universität
uulm

Fakultät für Ingenieurwissenschaften, Informatik und Psychologie
Datenbanken und Informationssysteme

Enhancing BPMNGen with Prompting Strategies for Automated BPMN 2.0 Process Model Generation

Abschlussarbeit an der Universität Ulm

Vorgelegt von:
Philipp Letschka
philipp.letschka@uni-ulm.de
1050994

Gutachter:
Prof. Dr. Manfred Reichert

Betreuer:
Luca F. Hörner

2025

Fassung 10. November 2025

Satz: PDF-LaTeX 2ε

Danksagung

Das ist der Text der Danksagung

Zusammenfassung

Das ist der Text der Zusammenfassung

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

1.2 Problemstellung und Zielsetzung

1.3 Struktur der Arbeit

2 Theoretische Grundlagen

3 Umstrukturierung und Innovation

Als erstes gilt es herauszufinden welcher Teil des Code, der zuständig für das prompting ist, wie verbessert werden kann. Das Projekt von TBA benutzt zur Erstellung von BPMN Diagrammen die OpenAI API und verwendet hier die bereitgestellte Technologie der Assistants.

3.1 Generelle Umstrukturierungen

Während der Code gut für seinen (bisherigen) speziellen Anwendungsfall ist, können hier einige Verbesserungen gemacht werden.

3.1.1 Objektorientierter Ansatz

Das Zeil ist es, den Code einfach erweiterbar und wartbar zu machen. Hierfür ist es wichtig, den code möglichst schnell an Änderungen der OpenAI Api anpassen zu können. Um das zu erreichen wird ein Objektorientierter Ansatz gewählt. Die objektorientierte Programmierung bietet für den Aufbau des Prompting-Codes viele Vorteile und macht die Entwicklung langfristig übersichtlicher und wartbarer. Wir erstellen eine abstrakte Klasse ‘ai.ts’ welche die gesamte Logik des Prompting beinhält und eine Klasse ‘openai.ts’ welche von der AI Klasse erbt. Die OpenAi Klasse muss nun nur noch Methoden implementieren, welche konkret auf die aktuelle version der API angepasst sind. Durch die Verwendung von abstrakten Methoden wie generateContent(), createTitle() oder processResponse() wird sichergestellt, dass jede konkrete Implementierung dieselbe Schnittstelle einhält, aber ihre eigenen internen Abläufe definieren kann. Dies erleichtert den Austausch und die Erweiterung von Modellen, ohne den restlichen Code verändern zu müssen. Darüber hinaus

werden wiederkehrende Prozesse, etwa das Speichern von Verläufen, das Verarbeiten von Antworten oder die Konvertierung zwischen Formaten, zentral in der Basisklasse gekapselt. Falls sich die API ändert, kann dies nun einfach in der Erbenden Klasse angepasst werden, ohne die dahinterliegenden Logik verändern zu müssen.

So sieht nun in abgespeckter Variante die Klasse für die OpenAI API aus. Es gibt eine Methode `mapPromptInput();` um den Prompt in das richtige Format der API zu bringen, `generateContent();` um den eigentlichen API aufruf durchzuführen und `preocessResponse();` um die Antwort der API auszulesen.

```
export class ChatGPT extends Ai {
  openai = new OpenAI({
    apiKey: OPENAI_API_KEY,
  });
  assist = await openai.beta.assistants.retrieve("asst_...");

  protected mapPromptInput(input) {
    return {
      input: input.prompt,
      model: this.model,
    };
  }

  protected async generateContent(input) {
    return await openai.beta.threads.runs.createAndPoll(
      thread_id,
      { assistant_id: assist.id },
      { role: "user", content: input }
    );
  }

  protected processResponse(response) {
    return response.output_text.toString();
  }
}
```

3.1.2 Von Assistants zu Responses

Die Änderungen welche im vorherigen Kapitel beschrieben wurden, zeigen gleich ihren Effekt, da OpenAI ankündigt ihre Assistants API einstellen zu wollen. Sie empfehlen einen Umzug zu ihrer neuen Responses API. Dies ist nun recht einfach umzusetzten, da wir nur die elementaren Methoden der API ändern müssen. So sieht nun die neue Methode `generateContent();` aus:

```
1 protected async generateContent(input) {
2   return this.openai.responses.create(input);
3 }
```

