Joint Specification and Testing of Safety and Security Requirements

Prof. Holger Schlingloff (ZeSys e.V.)
Dr. Mirko Conrad / Harald Schülzke (samoconsult GmbH)
Dr. Sadegh Sadeghipour (ITPower Solutions GmbH)

IFIP-WG1.3 meeting, Salzburg, 5.2.2024

Contents of this talk

- Safety and Security Co-Engineering
- Controlled Natural Languages
- The Domain-Specific Language LESS
- Refinement and Testing Methods based on LESS
- Outlook onto LLMs

Acknowledgement

Results from the EmbeddedSafeSec Project

samoconsult | ZESYS | ITPOWER SOLUTIONS

EUROPÄISCHE UNION
Europäischer Fonds für regionale Entwicklung

- A "small", local project
- Oct 2020 - Sep 2023
- Industry-driven
- Goal:
  Support consulting in safety- and security-engineering

Motivation

Co-Engineering Functional Safety and Cybersecurity

Embedded SafeSec

- Separate standards for functional safety (e.g., ISO 26262) and Cybersecurity (e.g., ISO/SAE 21434)
- Acknowledgement of interdependencies, but high-level guidance only

> **"The organization shall institute and maintain effective communication channels between functional safety, cybersecurity, and other disciplines that are related to the achievement of functional safety."**
>
> [ISO 26262-2:2018]

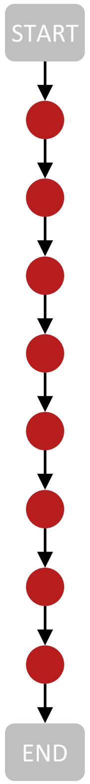
- How to combine life cycle of safety and security requirements?
- How to even *formulate* requirements for safety and security?

Safety and Security Co-Engineering

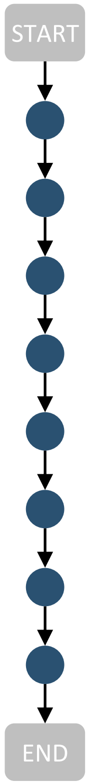
Mapping of Safety and Security Activities

Ideal World (Project Proposal)

