Typprüfung und Namensanalyse für textuelle Anforderungsbeschreibungen*

Sebastian Wendt und Wolf Zimmermann

Martin-Luther-Universität Halle-Wittenberg
sebastian.wendt@informatik.uni-halle.de
wolf.zimmermann@informatik.uni-halle.de

Zusammenfassung. Anforderungsbeschreibungen für eine technische Domäne sollen in einer Sprache verfasst werden, die Mehrdeutigkeiten eliminiert, und die automatisch durch einen Übersetzer überprüft werden kann. Dieser Übersetzer soll die konsistente Verwendung von Namen mittels Namensanalyse und die korrekte Kombination von in einem Glossar definierten Begriffen mittels Typanalyse überprüfen.

1 Einleitung

Natürliche Sprache ist in vielen Projekten der Ausgangspunkt zur Modellierung von Anforderungen. Um ein Anforderungsmanagement umzusetzen, das alle Phasen eines Projekts umfasst, müssen diese natürlichsprachlichen Anforderungen durch die Werkzeuge mit einbezogen werden. Rückverfolgbarkeit von der Implementierung zurück, nicht nur bis zur ersten formalen Beschreibung, sondern bis zur textuellen Beschreibung, kann dann auch Anforderungen von Stakeholdern erfassen, die nicht mit formalen Modellbeschreibungen umgehen können. Wenn natürliche Sprache von automatisierten Werkzeugen behandelt wird, dann gewinnt der Prozess zudem an Geschwindigkeit, Zuverlässigkeit und Reproduzierbarkeit.

Für die Verarbeitung von natürlichsprachlichen Texten wird oft auf heuristische und probabilistische Techniken zurückgegriffen. Probabilistische Techniken versprechen, die Analyse eines Textes dann zuverlässig durchzuführen, wenn man sie auf einem hinreichend großen Datensatz (Korpus) trainiert. Doch wie sich Stakeholder nicht über Begrifflichkeiten einigen können, so gibt es auch Abweichungen zwischen den Annotationen in den Korpora und den Autoren der Anforderungsdokumente einer technischen Domäne. Je umfangreicher der Korpus ist, desto wahrscheinlicher ist es, auf Interpretationen zu treffen, die für die Domäne nicht relevant sind und unnötige Mehrdeutigkeiten einführen, die Fehler verschleiern. Eine hoher Anteil an falsch-positiven Interpretationen (Falsches, das als richtig erkannt wird) ist die Folge.

Wir beschreiben in Abschnitt 3 die konkrete Syntax einer kontrollierten Sprache, welche die Syntax natürlicher Sprache (Deutsch) imitiert. Die Syntax basiert

* Diese Arbeit wurde im Rahmen des vom Bundesministerium für Bildung und Forschung (BMBF) geförderten Forschungsprojekts „ELSY“ (Nr. 16M3202D) erstellt.

auf der Idee der Schablonen, und setzt sie als Grammatik um, statt es bei einer Menge von *best-practice* Formulierungsrichtlinien zu belassen. Aus dieser Grammatik erzeugen wir einen Parser, der Teil eines Übersetzers ist.

In den Abschnitten 4 und 5 beschreiben wir Namensanalyse und Typprüfung, die dazu dienen, die Konsistenz innerhalb der Anforderungsbeschreibung abzusichern.

Die Ziele dieser Arbeit sind:

- Eindeutigkeit der textuellen Formulierung herzustellen und Mehrdeutigkeit natürlicher Sprache zu eliminieren,
- Überprüfbarkeit der Beschreibung herzustellen, vollständig und automatisch, statt heuristisch oder per Hand,
- Konsistenz der Beschreibungen sicherzustellen, durch Typanalyse statt *best-practice* Regeln
- Lesbarkeit der Texte zu erhalten, statt eine formale Syntax zu benutzen, die Stakeholder ohne Einarbeitung ausschließt.

2 Verwandte Arbeiten

Eine kontrollierte Sprache, wie wir sie in dieser Arbeit vorstellen, ist im Allgemeinen eine natürliche Sprache, der gewisse syntaktische und semantische Einschränkungen auferlegt sind. Dadurch wird eine kontrollierte Sprache noch nicht zu einer formalen Sprache. So sind *Schablonen* [Hull u. a., 2005; Rupp, 2014] Vorlagen für den organisierten Satzaufbau, um Anforderungstexte zu schreiben, die für Dritte leichter lesbar sind.

Leichtes Deutsch wird von öffentlichen Einrichtungen auf deren Webseiten eingesetzt, um Inhalte barrierefrei für Personen mit Verständnisschwierigkeiten zu transportieren. *Einfache Sprache* verfolgt ebenfalls dieses Ziel und ist für Nicht-Deutsch-Muttersprachler entwickelt worden.

Rechtschreibprüfungs-Programme als Bestandteil von Textverarbeitungs-Programmen vergleichen im einfachsten Fall Wörter mit einem Wörterbuch. Es gibt weiter auch solche, dich auch die Grammatik überprüfen, z.B. auf Basis von *link grammars* von Lachowicz u. a. [2015]. Rechtschreibprüfungsprogramme dürfen nicht zu viele Fehlermeldungen produzieren, da Nutzer das Programm sonst schnell als zu empfindlich und untauglich abtun. Daher zeigen sie weniger potentielle Fehler an (mehr falsch-Positive, weniger Negative).

Die Prädikat-Argument-Struktur der Präpositionalphrasen in konkreter Syntax ist der Prädikatenlogik bzw. Prolog nachempfunden. Kuhn u. Bergel [2014] zeigten in einem Experiment an Studentengruppen, dass Gruppen, die Aussagen in natürlicher Sprache verarbeiten sollten, genauer und schneller arbeiteten, als Gruppen, die dieselben Aussagen in Form von Prolog-Statements erhielten. Die Prädikat-Argument-Struktur ist auch Grundlage der konstruierten natürlichen Sprache Lojban Cowan [2000].

Circe [Ambriola u. Gervasi, 2006] ist ein Übersetzer, der Anforderungsbeschreibungen in natürlicher Sprache (Italienisch) auf heuristische Art analysiert und in der Lage ist, Transformationen nach UML, ER-Diagramm oder

ASM durchzuführen. Die Grammatik des Eingabetextes ist nicht beschränkt; der Übersetzer unterzieht den Text einem *part-of-speech-tagging*, und führt dann iterativ eine Bewertung einer Menge von Regeln auf den klassifizierten Token durch, woraus sich eine Punktwertung für jede mögliche Regelanwendung ergibt, wendet die am besten bewertete Regel an, und wiederholt den Vorgang bis ein Terminierungskriterium erreicht ist. Die Regeln beziehen sich auf Attribute (*tags*) der Token, die aus der syntaktischen Analyse oder aus Regelanwendungen stammen können. Beispiele zeigen Regeln, die ein Typsystem beschreiben. Die Regeln können durch die Benutzer verändert werden und sind in einer speziellen formalen Syntax verfasst.

[Farfeleder, 2012] beschreibt in seiner Dissertation ein Verfahren zur Transformation von natürlicher Sprache in Schablonen auf Englisch (*boilerplates*) für die Domäne der eingebetteten Systeme. Ausgangspunkt sind vorliegende Texte, die durch halbautomatische, überwachte Transformation in eine Form gebracht werden soll, die besser analysierbar ist, d.h. deren Analyse weniger korrigierende Eingriffe benötigt. Die Schablonen werden benutzt um daraus Informationen über die Sicherheit des modellierten Systems zu gewinnen (*safety analysis*).

