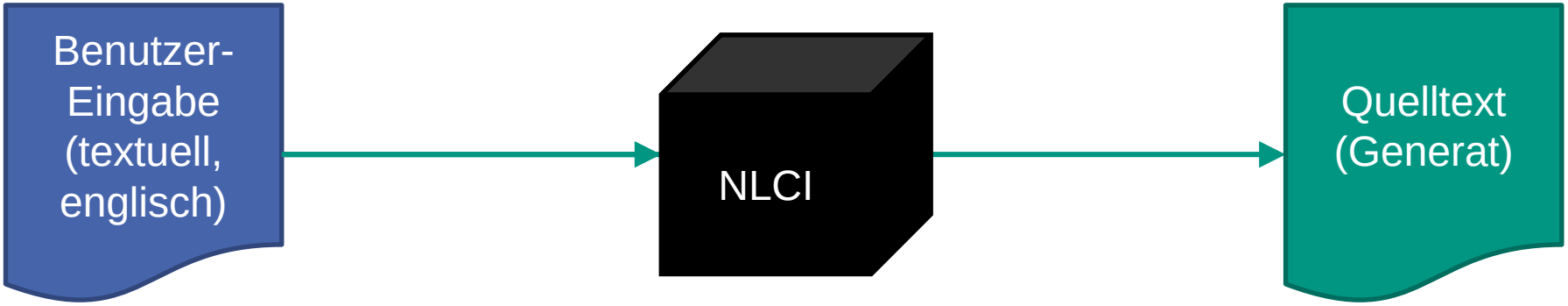
Karlsruher Institut für Technologie

Eine Architektur für Programmsynthese aus natürlicher Sprache

Mathias Landhäußer

IPD Tichy, Fakultät für Informatik

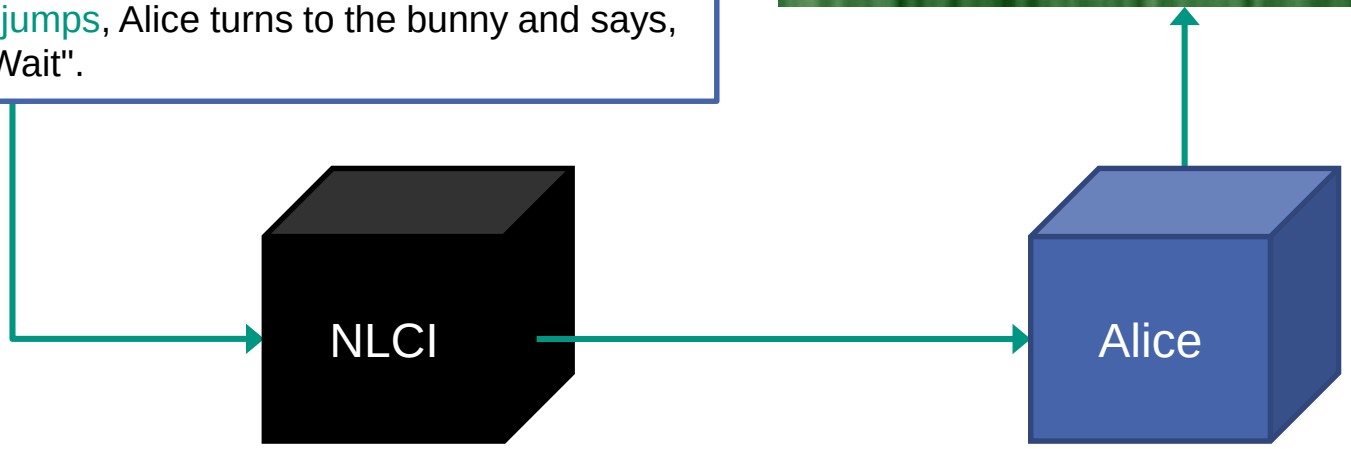
Ziel dieser Arbeit



Natural **L**anguage **C**ommand **I**nterpreter

Einführendes Beispiel
Erzeugen von Animationen mit CMU Alice

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, "Ahhhhhhh!". 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".



