear i is also kept for the time
 j + 2, even if the initial gear for j + 2 would be i + 1;
```

Fig. D.5. The WLTP gear shift requirements expressed in CNL Vierhauser.

```
 */

// TODO: Wrong, should say not occurred previously
trigger = if event "gearshift_deceleration" occurs
condition = events "acceleration_stop", "gearshift_acceleration"
occurred previously in the last 2 seconds.

 /*
  * (e) If a gear i is used for a time sequence of 1 to 5 s and the gear before
this sequence is the same as the gear after this sequence, e.g. i - 1, the
gear use for this sequence shall be corrected to i - 1.
 * Example:
 * [...]
 * For all cases (i) to (v), g(min) <= i must be fulfilled;
 */

// TODO: Still wrong, because event "gearshift" must not occur,
trigger = if event "gearshift_acceleration" occurs
condition = event "gearshift" occurs with
key("transmission", "gear") != key("transmission", "previous_gear_min_one")
within 5 seconds.

/*
 * (f) a gear sequence i, i - 1, i, shall be replaced by i, i, i, if the
following conditions are fulfilled:
 * (i) engine speed does not drop below n(min); and
 * (ii) the sequence does not occur more often than four times each
 * for the low, medium and high speed cycle phases and not more
 * than three times for the extra high speed phase.
 * Requirement (ii) is necessary as the available power will drop below
 * the required power when the gear i - 1, is replaced by i;
 */

// TODO: Still wrong, because event "gearshift" must not occur
trigger = if event "gearshift_deceleration" occurs
condition = event "gearshift" occurs with
key("transmission", "gear") != key("transmission", "previous_gear_min_one"),
key("transmission", "engine_speed") > key("transmission", "min_engine_speed")
within 3 seconds.

/*
 * (g) If, during an acceleration phase, a lower gear is required at a higher
vehicle speed, the higher gears before shall be corrected to the lower
gear, if the lower gear is required for at least 2 s.
 * Example: [...]
 * Since the above modifications may create new gear use sequences
which are in conflict with these requirements, the gear sequences shall
be checked twice
 */

// TODO: Wrong, the event must not necessarily occur.
// TODO: Would be nice to know what gearshift_speed means.
// Would be nice if i - 1 could be used.
trigger = if event "gearshift_acceleration" occurs
condition = events "gearshift_speed", "gearshift_speed" occur
within 2 seconds.
```

Fig. D.6. The WLTP gear shift requirements expressed in CNL Vierhauser (cont'd).

```
/*
 * (a) First gear shall be selected one second before beginning an
acceleration phase from standstill with the clutch disengaged. Vehicle
speeds below 1 km/h imply that the vehicle is standing still;
 */

trigger = if events "clutch_disengaged"
with key ("transmission", "clutch_disengaged") == true AND "gear_selected" with
key ("transmission", "active_gear") == 1 occur
condition = future event "acceleration_phase" with key ("transmission",
"acceleration_phase") == true
may sometimes occur within 1 seconds.

// Vehicle speeds below 1 km/h imply that the vehicle is standing still;
trigger = if event "standstill" occurs
condition = data key("transmission", "speed") < 1 .

/*
 * [(b) Gears shall not be skipped during acceleration phases. Gears used
during accelerations and decelerations must be used for a period of at
least three seconds (e.g. a gear sequence 1, 1, 2, 2, 3, 3, 3, 3, 3 shall be
replaced by 1, 1, 1, 2, 2, 2, 3, 3, 3);]
 */

trigger = if event "acceleration_phase" with key("transmission",
"acceleration_phase") == true occurs
condition = future events "gearshift" with key("transmission", "active_gear") ==
key("prompt", "current"),
"gearshift" with {
      key("transmission", "active_gear") != ( key("prompt", "current") + 1 )
      or key("transmission", "active_gear") != ( key("prompt", "current") - 1 )
}
may sometimes not occur.

trigger = if events "acceleration_phase" with key("transmission",
"acceleration_phase") == true AND
"gearshift" with key("transmission", "active_gear") == key("prompt",
"selected_gear") occur
condition = future event "gearshift" with key("transmission", "active_gear") ==
key("prompt", "selected_gear")
may sometimes not occur within 3 seconds.

trigger = if events "deceleration_phase" with key("transmission",
"deceleration_phase") == true AND
"gearshift" with key("transmission", "active_gear") == key("prompt",
"selected_gear") occur
condition = future event "gearshift" with key("transmission", "active_gear") ==
key("prompt", "selected_gear")
may sometimes not occur within 3 seconds.