3 Kontrollierte natürliche Sprache

In dieser Arbeit entwickeln wir eine Sprache, die das Aussehen natürlicher Sprache besitzt, und wählen die syntaktischen Einschränkungen so, dass wir eine formale Sprache erhalten. Dadurch bleibt die Lesbarkeit für Dritte ohne Vorkenntnisse erhalten. Auf dieser Sprache führen wir in späteren Abschnitten Konsistenzprüfungen mittels Namensanalyse und Typprüfung durch und definieren zu diesem Zweck ein domänenspezifisches Glossar. Die größte Schwierigkeit bei Projekten, die natürliche Sprache verarbeiten (*natural language processing*, NLP) besteht in dem Anspruch der Nutzer, das Chaos an überladenen Bedeutungen und Sonderfällen, das wir als natürliche Sprache bezeichnen, mit gleicher oder besserer Präzision und Vorhersagekraft verarbeiten zu können. Diese Arbeit erhebt nicht den Anspruch, natürliche Sprache vollständig abzudecken. Der Anteil an natürlichsprachlichen Ausdrücken, der von unseren Konsistenzprüfungen abgelehnt wird (falsch-Negative) ist daher groß im Vergleich zu NLP-Parsern. Im Gegenzug streben wir an, den Anteil an Ausdrücken, der nicht korrekt ist, und der trotzdem akzeptiert wird (falsch-Positive) zu eliminieren . Dies gelingt uns durch die Einschränkungen, die wir treffen[1].

Die Merkmale unserer kontrollierten Sprache, auf die wir anschließend im Einzelnen eingehen werden, sind:

- Prädikat-Argument-Struktur wie in Prädikatenlogik (Abschnitt 3.1)
- Unterscheidung Definition und Benutzung durch Artikel (*determiner*) (Abschnitt 4)
- Bedingte Anforderungen (Wenn-Dann) (Abschnitt 3.2)
- Zusammenfassen der Zeitformen (Abschnitt 3.3)

[1] zu beachten sind einige Spezialfälle; siehe Abschnitt 3.4

Morphologie verwerfen Eine Grundannahme der konkreten Syntax ist, dass wir, wo möglich, auf morphologische Informationen (Beugung, Deklination) verzichten wollen. Diese Informationen sind inhärent unzuverlässig, weil durch die Entwicklung der Sprache viele Fälle entstanden sind, in denen syntaktisch identische Terme für semantisch unterschiedliche Konzepte verwendet werden müssen. Dies gilt für Artikel (der Mann, Nominativ; der Frau, Genitiv), Substantive (Anzeige, wie Bildschirm; Anzeige, wie Gerundium zu *anzeigen*) und Verben (schnellen, wie *hervorschnellen*; schnellen wie Partizip 2 Plural von *schnell*). Morphologische Information erlaubt im Allgemeinen keine eindeutige Klassifizierung. Wir verwerfen daher Informationen über Singular oder Plural, grammatisches Geschlecht und Deklination von Substantiven, Adjektiven und Artikeln (Fälle). Die Konjugation von Verben klassifizieren wir in 2 Klassen entsprechend der Zeitform und verwerfen die Information über Zahl und Person.

3.1 Prädikat-Argument-Struktur

Unbedingte Anforderungssätze bilden den Grundbaustein für Schablonen in der konkreten Grammatik. Die Struktur der Anforderungssätze, die wir in unserer kontrollierten Sprache erlauben, ähnelt der von Prädikaten in Prädikatenlogik. Für Benutzer ohne Kenntnis von Prädikatenlogik ist dies jedoch nicht zu erkennen und nicht erforderlich.

Ein Anforderungssatz in kontrollierter Sprache könnte wie im folgenden Beispiel lauten:

> *Der Benutzer „Anlagenfahrer“ soll das System „Pumpe“ über das Bedienterminal innerhalb 1 Minute hochfahren können.*

Die konkrete Grammatik dieses Satzes besteht aus einer Verbalphrase (VP) mit mehreren Präpositionalphrasen (PP):

```
VP ::= ( PP | Modalverb )* Prädikat Modalverb? .
PP ::= Präposition? Artikel? Prädikat Bezeichner? .
```

Präposition, Artikel und Bezeichner sind optional. Die Bestandteile des Beispielsatzes lassen sich wie in Tabelle 1 aufteilen.

	Präposition	Artikel	Prädikat	Bezeichner	Modalverb
PP		Der	Benutzer	Anlagenfahrer	
Modal					soll
PP		das	System	Pumpe	
PP	über	das	Bedienterminal		
PP	innerhalb	1	Minute		
Prädikat			hochfahren		
Modal					können

Tabelle 1. Beispiel Prädikat-Argument-Struktur, zeilenweise zu lesen

In der Praxis sind Sätze meist kürzer und enthalten weniger Präpositionen. Im Anhang befindet sich ein Ausschnitt aus einem technischen Handbuch und seine Übersetzung in unsere Sprache.

Präpositionen Präpositionen sind eine endliche Mengen von Wörtern (*closed set*). Wir setzen sie ein als Beschriftung (*label*), um die Argumente eines Prädikats einer semantischen Rolle, d.h. einem der Parameter des Prädikats zuzuordnen. Dieses sogenannte *semantic role labeling* ist ein Teilproblem von NLP und wird i.d.R. heuristisch bzw. probabilistisch gelöst (Palmer u. a. [2010]). Wir tragen der Beweglichkeit in der natürlichen Sprache dadurch Rechnung, dass wir das freie Verschieben von Argumenten mit unterschiedlichen Beschriftungen erlauben, fordern aber, dass Argumente mit der gleichen Beschriftung der Reihenfolge folgen, wie sie für das jeweilige Prädikat im Glossar definiert wurde. Diese Regelung betrifft am häufigsten die *leere Beschriftung* (ε), die für Argumente ohne Präposition angenommen wird.

Beispiel[2]:

(1) *Der Benutzer soll die Pumpe mit einem Fördermedium füllen.*
(2) *Der Benutzer soll mit einem Fördermedium die Pumpe füllen.*
(3) *Die Pumpe mit einem Fördermedium soll der Benutzer füllen.*

In Prädikatschreibweise:

(1) füllen(Benutzer,Pumpe,~mit:Fördermedium)
(2) füllen(Benutzer,~mit:Fördermedium,Pumpe)
(3) füllen(Pumpe,~mit:Fördermedium,Benutzer)

Die Sätze (1) und (2) erzeugen eine äquivalente Prädikatstruktur, da die Reihenfolge von *Benutzer* und *Pumpe* erhalten bleibt. Bei Satz (3) ist dies nicht der Fall.

Präpositionen sind gelegentlich Kontraktionen unterworfen, z.B. wird *von+dem* zu *vom*. Wir lösen diese Kontraktionen vor der Typprüfung auf, so dass Verbdefinitionen nicht auf kontrahierte Varianten Rücksicht nehmen müssen.

Artikel Artikel sind eine endliche Menge von Wörtern. Wir setzen sie zusammen mit Präpositionen als syntaktischen Trenner für die Argumente eines Prädikats in einer Liste von Präpositionalphrasen (PP) ein. Wir verwerfen die Information über den grammatischen Fall (Kasus) aus dem Artikel, da diese nicht zuverlässig ist. Zulässige Artikel bzw. sogenannte *determiner* sind:

- unbestimmte Artikel (ein, eine)
- bestimmte Artikel (der, die, das)
- Kardinalzahlen (1,2,...)
- Negationen (kein, keine)

[2] vgl. erster Satz im Anhang A.2; Prädikat im Aktiv ergänzt um *Benutzer*

Wir benutzen unbestimmte Artikel, um eine definierende Instanz eines Bezeichners zu kennzeichnen, und bestimmte Artikel und Kardinalzahlen, um benutzende Instanzen zu kennzeichnen (siehe auch Abschnitt 4). Negationen wirken als definierende Instanzen.

