Universität Kassel

Abschlussarbeit

Restrukturierung und Kopplung von C++ Programmen zur Simulation von Umweltprozessen

Student:
Bernhard Zeidler

Matrikelnummer:
31202411

Erstprüferin:
Prof. Dr. Claudia Fohry

Zweitprüfer:
Apl. Prof. Dr. Rüdiger Schaldach

08.01.2019

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendeten Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Kassel, den 08. Januar 2019

Bernhard Zeidler

Inhaltsverzeichnis

1 Einleitung

Das Center of Environmental Research, kurz CESR, erforscht unter Anderem die Auswirkungen menschlichen Handelns auf dessen Umwelt. Ein Teil dieser Forschungsarbeit befasst sich mit der Frage, wie oder wozu Menschen Landflächen nutzen. Beispiele für solche Landnutzung sind Feldanbau oder die Nutzung als Wohngebiet. Um die Veränderung der Landnutzung über einen definierbaren Zeitraum simulieren zu können wurden einige Anwendungen entwickelt. Zwei dieser Anwendungen sind LandSHIFT und das Irrigation - Modul.

LandSHIFT simuliert hierbei die tatsächliche Landnutzung sowie die zeitliche Veränderung dieser. So könnte ein Areal, welches zunächst als Waldfläche genutzt wird, zu einem späteren Zeitpunkt gerodet werden und als Weidefläche für Tierhaltung verwendet werden.

Die zweite der genannten Anwendungen, das Irrigation - Modul, berechnet Wasserentnahme und Wasserverbrauch bei Feldanbau. Der Unterschied zwischen Entnahme und Verbrauch ist, dass entnommenes Wasser teilweise von Feldern abfließen und wieder in Flüsse fließen kann und so später wieder entnommen werden kann. Tatsächlich verbrauchtes Wasser kann nicht weiter entnommen werden.

Da Feldanbau nur dort betrieben werden kann, wo auch Wasser zum gießen der Pflanzen vorhanden ist beeinflusst das Irrigation - Modul die Simulation von LandSHIFT. Offensichtlich kann ein Gebiet nur dann zum Feldanbau genutzt werden, wenn dort genügend Wasserreserven vorhanden sind. Andererseits beeinflusst die LandSHIFT Simulation die Berechnung des Irrigation - Moduls, da mehr Feldanbau eine erhöhte Wasserentnahme und einen erhöhten Wasserverbrauch bedeutet. Beide Anwendungen beeinflussen sich also gegenseitig.

Die Ein- und Ausgabe Methoden der beiden Anwendungen unterscheiden sich grundsätzlich: LandSHIFT bezieht seine Eingaben aus einer Datenbank, während das Irrigation - Modul seine Ergebnisse in Dateien schreibt. Damit LandSHIFT also benötigte Berechnungen des Irrigation - Moduls verwenden kann, müssen diese zunächst in die Datenbank geschrieben werden. Hierfür wird ein unbenanntes R - Skript verwendet. Dieses Skript hat neben dem befüllen der Datenbank allerdings noch einige weitere Aufgaben: Neben dem Irrigation - Modul gibt es einige weitere Anwendungen die ähnliche Berechnungen für andere Arten von Wassernutzung durchführen. Einige, aber nicht alle, dieser weiteren Anwendungsgebiete sind:

- Domestic - Wassernutzung in Haushalten,
- Electricity - Wassernutzung zur Stromerzeugung und

- Livestock - Wassernutzung zur Viehzucht

Auch bei diesen Anwendungsgebieten wird jeweils Entnahme und Verbrauch unterschieden. Alle Wassernutzungsarten werden von diesem Skript zusammengetragen. Zusätzlich finden noch einige weitere Berechnungen statt. Diese werden später erläutert.

Die Zielsetzung dieser Arbeit ist die Kopplung der beiden genannten Anwendungen sowie des Skriptes. Das Bedeutet, dass zunächst LandSHIFT seine Simulation durchführen soll, im Anschluss soll das Irrigation - Modul seine Berechnungen durchführen und zuletzt das R - Skript seine. Eine zusätzliche Randbedingung dieser Arbeit ist, dass die Ergebnisse dieser Arbeit zukünftig für Forschungsarbeit des Fachgebietes Programmiersprachen und Methodik verwendet werden sollen. Das bedeutet insbesondere, dass die einzelnen Anwendungen und deren Kopplung parallelisierbar sein sollen. Dementsprechend ist auch die Dokumentation der Eigenentwicklungen und Modifikationen von großer Bedeutung.

Zur Umsetzung dieser Abschlussarbeit wurde zuerst das R - Skript in Form eines C++ Programms neu geschrieben. Dieses Programm wird im Folgenden AlternativeWatergap genannt. Nach dieser Konvertierung wurde das Irrigation - Modul so modifiziert, dass es in eine weitere Anwendung eingebettet werden kann und von dort aus aufgerufen beziehungsweise gesteuert werden kann. Für die Kopplung der Anwendungen ist ein gewisser Datenfluss notwendig: Für den Feldanbau genutzte Landfläche muss vom Irrigation - Modul berücksichtigt werden, während die Ergebnisse des Irrigation - Moduls in die Berechnungen von AlternativeWatergap eingehen müssen. Für diesen Datenfluss wurde eine entsprechende Bibliothek entwickelt, welche nun von allen Anwendungen verwendet wird. Für die Kopplung selbst wurde ein weiteres Programm entwickelt, welches alle anderen Anwendungen initialisiert und nacheinander ausführt. Hierfür wurden kleinere Modifikationen an AlternativeWatergap durchgeführt, welche vor Vollendung des Datenaustauschs mit dem Irrigation - Modul noch nicht möglich waren. Die Kopplung von LandSHIFT ergibt sich schlicht aus dem Aufruf dessen.

Im Anschluss an diese Einleitung werden zunächst die gegebenen Anwendungen sowie die Funktionsweise des R - Skriptes näher erläutert. Hierbei werden ebenfalls Eingaben sowie die Ausführung der Anwendungen beschrieben.

Im folgenden, dritten, Kapitel wird die Entwicklung von AlternativeWatergap beschrieben. Das Kapitel enthält außerdem eine Beschreibung einer entwickelten Bibliothek, LibIO, sowie einer zusätzlichen Anwendung. Diese extra Anwendung, OutletCellConverter genannt, berechnet eingaben, welche von AlternativeWatergap benötigt werden.

Das vierte Kapitel beschreibt die Kopplung aller Anwendungen. Hierbei werden die benötigten Modifikationen an allen gegebenen Anwendungen beschrieben sowie die Bibliothek zum Datenaustausch zwischen den Anwendungen. Die Anwendung, welche die Kopplung aller Anwendungen darstellt wurde „LandIrrigationGap“ genannt.

Da diese Arbeit in Zukunft fortgeführt werden soll, beschreibt das fünfte Kapitel einige Ansätze dafür. An dieser Stelle werden einige offene Punkte sowie mögliche Lösungsansätze für diese diskutiert.

Für eine mögliche Fortführung dieser Arbeit ist ein erfolgreiches Erstellen der Anwendungen zwangsweise erforderlich. Im sechsten Kapitel wird das verwendete Build Tool, CMake, kurz eingeführt. Dies enthält neben einigen wichtigen Befehlen zur Steuerung des Tools ebenfalls die Verwendung dessen.

Im siebten Kapitel werden einige technische Details, wie Dateiformate und Ausgaben, näher diskutiert. Dies umfasst sämtliche Dateiformate aller Anwendungen. Da die dort erläuterten UNF Dateien in vielen anderen Kapiteln referenziert werden sollte der entsprechende Abschnitt eventuell vorab gelesen werden.

Kapitel Acht beschreibt einige Ressourcen, welche für diese Arbeit von Relevanz sind. Gemeint ist damit das verwendete GIT - Repository sowie einige Verzeichnisse und Dateien die auf den Servern des Fachgebietes Programmiersprachen und Methodik zu finden sind.

Das neunte Kapitel beschreibt den Versuch die von LandSHIFT verwendete Datenbankschnittstelle zu modernisieren. Diese Modernisierung wurde letzten Endes allerdings verworfen, da hierdurch Probleme entstanden, deren Lösung wesentlich mehr Zeit in Anspruch genommen hätte.

Den Abschluss dieser Arbeit bildet eine Zusammenfassung in Kapitel zehn.

2 Anwendungsübersicht

2.1 LandSHIFT

Wie Eingangs bereits erwähnt berechnet LandSHIFT mögliche Veränderungen in der Landnutzung durch den Menschen über einen definierbaren Zeitraum. Diese Simulation basiert auf einer Einteilung des Globus in einzelne Zellen, welche eine Breite und Höhe von fünf Bogenminuten am Äquator besitzen. Das bedeutet, dass eine Zelle in Äquatornähe eine annähernd quadratische Form besitzt. Zellen, die nördlich oder südlich des Äquators liegen verjüngen sich, was der eher sphärischen Form des Planeten geschuldet ist.

Die Anwendung besitzt einen interaktiven Modus, welcher allerdings nicht für diese Anwendung relevant ist. Zusätzlich kann LandSHIFT mit einer grafischen Oberfläche verwendet werden. Entsprechende Stellen im Quelltext sind allerdings in der zur verfügung gestellten Version auskommentiert. Die durch die GUI entstehenden OpenGL Abhängigkeiten, namentlich freeGLUT, entfallen somit.

2.1.1 Eingaben und Ausführung

Die einzigen Eingaben, die LandSHIFT benötigt, sind eine Konfigurationsdatei, sowie einige Kommandozeilenargumente. Alle weiteren benötigten Daten werden zur Laufzeit aus einer Datenbank abgefragt. Die verwendete Datenbank ist eine inzwischen relativ alte Version von MySql.

Die Konfigurationsdatei befindet sich in einem „INPUT“ Verzeichnis. Der INPUT-Ordner selbst befindet sich im selben Verzeichnis wie auch die ausführbare Datei. Der genaue Pfad der Datei ist allerdings irrelevant, da dieser als Kommandozeilenargument übergeben wird.

Die Kommandozeilenargumente wurden von Herrn Wimmer wie folgt beschrieben: Zum Starten von LandSHIFT:

```
./Landhsift_Irrigation -w
    -c /Pfad/Zu/Eingabe/config.cfg
    -o /Pfad/Zu/Ausgabe -S SSP2-NoCC
    --aez -l 1
```

Die Argumente bedeuten hierbei im Detail:

- w : Ausgaben schreiben
- c : Pfad zur Konfigurationsdatei
- o : Ausgabepfad

- S : Szenario Bezeichner
- aez : Ein Flag: Modell nutzt Agro Ecological Zones bei der Eignungsbewertung
- l : Ausgabestufe, 1 = minimal, 2 = zusätzliche Ausgaben, 3 = sehr umfangreiche Ausgaben

Weitere Details zu Eingaben, Ausgaben und den Datenbanken werden im Kapitel 7 beschrieben.

2.2 Irrigation-Modul

In der Einleitung wurde bereits erwähnt, dass das Irrigation - Modul die Wasserverwendung für Feldanbau berechnet. Im Gegensatz zu LandSHIFT besitzt das Irrigation - Modul keine alternativen Ausführmodi. Es gibt also keine interaktive oder grafische Option. Dementsprechend gibt es auch keine solchen Abhängigkeiten in externe Bibliotheken. Ebenfalls unterschiedlich ist die Datenherkunft: LandSHIFT bezieht praktisch alle Daten aus einer Datenbank, während das Irrigation - Modul sämtliche Daten aus Dateien einliest.

Wie auch LandSHIFT arbeitet das Irrigation - Modul zellenweise, allerdings besitzen Zellen im Irrigation - Modul eine andere Identifikation, als in LandSHIFT.

2.2.1 Eingaben und Ausführung

Das Irrigation - Modul benötigt eine Vielzahl an Daten. Zur Berechnung des tatsächlichen Wasserverbrauches muss bekannt sein, welche Größe die zu bewässernden Landflächen haben. Zusätzlich muss bekannt sein, was in einer gegebenen Fläche überhaupt angebaut wird und wie groß der Wasserverbrauch der angebauten Feldfrüchte ist. Zusätzlich zum Verbrauch der Feldfrüchte wird einiges Wasser schlicht abfließen, was wiederum die Menge des zu entnehmenden Wassers erhöht. Klimadaten, aus denen sich Wasserverdunstung oder die Menge vorhandenen Regenwassers ergibt werden ebenso berücksichtigt. Zusätzlich zu den genannten Daten werden noch eine Vielzahl weiterer benötigt.

Verzeichnisse, in denen nach Eingabedateien gesucht wird, werden in einer Datei aufgelistet. In den gegebenen Materialien heißt diese Datei „DATA_eu.DIR“ für Berechnungen in Europa. Das Kontinentalkürzel „eu“ kann für andere Kontinente, wie Australiens „au“, entsprechend geändert werden. Zusätzlich gibt diese Datei das Ausgabeverzeichnis an. Der Pfad zu dieser

DIR Datei stellt auch eine einzigen Kommandozeilenargumente für das Irrigation - Modul dar. Ein kompletter Programmaufruf hätte die Form:

```
./wg_irrigation32 D -d/Pfad/zu/DIR
```

Wobei

- D: Eine Ziffer die den Kontinent identifiziert
  1. Europa
  2. Afrika
  3. Asien
  4. Australien
  5. Nord Amerika
  6. Süd Amerika
- d: Pfad zu der dem Kontinent entsprechenden DIR Datei.

Zu beachten ist hierbei, dass „D“ durch die Ziffer ersetzt wird und dass es kein Leerzeichen zwischen „d“ und dem Pfad gibt.

2.3 R-Script

Zuvor wurde bereits erwähnt, dass das Irrigation Modul Wasserentnahme und dessen Verbrauch berechnet. Diese Berechnungen finden Zellenweise statt. Tatsächlich können einzelne Zellen aber nicht vollkommen getrennt betrachtet werden. Sie müssen viel mehr in einem Verbund betrachtet werden, da ein Fluss nur eine bestimmte Menge Wasser führt und insgesamt nicht mehr Wasser entnommen werden kann, als real zur Verfügung steht.

