School-Planner

Raphael Ritter / Roman Stoffel

Bachelorarbeit

Version 2.14

School-Planner 

Projektleiter	22.02.2010	Raphael Ritter / Roman Stoffel
Betreuer	22.02.2010	Josef Joller / HSR
Experte	22.02.2010	Matthias Lips

Dokumentenverwaltung

Dokumenthistorie

Version	Status	Datum	Verantwortlicher	Änderungsgrund
1.0	In Bearbeitung	22.09.09	R. Ritter	Projektbeginn
1.1	Überarbeitet	22.09.09	Team	Für erstes Meeting Überarbeitet
1.2	Updated	16.10.09	R. Ritter	Auf aktuellen Stand bringen
1.3	Überarbeitet	01.11.09	R. Ritter	Dokumentation Algorithmus
1.4	Ergänzt	08.12.09	R. Stoffel	Rating, allgemeine Aspekte hinzufügt
1.5	Updated	09.12.09	R. Ritter	Algorithmus Dokumentation überarbeitet
1.6	Überarbeitet	15.12.09	R. Stoffel	Korrekturen Rechtschreibung
1.7	Final	17.12.09	R. Ritter	Layouten
2.1	Updated	01.03.2010	R. Ritter	Beschreibung der Raters
2.2	Updated	31.03.2010	R. Ritter	Dokumentation heuristischer Algorithmus
2.3	Ergänzt	08.04.2010	R. Stoffel	Dokumentation genetischer Algorithmus
2.4	Ergänzt	12.04.2010	R. Ritter	Dokumentation Graphen Algorithmus
2.5	Ergänzt	15.04.2010	R. Ritter	Dokumentation kombinierter Algorithmus
2.6	Ergänzt	27.04.2010	R. Ritter	Einleitung mit Aufgabenstellung
2.7	Ergänzt	15.05.2010	R. Stoffel	Vergleiche Algorithmen
2.8	Überarbeitet	17.05.2010	R. Ritter	Alles überarbeitet
2.9	Ergänzt	25.05.2010	R. Ritter	Dokumentation Exporting
2.10	Ergänzt	28.05.2010	R. Ritter	Einfügen neuer Abschnitte
2.11	Ergänzt	09.06.2010	R. Stoffel	Undo / Redo und Qualitätsmassnahmen
2.12	Überarbeitet	14.06.2010	R. Ritter	Dokument überarbeiten, layouten
2.13	Überarbeitet	15.06.2010	R. Stoffel	Review
2.14	Überarbeitet	16.06.2010	R. Ritter	Abschlusskontrolle

Inhaltsverzeichnis

1 Einführung

Im Folgenden Abschnitt werden die gesetzten Ziele kurz beschrieben. Ausserdem wird explizit festgehalten was nicht zu den Zielen dieser Bachelorarbeit gehört.

1.1 Ziele

Viele Schulen im Primar- und Oberstufenbereich arbeiten heute noch mit Excel für die Stundenplanerstellung. Dies führt oft dazu, dass viel Zeit verbraucht wird um funktionierende Stundenpläne zu erstellen. Es gibt einige Software für die Stundenplanerstellung. Diese erfüllen aber grösstenteils nicht die Bedürfnisse von kleineren Schulen, sondern sind für grosse Schulen ausgelegt. Daher sollte eine Software erstellt werden, welche die Bedürfnisse kleiner Schulen abdeckt. Zusätzlich sollten verschiedene Algorithmen für die voll automatische Erstellung von Stundenplänen ausgetestet werden. Damit diese auch anhand fester Kriterien verglichen werden können, musste ein Bewertungsframework aufgebaut werden.

1.1.1 Applikation

Am Ende der Arbeit soll eine Applikation vorhanden sein, mit der Stundenpläne für Primar- und Oberstufenschulen erstellt werden können. Es sollen alle benötigten Stammdaten für die Algorithmen möglichst intuitiv eingegeben werden können. Ausserdem soll auch das komplett manuelle erstellen von Stundenplänen mit modernem Userinterface möglich sein.

Das manipulieren von Stundenplänen soll von der Applikation unterstützt werden indem anfallende Probleme wenn immer möglich visuell anzeigt werden.

1.1.2 Framework

Es soll ein Framework aufgebaut werden, in das neue Algorithmen eingefügt und ausgeführt werden können. Das Framework soll zudem eine direkte Schnittstelle zur ‚School-Planner' Applikation haben. Das heisst die Resultate der Algorithmen sollen in der Applikation dargestellt und mutiert werden können.

Das Framework stellt auch die Datenstrukturen zur Verfügung, die ein Algorithmus verwenden kann. Diese Daten sollen mit Hilfe der Applikation erfasst werden können.

Es soll in dieser Dokumentation so detailliert wie nötig festgehalten werden, wie neue Algorithmen in das Framework eingefügt werden können.

1.1.3 Algorithmen

Es sollen verschiedene Algorithmen implementiert werden. Ziel dieser Algorithmen ist es nicht ein hervorragendes Resultat zu erreichen, da der Zeitaufwand für die Optimierung der Algorithmen viel zu gross wäre. Man weiss, dass gerade generelle Ansätze ohne zusätzlich eingebaute Heuristiken nur mässige Ergebnisse liefern werden. Es soll daher analysiert werden, welcher Ansatz welche Probleme gut löst. Danach werden Schlussfolgerungen gezogen wie man diesen Ansatz ergänzen müsste um bessere Resultate zu erreichen.

Zusätzlich soll auch die Möglichkeit untersucht werden, wie einzelne Algorithmen kombiniert werden könnten. Falls genügend Zeit vorhanden ist, soll ein kombinierter Algorithmus implementiert werden.

1.1.4 Bewertung

Es soll ein Bewertungssystem aufgebaut werden über welches Resultate eines Algorithmus anhand verschiedener Kriterien bewertet werden. Eine Bewertung muss in Excel abgespeichert werden können. Es sollen mindestens fünfzehn verschiedene Bewertungskriterien gefunden und implementiert werden.

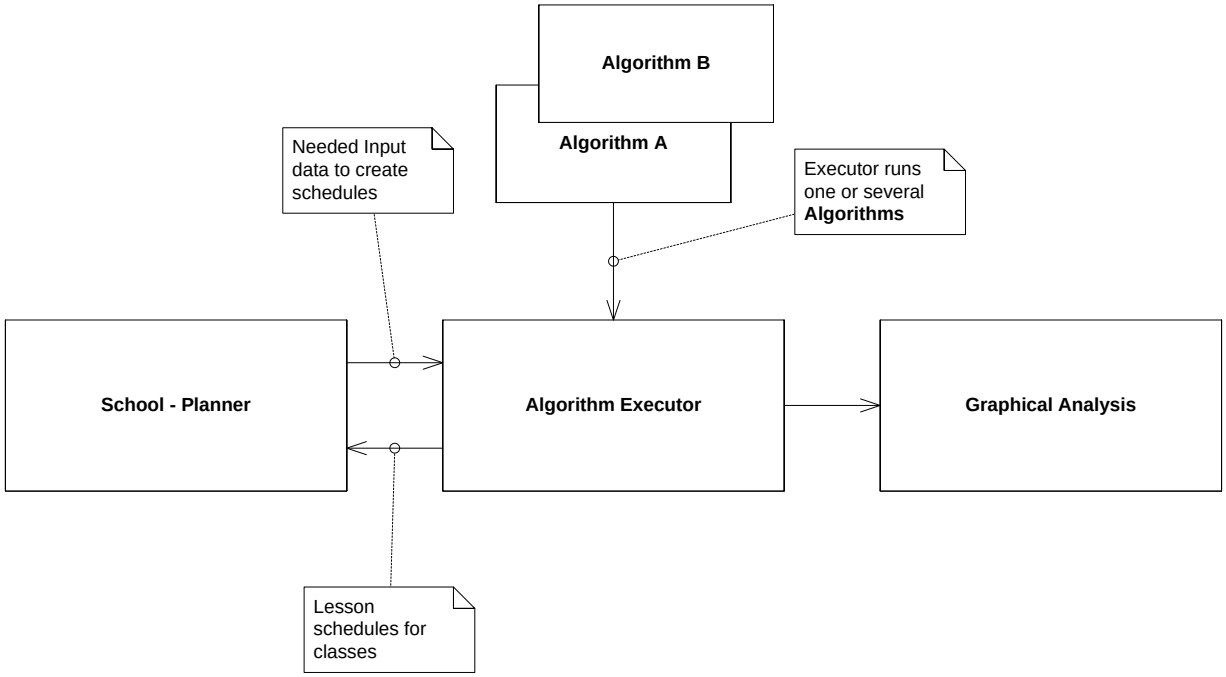
1.1.5 Auswertung

Es sollen verschiedene Testdatenbanken erstellt werden, mit unterschiedlichem Schwierigkeitsgrad. Für jede dieser Testdatenbanken soll eine Referenzdatenbank erstellt werden, mit welcher die Resultate verglichen werden.

Für jeden implementierten Algorithmus soll eine Auswertung für jede Testdatenbank gemacht werden. Diese Auswertung soll detailliert dokumentiert werden. Die Vor- sowie Nachteile des Algorithmus gegen über den anderen Algorithmen soll hervorgehoben werden. Bei allfälligen offensichtlichen Problemen sollen Gründe dafür und mögliche Lösungsansätze aufgezeigt werden.

2 Design

2.1 Allgemein



2.1.1 School – Planner

- Die Stundenpläne können grafisch angesehen und vom User angepasst werden.
- Die ganze Stammdatenverwaltung findet hier statt.
- Spezielle Konfigurationen, die nur Algorithmen benötigt, können hier erstellt werden.
- Der Output des Modells kann hier angesehen werden.

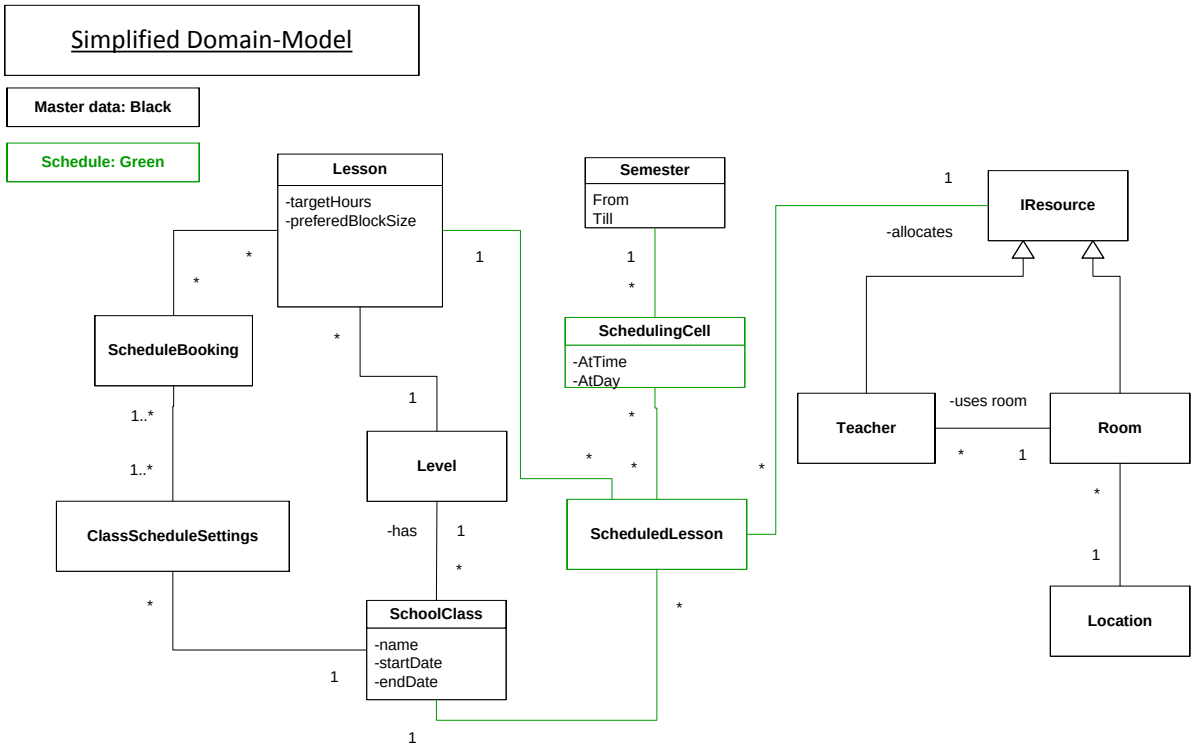
2.1.2 Algorithm Executor

- Führt verschiedene Stundenplan-Erstellungs-Algorithmen aus.

2.1.3 Grafische Analyse

- Ergebnisse der automatischen Stundenplan Erstellung werden anhand von Referenz Stundenplänen verglichen.
- Es gibt verschiedene Bewertungs-Algorithmen mit unterschiedlichem Fokus (z.B. Bewertung aus Sicht des Schülers oder Bewertung der Tage anhand spezifischer Kriterien).