Prädikate Das Prädikat als Argument deklariert den a-posteriori Typ des darauffolgenden Bezeichners. Existiert kein Bezeichner, so wird ein anonymer Bezeichner angenommen. Im aktuellen Gültigkeitsbereich muss dann genau ein Bezeichner vorhanden sein, der sich an den deklarierten Typ anpassen lässt. Ist kein solcher Bezeichner oder mehrere vorhanden, handelt es sich um einen Typfehler.

Das Prädikat als Verb konstituiert einen typisierten Ausdruck. Der Typ der Argumente (PP des Satzes) wird gegen die Typen der für das jeweilige Literal deklarierten Parameter geprüft (laut Glossar). Die Argumente werden entsprechend ihrer Beschriftungen (*labels*) sortiert; die Reihenfolge gleich beschrifteter Argumente jedoch nicht verändert.

Bezeichner Bezeichner verbinden die Menge von Anforderungen untereinander. Bezeichner sind optional; wenn sie entfallen wird eine anonyme Referenz erzeugt, die einen im Gültigkeitsbereich eindeutigen Typ besitzen muss, über den sie identifiziert wird. Wir unterscheiden definierende und benutzende Instanzen von Bezeichnern und führen eine Namensanalyse durch (siehe Abschnitt 4).

Modalverben Modalverben im Satz beschreiben das Merkmal der Variation für die Anforderung. Wir unterscheiden:

- Soll-Anforderungen
- Kann-Anforderungen
- Darf-Nicht-Anforderungen
- Tun

Die Art des Modalverbs beschreibt die, ob ein Variationspunkt für die betreffende Anforderung erstellt wird: Eine Kann-Anforderung impliziert mehrere Umsetzungsmöglichkeiten, während eine Soll-Anforderung strikt nach genau einer Umsetzung verlangt. Eine Darf-Nicht-Anforderung bezeichnet eine negative Abhängigkeit im Anforderungsgraphen, so dass die bezeichnete Anforderung nicht erfüllt sein darf, wenn die bezeichnende Anforderung erfüllt sein soll.

Das Modalverb *tun* dient behelfsweise bei der Umformulierung von einer Zeitform in eine andere (Abschnitt 3.3). Sätze mit diesem Modalverb werden wie Sätze ohne Modalverb, d.h. Bedingungen, behandelt.

3.1.1 Relativsätze Relativsätze dienen in unserer Sprache der Verkürzung der Beschreibung. Statt zwei getrennte Hauptsätze hintereinander zu schreiben, kann ein Argument eines Satzes gewissermaßen *inline* durch eine Beschreibung in einem Relativsatz näher bestimmt oder näher beschrieben werden. Relativsätze

sind syntaktischer Zucker und können zu separaten Hauptsätzen umgewandelt werden. Wir unterscheiden zwei Arten von Relativsätzen:

- beschreibende Relativsätze
- bestimmende Relativsätze

Beschreibende Relativsätze müssen ein Modalverb (kann,soll,..) enthalten. Sie werden in separate Anforderungen übersetzt. Bestimmende Relativsätze sind Relativsätze, die kein Modalverb enthalten; sie werden in eine Bedingung übersetzt. Auf diese Weise kann eine Anforderung zu einer bedingten Anforderung werden, auch ohne die explizite Wenn-Dann-Syntax zu verwenden (Abschnitt 3.2).

Relativsätze sind ein Kompromiss zwischen der Normalform, die nur aus Hauptsätzen besteht, und der gebräuchlichen Schriftsprache, die Bedingungen, Anforderungen und zugehörige (implizierte) Prädikate in ein einziges Partizip zwängt. Eine übliche (aber nach unserer Syntax nicht zulässige) Präpositionalphrase könnte lauten:

Die kapazitiven Lasten

Die PP muss umformuliert werden, in eine der beiden Varianten:

Die Lasten, die kapazitiv sind, ...
Die Lasten, die kapazitiv sein sollen, ..

Die erste Variante, wenn eine bestimmende Wirkung (nur solche Lasten, die kapazitiv sind) beabsichtigt ist, und die zweite, wenn eine beschreibende Wirkung (Lasten, die außerdem immer kapazitiv sein sollen) beabsichtigt ist.

Relativsätze ziehen leider einige Schwierigkeiten bezüglich der Argumentreihenfolge nach sich, die wir in Abschnitt 3.4 betrachten.

3.2 Bedingte Anforderungen

Eine Anforderungsbeschreibung untergliedert sich im Allgemeinen in einen Glossar-Teil, der für alle Projekte innerhalb einer Domäne (eines Gewerks) gültig ist und vorab erstellt wird, sowie einen Anforderungs-Teil, der projektspezifisch ist.

Eine Anforderung besteht im Allgemeinen aus einer Bedingung, unter der sie zu erfüllen ist, und einer Konsequenz, der beschreibt, was zu erfüllen ist:

```
Anforderung ::= (Bedingungen 'dann')? Konsequenz .
Bedingungen ::= Bedingungen 'und' 'wenn' VP | 'Wenn' VP .
Konsequenz  ::= VP .
Anforderung ::= Bedingungen 'dann' ':' Spiegelstriche .
Spiegelstriche ::= Spiegelstriche '-' VP | '-' VP .
```

Wird keine Bedingung angegeben, so soll die Konsequenz universell gültig sein.

Bedingungen (Wenn) beschreiben eine zur Laufzeit erfolgte Zustandsänderung („ist gestartet“), oder eine Eigenschaft (Invariante, „ist aus Holz“), oder eine Modell-Variante („kann aus Holz sein“). In der konkreten Syntax können sie statt eines „echten“ Modalverbs auch den Platzhalter „tun“ enthalten.

Konsequenzen (Dann) beschreiben eine Anforderung und deren Modus (*soll/muss*, *kann/muss-nicht*, *darf-nicht*). In der konkreten Syntax müssen sie immer ein Modalverb enthalten.

Durch eine Art von Relativsätzen – bestimmende Relativsätze – können Anforderungen um eine Bedingung erweitert werden, ohne dass die charakteristischen Schlüsselwörter *wenn* und *dann* vorhanden sind (siehe Abschnitt 3.1.1). Dies ist ein Kompromiss zu Lasten der einfachen Lesbarkeit und zu Gunsten der leichten Transformierbarkeit bestehender Anforderungen (mit Nebensätzen) in solche der kontrollierten Sprache.

Ein Erweiterung, von der wir im Beispiel im Anhang Gebrauch machen, sind Spiegelstriche anstatt der Konsequenzen, die vermeiden, dass die Bedingung in jedem Satz wiederholt werden muss.

3.3 Zeitformen

Für die Beschreibung von Systemverhalten erachten wir eine Beschreibung des Zustands zeitlich relativ zu Operationen, die auf ihn einwirken können, für erforderlich. Wir wollen daher in 3 Zeitformen unterscheiden:

Vorzeitigkeit $<$ Die durch das Verb angegebene Handlung hat noch nicht stattgefunden (entspricht Futur 1).
Gleichzeitigkeit $=$ Die Handlung findet gerade statt (entspricht Partizip 1).
Nachzeitigkeit $>$ Die Handlung ist abgeschlossen (entspricht Perfekt).

Durch Kombination von Zeitformen und Passiv ergibt sich eine Vielzahl von gebeugten Verbformen. Die gebeugten Verbformen sollen auf eine Normalform reduziert werden. Die Tabelle zeigt die Kombinationsmöglichkeiten von Zeitform und Aktiv/Passiv. Die Bestimmung der Zeitform ist nicht heuristisch, nur unübersichtlich.

Die Tabelle 2 unterstrichenen Varianten sind verboten, denn:

$=_{[d]}$: Aussagesätze sind keine Anforderungen
$<_{[d]}$: Anforderungen dürfen nicht rückwirkend formuliert sein
† Vertauschen von unmarkierten Argumenten ohne Passiv verändert die Semantik ohne die Syntax anzupassen (Artikel, sog. *determiner* werden verworfen)

Anhand der Information über das Modalverb (*soll*, *kann*, *tun* oder keines) lässt sich aus dem Hilfsverb (*werden* usw.) die Zeitform eindeutig bestimmen, wie in Tabelle 3 dargestellt.

