NLCI
A Natural Language Command Interpreter

Mathias Landhäußer, Sebastian Weigelt, Walter F. Tichy
There is plenty of discussion about how to make software development… more efficient, faster, and easier.
But scarcely 0.3 percent of the global population have at least rudimentary programming skills\textsuperscript{1,2}.

\textsuperscript{1} “There are 18.5 million software developers in the world - of which 11 million are professionals and 7.5 million are hobbyists.”
\textsuperscript{2} http://www.techrepublic.com/blog/european-technology/there-are-185-million-software-developers-in-the-world-but-which-country-has-the-most/
On the other hand, there are approximately…

**2 billion** PCs\(^3\) and **7 billion** tablet computers and smartphones\(^4\).

3) [http://www.worldometers.info/computers/](http://www.worldometers.info/computers/)
In other words, there are at least \textbf{9 billion} programmable devices, \textbf{1.2} for each human on earth\textsuperscript{5}.

...and only \textbf{0.3} percent are capable to \textbf{exploit} the \textbf{full potential} of these devices!

\textsuperscript{5} \url{http://www.worldometers.info/world-population/}
How to *empower* the remaining 99.7 percent?
“The only way a person can truly concentrate on his problem and solve it [. . .] are if he is able to communicate directly with the computer without having to learn some specialized intermediate language.”

Jean E. Sammet, 1966
Objective

Input: Englisch prose

NLCI

Output: Source Code

Natural Language Command Interpreter
The bunny says, "Hello". Alice answers to the bunny. Alice says, "???". The bunny jumps three times and then says, "Come over here". Alice shakes her head. The bunny says, "Come on! What time is it?". Alice goes to the bunny. While she goes, she says, "Ohh" and after a little while, "Okay". A cat with a broad grin appears on the tree. The cat turns its head to the bunny and says, ":-D". Alice and the bunny turn their heads to the cat at the same time. The bunny turns its whole body to face the black hole. While it turns, it says, "Ahhhhhh!". The bunny jumps to the hole and then jumps into the hole. While the bunny jumps, Alice turns to the bunny and says, "Hey, Wait".
Related Work & State of the Art

“I suspect that machines to be programmed in our native tongues […] are as damned difficult to make as they would be to use.”

[Edsger W. Dijkstra]
Related Work & State of the Art

"Choose a row in the matrix."
"Put the average of the first four entries in that row into its last entry."

\[
\text{select}($A)
\]
\[
A30 := \text{avg}(A1:A4)
\]

[Ballard1979]

1960s
"Impossible"
"no need"
[Dijkstra1963]
[Dijkstra1964]
[Hill1972]

1979
Natural Language Computer
[NLCI – A Natural Language Command Interpreter]
[Ballard1979]
Related Work & State of the Art

“There is a bar with a bartender who makes drinks.”

```python
class bar:
    the_bartender = bartender()
class bartender:
    def make(drink): pass
```

[Liu2005]

1960s
- „Impossible“
- „no need“
  - [Dijkstra1963]
  - [Dijkstra1964]
  - [Hill1972]

1979
- Natural Language Computer
  - [Ballard1979]

2000s
- Code stubs from prose
  - [Liu2005]
- Studies on wording of layperson
  - [Pane2001, 2002]

NLCI – A Natural Language Command Interpreter
Related Work & State of the Art

1960s
"Impossible"
"no need"
[Dijkstra1963]
[Dijkstra1964]
[Hill1972]

1979
Natural Language Computer
[Ballard1979]

2000s
Code stubs from prose
[Liu2005]
Studies on wording of layperson
[Pane2001,2002]

2010s
IBM Watson
[Ferruci2010]
SmartSynth
[Le2013]
Apple‘s Siri
[Bellegarda2014]
Related Work & State of the Art

State of the Art

- Restricted wording: **Usability**?
- Domain-dependent: **Portability**?

1960s
- "Impossible"
- "no need"
- [Dijkstra1963]
- [Dijkstra1964]
- [Hill1972]

1979
- Natural Language Computer
- [Ballard1979]

2000s
- Code stubs from prose
- [Liu2005]
- Studies on wording of layperson
- [Pane2001,2002]

2010s
- IBM Watson
  - [Ferruci2010]
- SmartSynth
  - [Le2013]
- Apple’s Siri
  - [Bellegarda2014]

Programming in Natural Language
[?]
Contribution

- The NLCI architecture offers
  - automatic generation of (imperative) source code from English prose and
  - separation of language analyses and the domain, i.e., target API.

- The textual input
  - is unrestricted regarding the wording of descriptions,
  - may be out of order, and
  - can comprise control structures.