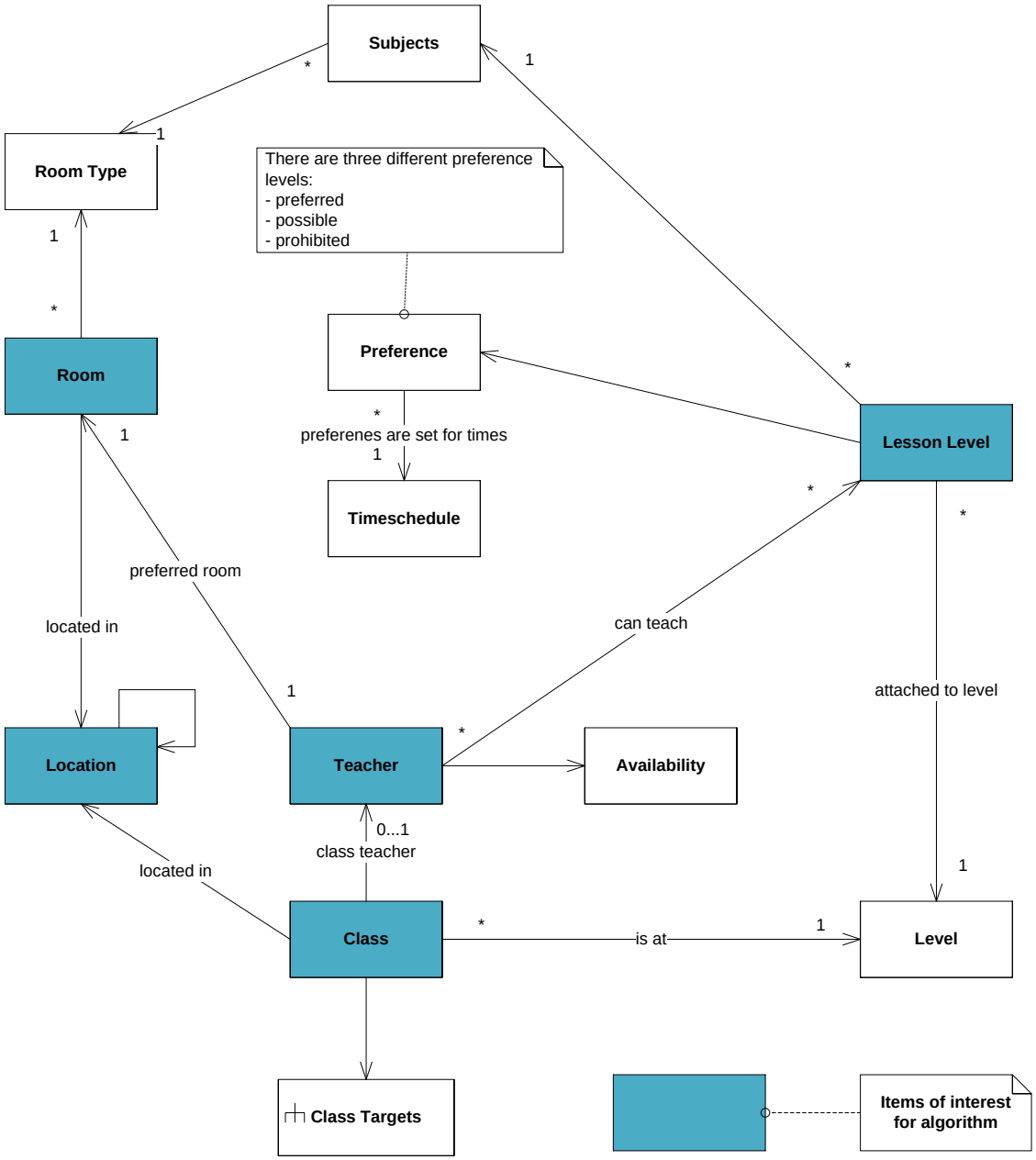
2.2 Vereinfachtes Domain Modell



Hier sieht man ein stark vereinfachtes Domain-Model. Wir haben zwei wichtige Teile. Zuerst sind die Stammdaten (schwarz), welche erfasst werden. Die wichtigsten sind die Lektionen (Lesson). Eine Lektion hat eine bestimmte Stufe (Level), z.B. Erste Primarstufe. Man kann für eine Klasse (SchoolClass) zusätzliche Einstellung machen (ClassSchedulingSetting) wie die Lektionen zu verteilen sind (ScheduleBooking). Ebenfalls zu den Stammdaten gehören die Ressourcen wie Lehrer und Räume.

Der zweite Teil ist sind die wirklichen Stundenpläne (grün). Pro Semester gibt es ein Raster aus Stundenplan-Zellen (SchedulingCell). Nun werden die Lektionen auf die entsprechenden Zellen auf den Stundenplan ausgelegt (ScheduledLesson). Dabei wird auch festgelegt, welche Ressourcen benutzt werden.

2.3 Schnittstellen



Damit ein Stundenplan-Algorithmus seine Planung durchführen kann braucht es relativ viele Informationen.

Folgende Fragen müssen beantwortet werden können:

- Was für Räume habe ich?
- Was für einen Typ haben diese Räume (Werkstatt, Turnhalle,…)?
- Was für Lehrer gibt es?
- Was für Fächer (Deutsch, Mathe, ...) können diese Lehrer unterrichten?
- Wann ist der Lehrer überhaupt verfügbar um zu unterrichten?
- Wann sind die bevorzugten Unterrichtszeiten für ein Fach (z.B. über Mittag für Haushalts-Fach)?
- Wann dürfen bestimmte Lektionen nicht unterrichtet werden?
- Was für Klassen gibt es?
- Wie viele Stunden von welchem Fach soll jede Klasse unterrichtet werden?

2.3.1 Konzepte und Szenarien

Lehrer (Teacher)

Damit ermittelt werden kann wann und wo ein Lehrer eine Klasse unterrichten kann, muss genügend Information über den Lehrer bekannt sein:

Erstens einmal kann nicht jeder Lehrer jedes Fach unterrichten. Gerade auf höheren Schulstufen ist er nur noch für bestimmte Fächer ausgebildet. Desweiteren kann ein Lehrer nur ein bestimmtes Niveau unterrichten. Es genügt nicht zu wissen, dass ein Lehrer X Deutsch und Mathe unterrichtet. Weil er unterrichtet beispielsweise nur 5. oder 6. Klassen in Deutsch. Ein anderes Beispiel wäre bei einem Oberstufen Lehrer. So wird ein Sekundarlehrer Mathematik und Geometrie auf Sekundarstufe unterrichten und nicht auf Realstufe.

Ein zweiter wichtiger Punkt ist, dass ein Lehrer nicht zwangsläufig zu jedem Zeitpunkt an der Schule unterrichten kann. Es ist beispielsweise möglich, dass ein Lehrer nur Teilzeit angestellt ist. Daher unterrichtet er nur am Montag, Dienstag und Donnerstag-Morgen. Dies bedeutet dass er zu allen anderen Zeiten keinen Unterricht halten kann.

Eine weitere Information über einen Lehrer ist sein bevorzugtes Zimmer. Gerade auf der Primar- oder Sekundarstufe ist es normal, dass jeder Lehrer sein fixes Zimmer hat. Dieses verlässt er nur für spezielle Stunden wie Turnen, Werken oder Singen.

Fächer (Subjects), Stufen (Levels) und Lektionen (Lessons)

Ein Fach ist in unserem Fall zum Beispiel Deutsch, Mathematik usw. Bei einem Fach können spezielle Bedingungen gesetzt werden welche anderen Fächer nachfolgen dürfen. Z.B. nach Singen kann ein Beliebiges Fach stattfinden. Aber nach Französisch darf kein Fremdsprachen-Fach wie Englisch folgen.

Eine Stufe bestimmt den Fortschrittsgrad für ein Fach oder ein Klasse. Z.B. sind Erste Primar, Zweite Primar,… Stufen.

Eine Lektion ist ein Fach für eine bestimmte Stufe. Z.B. Deutsch für die erste Primar. Dadurch sind die Anforderungen an den Lehrer genau vorgegeben. Zusätzlich wird bei einer Lektion festgelegt wie viele Stunden pro Woche von diesem Fach für diese Stufe gehalten werden und was für eine Blockgrösse verwendet wird. Die Blockgrösse gibt an wie viele Stunden von dieser Lektion nacheinander gehalten werden. Zum Beispiel wird Deutsch bevorzugt in Doppel-Stunden unterrichtet. Der Normalfall wird sein, dass gewisse Fächer als Doppelstunden unterrichtet werden, falls dies möglich ist.

Zeitausprägung (Preferences)

An vielen Schulen ist es üblich, dass es für Lektionen bevorzugte Zeiten gibt. So werden anspruchsvolle Stunden wie Deutsch, Mathematik am Morgen unterrichtet. Denn am Morgen ist die Konzentration der Schüler höher. Umgekehrt gibt es Fächer die geeignet für den Nachmittag sind, wie Werken oder Turnen.

Zudem muss die Möglichkeit bestehen gewisse Zeiten zu sperren. So findet beispielsweise in einer Primarschule am Mittwoch-Nachmittag kein Unterricht statt. Dies kann ebenfalls über die Zeitausprägung erreicht werden, in dem man für alle Stunden den Mittwochnachmittag als gesperrt markiert.

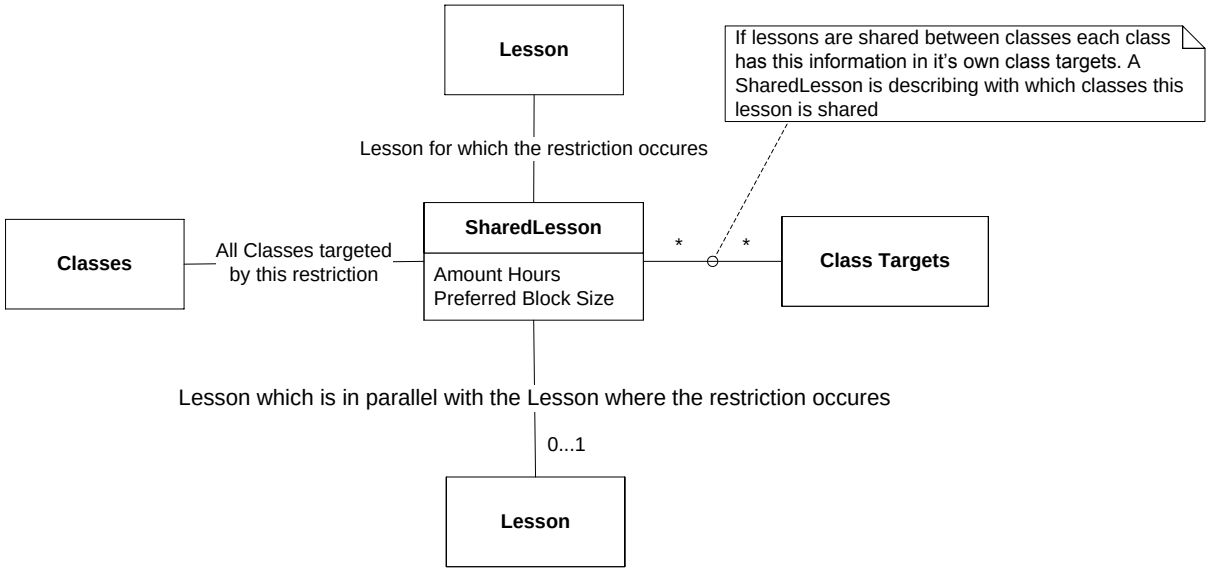
Klassen (Classes)

Eine Klasse befindet sich jeweils auf einer bestimmten Stufe (z.B. Zweite Primarstufe). Anhand dieser Stufe werden die dazugehörigen Lektionen ausgewählt. Beispielsweise hat eine zweite Primar-Klasse vier Deutsch-Stunden. Anhand der Lektionen ist nun ersichtlich, wieviel Stunden für diese Klasse unterrichtet wird. Ausserdem hat eine Klasse noch einen optionalen Klassenlehrer. Dieser übernimmt in der Primarstufe den grössten Teil aller Fächer.

Stunden-Raster (Timeschedule)

Für die Stundenplan-Erstellung muss klar definiert sein an welchen Tagen überhaupt unterrichtet wird. Zudem muss definiert sein wie viel Lektionen pro Tag vorhanden sind und wann diese zeitlich stattfinden. Das heisst man braucht ein leeres Raster, in welchem man dann Lektionen positionieren kann.

2.3.2 Klassenziele (Class Targets)



Ein Problem ist die Tatsache, dass es Stunden gibt, die parallel ablaufen. So ist es an vielen Schulen üblich, dass Fächer wie Handarbeit und Werken parallel stattfinden. Ein weiteres Beispiel ist der Turnunterricht, bei dem die Klassen nach Geschlecht aufgeteilt werden. Um dies zu erreichen gibt es die Möglichkeit festzulegen, dass eine Lektion parallel zu anderen Lektionen läuft. Es wird generell aber nicht der Normalfall sein, das Stunden parallel ablaufen.

Ein weiterer Fall ist, dass verschiedene Klassen zusammen genommen werden, z.B. bei Fächern wie Turnen, Singen, Werken. Das heisst es findet eigentlich nur eine Singstunde statt, aber mehrere Klassen besuchen diese gleichzeitig. Diese Situation wird aufgelöst in dem mehre Klassen auf dieselbe Restriktion referenzieren. So ist es möglich, dass Klasse A, B und C für den Singunterricht dasselbe ‚SharedLesson'-item verwenden. Dadurch wird dann von den Ressourcen her gesehen auch nur ein Lehrer und ein Raum für den Singunterricht benötigt. Dieser Unterricht findet auch automatisch für alle Klassen mit dieser speziellen Restriktion zum selben Zeitpunkt statt.

Alle normalen Lektionen sowie alle spezial Fälle werden als ‚Booking' repräsentiert. Ein ‚Booking' enhält die Informationen über die Lektion, die Klasse sowie über die speziellen Einschränkungen.