Entwicklung und verwandte Arbeiten

Problem:
Wenn es um Steuerung oder Programmierung geht, ist die Textanalyse oft eng mit der Domäne verwoben und weder austauschbar noch wiederverwendbar.

1960er
„Geht nicht"
„Will man nicht"
[Dijkstra1963]
[Dijkstra1964]
[Hill1972]

1979
Natural Language Computer
[Ballard1979]

2000er
- Programm-rümpfe aus Texten
[Liu2005]
- Studien zur Ausdrucksweise von Laien
[Pane2001,2002]
- NaturalJava
[Price2000]

2010er
- Googles Übersetzer
- IBM Watson
[Ferruci2010]
- SmartSynth
[Le2013]
- Apples Siri
[Bellegarda2014]
- Hello Barbie

Programmieren in natürlicher Sprache
[?]

Beitrag dieser Arbeit

- Die vorgestellte Architektur ermöglicht
  - die **automatische Übersetzung** von natürlichsprachlich ausgedrückten, imperativen Programmen in Quelltext und
  - **entkoppelt die Textanalyse** von der anzusprechenden API.

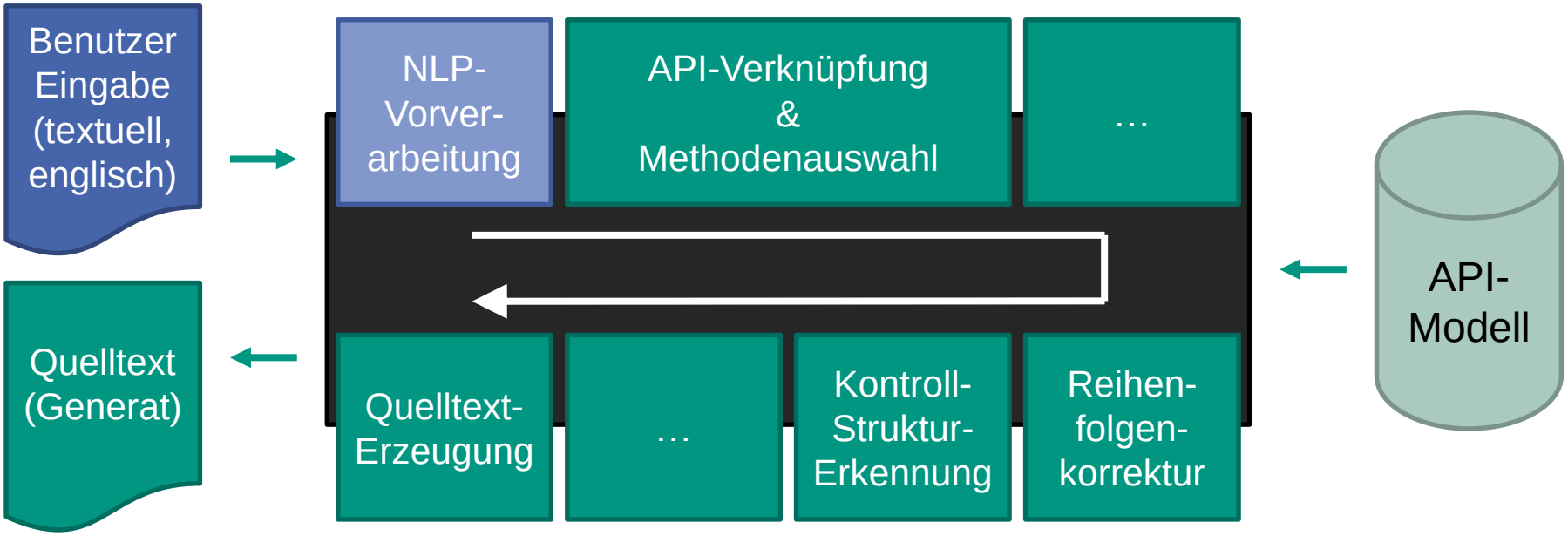
- Die Eingabetexte
  - müssen **nicht sprachlich eingeschränkt** werden,
  - können **chronologisch korrekt umgesetzt** werden und
  - können natürlichsprachlich umschriebene **Kontrollstrukturen** enthalten.

