NLP-Based Requirements Formalization for Automatic Test Case Generation

Robin Gröpler[1], Viju Sudhi[1], Emilio José Calleja García[2] and Andre Bergmann[2]

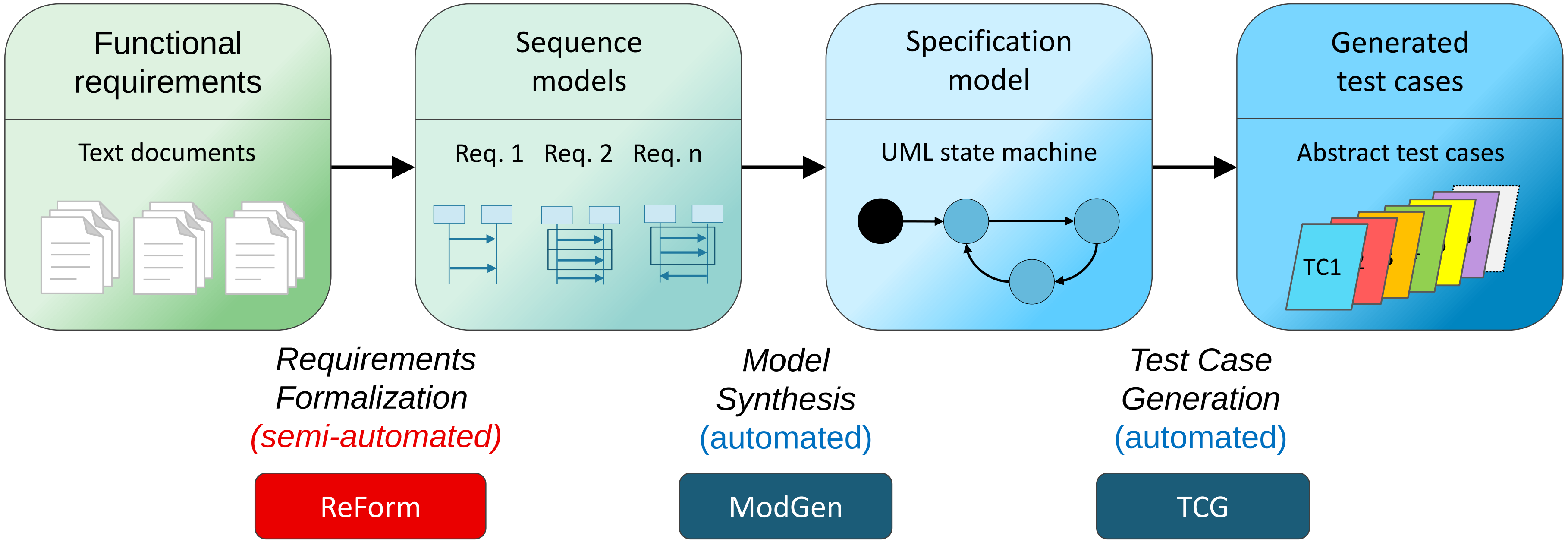
[1]ifak Institut für Automation und Kommunikation e.V., 39106 Magdeburg, Germany
[2]AKKA Germany GmbH, 80807 München, Germany

29th International Workshop on Concurrency, Specification and Programming (CS&P'21)
Berlin, Germany, 27. - 28. September 2021