Aus den normalisierten Zeitformen und dem Wenn-Dann-Kontext lässt sich die folgende Semantik ableiten:

Zeitform	Aktiv	Passiv
Anforderung ($<_{[d]}$)	Der A soll das B dem C geben.	Das B soll dem C *durch* A gegeben werden.
Anforderung ($=_{[d]}$)	Der A gibt das B dem C.	Das B wird dem C durch A gegeben.
Anforderung ($>_{[d]}$)	Der A soll das B dem C gegeben haben.	Das B soll dem C durch A gegeben worden sein.
Zukunft ($<_{[w]}$)	(Wenn) der A das B dem C geben (soll)	(Wenn) das B dem C durch A gegeben werden (soll)
Gegenwart ($=_{[w]}$)	(Wenn) der A dem B das C gibt (~~soll~~)	(Wenn) das B dem C durch A gegeben wird
Perfekt ($>_{[w]}$)	(Wenn) der A dem B das C gegeben hat	(Wenn) das B dem C durch A gegeben wurde/worden ist

Verb ohne unmarkiertes Akkusativobjekt: keine Passivform vorhanden

Anforderung ($<_{[d]}$)	Das A soll dem B gleichen.	†Dem B soll das A gleichen. Dem A soll das B gleichen(d) werden.
Zukunft ($<_{[w]}$)	(Wenn) das A dem B gleichen (soll)	†(Wenn) dem B das A gleichen (soll) (Wenn) dem A das B gleichen(d) werden (soll)
Gegenwart ($=_{[w]}$)	(Wenn) das A dem B gleicht (~~soll~~)	†(Wenn) dem B das A gleicht (Wenn) das A dem B gleichend ist
Perfekt ($>_{[w]}$)	(Wenn) das A dem B geglichen hat	†(Wenn) dem B das A geglichen hat (Wenn) das A dem B gleichend gewesen ist

Adjektiv-Prädikat (ohne Objekt):

Anforderung ($<_{[d]}$)	Das A soll aktiv sein.	Das A soll aktiv werden.
Zukunft ($<_{[w]}$)	(Wenn) das A aktiv sein soll (soll)	(Wenn) das A aktiv werden (soll)
Gegenwart ($=_{[w]}$)	(Wenn) das A aktiv ist (~~soll~~)	(Wenn) das A aktiv geworden ist
Perfekt ($>_{[w]}$)	(Wenn) das A aktiv war	(Wenn) das A aktiv gewesen ist

Tabelle 2. Zeitformen in Kombination mit Passiv-Aktiv-Formen

Zeitform	Aktiv	Passiv
Anforderung ($<_{[d]}$)	soll + Infinitiv	soll + Partizip + werden
Anforderung ($=_{[d]}$)	Präsens	werden + Partizip
Anforderung ($>_{[d]}$)	Partizip + haben.	Partizip + worden sein.
Zukunft ($<_{[w]}$)	(Wenn) Infinitiv (soll)	(Wenn) Partizip + werden (soll)
Gegenwart ($=_{[w]}$)	(Wenn) Präsens (~~soll~~) / (Wenn) Infinitiv + tun (~~soll~~)	(Wenn) Partizip + werden (~~soll~~)
Perfekt ($>_{[w]}$)	(Wenn) Partizip + haben	(Wenn) Partizip + wurden/worden sind

Tabelle 3. Zusammenfassung Zeitformen

- Anforderung ($<_{[d]}$): Definition einer Nachbedingung
- Zukunft ($<_{[w]}$): Introspektion (*reflection*) einer Anforderung
- Gegenwart ($=_{[w]}$): Prüfen einer Invariante (über einen Zeitverlauf)
- Perfekt ($>_{[w]}$): Auslösen eines Ereignisses/Signals/Callbacks

3.4 Spezialfälle der konkreten Syntax

Obwohl die konkrete Syntax so einfach wie möglich gehalten wurde, kann sie nicht alle mehrdeutigen Formulierungen natürlicher Sprache unterbinden. Relativpronomen sind im Allgemeinen nicht geeignet, um die beabsichtigte Position eines Arguments in der Parameterliste festzustellen, wie das folgende Beispiel zeigt:

Die Pumpe, die das System steuert.

Soll hier die Pumpe das (Fluid-)System steuern, oder durch das (Steuer-)System gesteuert werden? Da das Relativpronomen fest an erster Stelle des Relativsatzes steht, sind Konventionen notwendig, um die korrekte Reihenfolge der Argumente herzustellen.

In diesem Abschnitt stellten wir die konkrete Syntax einer kontrollierten Sprache vor, die Lesbarkeit von natürlicher Sprache erhält und die Voraussetzungen für Überprüfbarkeit wie eine formale Sprache bereitstellt.

4 Namensanalyse

Wir wollen eine Namensanalyse durchführen, um sicherzustellen, dass Objekte, die bei der Beschreibung von Prozessen herangezogen werden, zuvor selbst beschrieben wurden. Diese Vollständigkeitsprüfung ist Teil unseres Ziels, natürliche Sprache automatisiert überprüfbar zu machen. Die im vorangegangenen Abschnitt vorgestellt Syntax liefert die Grundlage dazu.

Um eine Namensanalyse auf Ebene der Bezeichner, die Bestandteil der Präpositionalphrasen (PP) sind, müssen wir unterscheiden zwischen definierenden und benutzenden Bezeichnern. Wir erlauben kein Verdecken (*shadowing*) von Bezeichnern; mehrmalige Definitionen eines Bezeichners mit dem gleichen Namen sind Fehler. Den Bezeichnern wird bei ihrer Definition der Typ des vorangestellten Prädikats zugewiesen.

Wir unterscheiden zwischen Definition und Benutzung anhand des Artikels:

Definition *Eine Pumpe P1 ...*
Benutzung *Die Pumpe P1 ...*

Unbestimmte Artikel (ein, eine, usw.) definieren einen Bezeichner, bestimmte Artikel (der, die, das, usw.) und Kardinalzahlen (1,2, usw.) greifen benutzend auf einen Bezeichner zu.

Anonyme Bezeichner Im Sprachgebrauch wird eine Instanz regelmäßig nicht benannt, wenn der Typ ein Objekt eindeutig identifiziert.

Es ist möglich, den Bezeichner in einer Präpositionalphrase auszulassen. Der Übersetzer erzeugt dann eine anonyme Referenz mit dem deklarierten Typ. Auf diese anonyme Referenz kann über den Typ zugegriffen werden, wenn dieser im Gültigkeitsbereich eindeutig ist. Ist ein anonymer Bezeichner nicht eindeutig über den Typ identifizierbar, liegt ein Fehler vor. Eine Ausnahme bilden implizite Bezeichner für Verben im folgenden Abschnitt.

Beispiel[3]:

> *Wenn eine Pumpe in Betrieb genommen werden soll, dann ...*
> *... die Pumpe muss mit einem Fördermedium gefüllt sein.*

Im ersten Beispielsatz wird ein anonymer Bezeicher b_1 für *Pumpe* erzeugt. Im zweiten Satz kann auf diesen anonymen Bezeicher b_1 zugegriffen werden, da b_1 der einzige Bezeichner vom Typ *Pumpe* ist, der außerdem keine Untertypen hat[4].