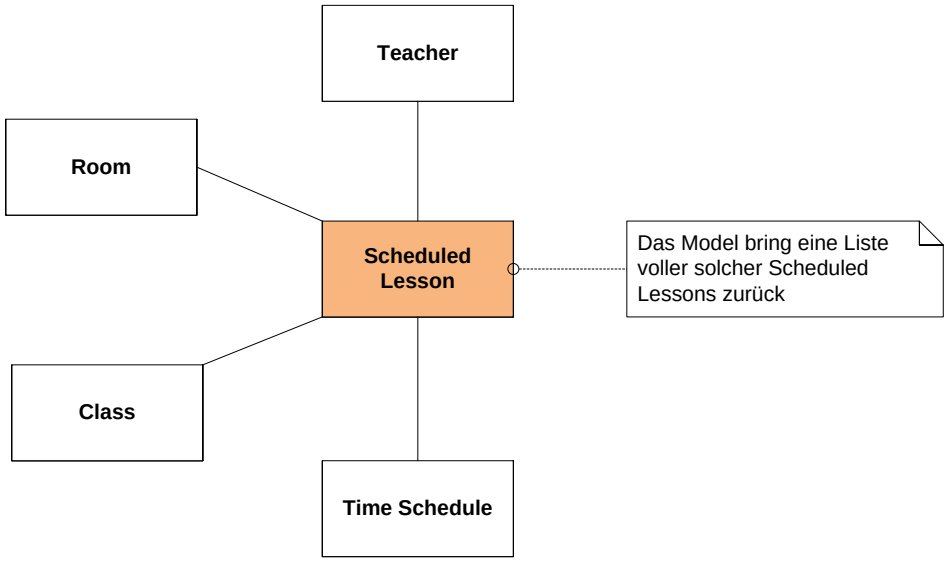
2.3.3 Standorte (Locations)

Viele Schulen haben ihre Unterrichtsgebäude auf verschiedene Standorte verteilt. Dies hat einen Einfluss auf den Stundenplan, denn man will die Anzahl Standortwechsel möglichst gering halten. Das heisst es wird für die Klasse und für die Räume festgelegt, wo sich diese befinden. Zusätzlich muss auch eine Standortabfolge angegeben werden. Dies dient dazu den nächsten Ersatz-Standort zu finden, falls ein Standort die Anforderung nicht erfüllen kann.

Wir haben zum Beispiel eine Schule mit drei Standorten (A, B, C). Nur Standort A und B haben einen Werkraum und eine Turnhalle. Dies bedeutet, dass alle Klassen die dem Standort C zugeteilt sind, entweder am Standort A oder B turnen und werken müssen. Standort A liegt jedoch deutlich näher und deswegen werden wenn immer möglich die Räumlichkeiten des Standorts A bevorzugt. Dies wird durch die Nachfolgehierarchie von Standort C festgelegt. Dort wird festgelegt, dass Standort A gegenüber Standort B bevorzugt wird. Zudem ist sichergestellt,

dass für alle Lektionen für welche am Standort der Klasse Räumlichkeiten vorhanden sind auch diese genutzt werden. Damit wird verhindert, dass nicht jede Lektion an einem anderen Standort stattfindet.

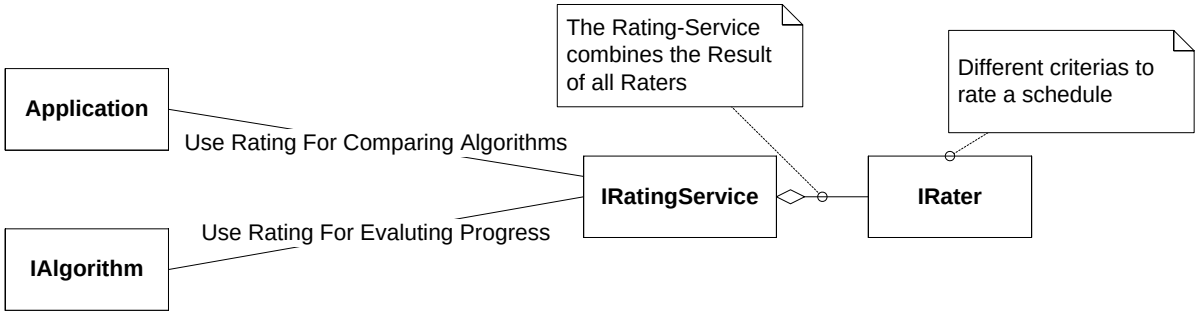
2.3.4 Algorithmen Resultat



Das Ergebnis eines Algorithmus besteht immer aus einer Ansammlung von solcher ‚Scheduled Lessons'. Jede ‚Scheduled Lesson' beinhaltet alle Informationen die nötig sind, um daraus wieder einen Stundenplan zusammenzustellen, da jede Schulstunde nur durch vier verschiedenen Eigenschaften identifiziert wird:

- Klasse, zu der die gebuchte Lektion gehört.
- Raum, in welchem die Lektion stattfindet.
- Lehrer, der die Lektion abhält.
- Zeitpunkt, sowie Tag an dem die Lektion stattfindet.

2.4 Bewertungsalgorithmen



Die Bewertungs-Algorithmen haben zwei Zwecke. Der erste ist für die Algorithmen selber. Algorithmen können das Bewertungs-System aufrufen, um den aktuellen Fortschritt zu prüfen. Der zweite Zweck ist das Vergleichen von verschiedenen Algorithmen. Damit ein Vergleich möglich ist, müssen harte Bewertungs-Kriterien erstellt werden.

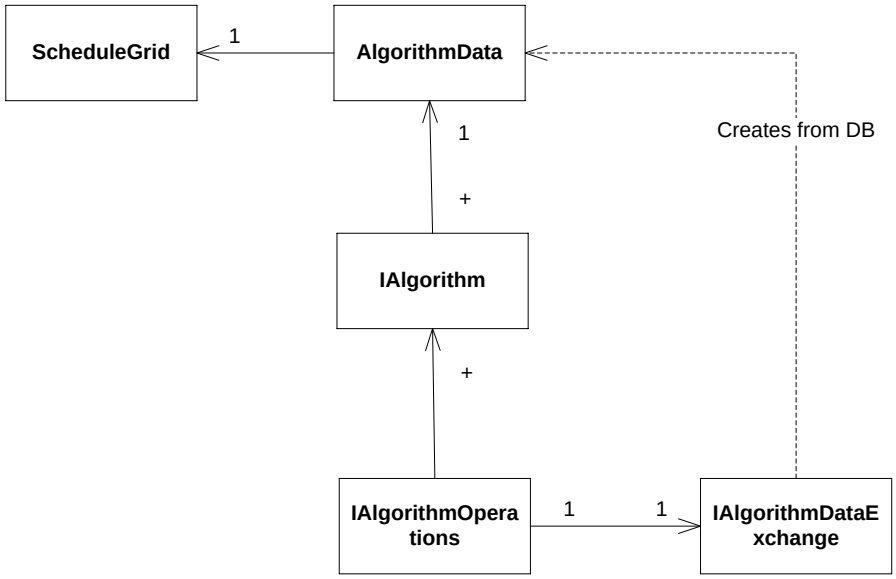
Es gibt extrem viele Aspekte, die bewertet werden können. Daher gibt es ein kleines Bewertungs-Framework. Dieses erlaubt es beliebig viele Bewertungs-Kriterien einzubinden.

Es gibt zwei Bewertungs-Arten.

- Tageswertung: Diese bewertet wie gut ein Tag geplant ist.
- Gesamt-Wertung: Diese beurteilt den gesamten Stundenplan.

3 Framework für Algorithmen

3.1 Statics



3.1.1 AlgorithmData

Hier sind einerseits die Schnittstellen-Informationen gehalten. Dies wären alle Lehrer, Räume, Klassen etc. Es gibt zusätzlich auch Komfortfunktionen, um beispielsweise alle Räume für einen bestimmten Raumtyp zu bekommen. Alle Daten in der ‚AlgorithmData' Klasse sind unveränderlich und können nur gelesen werden. Diese Daten stammen von der School-Planner Applikation und stellen den Algorithmus-Input dar.

3.1.2 IAlgorithmdataExchange

Hier werden aus der Datenbank alle initialen Informationen ausgelesen, welche für einen Algorithmus interessant sind. Ausserdem liegt auch hier die Verantwortlichkeit die Daten wieder zurück in die Datenbank zu schreiben. Es ist also die Schnittstelle für die Daten zwischen dem Algorithmen Framework und der School-Planner Applikation.

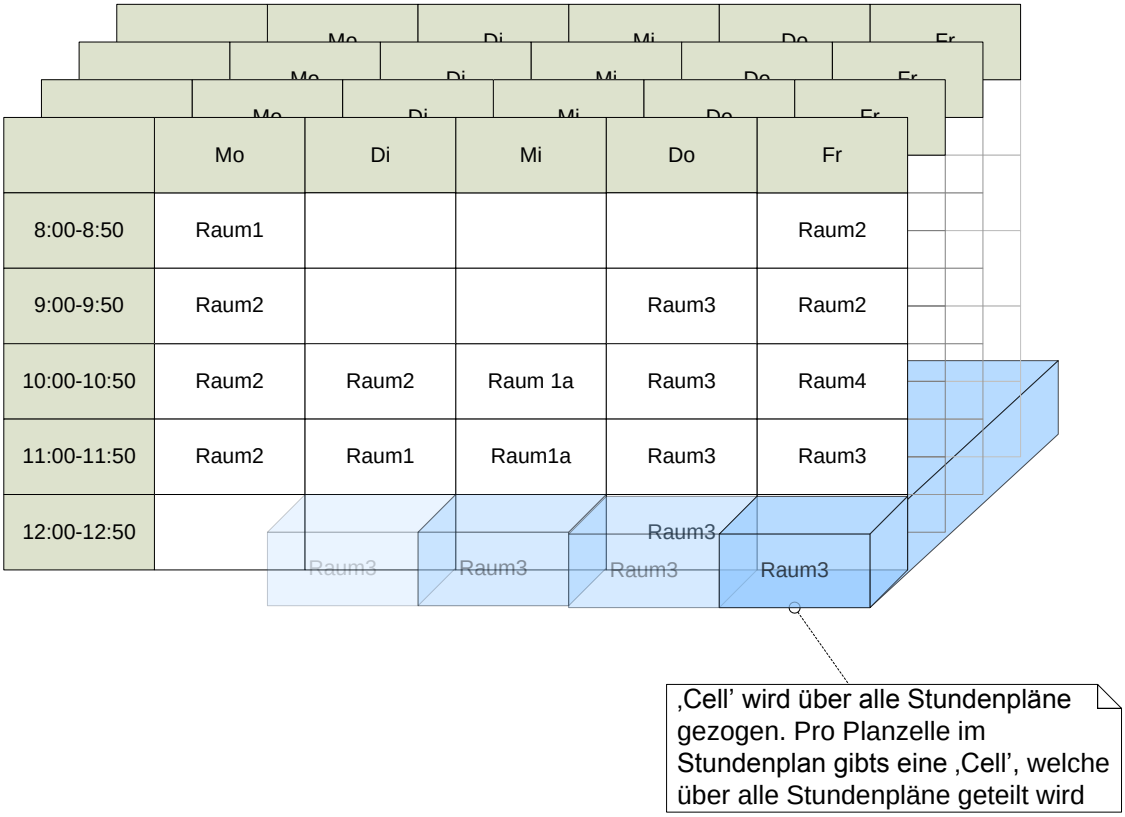
3.1.3 IAlgorithmOperations

‚IAlgorithmOperations' ist die Facade für die ganze Algorithmen Logik. Die Facade wird von der School-Planner Applikation verwendet um das vom User ausgelöste automatische Erstellen des Stundenplans zu starten. Dabei wird in einem ersten Schritt zuerst mit Hilfe des ‚IAlgorithmDataExchange' Klasse die Ausgangsdaten (‚AlgorithmData') zusammengetragen und nachher die Run-Methode des spezifischen Algorithmus aufgerufen. Sobald der Algorithmus fertig ist, werden dann die Daten wieder zurück geschrieben.

3.1.4 IAlgorithm

Die Run Methode dieses Interfaces wird aufgerufen, falls der User einen bestimmten Algorithmus ausführen möchte.

3.1.5 Studenplan (ScheduleGrid)

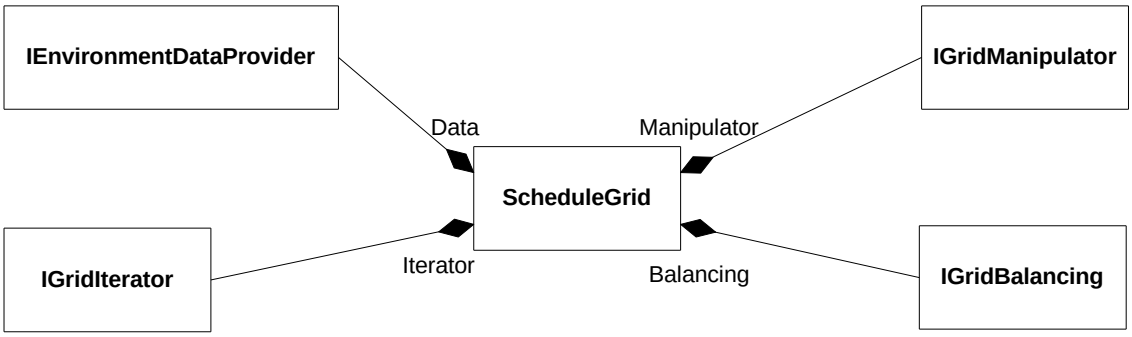


Diese zweidimensionale Tabelle (Tage, Zeiten) beinhaltet Stundenplan-Zellen. Auf diesen Zellen können dann Stunden für eine Klasse gebucht werden. Ausserdem bietet die Zelle auch die Möglichkeit abzufragen, ob gewisse Ressourcen bereits verwendet sind und daher ein Konflikt entsteht oder nicht.

