Hochschule Aalen

Projektbericht
Studiengang Medieninformatik

Entwurf und Implementierung offener Typen in C++

von

Laurin Agostini

60526

Betreuender Professor: Prof. Dr. Christian Heinlein

Einreichungsdatum: ??. Oktober 2019

Eidesstattliche Erklärung

Hiermit erkläre ich, **Laurin Agostini**, dass ich die vorliegenden Angaben in dieser Arbeit wahrheitsgetreu und selbständig verfasst habe.

Weiterhin versichere ich, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben, dass alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und dass die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

Ort, Datum Unterschrift (Student)

Kurzfassung

Ziel der Kurzfassung ist es, einen (eiligen) Leser zu informieren, so dass dieser entscheiden kann, ob der Bericht für ihn hilfreich ist oder nicht (neudeutsch: Management Summary). Die Kurzfassung gibt daher eine kurze Darstellung

- des in der Arbeit angegangenen Problems
- der verwendeten Methode(n)
- des in der Arbeit erzielten Fortschritts.

Dabei sollte nicht auf die Struktur der Arbeit eingegangen werden, also Kapitel 2 etc. denn die Kurzfassung soll ja gerade das Wichtigste der Arbeit vermitteln, ohne dass diese gelesen werden muss. Eine Kapitelbezogene Darstellung sollte sich in Kapitel 1 unter Vorgehen befinden.

Länge: Maximal 1 Seite.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

RUP Rational Unified Process 5

1. Einleitung

Die Einleitung dient dazu, beim Leser Interesse für die Inhalte Praxissemesterberichts zu wecken, die behandelten Probleme aufzuzeigen und die zu ihrer Lösung entwickelten Konzepte zu beschreiben.

1.1. Motivation

In der Motivation wird dargestellt, welche Bedeutung die im Praxissemester zu entwickelnden Lösungen für das betreuende Unternehmen haben. Es wird beispielsweise aufzeigt, in welches Produkt sie eingehen, welcher Ablauf verbessert werden soll etc.

1.2. Problemstellung und -abgrenzung

Die Problemstellung dient dazu, das zu lösende Problem klar zu definieren und abzugrenzen. Der Praktikant soll ein klares Verständnis des zu lösenden Problems haben. Insbesondere soll auch verhindert werden, dass zu viele Probleme gleichzeitig angegangen werden. Eine Negativabgrenzung verhindert, dass beim Leser später nicht erfüllte Erwartungen geweckt werden.

1.3. Ziel der Arbeit

Mit dem Ziel der Arbeit wird der angestrebte Lösungsumfang festgelegt. An diesem Ziel wird entschieden, ob das Praktikum erfolgreich absolviert wurde oder nicht.

1.4. Vorgehen

Nachdem mit Problemstellung und Ziel gewissermaßen Anfangs- und Endpunkt des Praktikums beschrieben sind, wird hier der zur Erreichung des Ziels eingeschlagene

Weg vorgestellt. Dazu werden typischerweise die folgenden Kapitel und ihr Beitrag zur Erreichung des Ziels der Arbeit kurz beschrieben. Die folgenden Kapitel sind ein – möglicher – Aufbau, Abweichungen können durchaus notwendig sein. Zur Darstellung des Vorgehens ist eine grafische Darstellung sinnvoll, bei der die einzelnen Lösungsschritte und ihr Zusammenhang dargestellt werden. Ein Beispiel hierfür findet sich in Abbildung 1.1.