Outline

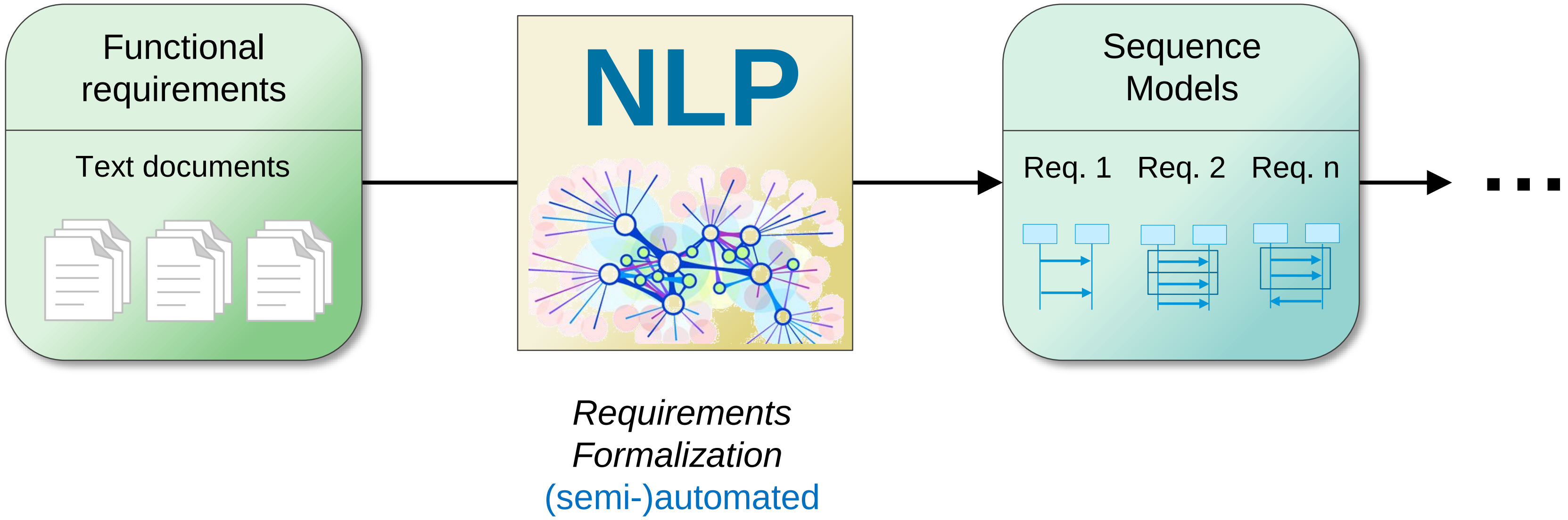
1. Introduction

- Rapidly changing and growing number of **requirements** in the life cycle of a device, component or system
- Increasing effort for verifying requirements and **testing** of the implementation
- Agile methods for **model-based testing** to manage test complexity and reduce test effort and cost
- **Toolchain** for requirements-based test case generation
- Time-consuming **manual step** is the creation of requirement models from textual requirements documents



1. Introduction

- Recent advances in **natural language processing (NLP)** show promising results
- We propose a **new, semi-automated technique for requirements-based model generation**



1. Introduction

State of the art

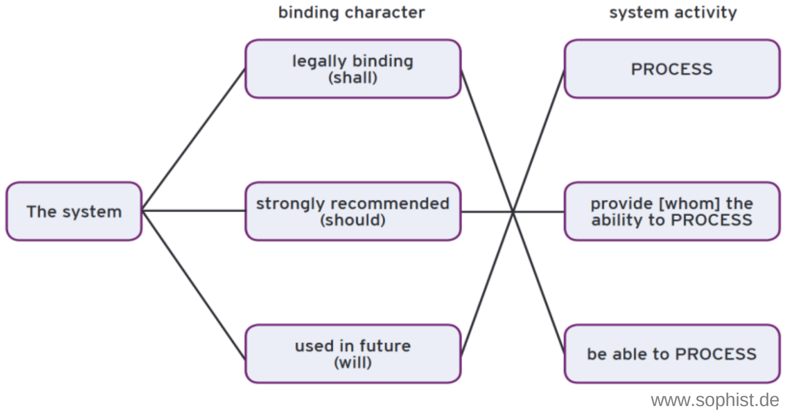
- Several NLP approaches and tools have been investigated in recent years
- Many authors restrict their NLP approaches to a prescribed format
    - Templates
    - Set of structural rules
    - Controlled natural language
- Specific output format of generated models
    - Expert knowledge for inspection needed
    - Restricted to some requirements engineers
    - Coding practices necessary

A Comprehensive Investigation of Natural Language Processing Techniques and Tools to Generate Automated Test Cases

Imran Ahsan[1], Wasi Haider Butt[2], Mudassar Adeel Ahmed[3] and Muhammad Waseem Anwar[4]

NLP-assisted software testing: A systematic mapping of the literature

Vahid Garousi[a,*], Sara Bauer[b], Michael Felderer[b,c]




```
BodySense.allInstances() ->
  forAll(b|b.occupancyStatus.occupantClassFor
  AirbagControl<>OccupantClass::Error)
VoltageError.allInstances() ->
  forAll(v|v.detected = true)
BodySense.allInstances() ->
  forAll(b|b.occupancyStatus.occupantClassFor
  AirbagControl = null)
```