Da offensichtlich nicht alle Zellen aus den selben Quellen bewässert werden können, werden alle Zellen auf unabhängige Wassereinzugsgebiete verteilt. Wassereinzugsgebiete sind genau dann unabhängig, wenn eine beliebige Menge Wasser unmöglich, ohne äußere Einwirkungen, in ein anderes Wassereinzugsgebiet migrieren kann. Abhängige Einzugsgebiete werden als ein einziges Gesamteinzugsgebiet betrachtet. Wassereinzugsgebiete werden im Folgenden auch „Basin“ genannt. Diese Basins werden zur Berechnung in Sub - Basins unterteilt, falls ein Basin zu groß ist. Die maximale Größe von Sub - Basins beträgt $20'000km^2$. Diese Unterteilung erfolgt in fließrichtung der im Basin enthaltenen Gewässer. Die Aufgabe dieses Scriptes ist es, die zellenweise zur Verfügung stehenden Ergebnisse des Irrigation - Moduls, sowie weitere Eingaben, basinweise zusammenzutragen.

Da das originale Script weder zur Verfügung steht noch benötigt wird, kann an dieser Stelle die Nutzung jenes Skriptes nicht erläutert werden.

3 Konvertierung des R - Skriptes zu C++

Die Konvertierung des Skriptes hat, inklusive Fehlerbehebungen und der Entwicklung zweier weiterer Anwendungen, den größten zeitlichen Anteil dieser Abschlussarbeit vereinnahmt. Die beiden zusätzlichen Anwendung, sind einerseits ein Konvertierer, der benötigte Eingaben produziert, sowie eine Testanwendung zur Fehleranalyse. Zusätzlich zu diesen beiden Anwendungen wurde eine Bibliothek entwickelt, welche von dem erwähnten Konverter, AlternativeWatergap sowie dem später detaillierten LandIrrigationGap verwendet wird.

3.1 LibIO

Diese Bibliothek war zu Beginn fester Bestandteil von AlternativeWatergap und wurde erst zu einem späteren Entwicklungszeitpunkt extrahiert. Wie der Name bereits andeutet sind in dieser Bibliothek Ein- und Ausgabefunktionen gekapselt. Zusätzlich dazu sind hier allerdings auch einige weitere Strukturen sowie eine Hilfsfunktion enthalten.

Das Kernstück der Bibliothek bildet der Namespace IO, deklariert in der Datei „fileReader.h“. Dieser Namespace bietet einige Funktionen zum Einlesen von Dateien sowie eine Möglichkeit zum Bestimmen der Dateigröße von UNF Dateien. Für weitere Details zu diesem Dateiformat wird auf den Abschnitt 7.2.2 verwiesen. Der genannte Namespace wird im folgenden Unterkapitel näher beschrieben.

Der „helper“ - Namespace wird in der gleichnamigen Headerdatei deklariert. Dieser Namespace umfasst lediglich eine „split“ - Funktion. Die genaue Signatur der Funktion lautet:

```
std::vector<std::string>split(std::string s
    , const char del = ':')
```

Die Aufgabe dieser Funktion ist es, einen ihr übergebenen std::string in kleinere Teile zu spalten. Diese Teilstrings werden in Form eines std::vector zurückgegeben. Zum zerlegen des Strings wird dieser Funktion ein Trennzeichen mitgeteilt, an dem der String zerteilt wird. Das Trennzeichen selbst wird beim Teilungsprozess in keinem Teilstring aufgenommen.

Ein Beispiel zur Anwendung: Der String „abc:def“ soll zerteilt werden. Das Trennzeichen ist „:“. Die Rückgabe ist ein Vector der Form {„abc“, „def“}.

Die Datei „singleton.h“ stellt genau das dar: Eine Implementierung des Singleton - Entwicklungsmusters. Hierbei wird angenommen, dass dieses Entwicklungsmuster wohlbekannt ist. Aus diesem Grund wird an dieser Stelle

keine nähere Beschreibung dieses Musters geboten.

Der letzte zu erwähnende Teil der Bibliothek ist die struct „YearConfig“. Diese Struktur stellt eingelesene Daten aus einer Eingabedatei dar. Die Eingabedatei beschreibt Jahreszahlen, die von AlternativeWatergap zur Berechnung berücksichtigt werden sollen. So ist beispielsweise das Startjahr der Berechnung enthalten oder die Zeitspanne in der Wasserverfügbarkeit als Eingabe berücksichtigt werden soll.

3.1.1 Namespace IO

An dieser Stelle werden kurz die Funktionen des Namespaces erläutert. Vorab allerdings noch eine Anmerkung: Es gibt einen statischen bool `g_swapBytes`. Mit diesem Schalter wird das Verhalten der später beschriebenen „get“ Funktion gesteuert: Falls dieser Wert auf TRUE gesetzt ist, wird die Byte - Reihenfolge vor der Rückgabe des Ergebnisses vertauscht. So kann zwischen Little- und Bigendian Darstellung getauscht werden.

```
template<typename T>
void swapBytes(T & var)
```

Diese Funktion führt die eingangs erwähnte Vertauschung der Byte - Reihenfolge durch. Die Vertauschung wird innerhalb der übergebenen Eingabe - Referen Referenzz durchgeführt.

```
template<typename T>
std::size_t fileSize(const std::string & path)
```

Öffnet eine UNF Datei, siehe Abschnitt 7.2.2, und gibt die Anzahl der darin enthaltenen Werte zurück. Der Template - Parameter T gibt den Datentyp der in der Enthaltenen Werte an.

```
template <typename T>
inline T get(std::ifstream & is)
```

Liest den nächsten Wert von Typ T aus dem gegebenen Stream und gibt diesen zurück. Die Annahme ist hierbei, dass der Stream ausschließlich serialisierte Daten enthält.

```
template<>
inline float get(std::ifstream & is)
```

Überladung der get Funktion. Dies ist notwendig, da die reguläre Template - Implementierung einen Bitshift - Operator verwendet. Dieser Operator ist für Fließkommazahlen nicht implementiert.

```
template <typename T>
T getPosition(std::ifstream & is, std::size_t position)
```

Liest einen Wert von Typ T aus dem gegebenen Stream. Der gelesene Wert befindet sich im Stream „position - viele“ Werte nach dem ersten Wert. Die Zählweise beginnt bei Null. Soll also der erste Eintrag gelesen werden muss position = 0 übergeben werden. Intern wird die get - Funktion zum lesen verwendet.

```
template<typename T>
T * getMultiple(std::ifstream & is, size_t count)
```

Liest „count - viele“ aufeinander folgende Einträge aus dem Stream. Die Rückgabe erfolgt als Array von Typ T. Verwendet „get“ in einer Schleife.

```
typedef std::vector<size_t> PositionVector;
```

Typ Definition zur Lesbarkeit. Ausschließlich von „ getMultiplePositions“ verwendet.

```
template <typename T>
T * getMultiplePositions(std::ifstream & is
        , const PositionVector & positions
        , size_t count = 1)
```

Inhaltlich eine Kombination aus „getMultiple“ und „getPosition“. Liest mehrere Werte von unterschiedlichen Positionen aus dem Stream. Positionen müssen nicht aufeinanderfolgend sein. Zu beachten ist hierbei, dass der „Lesekopf“ während der Ausführung dieser Funktion durch den Stream springen kann. Die Position des Lesekopfes ist nach der Ausführung der Funktion hinter dem zuletzt gelesenen Eintrag.

```
template<typename T>
T * getAll(std::ifstream & is, size_t numElems)
```

Liest „numElems - viele“ Einträge aus dem Stream. Verwendet eine andere Implementierung als „getMultiple“. Hier wird ein Speicherbereich der Größe sizeof(T) * numElems eingelesen und reinterpretiert.

```
inline bool fileExists(const std::string & path)
```

Versucht eine Datei zu öffnen. Gibt TRUE zurück, falls der Stream geöffnet werden konnte.

```
bool getCsvLine(std::istream & is, int32_t & lhs
    , int32_t & rhs);
```

Liest eine Zeile aus einer CSV - Datei ein. Diese Zeile muss dabei die Form „Zahl“, „Zahl“ - ohne Leerzeichen - haben. Die erste gelesene Zahl wird in lhs,

die zweite in rhs zurückgegeben. Falls der Lesevorgang erfolgreich war gibt diese Funktion TRUE zurück, sonst FALSE. Intern wird hier die Hilfsfunktion „split“ verwendet.

```
YearConfig readYearConfig(std::istream & is
    , uint16_t lineNumber);
```

Liest eine Zeile aus der Jahreskonfigurationsdatei aus dem gegebenen Stream. Der Parameter lineNumber identifiziert die zu lesende Zeile. Die Zählweise beginnt bei Null wobei die Kopfzeile ignoriert wird. Falls lineNumber = 0 übergeben wird wird also die erste Zeile nach der Kopfzeile gelesen.

3.2 OutletCellConverter

Wie der Name des OutletCellConverters nahe legt, ist es die Aufgabe dieser Anwendungen Daten zu Konvertieren. Diese konvertierten Daten werden dann als Eingabe für AlternativeWatergap verwendet. Da sich die Daten nicht verändern genügt ein einmaliges Ausführen und schreiben dieser Daten.

Im Folgenden Abschnitt wird zwischen zwei Arten von Wasserverfügbarkeit unterschieden. Um die beiden Arten von Wasserverfügbarkeit besser unterscheiden zu können sollen vorab die Begriffe „Grundverfügbarkeit“ und „Restverfügbarkeit“ eingeführt werden. Wenn die Rede von Wasserverfügbarkeit vor allen Abzügen - also vor Verbrauch - ist, wird dies Grundverfügbarkeit genannt. Restverfügbarkeit ist dementsprechend die Wasserverfügbarkeit nach allen Abzügen.

Zur genauen Definition der vom OutletCellConverter produzierten Daten muss zunächst ein gewisser Hintergrund geschaffen werden: AlternativeWatergap berechnet die Restverfügbarkeit von Wasser in einem Basin. Ausgenommen von diesen Abzügen ist der Verbrauch durch Feldanbau, da das Ergebnis darstellen soll, welche Menge Wasser für den Feldanbau zur Verfügung steht. Zu dieser Berechnung muss die Grundverfügbarkeit von Wasser bekannt sein. Diese Grundverfügbarkeit wird in Form von UNF Dateien gegeben. Genauere Details zu UNF Dateien sind im entsprechenden Abschnitt 7.2.2 zu finden. Vorab sei erwähnt dass diese Dateien jeweils einen Wert für jede Zelle enthalten. Diese Werte sind immer in einer bestimmten Reihenfolge geschrieben. Die Grundverfügbarkeit an Wasser für jeweils ein Basin ist an jeweils einer Stelle einer UNF Datei enthalten, wobei eine UNF Datei alle Basins eines Kontinentes zu einem gegebenen Zeitpunkt beschreibt.. Zum extrahieren der Grundverfügbarkeit eines Basins muss also genau eine Zahl aus einer Datei gelesen werden, wobei diese Zahl an einer fixen Position steht. Die Zelle, die an dieser Position dargestellt wird, wird „OutletCell“ genannt. Der OutletCellConverter sucht nach diesen Positionen und schreibt eine Zu-

weisung, wobei jedem Basin eine OutletCell - Position zugewiesen wird, in eine Datei.

Zum Finden der OutletCell - Position wird weiteres Hintergrundwissen genutzt: Basins werden anhand von Identifikationsnummern unterschieden. Diese Nummern besitzen jeweils eine führende Ziffer, welche den Kontinent darstellt. Eine führende Eins bedeutet beispielsweise „Europa“. Im Anschluss folgen sechs Ziffern zur Identifikation des Basins. Die eigentliche ID eines jeden Basins ist also nur sechsstellig. WaterGAP, eine weitere, nicht näher erläuterte Anwendung, verwendet ebenfalls Zahlen zur Identifikation von Zellen. Hier ist die Zählweise allerdings unterschiedlich. Relevant ist hier nur die Tatsache, dass die ID einer WaterGAP Zelle genau dann gleich der Basin ID ist, wenn es sich bei der Zelle um die OutletCell handelt. Beide Daten sind in Form von UNF Dateien gegeben, die jeweils Zellen in der gleichen Reihenfolge darstellen.

Auf dieser Erkenntnis beruht die Funktionsweise des OutletCellConverters: Prüfe für jeden Eintrag der WaterGAP ID Datei, ob der gelesene Wert gleich einer Basin ID ist. Falls ja, Weise der Basin ID die Position des gelesenen Eintrags zu.

3.3 AlternativeWatergap

In der Einleitung wurde erwähnt, dass AlternativeWatergap, beziehungsweise das Skript auf dem es beruht, Ergebnisse des Irrigation - Moduls zusammenträgt. Für eine grobe Umschreibung ist dies auch korrekt. Im Detail betrachtet ist die Aufgabe allerdings komplexer. Zum Verständnis des vorherigen Abschnitts musste ein Teil der Aufgabe AlternativeWatergaps vorab detailliert werden: AlternativeWatergap berechnet die Restverfügbarkeit von Wasser nachdem jeder Verbrauch von der Grundverfügbarkeit abgezogen wurde. Von diesem Abzug wird die Menge Wasser ausgeschlossen, die für den Feldanbau verbraucht wird. Dieses Verhalten ist allerdings nur innerhalb eines Sub - Basins korrekt. Zwangsweise kann in einem Sub - Basin nur die Menge Wasser genutzt werden die lokal zur Verfügung steht. Hierbei muss auch berücksichtigt werden, dass Wasser, welches in einem flussaufwärts gelegenen Sub - Basin verbraucht wurde nicht mehr zur Verfügung stehen kann. Dies gilt insbesondere auch für den Verbrauch durch Feldarbeit.

Die gesamte Funktionsweise von AlternativeWatergap kann in wenigen Worten zusammengefasst werden:

1. Erzeuge eine interne Darstellung der Sub - Basins

2. Lese und verteile Eingaben zu benötigten Daten.

3. Sobald ein Basin komplett berechnet wurde: produziere eine Ausgabe

Diese Einzelpunkte werden in den folgenden Abschnitten detaillierter betrachtet.