- Evaluierung des Prototyps in zwei Fallstudien
  - Heimautomation mit openHAB
  - Erzeugen von 3D-Animationen mit Alice

Überblick über die Architektur

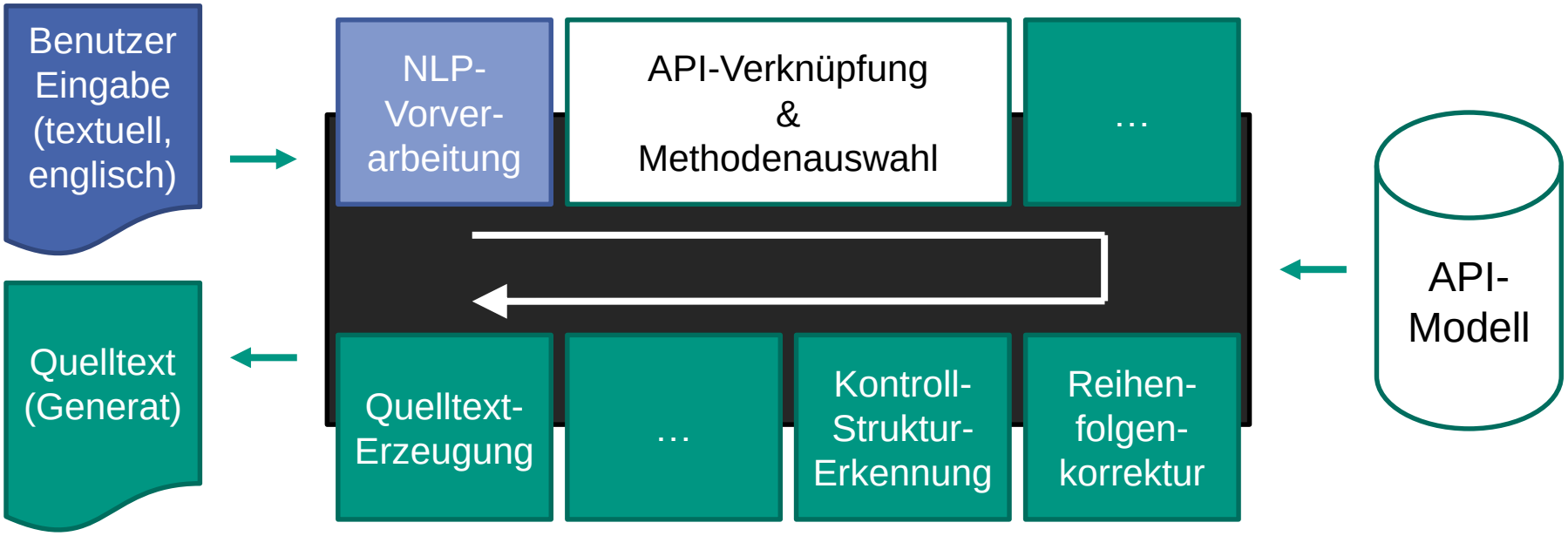
- Modulare Architektur zur Übersetzung von Text nach Quelltext
  - Entkopplung von Textanalyse und Wissen über die API
  - API ist Teil der Konfiguration von NLCI
  - API-Modell kann automatisch erzeugt werden und wird automatisch aufbereitet (Bezeichner, Synonyme, etc.)