Das heisst eine Zelle merkt sich einerseits alle gebuchten Stunden für alle Klassen, sowie alle Ressourcen, die auf dieser Zelle bereits verbraucht wurden.

Das ‚ScheduleGrid' stellt ein Blackboard dar (siehe nachfolgendes Kapitel). Nur auf dem ‚ScheduleGrid' können Daten verändert werden.

Um mit dem Blackboard zu arbeiten gibt es verschiedene Hilfsschnittstellen, die dies ermöglichen. Das Blackboard stellt diese Helfer jedem Client zu Verfügung.



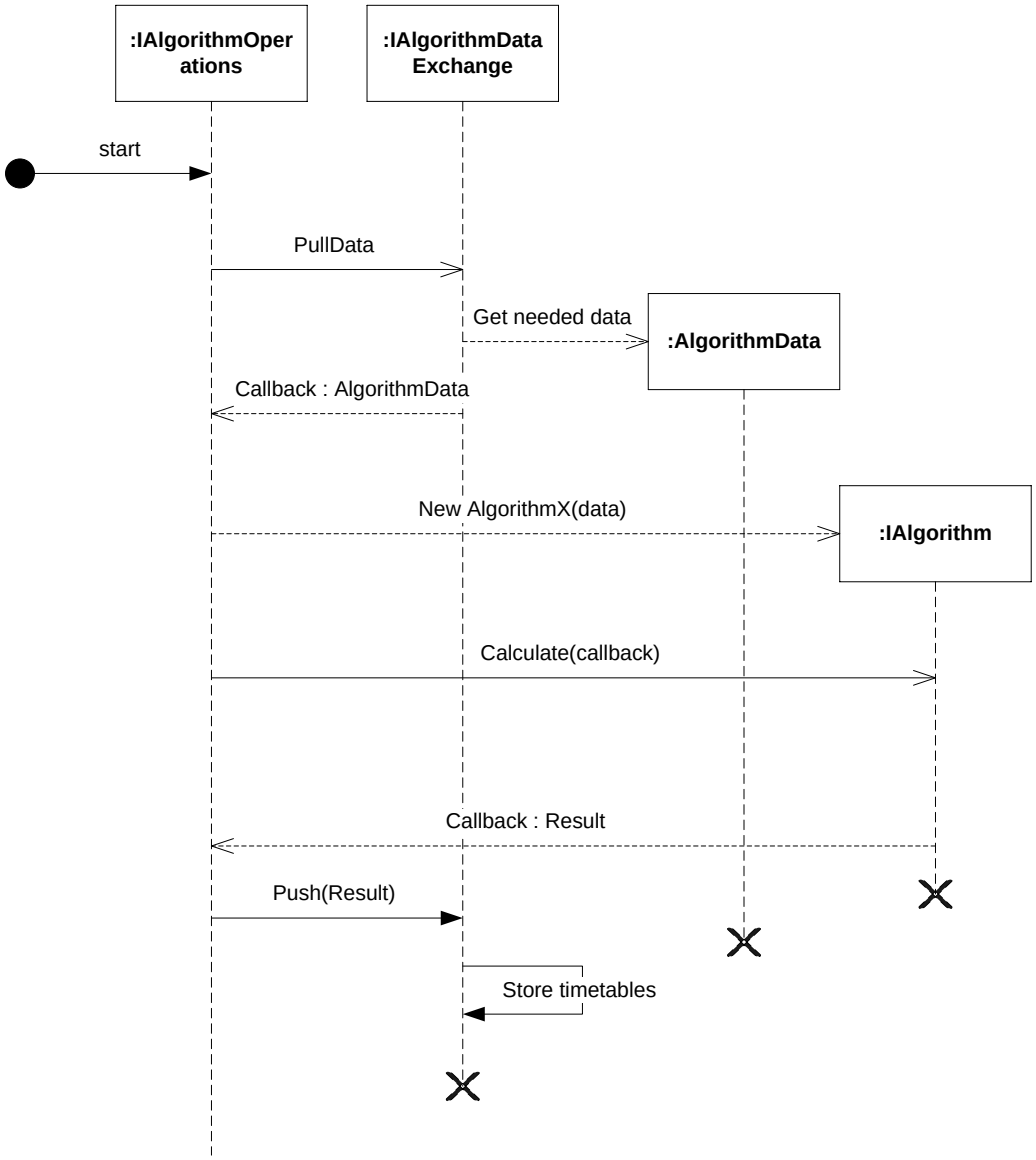
Das **IEnvironmentDataProvider** Interface stellt die Funktionalität zur Verfügung, um mögliche Ressourcen für eine bestimmte Lektion abzufragen.

Der **IGridIterator** stellt verschiedene Möglichkeiten zur Verfügung über das ‚ScheduleGrid' zu iterieren.

Der **IGridManipulator** gibt dem Client die Möglichkeit neue Stunden in das ‚ScheduleGrid' einzutragen oder bereits eingetragene Stunden zu löschen.

Die **IGridBalancing** Schnittstelle kann verwendet werden um herauszufinden wie viele Stunden und Bookings pro Tag eine Klasse hat, sowie um Tagesstundenpläne einer Klasse abzufragen.

3.2 Dynamics



Um einen Algorithmus verwenden zu können, muss der User zuerst alle benötigten Stammdaten erfassen. Ist dies getan, wählt der Benutzer im Algorithmen Menu den Algorithmus aus, den er für die Stundenplangenerierung benutzen will. Sobald der Benutzer dort einen Algorithmus ausgewählt hat, ruft die Applikation die Facade des Algorithmus-Frameworks auf (‚IAlgorithmOperations'). Dort werden nun zuerst über einen Data Provider (‚IAlgorithmDataExchange') alle erfassten Stammdaten von der Datenbank gelesen und in einem Datenkonstrukt abgespeichert (‚AlgorithmData'). Danach wird von der Facade der gewünschte Algorithmus mit den ausgelesenen Daten aufgerufen.

Sobald der Algorithmus fertig ist mit seinen Berechnungen, wird das berechnete Resultat an die Facade zurückgegeben.

Die Facade ruft nun den Data Provider auf, um das berechnete Resultat in die Datenbank, in Form von ausgefüllten Stundenplänen, zurück zu schreiben.

4 Heuristischer Algorithmus

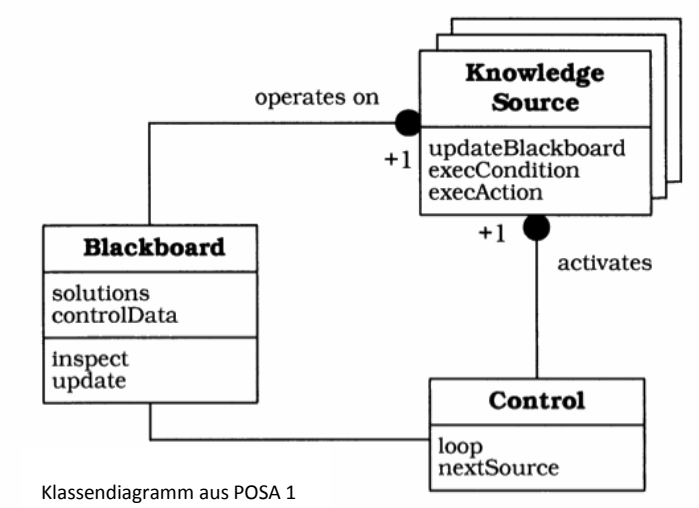
Bei der Implementierung haben wir uns vom Blackboard Pattern (POSA 1) inspirieren lassen. Deswegen wird im folgenden Abschnitt dieses Pattern kurz erläutert. Bei der anschliessenden Erklärung unseres Algorithmus werden Referenzen zu diesem Pattern gemacht. Dadurch kann deren Funktion sehr klar beschrieben werden.

4.1 Blackboard Pattern (POSA 1)

Eine Sammlung von unabhängigen Programmen (Knowledge Sources / Experts) arbeiten zusammen an einer gemeinsamen Datenstruktur (Blackboard). Jedes Programm ist dabei spezialisiert auf sein Teilgebiet. Die einzelnen Programme rufen sich weder gegenseitig auf, noch ist eine bestimmte Reihenfolge der Einsätze der Programme festgelegt.

Ein zentraler Kontrollmechanismus (Control Component / Moderator) evaluiert den aktuellen Zustand des Lösungsfortschrittes und koordiniert die spezialisierten Programme (Opportunistic Problem Solving). Zwischenlösungen werden von den Experten kombiniert, verändert oder abgewiesen. Der Kontrollmechanismus versucht die Lösungen auf ein höheres Abstraktionslevel zu bringen. Das System kann entweder durch eine Knowledge Source oder die Control Komponente aus folgenden Gründen angehalten werden:

- Es wurde eine akzeptierbare Hypothese für das Problem gefunden.
- Technische oder zeitliche Ressourcen wurden aufgebraucht.



Klassendiagramm aus POSA 1

Blackboard

- Verwaltet die gemeinsamen Daten und stellt sie der Control Component und den Knowledge Sources zur Verfügung.

Knowledge Sources (Experts)

- Condition Part: Evaluiert den aktuellen Zustand des Lösungsfortschrittes.
- Action Part: Produziert Resultate anhand seiner Informationen und schreibt diese auf das Blackboard.

Control Component

- Überwacht das Blackboard.
- Koordiniert, wählt und aktiviert Knowledge Sources anhand von vorgegebenen/heuristischen Strategien.

4.2 Klassen

4.2.1 IPlacingStrategy

Eine Placing-Strategie bekommt Bookings und die AlgorithmData. Sie versucht anschliessend so viele Bookings wie ihr möglich sind sinnvoll zu platzieren und gibt alle unplatzierten Bookings zurück.

4.2.2 IImprovementStrategy

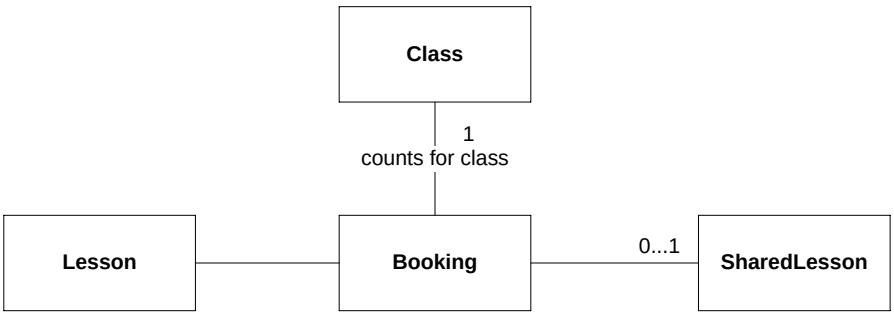
Eine Verbesserungsstrategie manipuliert das Grid, fügt jedoch keine neuen Elemente ein oder löscht Elemente. Das heisst, es werden nur Tauschoperationen zwischen Zellen des Grids durchgeführt.

4.2.3 HeuristicAlgorithm

Der Algorithmus ist nichts anderes als ein Kontroller der verschiedene Placing und Improvement Strategien laufen lässt. Wann er welche Strategie einsetzt ist folgendermassen realisiert:

- PlacingStrategies kommen immer vor ImprovementStrategies
- Strikte Reihenfolge durch Prioritäten unter den Strategien der jeweiligen Kategorie

4.3 Bookings

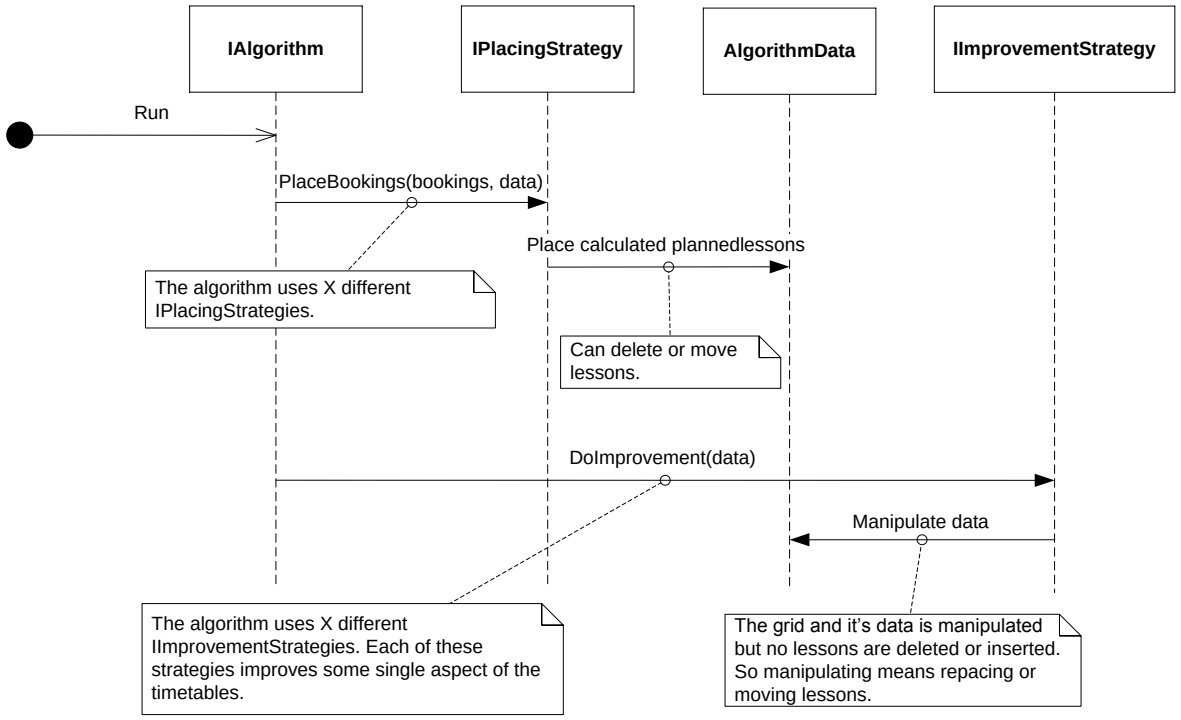


Jede Klasse hat eine Liste von Bookings die aus den Stammdaten berechnet werden können. Die Bookings bestehen einerseits aus der Anzahl Stunden für ein Fach. Diese sind bei den Lektionen definiert und werden zusätzlich ergänzt durch die Klassenziele. Dort werden weitere Restriktionen festgehalten, wie dass Stunde XY parallel zu Stunde YZ stattfindet. Ausserdem liefert ein Booking immer mit für welche Klasse es gilt.