Safety Lifecycle

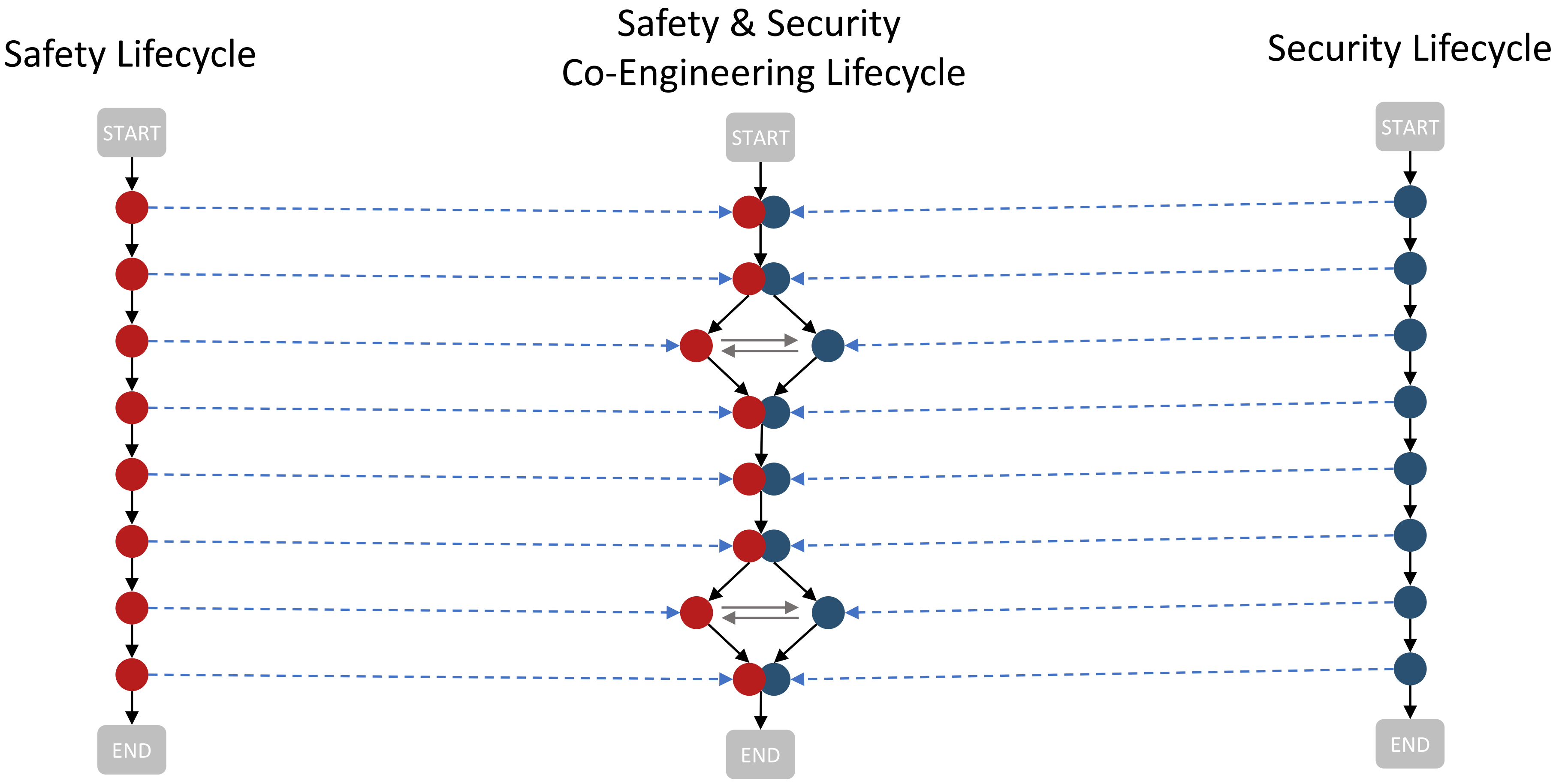


Security Lifecycle



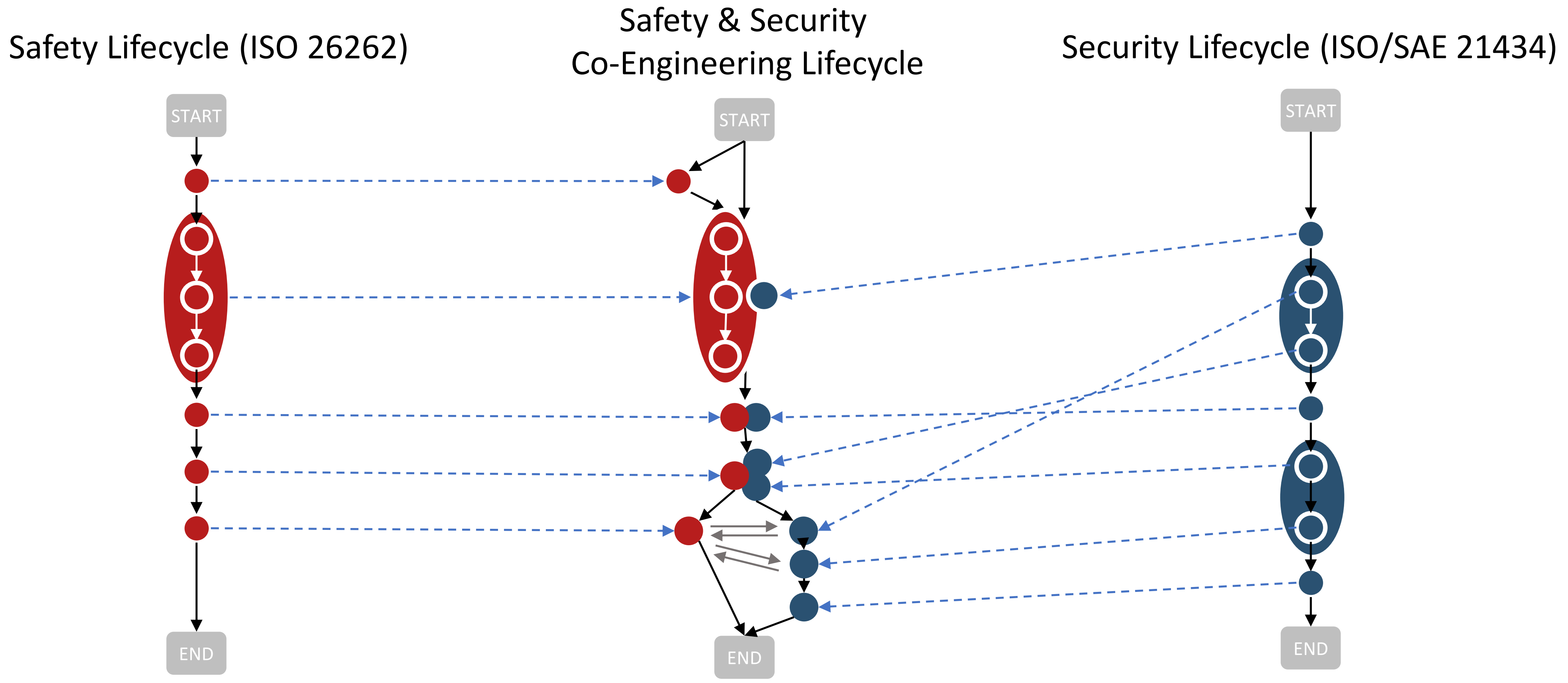
Mapping of Safety and Security Activities

Ideal World (Project Proposal)