1. Introduction

- **Aim:** Development of a new approach and tool
  1) Handle an **extended range of domains and formats** of requirements
  2) Provide **enhanced but easily interpretable intermediate results**

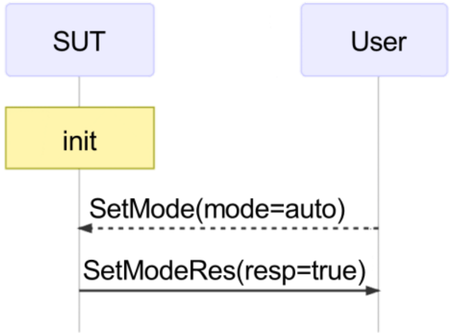
Textual representation (IRDL)

```
Requirement{
  Declaration{
    name: "normal operation";
    Component{name: SUT; description: "SUT";}
    Actor{name: User; description: "User";}
  }
  State "init" at sut;

  Message(user->sut): SetMode(mode=auto);

  Message(sut->user): SetModeRes(resp=true);
}
```

Graphical representation



- **Main contributions:**
  1) Development of a **rule-based approach based on NLP information**
  2) **Evaluation on an industrial use case** using meaningful metrics

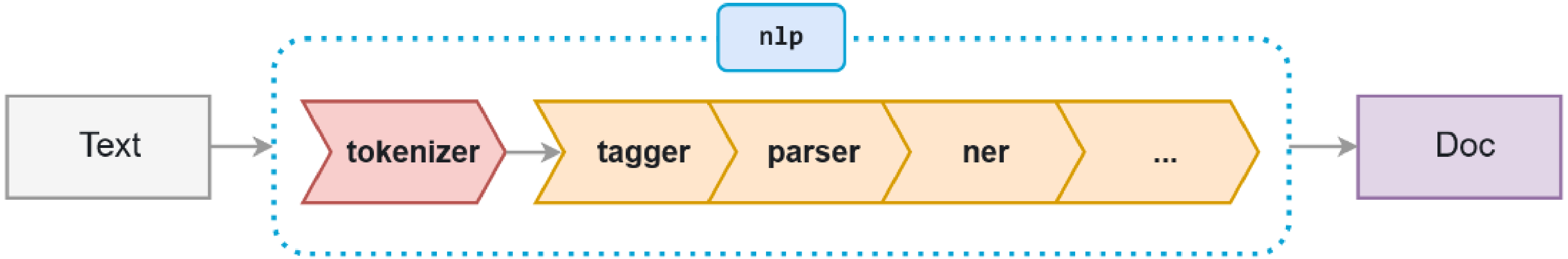
2. Related work

- NLP approaches restricted to a **specific domain or a prescribed format**:
  - Riebisch (2005): Creating activity diagrams from requirements following a **predefined structure**
  - Rupp (2014): SOPHIST method: formalization of structured texts, **text templates** with a defined structure
  - Mavin (2009): Small set of **structural rules** to address ambiguity, complexity and vagueness
  - Carvalho (2015): **Controlled natural language**, requirements from Data-Flow Reactive Systems (DFRS)
  - Allala (2019): Generate test cases from use cases or user stories, have to comply with a **specified format**
  - Nebut (2006): Formalization of use case descriptions by writing pre- and post-conditions in a **predefined format**
  - Goffi (2016): Complete missing test cases, provided the documentation is in a **specified template**
  - Blasi (2018): Artifacts that programmers create, belong to a **smaller subset** of specifications

- Specific output format of generated models
  - Wang (2020): Restricted Use Case Modeling (RUCM) specifications, **inspect** generated OCL constraints
  - Yue (2015): Restricted Test Case Modeling (RTCM) , **inspect** generated OCL constraints
  - Silva (2015): Test case generation using Petri Net simulation, **interpret** Colored Petri Nets
  - Fischbach (2020): Recursive dependency matching to formulate test cases from user stories

3. Methodology

- **Linguistic pre-processing**
  - NLP parser: spaCy
    - **Tokenization:** segment text into words, punctuations marks etc.
    - **Lemmatization:** assign the base forms of words
    - **Part-Of-Speech (POS) Tagging:** assign part-of-speech tags (noun, verb, etc.)
    - **Dependency Parsing:** assign dependency labels (parent, children tokens)



