Programmieren in natürlicher Sprache: Automatische Code-Generierung

Dokumentenart:	Exposé für eine Bachelorarbeit
Autor:	Artem Titarenko
Matrikel-Nr.:	1622831
Studiengang:	Informatik Bachelor
Betreuer:	Mathias Landhäußer

1 Motivation

Das Ziel des Projektes „Programmieren in natürlicher Sprache“ ist es aus englischen Texten Animationen zu generieren. Dabei wird das *Alice-Animations-framework* [Car] verwendet, welches nicht nur die gängigen Eigenschaften einer Objektorientierten Programmiersprache bietet, sondern auch Modelle die eine Abbildung der realen Umgebung darstellen.

Als Informatiker versucht man ja oft beim Programmieren Zeit und Arbeit zu sparen. Sei es mit Hilfe von Shortcuts in Programmen oder out-of-the-box Funktionalität bei Frameworks. Eine weitere Möglichkeit ist automatische Code-Generierung wie man sie beispielweise aus Tools für UML-Diagramme kennt, die einem doppelten Arbeitsaufwand erspart. Und eben dieser Arbeitsschritt, automatische Code-Generierung, fehlt noch im Projekt.

2 Programmieren in natürlicher Sprache

Der englische Text, welcher eine Animation in natürlicher Sprache beschreibt, durchläuft mehrere Analyse-Schritte, wobei jedes Mal neue Annotationen an den Text angefügt oder bestehende erweitert werden. Für die Initialisierung wird einmalig eine Ontologie aus der Alice-API gebaut [Pet12] und mit WordNet Assoziationen angereichert [Wei12]. Dadurch erhält man einen Zugang zu allen Objekten und Methoden des Alice-Frameworks, sowie deren Synonyme. In den weiteren Arbeitsschritten, werden mit Hilfe des Stanford Core NLP Tools und der Ontologie Annotationen zur Erkennung von Objekten und Methoden[Wei14], Parametern[Sek14], Kontrollstrukturen[Koh13, Hug14] und zeitlicher Sortierung der Sätze[Hey13] erstellt beziehungsweise erweitert. Eine weitere Studienarbeit hatte sich auch schon mit der Positionierung der Objekte in der Szene[Kob13] beschäftigt.

3 Automatische Code-Erzeugung

Der letzte Verarbeitungsschritt, der mit der hier vorgeschlagenen Studienarbeit abgedeckt wird, soll ein läuffähiges Alice-Skript aus dem annotierten Text generieren. Dazu wird ein Golden-Gate-Plugin entworfen mit einer Schnittstelle zum Szenerie-Analyzer und einem Alice-World-Generator, der ebenfalls in dieser Studienarbeit entwickelt werden soll.

Der World-Generator soll ein Template einer leeren Alice-Welt-Datei nutzen und mit den Informationen aus der Szenerie die Objekte einfügen. Außerdem soll der World-Generator eine Schnittstelle bieten um die Kontrollstrukturen und Methodenaufrufe korrekt einzufügen. Der Animationsablauf wird auf sehr triviale Weise in der Alice-Welt gespeichert. Und zwar werden Kontrollstrukturen in Form eines Ordners mit einer XML-Datei im Inneren angelegt. In der XML-Datei wird der Typ, Inhalt, Reihenfolge und weitere typbedingte Eigenschaften spezifiziert. Methodenaufrufe folgen dem gleichen Schema. Für jeden Methodenaufruf wird ebenfalls ein Ordner mit einer XML-Datei zur Spezifikation erstellt.

Der Code-Generator soll einen geordneten Baum, also ein Baum bei dem für jeden Knoten eine feste Reihenfolge der Kindknoten besteht, erstellen. Dabei bilden die inneren Knoten Kontrollstrukturen und die Blattknoten Methodenaufrufe ab. Die Knoten müssen selbstverständlich auch die benötigten Parameter enthalten, beispielsweise die Bedingung für eine While-Schleife. Diese Datenstruktur stellt einen Pseudocode dar und wird schlussendlich dem World-Generator übergeben, deshalb ist es wichtig sie in einer seperaten API zu beschreiben. Der Aufbau dieser Datenstruktur erfolgt schrittweise. Im Ersten Schritt werden die Annotationen zu den Kontrollstrukturen, im Zweiten die Methodenaufrufe und im letzten die zeitliche Sortierung ausgewertet.