Einer der zentralen Unterschiede zwischen der Assistants- und der Responses-API besteht darin, dass die System-Instructions bei der Verwendung der Responses-API manuell übergeben werden müssen. Dadurch ist es notwendig, die entsprechenden Anweisungen bei jeder Anfrage erneut mitzusenden. Dieser Umstand bringt jedoch nicht nur zusätzlichen Aufwand mit sich, sondern eröffnet auch neue Möglichkeiten: Die Instructions können flexibel und situationsabhängig angepasst werden, wodurch sich das Verhalten des Modells dynamisch steuern lässt. Im folgenden Abschnitt wird gezeigt, wie dieser Ansatz weiter verbessert und effizienter gestaltet werden kann.

3.1.3 Verbesserung der instructions

Da die Instructions nun manuell mit jeder Anfrage übergeben werden, bietet sich die Möglichkeit, deren Aufbau gezielt zu optimieren. Ziel dieser Optimierung ist es, die Anzahl der benötigten Input-Tokens zu reduzieren, ohne dabei Qualität einzubüßen. Im besten Fall wird die Ausgabequalität sogar verbessert. Der Assistant erhält als Grundlage zwei PDF-Dateien, die BPMN-Diagramme im Detail beschreiben, sowie zwei Textdateien: eine mit der Definition des verwendeten JSON-Formats und eine mit allgemeinen Regeln zum Aufbau der Diagramme. Die beiden PDF-Dokumente umfassen zusammen mehr als 10 MB und über 100 Seiten Text. Da ChatGPT bereits ein solides Grundverständnis von BPMN-Diagrammen besitzt, werden diese umfangreichen Dateien aus den Instructions entfernt. Mehrere Tests zeigen, dass

sich diese Reduktion nicht negativ auf die Ergebnisqualität auswirkt. Dadurch lassen sich eine große Zahl an Tokens sowie Rechenzeit und Kosten einsparen.

Die beiden verbliebenen Textdokumente werden anschließend zusammengeführt, überarbeitet und in ein einheitliches, strukturiertes Format gebracht. Alle Regeln sind in einer geordneten Liste zusammengefasst und durch sogenanntes structured prompting klarer und maschinenlesbarer gestaltet. Ergänzend werden den Instructions zwei illustrative Beispiele hinzugefügt: Zum einen ein minimales Beispiel, das den grundsätzlichen Aufbau des JSON-Formats verdeutlicht und die obligatorischen Elemente zeigt. Zum anderen ein umfangreicheres, praxisnahes Beispiel, das ein vollständiges BPMN-Diagramm mit allen relevanten Komponenten abbildet. Diese Kombination sorgt dafür, dass der Assistant sowohl einfache als auch komplexe Diagramme präzise interpretieren und reproduzieren kann.

3.2 Formatauswahl

Bisher wird die KI angewiesen, das Diagramm in einem eigens definierten JSON-Format zu erzeugen. Dieses Format wurde jedoch speziell für diesen Anwendungsfall entworfen und existiert in dieser Form nicht offiziell. Entsprechend konnte das Modell während des Trainings kein Vorwissen darüber erwerben, sondern muss das Format ausschließlich auf Grundlage der bereitgestellten Instructions erlernen. Dadurch besteht die Möglichkeit, dass Fehler auftreten, etwa dann, wenn die Anweisungen unvollständig sind oder dem Modell bestimmte Kontextinformationen fehlen.

Um dieses Problem zu vermeiden, wird künftig die Option ergänzt, dass die KI ihre Ausgabe auch direkt im offiziellen Standardformat erzeugen kann. Das weltweit am häufigsten verwendete Austauschformat für BPMN-Diagramme ist XML, zu dem umfangreiche Dokumentation und etablierte Werkzeuge existieren. Dennoch bietet das eigens entwickelte JSON-Format einen entscheidenden Vorteil: Es besitzt eine deutlich höhere Informationsdichte und lässt sich dadurch kompakter und effizienter verarbeiten. Beide Formate haben somit ihre jeweiligen Stärken und Einsatzgebiete.