3. Methodology

- **Linguistic pre-processing**
  - Pronoun resolution
    - Pronouns are identified and resolved with the farthest subject, proposed by Lappin (1994), Qiu (2004)
    - Due to simplicity of the task, no need to use more sophisticated algorithms
    - Resolve pronouns only if the grammatical number of the pronoun agrees with that of the antecedent
    - Pleonastic pronouns (pronouns without a direct antecedent) are cited but not replaced

  - Example:

”If the temperature of the battery is below Tmin or it exceeds Tmax, charging approval has to be withdrawn.”

replace

3. Methodology

- **Linguistic pre-processing**
  - Decomposition
    - Requirements with **multiple conditions and conjunctions** need to be mapped to individual relations
    - Decomposition of complex requirements into simple clauses
      - Multiple conditions (sentences with multiple if's, while's, etc.)
      - Root conjunctions (sentences with multiple roots connected with a conjunction)
      - Noun phrase conjunctions (sentences with multiple subjects and/or objects connected with a conjunction)

➢ Example:

"If the temperature of the battery is below Tmin or the temperature of the battery exceeds Tmax,

charging approval has to be withdrawn."

3. Methodology

- **Syntactic entity identification**
  - Behavioral requirements describe a particular **action** (linguistically, verb) done by an **agent** (linguistically, subject) on the system of interest (linguistically, **object**)
    - **Action**: Main action verb in the requirement
      - Nominal action, Boolean action, simple action
    - **Subjects and Objects**: Tokens with dependencies subj and obj (and their variants like nsubj, pobj, dobj, etc.)
      - Noun chunks, compound nouns, single tokens
  - Resolve **logical comparisons** (e.g. greater than, exceeds, etc.)
    - **Synonym** hyperlinks from Roget's Thesaurus
    - **Map** to corresponding equality (=), inequality (!=), inferiority (<, <=) and superiority (>, >=) symbols

➢ Example: *"[if the temperature of the battery is below Tmin], [charging approval has to be withdrawn]"*

- *the temperature of the battery* → Subject: Battery_Temperature
- *is below* → <
- *Tmin* → Object
- *charging approval* → Subject: Charging_Approval
- *withdrawn* → Boolean action

3. Methodology

- **Semantic entity identification**
  - **Mapping** of syntactic to semantic entities
    - **Actor or Component:** Participants involved in an interaction
    - **Signal:** Interaction between different participants
    - **Attributes:** Variables holding the status of an interaction
    - **State:** Initial, intermediate and final states of an interaction
  - **More complex rules** used (including action and verb types)

Syntactic entities	Semantic entities
Action	Signal
Action constraints	Attributes
Subject / Object	Actor / Component

➢ Example: *"[if the temperature of the battery is below Tmin], [charging approval has to be withdrawn]"*

- *the temperature of the battery* → Subject: Battery_Temperature — Signal — Actor: battery
- *is below* → <
- *Tmin* → Object — Attribute
- *charging approval* → Subject: Charging_Approval — Signal — Actor: unknown
- *withdrawn* → Boolean action — Attribute

3. Methodology

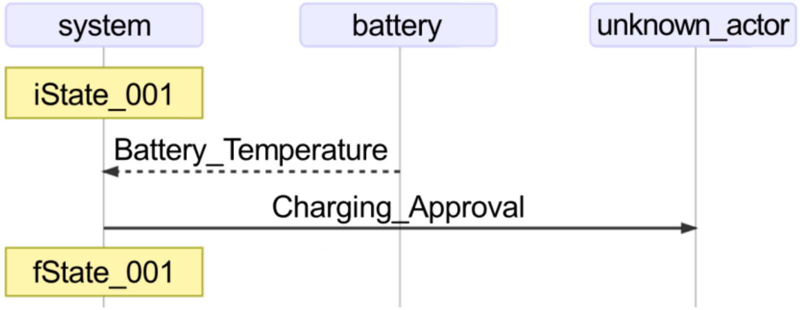
- **Transformation to requirement model**
  - Textual representation: ifak requirements description language (IRDL)
    - Simple text-based domain-specific language (DSL)
    - On the basis of UML sequence diagrams, a series of model elements and structures are defined
  - Graphical representation: Generated sequence diagram
  - Create IRDL relations for each clause and then combine them together
    - Incoming messages: SUT receives these messages provided the guard expression evaluates to be true
    - Outgoing messages: SUT sends these messages to other interaction participants

➢ Example:

Textual representation (IRDL)