Placing Strategien arbeiten nur noch auf Bookings, da dort alles Nötige festgehalten ist um Stunden zu platzieren. Wenn eine Stunde platziert ist wird das Booking ‚gelöscht'. Die zurück bleibenden Bookings repräsentieren alle Stunden, welche noch zu platzieren sind.

Der ScheduleGrid entspricht dem Blackboard. Die Knowledge Sources bekommen zusätzlich die AlgorithmData. Diese sind unveränderbar, können also nur gelesen werden. Sie enthalten Grund-Information wie Lehrer, Räume, Klassen. Die berechneten Lösungen werden im ScheduleGrid abgespeichert. Dort werden dann auch Lösungen abgeändert oder verworfen.

4.4 Dynamics



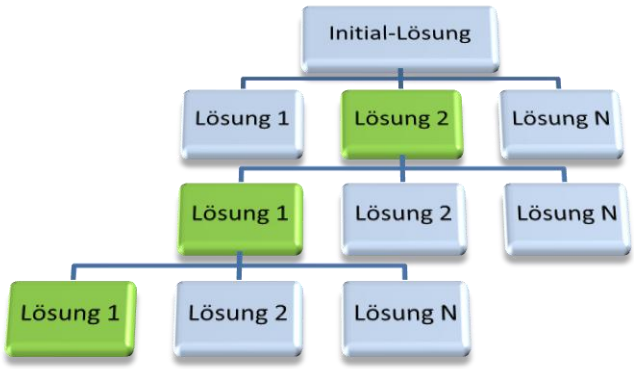
Dieses Diagramm zeigt die Umsetzung. Es entspricht ziemlich genau dem allgemeinen Ablauf der im vorhergehenden Kapitel beschrieben ist. Darum wird hier der Zusammenhang zwischen Blackboard-Pattern und unserer Implementation aufgezeigt.

Die IAlgorithm-Implementierung ist der Kontroller. Er entscheidet, welche Knowledge Source als nächstes angeworfen wird.

Eine IPlacingStrategy und IImprovementStrategy entspricht einer Knowledge Source. Während eine IPlacingStrategy versucht gewisse ‚Bookings' anhand von definierten Kriterien auf dem Blackboard zu positionieren, wird eine IImprovementStrategy lediglich das Blackboard selber manipulieren. Eine IimprovementStrategy wird also keine neuen Daten hinzufügen oder entfernen. Es ist grundsätzlich möglich dass eine Knowledge Source auftretende Konflikte oder andere für die Bewertung ungünstigen Aspekte ignoriert. Diese werden dann von einer anderen Knowledge Source verbessert. D.h. die Knowledge Sources können sehr flexibel aufgesetzt werden. Eine Knowledge Source sollte sich immer nur um einen spezifischen Aspekt kümmern und alle anderen Aspekte soweit als möglich ignorieren.

4.5 Allgemeiner Ablauf

Der Algorithmus setzt zu allererst einen Kontext auf, der Ressourcen wie Lehrer und Räume jeweils einem Fach und einer Klasse assoziiert. Der Algorithmus platziert danach alle Stunden, die mit speziellen Restriktionen versehen sind. Dies wären beispielsweise Stunden, die von mehreren Klassen gleichzeitig besucht werden oder die parallel stattfinden. Ist dies getan werden diese Stunden von Algorithmus ignoriert, das heisst diese Stunden sind fix im Stundenplan platziert.



Nach diesem ersten Schritt wird ein iterativer Ansatz verwendet, um die restlichen Stunden möglichst optimal zu platzieren. Dabei werden wie bei einem genetischen Algorithmus für jede Iteration eine durch die Settings vorgegebene Anzahl Lösungen erstellt. Wobei lediglich die beste Lösung in einer nächsten Iteration als Ausgangspunkt weiter verwendet wird.

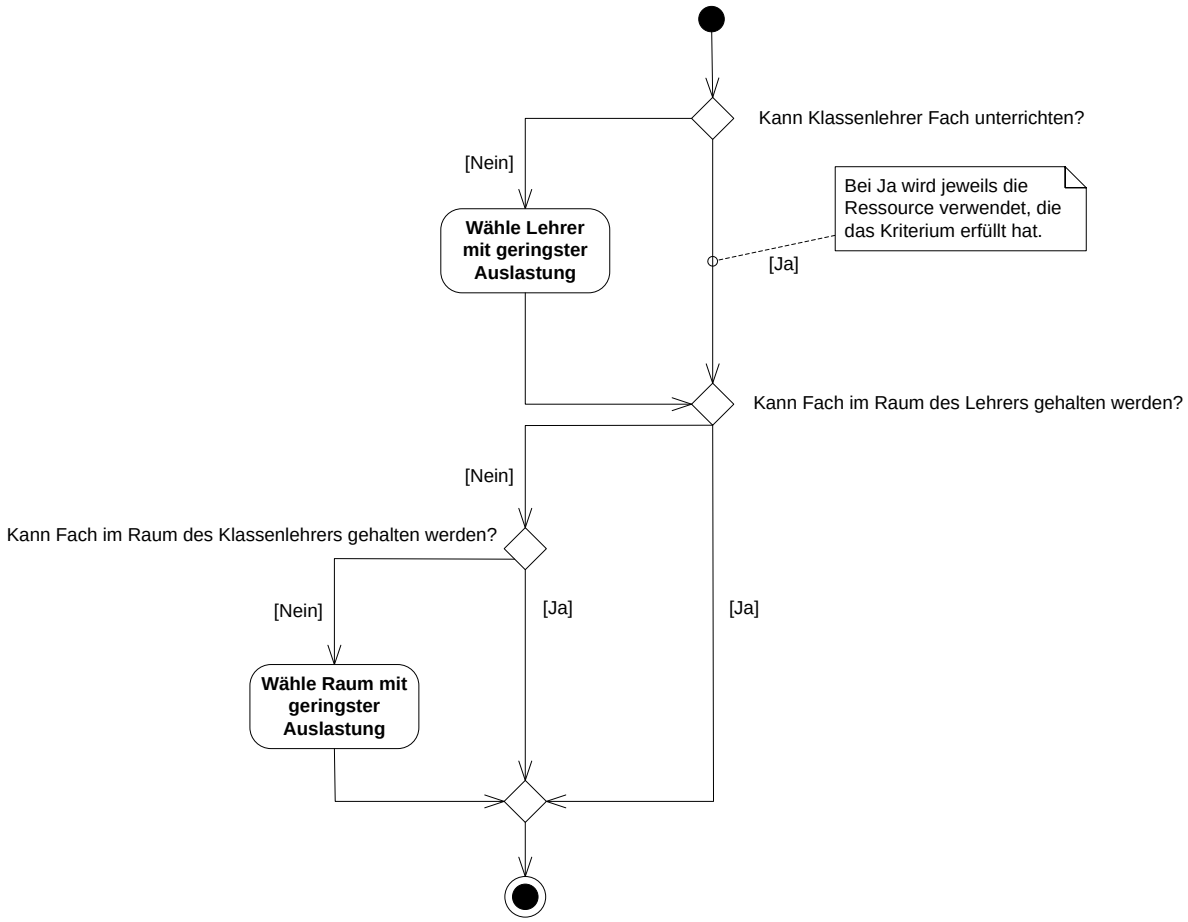
Das Erstellen der einzelnen Lösungen pro Iteration findet parallel statt, wobei das Bestimmen der besten Lösung erst stattfinden kann wenn alle Resultate vorhanden sind.

4.6 Kontext

Der Kontext weist anhand von gewissen Regeln Ressourcen einem effektiven Fachtyp für eine spezifische Klasse zu. Das heisst es werden jedem Fach einer Klasse ein Raum und ein Lehrer zugewiesen. Alle Platzierungsstrategien verwenden den Datenkontext. Die Ressourcen werden also Initial auf die Schulstunden verteilt. Diese Verteilung wird anschliessend nie mehr verändert.

Der Grund weswegen die Verteilung der Ressourcen auf die Schulstunden vorgängig gemacht wird, ist einerseits aus Performancegründen und andererseits um eine möglichst faire Lastverteilung zu erlangen.

Die Ressourcenverteilung findet anhand der folgenden Regeln statt (siehe Diagramm), wobei immer zuerst ein passender Lehrer ausfindig gemacht wird und erst nachher der Raum dazu. Die Rangfolge welche Schulstunden zuerst Lektionen zugeteilt bekommen wird durch das mögliche Konfliktpotential bestimmt.



4.6.1 Konfliktpotential

Damit es nicht passieren kann, dass plötzlich keine Ressource mehr vorhanden ist, um eine Nachfrage zu erfüllen, werden kritischen Stunden zuerst Ressourcen zugewiesen. Würde dies nicht so gemacht, gäbe es zum Beispiel in folgendem Fall Probleme: Man hat zwei Lehrer die Deutsch unterrichten können, aber nur einer von diesen kann auch Mathematik unterrichten. Ohne Ordnung nach Kritikalität wäre es möglich, dass für die Deutschstunde der Lehrer genommen wird, der beides unterrichten kann und diese Ressource dann bereits ausgebucht ist für Mathematik (dasselbe Problem existiert auch für die Räume).

Das mögliche Konfliktpotential wird mit folgender Formel berechnet:

$$\text{KonfliktPontenzial} = \frac{Anzahl\ Lektionen * Preferred\ Size}{Anz.mögliche\ Slots * Anz.ideale\ Slots * Anz.Lehrer * Anz.Räume}$$

4.7 Platzierungsstrategien

Platzierungsstrategien fügen Schulstunden in das Grid ein oder entfernen welche. Es gibt drei verschiedene Platzierungsstrategien, wobei lediglich die beiden Strategien PriorityPlacer und RandomNoiseMaker im iterativen Teil des Alogrithmus verwendet werden. Der RestrictionPlacer wird genau initial einmal ausgeführt und diese Stunden werden anschliessend nicht mehr bewegt.

4.7.1 RestrictionPlacer

Der ‚RestrictionPlacer' plaziert alle Bookings mit speziellen Einschränkungen anhand dieser Einschränkungen. Für die Platzierung wird der Kontext verwendet und dessen Vorgaben, welche Ressourcen verwendet werden sollen. Der RestrictionPlacer wird nur Initial ausgeführt.

4.7.2 PriorityPlacer

Hier werden alle Bookings, die keine Einschränkungen besitzen platziert. Zur Platzierung wird der Kontext verwendet und es wird eine konfliktfreie Position gesucht. Wird keine konfliktfreie Position gefunden, wird die Stunde auch konfliktbehaftet platziert. Sobald diese Strategie durchgelaufen ist, gibt es keine Bookings mehr, die noch platziert werden müssten.

4.7.3 RandomNoiseMaker

Damit beim Iterieren keine Stagnation in einem lokalen Minimum geschieht, muss an der Lösung jeweils etwas gerüttelt werden. Je besser die Lösung ist, umso weniger wird gerüttelt. Das Rütteln geht folgendermassen vor sich:

- Es wird für jeden Tag für jede Klasse eine Zufallszahl berechnet zwischen null bis eins.
- Falls die Zufallszahl höher ist als die Bewertung für den Tag, wird der gesamte Stundenplan der Klasse gelöscht.

Zusätzlich werden die Stundenpläne für alle Klassen entfernt die Konflikte aufweisen. Dies wird getan, weil die Platzierungslogik sehr gut ist für die getesteten Fälle. Bei Konflikten handelt es sich meistens um eine Verknüpfung von verschiedenen ungünstigen Platzierungsentscheidungen und nicht nur um einen einzelnen ungünstigen Platzierungsentscheid. Somit kann ein erneutes Plazieren des gesamten Stundenplanes oft den Konflikt beheben.

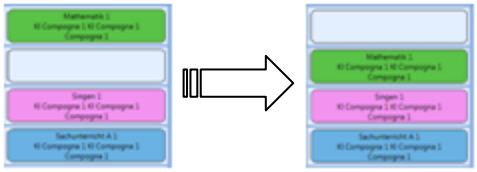
4.8 Verbesserungsstrategien

Verbesserungsstrategien verschieben lediglich bereits platzierte Stunden auf dem Grid. Es werden also keine neuen Stunden hinzugefügt oder platzierte Stunden gelöscht. Ziel der Verbesserungsstrategien ist es optimalere Stundenpläne zu erhalten, da die Platzierungsstrategien verschiedene Aspekte komplett ignorieren wie beispielsweise die Verteilung der Schulstunden eines Faches unter der Woche.