3.3.1 Darstellung und Sortierung von Daten

Die zur Berechnung benötigten Daten sind:

- Grundwasserverfügbarkeit
- Unterschiedliche Arten von Wasserentnahme und Verbrauch
- Verbrauch und Entnahme von Wasser in flussaufwärts gelegenen Sub - Basins.
- Flächengröße des Sub - Basins
- Restwasserverfügbarkeit

Zusätzlich zu diesen notwendigen Werten gibt es einen weiteren Dateneintrag, welcher eingelesen und später geschrieben wird: Die „Rout Area“. Diese Fläche hat keine rechnerische Relevanz und wird nur der Vollständigkeit wegen erwähnt.

Abgesehen von diesen Werten muss auch die Abhängigkeit oder Beziehung zwischen Sub - Basins dargestellt werden. In Worte gefasst könnte diese Beziehung wie folgt beschrieben werden: Wie bereits erwähnt wird ein zu großes Basin in kleinere Sub - Basins geteilt. Diese Unterteilung findet in Flussrichtung statt. Dementsprechend geht ein Sub - Basin in ein anderes über. Tatsächlich geht ein Sub - Basin in genau ein Sub - Basin über, nie in mehrere. Bisher wäre eine verkettete Liste also eine realistische Darstellung dieser Abhängigkeit. Es kann allerdings vorkommen das mehrere Sub - Basins in ein einzelnes Sub - Basin übergehen. So gesehen ist ein gesamt Basin, welches in Sub - Basins geteilt wurde, als unausgeglichener Baum darstellbar, wobei ein Knoten mehrere Nachfolger haben kann. Bei dieser Betrachtung sind die flussaufwärts gelegenen Sub - Basins die Nachfolger. Wenn man hier noch einen theoretischen Wurzelknoten hinzunimmt, können alle Basins eines Kontinents in einem Baum dargestellt werden. Bildlich gesprochen könnte diese Wurzel die Summe aller Weltmeere darstellen.

Da der Charakter der bisher erwähnten darzustellenden Informationen grundsätzlich verschieden ist, wurde die interne Darstellung ebenso getrennt. Es gibt einerseits die struct „Basin“ welche schlicht alle zu Beginn dieses Abschnitts erwähnten Werte eines Basins enthält, andererseits gibt es die

Klasse „RelationNode“ welche die baumartige Struktur darstellt. Ein Objekt des Typs RelationNode verweist dabei auf ein Objekt des Typs Basin. Ein Basin ist hierbei tatsächlich nur eine Sammelklasse für Werte, wobei ein Basin auch sicherstellt, dass hinzuzufügende Werte korrekt skaliert sind. Die Überwachung der Skalierung ist deshalb notwendig, da einige Werte in unterschiedlichen Einheiten zur Verfügung stehen. Beispielsweise werden Wasserentnahmen und -verbrauch in Millionen Kubikmetern gegeben, während Grundwasserverfügbarkeit in Kubikkilometern gegeben ist.

Eine weitere, bisher unerwähnte, benötigte Information zu einem Sub - Basin ist die der Zellen, welche zu diesem gehören. Diese Information wird in Form einer Liste im entsprechenden RelationNode gespeichert. Der Grund weshalb diese Information hier abgelegt wird und nicht im Basin direkt ist, dass die RelationNode Klasse im Verlauf der Arbeit inhaltlich gewachsen ist. Zunächst stellte die Klasse tatsächlich nur die Beziehung zwischen Sub - Basins dar und bot Funktionen zum Zugriff auf diese. Später wurde diese Klasse inhaltlich so erweitert, dass sie fixe Informationen hält, welche sich während eines gesamten Programmablaufs nicht verändern werden. Dazu zählen auch Zellen, die zusammen ein Basin bilden. Für diese Zellen werden jeweils nicht die IDs gespeichert, sondern die ebenfalls eindeutigen Positionen innerhalb der UNF Dateien. Da ein RelationNode die Positionen der zu lesenden Einträge innerhalb einer Eingabedatei kennt, bietet diese Klasse ebenfalls Funktionen zum lesen der benötigten Werte.

Der letzte offene Punkt in diesem Abschnitt ist die Sortierung der RelationNodes in die verwendete Baumstruktur. Die Abhängigkeiten zwischen Sub - Basins ist in Form einer CSV Datei gegeben. In dieser Datei ist das flussabwärts liegende Basin als „lower“ bezeichnet, während das flussaufwärts liegende „upper“ genannt wird. Diese Datei enthält transitive Abhängigkeiten. Ein Beispiel: Das Basin A besitzt das direkte upper Bassin B, welches wiederum die beiden upper Basins C und D hat. In der Datei werden für das Basin A alle drei Basins - B, C und D - als upper Basin geführt. Die Baumstruktur verlangt allerdings eine Auflösung dieser transitiven Abhängigkeit.

Zur Sortierung der Knoten wird zunächst die gesamte CSV Datei eingelesen. Dabei wird für jedes in der Datei genannte Basin eine Liste gebildet. Diese Liste stellt, entgegen der Darstellung innerhalb der Datei, allerdings sämtliche lower Basins dar. Nach dieser Vorarbeit gestaltet sich die Sortierung trivial: Suche das Basin X mit der längsten Liste an lower Basins. Dieses Basin muss zwangsweise am weitesten flussaufwärts liegen. Aus der Liste der lower Basins von X: Suche das Basin Y mit der längsten Liste an Vorgängern. X muss upper Basin von Y sein. Dies wird wiederholt bis alle Knoten sortiert sind.

Zur Verdeutlichung ein Sortierschritt zu dem vorher erwähnten Baum

aus den Knoten A, B, C und D: Die beiden Knoten C und D haben gleich lange Listen an Vorgängern: {A, B}. Für den Knoten C wird nun die Liste seiner Vorgänger nach dem Knoten mit den meisten Vorgängern durchsucht. Der Knoten A hat keine Vorgänger, Knoten B hat einen. Knoten B muss Vorgänger von C sein. Darauf folgend würde ein Sortierschritt für den Knoten D ausgelöst werden, da dieser nun die längste Liste Vorgänger hat, dann für B.

3.3.2 Datei Management und Settings

Alle von AlternativeWatergap benötigten Daten können aus Dateien gelesen werden. Für die Verwaltung der einzulesenden Dateien wurde ein in zwei Teile getrennter Mechanismus implementiert: Eine „externe Verwaltung“ bei der ein Nutzer die zu lesenden Dateien und deren Pfade spezifiziert, sowie eine „interne Verwaltung“ zur Laufzeit.

Die externe Verwaltung wird vom Nutzer in Form einer Konfigurationsdatei durchgeführt. Hierbei werden in dieser Datei Pfade zu Dateien oder Verzeichnissen angegeben sowie Dateinamen oder Muster aus denen vollständige Namen generiert werden. Die Konfigurationsdatei selbst ist kommentiert, sodass beim editieren oder lesen dieser eindeutig identifizierbar ist welche Eingabe erwartet wird. Die Konfigurationsdatei folgt dabei immer dem Muster „Schlüssel“:„Wert“. Prinzipiell ist die Struktur der einzelnen Einträge wie folgt aufgebaut: Die meisten Werte sind in zwei Teile getrennt: Den Pfad zum Verzeichnis sowie den Dateinamen. Ausnahmen hiervon sind solche Dateipfade, welche sich selten oder nie ändern sollten. Bei diesen Dateien wird der vollständige Pfad erwartet.

Zur erhöhten Lesbarkeit der Datei wurde darauf verzichtet alle Dateinamen aller benötigten Eingaben einzeln aufzulisten: Einige der Dateien stellen die selben Daten zu unterschiedlichen Zeitpunkten dar. Dementsprechend würde bei einer Auflistung aller Namen der selbe Name mehrfach vorkommen, wobei sich die Einträge nur bei einer Zahl unterscheiden. Bei diesen Dateien wird jeweils ein Muster erwartet, aus dem zur Laufzeit der vollständige Name generiert wird. Das Muster ist hierbei ein Format - String, wie er auch von der Funktion „printf“ erwartet wird. In der gegebenen Fassung der Konfigurationsdatei wird „%i“ als Platzhalter für Jahreszahlen verwendet.

Die Schlüssel werden im Quelltext benötigt und sollten nicht verändert werden. Falls sie dennoch modifiziert werden müssen, müssen sie sowohl in der Konfigurationsdatei als auch in der Datei „settingKeys.h“ angepasst werden. Dieser Header stellt dem Programm alle Schlüsselwerte als Konstanten zur Verfügung.

Die Konfigurationsdatei wird von AlternativeWatergap eingelesen und

zur Laufzeit im speicher gehalten. Zur Laufzeitverwaltung dieser externen Eingaben wurde eine Settings - Klasse entwickelt. Diese Klasse stellt einen Singleton dar. Einträge können über die selben Schlüssel abgefragt werden, wie sie auch in der Konfigurationsdatei enthalten sind, beziehungsweise über die entsprechenden Konstanten.

Zur Laufzeit werden die einzulesenden Dateien, beziehungsweise Datei - Streams, von einem FileManager verwaltet. Hierbei handelt es sich um die eingangs erwähnte „interne Verwaltung“. Eine Referenz auf einen solchen Stream kann hierbei über einen Identifikator eingeholt werden. Der Identifikator ist als typedef im FileManager definiert:

```
typedef std::pair<uint16_t, FileType> FileIdent;
```

Der unsigned Integer stellt hierbei eine Jahreszahl dar, während „FileType“ eine Enumeration ist, welche die unterschiedlichen Datenarten darstellt. Datenarten können unter Anderem DOMESTIC, BASIN_RELATION oder OUTLET_CELL sein. Für Datenarten, bei denen zwischen Wasserentnahme und -verbrauch unterschieden wird, werden jeweils zwei Streams geöffnet.

3.3.3 Ausgabemechanismus

Sobald ein Sub - Basin vollständig berechnet ist, werden dessen Werte geschrieben. Aktuell wird hierfür eine CSV - Datei geschrieben. Mit der Absicht unterschiedliche Ausgabevarianten möglichst einfach einbauen zu können wurde der Ausgabemechanismus in zwei Teile geteilt: Dispatcher und Publisher. Der Dispatcher stellt hierbei ein konfigurierbares Interface dar. Sobald ein Basin abgeschlossen ist, wird es an den Dispatcher übergeben. Entsprechend der eigenen Konfiguration löst der Dispatcher dann das Schreiben des Basins aus. Der Dispatcher ist ebenfalls ein Singleton.

Zum Schreiben eines Basins wird dann ein Publisher verwendet. Ein spezieller Publisher ist hierbei eine Implementierung der abstrakten Publisher - Klasse. Die Schnittstelle besteht aus einer einzigen virtuellen Funktion. Die Implementierung dieser Funktion regelt dann das genaue Verhalten des speziellen Publishers.

Der Vorteil dieser geteilten Implementierung ist, dass der Aufruf zur Ausgabe immer gleich bleibt, unabhängig von der Ausgabe selbst. Zur Veränderung der Ausgabe muss lediglich ein neuer Publisher implementiert werden. Im Anschluss muss dieser noch dem Dispatcher mitgeteilt werden.

3.3.4 Unterschiedliche Ausführmodi

Während der Implementierung von AlternativeWatergap haben sich einige unterschiedliche Ausführungsszenarien ergeben: Einerseits kann es sinnvoll

sein während der Ausführung die baumartige Struktur der Sub - Basins auszunutzen. Demzufolge wäre eine Bearbeitungsreihenfolge der Sub - Basins von den Blättern zur Wurzel sinnvoll. Hierfür müssten Eingabedateien so gelesen werden, dass Daten von bestimmten Positionen gelesen werden. Eine zusätzliche Ausführvariante wäre eine, bei der die Eingabedateien schlicht von Anfang bis ende eingelesen werden, ohne Rücksicht auf die Baumstruktur. Die dritte Variante, im späteren Verlauf für die Kopplung benötigt, wäre ein Ablauf bei dem Daten pro Land eintreffen. Alle genannten Varianten werden prinzipiell unterstützt und können verwendet werden. Hierfür müssen, unter Umständen, kleinere Änderungen am aktuellen Quelltext vorgenommen werden: Der aktuelle Ausführmodus von AlternativeWatergap ist Zellenweise, entsprechend der Dateireihenfolge. Dies entspricht der zweiten genannten Variante. Zur Nutzung der Baumstruktur müssen die aktuell in der Main - Funktion auskommentierten Zeilen „ein kommentiert“ werden. Zusätzlich muss Zeile 78 auskommentiert werden. Dadurch wird die Bedingung der while - Schleife durch eine andere ersetzt. Wurden alle Änderungen korrekt vorgenommen wäre der Programmablauf der Folgende:

1. Erzeuge eine Liste von fertig berechenbaren Knoten.
2. Bearbeite einen berechenbaren Knoten
3. Falls durch vollständige Abarbeitung des Knotens ein weiterer Knoten berechenbar wird: Füge diesen in die Liste der berechenbaren Knoten ein.
4. Falls die Liste nicht leer ist: gehe zu 2

Hierbei ist ein Knoten genau dann berechenbar, wenn es keine upper - Basins gibt, oder alle upper - Basin berechnet wurden.

Die Länderweise Programmvariante wird aktuell in der später erläuterten Anwendung LandIrrigationGap verwendet und kann dort eingesehen werden.

4 Kopplung der Anwendungen

Das Ziel dieser Arbeit ist eine Anwendung zu entwickeln, welche LandSHIFT, das Irrigation - Modul sowie AlternativeWatergap vereint. Diese gesamt - Anwendung wird im Folgenden LandIrrigationGap genannt, was schlicht eine Kombination der Einzelnamen ist.