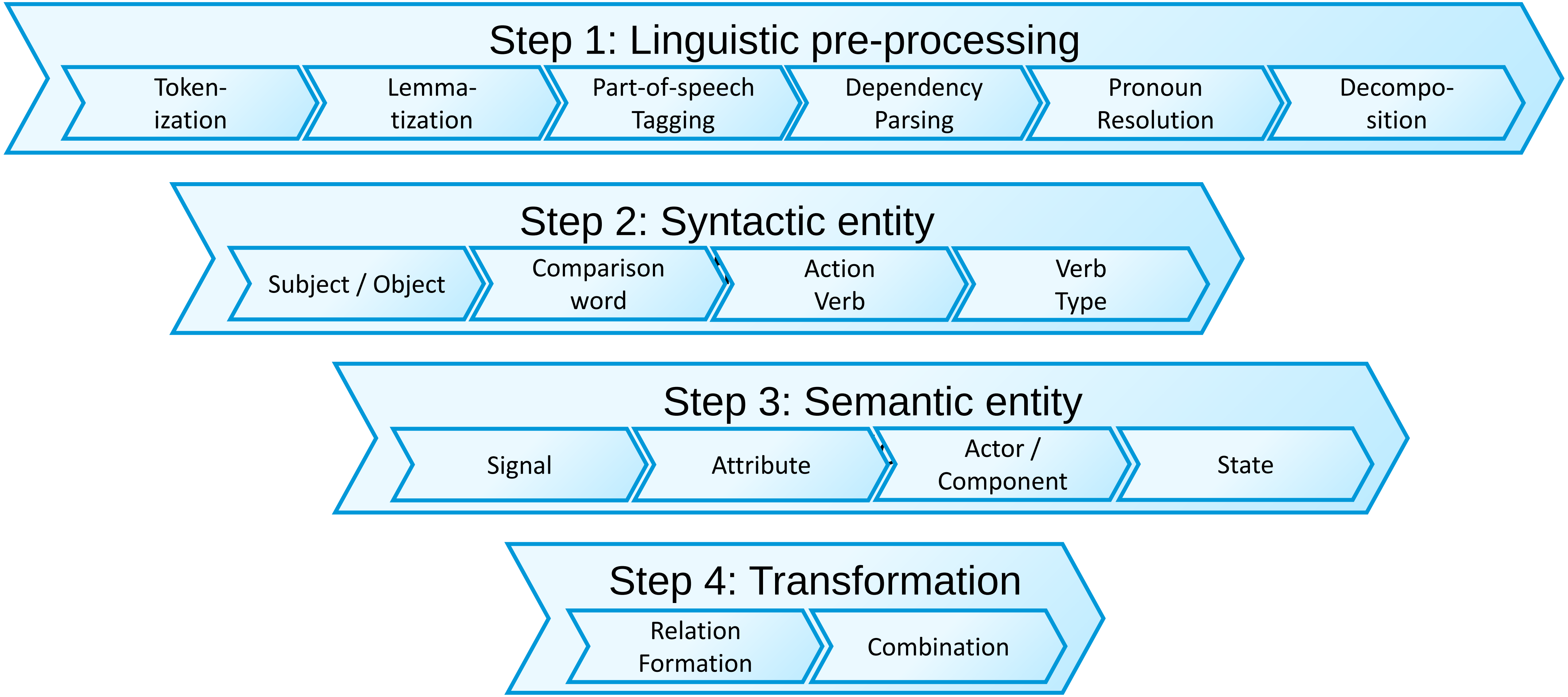
```
State iState_001 at system;
Check(battery->system):Battery_Temperature
  [msg.value < Tmin || msg.value > Tmax];
Message(system->unknown_actor):
  Charging_Approval(false);
State fState_001 at system;
```

Graphical representation



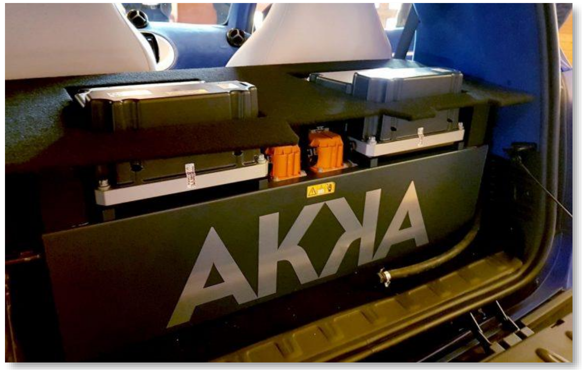
3. Methodology

- Steps in our NLP approach:



4. Application

- **Industrial use case** from the e-mobility domain defined by AKKA
- **Charging approval system** of an electric vehicle with a charging station
- Aims to provide a typical **basic scenario** and development workflow in software development for an automotive electronic control unit (ECU)
- Defining **requirements**, using model-based software development and deploying the functionality on an ECU
- Function “charging approval” implements a simple function, decides upon specific input signals, if the charging process of the battery is allowed or not
- For example, charging approval is given or withdrawn depending on the
  - battery temperature, voltage or state of charge
  - the requested current is adjusted according to the battery temperature
  - error behavior is handled for certain conditions



5. Results

- Evaluation metrics
  - R — set of textual requirements
  - X_r — set of expected artifacts (semantic entities) for a requirement $r \in R$
  - Y_r — set of generated artifacts (semantic entities) for a requirement $r \in R$
  - $X = \bigcup_{r \in R} X_r$ — set of expected artifacts in all requirements
  - $Y = \bigcup_{r \in R} Y_r$ — set of generated artifacts in all requirements

- Definition of **metrics** to measure the performance of the method:
  1) **Completeness:** For an individual requirement, this metric denotes the number of expected artifacts $x \in X_r$ for which a corresponding (not necessarily identical) generated artifact $y \in Y_r$ exists, in relation to the total number of expected artifacts $|X_r|$.
  2) **Correctness:** For an individual requirement, this metric denotes the number of generated artifacts $y \in Y_r$ for which a corresponding, semantically identical (up to naming conventions) expected artifact $x \in X_r$ exists, in relation to the total number of generated artifacts $|Y_r|$.
  3) **Consistency:** This metric denotes the number of generated artifacts $y \in Y$ for which a corresponding expected artifact $x \in X$ exists and is used identically in all requirements $r \in R$, in relation to the total number of generated artifacts $|Y|$.

5. Results

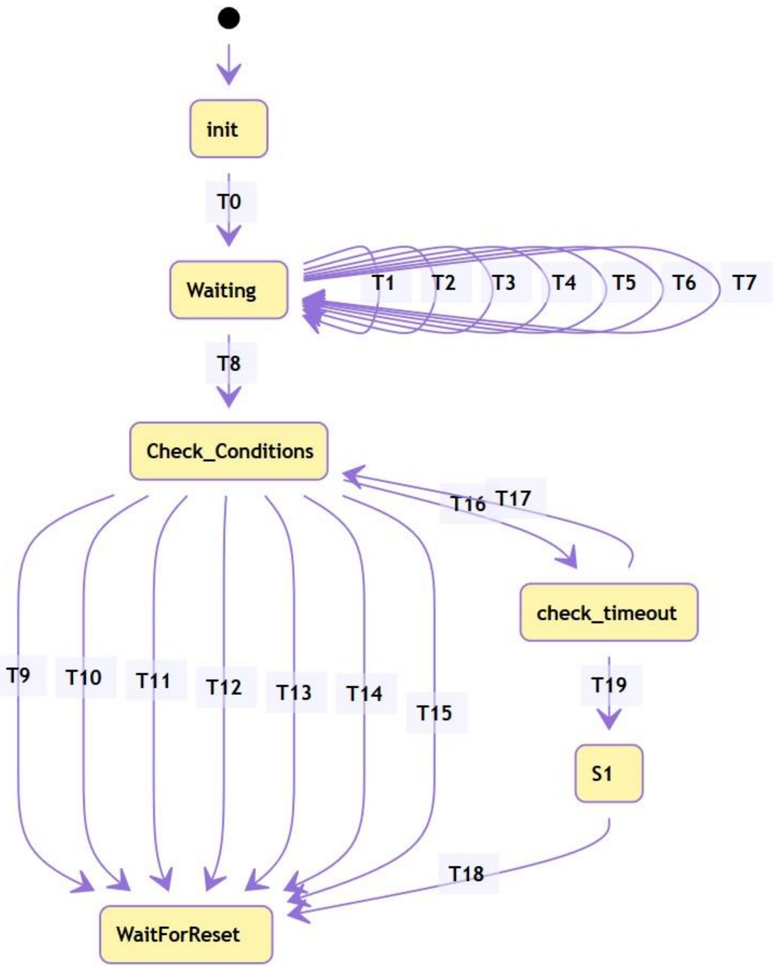
- **Evaluation** of the algorithm on the charging approval system
  - Charging approval SUT is described by **14 separate requirement statements**
  - Domain knowledge: **Predefined list** of signals, attributes, etc. or integrated by direct **user interaction** from an expert with knowledge about the system