Überblick über die Architektur



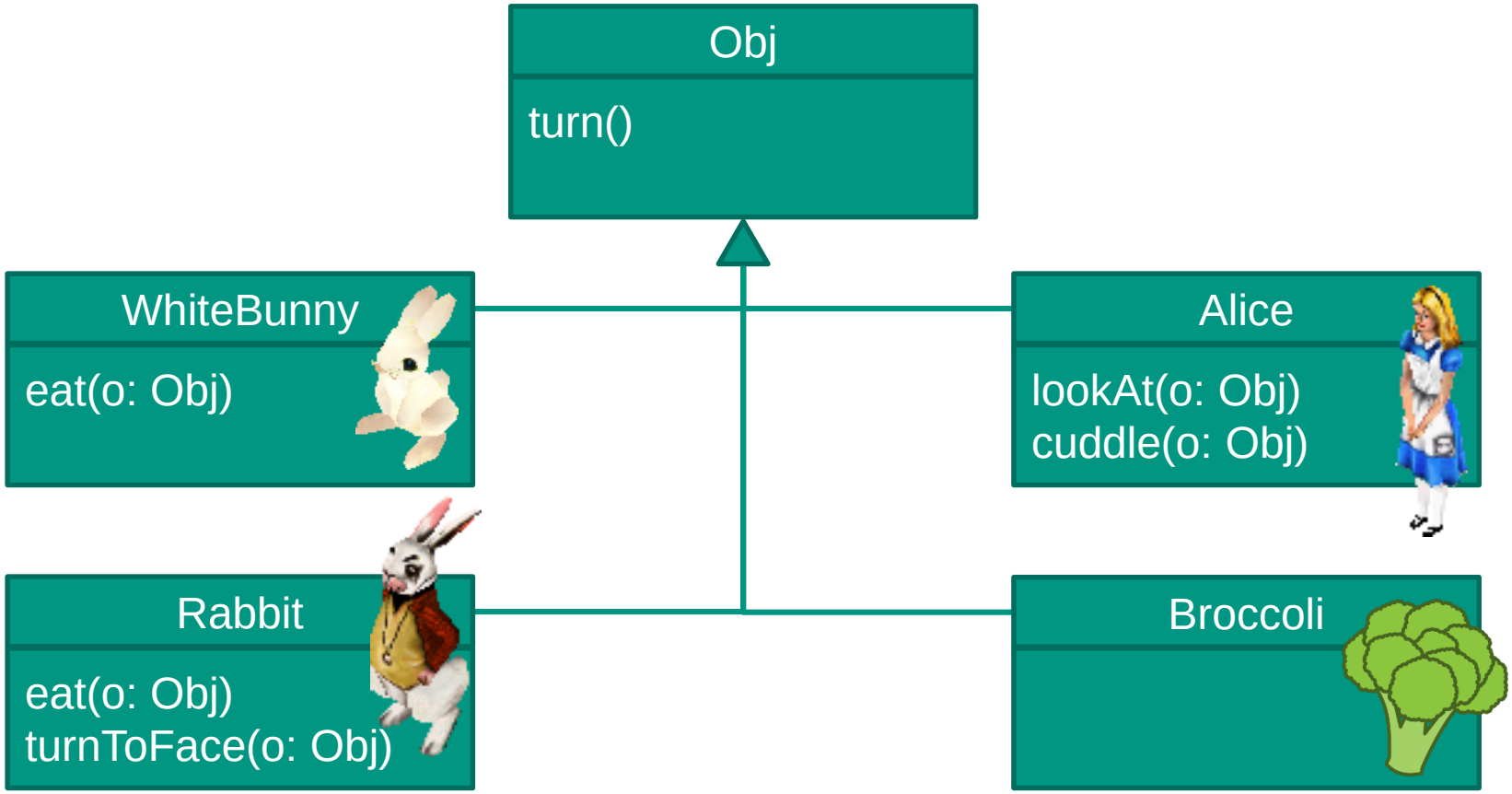
- Modulare Architektur zur Übersetzung von Text nach Quelltext
  - Entkopplung von Textanalyse und Wissen über die API
  - API ist Teil der Konfiguration von NLCI
  - API-Modell kann automatisch erzeugt werden und wird automatisch aufbereitet (Bezeichner, Synonyme, etc.)