Verben erzeugen anonyme Bezeichner Unsere Syntax erlaubt keine Adverbien, Adjektive oder andere vorangestellten Modifikatoren. Trotzdem ist es notwendig, Prozesse näher zu beschreiben. In natürlicher Sprache dienen Adverbien diesem Zweck; eine Konvention, um für Verben direkt bei deren Gebrauch mit einen Bezeichner zu benennen, hat sich in natürlicher Sprache nicht herausgebildet. Vermutlich liegt dies daran, dass Menschen eher überladene Verbbedeutungen dynamisch auflösen als für jeden Zweck neue Verben zu kreieren. Im technischen Kontext wollen wir aber genau das tun, um anhand der Literale ihre Bedeutung unterscheiden zu können – und nicht anhand des Kontextes, was immer das auch sein mag.

Wollen wir Verben modifizieren, können wir die nur tun, indem wir sie als Argumente innerhalb eines anderen Satzes benutzen. Für jede Benutzung eines Verbs erzeugt unser Übersetzer implizit einen anonymen Bezeichner. Im Gegensatz zu implizit deklarierten Argumenten, ist es kein Fehler, wenn ein impliziter Bezeichner, der auf ein Verb verweist (erkennbar anhand seines a-priori Typs), nicht eindeutig einer Instanz zugeordnet werden kann. In diesem Fall wird die zuletzt erzeugte Instanz gewählt und eine Warnung ausgegeben. Grund ist, dass syntaktisch keine direkte Möglichkeit besteht, Bezeichner für Verben zu deklarieren.

Beispiel:

> *Die Pumpe muss angeschlossen werden*$_1$*.*
> *Der Frequenzumrichter muss angeschlossen werden*$_2$*.*
> *Das Anschließen*$_2$ *(..) muss vorschriftsmäßig sein.*

[3] siehe Abschnitt A.2, erster Absatz
[4] siehe Diagramm der Typhierarchie in Abbildung 2 im Anhang

Das Verb *angeschlossen werden* erzeugt zwei anonyme Instanzen. Dem Verb ist im Glossar der Typ *Anschließen* zugewiesen[5]. Mittels dieses Typs wird auf die (zuletzt) im zweiten Satz deklarierte Instanz zugegriffen. Der Übersetzer erzeugt dazu eine passende Warnung, dass der anonyme Bezeichner mehrere Kandidaten hatte, um auf eventuelle Fehler in der Reihenfolge der Sätze aufmerksam zu machen.

5 Typsystem

Wir wollen eine Typanalyse durchführen, um sicherzustellen, dass gleich benannte Objekte auch die gleiche Sache bezeichnen. Dazu vergleichen wir den deklarierten Typ bei der Definition eines Bezeichners mit dem deklarierten Typ bei dessen Benutzung. Weiterhin vergleichen wir in Abschnitt auch die deklarierten (a-priori) Typen mit den von der Umgebung erwarteten (a-posteriori) Typen. Diese Konsistenzprüfung ist ein weiterer Schritt zu unserem Ziel, natürliche Sprache überprüfbar zu machen.

Das Glossar ist die Sammlung aller Typdeklarationen, die durch den Domäneningenieur geschrieben werden sollen. Das Glossar soll für die Domäne relevanten Klassen von Objekten (Pumpen, Systeme, Nachrichtenarten, Anschlussarten, usw.), von Merkmalen (energieeffizient, schnell, blau, usw.) und von Prozessen (anschalten, benachrichtigen, anschließen, usw.) als Typdeklarationen enthalten. Das Typsystem kann in der Definitionstabelle des Übersetzers abgelegt werden. Die Typdeklarationen unterscheiden sich von und werden ergänzt durch die Definition von Verben, die wir in Abschnitt 5.3 erläutern.

5.1 Typdeklarationen

Typdeklarationen bilden den benutzerdefinierten Teil der abstrakten Syntax der domänenspezifischen Anforderungssprache. Typdeklarationen beschreiben, was man linguistisch als *Konzepte* bezeichnen würde. Sie sind Äquivalenzklassen, die Verben, Adjektive und Substantive zu Prädikaten zusammenfassen. Im nichtbenutzerdefinierten Teil der abstrakten Syntax befinden sich z.B. die Interpretation der Modalverben (soll, kann) aus Abschnitt 3.1. Eine einfache Typdeklaration für ein Substantiv lautet z.B.:

Eine Pumpe ist ein System.
Ein System ist ein Objekt

Diese Deklaration deklariert den Typ *Pumpe* und setzt ihn zugleich in Untertypsbeziehung zum Typ *System*. Der Typ *Objekt* ist vordefiniert und gemeinsamer Obertyp aller Substantive in unserer Sprache. Verben und Adjektive induzieren ebenfalls Typen und benötigen daher eine Typdeklaration:

[5] Verbdefinition in Abschnitt 5.3, Glossardefinition in Abschnitt A.2.1; der erste Beispielsatz wurde ergänzt

Eine Inbetriebnahme ist ein Prozess.
Eine Vorschriftsmäßigkeit ist ein Merkmal.

Wir deklarieren zunächst ausschließlich das Gerundium (die substantivierte Form) als Typ. Diese benötigen wir um Verben und Adjektive selbst näher beschreiben zu können. Später definieren wir Aliase für die Substantivierungen, um Verben und Adjektive als Prädikat eines Satzes verwenden zu können.

In Fällen wie *Pumpen*, in denen der Plural eines Substantivs mit dem Gerundium eines Verbs zusammenfällt, muss darauf geachtet werden, eine Variante der Substantivierung zu wählen, für die keine Typkollisionen vorliegen. Dank der Namensanalyse fallen solche Fälle auch bei nachträglichen Ergänzungen auf. Dies ist möglich, da Substantive, Verben und Adjektive jeweils einen eigenen gemeinsamen Obertyp besitzen. Das Typsystem verfügt über einen gemeinsamer Top-Typ $\top$ aller Typen, dessen Literal der aber nicht als Obertyp deklariert werden kann.

Fest vordefinierte Typen im Typsystem sind:

- Objekt
- Prozess
- Merkmal

5.2 Untertypen und Polymorphie

Wir wollen Synonyme für Wörter benutzen können. Weiter wollen wir Bezeichner zusammen mit einem Verb benutzen können, ohne ihren exakten Typ benennen zu müssen. Dann können wir ihre Definition nachträglich und modular verändern, ohne jede Benutzung abändern zu müssen. Verben sollen nicht überladen werden müssen um eine Überbestimmung in ihren Parametertypen auszugleichen.

Wir erlauben daher die Definition von Obertypen bei der Deklaration eines Typs, also:

Eine Pumpe ist ein Objekt.

Dies deklariert den Typ *Pumpe* als Untertyp des Typs *Objekt*.

Obertypen können mehrfach deklariert werden:

Eine Pumpe ist ein Objekt.
Eine Pumpe ist ein System.

Dies deklariert den Typ *Pumpe* als Untertyp sowohl zu *Objekt* als auch *System*. Die so deklarierten Obertypen werden zu einer Menge zusammengefasst. Nach Aufsammeln aller Typdeklarationen in der Anforderungsbeschreibung

Die Syntax für Untertypsdeklarationen lautet:

```
TDecl ::= PP ('ist'|'sind') PP '.'
```

5.3 Verbdefinitionen

Verben besitzen in der Regel mehrere Parameter (Valenz) von unterschiedlichem Typ. Die Typen dieser Parameter sollen deklariert und durch unseren Übersetzer geprüft werden. Die Argumente einer Verbalphrase können zudem mit Präpositionen beschriftet sein (siehe Abschnitt 3.1). Die Anzahl der Parameter, ihr Typ und ihre Beschriftung werden in der Verbdefinition festgelegt. Die Verbdefinition konstruiert daraus einen zuvor deklarierten Typ .

Verben sowie Adjektive, deren Partizipien das Prädikat eines Satzes bilden können (z.B. *energieeffizient sein*), benötigen zudem Aliase für verschiedene Flexionsformen (Beugungen). Ein einzelnes Verb oder Adjektiv umfasst 3 Untertypen: eine für jede Zeitform (siehe Abschnitt 3.3).