/*
 * [(c) Gears may be skipped during deceleration phases. For the last phase
of a deceleration to a stop, the clutch may be either disengaged or the
gear lever placed in neutral and the clutch left engaged;]
 */

// Skipped, because it not a requirement. A requirement is "The system shall..."

/*
 * (d) There shall be no gearshift during transition from an acceleration
phase to a deceleration phase. E.g., if v(j) < v(j+1) > v(j+2) and the gear
for the time sequence j and j + 1 is i, gear i is also kept for the time
```

Fig. D.7. The WLTP gear shift requirements expressed in revised CNL Vierhauser.

```
 j + 2, even if the initial gear for j + 2 would be i + 1;
 */

trigger = if event "acceleration_phase" with key("transmission",
"acceleration_phase") == false occurs
condition = future events "acceleration_stop" with key("transmission",
"acceleration_stop") == true,
"gearshift" with key("transmission", "active_gear") == key("transmission",
"selected_gear"),
"deceleration_phase" with key("transmission", "deceleration_phase") == true
may sometimes not occur consecutively.

/*
 * (e) If a gear i is used for a time sequence of 1 to 5 s and the gear before
this sequence is the same as the gear after this sequence, e.g. i - 1, the
gear use for this sequence shall be corrected to i - 1.
 * Example:
 * [...]
 * For all cases (i) to (v), g(min) <= i must be fulfilled;
 */

trigger = if event "gearshift" with
key("transmission", "active_gear") == key("transmission", "requested_gear")
occurs
condition = future events "gearshift" with
key("transmission", "active_gear") == ( key("transmission", "requested_gear") +
1 ),
"gearshift" with
key("transmission", "active_gear") == key("transmission", "requested_gear")
may sometimes not occur within 5 seconds.

/*
 * (f) a gear sequence i, i - 1, i, shall be replaced by i, i, i, if the
following conditions are fulfilled:
 * (i) engine speed does not drop below n(min); and
 * (ii) the sequence does not occur more often than four times each
 * for the low, medium and high speed cycle phases and not more
 * than three times for the extra high speed phase.
 * Requirement (ii) is necessary as the available power will drop below
 * the required power when the gear i - 1, is replaced by i;
 */

trigger = if event "gearshift" with key("transmission", "active_gear") ==
key("transmission", "requested_gear") occurs
condition = future events "gearshift" with key("transmission", "active_gear") ==
( key("transmission", "requested_gear") - 1 ),
"gearshift" with key("transmission", "active_gear") == key("transmission",
"requested_gear"),
"engine_speed_below_minimum" with key("transmission", "min_engine_speed") >
key("transmission", "engine_speed")
may sometimes not occur within 3 seconds.

/*
 * (g) If, during an acceleration phase, a lower gear is required at a higher
vehicle speed, the higher gears before shall be corrected to the lower
gear, if the lower gear is required for at least 2 s.
 * Example: [...]
 * Since the above modifications may create new gear use sequences
which are in conflict with these requirements, the gear sequences shall
be checked twice
 */
```

Fig. D.8. The WLTP gear shift requirements expressed in revised CNL Vierhauser (cont'd).

```
trigger = if event "acceleration_phase" with key("transmission",
"acceleration_phase") == true occurs
condition = future events "gearshift" with
key("transmission", "active_gear") == key("transmission", "requested_gear"),
"accelerate" with key("transmission", "speed") > key("transmission",
"previous_speed"),
"gearshift" with key("transmission", "active_gear") == ( key("transmission",
"requested_gear") + 1 ),
"accelerate" with key("transmission", "speed") > key("transmission",
"previous_speed"),
"gearshift" with key("transmission", "active_gear") == key("transmission",
"requested_gear"),
"accelerate" with key("transmission", "speed") > key("transmission",
"previous_speed")
may sometimes not occur within 2 seconds.
```

Fig. D.9. The WLTP gear shift requirements expressed in revised CNL Vierhauser (cont'd).

```
/*
 * F-1 Permanent activation when ignition on
 * After the ignition has been switched on the instrument cluster is activated.
 */

trigger = if event "ignition_on"
with key ("instrument", "ignition") == true occurs
condition = future event "activated"
with key ("instrument", "instrument_cluster") == true
must always occur.

/*
 * F-2 Deactivation by switching off ignition
 * Half a minute after the ignition has been switched off the
 * instrument cluster is deactivated and all (warning) lights dim out.
 */

trigger = if event "ignition_off"
with key ("instrument", "ignition") == false occurs
condition = future event "deactivated" with
key ("instrument", "instrument_cluster") == false
must always occur within 30 seconds.