Zusätzlich muss eine automatische Fehlererkennung und Korrektur implementiert werden, die gewährleistet, dass eine lauffähige Alice-Welt generiert wird. Dies ist zum Einen möglich, da jede Annotation eine Wahrscheinlichkeit zu der eigenen Korrektheit enthält. Und zum Anderen, da es für jedes Wort meist mehrere Annotationen gibt, beispielsweise mehrere Möglichkeiten für die Zugehörigkeit eines Verbes zu Methoden.

4 Evaluation

Abschließend soll das Plugin auf seine Funktionalität überprüft werden. Hierzu wird der Alice-Korpus[Ham12] verwendet, der zu zahlreichen Animationen eine Musterlösung und Beschreibungen von verschiedenen Probanden in textueller Form bietet. Die Musterlösung beschreibt den Quellcode der Animation, deshalb sollte im Idealfall die aus der Musterlösung generierte Animation den selben Ablauf wie die der Musterlösung zugrunde liegenden haben. Dazu soll der Quellcode von beiden Animationen verglichen werden.

Bei den von Probanden erstellten Beschreibungen soll zur Evaluation manuell die Anzahl der korrekt verarbeiteten Annotationen ausgewertet werden. Eine qualitative Auswertung der entstandenen Animation ist nicht aussagekräftig, da der Code-Generator auf den Analyzern aufbaut und die Qualität der Animation von denen der Analyzer stark abhängt. Im Gegensatz dazu soll jedoch eine qualitative Auswertung der Fehlererkennung und Korrektur erfolgen.

Literatur

[Car] Carnegie Mellon University: *Alice*. `http://www.alice.org`

[Ham12] Hampel, Sina: *Programmieren in natürlicher Sprache: Aufbau des Alice-Korpus*, Karlsruher Institut für Technologie (KIT) – IPD Tichy, Bachelor's Thesis, November 2012. `https://svn.ipd.kit.edu/trac/AliceNLP/wiki/Theses/hampel_ba`

[Hey13] Hey, Tobias: *Programmieren in natürlicher Sprache: Zeitliches Sortieren*, Karlsruher Institut für Technologie (KIT) – IPD Tichy, Bachelor's Thesis, Dezember 2013. `https://svn.ipd.kit.edu/trac/AliceNLP/wiki/Theses/hey_ba`

[Hug14] Hug, Ronny: *Programmieren in natürlicher Sprache: Alice-Kontrollstrukturen*, Karlsruher Institut für Technologie (KIT) – IPD Tichy, Bachelor's Thesis, März 2014. `https://svn.ipd.kit.edu/trac/AliceNLP/wiki/Theses/hug_ba`

[Kob13] Kober, Stefan: *Programmieren in natürlicher Sprache: Erzeugen von Alice-Szenerien aus Text*, Karlsruher Institut für Technologie (KIT) – IPD Tichy, Diplomarbeit, Dezember 2013. `https://svn.ipd.kit.edu/trac/AliceNLP/wiki/Theses/kober_da`

[Koh13] Kohlmann, Andrea: *Programmieren in natürlicher Sprache: Alice-Kontrollstrukturen aus natürlicher Sprache*, Karlsruher Institut für Technologie (KIT) – IPD Tichy, Bachelor's Thesis, September 2013. `https://svn.ipd.kit.edu/trac/AliceNLP/wiki/Theses/kohlmann_ba`

[Pet12] Peters, Oleg: *Programmieren in natürlicher Sprache: Extraktion einer Alice-API-Ontologie*, Karlsruher Institut für Technologie (KIT) – IPD Tichy, Bachelor's Thesis, November 2012. https://svn.ipd.kit.edu/trac/AliceNLP/wiki/Theses/peters_ba

[Sek14] Seker, Ali K.: *Programmieren in natürlicher Sprache: Erfassung von Methodenargumenten aus natürlicher Sprache*, Karlsruher Institut für Technologie (KIT) – IPD Tichy, Diplomarbeit, Mai 2014. https://svn.ipd.kit.edu/trac/AliceNLP/wiki/Theses/seker_da

[Wei12] Weigelt, Sebastian: *Programmieren in natürlicher Sprache: Aufbau einer Alice-Ontologie – Korpus-Ontologie-Assoziation*, Karlsruher Institut für Technologie (KIT) – IPD Tichy, Studienarbeit, November 2012. https://svn.ipd.kit.edu/trac/AliceNLP/wiki/Theses/weigelt_sa

[Wei14] Weigelt, Sebastian: *Programmieren in natürlicher Sprache: Erkennung und semantische Assoziation von Entitäten in natürlichsprachlichen Texten*, Karlsruher Institut für Technologie (KIT) – IPD Tichy, Diplomarbeit, Februar 2014. https://svn.ipd.kit.edu/trac/AliceNLP/wiki/Theses/weigelt_da