Kriterium	JSON-Format	XML-Format
Vorteile	hohe Informationsdichte, geringer Tokenverbrauch, schnelle Generierung.	Standardisiert, gut dokumentiert, weit verbreitet, einfachere Instructions
Nachteile	Kein Standard, Lernaufwand, komplizierter Konvertierungsalgorythmus.	hoher Tokenverbrauch, umfangreiche Syntax, unübersichtlicher, mehr Kosten, längere Generierung.

Tabelle 3.1: Vergleich der Vor- und Nachteile der unterstützten Ausgabeformate

Für die Weiterentwicklung ist ein flexibles System das Ziel, das eine dynamische Auswahl des Ausgabeformats besitzt. Dadurch kann der Nutzer selbst entscheiden, welches Format im jeweiligen Anwendungsfall die besseren Ergebnisse liefert. Da sich die Formatwahl ähnlich wie die Wahl des verwendeten Modells direkt auf die Qualität der Ergebnisse auswirkt, wird die Auswahlmöglichkeit direkt in die Modellkonfiguration integriert. So kann beispielsweise zwischen Varianten wie ‘gpt-4.1-mini (xml)’ und ‘gpt-4.1-mini (json)’ gewählt werden. Wird kein Format angegeben, erfolgt die Ausgabe standardmäßig im XML-Format.

Eine Anfrage sieht damit z.B. folgendermaßen aus:

```
// POST /threads
{
  "inputString": "Bitte generiere mir ein BPMN Diagram,
      welches den Ablauf in einem Restaurant zeigt",
  "model": "gpt-5 (xml)",
}
```

Bei einer Anfrage kann dann die jeweilige AI über eine Map

```
const availableGPTs: Map<string, Ai>
```

zugeordnet werden, welche die Anfrage bearbeitet.

// TODO: update neuerungen noch machen

3.3 Dateien

Um die Qualität des Promptings weiter zu verbessern, soll eine Funktionalität implementiert werden, die es ermöglicht, auch Dateien direkt an die KI zu übermitteln. Dabei steht im Vordergrund, dass das Verfahren sowohl im Frontend als auch im Backend möglichst unkompliziert umgesetzt werden kann. Es soll keine aufwendigen oder zeitraubenden Konvertierungen erfordern und eine breite Auswahl an Dateitypen unterstützen, um die Nutzung so flexibel wie möglich zu gestalten. Nach einer Analyse der OpenAI-Dokumentation unter `https://platform.openai.com/docs/guides/images-vision?api-mode=responses&format=base64-enco` `analyze-images` zeigt sich, dass die einfachste und zugleich effizienteste Methode zur Dateiübertragung die Verwendung einer Base64 Data URL ist. Diese Variante bietet eine einfache Möglichkeit, Binärdaten wie Bilder oder Dokumente direkt in Textform zu kodieren und zu übermitteln, ohne zusätzliche Infrastruktur oder spezielle Upload-Mechanismen zu benötigen.

Eine Base64 Data URL ist im eine Textdarstellung einer Datei, die direkt in eine URL eingebettet wird. Dabei werden die ursprünglichen Binärdaten in ein spezielles Textformat namens Base64 umgewandelt. Diese kodierten Daten beginnen typischerweise mit einer Kennzeichnung wie ... und enthalten danach die eigentlichen kodierten Inhalte. Der große Vorteil liegt darin, dass der Browser oder die API diesen Text automatisch wieder in die ursprüngliche Datei zurückwandeln kann. Auf diese Weise lassen sich Dateien direkt in JSON API-Anfragen integrieren, ohne dass zusätzliche Dateipfade, Server oder externe Speicherorte erforderlich sind. Dieses Verfahren ist daher besonders gut geeignet, um eine einfache, schnelle und universell kompatible Dateiübertragung zu ermöglichen. Dadurch, dass die Datei nun einfach als String übergeben werden kann, können wir die Datei dem Body der Anfrage hinzufügen.