Zu Beginn dieses Kapitels wird zunächst der Ansatz umrissen, nach dem die Anwendungen kombiniert wurden:

Die einzelnen Anwendungen selbst wurden so wenig wie möglich modifiziert. Die Kopplung stellt prinzipiell nur eine „Master - Anwendung“ dar, welche die einzelnen Anwendungen jeweils initialisiert und ausführt. Hierfür wurden aus den Anwendungen Bibliotheken generiert, welche dann Abhängigkeiten von LandIrrigationGap wurden.

4.1 Modifikationen an LandSHIFT

Die Anbindung von LandSHIFT gestaltete sich insgesamt relativ simpel. Es war lediglich notwendig den Start des Algorithmus in eine separate Funktion zu extrahieren. Vorher wurde der Algorithmus am Ende des Konstruktors ausgelöst. Dem ursprünglichen Programmverhalten entsprechend berechnet diese Funktion entweder einen einzelnen Zeitschritt, oder alle Zeitschritte. Dieses Verhalten wird durch Simulationsparameter gesteuert. Der Aufruf von LandSHIFT in der gekoppelten Anwendung war in erster Näherung dann eine Kopie der LandSHIFT Main - Funktion.

Da die Ausführung von LandIrrigtionGap aber weder Global noch Kontinental sein soll musste die Ausführung von LandSHIFT auf eine länderweise beschränkt werden. Der gesamte Programmablauf iteriert dann über alle Länder und ruft für jedes Land eine Ausführung von LandSHIFT auf. Die länderweise Ausführung wird von LandSHIFT bereits unterstützt und musste entsprechend nicht neu eingebaut werden. Die Auswahl der Länder, für die LandSHIFT gestartet werden soll wird anhand einer Konfigurationsdatei gesteuert. Genauere Details zu dieser Datei können in Abschnitt 7.1 gefunden werden. In dieser Datei wird in einer Zeile ein Sql - Befehl bestimmt, der dann von LandSHIFT verwendet wird um Daten aus der Datenbank einzulesen. Soll die Anwendung also nur ein einziges Land berechnen muss also lediglich dieser Befehl geändert werden. Diesem Befehl können unter Anderem modifizierte ISO - Länderkennziffern übergeben werden, wodurch LandSHIFT angewiesen wird dieses Land, beziehungsweise diese Länder, zu berechnen.

Der implementierte Mechanismus liest eine Datei ein, aus der sich die erwähnten ISO - Kennzahlen ergeben und iteriert über eine Liste dieser Kennzahlen. Hierbei wird jeweils eine Konfigurationsdatei generiert, in welcher die

aktuelle Kennzahl gesetzt wird. Mit dieser Datei wird dann LandSHIFT gestartet.

4.2 Die DataExchange Bibliothek

Zur Berechnung einzelner Werte im Irrigation - Modul und später in AlternativeWatergap müssen einzelne Ergebnisse aus LandSHIFT exportiert werden, welche dann vom Irrigation - Modul verarbeitet werden müssen. Ebenso müssen Ergebnisse aus dem Irrigation - Modul in AlternativeWatergap verwendet werden. Zu diesem Zweck wurde eine Bibliothek zum Datenaustausch entwickelt.

Zum Datenaustausch wurde eine eigene template Klasse entwickelt. Diese Klasse ist prinzipiell eine Liste mit einigen Sonderfunktionen. Dementsprechend handelt es sich bei der Klasse um eine Spezialisierung des std::vector. Hierbei handelt es sich um ein Template, da unterschiedliche darzustellende Werte unterschiedliche Datentypen haben. Zusätzlich handelt es sich bei dieser Klasse um einen Multiton, um unterschiedliche Arten von Daten unterscheiden zu können während garantiert wird, dass es pro Datenart nur jeweils eine Instanz geben kann. Der Begriff „Datenart“ umschreibt hierbei den Inhalt der darzustellenden Daten, was unter Anderem die Landfläche oder eine Verwendete Wassermenge sein kann.

Da AlternativeWatergap auf die Zugehörigkeit eines Wertes zu einem Basin angewiesen ist musste diese Information neben dem tatsächlichen Wert berücksichtigt werden. Die Zugehörigkeit zu einem Basin ergibt sich anhand der Position eines Wertes. Die Reihenfolge in der Liste muss hierfür identisch sein mit der Werte - Reihenfolge in UNF Dateien. Da LandSHIFT selbst allerdings unabhängig von UNF Dateien ist muss gewährleistet sein, dass alle Anwendungen die gleichen Zellen verwenden. Hierfür wurde die LandSHIFT - Zellen ID eingebaut. Die Einträge in dieser Liste sind also Paare von Werten, wobei der erste Eintrag die Zell ID darstellt

Zur Verdeutlichung der Klassentyp sowie eine Schnittstellenbeschreibung:

```
template<typename T>
class IpcData
    : public std::vector<std::pair<int32_t, T>>
```

Genauer Datentyp der entwickelten Klasse. Stellt einen Vektor dar, der Paare von LandSHIFT - Zell ID und einem von einer Anwendung berechneten Wert speichert. Die Reihenfolge der Werte innerhalb des Vektors ist hierbei von Bedeutung, da AlternativeWatergap anhand der Reihenfolge der Einträge die Zugehörigkeit zu einem Basin bestimmt.

`enum DataKind`

Identifiziert die Art der enthaltenen Daten. Wird verwendet um einen Pointer auf eine spezielle Instanz des Multitons einzuholen.

```
static inline IpcData<T> * instance(DataKind kind)
```

Befehl um eine spezielle Instanz dieses Multitons abzurufen.

```
inline T * unfElements() const
```

Erzeugt ein Array von Werten und gibt dieses zurück. Das Array stellt hierbei nur die Werte dar, besitzt also keine Zell - ID. Die Reihenfolge der Werte ist identisch der Reihenfolge innerhalb des Vektors.

```
typedef std::vector<std::pair<int32_t, T>> Type;
```

Datentyp dieser Klasse als typedef.

```
typename Type::iterator findId(int32_t id)
```

Durchsucht diesen Vektor nach einem Paar mit der gegebenen ID. Gibt einen Iterator auf das gefundene Element zurück, falls ein solches existiert. Falls kein Eintrag gefunden wurde gibt diese Funktion einen ungültigen, mit end() vergleichbaren, Iterator zurück.

```
void convertLUT(IpcData<int32_t> & lutData);
```

Da LandSHIFT andere Zahlen zur Beschreibung von Landnutzungstypen verwendet als das Irrigation - Modul, müssen diese Konvertiert werden. Landnutzungstypen sind eine Variante der von dieser Klasse darzustellenden Daten und werden vom Irrigation - Modul zur Ausführung benötigt. Die genaue Zuweisung der Landnutzungstypen kann in Tabelle 1 eingesehen werden.

4.3 DataExchange in LandSHIFT

Der Einbau der DataExchange Bibliothek in LandSHIFT ist ähnlich trivial wie die Verwendung von LandSHIFT im allgemeinen. LandSHIFT berechnet Landnutzungstypen zellenweise. Sobald bei einer Zelle der Landnutzungstyp gesetzt wird, wird der Typ im entsprechenden Vektor aktualisiert. Hierfür wird die Zell ID zur Identifikation des korrekten Eintrages verwendet. Die Aktualisierung der Landnutzungsfläche erfolgt analog.

LandSHIFT	Irrigation - Modul
3000	123
3001	122
3002	102
3003	105
3004	115
3005	112
3006	-99
3007	104
3008	120
3009	109
3010	100
3011	106
3012	103
3013	121
3014	122
3015	108
3016	110
3017	116
3018	114
3019	113
3020	101

Tabelle 1: Zuweisung der Landnutzungstypen von LandSHIFT zum Irrigation - Modul

Die genauen Änderungen zum setzten der Landnutzungstypen sind in den Zeilen 373 - 378 zu finden:

```
371 void update_irri() {
372     LUT_irri = newLUT_irri;
373     typedef IpcData<int32_t> IpcInt;
374     IpcInt * lutData(IpcInt::instance(
            IpcInt::LandUse));
375
376     auto it(lutData->findId(identifier));
377     if(it != lutData->end())
378         it->second = LUT_irri;
379
380     this->setNLUT_irri(0);
381 }
```

Analog wird die Landfläche in den Zeilen 514 - 518 gesetzt

```
511 void setIrriArea(float val)
512 {
513     IrriArea = val;
514     typedef IpcData<float> IpcFloat;
515     IpcFloat * data(IpcFloat::instance(
            IpcFloat::IrrigatedArea));
516     auto it(data->findId(identifier));
517     if(it != data->end())
518         it->second = val;
519 }
```

4.4 Kopplung des Irrigation Moduls

Zur Kopplung des Irrigation - Moduls wurde die gesamte Funktionalität des Moduls hinter einem Wrapper - namespace versteckt. Dieser namespace steuert die bisherige Funktionalität nachdem kleinere Änderungen am original Quelltext vorgenommen wurden.

Für diese Implementierung wurden zunächst alle Vorkommen von Objekten, welche als „extern“ definiert wurden durch statische Pointer ersetzt, welche zur Laufzeit aktualisiert werden. Zum Setzen dieser Pointer wurden ebenfalls statische Funktionen in den jeweiligen Klassen implementiert. Des Weiteren wurden häufig verwendete string - Literale durch Konstanten ersetzt sowie wiederholte Array - Zugriffe durch das Setzen und Verwenden von Variablen. Abgesehen von diesen Kleinigkeiten wurde die Ausführung des Algorithmus aus dem Konstruktor in eine eigene Funktion extrahiert.

Eine spätere Modifikation des existierenden Quelltextes bestand in der Extraktion von „zellenweisen Funktionen“. Diese Funktionen fassen jeweils die Einzelschritte des Algorithmus zusammen, welche für alle Zellen wiederholt werden müssen. Es wurden insgesamt drei solcher Stellen identifiziert und entsprechend modifiziert. Die letzte Veränderung an den gegebenen Quellen war das auflösen von doppelt existierenden Datei - Paaren, wobei eines offensichtlich eine Kopie des anderen war. Der Einfachheit halber wurde ein Paar lediglich umbenannt. Ein Datei - Paar beschreibt ein Paar aus Header und dazugehörender Quelldatei.

Eine Anmerkung zum aktuellen Stand des Wrappers: Es ist zur Zeit nicht möglich das Irrigation - Modul Kontinental auszuführen. Dies ist der Initialisierung geschuldet, welche zur Zeit zwangsweise länderweise stattfindet.

Der Wrapper - namespace setzt sich dann wie folgt zusammen:

```
static gridioClass * s_gridIO;
static geoClass * s_geo;
static countryInfoClass * s_countryInfo;
static OptionsClassIrr32 * s_options;
static climateClass * s_climate;
static wg_irrigationClass * s_irrigation;
```

Statische Objekte welche vor Beginn als „extern“ markiert waren.

```
typedef std::vector<size_t> IsoVector;
```

Typdefinition zur Lesbarkeit.

```
static IsoVector s_isoVector;
```

Ein Vektor zum halten von UNF - Positionsdaten.

```
static FileData s_fileData;
```

Eine Map, welche Daten des aktuell bearbeiteten Landes hält.

```
static int32_t s_countryId;
```

Die ISO - Länderkennzahl des aktuellen Landes.

```
static int32_t s_continent;
```

Kennzahl des aktuellen Kontinents.

```
static int32_t s_maxcharlength = 255;
```

Die verwendete maximale String - Länge.

```
static bool s_coupled;
```

Wird auf TRUE gesetzt, falls die Anwendung im gekoppelten Modus ausgeführt wird. Hierdurch wird gesteuert, dass einige Eingaben nicht aus Dateien gelesen werden sollen. Die benötigten Daten kommen dann aus LandSHIFT.

```
void generalInit(int argc, char * argv[]);
```

Funktion zum initialisieren genereller Daten. „Generelle Daten“ sind unabhängig vom aktuell zu bearbeitenden Land.

```
void initIsoData(const std::string & path
    , int32_t isoId);
```

Liest eine ISO - ID UNF Datei und speichert Positionen der benötigten Zellen.

```
void initCountrySpecific();
```

Initialisiert all jene Daten, welche vom zu berechnenden Land abhängen.

```
void run()
```

Startet die Ausführung.

Die tatsächliche Einbindung des Irrigation - Moduls verwendet nun die bereits definierten Funktionen des Wrappers und startet das Irrigation - Modul länderweise nach jeder Ausführung von LandSHIFT. Dabei musste beachtet werden, dass beide Anwendungen unterschiedliche ISO - Kennzahlen verwenden. Das Irrigation - Modul verwendet originale Kennzahlen, während LandSHIFT von der Norm abweicht. Genaueres zu dieser Abweichung kann am Ende des Abschnitts 7.1 gefunden werden. Grundlegend bedeutet diese Abweichung, dass das Irrigation - Modul gegebenenfalls mehrfach für eine LandSHIFT Iteration ausgeführt werden muss.

4.5 Kopplung von AlternativeWatergap

Die letzte anzubindende Anwendung ist AlternativeWatergap. Die bereits implementierte Funktionsweise hat ermöglicht, dass diese Kopplung ohne größere Änderungen vorgenommen werden konnte. Der einzige umzusetzende Punkt lag im Anbinden der DataExchange Bibliothek. Abgesehen davon musste AlternativeWatergap von LandIrrigationGap heraus aufgerufen werden. Hierfür wurde lediglich der Inhalt der AlternativeWatergap Main - Funktion nach LandIrrigationGap kopiert und leicht modifiziert, da die Eingaben zur Wassernutzung nun teilweise aus dem Irrigation - Modul kommen. Alle weiteren Daten müssen nach wie vor aus Dateien gelesen werden.

5 Ansätze zur Fortführung dieser Arbeit

Diese Arbeit soll in Zukunft vom Fachgebiet Programmiersprachen und Methodik für Forschungszwecke weitergeführt werden. Mit der Absicht dem zukünftigen Bearbeiter beziehungsweise der zukünftigen Bearbeiterin einige Anreize zur Fortsetzung zu geben werden in diesem Kapitel einige Punkte erläutert, die offen geblieben sind.