/*
 * F-3 Permanent activation by setting ignition key in position radio
 * After the ignition key is set in position radio, the instrument cluster is
activated.
 */

trigger = if event "ignition_radio"
with key ("instrument", "ignition_radio") == true occurs
condition = future event "activated"
with key ("instrument", "instrument_cluster") == true
must always occur.

/*
 * F-4 Temporal activation by opening driver's door
 * After the driver's door has been opened the instrument cluster is activated
for half a minute.
 */

trigger = if event "door_opened"
with key ("instrument", "door_opened") == true occurs
condition = future events "activated" with
key ("instrument", "instrument_cluster") == true, "deactivated" with
key ("instrument", "instrument_cluster") == false
must always occur within 30 seconds.

/*
 * F-5 Temporal activation by closing driver's door
 * After the driver's door has been closed the instrument cluster is activated
for half a minute.
 */

trigger = if event "door_closed"
with key ("instrument", "door_opened") == false occurs
condition = future events "activated" with
key ("instrument", "instrument_cluster") == true, "deactivated" with
key ("instrument", "instrument_cluster") == false
must always occur within 30 seconds.

/*
```

Fig. D.10. The DC instrument cluster requirements expressed in revised CNL Vierhauser.

```
 * F-6 Temporal activation by switching on headlights
 * After the headlights have been switched on the instrument cluster is
activated for half a minute.
 */

trigger = if event "headlights_on"
with key ("instrument", "headlights") == true occurs
condition = future events "activated" with
key ("instrument", "instrument_cluster") == true, "deactivated" with
key ("instrument", "instrument_cluster") == false
must always occur within 30 seconds.

/*
 * F-7 Temporal activation with the push-button
 * After the instrument cluster push button has been applied the instrument
cluster is activated for half a minute.
 */

trigger = if event "push_button_applied"
with key ("instrument", "push_button") == true occurs
condition = future events "activated" with
key ("instrument", "instrument_cluster") == true, "deactivated" with
key ("instrument", "instrument_cluster") == false
must always occur within 30 seconds.

/*
 * UC1
 * The driver starts the car and the instrument cluster is turned on.
 * Basic flow:
 * 1. The use case begins when a driver sits in the car and the instrument
cluster is deactivated. The ignition is turned off.
 * 2. The driver switches on the car (ignition key in position ignition on).
 * 3. The instrument cluster is turned on and stays active.
 * 4. After the trip the driver switches off the ignition.
 * 5. The instrument cluster stays active for 30 seconds and then turns itself
off.
 * 6. The driver leaves the car.
 */

// Already supported by R9-R10.

/*
 * UC2
 * The instrument cluster is turned on temporarily by the driver.
 * Basic flow:
 * 1. The use case begins when a driver enters the car. The ignition is turned
off.
 * 2. The driver opens the door.
 * 3. The instrument cluster is activated temporarily.
 * 4. The instrument cluster turns itself off after 30 seconds.
 * 5. The driver leaves the car.
 */

// Already supported by R12.
```

Fig. D.11. The DC instrument cluster requirements expressed in revised CNL Vierhauser (cont'd).

```
import "Transmission.ecore"

system specification Gearshifts {

      domain Transmission
      define Environment as uncontrollable
      define GearSelector as controllable
      define GearController as controllable

      collaboration Gearshifts1 {

            static role Environment env
            static role GearSelector gs
            static role GearController gc

            /*
             * (a) First gear shall be selected one second before beginning an
            acceleration phase from standstill with the clutch disengaged.
Vehicle
            speeds below 1 km/h imply that the vehicle is standing still;
             */
            requirement scenario R01_firstGearAfterClutchDisengaged {
                  message env -> gs.setClutchDisengaged(true)
                  message requested env -> gc.setActiveGear(1)
                  // Clock
                  var EInt c = 0
                  message requested env -> gs.setAccPhase(true)
                  violation if [ c >= 1 | ! gs.clutchDisengaged ]
            } constraints [
                  interrupt message env -> gs.setClutchDisengaged(false)
            ]

            /*
             * [(b) Gears shall not be skipped during acceleration phases. Gears
used
            during accelerations and decelerations must be used for a period of
at
            least three seconds (e.g. a gear sequence 1, 1, 2, 2, 3, 3, 3, 3, 3
shall be
            replaced by 1, 1, 1, 2, 2, 2, 3, 3, 3);]
             */
            requirement scenario R02_NextGearAfterAccPhaseBegins {
                  message env -> gs.setAccPhase(true)
                  var EInt cur = gc.activeGear
                  alternative {
                        message requested gs -> gc.setActiveGear(cur + 1)
                        interrupt if [ true ]
                  } or {
                        message requested gs -> gc.setActiveGear(cur - 1)
                        interrupt if [ true ]
                  }
                  violation if [ true ]
            } constraints [
                  interrupt message env -> gs.setAccPhase(false)
            ]

            requirement scenario R02_MinGearUse {
                  message gs -> gc.setActiveGear(bind to cur)
                  var EInt cur = -1
                  interrupt if [ ! gs.accPhase & ! gs.decPhase ]
                  // Clock
                  var EInt c = 0
```

Fig. E.1. The WLTP gear shift requirements expressed in SML.