Überblick über die Architektur

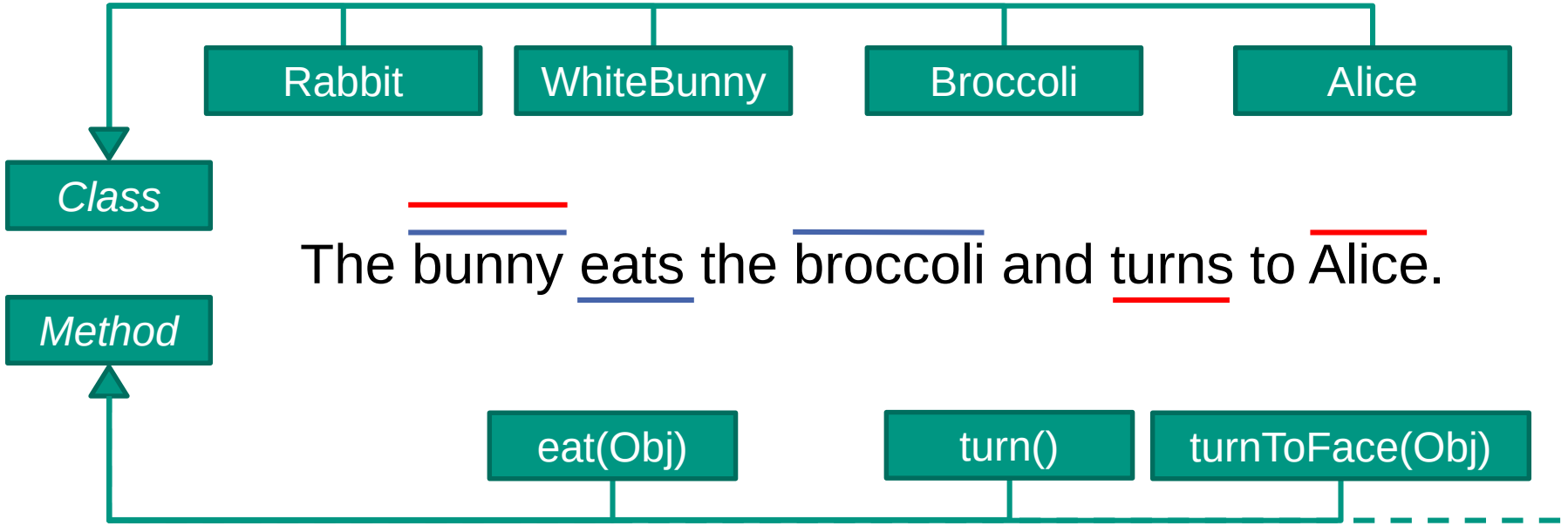


- Modulare Architektur zur Übersetzung von Text nach Quelltext
  - Entkopplung von Textanalyse und Wissen über die API
  - API ist Teil der Konfiguration von NLCI
  - API-Modell kann automatisch erzeugt werden und wird automatisch aufbereitet (Bezeichner, Synonyme, etc.)

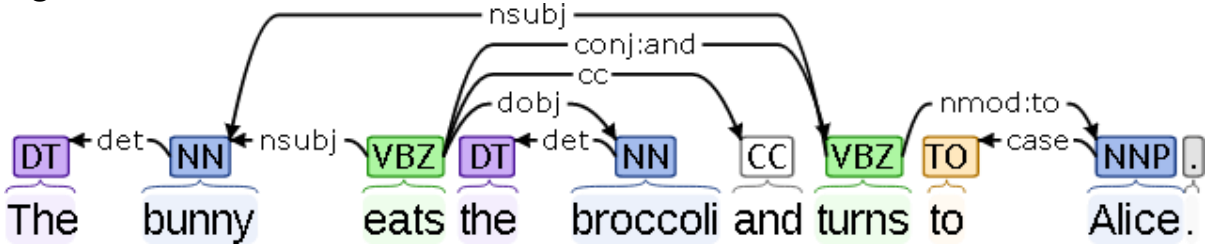
Verknüpfung der Textelemente mit der API: Beispiel-API: Auszug aus Alice