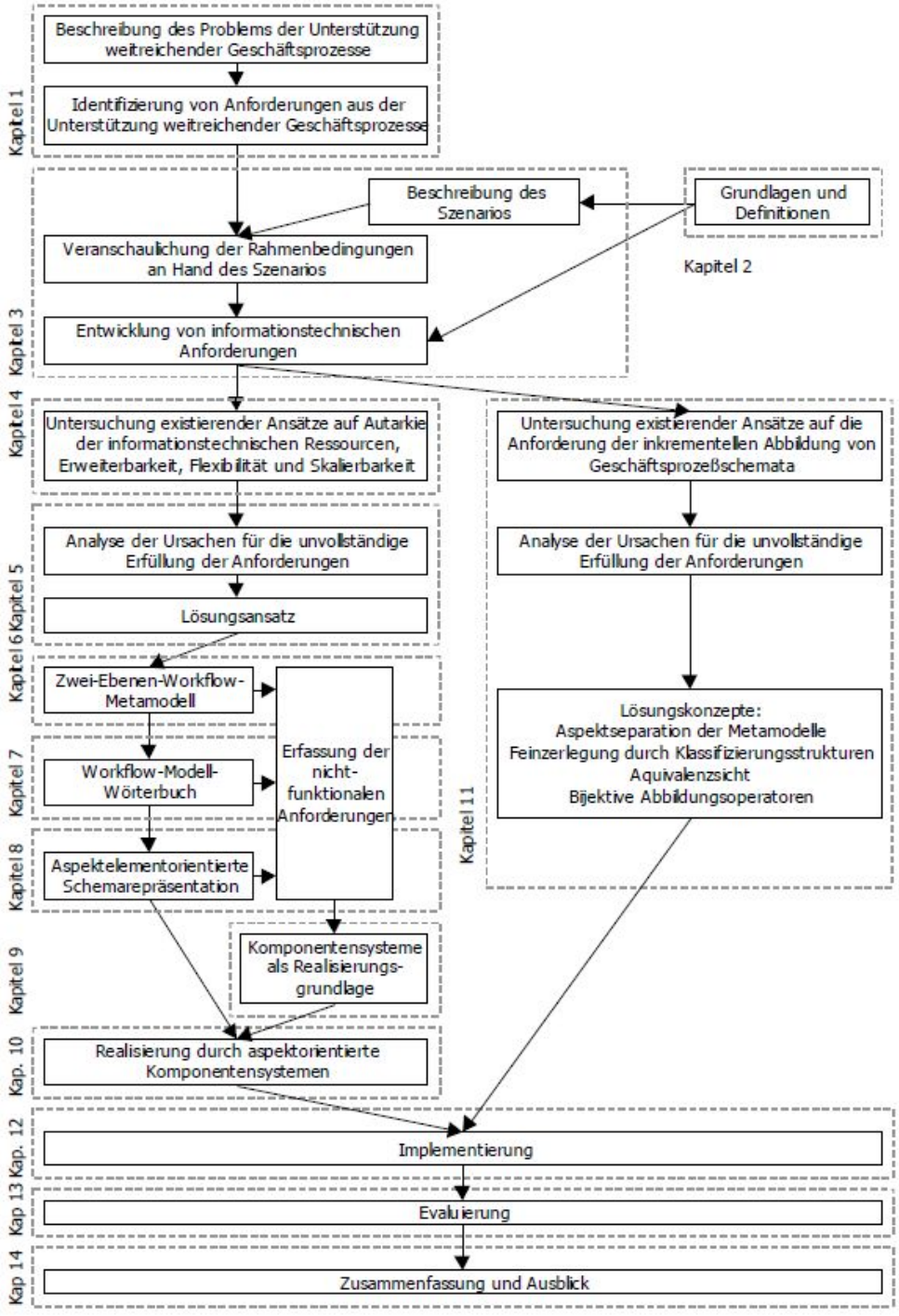

Abbildung 1.1.: vorgehen nach [**Schmidt:Geschaeftsprozesse**]

2. Grundlagen

In diesem Kapitel das für das Praktikum relevante Grundlagenwissen dargestellt. Der Praktikant soll hierzu das ihm durch Vorlesungen bekannte, bzw. durch Recherchen vertiefte theoretische Wissen darstellen, das für die Lösung der im Praktikum gestellten Probleme notwendig ist.

Dabei ist darauf zu achten, nur solche Inhalte in das Grundlagenkapitel aufzunehmen, die später auch verwendet werden (Problembezogenheit). Ebenso ist auf eine ausreichend tiefe und vollständige Darstellung der Grundlagen zu achten.

Für die Erstellung des Literaturverzeichnisses wird das Werkzeug JabRef[**JabRef:JabRef**] verwendet.

Sie können aber auch das Werkzeug Citavi[**SAS:Citavi**] benutzen und dort nach BIBTEX exportieren.

2.1. Grundlagengebiet A

2.1.1. Definition AA

2.1.2. Definition AB

2.2. Grundlagengebiet B

2.2.1. Definition BA

2.2.2. Definition BB

3. Problemanalyse

Die Analyse des zu lösenden Problems ist Grundlage für jedes ingenieurmäßige Vorgehen. Daher soll in diesem Kapitel das zu lösenden Problem auf Basis des im Grundlagenkapitel aufbereiteten Wissens analysiert werden. Hierzu ist insbesondere notwendig zu klären, wie sich das Gesamtproblem in Teilprobleme zerlegen lässt und welche Abhängigkeiten zwischen diesen bestehen.