Die konkrete Syntax für Typdeklarationen und Verbdefinitionen lautet:

```
VDef ::= PP ('ist'|'sind') ':' VP '.'
```

Zu beachten ist, dass der Obertyp aus Lesbarkeitsgründen bei Verbdefinitionen auf der rechten Seite des *ist* steht.

Beispiel[6]:

> *Ein Füllen ist: die Pumpe soll mit einem Fördermedium gefüllt sein.*
> *Ein Betreiben ist: der Frequenzumrichter kann als Generator betrieben werden.*

In Abschnitt 3.1 haben wir erklärt, Präpositionen als Beschriftungen für Argumente von Verben zu benutzen. Diese Beschriftungen sind spezifisch für jedes Verb. Sie können spezifisch für jede Konjugationsform definiert werden; in der Regel sind sie jedoch für alle Konjugationen jeweils in der Aktiv- und der Passiv-Form eines Verbs identisch (*betreiben* bzw. *betrieben werden*).

Bei der Definition von Verben wird die Namensanalyse außer Kraft gesetzt: die Form des Artikels wird ignoriert.

Repräsentation in der Definitionstabelle Aus den im Beispiel gezeigten Definitionen leiten wir die in Tabelle 4 dargestellte Repräsentation in der Definitionstabelle ab.

5.4 Typberechnungen

Wir definieren Typen als Mengen von Eigenschaften. Sei E die Grundmenge aller atomaren Eigenschaften (die Literale der Typnamen), und sei ein Typ $A \subseteq E$ eine Teilmenge aller möglichen Eigenschaften, dann ist die Berechnung, ob der A Untertyp $\sqsubseteq$ von $B \subseteq E$ ist, abbildbar auf die Teilmengeneigenschaft:

$$A \sqsubseteq B \Leftrightarrow closure(A) \supseteq closure(B)$$

[6] siehe Abschnitt A.2.1, Glossareinträge 7 und 16

id	VName	VType	VLabels	VArgs
#1	*gefüllt sein*	Ausgeben	{$\varepsilon \Rightarrow$#2, mit$\Rightarrow$#3}	
#2				[Pumpe]
#3				[Fördermedium]
#4	*betrieben werden*	Betreiben	{$\varepsilon \Rightarrow$#5, als$\Rightarrow$#6}	
#5				[Freqenzumrichter]
#6				[Generator]

Tabelle 4. Verben *gefüllt sein* und *betrieben werden* in der Definitionstabelle

Dazu sammeln wir zuerst alle Typdeklarationen durch den Übersetzer auf, bilden den transitiven Abschluss über die Obertypsrelation (Abschnitt 5.2), und das Ergebnis als $closure(A)$ für jeden Typ A in der Definitionstabelle. Enthielten A und B zuerst nur eine konkrete Eigenschaft, nämlich die Mitgliedschaft im Typ ihres Namens, d.h. $A = \{a\}, B = \{b\}$ mit $a, b \in E$, so ergänzt die Abbildung $closure :: E \to E$ alle diejenigen Eigenschaften, die in Obertypen von A bzw. B enthalten sind. Das Typsystem bildet einen Verband mit den Operationen Typerweiterung $\sqcup$ und kleinster gemeinsamer Obertyp $\sqcap$, $\top = \emptyset$, $\bot = E$. Der gemeinsame Obertyp $\top = \emptyset$ ist die Schnittmenge aller atomaren Eigenschaften; der gemeinsame Untertyp aller Typen $\bot = E$ ist die Vereinigung aller atomaren Eigenschaften, also die Menge aller möglichen Eigenschaften. Für domänenspezifische Anwendungen kann es sinnvoll sein, Kriterien für den gegenseitigen Ausschluss von Eigenschaften zu definieren. Da wir in dieser Arbeit keine Codegenerierung betrachten, ist es unerheblich, ob $\bot$ tatsächliche Werte repräsentieren kann.

Wird ein Verb im Text aufgefunden, wird seine Definition aus der Definitionstabelle als erwarteter (a-posteriori) Typ geladen. Der a-priori Typ ist die Liste der PP. Um zu überprüfen, ob der a-priori Typ auf den a-posteriori Typ anpassbar ist, wird die Liste der PP von links nach rechts darauf überprüft, ob sich das jeweilige Paar von Beschriftung (*label*) und Argument in der Definition des Verbs findet. Dabei wird mittels einer Hilfsfunktion die Multiplizität von Präpositionen (Beschriftungen) berechnet und daraus der a-posteriori Typ für das jeweilige Argument mit dem benutzten Parameter bestimmt.

Beispiel:

> *Ein Ausgang ist ein Anschluss. Ein Anschluss ist ein Objekt. Die Lasten sind eine Last. Eine Last ist ein Objekt. Ein Anschließen ist ein Prozess.*

Eine Untertypsbeziehung wird für diesen Ausschnitt des Glossars (siehe Abbildung 1 und Abschnitt A.2.1) wiefolgt berechnet:

$$Ausgang \sqsubseteq Anschluss$$
$$\Updownarrow$$
$$\{Ausgang, Anschluss, Objekt\} \supseteq \{Anschluss, Objekt\}$$

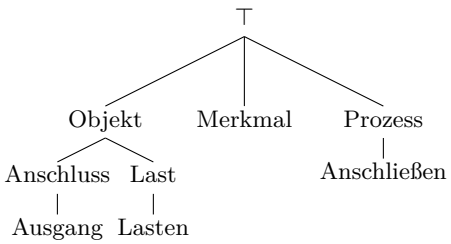

Abb. 1. Ausschnitt der Typhierarchie

Multiplizität von Argumenten Mehrfach auftretende Beschriftungen behandeln ihre Argumente positional; unterschiedliche Beschriftungen können dagegen beliebig untereinander vertauscht werden. In der Regel muss nur das unbeschriftete ε-Argument mehrfach auftreten. Seltener findet sich noch ein mehrfaches Auftreten des Präposition *von*, *durch* oder *zwischen* in einem Satz. Wir behandeln alle Präpositionen gleich, und erlauben eine beliebige Anzahl an Wiederholungen.

6 Schlussfolgerungen und Ausblick

Der Beitrag dieser Arbeit zur Unterstützung der Anforderung besteht in folgenden Punkten: zuerst definieren wir eine Grammatik für Schablonen zur Anforderungsbeschreibung. Aus der Grammatik erzeugen wir einen Parser, um syntaktische Korrektheit zu automatisch überprüfen, statt lediglich *best-practice* Regeln zu befolgen (Abschnitt 3). Wir führen eine Namensanalyse auf Bezeichnern (benannten Argumenten von Verben) durch (Abschnitt 4). Weiter führen wir eine Typprüfung auf Bezeichnern und Parametern von Verben durch (Abschnitt 5.3), um deren richtige Kombination sicherzustellen. Die Syntax für Typdeklarationen ist eine Teilmenge der natürlichen Sprache (Deutsch), genauso wie die Syntax für Terme (Anforderungen), und das obwohl Typdeklarationen ein Konzept sind, das in natürlicher Sprache nicht vorkommt. Damit erhalten wir ein Werkzeug, das das Schreiben von konsistenten Anforderungsdokumenten maßgeblich vereinfachen kann.

Diese Arbeit beschreibt Maßnahmen zur Konsistenzsicherung und -prüfung für die textuelle Form von Anforderungen, jedoch nicht für den Entwurf, den die Anforderungen beschreiben. Die kontrollierte Sprache sollte daher eingebettet werden in ein System zur Analyse der Zusammenhänge zwischen Anforderungen. Da die Zusammenhänge zwischen Anforderungen ebenfalls in natürlicher Sprache geschrieben werden (Bedingte Anforderungen, Modalverben) können die dafür notwendigen Informationen entweder aus der abstrakten Syntax entnommen oder mittels Codegenerierung für eine Zwischensprache zur Anforderungsmodellierung in ein Zwischenformat ausgeben werden.