Mapping of Safety and Security Activities

Real World – ISO 26262 and ISO/SAE 21434



Mapping of Safety and Security Activities

Development Phases – Activity Mapping

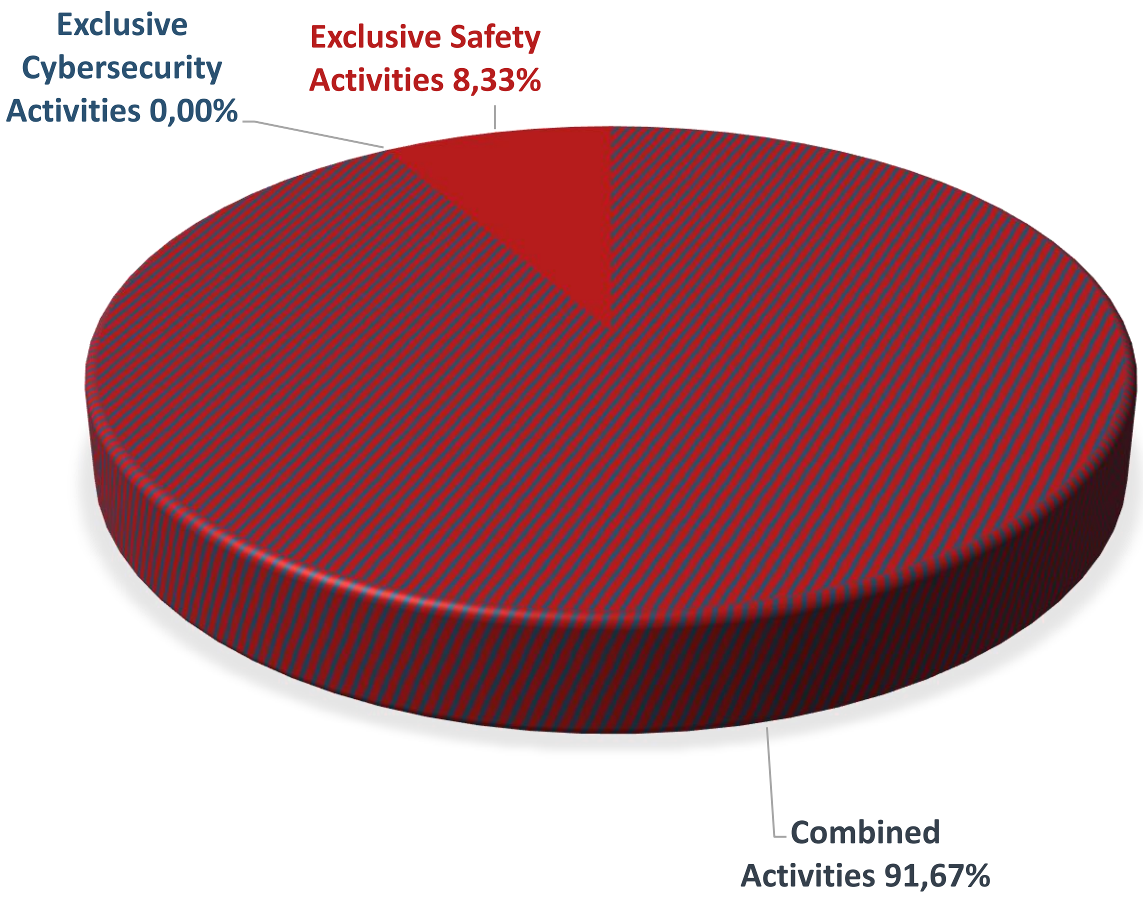
Concept Phase

Development Phases

System Development I

Hardware Development | Software Development

System Development II

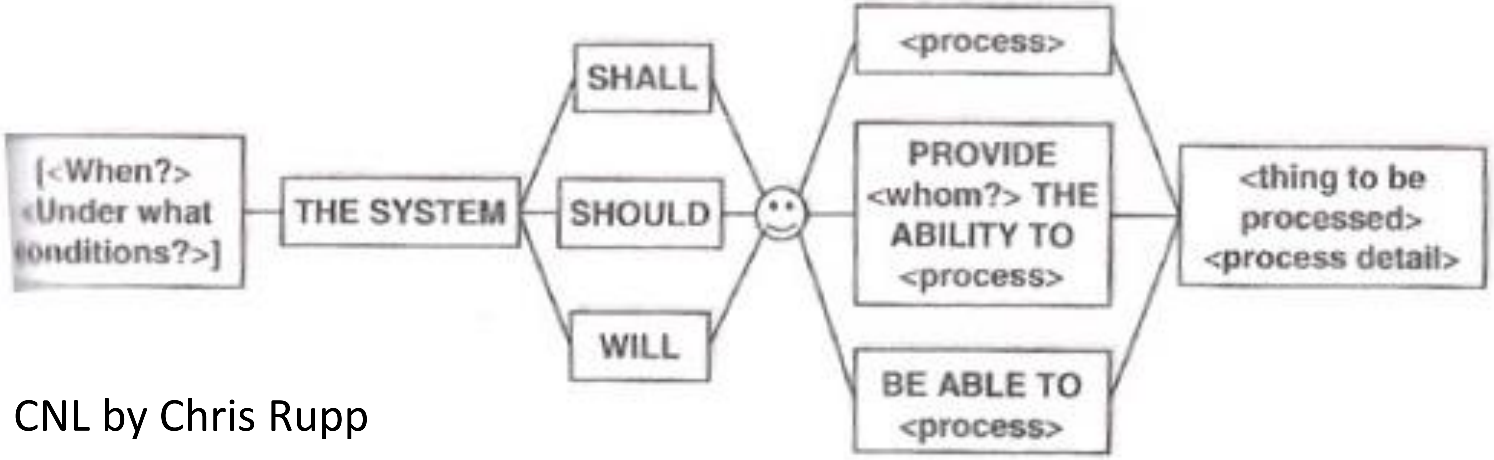


Controlled Natural Languages

Specifying Safety and Security

Embedded SafeSec

- We looked at various specification method
  - Formal Languages, e.g., B, Z, LTL, …
  - Controlled Natural Languages (CNLs)
- None supports the co-development of safety and security
- Developed a new DSL for this purpose



CNL by Chris Rupp

Three Definitions

A controlled language (CL) is a restricted version of a natural language which has been engineered to meet a special purpose, most often that of writing technical documentation for non-native speakers of the document language. A typical CL uses a well-defined subset of a language's grammar and lexicon, but adds the terminology needed in a technical domain. (Kittredge 2003, page 441)

Controlled natural language is a subset of natural language that can be accurately and efficiently processed by a computer, but is expressive enough to allow natural usage by non-specialists. (Fuchs and Schwitter 1995, page 1)

A **controlled natural language** is a constructed language that is based on a certain natural language, being more restrictive concerning lexicon, syntax, and/or semantics, while preserving most of its natural properties.

Characterization

A language is called a **controlled natural language** if and only if it has all of the following four properties:

1. It is based on exactly one natural language (its "base language").
2. The most important difference between it and its base language (but not necessarily the only one) is that it is more restrictive concerning lexicon, syntax, and/or semantics.
3. It preserves most of the natural properties of its base language, so that speakers of the base language can intuitively and correctly understand texts in the controlled natural language, at least to a substantial degree.
4. It is a constructed language, which means that it is explicitly and consciously defined, and *is not* the product of an implicit and natural process (even though it is based on a natural language that *is* the product of an implicit and natural process).

Classification Scheme for CNLs

According to design goals

- C: Communication among humans (comprehensibility)
- T: Translation into formal objects (translatability)
- F: Formal notation representation

„human-oriented" vs. „computer-oriented"

- A: academic, I: industrial, G: governmental
- D: domain-specific
- W: written, S: spoken

expressiveness vs. complexity

sources:
T. Kuhn, A Survey and Classification of Controlled Natural Languages, https://arxiv.org/abs/1507.01701
A. Wyner et al., On Controlled Natural Languages: Properties and Prospects, https://link.springer.com/chapter/10.1007/978-3-642-14418-9_17
Fuchs & Schwitter, Attempto Controlled Natural Language for Requirements Specifications

Dimensions of CNLs

Scale of 1 to 5 points:

- **P**recision (semantic non-ambiguity, predictability, formality)
- **E**xpressiveness (quantification, negation, if-then, etc.)
- **N**aturality (understandability, look-and-feel, brackets, etc.)
- **S**implicity (in syntax & semantics of language description)

- English: P^1 E^5 N^5 S^1
- Propositional logic: P^5 E^1 N^1 S^5

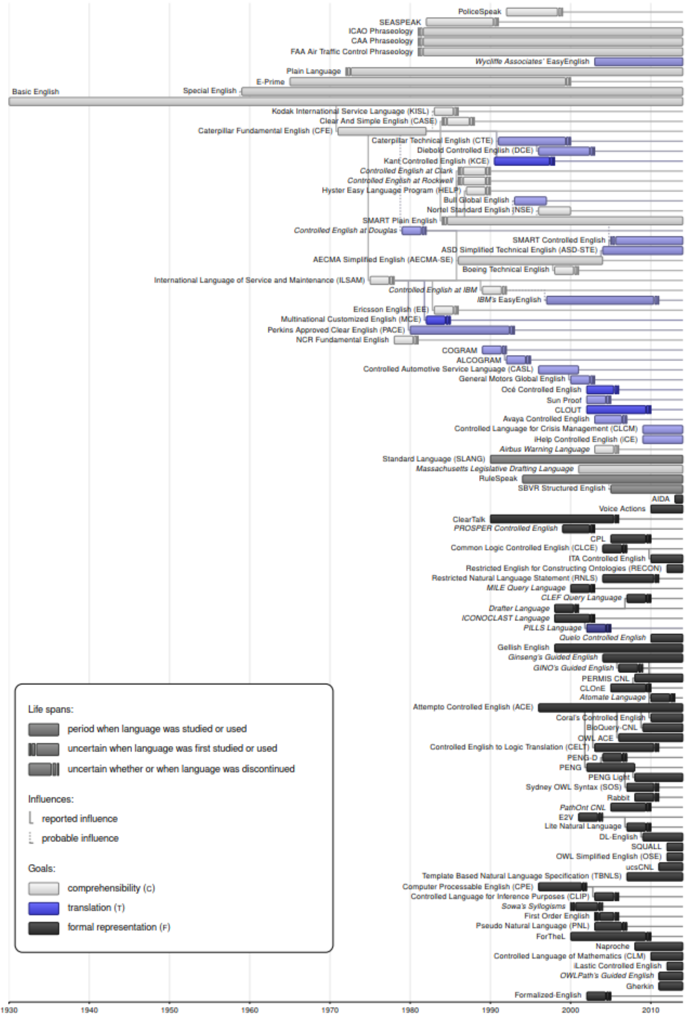
negative correlation: More expressive languages tend to be more complex ($\rho = -0.82$). In addition, naturalness/expressiveness are strongly positively ($\rho = 0.77$) and naturalness/simplicity strongly negatively correlated ($\rho = -0.76$). At a slightly lesser degree, negative correlation values are obtained for the pairs precision/naturalness ($\rho = -0.67$) and precision/expressiveness ($\rho = -0.66$). These observations seem to be in line with what one would intuitively expect.