5.1 Erforschung von LandSHIFT Abstürzen

Es kann durchaus möglich sein, dass dieses Problem nur auf dem Computer auftritt, auf welchem diese Abschlussarbeit entwickelt wurde. Ebenso kann es sein, dass die verwendete LandSHIFT - Version an dieser Stelle fehlerhaft ist. Der Fehler, der im folgenden noch genauer spezifiziert wird, tritt an einer Stelle im Quelltext auf, die von dieser Abschlussarbeit nicht modifiziert wurde. Dementsprechend ist die Annahme, dass dieser Fehler nicht von eingebauten Modifikationen ausgelöst wird. Aus zeitlichen Gründen konnte dieser Fehler allerdings nicht weiter erforscht werden.

Zu Beginn der komplette Callstack:

```
#0  0x00007ffff6d24e5f in
    std::basic_string<char, std::char_traits<char>
        , std::allocator<char> >::~basic_string()
    () from /usr/lib/x86_64-linux-gnu/libstdc++.so.6

#1  0x000055555565e3f4 in
    boost::filesystem::path::~path
    (this=0x7fffffffcaa0, __in_chrg=<optimized out>)
    at /.../waterland/LandSHIFT_Irrigation/Source/
        ../../Dependencies/include/boost/filesystem/
        path.hpp:56

#2  0x0000555555660bc5 in
    Observer::touchFile<IrrUnit>
    (this=0x7fffffffd0f0, obj=...)
    at /.../waterland/LandSHIFT_Irrigation/Source/
        Observer.h:55

#3  0x0000555555671732 in
    LandShift::LandShift
    (this=0x7fffffffd9d0, argc=0x7fffffffd9ac
        , argv=0x7fffffffdd28)
```

```
    at /.../waterland/LandSHIFT_Irrigation/Source/
       LandShift.cpp:88

#4  0x00005555556511aa in
    main (argc=11, argv=0x7fffffffdd28)
    at /.../waterland/LandSHIFT_Irrigation/Source/
    modelclient.cpp:58
```

Der Fehler tritt also im Destruktor eines std::string auf, was wiederum in einer Boost Routine ausgeführt wird. Der Aufruf an diese Funktion geschieht in der Headerdatei der Observer Klasse, in Zeile 55. Hier ein Auszug zu der betroffenen Stelle:

```
52 template<class T>
53 bool Observer::touchFile(T & obj)
54   {
55     bfs::path data_dir(basepath);
```

Der verwendete namespace „bfs“ ist eine verkürzte Schreibweise des namespaces „boost::filesystem“. Wie diese Zeile, die lediglich ein Objekt anlegt zum Programmabsturz führt ist offen.

5.2 Validierung der Änderungen am Irrigation - Modul

In Abschnitt 4.4 wurde erwähnt, dass das Irrigation - Modul zur Zeit nicht kontinental arbeiten kann. Dementsprechend konnten die Ergebnisse der modifizierten Variante nicht validiert werden. Ein möglicher Ansatz zur Prüfung könnte aus kontinentalen Ergebnissen jene Werte extrahieren, welche auch von der modifizierten Fassung berechnet wurden. Hierfür könnte wieder die Tatsache ausgenutzt werden, das UNF Dateien immer die selbe Reihenfolge an Zellen darstellen. So müssten exakt die Positionen ausgelesen werden, welche in einer anderen Datei die selben Länder - Kennzahlen besitzen wie auch das berechnete Land. Wird hierbei sichergestellt, dann originale und modifizierte Ergebnisse wieder die selbe Reihenfolge besitzen wäre der Vergleich trivial.

5.3 Anbindung von AlternativeWatergap an die Datenbank

Zur Zeit schreibt AlternativeWatergap seine Ergebnisse in eine Datei. Für eine Folgeiteration würde LandSHIFT diese Werte allerdings in der Daten-

bank suchen. Dementsprechend sollte AlternativeWatergap seine Ergebnisse dort ablegen. Zur Realisierung müsste theoretisch lediglich ein entsprechender Publisher geschrieben werden, welcher eine Verbindung zur Datenbank aufbaut und diese später auch wieder schließt. Die Implementierung der publish Funktion müsste dann lediglich einen Sql - Befehl an die Datenbank senden.

In der aktuellen Fassung müsste dem Dispatcher dieser neue Publisher bekannt gemacht werden und die Funktion writeMode entsprechend angepasst werden. Zusätzlich müsste das verwendete Enum um einen Eintrag erweitert werden. Eine simplere Alternative bestünde in der Entfernung der writeMode Funktion. Stattdessen könnte eine simple set Funktion verwendet werden, welche ein Publisher - Objekt setzt. Hierdurch würde ebenfalls ein zukünftiges Hinzufügen neuer Publisher vereinfacht werden.

5.4 Iteration über die Jahreskonfiguration

Gegenwärtig verwendet AlternativeWatergap nur die erste Zeile der Jahreskonfiguration. Für eine sinnvolle Ausführung müsste allerdings dynamisch eine bestimmte Zeile der Datei gewählt werden. Die Auswahl der Zeile geschieht aktuell über ein Literal in der Funktion „configuratedYears“. Diese Funktion ist Teil der öffentlichen Schnittstelle der RelationNode Klasse. Wird dieser Funktion schlicht ein Parameter hinzugefügt, welcher die Zeilennummer innerhalb der Datei identifiziert, könnte über die Anzahl der Zeilen iteriert werden. Als Hinweis: Eine Zeile dieser Konfigurationsdatei stellt einen Programmlauf der Anwendung dar.

5.5 Benennung von Szenarien in AlternativeWatergap

Namen von Szenarien müssten vollständig implementiert werden. Sie waren während der Bearbeitung dieser Arbeit schlicht nie von Bedeutung. Es gibt lediglich einen entsprechenden Eintrag in der Basin struct. Dieser könnte allerdings statisch sein, da das Szenario die gesamte Berechnung beschreibt, kein einzelnes Basin. So muss dieser Wert prinzipiell nur einmal pro Ausführung gesetzt werden.

5.6 Inkonsistenz der AlternativeWatergap Konfigurationsdatei

Ein kleiner Punkt existiert in der Konfigurationsdatei zu AlternativeWatergap. In dieser Datei sollen Werte, welche sich selten oder nie ändern in Form eines vollständigen Pfades gegeben werden. Einträge, welche sich regelmäßig

ändern sollten in Ordnerpfad und Dateipfad getrennt sein. Eine Datei folgt diesem Schema nicht: Der Pfad der Basin ID - UNF Datei ändert sich pro Kontinent. Hier sollte der Pfad getrennt sein, was in der aktuellen Fassung nicht der Fall ist.

6 CMake

CMake ist ein plattformübergreifendes Buildtool. Zur Verwendung von CMake werden Textdateien mit einem bestimmten Namen benötigt. Soll eine Datei von CMake verwendet werden, muss sie „CMakeLists.txt“ genannt werden. Wird CMake dann mit einem Verweis auf den Ordner, der diese Datei enthält, ausgeführt, analysiert CMake diese und generiert einige benötigte Hilfsdateien. Diese Ansammlung an Hilfsdateien ist der „CMake-Cache“. Wird CMake dann erneut ausgeführt werden nur einige Einträge in diesem Cache aktualisiert, falls dies notwendig ist. Die mögliche Ausgabe von CMake ist Vielseitig. Einerseits können Projektverwaltungsdateien für Entwicklungsumgebungen wie CodeBlocks oder Eclipse generiert werden. Andererseits können auch Unix Makefiles generiert werden. Zusätzlich gibt es Entwicklungsumgebungen, die ihrerseits CMake - Dateien direkt als Projektdateien verwenden können und CMake entsprechend zur Cache - Verwaltung einsetzen. Beispiele für solche Anwendungen sind CLion oder VisualStudio.

6.1 Grundlegende Befehle

Da diese Arbeit unter Anderem eine Vorbereitung für zukünftige Forschungsarbeiten ist, werden an dieser Stelle einige Grundlagen für CMake erläutert. Diese kurze Einführung ist allerdings in keiner Weise vollständig. Für weiterführende Informationen wird an die offizielle Dokumentation oder entsprechende Foren im Internet verwiesen.

Eingangs wurde bereits erwähnt, dass CMake Textdateien benötigt, die „CMakeLists.txt“ genannt sein müssen. In diesen Dateien wird dann das Softwareprojekt beschrieben. Hierfür wird eine Reihe von Befehlen verwendet. Neben diesen Anweisungen gibt es auch Kontrollstrukturen, beispielsweise „if - else“ Blöcke, wie sie auch in Programmiersprachen bekannt sind. Insgesamt kann diese CMake - Sprache als Projektbeschreibungssprache bezeichnet werden.

Der minimale Inhalt einer CMakeLists - Datei besteht aus der Angebe einer minimal benötigten CMake - Version, einem Projektnamen sowie einem Ziel, welches erstellt werden soll. Ein Ziel kann hierbei entweder eine ausführbare Anwendung oder eine statische/dynamische Bibliothek sein. Innerhalb der Datei werden diese Informationen mit den folgenden Befehlen gegeben:

```
cmake_minimum_required(VERSION "X.Y")
project("Name")
add_executable("Name" "Quellen") oder
add_library("Name" STATIC|SHARED "Quellen")
```

Bei den genannten Befehlen müssen die Anführungszeichen jeweils weggelassen werden.

Die Versionsnummer besteht aus zwei Zahlen, welche durch einen Punkt getrennt werden.

Die Namen, welche unter project und add_executable/add_library verwendet werden, können sich unterscheiden. Falls das Ziel per add_library gesetzt wird, wird unter Linux Systemen automatisch die Konvention eingehalten, dass einer Bibliothek das „lib“ - Präfix vorangestellt wird. Sollte beispielsweise das Ziel per

```
add_library(MyLib SHARED myLib.cpp)
```

gesetzt werden, wird die später kompilierte und gelinkte Bibliothek den Namen „libMyLib.so“ bekommen. Das Namenspräfix wird immer vorangestellt, auch wenn der übergebene Name bereits ein vorangestelltes „lib“ enthält. In diesem Fall würde der Name entsprechend mit „liblib“ beginnen.

In dem gerade aufgeführten Beispiel wurde bereits vorweggenommen, dass add_library, unter Linux, .so - Dateien erzeugt, falls SHARED angegeben wird. Analog werden .a - Dateien erzeugt, falls STATIC angegeben wird. Unter Windows - Betriebssystemen werden entsprechende .dll und .lib Dateien erzeugt.

Die Befehle add_executable und add_library benötigen jeweils Quelldateien, welche kompiliert werden sollen. Die Pfade dieser Dateien werden hierfür durch Leerzeichen oder Zeilenumbrüche getrennt aufgelistet. Pfade sind relativ zur CMakeLists - Datei anzugeben.

Einige Projekte besitzen Abhängigkeiten zu externen Bibliotheken. Zum auflösen dieser Abhängigkeiten muss CMake mitgeteilt werden welche Bibliothek wo zu finden ist. Hierfür gibt es mehrere Möglichkeiten. In dieser Arbeit wurde allerdings nur eine Option verwendet

```
find_library("Ergebnis Name" "Name"
    PATHS "Pfad" NO_DEFAULT_PATH)
```

Mit diesem Befehl wird CMake angewiesen im Verzeichnis „Pfad“ nach einer Bibliothek „Name“ zu suchen und das Ergebnis dieser Suche in „Ergebnis Name“ zu speichern. Mit dem Flag „NO_DEFAULT_PATH“ wird CMake angewiesen keine Standardverzeichnisse nach dieser Bibliothek zu durchsuchen. Das ist dann notwendig, wenn eine Bibliothek in mehreren Versionen installiert ist oder sichergestellt werden soll, dass eine bestimmte Variante der Bibliothek verwendet wird. Der Eintrag „Ergebnis Name“ stellt eine Variable in CMake dar. Der Inhalt dieser Variablen ist entweder der Pfad der gesuchten Bibliothek, falls sie gefunden wurde, oder „Name“_NOTFOUND. Falls die Bibliothek gefunden wurde, kann sie beim linken verwendet werden.

Um ein Ziel mit externen Abhängigkeiten kompilieren zu können werden in aller Regel noch include - Verzeichnisse benötigt. Diese Verzeichnisse werden mittels

```
target_include_directories("Zielname" PUBLIC "Pfad")
```

gesetzt. Der Zielname muss hierbei der Name eines Zieles sein, welcher zuvor mit add_executable/add_library bekannt gemacht wurde.

Um CMake anzuweisen, dass Dateien gelinkt werden sollen wird der Befehl

```
target_link_libraries("Zielname" "Bibliotheken")
```

verwendet. Die Bedeutung von „Zielname“ ist hierbei identisch zu der im letzten Absatz erläuterten. Bibliotheken stellt eine, durch Leerzeichen oder Zeilenumbrüche getrennte, Liste von Bibliotheken dar.

Zur Verwendung von Variablen, beispielsweise zum linken gefundener Bibliotheken, wird der Name der Variablen in geschweifte Klammern gesetzt und ein Dollarsymbol vor die öffnende Klammer gesetzt. Ein Beispiel wäre ${Variable}, wobei „Variable“ der Name der Variablen ist. Variablen können auch manuell auf einen bestimmten Wert gesetzt werden.

```
set("Var" "Value")
```

Mit diesem Aufruf wird der Variablen „Var“ der Wert „Value“ zugewiesen. Soll „Value“ einen Text mit Leerzeichen enthalten, muss dieser mit Anführungszeichen umschlossen werden, sonst können diese entfallen. Der Name der Variablen darf nicht von Anführungszeichen umschlossen sein. Soll eine Variable eine Liste, beispielsweise von Quelldateien oder Bibliotheken, enthalten, werden mit dem Befehl

```
list(APPEND "Name" "Values")
```

der Liste „Name“ die Werte „Values“ angehangen. Die Verwendung einer Liste geschieht wieder durch die bereits erwähnte ${Name} Syntax. Listen oder Variablen werden nicht explizit erzeugt oder sonst unterschiedlich behandelt. Beide können Verwendet werden, nachdem sie per „set“ oder „list“ bekannt gemacht wurden.