- Evaluation: two case studies
  - Home automation: openHAB
  - 3D animations: CMU Alice
Modular architecture to generate source code from written English prose
- Separation of language analyses and domain knowledge
- The API is part of the configuration of NLCI
- The API model can be generated (semi-)automatically and is enriched automatically (use meaningful identifiers, add synonyms, etc.)
Modular architecture to generate source code from written English prose

- Separation of language analyses and domain knowledge
- The API is part of the configuration of NLCI
- The API model can be generated (semi-)automatically and is enriched automatically (use meaningful identifiers, add synonyms, etc.)
Modular architecture to generate source code from written English prose

- Separation of language analyses and domain knowledge
- The API is part of the configuration of NLCI
- The API model can be generated (semi-)automatically and is enriched automatically (use meaningful identifiers, add synonyms, etc.)
Mapping textual entities to the API: Example API: NLCI ontology structure

- **Origin**
  - openhab.jar
  - alice.jar
  - slf4j.jar

- **Class**
  - Object
    - man
    - WhiteBunny
    - Rabbit
    - Broccoli
  - Component
    - arm
    - leftEar

- **Method**
  - Predefined Method
    - turn()
    - move()
    - toString()
  - User-defined Method
    - turnToFace(c: class)
    - lift(arm: Component)

- **Parameter**
  - dist
  - length
  - count

- **Data type**
  - int
  - double
  - String
Mapping textual entities to the API: Sentence analysis and API-Mapping

- Preprocessing: Dependency Parsing (Stanford CoreNLP)
- Special treatment: Active and passive voice, imperative, ellipses, adjectives, subordinated clauses, conjunctions, appositions, etc.

The bunny eats the broccoli and turns to Alice.

- Class
  - Rabbit
  - WhiteBunny
  - Broccoli
  - Alice

- Method
  - eat(Class)
  - turn()
  - turnToFace(Class)
The bunny eats the broccoli and turns to Alice.

- Candidates for "bunny eats broccoli"
  - WhiteBunny.eat(Broccoli), Rabbit.eat(Broccoli)
- Candidates for "bunny turns to Alice"
  - Rabbit.turnToFace(Alice), Rabbit.turn(), WhiteBunny.turn()
Input: English Prose

Output: Source Code

Architecture

- NLP-Pre-processing
- API-Mapping & Method Call Construction
- ... 
- Generation of Source Code
- Detection of Control Structures
- Correction of Timelines

Further processing:
- Correction of timelines [Landhaeusser2014]
- Detection of control structures [Landhaeusser2015]
- Generation of source code
Case study and evaluation: Overview

- Home automation software: openHAB
  - Imperative: „turn on the light in the kitchen“
  - 114 classes, 9 (distinct) methods
  - Pilot study

- 3D animations: CMU Alice
  - Prose (aktive and passive voice): „While the bunny jumps, Alice turns to the bunny and says, ‘Hey, Wait’“.
  - 914 classes, 393 (distinct) methods
## Evaluation: Method call construction
**CMU Alice & openHAB**

### Corpora

<table>
<thead>
<tr>
<th></th>
<th>Texts</th>
<th>Sentences</th>
<th>Words</th>
<th>API calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMU Alice</td>
<td>50</td>
<td>703</td>
<td>6764</td>
<td>570</td>
</tr>
<tr>
<td>openHAB</td>
<td>5</td>
<td>15</td>
<td>149</td>
<td>20</td>
</tr>
</tbody>
</table>

### Results

<table>
<thead>
<tr>
<th></th>
<th>Metrics [%]</th>
<th>0 Param.</th>
<th>1 Param.</th>
<th>2 Param.</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMU Alice</td>
<td>Precision</td>
<td>85,7</td>
<td>86,1</td>
<td>100,0</td>
<td>86,5</td>
</tr>
<tr>
<td></td>
<td>Recall</td>
<td>74,5</td>
<td>70,9</td>
<td>54,1</td>
<td>71,5</td>
</tr>
<tr>
<td></td>
<td>F₁</td>
<td>79,7</td>
<td>77,8</td>
<td>70,2</td>
<td>78,8</td>
</tr>
<tr>
<td>openHAB</td>
<td>Precision</td>
<td>73,7</td>
<td></td>
<td></td>
<td>73,7</td>
</tr>
<tr>
<td></td>
<td>Recall</td>
<td>70,0</td>
<td></td>
<td></td>
<td>70,0</td>
</tr>
<tr>
<td></td>
<td>F₁</td>
<td>71,8</td>
<td></td>
<td></td>
<td>71,8</td>
</tr>
</tbody>
</table>
Conclusion & Future Work

- The NLCI architecture
  - facilitates source code generation from English prose and
  - eases the accessibility of new domains, i.e., APIs.

- The evaluation shows, that
  - there is no need to restrict the language,
  - radically different APIs can be addressed,
  - timelines are recover- and control structures synthesizable.

- To improve NLCI, one might
  - diminish or compensate flaws of the NLP preprocessing,
  - gain more natural language understanding (e.g., coreference analysis) and
  - use additional API features (e.g., pre- and postconditions).
Thank You!

More technical details may be found in our video about our NLCI lab course: https://youtu.be/Z_vt1-imBUE