Some CNLs

- **Syllogisms**: Every A is a B. … Some A is not a B.
- **Basic English / Simple English:** only 850 words allowed
- **Caterpillar:** „language recommendations“ do this – avoid that
  - used for instruction manuals etc.
- **Air Traffic Control Phraseology:** ~300 fixed phrases
- **Simplified Technical English:** NL with restrictions
- **SLANG:** machine-readable instructions
- **OMG SBVR Structured English / RuleSpeak:** extensible sentence constituents, formal semantics
- **Attempto Controlled English:** DRT semantics
- …

Controlled Natural Languages

Hundred CNLs



- Timeline 1930-2010
- PENS classification
- Evaluation of comprehensibility, translatability and formality

Design Criteria

- Is the language easy to describe, teach, and learn?
- Is it easy to read and scan? easy to write? to understand?
- Is the semantics predictable and unambiguous?
- Is the syntax defined formally or informally?
- How are semantic restrictions handled?
- Are statements translatable into a logic?
- Are discourses translated into a logic?
- What are the formal properties of the language?
- How is the language evaluated?
- Is the language easily and systematically extensible?
- …

Attempto for Requirements

```
The customer enters a card and a numeric personal code.
If it is not valid then SM rejects the card.
```

- sentences of the form subject – verb – object
- if-then sentences, references, …
- Translation to PROLOG
- yes/no queries, wh-queries

```
[A, B, C, D]
customer(A)
card(B)
enter(A, B)
numeric(C)
personal_code(C)
enter(A, C)
named(D, simplemat)
IF:
  []
  NOT:
    []
    valid(C)
THEN:
  []
  reject(D, B)
```

```
fact(customer(0)).
fact(card(1)).
fact(enter(0, 1)).
fact(numeric(2)).
fact(personal_code(2)).
fact(enter(0, 2)).
fact(named(3, simplemat)).
fact((reject(3, 1):-neg(valid(2)))).
```

Executing Attempto

- yes/no queries, wh-queries

```
Does the customer enter a card?
yes
Is the personal code numeric?
yes
Who enters a card?
Result:  who = a customer
Answer:  a customer enters a card.
```

Example Specification B

The customer enters a card and a numeric personal code that SimpleMat checks.

If the personal code is valid then SM accepts the card.

If the personal code is not valid then SM rejects the card.

Info: Execution

```
user:   customer1 is a customer
user:   card1 is a card

event:  customer1 enters card1

user:   code1 is a numeric personal_code

event:  customer1 enters code1

user:   s1 is a simplemat

event:  s1 checks code1

user:   code1 is not valid

event:  s1 rejects card1
```

The Domain-Specific Language LESS

Typical Safety / Security Requirements

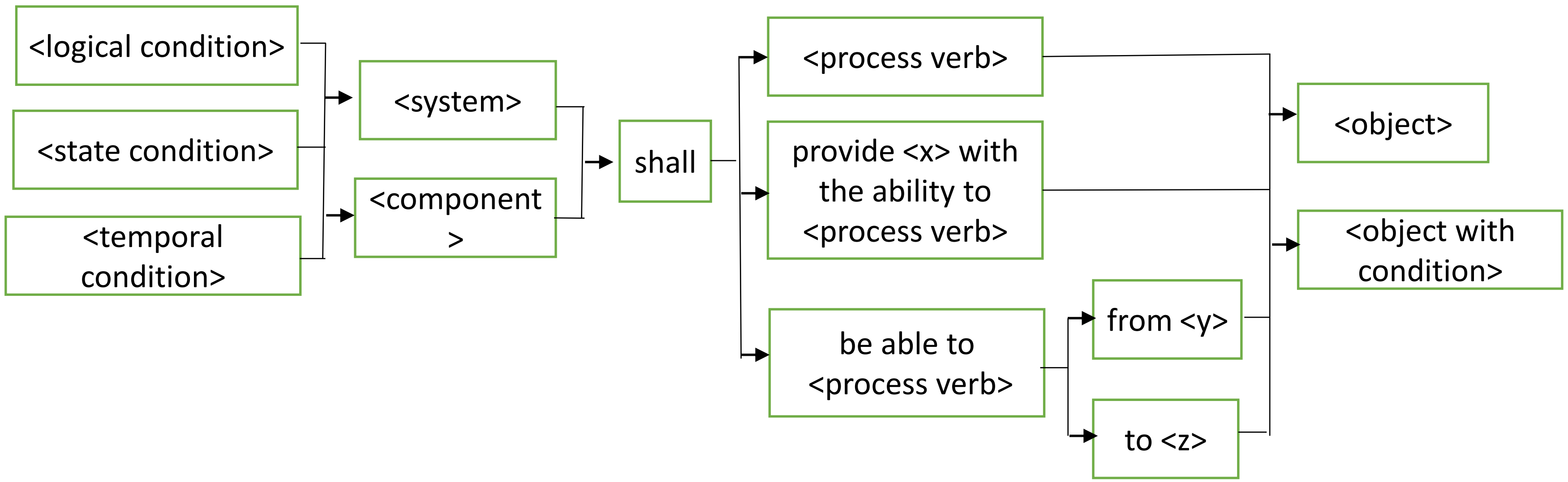
- In any state, the system must prevent unintended acceleration.
- Upon login attempt, the server must ensure authorized login.

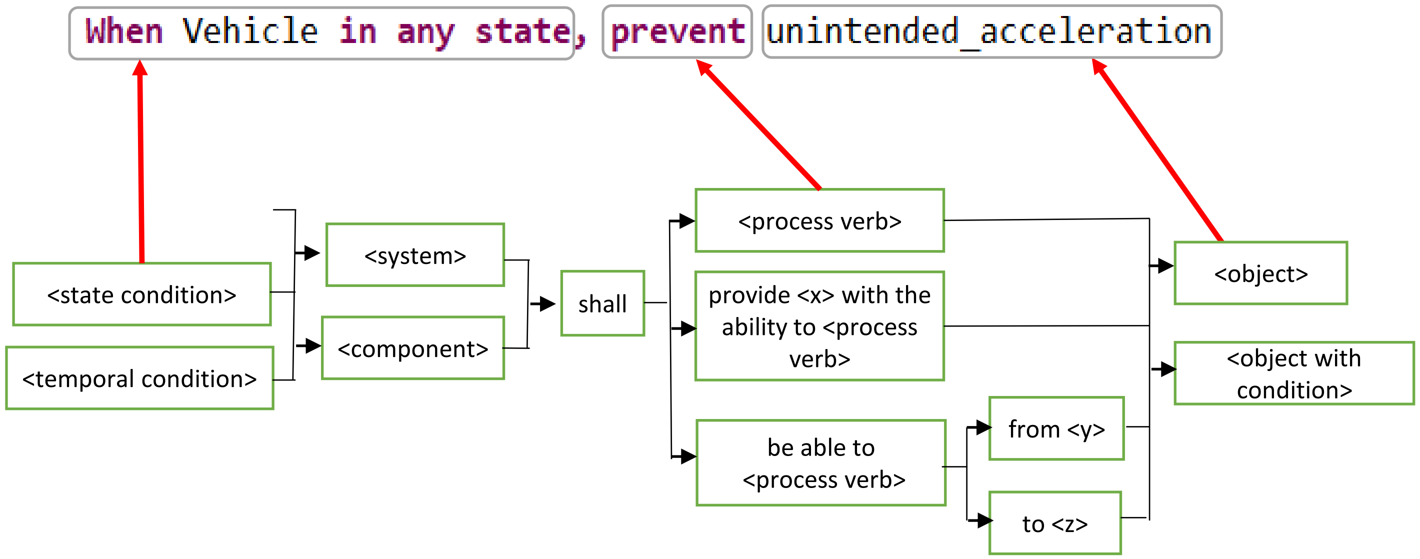
- If the temperature is too low, the error alarm must be triggered.
- If there is a failed login attempt, a log entry recording must be made.