Verknüpfung der Textelemente mit der API: Satzanalyse und API-Suche

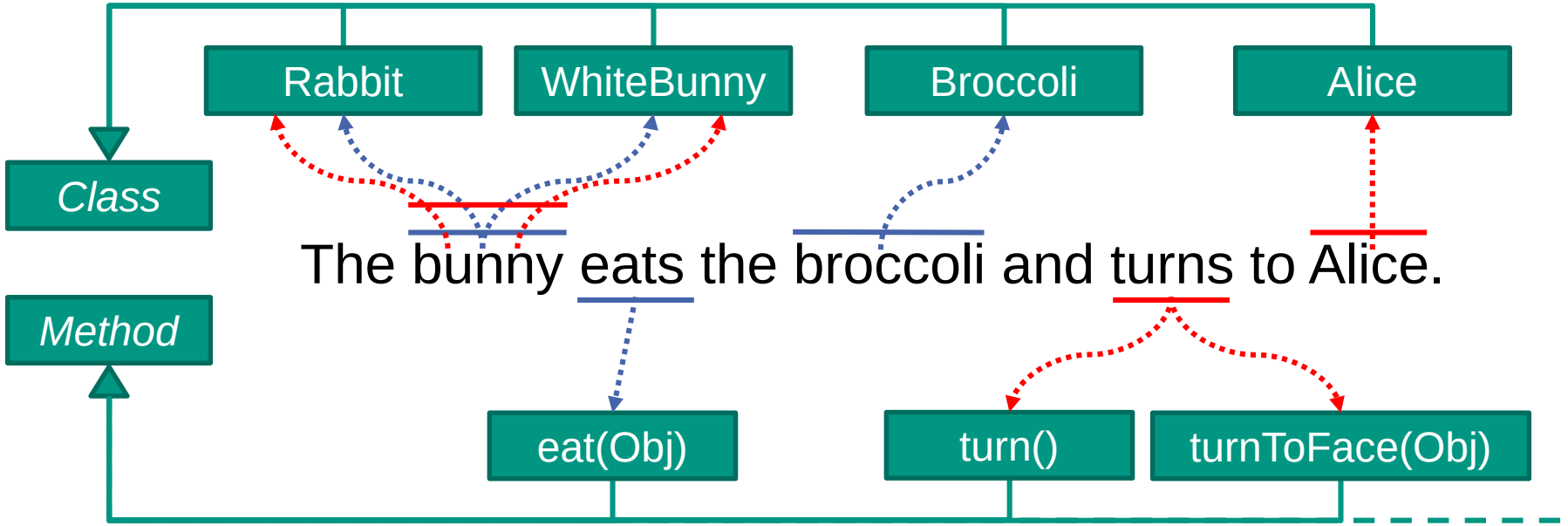


- Vorverarbeitung mit Stanford CoreNLP



- Textanalyse betrachtet Aktiv und Passiv, Adjektive, Nebensätze, Konjunktionen, Appositionen etc.