Die Verbesserungsstrategien laufen in einer vorbestimmten Reihenfolge ab. Ansonsten ist es möglich dass die eine Strategie das Resultat einer anderen komplett zunichte macht. Die Reihenfolge ist:

- ‘DayCompacter’, ‘DayBalancer’, ‘LessonBalancer’ / Constraint Resolver, ‘ConflictResolver’

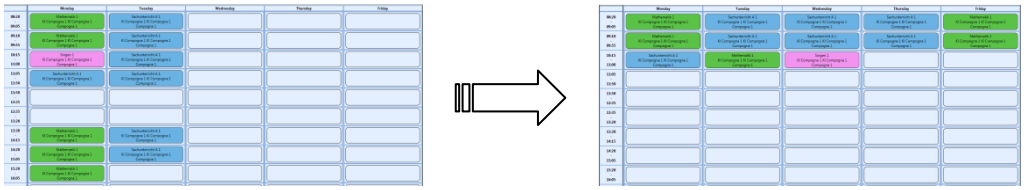
4.8.1 DayCompacter



Der DayCompacter ist dafür verantwortlich mögliche Lücken, die im Stundenplan einer Klasse entstanden sind, zu schliessen. Das heisst, es wird versucht einen lückenfreien Stundenplan zu erhalten (siehe Beispiel). Diese Strategie ist jedoch ungeeignet für Oberstufen, da es dort zusätzliche Komplexität gibt. In Oberstufen gibt es Fächer wie Kochen, die über Mittag stattfinden dürfen, andere hingegen nicht. Dadurch wird diese Strategie in so einem Fall versuchen die Mittagsstunden auch zu füllen und es entstehen unnötig viele Konflikte, die für den ‚ConflictResolver' schwer wieder aufzulösen sind. Daher wird diese Strategie nur für die Primarstufe empfohlen.

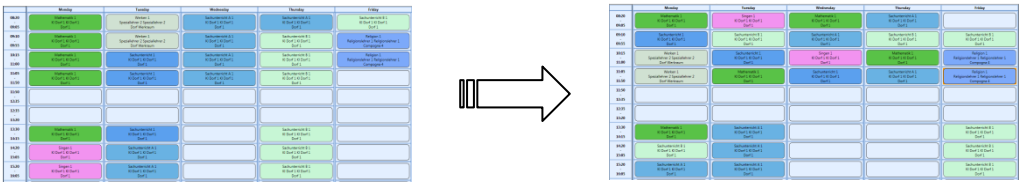
4.8.2 DayBalancer

Das Ziel dieser Strategie ist es die Anzahl Stunden pro Schultag möglichst ausgeglichen zu halten (wobei dies je nachdem gar nicht möglich ist, da beispielsweise viele Schulen am Mittwochnachmittag frei haben). Es werden solange Stunden zwischen den Tagen eines Stundenplans einer Klasse herum geschoben bis die Tage perfekt ausgeglichen sind oder kein weiterer Ausgleich mehr möglich ist.



4.8.3 LessonBalancer

Der LessonBalancer ist dafür verantwortlich eine gute Verteilung der Schulstunden eines Faches (z.B. Deutsch, Mathematik,…) auf die verschiedenen Tage zu erhalten. Um dies zu erreichen überprüft er jeden einzelnen Stundenplan einer Klasse. Dabei schaut diese Strategie ob an einem Tag zu viele Stunden eines Faches abgehalten werden. Falls dies der Fall ist werden die Stunden mit Stunden an einem anderen Tag abgetauscht. Beim Abtausch der Stunden wird auch gleich überprüft, dass nicht neue Konflikte entstehen. Falls ein Abtauschen ohne weitere Verletzungen der Verteilung und ohne neue Konflikte unmöglich ist, wird für diese Stunde kein Abtausch vorgenommen und die ungünstige Balance bleibt bestehen.



4.8.4 ConflictResolver

Diese Strategie löst so viele Konflikte wie möglich auf. Es wird versucht jeden Konflikt in zwei Phasen aufzulösen.

In der ersten Phase wird geschaut ob sich im Stundenplan der Klasse noch irgendwo freie Slots befinden. Sollte dies der Fall sein, so wird überprüft ob an dieser Position die Stunde konfliktfrei eingefügt werden könnte. Sollte dies der Fall sein wird der Konflikt so aufgelöst.

Falls in der ersten Phase der Konflikt nicht aufgelöst werden konnte, wird in einer zweite Phase versucht ob der Konflikt durch das Abtauschen mit einer anderen Stunde aufgelöst werden könnte. Wird auch hier keine Lösung gefunden so wird der Konflikt belassen.

4.8.5 ConstraintResolver

Hier wird überprüft ob wirklich alle Nachfolgebedingungen erfüllt sind, die ein Fach hat. Für alle Verletzungen von solchen Bedingungen wird versucht durch Verschieben dieser Stunden eine Lösung zu erarbeiten, die weniger oder keine Verletzungen mehr aufweist.

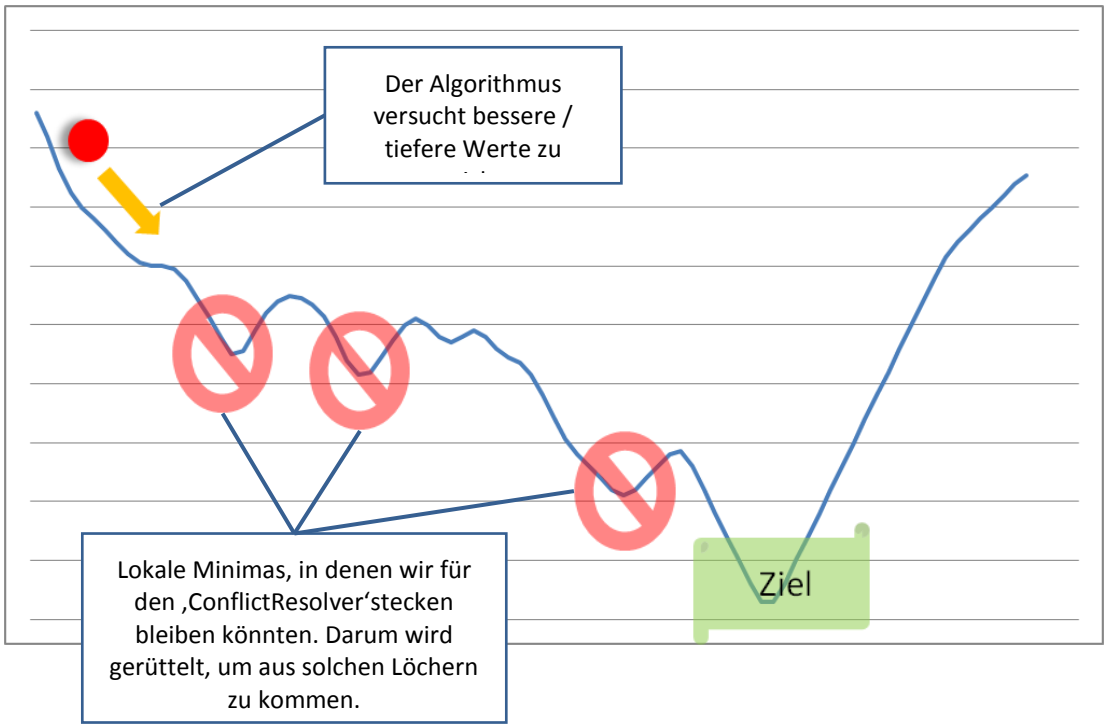
4.9 Settings

Es gibt Settings die jeweils erst zur Laufzeit eingelesen werden. Diese Settings befinden sich im „MySchoolPlanner.exe.config“ File. Dort können folgende Parameter eingestellt werden, die das Verhalten des Algorithmus beeinflussen.

- TreeSpan: Wie viele Lösungen pro Iteration entworfen werden sollen.
- AcceptableRating: Ab welcher Bewertung die Lösung als gut genug erachtet wird.
- MaxHistory: Wie viele Iterationen in einem Historien Cache gehalten werden sollen.

- StagnationDifference: Sobald sich das Rating, der sich in der Historie befindenden Resultate lediglich um diese Differenz unterscheidet wird der Vorgang abgebrochen, da sich der Algorithmus in einem lokalen Minimum verrannt hat.
- MaxRunsAllowed: Wie viele Iterationen maximal durchlaufen werden sollen (danach wird abgebrochen).
- MaxResourceLoad: Hier kann eingestellt werden, ab wie vielen Schulstunden eine Ressource ausgebucht ist. Je geringer dieser Werte, desto geringer die Komplexität, um beispielsweise Konflikte aufzulösen.

4.10 Lokale Minima vermeiden



Die Implementation ist ein nicht-deterministischer, heuristischer Algorithmus. Dieser versucht nach und nach Verbesserungen zu finden, indem er die Lösung mit besserer Bewertung nimmt. Bei solchen Algorithmen kann es sein, dass er in ein lokales Minimum gerät. Dass heisst, alle Änderungen führen zu Verschlechterungen. Dabei könnte eine viel bessere Lösung ausserhalb dieses Minimums erreicht werden.

In der Illustration ist die mögliche Lösung als Funktions-Graph dargestellt. Der Algorithmus ist der Ball, welcher entlang des Lösungs-Graphen Richtung Minimum rollt. Nun hat es lokale Minima, hier mit dem Verbotsschild markiert. Wenn der Algorithmus strikt immer in Richtung des Gefälles arbeiten würde, bleibt er in einem solchen lokalen Minimum stecken.

Eine der einfachsten Lösung bei solchen Problemen ist, dass man ein zufälliges Rauschen beifügt. Dieses Rauschen ‚rüttelt' an der Lösung, so dass der Algorithmus wieder aus den Minimum heraus springt. Nun wird dieses Rauschen langsam vermindert / abgekühlt. Somit springt der Algorithmus gegen Ende weniger aus den Minima hinaus. Am Schluss liegt die gefunden Lösung hoffentlich nahe bei einem der besten Minima.

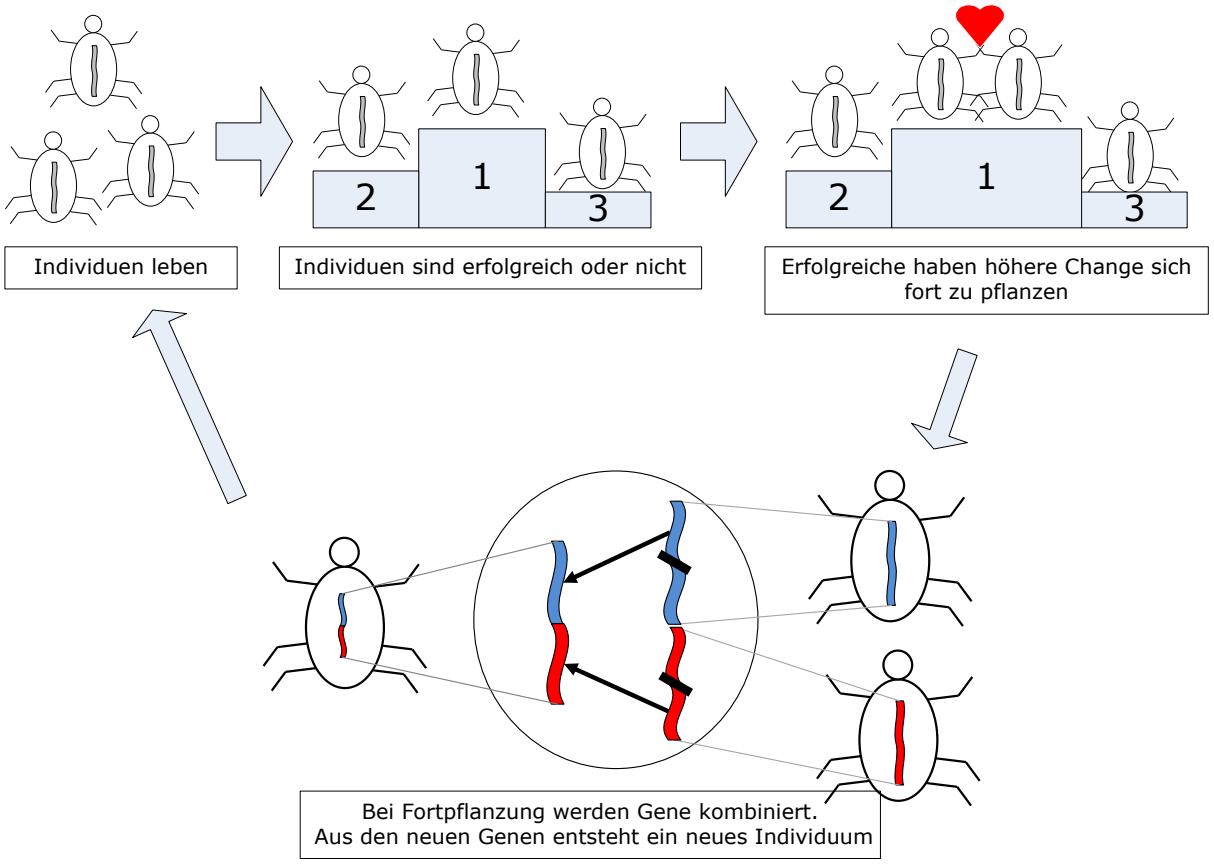
In unserem Algorithmus wird dieses ‚Rauschen' durch zufälliges erneutes verteilen von Lektionen realisiert. Dabei wird die Bewertung benutzt um die ‚Rausch'-Stärke zu regulieren. Bei besseren Wertungen rauschen wir weniger als bei schlechteren.

5 Genetischer Algorithmus

5.1 Grundsätzliche Idee Genetischer Algorithmen

Die Genetischen Algorithmen nehmen die natürliche Evolution als Inspirations-Quelle. Die Evolution hat unendlich viele erfolgreiche Lebewesen-Gattungen hervorgebracht, welche kompliziertesteste Probleme lösen. Wenn die Evolution solch gute Lösungen hervorbringt, warum sollte derselbe Prozess nicht auch andere Probleme erfolgreich lösen können?