Codegenerierung soll die Anbindung der Anwendungsspezifikation an andere Artefakte des Entwicklungsprozesses (z.B. VHDL-Code in technischen Syste-

men) ermöglichen, die einen höheren Grad an Verfeinerung aufweisen, als sich dies sinnvoll in natürlicher Sprache beschreiben lässt. Dazu stellt man idealerweise eine bidirektionale Korrespondenz mittels Parser und Codegenerator für die Zielsprache her, so dass Änderungen (am Entwurf bzw. der Implementierung) in beide Richtungen propagiert werden können.

Für die weitere Entwicklung des gesamten Ansatzes, natürliche Sprache für domänenspezifische Modellierung einzusetzen, ist es interessant, einen Blick auf andere Domänen zu werfen als die in dieser Arbeit fokussierten technischen Systeme. Interessante Domänen sind z.B. die Softwareentwicklung (starke Ähnlichkeit zu technischen Systemen), Kochrezepte (schablonenbasierte Syntax, nichttechnischer Natur) oder juristische Texte (fokussiert auf Syntax und Äquivalenz auf Termebene anstatt abstrakter Assoziation).

Das vorgestellte Typsystem basiert auf expliziten Deklarationen. Hier wäre stattdessen der Einsatz von Typinferenz möglich.

Danksagung Diese Arbeit entstand im Rahmen des vom Bundesministerium für Bildung- und Forschung (BMBF) geförderten Forschungsprojekts ELSY (Nr. 16M3202D).

Literaturverzeichnis

[Ambriola u. Gervasi 2006] Ambriola, Vincenzo ; Gervasi, Vincenzo: On the Systematic Analysis of Natural Language Requirements with Circe. In: *Automated Software Engineering* 13 (2006), S. 107–167

[Cowan 2000] Cowan, John W.: *The Complete Lojban language*. The Logical Language Group, Inc., 2000. – ISBN 0–9660283–0–9

[Farfeleder 2012] Farfeleder, Stefan: *Requirements Specification and Analysis for Embedded Systems*, Technische Universität Wien, Diss., November 2012

[Hull u. a. 2005] Hull, Elizabeth ; Jackson, Ken ; Dick, Jeremy: *Requirements Engineering*. Springer, 2005. – ISBN 978–1–85523–879–4

[KSB 2014] KSB: *Betriebs-/ Montageanleitung PumpDrive 2*. KSB Aktiengesellschaft, Johann-Klein-Str. 9, 67225 Frankenthal, Juni 2014. http://www.ksb.com/propertyblob/2286696/data/PumpDrive%202.pdf

[Kuhn u. Bergel 2014] Kuhn, Tobias ; Bergel, Alexandre: Verifiable source code documentation in controlled natural language. In: *Sciene of Computer Programming* 16 (2014), S. 121–140

[Lachowicz u. a. 2015] Lachowicz, Dom ; Figuière, Hubert ; Sevior, Martin: *Link Grammar*. http://www.abisource.com/papers/. Version: August 2015

[Palmer u. a. 2010] Palmer, Martha ; Gildea, Daniel ; Xue, Nianwen: Semantic Role Labeling. In: *Synthesis Lectures on Human Lanugage Technologies* (2010)

[Rupp 2014] Rupp, Chris: *Requirements-Engineering und -Management*. Hanser, 2014. – ISBN 978–3–446–43893–4

A Praxisbeispiel

Wir zeigen am Beispiel der *Betriebs-/ Montageanleitung* des *PumpDrive2* [KSB, 2014], einem Frequenzumrichter zur Steuerung von Pumpenaggregaten, wie die kontrollierte Sprache zur Modellierung von Anforderungen eingesetzt wird.

In Abschnitt A.1 ist der Originalausschnitt wiedergegeben. Danach folgt in Abschnitt A.2 der in kontrollierte Syntax umformulierte Text. Abschnitt A.2.1 enthält das daraus erarbeitete Glossar. In Abbildung 2 ist die deklarierte Typhierarchie abgebildet.

A.1 Original-Abschnitt: 7 Inbetriebnahme / Außerbetriebnahme[7]

Vor Inbetriebnahme müssen folgende Punkte sichergestellt sein:

– Pumpe ist entlüftet und mit Fördermedium gefüllt.

[7] vgl. [KSB, 2014] S. 48

- Pumpe wird nur in Auslegefliessrichtung durchströmt, um einen generatorischen Betrieb des Frequenzumrichters zu vermeiden.
- Ein plötzliches Anfahren des Motors bzw. des Pumpenaggregats verursacht keine Schäden an Personen und Maschinen.
- Es sind keine kapazitiven Lasten z.B. zur Blindstromkompensation an den Ausgängen des Geräts angeschlossen.
- Die Netzspannung entspricht dem für den Frequenzumrichter zugelassenen Bereich.
- Der Frequenzumrichter ist vorschriftsmäßig elektrisch angeschlossen (⇒ Kapitel 5.4 Seite 21)
- Freigaben und Startbefehle, die den Frequenzumrichter starten können, deaktiviert sind (siehe Digitaleingänge DI-EN Digitaler Freigabe-Eingang und DI1 Anlagenstart).
- Am Leistungsmodul des Frequenzumrichters liegt keine Spannung an.
- Der Frequenzumrichter bzw. das Pumpenaggregat darf nicht über die zugelassene Nennleistung belastet werden.

A.2 Abschnitt 7 in kontrollierter Syntax

Wenn eine Pumpe in Betrieb genommen werden soll, dann:

- die Pumpe muss entlüftet sein
- die Pumpe muss mit einem Fördermedium gefüllt sein
- die Pumpe darf nicht entgegen Auslegefliessrichtung durchströmt werden
- der Frequenzumrichter darf nicht als Generator betrieben werden.

Wenn die Pumpe in Betrieb genommen werden soll, und wenn ein Motor angefahren wird, dann:

- keine Schäden dürfen an Personen verursacht werden
- keine Schäden dürfen an Maschinen verursacht werden.

Wenn die Pumpe in Betrieb genommen werden soll, dann dürfen keine Lasten, die kapazitiv sind, an den Ausgängen, die durch den Frequenzumrichter besessen werden, angeschlossen sein. Eine Blindstromkompensation soll eine Last, die kapazitiv sein soll, beschreiben.

Die Netzspannung muss dem Bereich, für den der Frequenzumrichter zugelassen ist, entsprechen.

Der Frequenzumrichter muss angeschlossen werden. Das Anschließen, das die Elektrizität betrifft, muss vorschriftsmäßig sein.

Wenn die Pumpe in Betrieb genommen werden soll, dann:

- die Freigaben, die den Frequenzumrichter starten können, müssen deaktiviert sein
- die Startbefehle, die den Frequenzumrichter starten können, müssen deaktiviert sein
- am Leistungsmodul, das durch den Frequenzumrichter besessen wird, soll keine Spannung anliegen.

Der Frequenzumrichter darf nicht oberhalb der Nennleistung, für die der Frequenzumrichter zugelassen ist, belastet werden.

Das Pumpenaggregat darf nicht oberhalb der Nennleistung, für die das Pumpenaggregat zugelassen ist, belastet werden.

A.2.1 Notwendige Glossareinträge für Abschnitt 7

Hinweis: die Typen *Objekt*, *Merkmal* und *Prozess* sind vordefiniert, siehe Abschnitt 5.1.

Eine Pumpe ist ein System.

Eine Inbetriebnahme ist: die Pumpe soll in Betrieb genommen werden[8]. Eine Inbetriebnahme ist ein Prozess. Ein Betrieb ist eine Inbetriebnahme.