	without domain knowledge		with domain knowledge	
	macro avg.	**micro avg.**	**macro avg.**	**micro avg.**
Completeness	78.2%	79.8%	81.4%	84.1%
Correctness	74.9%	78.8%	78.3%	82.1%
Consistency	94.1%		96.4%	

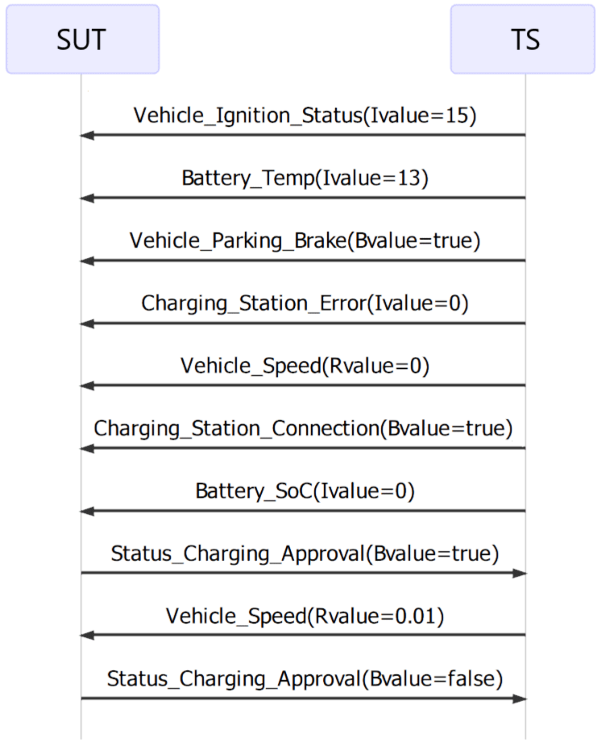
- Method shows **good results**, most of signals and other artifacts were detected **correctly and completely**
- Having a list of artifact declarations in advance produces even more accurate predictions
- Should **save a lot of time**, manual creation of sequence diagrams takes a lot of time by reading documentations and having discussions, to create the logical structure and to add all the details

5. Results

- **Model synthesis and test generation**
  - Requirement models as input for model synthesis using ifak's prototypical tool ModGen
  - UML state machine
    - 6 states
    - 20 transitions with appropriate signals, guards and actions
  - ifak's prototypical tool TCG with coverage criteria "all-paths"
    - 73 test cases



UML state machine of charging approval



Exemplary test case

6. Conclusion

- **NLP-based method** for machine-aided model generation from textual requirements
  - Cover a **wide range** of requirements formulations without being restricted to a specific domain or format
  - Generated requirement models are given in a **user-friendly, comprehensible** representation
- **Evaluated** our approach on the industrial use case of a battery charging approval system
  - Produce **complete, correct and consistent** artifacts to a high degree
  - **Reduce the human effort** of creating test cases from textual requirements

- **Outlook:**
  - **Refine** the rule-based approach further, reducing the need for manual modifications
  - Identify **semantic entities** by
    - Training of a Named Entity Recognition (NER) algorithm: Intensive labelling work
    - Semantic Role Labeling (SRL): Pre-trained models available
  - Consider **further application domains**, such as in rail, industrial communication and automotive

Acknowledgments

- This research was funded by the German Federal Ministry of Education and Research (**BMBF**) within the ITEA 3 projects
- **TESTOMAT** under grant no. 01IS17026G
- **XIVT** under grant no. 01IS18059E
- Thanks to our former colleague **Martin Reider** and our research assistant **Libin Kutty** from ifak for the valuable contributions to this paper
- Thanks to **AKKA** Germany GmbH for providing an industrial use case for the evaluation of the presented method

Thank you for your attention!

Institut für Automation
und Kommunikation e.V.
Werner-Heisenberg-Str. 1
39106 Magdeburg
Germany

Contact person:
Robin Gröpler
Email: robin.groepler@ifak.eu
Telephone: +49 (0) 391 9901 451
https://www.ifak.eu/