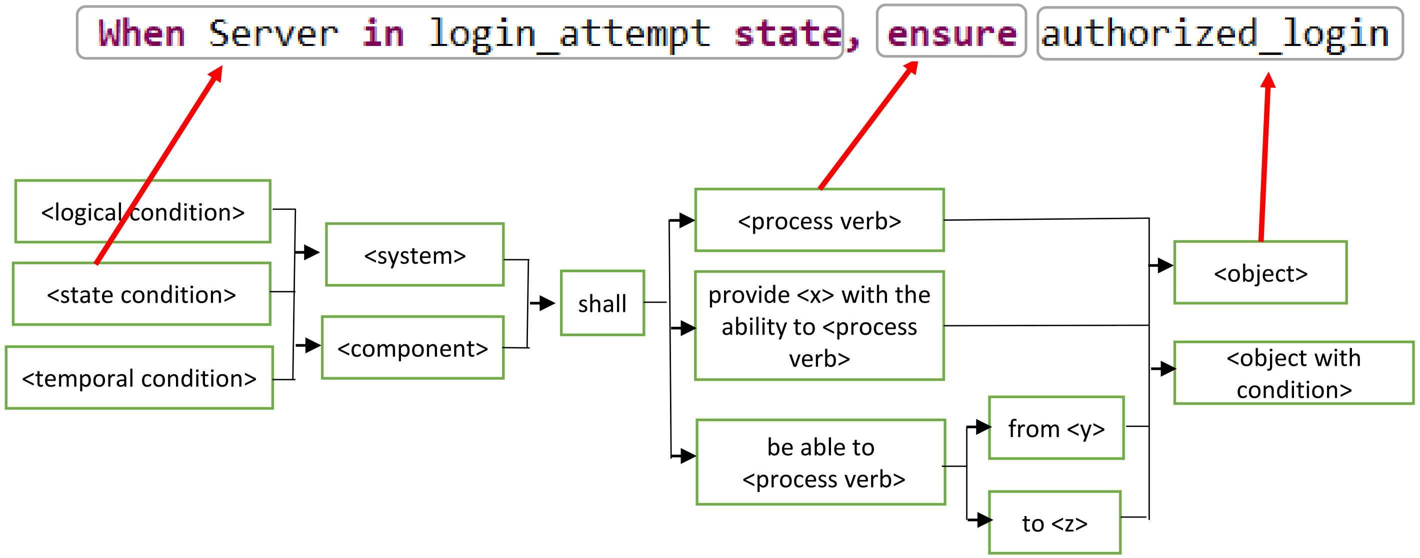
- The charging approval shall be given if the connection with the charging station is active.
- Setting of control variables is possible only if the user is authorized.

A DSL for Specifying Safety and Security

- Developing a DSL for this purpose
- First approach







Embedded SafeSec

E-Gas Case Study

- Standardized E-Gas Monitoring Concept for Gasoline and Diesel Engine Control Units' (E-Gas concept).
  - a variety of functional and safety requirements for engine control units at different levels of abstraction
- *Example: [SReq-01*] 'Sensors shall be plausibility checked' (Component: Drive pedal)*
- ***NL specification must be reworded to***
  'The Drive_Pedal SHALL check the sensor_ signals of the Drive_Pedal for plausibility'.

ID	…	5	6	7	…	10	11	…
[SReq-01]		The Drive_Pedal	SHALL	check		the sensor_signals of the Drive_Pedal	for plausibility.	

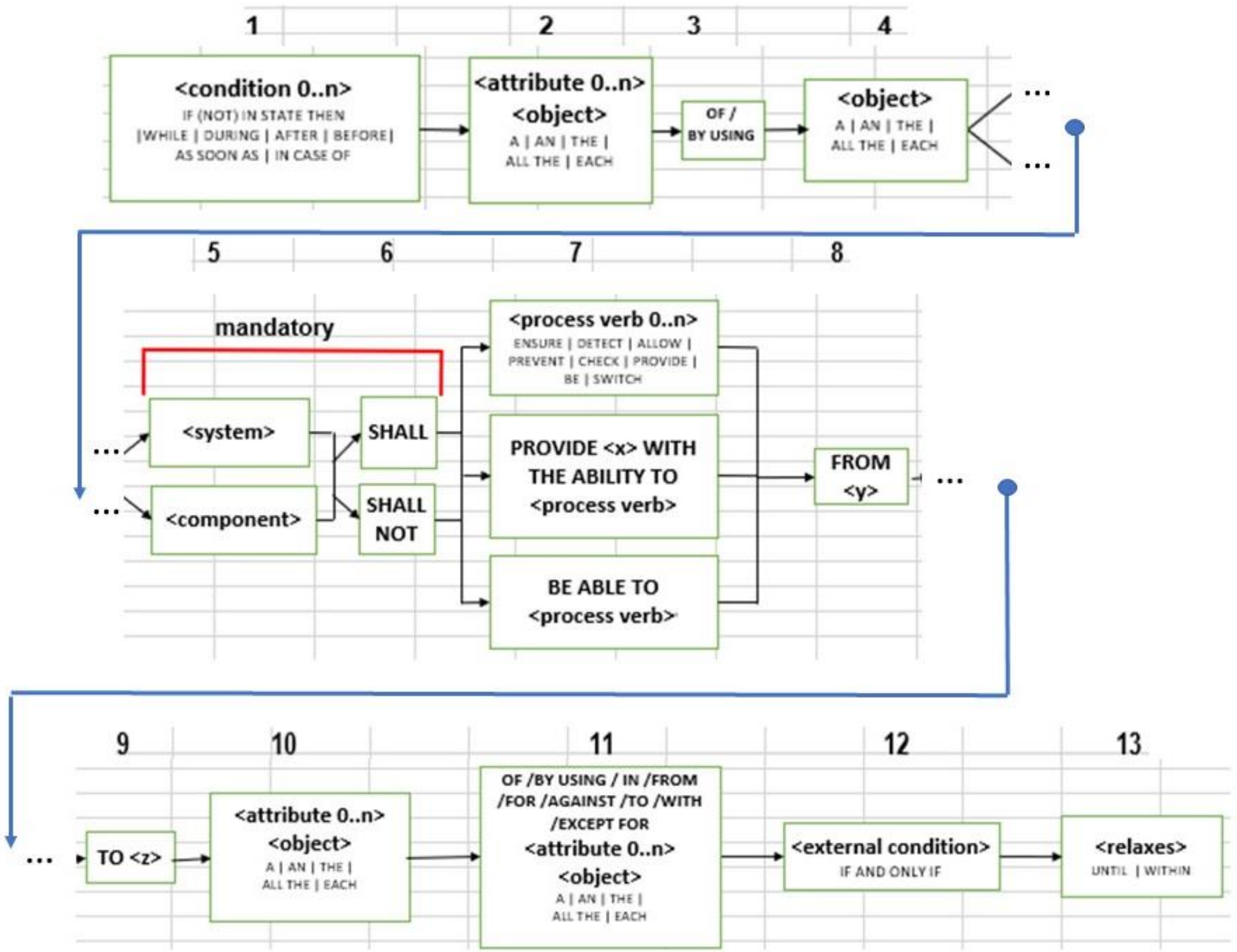
E-Gas Case Study

- *[SReq-06*] 'A safety concept shall be implemented in the engine control unit which detects and confirms undesired states of a high driving torque or an unintended acceleration. In case of a fault the engine control unit shall switch to a safe state.'*

ID	1	…	5	6	7	…	10	11	…
[SReq-06a.1]			The Engine_Control_Unit	SHALL	detect		undesired states	of High_Driving_Torque.	
[SReq-06a.2]			The Engine_Control_Unit	SHALL	confirm		undesired states	of High_Driving_Torque.	
[SReq-06a.3]			The Engine_Control_Unit	SHALL	detect		an Unintended_ Aacceleration.		
[SReq-06a.4]			The Engine_Control_Unit	SHALL	confirm		an Unintended_ Aacceleration.		
[SReq-06b]	In case of a fault		The Engine_Control_Unit	SHALL	switch to		a safe state.		