Zusätzlich zu benutzerdefinierten Variablen gibt es noch einige von CMake vorgegebene Variablen. Hiervon werden allerdings nur einige erläutert, welche auch verwendet wurden:

- CMAKE_BUILD_TYPE Legt fest ob ein „Debug“ oder „Release“ Build gebaut werden soll.
- CMAKE_CXX_STANDARD Legt den zu verwendenden C++ Standard fest.

- CMAKE_CXX_FLAGS Hiermit können Flags direkt an den Compiler übergeben werden.
- CMAKE_VERBOSE_MAKEFILE Falls der gesetzte Wert „ON“ ist werden detailliertere Statusmeldungen von Compiler und Linker ausgegeben.
- CMAKE_ARCHIVE_OUTPUT_DIRECTORY Pfad zu einem Verzeichnis: Erzeugte statische Bibliotheken sind nach dem Erstellungsprozess hier zu finden.
- CMAKE_LIBRARY_OUTPUT_DIRECTORY Pfad zu einem Verzeichnis: Erzeugte dynamische Bibliotheken sind nach dem Erstellungsprozess hier zu finden.

Eine Anmerkung zu den Ausgabeverzeichnissen für statische und dynamische Bibliotheken: Die Pfade dieser Verzeichnisse können auf unterschiedliche Art und Weise gesetzt werden. Einerseits können hierfür die bereits erläuterten Variablen mit entsprechenden Werten gesetzt werden. Eine ebenfalls im Projekt verwendete Alternative ist das setzen von Projekteigenschaften. Diese Alternative soll am Beispiel des „DataExchange“ - Teilprojektes erläutert werden:

```
set_target_properties(DataExchange
PROPERTIES
ARCHIVE_OUTPUT_DIRECTORY "${CMAKE_BINARY_DIR}/../lib"
LIBRARY_OUTPUT_DIRECTORY "${CMAKE_BINARY_DIR}/../lib"
)
```

Hierbei stellt „DataExchange“ den Namen des Ziels dar, für welches die Eigenschaften gelten sollen. Nach dem Schlüsselwort „PROPERTIES“ folgen die Eigenschaften, die gesetzt werden sollen, mit den Werten, die sie annehmen sollen. Zu beachten ist hierbei das fehlende CMAKE_ - Präfix zu Beginn der Eigenschaftsnamen. Die Variable CMAKE_BINARY_DIR beschreibt den Pfad des Cache - Verzeichnisses.

6.2 Erstellen der einzelnen Teilprojekte

Das erstmalige erstellen des Gesamtprojektes, unter Verwendung von CMake kann gegebenenfalls Ungewohnt sein. Da das Projekt in Zukunft aber von anderen Bearbeitern weitergeführt werden soll ist ein erfolgreiches Erstellen auf anderen Computern von anderen Personen zwangsweise erforderlich. Um diesen Prozess zu erleichtern wird in diesem Kapitel der Bauvorgang erläutert.

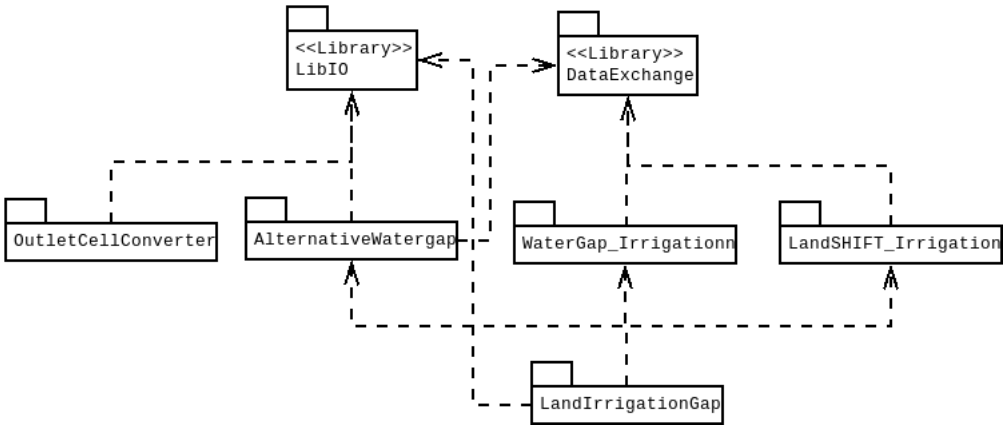

Abbildung 1: Abhängigkeiten der einzelnen Teilprojekte

6.2.1 Projekt interne Abhängigkeiten

Da die einzelnen Teilprojekte teilweise abhängig von einander sind, müssen einige von ihnen in einer bestimmten Reihenfolge erstellt werden. Diese Abhängigkeiten der einzelnen Teilprojekte werden in Abbildung 1 graphisch dargestellt.

Insgesamt enthält das Projekt zwei Bibliotheken:

- LibIO und
- DataExchange

Beide Bibliotheken sind jeweils unabhängig von anderen Teilen dieser Arbeit.

Alle anderen Teilprojekte sind von einer dieser beiden, oder beiden, Bibliotheken abhängig. Die folgenden Anwendungen sind von LibIO direkt Abhängig:

- AlternativeWatergap
- LandIrrigationGap
- OutletCellConverter

Abhängigkeiten von DataExchange sind:

- AlternativeWatergap
- LandIrrigationGap
- LandSHIFT_Irrigation

- WaterGap_Irrigation

Da LandIrrigationGap eine gekoppelte Variante der drei Anwendungen AlternativeWatergap, LandSHIFT_Irrigation sowie WaterGap_Irrigation darstellt ist es offensichtlich von diesen Anwendungen abhängig.

6.3 Projekterstellung aus der Konsole

Prinzipiell besteht der erste Bauvorgang aus zwei Schritten. Zunächst muss der CMake - Cache generiert werden. Hierzu wird einfach der Befehl `cmake` verwendet. Diesem Befehl wird noch der Pfad zu einem Quellverzeichnis mitgeteilt. Das Quellverzeichnis ist hierbei das Verzeichnis, welches die CMakeLists.txt Datei enthält. Zu beachten ist bei der Ausführung des Befehls, dass der Cache genau in dem Verzeichnis angelegt wird, aus dem der Befehl ausgeführt wird. Befindet man sich aktuell also im Quellverzeichnis selbst, würde mit dem Befehl cmake . der Cache ebenfalls im Quellverzeichnis angelegt. Um ein späteres Neuerzeugen des Caches zu vereinfachen sollte der Cache in einem separaten Verzeichnis generiert werden. Hierdurch kann der Cache entfernt werden, indem der gesamte Ordnerinhalt gelöscht wird.

CMake wird während der Ausführung des Befehls einige Statusmeldungen in die Konsole schreiben, unter Anderem den detektierten Compiler. Unter Linux ist dies, falls kein anderer installiert wurde, der gcc / g++. Dementsprechend werden dann Linux Makefiles generiert.

Im Anschluss an diesen ersten Befehl muss ein Projekt kompiliert und gelinkt werden. Hierfür stehen, unter Linux, zwei Varianten zur Verfügung: Einerseits kann ein Projekt per „make“ gebaut werden, da die entsprechenden Makefiles generiert wurden, andererseits kann wieder der „cmake“ Befehl verwendet werden. Unter der Annahme, dass man sich in dem Verzeichnis befindet, in dem der Cache generiert wurde, kann ein Projekt mit dem folgenden Befehl erstellt werden:

```
cmake --build . -j X
```

Die Option „- -build .“ weist CMake hierbei an, aus dem aktuellen Verzeichnis die Ziele zu erstellen, welche vorher in der CMakeLists - Datei definiert wurden. Die zweite Option, „-j X“ wird an das geschachtelte „make“ weitergegeben, und weist den Erstellungsprozess an X Threads zum Erstellen zu verwenden. X stellt hierbei eine Zahl dar.

Die aktuell enthaltenen CMakeLists Dateien sorgen bereits dafür, dass von anderen Projekten erzeugte Abhängigkeiten gefunden werden, falls das Gesamtprojekt aus dem Repository ausgecheckt wird und Verzeichnisse nicht verschoben werden. Sollten Verzeichnisse von Teilprojekten verschoben wer-

den, müssen die darauf verweisenden Pfade in den CMakeLists Dateien der abhängigen Projekte entsprechend angepasst werden.

Der Vollständigkeit halber werden an dieser Stelle alle Befehle zum Erstellen aller Teilprojekte aufgeführt. Hierzu wird angenommen, dass das Projekt bereits aus dem Repository aus gecheckt wurde aber keine weiteren Änderungen vorgenommen wurden.

```
cd /Pfad/zu/waterland
mkdir AlternativeWatergap/bin
mkdir DataExchange/bin
mkdir LandIrrigationGap/bin
mkdir LandSHIFT_Irrigation/bin
mkdir LibIO/bin
mkdir OutletCellConverter/bin
mkdir WaterGAP_Irrigation/bin

cd LibIO/bin
cmake ../src
cmake --build .

cd ../../DataExchange/bin
cmake ../src
cmake --build .

cd ../../OutletCellConverter/bin
cmake ../src
cmake --build .

cd ../../AlternativeWatergap/bin
cmake ../src
cmake --build .

cd ../../LandSHIFT_Irrigation/bin
cmake ../Source
cmake --build .

cd ../../WaterGAP_Irrigation/bin
cmake ..
cmake --build .

cd ../../LandIrrigationGap/bin
```

```
cmake ../src
cmake --build .
```

Zu Beachten sind die abweichenden Pfade zu den jeweiligen Quellverzeichnissen bei den Projekten LandSHIFT_Irrigation sowie WaterGAP_Irrigation.

7 Technische Details

In diesem Kapitel werden einige Details zu Dateiformaten und Ausgaben von Anwendungen dokumentiert.

7.1 LandSHIFT Konfigurationsdatei

Der Inhalt der Konfigurationsdatei kann im wesentlichen auf drei Teile reduziert werden:

1. Datenbank-Zugriff,
2. Datenbank-Namen und
3. Länderauswahl

Der erste Abschnitt beschreibt die (Netzwerk-) Adresse der Datenbank sowie benötigte Anmeldedaten (Nutzername und Passwort). LandSHIFT wird dann versuchen eine Verbindung mit einem MySql - Server unter der gegebenen Adresse mit den zur Verfügung stehenden Anmeldedaten aufzubauen.

Punkt Zwei beschreibt eine Reihe von Datenbanken, die auf dem Server existieren müssen. Für eine genaue Auflistung der benötigten Datenbanken sei an dieser Stelle auf die Konfigurationsdatei selbst verwiesen.

Die Länderauswahl unter dem Eintrag „db_countries“ steuert LandSHIFT direkt. Hier wird ausgewählt, für welche Länder LandSHIFT eine Simulation durchführen soll. Die Auswahl erfolgt in Form eines Sql - Befehls welcher dann innerhalb der Anwendung verwendet wird. Der folgende Befehl weist LandSHIFT an, für alle Länder, also Global, zu rechnen.

```
select * from countries;
```

Zu beachten ist hierbei das Semikolon am Ende des Befehls.

Soll LandSHIFT nur eine Teilmenge der existierenden Länder berechnen, muss der Befehl wie folgt abgeändert werden:

```
select * from countries where iso_id = "ID";
select * from countries where iso_id in ("ID Liste");
```

Beide Varianten können verwendet werden. Im ersten Fall muss „ID“ durch eine gültige ISO - Länder - ID ersetzt werden, im zweiten Fall kann eine durch Kommata getrennte Liste solcher IDs angegeben werden. Hierbei ist zu beachten, dass die verwendeten IDs nicht zwangsweise echte ISO - IDs sind. Verwendet werden hier „fake_ISONUM“ - Schlüssel. Der Name ergibt sich aus der entsprechenden Spaltenbenennung in der Datenbank. Diese Fake-Nummern sind in den meisten Fällen Deckungsgleich mit den echten

fake_ISONUM	Reale ISO-Kennzahlen
9901	233, 428, 440
9902	56, 442
9903	156, 158, 344, 446
9904	438, 756
9905	0, 28, 44, 52, 92, 136, 212, 308, 312, 474, 500, 530, 531, 533, 534, 630, 652, 659, 660, 662, 663, 670, 780, 796, 850
9906	246, 248
9907	250, 492
9908	254, 328, 740
9910	336, 380, 470, 674
9911	504, 732, 0
9912	60, 74, 132, 234, 238, 239, 666, 678, 744
9913	70, 499, 688, 807, 891
9914	86, 162, 166, 174, 175, 260, 334, 462, 480, 638, 690
9915	16, 184, 258, 296, 316, 520, 540, 570, 574, 580, 581, 583, 584, 585, 612, 772, 776, 798, 876, 882
9916	48, 702, 48
9917	414, 512, 634, 784
9918	728, 736
9919	20, 292, 724
9920	826, 831, 832, 833

Tabelle 2: Zusammensetzung der Fake-ISO-Kennzahlen

ISO - Werten. Es gibt allerdings einige Abweichungen, bei denen mehrere reale Länder, beziehungsweise deren Kennzahlen, zu einer nicht realen, neuen Kennzahl zusammengefasst wurden. Die zusammengefassten Länder - Kennzahlen sowie die verwendeten Fakes sind in Tabelle 2 dargestellt. Diese Zuweisung wurde direkt aus der Datenbank erstellt.

7.2 Dateiformate

7.2.1 ASC-Dateien

Diese Dateien stellen prinzipiell Rasterkarten dar.

Die Karten werden zunächst in einem Header beschrieben. Im Anschluss an den Header werden alle Einträge wieder in tabellenartiger Form aufgelis-

tet. Einträge sind durch Whitespaces getrennt. Alle Einträge repräsentieren hierbei einen Wert, der für eine fünf Minuten Zelle berechnet wurde. Die Bedeutung des Wertes ergibt sich aus dem Pfad der Datei, inklusive Namen.