Eine Anfrage sieht damit z.B. folgendermaßen aus:

```
1 // POST /threads
2 {
3   "inputString": "Bitte generiere mir ein BPMN Diagram,
4       welches den Ablauf in einem Restaurant zeigt",
5   "model": "gpt-5 (xml)",
6   "file": "...",
```

```
}
```

Der string wird auf Validität geprüft

```
private checkBase64DataUrl(dataUrl: string): boolean {
  regex = /^data:([\w.+-]+\/[\w.+-]+)?;base64,[\w+\/]+=*$/;
  return regex.test(dataUrl);
}
```

und dann im richtigen Format an die OpenAI API gesendet:

```
const imageInstructions = {
  role: "user",
  content: [
    {
        type: "input_file",
        file_url: input.getFileDataUrl() as string,
    } as ResponseInputFile,
  ],
} as ResponseInputItem
```

3.4 Chain of Thought

Um nun die Erzeugung von BPMN Diagrammen weiter zu verbessern, wird eine Technik implementiert die ‘Chain of Thought’ heisst. Dadurch ermöglicht es dem Nutzer sich mit dem ChatBot zu unterhalten. Der Nutzer kann unter anderem um Rat fragen, Diagramme beschreiben lassen, sich Ideen anhören und vieles mehr. Der BPMN Bot soll zu einem vollständigem Chatbot werden, welcher nicht nur Diagramme erstellen kann sondern auch vieles mehr. Der Chatbot soll zudem auch Fragen stellen können, falls etwas unklar ist. Dafür benötigt der Chatbot völligen Zugriff auf die Bisherige Unterhaltung sowie den aktuellen Stand des Diagramms.

3.4.1 Implementierung eines neuen Modus

Um die erstellung eines Diagramm nicht komplizierter für den Nutzer zu machen ist das Ziel, dass die direkte Erstellung eines Diagramms weiterhin möglich ist. Dafür wird nun ein weiterer Parameter der Anfrage hinzugefügt. Über den $mode$ Parameter kann nun ausgewählt werden inwiefern die KI antwortet. Der bisherige Modus, welcher ausschließlich Diagramme erstellt, wird vorerst $quick$ genannt. Den Chain of Thought modus wird $detail$ genannt. Eine Anfrage welche nun eine Unterhaltung ermöglich sieht beispielsweise so aus:

```
// POST /threads
{
  "inputString": "Bitte schlag drei Prozessbeschreibungen
    vor, aus denen dann eine ausgewählt wird um ein Diagramm
    zu erstellen.",
  "model": "gpt-5 (xml)",
  "mode": "detail",
}
```

3.4.2 Konversationskontext

Da es nun darum geht ein Chat zu implementieren, bei dem die KI möglichst gut auf Nachrichten reagieren kann, ist es wichtig, dass die KI zugriff auf vorherige Nachrichten sowie auf das Diagramm hat. Wäre das nicht der Fall, könnte die KI keine Fragen über das Diagramm beantworten und auch keine richtige Unterhaltung führen, da immer nur die aktuelle Nachricht bereitgestellt wird. Bei LLM Anbietern wie OpenAI ist es notwendig, dass die Nachrichten historie in der Anfrage mitgeschickt wird.

Hierfür müssen alle Nachrichten eines Theads aus der Datenbank geladen werden. Die Nachrichten werden dann auf wesentliche gefiltert und auf ein kompaktes Foramt gebracht. Die OpenAI Klasse muss dann nur noch die Nachrichten auf das gefoderte Format bringen und in der Anfrage mitschicken. Da die Anfagen an die KI immer komplexer werden, wird nun eine Klasse $PromptInput$ angelegt. Diese Klasse beinhaltet alle Informationen welche der KI mitgesendet werden sollen. Bei