LESS

The LESS Template



Embedded SafeSec

E-Gas Case Study

- *[SReq-04*] 'Torques affecting requirements of other ECUs shall be protected in a signal compound of the engine control unit.'*
  'The engine control shall protect torques affecting requirements of other ECUs in a signal compound.'
- *'Torques affecting requirements of other ECUs in a signal compound'* does not match the available pattern
- Quoting mechanism (not interpreted by the language analysis tool)

ID	…	5	6	7	String
[SReq-04]		The Engine_Control_Unit	SHALL	protect	"torques affecting requirements of others ECUs in a signal compound"

- This allows to formulate 99% of the requirements given in the case study

VAD Case Study

- Ventricular Assist Device, which pumps blood to support a human heart
  - Alternative domain, many security requirements
- *[Req_BH40] 'The clinical user interface shall be able to switch from manual to auto mode, if and only if the user is logged in.'*

ID	...	5	6	7	8	9	...	12
[Req_BH40]		THE clinical_UI	SHALL	BE ABLE TO switch	FROM manual_mode	TO auto_mode		IF AND ONLY IF THE user IN logged_in

- Also for this case study, most requirements could be formulated without problems

Refinement and Testing Methods based on LESS

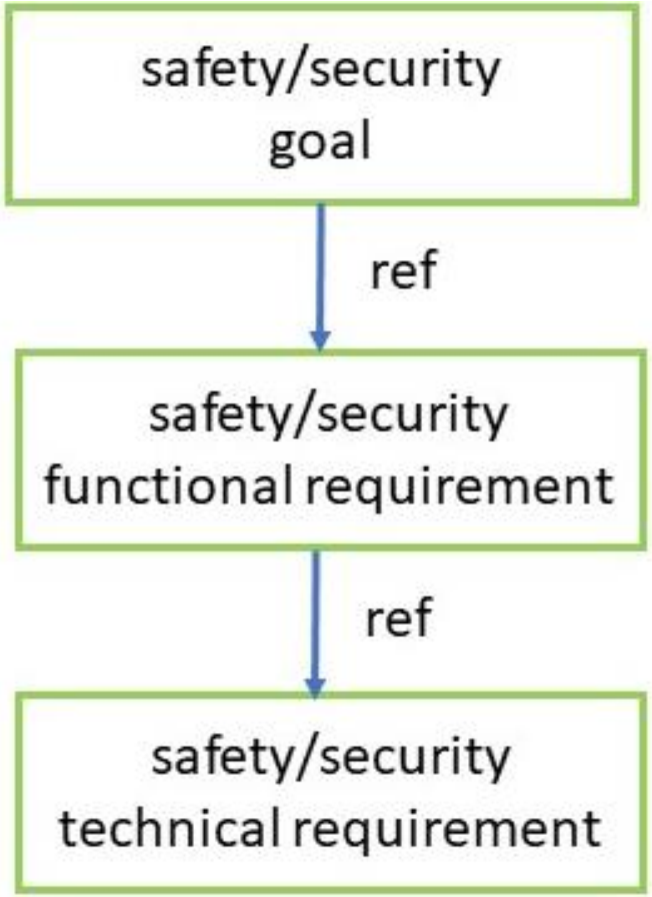
Three Steps to Analyze and Refine LESS Requirements

1. Analysis of the **syntactical structure** of the safety and security requirements expressed in a LESS specification document.
2. Designing **controlled conversations** with the user in order to achieve further semantic information needed for the analysis, refinement and derivation activities mentioned above.
3. Processing of the user's response and **generating the result**, which can be a consistency or completeness verdict, a refined requirement, or a test case.

Detailing and Refining S&S Requirements

- “Interaction wizard”
- Based on an interactive process, where the machine supports the human by asking relevant questions
- Responsibility at any moment rests with the safety/security engineers
- No need of formal tool qualification
- Possibility of extensions by more use cases

- Machine has parsed LESS requirements and asks questions about them



Example: Dealing with Vagueness

- Rule: “In case of a vaguely formulated condition, ask for a more detailed condition”
- Dialogue:

```
[SReq-06b] ‘In case of a fault the Engine_Control_Unit SHALL switch
to a safe state.’
(Functional Safety Requirement, ASIL B)

[SReq-06b] refers to the case of a ‘fault’.
Would you like to further define a ‘fault’? [Yes/No/Later]

>  Y
>  ‘fault’ := ‘Unintended_Acceleration’

[SReq-06b.1] ‘In case of Unintended_Acceleration the
Engine_Control_Unit SHALL switch to a safe state.’
(Functional Safety Requirement, ASIL B)
```

Example: Dealing with Missing Definitions

Embedded SafeSec

- Rule: “In case a requirement mentions a safe state, ask for detailed definition of it”
- Dialogue:

```
[SReq-06b.1] ‘In case of Unintended_Acceleration the
Engine_Control_Unit SHALL switch to a safe state.’
(Functional Safety Requirement, ASIL B)

[SReq-06b.1] refers to a safe state. Would you like to detail the
safe state mentioned in [SReq-06b.1]? [Yes/No/Later]

>  Y
>  [SS-01] := Injection driver of the gasoline engine is switched off
   (Safe State)

Would you like to review / update the original requirement [SReq-
06b.1] as well? [Yes/No/Later]

>  Y
>  [SReq-06b.1.1] ‘In case of Unintended_Acceleration the
   Engine_Control_Unit SHALL switch into Safe State [SS-01].’
   (Functional Safety Requirement, ASIL B)
```

Checking Consistency and Completeness

Embedded SafeSec

- Currently: identification of missing steps in a chain of actions
- Ex.: [SReq-06a.2] ‘The Engine_Control_Unit SHALL confirm undesired states of High_Driving_Torque. (Functional Safety Requirement, ASIL B)
- Here, we find a requirement with the verb 'confirm’

```
[SReq-06a.2] refers to the confirmation of 'undesired states of
High_Driving_Torque'.
Would you like to specify a subsequent action? [Yes/No/Later]

>  Y
>  [SReq-06a.2.1] In case of confirmed undesired states of
   High_Driving_Torque the Engine_Control_Unit SHALL switch
   into a safe state.
   (Functional Safety Requirement, ASIL B)
```

- New requirement is added to the data base

Test Case Generation

Embedded SafeSec

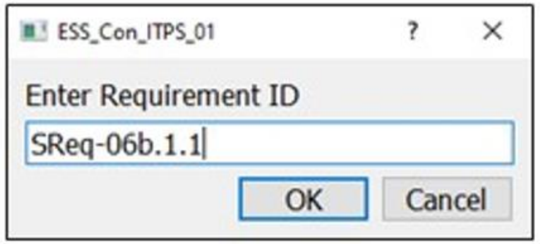
- Semi-automatic generation of test cases
- If a requirement mentions actions which are to be performed in certain states, the user is asked how to reach that state and how to confirm that the intended action has been performed
- Example:

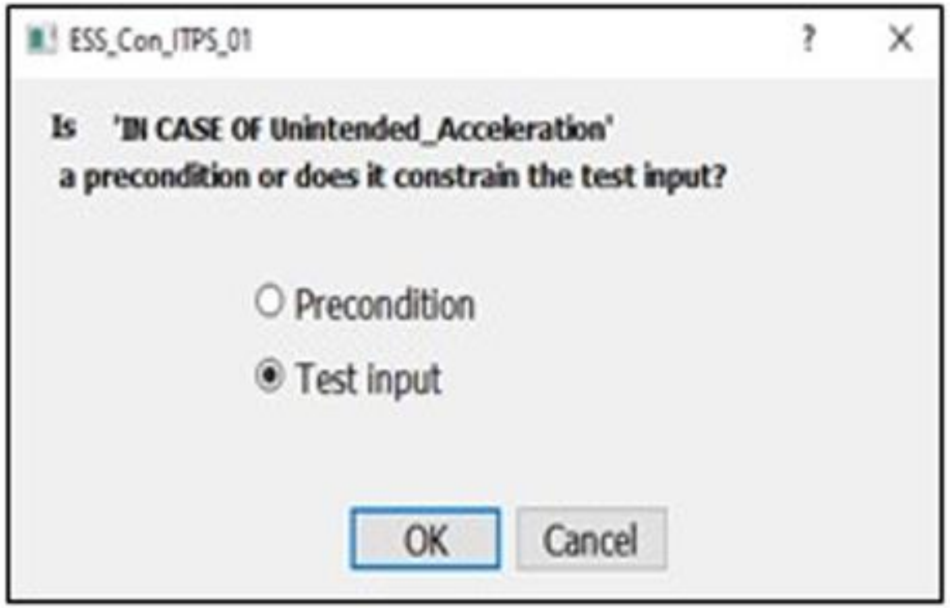
```
(1)   [SReq-06b.1.1] ‘In case of Unintended_Acceleration the
      Engine_Control_Unit SHALL switch into Safe State [SS-01].’
      (Functional Safety Requirement, ASIL B)
(2)   [SS-01] ‘Injection driver of the gasoline engine is switched
      off’
      (Safe State)
```

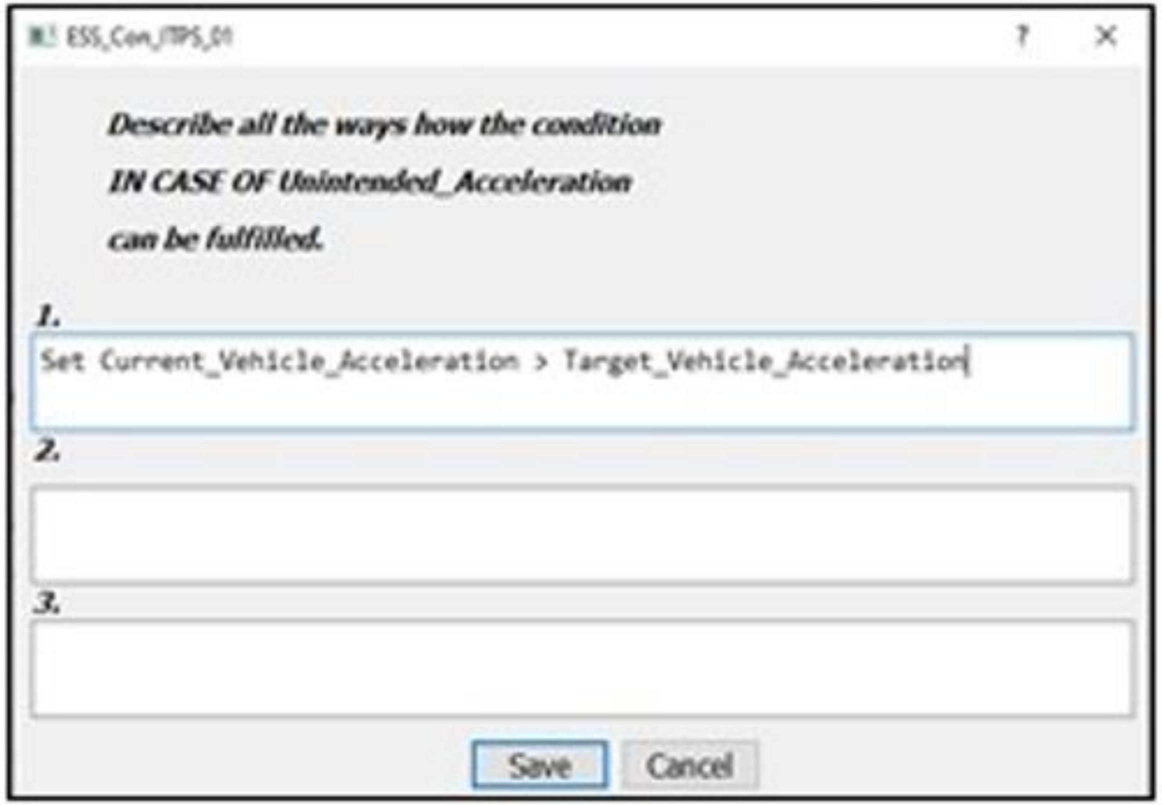
Here is the generated testcase:

Test ID	Precondition	Test Input	Expected Behaviour
TC_001 (SReq-06b.1.1)	System is running	1. Set Current_Vehicle_Acceleration > Target_Vehicle_Acceleration	1. Engine_Control_Unit SHALL switch to Safe State [SS-01].