Der Header besteht aus den Einträgen:

- ncols: Anzahl der Spalten
- nrows: Anzahl der Zeilen
- xllcorner: X Koordinate der unteren linken Ecke
- yllcorner: Y Koordinate der unteren linken Ecke
- cellsize: Zellgröße in Grad
- NODATA_value: Dieser Wert dient als Markierung für „Keine Daten“

Die beiden Koordinaten definieren die Position der Zelle der letzten Zeile und ersten Spalte auf dem Globus. Die Angegebenen Koordinaten sind in Längen- und Breitengrad gemessen.

7.2.2 UNF

Die meisten der vom Irrigation - Modul gelesenen Dateien sind im UNFX - Format gegeben. Hierbei ist das „X“ ein Platzhalter für eine der Ziffern: 0, 1,2 oder 4. Diese Ziffer am Ende des Formats gibt die Länge der enthaltenen Werte in Byte an. Die Ausnahme stellt hierbei die Null dar. Eine UNF0 Datei beinhaltet 32 Bit float Werte. Die UNF1, UNF2 und UNF4 beinhalten jeweils ein, zwei oder vier Byte breite, signed Integer. Die enthaltenen Zahlen sind serialisiert. Ein Eintrag stellt den Wert einer Zelle dar.

7.2.3 D.UNF

Zusätzlich zu den regulären UNF - Dateien gibt es eine weitere Variante: *.D.UNFX. Bei dieser Datei - Unterart ist eine doppelte Dateiendung gegeben. Auf den Dateinamen, zuvor mit „*“ markiert, folgt eine Zahl. Diese Zahl muss nicht einstellig sein. Auf diese Zahl folgt dann eine der bereits vorgestellten UNFX Endungen. Die zusätzliche Zahl in dieser doppelten Dateiendung gibt an wie viele Werte zusammen eine Zelle repräsentieren. Eine Datei mit der Endung .31.UNF0 würde dementsprechend angeben, dass pro Zelle 31 Werte des Typs 32Bit float gelesen werden müssen.

7.3 Ausgaben

7.3.1 LandSHIFT

LandSHIFT produziert eine Vielzahl an Ausgaben. An dieser Stelle werden die verwendeten Dateiformate sowie die Bedeutung von zwei der produzierten Ausgaben erläutert.

Die Dateiformate aller Ausgaben reduzieren sich auf drei Formate: TXT, CSV und ASC. Die TXT - Dateien stellen in nahezu allen Fällen tabellarische Ausgaben dar und sind prinzipiell CSV - Dateien. Diese Klartext - Tabellen verwenden jeweils Tabulatoren als Trennzeichen. Die einzige Ausnahme hierzu stellt die „agrivalidation.txt“ - Datei dar.

CSV Dateien sind genau das: Komma-Separierte-Werte.

Die größte Anzahl an Ausgabedateien besitzt das ASC - Dateiformat

7.3.2 Irrigation - Modul

Die vom Irrigation - Modul berechneten Mengen entnommenen und verbrauchten Wassers werden in Form von 12.UNF0 Dateien ausgegeben. Hierbei stellt jeder einzelne Eintrag die Entnahme beziehungsweise den Verbrauch eines Monats pro Zelle die dar. Zusätzlich schreibt das Irrigation - Modul den jährlichen Wasserverbrauch der in einer Zelle angebauten Feldfrucht. Hierfür wird ein 23.UNF0 Format verwendet. Dabei sind allerdings alle Werte, ausgenommen eines Einzigen, auf Null gesetzt. Anhand der Position des von Null verschiedenen Wertes lässt sich bestimmen, welche Feldfrucht in dieser Zelle angebaut wird. Das Irrigation - Modul produziert noch einige weitere Ausgaben, welche hier allerdings nicht weiter diskutiert werden sollen.

7.3.3 OutletCellConverter

Die Ausgaben des OutletCellConverters werden von AlternativeWatergap als Eingabe benötigt. Prinzipiell genügt es diese Anwendung einmal pro Kontinent auszuführen, da sich die Ergebnisse nicht mehr ändern. Die Ausgabe der Anwendung erfolgt als einfache CSV Datei, bei der jede Zeile zunächst die Basin ID ausgibt, danach, durch ein Komma getrennt, die Position der Outlet Cell in einer UNF Datei.

7.3.4 AlternativeWatergap

AlternativeWatergap schreibt seine berechneten Basin - Werte in Form einer CSV - Datei. Hierbei wird für jedes Basin eine Zeile mit den jeweiligen Werten geschrieben. Einzelne Werte sind durch Kommata getrennt. Für die Reihenfolge der einzelnen Werte sei an die Ausgabedatei selbst verwiesen.

8 Verwendete Ressourcen

Für die Umsetzung dieser Abschlussarbeit wurden einige Ressourcen des Fachgebietes Programmiersprachen und Methodik verwendet:

- ein GIT - Repository zur Quellverwaltung
- Interne Server zum Austausch größerer Datenmengen

8.1 PLM Server

Zum Abschluss dieser Arbeit befinden sich die im Folgenden erläuterten Daten und Verzeichnisse auf dem Server. Alles genannte ist im Verzeichnis „waterland“ zu finden. Zusätzlich zu den weiter erläuterten Daten gibt es die beiden Dateien

- DomesticConsumptionTest.txt sowie
- G_DOM_WC_m3_2005.csv

Beide Dateien wurden zu Vergleichszwecken erstellt und sollten in Zukunft nicht weiter benötigt werden. Die Inhalte der jeweiligen Unterverzeichnisse sind prinzipiell selbsterklärend. Der Vollständigkeit halber werden sie dennoch kurz erläutert.

8.1.1 alles.tar.gz

Dieses Archiv beinhaltet, in einem weiteren Unterordner platziert, einen Ausschnitt aus der CESR Datenbank sowie eine benötigte Laufzeitumgebung für das Irrigation - Modul.

Der Datenbank Auszug ist in Form von SQL Dateien gegeben. Diese Dateien können verwendet werden um auf einem anderen Datenbankserver Datenbanken zu erzeugen und diese mit Dateneinträgen zu füllen.

Die Laufzeitumgebung für das Irrigation - Modul beinhaltet sämtliche Eingabedateien, die zur Ausführung des Irrigation - Moduls benötigt werden. Zusätzlich ist ein Script enthalten, wodurch das Irrigation - Modul für alle Kontinente jeweils einmal ausgeführt wird. Dieses Script ist allerdings noch nicht ausführbar und muss zunächst mit einem entsprechenden Aufruf von „chmod“ als ausführbare Datei markiert werden.

8.1.2 DB-Test

Dieses Verzeichnis beinhaltet zwei gezipte Archive:

- MySqlTest.zip und
- MySqlConnector-8.zip

Das erste dieser Beiden beinhaltet die in Kapitel 9 beschriebene Testanwendung, mit der getestet werden kann, ob MySql Connector Version 8 verwendet werden kann, um mit einem laufenden MySql - Server eine Verbindung herzustellen. Zusätzlich zu den Quellen ist ein Script enthalten, mit dem die Anwendung kompiliert werden kann.

Das zweite Archiv beinhaltet die benötigten Shared - Libraries und Header Dateien, welche zum Kompilieren und Ausführen der Testanwendung benötigt werden.

8.1.3 dir_fwimmer

Dieses Verzeichnis ist tatsächlich in dem Eingangs erwähnten Archiv, „alles.tar.gz“, enthalten. Es handelt sich hierbei um den Unterordner, der den gesamten Inhalt es Archivs umfasst. Dem entsprechend wurden einige Inhalte bereits im Abschnitt des Archivs beschrieben. Diese Inhalte werden hier nicht wiederholt erläutert, sind aber ebenso enthalten.

Zusätzlich zu den bereits beschriebenen Inhalten existieren allerdings eine Reihe weiterer Daten, exklusiv in diesem Verzeichnis.

Das Unterverzeichnis bas20k_id_UNF beinhaltet UNF Dateien, welche jeder Zelle eine Basin - ID zuweist. Diese Dateien sind für jeden Kontinent gegeben.

Im Verzeichnis input_AW sind Eingabedaten für Alternative Watergap zu finden. Diese Eingaben sind in zwei weitere Unterverzeichnisse aufgeteilt: ALLOCATED_USE und HYDROLOGY. Das erste dieser beiden Verzeichnisse stellt Wasserentnahme- und Wasserverbrauchsdaten für andere Wassernutzungsarten(Domestic, Manufacturing, etc) zur Verfügung. Im HYDROLOGY - Verzeichnis wird die Wasserverfügbarkeit aus Flüssen gegeben, jeweils in kontinental sortierte Unterverzeichnisse verteilt. Alle dieser Daten sind ebenfalls im UNF - Format gegeben.

Das Verzeichnis landshift_id_5min beinhaltet UNF - Dateien, welche jeder Zelle eine LandSHIFT ID zuweisen. Vergleichbar hierzu stellt das landshift_iso_id Verzeichnis ISO - Länderkennungen zur Verfügung.

Vergleichsdaten zur Verifikation von AlternativeWatergap sind im Verzeichnis Testdaten_AW zu finden. Diese Daten wurden mit dem ursprüngli-

chen R - Script erstellt und dienen als Eingabe für die AlternativeWatergap - Testanwendung.

Die beiden Verzeichnisse WaterGap_cell_area und watergap_rout_area beinhalten die entsprechenden Flächen, jeweils als UNF - Dateien pro Kontinent.

Das letzte zu nennende Verzeichnis ist watergap_id_5min. Hierin sind WaterGap Zell - IDs im UNF - Format gegeben. Daten sind jeweils kontinental enthalten.

8.2 Versionsverwaltung

Zur Versionsverwaltung und der Dateistruktur gibt es nicht viel zu sagen. Verwendet wurde ein GIT Repository in welchem sämtliche Quelldateien verwaltet werden.

Prinzipiell stellt jeder entwickelte Teil (AlternativeWatergap, LibIO, etc.) ein separates Teilprojekt dar und ist dementsprechend in sich abgeschlossen und in einzelne Unterordner sortiert. Sämtliche Teilprojekte besitzen ein Quellverzeichnis, „src“ genannt und, falls benötigt, ein Verzeichnis für Eingabedaten. Libraries besitzen zusätzlich ein include - Verzeichnis für Headerdateien. Die Ordnerstruktur der gegebenen Anwendungen wurde nicht verändert. So besitzt das Irrigation - Modul kein separates Verzeichnis für Quelldateien. Das in LandSHIFT vorhandene Quellverzeichnis heißt „Source“.

Es gibt zwei Verzeichnisse, die nicht in dieses Schema passen „Dependencies“ und „Uploads“.

8.2.1 Dependencies

Wie der Name „Dependencies“ bereits andeutet sind hier sämtliche externen Abhängigkeiten untergebracht. Diese Abhängigkeiten sind:

- Boost: Filesystem,
- Boost: Program - Options,
- Boost: System,
- MySql: Client sowie
- MySql++

Die Boost Libraries sind in Version 1.54.0 enthalten, die Version von MySql Client ist 18.0.0 und die Versionsnummer von MySql++ ist 2.3.2. Die Shared Objects sind in einem Unterverzeichnis, „lib“, enthalten.

Zusätzlich zu diesen verwendeten externen Libraries sind einige aktuell nicht verwendete Libraries im Dependencies - Verzeichnis enthalten:

- Die selben Boost Libraries, allerdings in Version 1.69.0 und
- MySql Connector Version 8

Diese Zusatzinhalte sind in einem Unterverzeichnis, „newLibs“, gekapselt.

Die benötigten Header sind in einem include Verzeichnis mit entsprechenden Unterverzeichnissen gegeben. Die aktuell nicht verwendeten/benötigten Header sind im Unterverzeichnis „newIncludes“ gebündelt.

8.2.2 Uploads

Dieses Verzeichnis wurde zu Beginn dieser Arbeit zum Datenaustausch verwendet. Dementsprechend sind hier einige zip - Archive enthalten. Die meisten Inhalte der Archive sind tatsächlich eher als Backup zu interpretieren, da die Inhalte, direkt oder abgewandelt, an anderen Stellen entpackt vorhanden sind:

- Datenbankdumps und Startkonfigurationen für LandSHIFT sind auf dem PLM Server zu finden
- LandSHIFT und Irrigation - Modul Quellen sind, modifiziert, in eigene Verzeichnisse der Verwaltung entpackt
- MySql Abhängigkeiten sind im „Dependencies“ Verzeichnis enthalten

Die einzigen, nicht anderweitig enthaltenen Dateien sind im Archiv „bas20k_relation_upbasins.zip“ enthalten. Hierin sind Eingaben für AlternativeWatergap enthalten, welche zur Ausführung benötigt werden.

9 Modernisierung der Datenbankschnittstelle

Ein optionaler Zusatz dieser Abschlussarbeit bestand in der Modernisierung der inzwischen veralteten Datenbankschnittstelle. Wie bereits erwähnt wurde, wird ein MySql - Server verwendet. Dementsprechend wird auch eine MySql - Software - Schnittstelle verwendet. Die Verwendete Schnittstelle ist MySql++ Version 2.3.2. Zusätzlich wird MySql Client in Version 18.0.0 benötigt. Die aktuellste Version einer MySql Schnittstelle, während der Erstellung dieser Arbeit, ist MySql Connector Version 8. Diese Connector Schnittstelle bietet eine JDBC Anbindung an die Datenbank, welche der bereits verwendeten MySql++ Syntax sehr ähnlich ist. Da sich im Nachhinein nicht lösbare Probleme mit der modernisierten Schnittstelle ergeben haben, wurde auf die Verwendung der aktualisierten Variante verzichtet. Da diese Modernisierung prinzipiell existiert, wird sie hier dennoch vorgestellt, inklusive der damit verbundenen Problematik.

9.1 Machbarkeitsanalyse

Bevor irgendwelche Änderungen am vorhandenen DataManager durchgeführt wurden, wurde zunächst geprüft, ob die gewählte Connector-Version mit der Datenbank kommunizieren kann. Hierfür wurde eine Beispielanwendung aus der Connector - Dokumentation verwendet, welches wiederum leicht abgeändert wurde. Die Funktionsweise dieser Testanwendung ist simpel:

- Verbindung zur Datenbank aufbauen
- Alle gefundenen Tabellen ausgeben
- Verbindung schließen

Die Ausgabe dieser Testanwendung ist in Abbildung 2 dargestellt. Es ist also möglich mit MySql Connector eine Verbindung zur installierten Datenbank aufzubauen.