einer erstellung diese Objekt können alle Rohdaten wie Instructions, Dateien oder der Chatverlauf übergebe werden, welche die Klasse dann automatisch formatiert. Ein entscheidender Schritt hierbei ist es die Teile der Nachrichten zu entfernen, in denen die KI mit einem Diagramm geantwortet hat. Das ist wichtig, da die Diagramme viele Tokens beinhalten und eigentlich nur das aktuellste Diagram wichtig ist. Hier bei kann es aber auch sein, dass das Diagramm noch vom Nutzer bearbeitet wurde. Um dies zu berücksichtigen sind die Diagramme nuicht Teil der Nachrichten, welche mitgesendet werden. Die KI Schnittstellenimplementierung, kann dann ein Objekt von PromptInput entgegennehmen und dieses recht einfach durch fertige Helfermethoden auf das gewünschte Format bringen. Die Klasse hat diese Attribute:

```
export class PromptInput {
  instructions: string[];
  history: {role: "user" | "assistant", content: string}[];
  prompt: string;
  file?: ;
}
```

Es ist so nun möglich die Nachrichten aus der Datenbank abzurufen und diese werden in der PromptInput Klasse auf das oben angegebene Format gebracht...

```
const instructions = [formatInstructions, modeInstructions];
const chatsFromDB = getAllChatsFromDB(threadID);
new PromptInput(instructions, prompt, chatsFromDB, file);
```

...und dann auf das benötigte Format für die KI gebracht zu werden:

```
protected mapPromptInput(input: PromptInput) {
  [...]
  const historyInstructions = input.history.map((item) => {
    return {role: item.role, content: item.content};
  });
  return {
    input: [systemInstructions, historyInstructions,
            userInstructions, fileInstructions],
    model: this.model,
```

```
  };
}
```

Weitergehend soll nun auch die aktuellste version des Diagramms mitgeschickt werden, damit die KI dieses Bestensmöglichst sowohl bearbeiten als auch Fragen darüber beantworten kann. Da das aktuelle Diagramm nicht Teil der Nachrichten ist, wird dieses in die system instructions gesetzt. Wir nennen es die Update instructions und erhalten diese durch das Laden der aktuellen version aus der Datenbank.

```
protected updateInstructions(threadID: string, format: format){
  const diagram = getLatestDiagramFromDB(threadID);
  return [`The The following diagram has already been created:
    ${format == "xml" ? diagram?.xml : diagram?.json}`]
}
```

Damit hat die KI nun alle Informationen die sie benötigt um eine Konversation zu führen

3.4.3 Konversationen

3.5 Weitere Anbieter

3.5.1 Grok

3.5.2 Gemini

3.5.3 Claude

3.6 Streaming

3.7 Schema-Constraining

3.8 Evtl. Sampling

3.9 Evtl. Reflective Prompting

4 Performanzanalyse

4.1 Qualität

4.2 Geschwindigkeit

4.3 Kosten

5 Verwandte Arbeiten

Hier sind verwandte Arbeiten

6 Fazit

6.1 Zusammenfassung

6.2 Ausblick

Notizen

Gefundene Paper

A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT

- **Autoren:** Jules White et al. (2023)
- **Inhalt:** Katalogisierung von Prompting-Mustern zur systematischen Wiederverwendung.
- **Link:** arxiv.org/abs/2302.11382

Chain-of-Thought Prompting Elicits Reasoning in Large Language Models

- **Autoren:** Jason Wei et al. (2022)
- **Inhalt:** Einführung der „Chain-of-Thought“-Technik zur Verbesserung logischer Schlussfolgerungen.
- **Link:** arxiv.org/abs/2201.11903

The Prompt Report: A Systematic Survey of Prompting Techniques

- **Autoren:** Schulhoff et al. (2024)

- **Inhalt:** Systematische Übersicht mit 58 Techniken, Kategorisierung und Beispielen.
- **Link:** arxiv.org/abs/2406.06608

A Systematic Survey of Prompt Engineering in Large Language Models

- **Autoren:** Sahoo et al. (2024)
- **Inhalt:** Überblick über Methoden, Anwendungen, Modelle und Herausforderungen.
- **Link:** arxiv.org/abs/2402.07927

Training Language Models to Follow Instructions with Human Feedback (InstructGPT)

- **Autoren:** Ouyang et al. (OpenAI, 2022)
- **Inhalt:** Einsatz von RLHF zur Verbesserung der Instruktionsbefolgung durch Sprachmodelle.
- **Link:** arxiv.org/abs/2203.02155

Vorlagenbeispiele

Diese kleine Einleitung soll dem Nutzer helfen selbst die eigene Arbeit mit LaTeX zu schreiben. Sie enthält Beispiele zu den wichtigsten Themen .