Bei Software-Projekten befindet sich an dieser Stelle typischerweise die Anforderungsanalyse des Rational Unified Process (RUP).

4. Typen und Objekte

4.1. Definition von Typen

Analog zu klassischen Klassen (oder Strukturen) in C++ muss ein offener Typ vor Benutzung, d.h. Erzeugung von Objekten dieses Typen, erst definiert werden. Zuerst muss dafür ein Typ, hier als Beispiel der Typ *Person*, mit

```
OT_TYPE(Person)
```

definiert werden. Dies erzeugt intern zwei Klassen/Strukturen:

- `struct ot_types::Person : public ot::Type`
- `struct Person : public ot::Reference<ot_types::Person>`

ot_types::Person ist hierbei die interne Repräsentation des Typen und für den Benutzer offener Typen nicht direkt relevant. Sie erbt von der Basisklasse für alle offene Typen *ot::Type*. Die für den Benutzer wichtige Klasse *Person* erweitert den internen Typen um die Referenzsemantik.

4.2. Referenzsemantik

Um die aus anderen Hochsprachen, zum Beispiel Java oder C#, gewohnte Referenzsemantik auch für die offenen Typen zu ermöglichen erbt die, durch die Definition eines Typen entstehende, Klasse *Person* von

```
public ot::Reference<ot_types::Person>
```

ot::Reference<T> ist für die Speicherverwaltung und für boolesche Vergleichsoperatoren zuständig. *ot::Reference<T>* besitzt hierzu einen Zeiger auf den Typ T

```
T* ptr = nullptr;
```

Die booleschen Vergleichsoperatoren werden durch klassische Vergleiche auf Zeiger abgebildet. D.h zwei Referenzen sind gleich, wenn sie auf den gleichen Speicherbereich zeigen und eine Referenz kann implizit nach *bool* mit *ptr != nullptr* umgewan-

delt werden. Die Speicherverwaltung wird durch Referenzzählung implementiert. Um zu gewährleisten, dass der Speicherbereich erst freigegeben wird, wenn ihn keine Referenz mehr referenziert, ist die Referenzzählung in die Singleton-Klasse

```
class PtrManager
```

ausgelagert. Diese zählt die Referenzierungen für jeden Speicherbereich und bietet Methoden um die jeweilige Anzahl zu inkrementieren bzw. zu dekrementieren. Eine Referenz inkrementiert den Zähler, wenn sie mit einem Zeiger initialisiert wird. Sie dekrementiert den Zähler im Destruktor. Wenn die Referenz den Zeiger einer anderen Referenz kopiert, wird erst der Zähler für den aktuellen Zeiger dekrementiert und dann der Zähler für den kopierten Zeiger inkrementiert. Wenn nach einer Dekrementierung der Zähler gleich 0 ist, wird der Speicherbereich des Zeigers freigegeben. Für einen Zeiger auf *nullptr* wird die Inkrementierung/Dekrementierung übersprungen.

4.3. Erzeugen von Objekten

Mit dem Typen *Person* können nun, wie in C++ üblich, Objekte erzeugt werden

```
Person person;
```

4.3.1. Nil-Objekte

Standardmäßig ist ein Objekt eines Typen ein Nil-Objekt, d.h. die Referenz referenziert *nullptr*. Dies kann auch explizit mit

```
Person person = Person::nil();
```

geschrieben werden. Auch beim Testen, ob ein Objekt ein Nil-Objekt ist, kann der explizite Aufruf auf *Person::nil* verwendet werden.

```
if (person == Person::nil()) { /*...*/ }
```

ist hierbei gleich bedeutend wie

```
if (!person) { /*...*/ }
```

4.3.2. Leere Objekte

Um ein tatsächliches Objekt des Typen zu erzeugen muss dieses über die statische *create*-Methode des Typen

```
person = Person::create();
```

erzeugt werden. Eine andere Referenz kann nun diese Referenz kopieren

```
Person person2 = person;
```

5. Attribute

Um einem Typen ein ein- oder mehrwertiges Attribut zu geben, muss dafür der Attributname, der Typ und der Datentyp des Attributs angegeben werden

```
OT_ATTR1(name, Person, std::string)
```

bzw.

```
OT_ATTRN(tags, Person, std::string)
```

Die Attribute an sich sind als Funktionen mit dem Attributsnamen als Funktionsname implementiert. Für jede Zugriffsweise auf ein Attribut (Lesen, Schreiben, Löschen etc.) gibt es (mindestens) eine Funktion mit jeweils unterschiedlicher Parameterliste.