Test Case Generation Tool Support

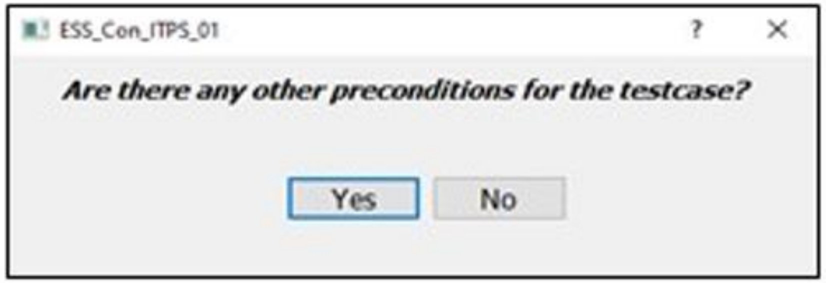


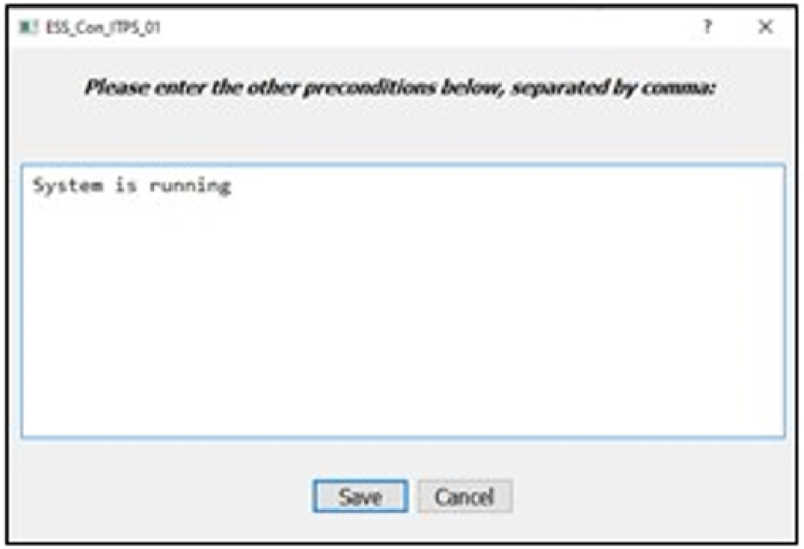


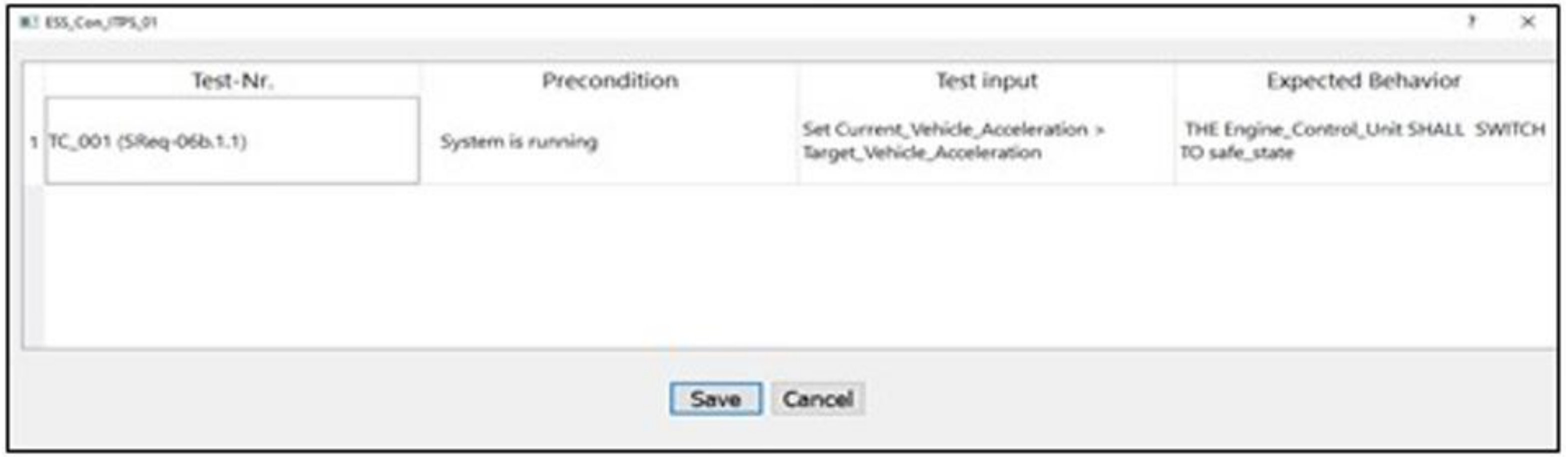


Refinement and Testing Methods based on LESS

Test Case Generation Tool Support







More Details

Embedded SafeSec

SQ Magazin June 2023

VDA Conference 2023

Joint Specification and Testing of Safety and Security Requirements

Sadegh Sadeghipour[1], Holger Schlingloff[2], Mirko Conrad[3], Harald Schülzke[3]

[1] ITPower Solutions GmbH, Berlin, Germany

[2] ZeSys e.V. and Fraunhofer FOKUS, Berlin, Germany

[3] samoconsult GmbH, Berlin, Germany

Abstract

Violation of safety or security in modern highly networked and automated devices and functions, such as those used for the Internet of Things, Industry 4.0, and autonomous driving, can lead to catastrophic consequences for people and the environment. Therefore, the development process of embedded systems and software is associated with demanding requirements regarding safety and security.

While the discipline of safety engineering is well established and supported by international standards like IEC 61508, ISO 26262, and EN 5012x, security engineering and its interaction with the safety process in the field of embedded systems is still in early phases of its development.

In this paper we present a methodology for the joint specification of safety and security requirements of embedded systems, and the derivation of test cases. Currently, safety and security are treated in two separate engineering processes. The advantage of a process for specification both safety and security at the same time is that possible redundancies and inconsistencies between safety and security requirements can be identified at an early stage.

The core of the methodology presented here is a domain-specific language (DSL) called LESS (Language for Embedded Safety and Security), which is based on natural language templates often used in requirements engineering. With the aid of a few simple rules and a small set of keywords users can define safety and security requirements in a formalized way without any need for a difficult-to-understand mathematical or a complex graphical notation.

We also present a set of methods that form the basis for implementing semi-automatic procedures for analyzing and refining requirements as well as deriving test cases from them. These methods are based on an analysis of the syntactical structure of the safety and security requirements expressed in LESS, and on the design of controlled conversations with the user. The results of the conversations are used to achieve further semantic information needed for the analysis, refinement and derivation activities mentioned above. Some of the developed methods have been implemented as prototypes and applied to case studies from automotive and medical technology.

Due to the easy-to-learn and well-understandable domain-specific language LESS, as well as the controlled wizard-like conversations with the user, the methodology presented here possesses a high potential to be used in all industrial sectors where safety- and security-related applications are developed.

Keywords: Safety and Security Co-Engineering, Language for Embedded Safety and Security (LESS)

---

* The described activities were conducted as part of the EmbeddedSafeSec project (Mar. 2021 – Dec. 2022). The EmbeddedSafeSec project was funded by the Investitionsbank Berlin (IBB) program for the promotion of research, innovation and technology – ProFIT – and by the European Regional Development Fund (ERDF).

1

Next Steps

HO **Du**

A so-called "Chris-Rupp-Sentence" consists of the following components: First a precondition, then the words "the system", the one of the words "shall", "should" or "will", then a process, and finally an object or process detail. A process is a word describing an activity, or the words "be able to" followed by a word describing an activity, or the word "provide" followed by an object followed by the words "the ability to" followed by a process. Is the following a Chris-Rupp-Sentence? "If the signal is on, the system should provide the student the ability to submit the thesis."

ChatGPT

Yes, the given sentence fits the structure of a Chris-Rupp-Sentence. Let's break it down:

- Precondition: "If the signal is on,"
- "the system"
- "should" (indicating a requirement)
- Process: "provide"
- Object: "the student"
- Process Detail: "the ability to submit the thesis."

So, the sentence follows the specified format for a Chris-Rupp-Sentence.

Me:

Can you re-formulate the following as a Chris-Rupp-Sentence? "Let us assume that the signal is set to on. Then the intended system must be programmed such that it offers students a possibility to hand in their work for evaluation as a thesis."

ChatGPT

Certainly! Here is the re-formulated sentence in the Chris-Rupp-Sentence format:

"If the signal is set to on, the system shall be programmed to offer students the ability to hand in their work for evaluation as a thesis."

ITEA Genius

- A new project to be submitted in 2 weeks
- >5 European countries, ~35 partners
- 3 main focus points:
  - Generation of (formal) requirements from natural language
  - Generation of code artifacts from natural language
  - Generation of test cases from natural language
- More to come!

Summary and Conclusion

- An easy-to-learn and easy-to-understand domain-specific language: LESS
- Refining and analyzing requirements as well as generating test cases
- Embedded into a joint safety / security life cycle
- Two case studies (E-Gas and VAD) successfully completed
- Used in first consultancy projects for automotive customers

- Work in progress: extraction of state diagrams for test case generation
- AI-based natural language dialogues
- Advancing tool support from prototype to product

Thank you for your attention!