In der Natur haben wir viele Individuen, wobei jedes Einzigartig ist. Die Gene eines Individuums repräsentieren dessen Bauplan. Jedes Individuum ist nun mehr oder weniger Erfolgreich im Leben, bedingt durch seine Genetische Veranlagung und durch seine Umwelt. Die erfolgreicheren Individuen haben eine höhere Chance sich fort zu pflanzen, als weniger begünstigte. Somit werden die Gene der Erfolgreicheren öfters weiter gegeben als die Gene weniger erfolgreicher Individuen. Über Generation hinweg werden so die besser angepassten Individuen bevorzugt, wobei sich die Population gesamthaft gesehen der Umwelt anpasst. Zusätzlich gibt es noch Mutationen im Genmaterial durch verschiedene Ursachen.



5.2 Komponenten des Genetischen Algorithmus

5.2.1 Fitness-Funktion

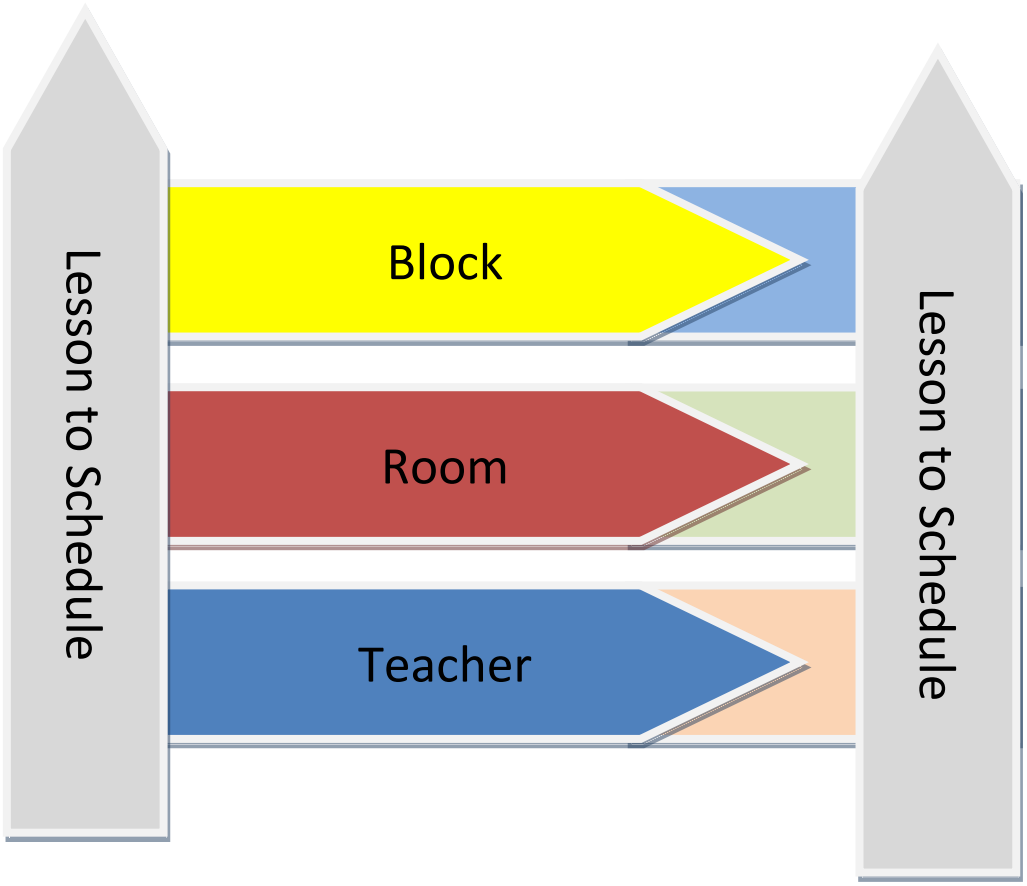
Die Fitness-Funktion repräsentiert die Umwelt in der ein Individuum lebt. Sie bestimmt wie gut ein Individuum abschneidet im Vergleich zu einem anderen Individuum.

Diese Funktion ist die wichtigste Komponente des Genetischen Algorithmus. Hier wird bestimmt wie gut eine Lösung ist. Der Genetische Algorithmus versucht dann die Lösung so zu optimieren, dass sie möglichst gut bei der Fitness-Funktion abschneidet.

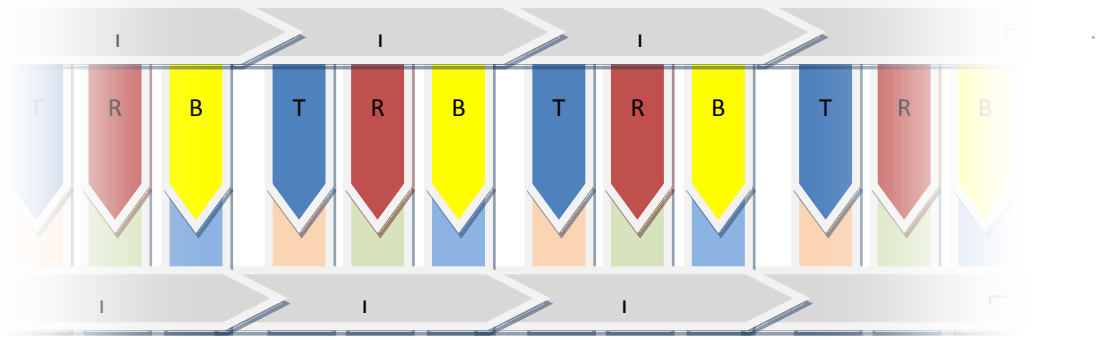
In unserem System wird das Rating-System als Fitness-Funktion benutzt. Denn das Rating-System bewertet ein breites Spektrum an Kriterien die einen guten Stundenplan ergeben. Dadurch sollte ein ständiges optimieren zu einem besseren Rating automatisch auch zu einem besseren Stundenplan führen.

5.2.2 Genom und Gen

Das Genom ist das gesamte Erbgut eines Individuums, während ein Gen nur ein Abschnitt aus dem Genom ist. In unserem Genetischen Algorithmus ist ein Gen folgendermassen aufgebaut. Es wird eine Lektion und die dazugehörige Klasse fix ausgewählt. Die stellt das Grundgerüst eines Gens dar. Anschliessend werden der Raum, der Lehrer sowie der Block (Tag und Zeit) angehängt.

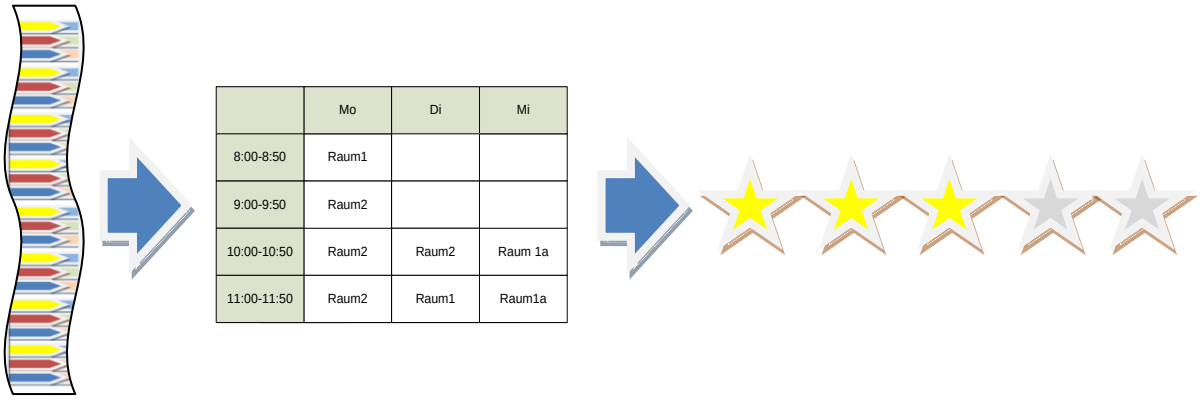


Das Genom besteht nun aus einer Kette solcher Gene. Dabei wird darauf geachtet, dass jede Lektion die geplant werden muss nur einmal vorkommt. Daraus folgt, dass die zu planenden Lektionen ein fixes Gerüst vorgeben. Das Genom ist also eine Kette wobei ein Ketten-Stück eine zu planende Lektion ist. Jedes Ketten-Stück hat einen Block (Tag und Zeit) sowie Raum und Lehrer angehängt.



5.2.3 Individuum

Das Genom repräsentiert den Bauplan eines Individuums. Das Individuum ist die Umsetzung dieses Bauplanes. Das bedeutet, dass anhand des Genoms ein Stundenplan erstellt wird. Anschliessend wird mittels der Fitnessfunktion die Fitness des Individuums gemessen.



	Mo	Di	Mi
8:00-8:50	Raum1		
9:00-9:50	Raum2		
10:00-10:50	Raum2	Raum2	Raum 1a
11:00-11:50	Raum2	Raum1	Raum1a

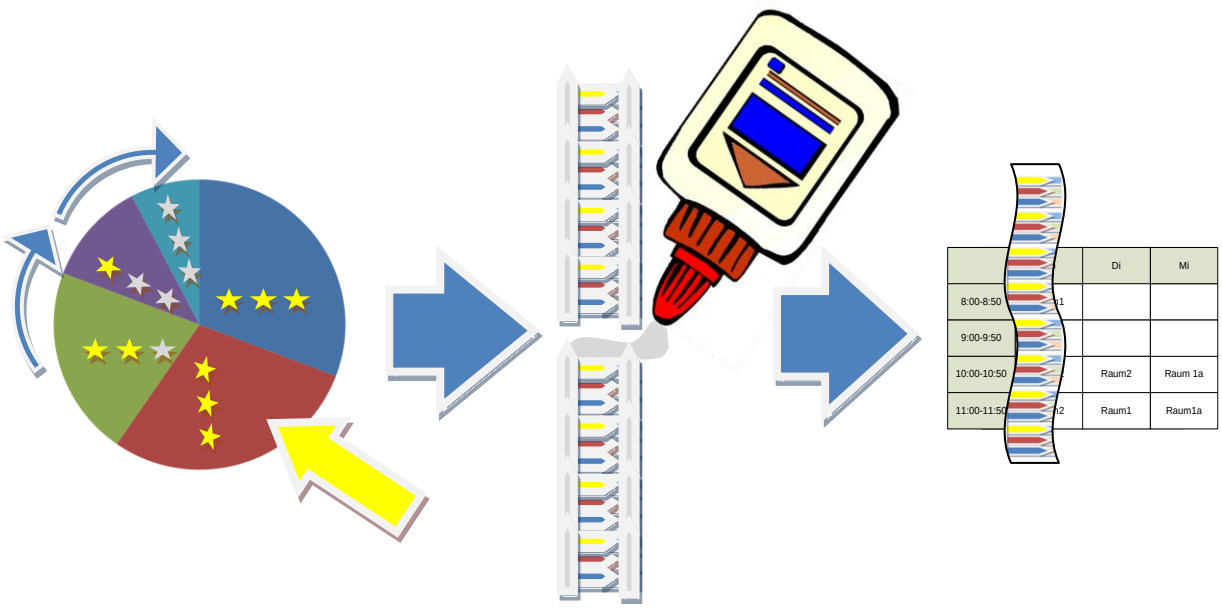
5.2.4 Generation

Ein Individuum gehört zu einer Generation. Jede Generation enthält viele Individuen, welche unterschiedlich ‚fit' sind. Eine Generation geht in eine nächste Generation durch die ‚Fortpflanzung' der einzelnen Individuen über.

5.2.5 Fortpflanzung

Mittels Fortpflanzung entsteht die nächste Generation. Dabei werden immer die Genome zweier Individuen gepaart. Die erfolgreichen Individuen haben eine höhere Chance für den Paarungs-Prozess ausgewählt zu werden. Somit werden die erfolgreichen Gen-Kombinationen mit einer höheren Wahrscheinlichkeit vererbt als weniger erfolgreiche Gen-Kombinationen.

Die Selektion funktioniert wie ein Glücksrad. Jedes Individuum ist ein Streifen auf dem Glücksrad. Erfolgreiche Individuen haben einen breiteren Streifen auf dem Glücksrad und somit eine grössere Chance selektiert zu werden.