5.1. Einwertige Attribute

Einwertige Attribute sind die einfachste Form eines Attributes. Entweder hat das Attribut einen Wert, oder nicht.

5.1.1. Lesen

Es gibt zwei Arten ein Attribut zu lesen:

- `std::string name = person[name];`
- `std::string name = name(person);`

Der überladene *[]*-Operator des Typen, ruft dann im Endeffekt für *person[name]* wiederrum intern *name(person)* auf. Wenn das Attribut nicht gesetzt wurde, wird der Standardwert des Attributdatentyps zurückgegeben. Im diesem Fall wäre das *std::string()* oder *""*(leerer String).

5.1.2. Schreiben

Um ein Attribut zu setzen bzw. zu überschreiben:

- `person(name, std::string("Hans"));`
- `name(person, std::string("Hans"));`

Da ein string-Literal in C++ den Typ *const char** hat, muss hier zwingend die explizite Umwandlung in einen *std::string* beim Setzen des Attributs passieren. Alternativ kann auch direkt ein Wert vom Typ *std::string* übergeben werden

```
std::string value = "Peter";
person(name, value);
```

5.1.3. Löschen

- `person -= name;`
- `name(person, ot::DeleteDefault{});`

Da sich die Parameterliste für jede Funktion mit dem gleichen Funktionsnamen unterscheiden muss, das Löschen eines (einwertigen) Attributes aber eigentlich keine zusätzlichen Parameter außer dem Objekt (wie beim lesenden Zugriff) braucht, muss hier beim Löschen über den Attributnamen noch ein Objekt des Typs *ot::DeleteDefault* übergeben werden.

5.2. Mehrwertige Attribute

5.2.1. ot::vector

5.2.2. Lesen der Attributliste

5.2.3. Hinzufügen eines Attributwerts am Ende der Liste

5.2.4. Hinzufügen eines Attributwerts an einer bestimmten Position der Liste

5.2.5. Löschen der gesamten Attributliste

5.2.6. Löschen eines bestimmten Attributwerts aus der Liste

5.2.7. Löschen eines Attributwerts an einer bestimmten Position aus der Liste

6. Bidirektionale Relationen

6.1. Asymmetrische Relationen

6.1.1. 1:1-Relationen

6.1.2. 1:N-/N:1-Relationen

6.1.3. N:N-Relationen

6.2. Symmetrische Relationen

6.2.1. 1:1-Relationen

6.2.2. N:N-Relationen

7. Typen und Attribute als Schablonen

7.1. Typschablonen

7.2. Einwertige Attribute für Typschablonen

7.3. Mehrwertige Attribute für Typschablonen

8. Namensbereiche

9. Serialisierung und Deserialisierung mit cereal

9.1. cereal

9.2. Serialisierung eines Typen

9.3. Deserialisierung eines Typen

10. Evaluierung

Aufgabe des Kapitels Evaluierung ist es, in wie weit die Ziele der Arbeit erreicht wurden. Es sollen also die erreichten Arbeitsergebnisse mit den Zielen verglichen werden. Ergebnis der Evaluierung kann auch sein, das bestimmte Ziele nicht erreicht werden konnten, wobei die Ursachen hierfür auch außerhalb des Verantwortungsbereichs des Praktikanten liegen können.

11. Zusammenfassung und Ausblick

11.1. Erreichte Ergebnisse

Die Zusammenfassung dient dazu, die wesentlichen Ergebnisse des Praktikums und vor allem die entwickelte Problemlösung und den erreichten Fortschritt darzustellen. (Sie haben Ihr Ziel erreicht und dies nachgewiesen).

11.2. Ausblick

Im Ausblick werden Ideen für die Weiterentwicklung der erstellten Lösung aufgezeigt. Der Ausblick sollte daher zeigen, dass die Ergebnisse der Arbeit nicht nur für die in der Arbeit identifizierten Problemstellungen verwendbar sind, sondern darüber hinaus erweitert sowie auf andere Probleme übertragen werden können.

11.2.1. Erweiterbarkeit der Ergebnisse

11.2.2. Übertragbarkeit der Ergebnisse

A. Anhang A

B. Anhang B