Ein Entlüften ist: die Pumpe soll entlüftet werden. Ein Entlüften ist ein Prozess.

Ein Füllen ist: die Pumpe soll mit einem Fördermedium gefüllt sein. Ein Füllen ist ein Prozess. Ein Fördermedium ist ein Material. Ein Material ist ein Objekt.

Ein Durchströmen ist: die Pumpe kann entgegen der Auslegefliessrichtung durchströmt werden. Ein Durchströmen ist ein Prozess. Eine Auslegefliessrichtung ist eine Richtung. Die Richtung ist eine Regel. Eine Regel ist ein Objekt.

Ein Betreiben ist: der Frequenzumrichter kann als Generator betrieben werden. Ein Betreiben ist ein Prozess. Ein Frequenzumrichter ist ein System. Ein Generator ist ein System. Ein System ist ein Objekt.

Ein Anfahren ist: der Motor soll angefahren werden. Ein Anfahren ist ein Prozess. Ein Motor ist ein System.

Ein Verursachen ist: Die Schäden an einem Objekt können verursacht werden. Ein Verursachen ist ein Prozess. Die Schäden sind ein Objekt. Die Personen sind ein Objekt. Die Maschinen sind ein Objekt.

Eine Kapazitivität ist: die Lasten können kapazitiv sein. Eine Kapazititivät ist ein Merkmal. Die Lasten sind eine Last. Eine Last ist ein Objekt.

Ein Besitzen ist: durch den Frequenzumrichter können die Ausgängen besessen werden. Ein Besitzen ist ein Prozess. Die Ausgängen sind ein Ausgang. Ein Ausgang ist ein Anschluss. Ein Anschluss ist ein Objekt.

Ein Anschließen ist: die Lasten können an den Ausgängen angeschlossen sein. Ein Anschließen ist ein Prozess.

Ein Beschreiben ist: eine Blindstromkompensation kann eine Last beschreiben. Ein Beschreiben ist ein Prozess. Eine Blindstromkompensation ist ein Prozess.

Ein Entsprechen ist: die Netzspannung muss einem Bereich entsprechen. Ein Entsprechen ist ein Prozess. Die Netzspannung ist eine Spannung. Die Spannung ist ein Objekt. Ein Bereich ist eine Regel.

[8] Das Verb lautet *genommen*; wir bilden es auf das Prädikat *Inbetriebnahme* ab

Ein Zulassen ist: für einen Bereich ist ein Frequenzumrichter zugelassen. Ein Zulassen ist ein Prozess.

Ein Anschließen ist: der Frequenzumrichter kann angeschlossen sein. Ein Anschließen ist ein Prozess.

Eine Vorschriftsmäßigkeit ist: das Anschließen muss vorschriftsmäßig sein. Eine Vorschriftsmäßigkeit ist ein Merkmal.

Ein Anschließen ist: die Elektrizität kann angeschlossen sein. Die Elektrizität ist ein Objekt.[9]

Ein Starten ist: die Freigaben können den Frequenzumrichter starten. Ein Starten ist ein Prozess. Die Freigaben sind eine Freigabe. Eine Freigabe ist eine Regel.

Ein Starten ist: die Startbefehle können den Frequenzumrichter starten. Die Startbefehle sind ein Startbefehl. Ein Startbefehl ist ein Ereignis. Ein Ereignis ist ein Objekt.

Ein Deaktivieren ist: die Freigaben können deaktiviert sein. Ein Deaktivieren ist ein Prozess.

Ein Deaktivieren ist: die Startbefehle können deaktiviert sein.

Ein Anliegen ist: am Leistungsmodul soll eine Spannung anliegen. Das Anliegen ist ein Prozess. Ein Leistungsmodul ist ein Objekt. Eine Spannung ist ein Objekt.

Ein Besitzen ist: ein Leistungsmodul kann durch den Frequenzumrichter besessen wird.

Ein Belasten ist: der Frequenzumrichter kann oberhalb einer Grenzleistung belastet werden. Ein Belasten ist ein Prozess. Eine Nennleistung ist eine Leistung. Eine Leistung ist eine Grenzleistung. Eine Grenzleistung ist eine Grenze. Eine Grenze ist ein Wert. Ein Wert ist ein Objekt.

Ein Belasten ist: das Pumpenaggregat kann oberhalb einer Grenzleistung belastet werden.

[9] betrifft/betreffen ist ein Platzhalter-Verb, das durch sein erstes Argument, das ein Prozess sein muss, ersetzt wird

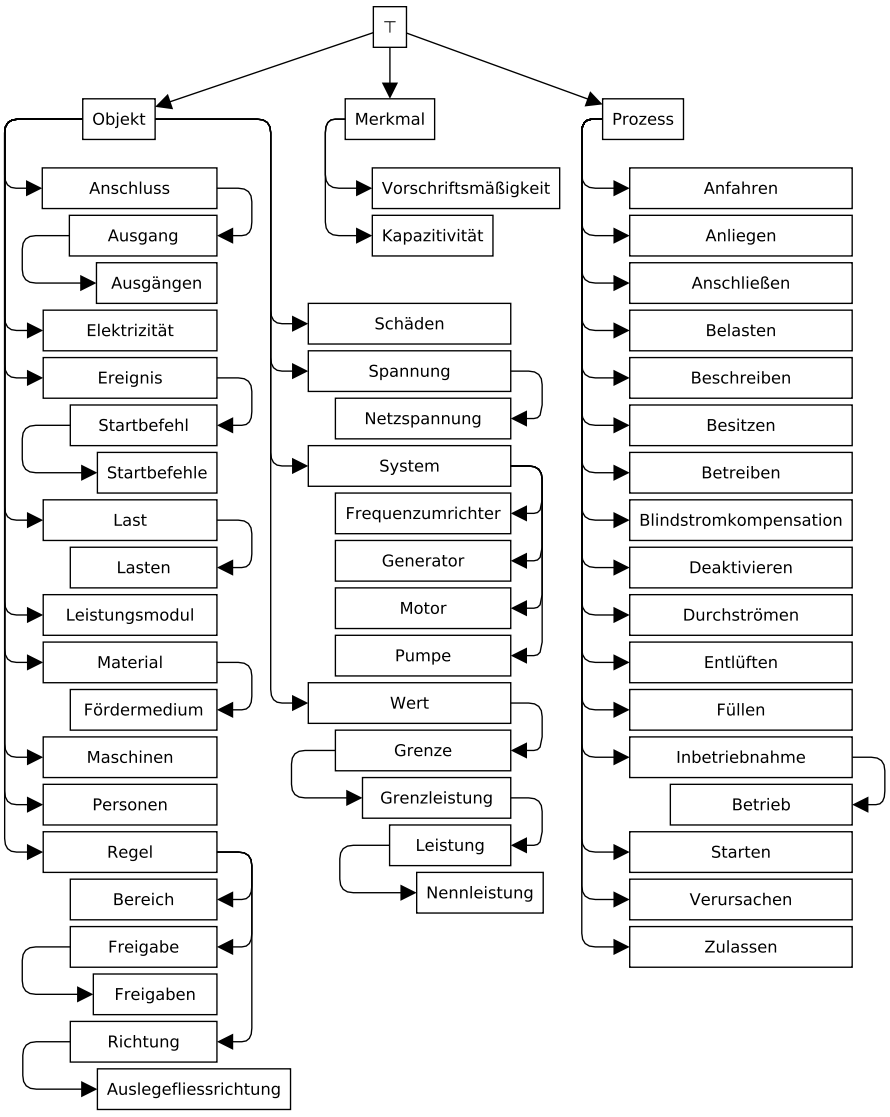

Abb. 2. Typhierarchie für Abschnitt 7