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Dokumentlgiederung

Für diese Arbeit verwendet man folgende LaTeX-Kommados zur Strukturierung:

```
\chapter{Einleitung}
\section{Dokumentgliederung}
\subsection{}
\subsubsection{}
```

Allerdings sollte man sich überlegen, ob man wirklich bis zur Stufe `subsubsection` Überschriften benötigt.

Illustrationen

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Bilder und Abbildungen

Auch in einer wissenschaftlichen Arbeit können Bilder und Abbildungen zur Veranschaulichung und zur Illustration sachlicher Inhalte integriert und einfügt werden. Für Fotografien und Bilder unterstützt PDF-LaTeX direkt `jpg` und `png`. Ansonsten empfiehlt es sich, Vektorgrafiken zu verwenden und diese als `pdf` zu speichern. Sollte ein Bild einmal von zu viel weißem Raum umgeben sein, kann man mit dem Werkzeug `pdfcrop` das Bild automatisch zuschneiden.

Mit Hilfe eines Labels `\label{fig:bild1}` kann man sich dann im fortlaufenden Text mittels eines Querverweises auf diese Grafik beziehen: `\ref{fig:bild1}`.

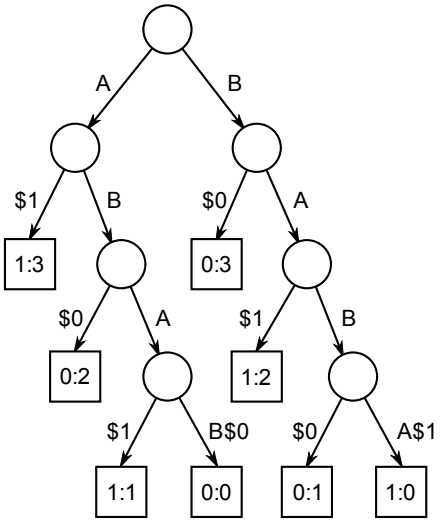

Abbildung 6.1: Beschreibung/Beschriftung des Bilds

An der Stelle des ref-Kommandos platziert LaTeX die Nummer der Abbildung: ‚siehe Abbildung 6.1‘.

Tabellen

Seite 21, Abschnitt 6.2, enthält Beispieltabelle 6.1. In vielen LaTeX-Büchern finden sich gute Anleitungen zum Erstellen von Tabellen. Komplexere Tabellen können sinnvollerweise in Excel oder einer anderen Tabellenkalulation vorgefertigt und mit einem Umwandlungsprogramm oder -werkzeug in LaTeX-Quellcode konvertiert werden.

A	B	C
x	x	x
x	x	x

Tabelle 6.1: Eine kleine Beispieltabelle

Formeln

Mathematische Formeln lassen sich in der Umgebung `math` erzeugen. Die Kurz-Schreibweise lautet `\( a^2+b^2=c^2 \)`; hierbei steht die Formel dann im laufenden Text: $a^2+b^2=c^2$. Die kürzeste Form ist mit zwei `$` um die Formel, z.B. so: Wasser ist H_2O. `H$_2$O`

Mit der Schreibweise `\[ y=x^2 \]` wird die Formel mittig in einer eigenen Zeile gesetzt, z.B.

$$y = x^2$$

Formeln in der Umgebung `equation` werden mittig in einer eigenen Zeile gesetzt und fortlaufend nummeriert:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{6.1}$$

Wenn wir z.B. über die beliebte Mitternachtsformel (Gleichung 6.1) Details im umliegenden Text schreiben wollen, lässt sich diese wie ein Bild oder eine Tabelle referenzieren, sofern man ihr ein Label zugewiesen hat..