9.2 Vergleich der Syntax

Zur Durchführung der Modernisierung müssen einige syntaktische Änderungen im DataManager durchgeführt werden. Diese Modifikationen werden an dieser Stelle kurz vorgestellt.

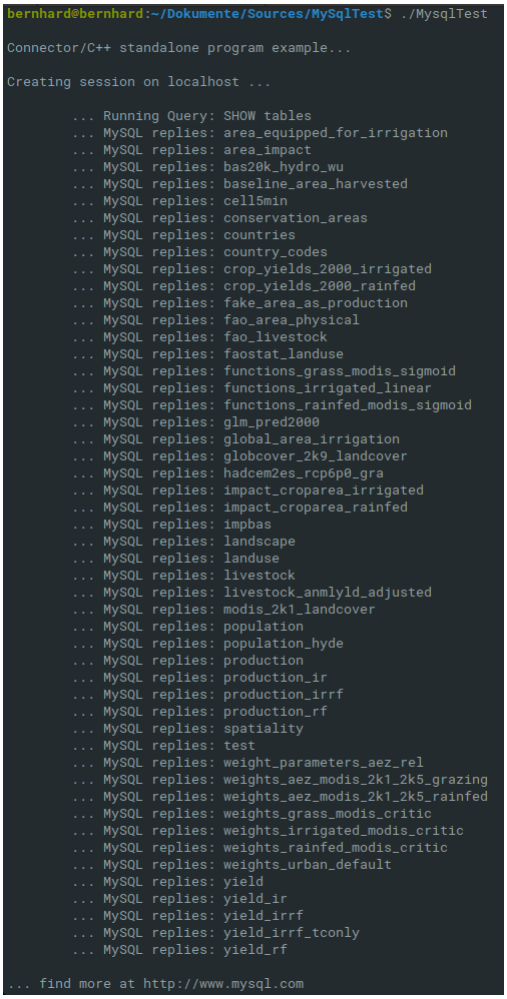

Abbildung 2: Ausgabe der Testanwendung

9.2.1 Verbindungsaufbau

Der Verbindungsaufbau in MySql++ ist ein simpler Funktionsaufruf. Hierbei wird auf einem `mysqlpp::Connection` - Objekt die `connect` Funktion aufgerufen. Dieser Funktion werden, jeweils als String, die Datenbank, Netzwerkadresse des Servers, Nutzername und Passwort mitgegeben.

In der Connector - Syntax muss zunächst eine Instanz eines `sql::Driver` - Objektes eingeholt werden. Dies geschieht über einen parameterlosen Funktionsaufruf. Auf diesem Objekt kann dann ebenfalls eine `connect` Funktion aufgerufen werden. Die Parameter sind dabei beinahe identisch zur MySql++ Syntax. Der einzige Unterschied besteht darin, dass der Datenbankname wegfällt. Diese connect Funktion gibt einen Pointer auf ein `sql::Connect` - Objekt zurück. Auf diesem Objekt muss dann noch ein Schema gesetzt werden, wofür hier der Datenbankname verwendet wird. Damit ist der Verbindungsaufbau abgeschlossen.

9.2.2 Anfragen an die Datenbank

Zum Benutzen der Datenbank muss in MySql++ Syntax zunächst ein Objekt vom Typ `mysqlpp::Query` erzeugt werden. Dies geschieht über eine Funktion auf einem `mysqlpp::Connection` - Objekt. In diesem Query - Objekt wird dann der Sql - Befehl gesetzt. Im Anschluss wird das Ergebnis dieses Querys in einem `std::vector` gespeichert. Hierfür wird auf dem Query - Objekt die `storein` Funktion aufgerufen, die als Parameter eine Referenz auf den vector erwartet. Zur Betonung: Die Anfrage an die Datenbank ist im Speichern des Befehls versteckt. Der `std::vector` hält nun alle gefundenen Einträge aus der Datenbank.

Bevor die Datenbank verwendet werden kann muss allerdings noch eine gewisse Vorarbeit durchgeführt werden. Anfragen an die Datenbank liefern Listen von Objekten. Die Klassen dieser Objekte müssen zuvor definiert werden. Hierfür werden in einer separaten Datei, DataTables.h, durch Verwendung von Makros entsprechende Klassen oder Structs beschrieben. Diesen Makros müssen alle Spalten mitgeteilt werden, welche die Ergebnisse der Sql - Anfragen haben werden. Soll ein Sql - Befehl geändert werden, muss sowohl das Query, als auch der entsprechende Makro - Aufruf entsprechend angepasst werden.

In JDBC Syntax wird zunächst ein String erzeugt. Dieser String stellt den Sql - Befehl dar, welcher in Klartext gesetzt wird. Um diesen Sql - Befehl auszuführen muss auf einem `sql::Connect` - Objekt die parameterlose `createStatement` Funktion aufgerufen werden. Auf dem daraus erzeugten Statement - Objekt wird dann die Funktion `executeQuery` aufgerufen, wel-

cher der zuvor erstellte Query - String übergeben wird. Das Ergebnis dieser Funktion ist dann ein `sql::ResultSet`, welches sämtliche gefundenen Einträge darstellt. Das ResultSet ist für alle Sql - Anfragen einheitlich.

9.2.3 Antworten der Datenbank

Die Verwendung der Ergebnisse ist in beiden Fällen relativ simpel. MySql++ liefert Objekte mit öffentlich zugänglichen Variablen, welche so verwendet werden können. In einer Iterator - Schleife kann über alle Einträge iteriert werden.

In JDBC setzt ein Funktionsaufruf, auf dem ResultSet den nächsten Eintrag. Falls dieser Aufruf erfolgreich ist, stellt das ResultSet - Objekt nun die nächste Zeile dar, während der Funktionsaufruf selbst TRUE zurück gibt. Schlägt der Aufruf fehl, falls beispielsweise keine weiteren Einträge existieren, gibt die Funktion FALSE zurück. Diese Funktion ist `next` und wird in einer WHILE Schleife verwendet. Zur Verwendung einzelner Einträge wird auf dem ResultSet `getInt`, `getDouble`, etc. aufgerufen. Diesen Funktionen wird entweder die Spaltenposition, beginnend bei Null, mitgeteilt, oder der Name der Spalte, als String. Der Rückgabewert der Funktion ist der entsprechende Eintrag.

9.3 Problematik und daraus resultierende Ablehnung der Modernisierung

Zu Beginn dieses Kapitels wird in der Machbarkeitsanalyse demonstriert, dass mit MySql Connector Version 8 eine Verbindung zu der verwendeten Datenbank aufgebaut werden kann.

Sobald die Änderungen entsprechend der geänderten Syntax im DataManager realisiert werden ergibt sich allerdings das folgende Problem: Die Ausführung von LandSHIFT wird terminiert. Die entsprechende Fehlermeldung kommt aus dem DataManager, genauer gesagt aus dem Verbindungsaufbau(Siehe Abbildung 3). Der ausgegebene Fehler ist „Access denied“ und „using password: NO“. Der Zweite Teil dieser Fehlermeldung ist hierbei von Bedeutung: er besagt, dass kein Passwort übertragen wird, obwohl ein solches benötigt wird. Würde hier ein Passwort übermittelt, welches allerdings falsch ist, wird eine ähnliche Fehlermeldung ausgegeben. Der Unterschied ist dass dann „using password: YES“ ausgegeben wird. Dieser Unterschied soll mit Abbildung 4 verdeutlicht werden.

Tatsächlich kann mit MySql Connector eine Verbindung aufgebaut werden, ebenso kann mit der LandSHIFT - Konfigurationsdatei, beziehungsweise den darin enthaltenen Anmeldedaten, eine Verbindung aufgebaut werden.

Nur mit der Kombination aus MySql - Connector und LandSHIFT schlägt die Anmeldung fehl. Wie es sein kann, dass der Verbindungsaufbau in einer Anwendung fehlerfrei funktioniert und in einer anderen Anwendung, mit identischen Eingaben, fehlschlägt, ist ungeklärt. Aus diesem Grund wurde diese Änderung verworfen.

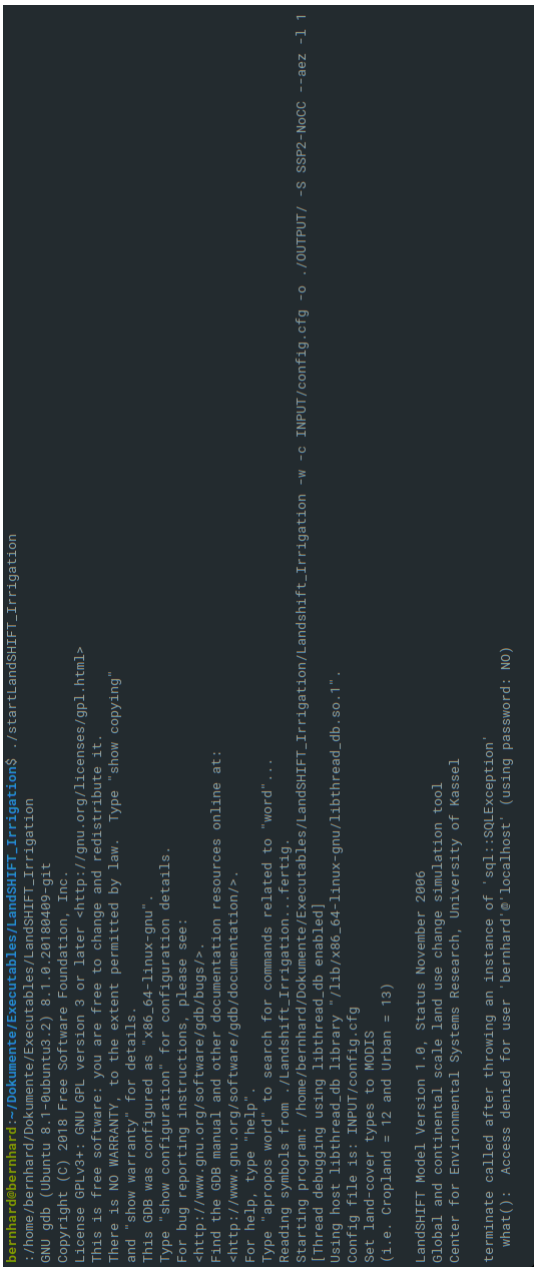

Abbildung 3: Fehlermeldung von LandSHIFT

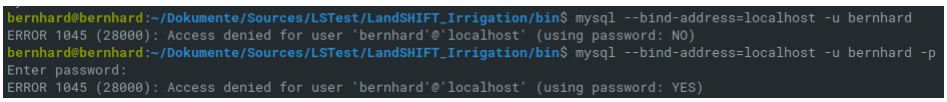

Abbildung 4: Vergleich der Fehlermeldungen; Ohne Passwort und mit Falschem

10 Zusammenfassung

Das Ziel dieser Arbeit sollte eine möglichst fein granulare Ausführung der Anwendungen LandSHIFT, Irrigation - Modul sowie des Skriptes sein. Zur Umsetzung dieses Ziels wurde zunächst ein bereits existierendes, jedoch unbekanntes, R - Skript in C++ implementiert. Inklusive der Entwicklung einer IO - Bibliothek und einer langwierigen Fehlersuche zur Validierung der Ergebnisse hat dieser Anteil die meiste Bearbeitungszeit vereinnahmt. Zur Ergebnisvalidierung musste zusätzlich eine Testanwendung implementiert werden. Diese Anwendung kann inzwischen allerdings als obsolet betrachtet werden, da die Funktionsweise von AlternativeWatergap so modifiziert wurde, dass die Testanwendung nicht länger korrekte Ausgaben produzieren kann.

Im Anschluss an diese relativ langwierige Umsetzung wurde dann eine mögliche Kopplung aller Anwendungen angestrebt. Hierfür sollten möglichst wenige Änderungen an den gegebenen Quellen durchgeführt werden. Der Hintergrund dieses Ansatzes besteht darin, dass durch möglichst kleine Änderungen die Korrektheit der Anwendungen gegeben bleiben sollte. Dementsprechend wurde in LandSHIFT lediglich die Verwendung einer zusätzlichen Datenstruktur implementiert. Diese Datenstruktur musste zunächst entworfen werden.

Diese Datenstruktur musste ebenfalls im Irrigation - Modul eingebaut werden, um einige Eingaben zur Verfügung zu stellen, welche sonst aus Dateien kämen. Zusätzlich musste die Funktionsweise des Moduls modifiziert werden, da die Anwendung standardmäßig nur kontinentale Eingaben verwenden kann. Da dies nicht dem Charakter der geplanten Implementierung entspricht wurden die intern verwendeten Listen so angepasst, dass sie Daten Länderweise verwenden können sollten.

Im Anschluss wurde AlternativeWatergap ebenso modifiziert. Durch diese Modifikation wurde die Anwendung in einen Zustand versetzt, dass sie ebenfalls die eigens entwickelte Datenstruktur als Eingabe verwenden kann. Da ein solcher Anwendungsfall von Beginn an geplant war, konnte dies problemlos umgesetzt werden.

Da diese Arbeit zukünftig von anderen Personen weitergeführt werden soll wurde bei der Entwicklung stets darauf geachtet sämtliche selbst entwickelten Funktionen zu dokumentieren. Ausgenommen von der Dokumentation sind lediglich einzeilige inline Funktionen sowie Konstruktoren, welche ebenso selbsterklärend sind. Die Dokumentation insgesamt ist vollständig Doxygen konform.

Zusätzlich zu den tatsächlich angewandten Modifikationen wurde eine Modernisierung der in LandSHIFT verwendeten Datenbankschnittstelle angestrebt. Nach einer erfolgreichen Machbarkeitsanalyse sowie einer entspre-

chenden Implementierung wurde diese Modernisierung allerdings verworfen, da unvorhergesehen Probleme mit dieser Modernisierung auftraten.

Tabellenverzeichnis

Abbildungsverzeichnis