Verknüpfung der Textelemente mit der API: Satzanalyse und API-Suche

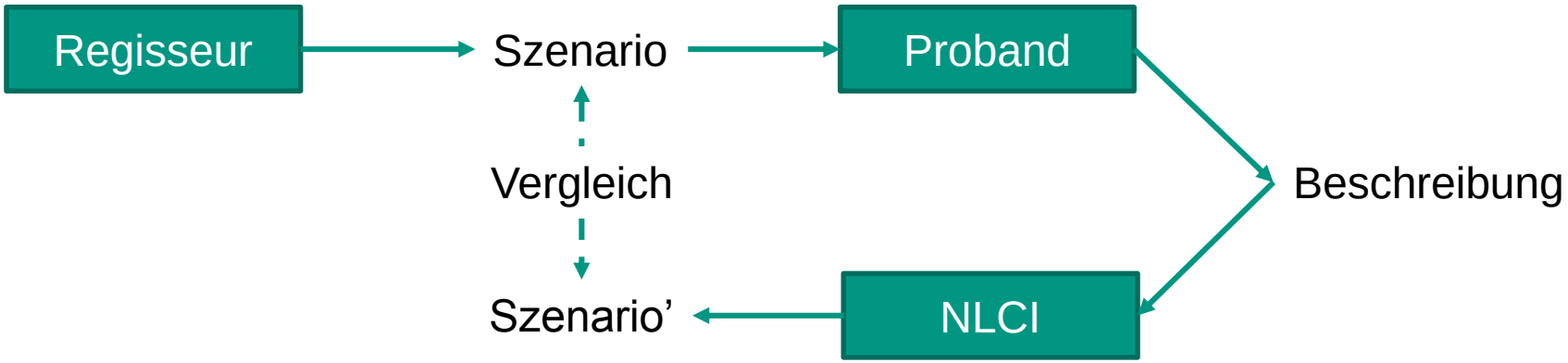


- Kandidaten für „bunny eats broccoli“
  → ~~`whiteBunny.eat(Broccoli)`~~, `Rabbit.eat(Broccoli)`
- Kandidaten für „bunny turns to Alice“
  → `Rabbit.turnToFace(Alice)`, ~~`Rabbit.turn()`~~, ~~`whiteBunny.turn()`~~

Fallstudien und Evaluation: Überblick und Vorgehen

- Heimautomation mit openHAB
  - Texte eher im Imperativ „schalte das Licht in der Küche ein“…
  - 114 Klassen mit insgesamt 9 Methoden
  - Kleinere Machbarkeitsstudie

- 3D-Animationen mit Alice
  - Texte sind beschreibende Prosa
  - 914 Klassen mit insgesamt 393 Methoden
  - Umfangreiche Auswertung von 103 Eingabetexten



Fallstudien und Evaluation: Ergebnisse der API-Suche

- API-Verknüpfung ist der Grundstein der Analysen

- Wie oft ist die richtige Kombination aus Akteur, Aktion und Objekt in den Suchergebnissen enthalten?

Maß [%]	TOP-1	TOP-2	TOP-3	TOP-5	TOP-10
Präzision	78,1	39,2	26,1	15,7	7,9
Ausbeute	67,0	68,1	68,2	69,5	70,0

- Kann NLCI die gewünschten Methodenaufrufe (inkl. aller Argumente) generieren?

Maß [%]	0 Parameter	1 Parameter	2 Parameter	Gesamt
Präzision	85,7	86,1	100,0	86,5
Ausbeute	74,5	70,9	54,1	71,5
F_1-Maß	79,7	77,8	70,2	78,8

Fallstudien und Evaluation: Auswirkung von Fehlern der NLP-Vorverarbeitung

- Die Güte der NLP-Vorverarbeitung beeinflusst das Ergebnis von NLCI

- 3 Fragestellungen
  - Wie gut ist die Analyse *mit* Fehlern in der Vorverarbeitung?
  - Kann man diese Fehler ausgleichen?
  - Wie gut ist die Analyse *ohne* Fehler in der Vorverarbeitung?

- Beispiel: Synthese von Kontrollstrukturen

Evaluation	gesamt	korrekt	falsch	fehlend
Mit NLP-Fehlern Standardkonfig.	795	649 82%	77 10%	140 18%
Mit NLP-Fehlern Angepasste Konfig.	795	705 89%	78 10%	79 10%
Ohne NLP-Fehler Standardkonfig.	529	513 97%	5 1%	16 3%

Zusammenfassung & Ausblick

- Die vorgestellte Architektur
  - erlaubt Programmsynthese aus englischem Text und
  - erleichtert die Erschließung von neuen Domänen (oder APIs).

- Die Evaluation des Prototyps zeigt, dass man
  - keine Einschränkung der Eingabesprache machen muss,
  - keine chronologisch korrekte Beschreibung fordern muss und
  - Kontrollstrukturen aus der Eingabe synthetisieren kann.

- Möchte man NLCI weiterentwickeln, dann sollte man
  - Fehler der NLP-Vorverarbeitung verringern oder ausgleichen,
  - mehr Sprachverständnis erreichen (z. B. Korreferenzen) und
  - weitere API-Eigenschaften nutzen (z. B. Vor- und Nachbedingungen).