Programmier-Code

Mehrzeiliger Programmier- und Quellcode kann mit `verbatim` in einer Umgebung gesetzt werden:

```
Dieser Text steht in einer verbatim-Umgebung und wird daher
in Schreibmaschinenschrift geschrieben.
LaTeX-Kommandos, z.B. \includegraphics[width=.6\textwidth]{bild.jp
werden nicht interpretiert, sondern "verbatim" ausgegeben.
```

Schöner und professioneller lässt sich Programmier-Code mit dem `listings`-Paket, eingeben, formatieren und ausgeben. Dazu kann man in der Präambel die Sprache angeben, in der die Quellcodes geschrieben sind.

```
1 public class Hello {
2     public static void main(String[] args) {
3         System.out.println("Hello World");
4     }
5 }
```

Innerhalb einer Zeile gibt man Wörter am Besten als `\verb##` an, dabei erwartet LaTeX zweimal das gleiche Zeichen als Begrenzer. Im Beispiel ist dies die Raute `#`, man kann aber auch jedes andere Zeichen nehmen, z.B. das Plus `+`.

Text

Textteile können bei Bedarf mit dem Befehl `\emph{}` *hervorgehoben* werden. Falls in einem Satz ein Punkt vorkommt, macht man danach kein Leerzeichen sondern eine Tilde (`z.~B.~so!`), denn dann fügt LaTeX den korrekten Abstand ein, z. B. so!

In der Präambel der vorliegenden tex-Datei gibt es den Befehl `hypenation`, der zur Silbentrennung da ist. LaTeX verfügt zwar über eine eingebaute Silbentrennung, die jedoch bei manchen Wörtern falsch trennt. Damit diese Wörter korrekt getrennt werden, gibt man sie dann mit dem Befehl in der Präambel an[1].

Fußnoten werden mit dem Befehl `footnote` mitten in den fortlaufenden Text eingefügt. [2]

In wissenschaftlichen Arbeiten muss man des öfteren andere Arbeiten zitieren. Dazu nutzt man die Stiloptionen und Zitierbefehle des Pakets `biblatex`, z. B. `numeric` (=Standard-Stil) oder `verbose` resp. `\cite{name}` oder `\autocite{name}`. In eckigen Klammern kann man noch die Seitenzahl angeben, falls notwendig. Der Name ist ein Schlüssel aus der Datei `bibliography.bib`. Falls einmal ein Werk nur indirekt zu einem Teil der Arbeit beigetragen hat, kann man es auch mit `nocite` angeben, dann landet es in der Literaturliste, ohne dass es im Text ausdrücklich zitert wird.

[1] Das Wort *Silbentrennung* ist hier das Beispiel
[2] Wie man schon im vorherigen Absatz sehen konnte.

Weiterführendes

Zum Schluss sei auf die Vielzahl an Büchern zu LaTeX verwiesen. In jeder Bibliothek wird sich eine Einführung finden, in der dann weitere Themen wie mathematische Formeln, Aufbau von Briefen und viele nützliche Erweiterungen besprochen werden.

A Quelltexte

In diesem Anhang sind einige wichtige Quelltexte aufgeführt.

```
function greet(name: string): void {
    console.log(`Hello, ${name}!`);
}
greet("World");
```

Literatur

[1] Jörg Knappen. *Schnell ans Ziel mit LATEX 2e*. 3., überarb. Aufl. München: Oldenbourg, 2009.

[2] Frank Mittelbach, Michel Goossens und Johannes Braams. *Der Latex-Begleiter*. 2., überarb. und erw. Aufl. ST - Scientific tools. München [u.a.]: Pearson Studium, 2005.

[3] Joachim Schlosser. *Wissenschaftliche Arbeiten schreiben mit LATEX : Leitfaden für Einsteiger*. 5., überarb. Aufl. Frechen: mitp, 2014.

[4] Thomas F. Sturm. *LATEX : Einführung in das Textsatzsystem*. 9., unveränd. Aufl. RRZN-Handbuch. Hannover [u.a.]: Regionales Rechenzentrum für Niedersachsen, RRZN, 2012.

[5] Herbert Voß. *LaTeX Referenz*. 2., überarb. u. erw. Aufl. Berlin: Lehmanns Media, 2010.

Name: Philipp Letschka Matrikelnummer: 1050994

Erklärung

Ich erkläre, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Philipp Letschka