```
                var EInt new = -1
                message gs -> gc.setActiveGear(bind to new)
                // Cannot be other than violated.
                //violation if [ c < 3 ]

            } constraints [
                interrupt message env -> gs.setAccPhase(false)
                interrupt message env -> gs.setDecPhase(false)
            ]

            /*
             * (d) There shall be no gearshift during transition from an
acceleration
            phase to a deceleration phase. E.g., if v(j) < v(j+1) > v(j+2) and
the gear
            for the time sequence j and j + 1 is i, gear i is also kept for the
time
             j + 2, even if the initial gear for j + 2 would be i + 1;
             */
            requirement scenario R04_GearUseBetweenAccToDec {
                message env -> gs.setAccPhase(false)
                var EInt new = -1
                message gs -> gc.setActiveGear(bind to new)
                message env -> gs.setDecPhase(true)
                violation if [ true ]
            }

            /*
             * (e) If a gear i is used for a time sequence of 1 to 5 s and the
gear before
            this sequence is the same as the gear after this sequence, e.g. i -
1, the
            gear use for this sequence shall be corrected to i - 1.
             * Example:
             * [...]
             * For all cases (i) to (v), g(min) <= i must be fulfilled;
             */
            requirement scenario R05_GearUseBetweenOneAndFiveSec {
                message gs -> gc.setActiveGear(bind to g1)
                var EInt g1 = -1
                // Clock
                var EInt c = 0
                message gs -> gc.setActiveGear(bind to g2)
                var EInt g2 = -1
                message gs -> gc.setActiveGear(bind to g3)
                var EInt g3 = -1
                violation if [ c < 5 & g1 == g3 ]
            }

            /*
             * (f) a gear sequence i, i - 1, i, shall be replaced by i, i, i, if
the following
            conditions are fulfilled:
             * (i) engine speed does not drop below n(min); and
             * (ii) the sequence does not occur more often than four times each
             * for the low, medium and high speed cycle phases and not more
             * than three times for the extra high speed phase.
             * Requirement (ii) is necessary as the available power will drop
below
             * the required power when the gear i - 1, is replaced by i;
             */
            requirement scenario R06_GearUseBetweenOneAndThreeSec {
```

Fig. E.2. The WLTP gear shift requirements expressed in SML (cont'd).

```
                  message gs -> gc.setActiveGear(bind to g1)
                  var EInt g1 = 1
                  message gs -> gc.setActiveGear(bind to g2)
                  var EInt g2 = 2
                  interrupt if [ g1 == g2 ]
                  // Clock
                  var EInt c = 0
                  var EInt g3 = -1
                  message gs -> gc.setActiveGear(bind to g3)
                  violation if [(c >= 1 & c <= 3) ]
                  // engine speed
                  //(sequence does not occur more than three times)]

            }

            /*
             * (g) If, during an acceleration phase, a lower gear is required at
a higher
            vehicle speed, the higher gears before shall be corrected to the
lower
            gear, if the lower gear is required for at least 2 s.
             * Example: [...]
             * Since the above modifications may create new gear use sequences
            which are in conflict with these requirements, the gear sequences
shall
            be checked twice
             */
            requirement scenario R07_HigherGear {
                  message env -> gs.setAccPhase(true)
                  message gs -> gc.setActiveGear(bind to g1)
                  var EInt g1 = 1
                  var EInt g1_speed = gs.speed
                  interrupt if [ ! gs.accPhase ]
                  // Clock
                  var EInt c = 0
                  var EInt g2 = 2
                  message gs -> gc.setActiveGear(bind to g2)
                  var EInt g2_speed = gs.speed
                  interrupt if [ g2_speed <= g1_speed | g2 != g1 + 1 ]
                  var EInt g3 = -1
                  message gs -> gc.setActiveGear(bind to g3)
                  interrupt if [ c < 2 ]
                  var EInt g3_speed = gs.speed
                  interrupt if [ g3_speed <= g2_speed | g3 != g1 ]
                  violation if [ true ]
            }

      }

}
```

Fig. E.3. The WLTP gear shift requirements expressed in SML (cont'd).

E.2 DC cluster instrument requirements expressed in SML

Figures E.4, E.5 and E.6 show the DC cluster instrument requirements expressed in SML.

F. TRANSFORMATION EXPRESSED IN XTEND

The implementation of the model transformation can be found online.

```
import "InstrumentCluster.ecore"

system specification Specification {

      domain InstrumentCluster

      define InstrumentClusterSelector as controllable
      define InstrumentClusterController as controllable
      define Environment as uncontrollable

      collaboration Collaboration {

            static role InstrumentClusterSelector ics
            static role Environment env
            static role InstrumentClusterController icc

            /*
             * F-1 Permanent activation when ignition on
             * After the ignition has been switched on the instrument cluster is
activated
             */
            requirement scenario R1 {
                  message env -> icc.setIgnition(true)
                  message strict requested ics -> icc.setInstrumentCluster(true)
            }

            /*
             * F-2 Deactivation by switching off ignition
             * Half a minute after the ignition has been switched off the
instrument cluster is deactivated
and all (warning) lights dim out.
             */
            requirement scenario R2 {
                  message env -> icc.setIgnition(false)
                  // Clock
                  var EInt c = 0
                  message strict requested ics -> icc.setInstrumentCluster(true)
                  violation if [ c > 30 ]
            }

            /*
             * F-3 Permanent activation by setting ignition key in position
radio
             * After the ignition key is set in position radio, the instrument
cluster is activated.
             */
            requirement scenario R3 {
                  message env -> icc.setIgnitionRadio(true)
                  message strict requested ics -> icc.setInstrumentCluster(true)
            }

            /*
             * F-4 Temporal activation by opening driver’s door
             * After the driver’s door has been opened the instrument cluster is
activated for half a
minute
             */
            requirement scenario R4 {
                  message env -> icc.setDoorOpen(true)
                  message strict requested ics -> icc.setInstrumentCluster(true)
                  // Clock
                  var EInt c = 0
```

Fig. E.4. The DC cluster instrument requirements expressed in SML.

```
                  message strict requested ics ->
icc.setInstrumentCluster(false)
                  violation if [ c < 30 ]
            }

            /*
             * F-5 Temporal activation by closing driver’s door
             * After the driver’s door has been closed the instrument cluster is
activated for half a
minute.
             */
            requirement scenario R5 {
                  message env -> icc.setDoorOpen(false)
                  message strict requested ics -> icc.setInstrumentCluster(true)
                  // Clock
                  var EInt c = 0
                  message strict requested ics ->
icc.setInstrumentCluster(false)
                  violation if [ c < 30 ]
            }

            /*
             * F-6 Temporal activation by switching on headlights
             * After the headlights have been switched on the instrument cluster
is activated for half
a minute.
             */
            requirement scenario R6 {
                  message env -> icc.setHeadlights(true)
                  message strict requested ics -> icc.setInstrumentCluster(true)
                  // Clock
                  var EInt c = 0
                  message strict requested ics ->
icc.setInstrumentCluster(false)
                  violation if [ c < 30 ]
            }

            /*
             * F-7 Temporal activation with the push-button
             * After the instrument cluster push button has been applied the
instrument cluster is
activated for half a minute.
             */
            requirement scenario R7 {
                  message env -> icc.setPushButton(true)
                  message strict requested ics -> icc.setInstrumentCluster(true)
                  // Clock
                  var EInt c = 0
                  message strict requested ics ->
icc.setInstrumentCluster(false)
                  violation if [ c < 30 ]
            }

            /*
             * UC1
             * The driver starts the car and the instrument cluster is turned
on.
             * Basic flow:
             * 1. The use case begins when a driver sits in the car and the
instrument cluster is deactivated. The ignition is turned off.
             * 2. The driver switches on the car (ignition key in position
ignition on).
```

Fig. E.5. The DC cluster instrument requirements expressed in SML (cont’d).

```
             * 3. The instrument cluster is turned on and stays active.
             * 4. After the trip the driver switches off the ignition.
             * 5. The instrument cluster stays active for 30 seconds and then
turns itself
            off.
             * 6. The driver leaves the car.
             */

            // Already supported by R1-R2.

            /*
             * UC2
             * The instrument cluster is turned on temporarily by the driver.
             * Basic flow:
             * 1. The use case begins when a driver enters the car. The ignition
is turned off.
             * 2. The driver opens the door.
             * 3. The instrument cluster is activated temporarily.
             * 4. The instrument cluster turns itself off after 30 seconds.
             * 5. The driver leaves the car.
             */

            // Already supported by R4.

      }

}
```

Fig. E.6. The DC cluster instrument requirements expressed in SML (cont'd).