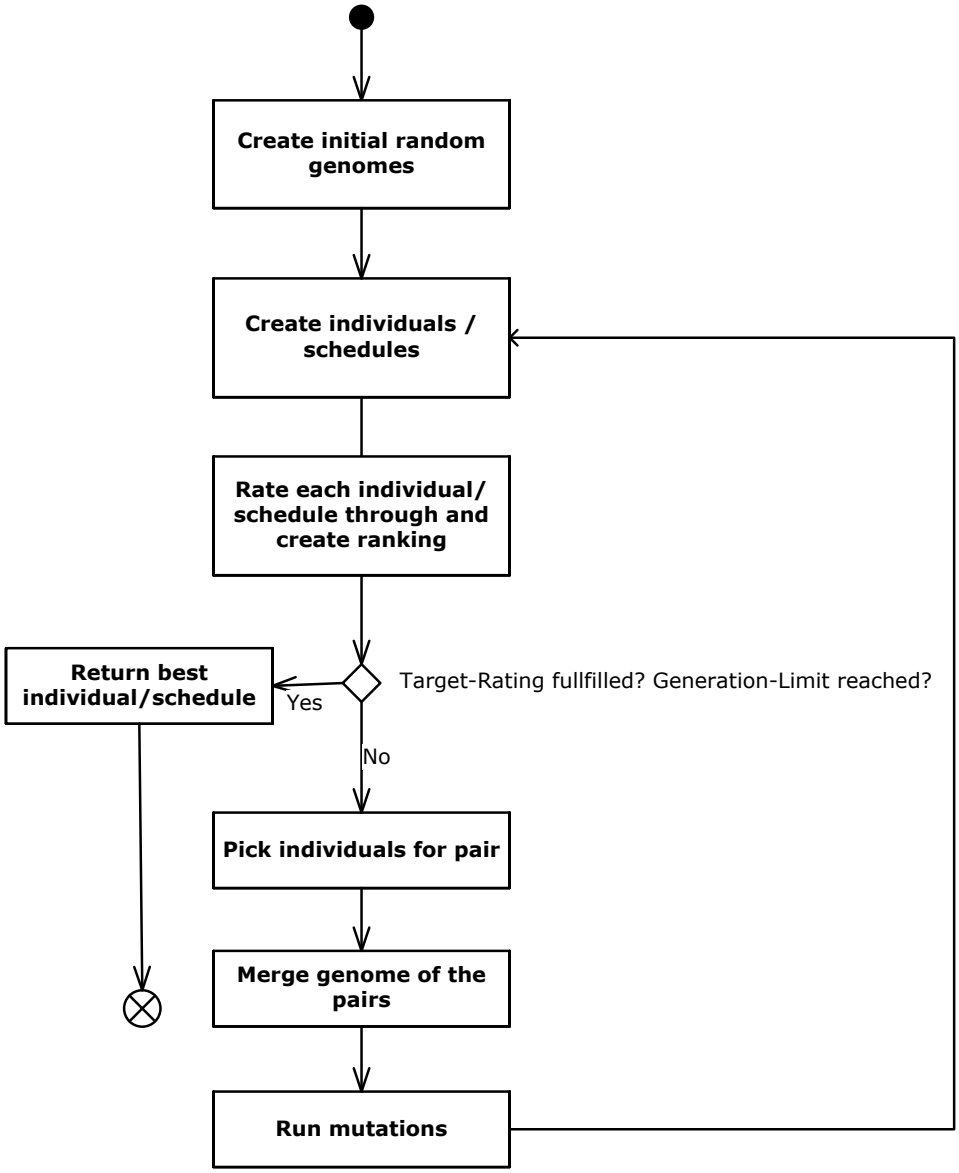
5.2.6 Mutationen

Mit einer gewissen Wahrscheinlichkeit treten Mutationen in den Genomen auf. Das heisst, dass ein Gen oder mehrere Gene zufällig ausgewählt und manipuliert werden. Es wird ein Gen zufällig ausgewählt und dann zufällig ein neuer Lehrer, Raum und Block zugewiesen. Diese Mutationen sollen ‚frisches' Blut in den vorhandenden Genpool einer Generation geben.

5.2.7 Abwechslung der Fitness-Funktion

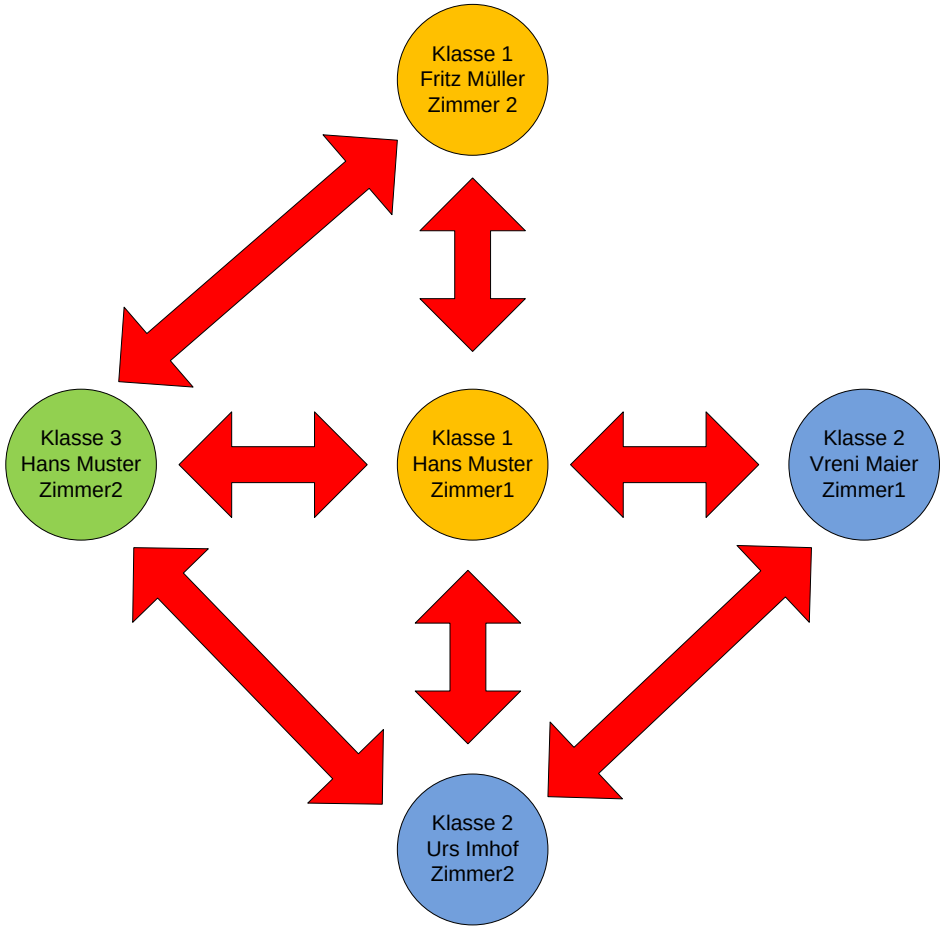
Als Fitness-Funktion wird das Bewertungs-System benutzt. Nun es gibt viele verschiedene Bewertungen die dort einbezogen werden. Jede Bewertung kostet Zeit. Somit ist das Bewertungs-System der Flaschen-Hals im Algorithmus. Jedes Individum einer Generation zu bewerten dauert lange und somit dauert es lange bis eine neue Generation entsteht. Um dieses Problem zu entschärfen wird das Bewertungs-System dynamisch benutzt. Zuerst werden nur einzelne Bewerter als Fitness-Funktion benutzt. Dadurch wird die Bewerting massiv schneller. Es wird immer das schlechteste Bewertungs-Kriterium benutzt. Somit wird für die Kriterien optimiert, welche noch nicht gut abschneiden. Gegen Ende werden dann alle Kriterien benutzt. Somit wird gegend Ende global optimiert. Der Anteile von globaler Bewertung und einzelnen Bewertung kann konfiguriert werden.

5.3 Ablauf Genetischer Algorithmus



6 Graphentheorie basierter Algorithmus

Die Idee bei Graphentheorie basierten Verfahren besteht darin einen Konfliktgraph aufzustellen. In diesem Konfliktgraph stellen die Knoten jeweils eine Schulstunde und ihre benutzten Ressourcen (Lehrer, Raum, Klasse) dar. Mögliche Konflikte werden durch das verbinden von Knoten dargestellt. Das heisst wenn wir nun zwei Stunden haben die gleiche oder teilweise gleiche Ressourcen verwenden, werden die Knoten miteinander verbunden. Dadurch baut man einen Graphen auf, der das gesamte Konfliktpotential zwischen abhängigen Lektionen aufzeigt (siehe Bild).



In einem nächsten Schritt wird der Graph gefärbt. In unserem Fall repräsentiert eine Farbe einen Zeitabschnitt an einem bestimmten Tag im Stundenplan. D.h. dieselbe Farbe ist immer derselbe Zeitabschnitt am selben Tag im Stundenplan. Dabei ist stets das Hauptziel eine minimale Färbung zu erreichen. Dies bedeutet für das Stundenplanproblem, dass möglichst wenige Blöcke verwendet werden um die Lektionen Konfliktfrei zu platzieren.

6.1 Erstellen der Lektionen

Damit der Konfliktgraph aufgebaut werden kann, müssen für alle Stunden die zu verbuchen sind die verwendeten Ressourcen festgelegt werden. Dazu wird der ‚IPlacingContext' des Heuristischen Algorithmus verwendet. Dadurch wird unter Rücksichtnahme auf alle Kriterien eine möglichst faire Ressourcenverteilung erreicht, die zusätzlich noch über Konfigurationsparameter beeinflusst werden kann.

6.2 Konfliktgraph

Der Konfliktgraph besteht aus beliebig vielen Knoten und Verbindungen dieser Knoten untereinander. Diese Verbindungen sind immer Bidirektional. Denn eine Verbindung symbolisiert eine auftretende Konfliktsituation, falls diese Stunden zur gleichen Zeit stattfinden würden. Es wird auch nicht festgehalten auf welchen Ressource(n) der Konflikt zwischen zwei Knoten besteht, da dies für unsere Anwendung nicht von Bedeutung

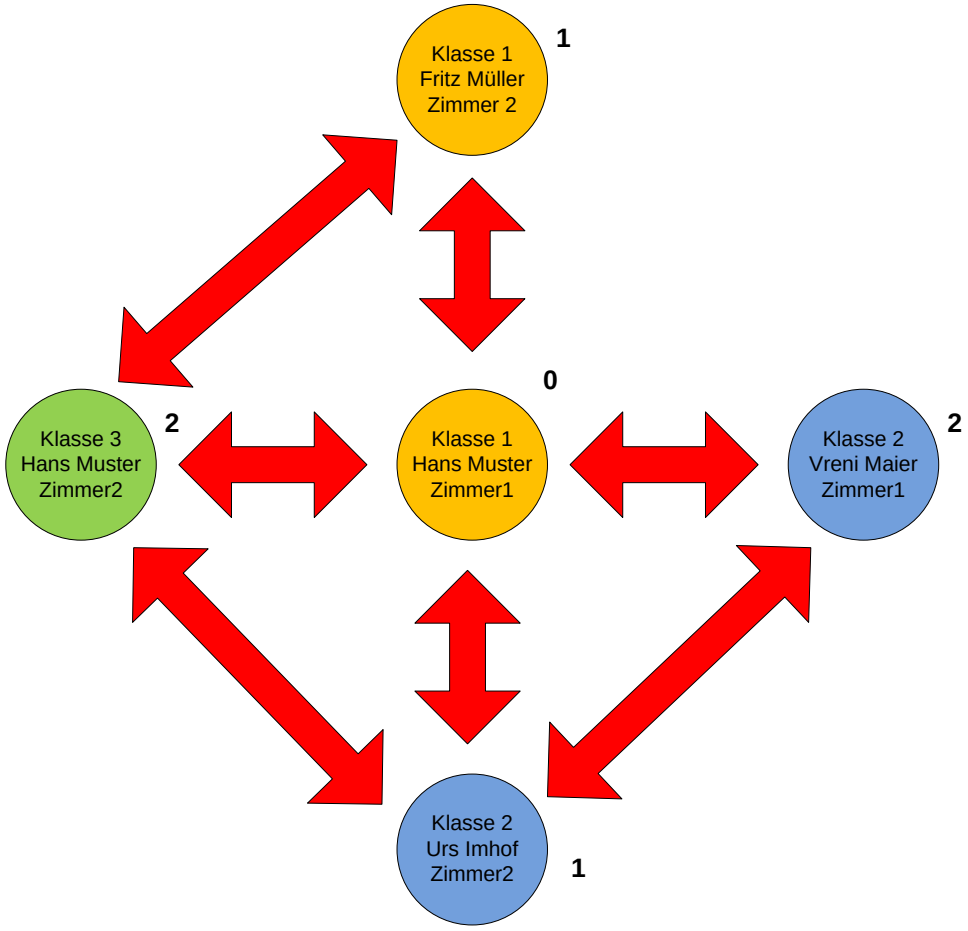
ist. Das Ziel ist einen konfliktfreien Stundenplan zu erhalten und dafür muss lediglich klar sein, welche Stunden nicht parallel stattfinden dürfen.

6.3 Färbung

Als Färbungsalgorithmus wird das ‚ExactMinChoice' Verfahren verwendet. Es ist einfach zu implementieren. Zusätzlich neigt es weniger dazu, sich in einem lokalen Minimum zu verrennen als beispielsweise ein ‚Greedy' Algorithmus.

Im Detail beschrieben ist das Verfahren im Paper ‚Diplomarbeit Tobias Baumann' Abschnitt 2.4 (Paper liegt der Abgabe bei).

Ziel des Verfahrens ist es mit Hilfe eines rekursiven Vorgehens, eine minimale Färbung des Konfliktgraphen zu erreichen. So eine minimale Färbung für unser Beispiel würde ungefähr so aussehen:



Bei der Entscheidung welcher Knoten als nächstes gefärbt werden soll, wird immer nach demselben Prinzip vorgegangen:

- Möglichst viele Einschränkungen durch anliegende Knoten bereits vorhanden (primär)
- Bei der Färbung dieses Knotens werden für möglichst viele andere Knoten neue Einschränkungen erstellt (dieses Kriterium wird nur verwendet, wenn es für das erste Kriterium Knoten gibt, die sich genau gleich gut anbieten)

Ist der Knoten gefunden der gefärbt werden soll, wird die kleinst mögliche Farbe gesucht und mit dieser gefärbt. Anschliessend wird dieser Vorgang rekursive erneut ausgeführt. Das Abbruchkriterium für die Rekursion ist eine komplette Färbung des Graphen. Dann wird die Anzahl Farben zurückgegeben die verwendet wurde um den Graphen einzufärben. Falls diese Anzahl Farben kleiner oder gleich der unteren Barriere für eine akzeptierte Lösung ist, wird diese Lösung zurückgegeben. Falls dies nicht der Fall ist, wird der gefärbte Knoten wieder als ungefärbt markiert und die nächste mögliche Farbe wird verwendet. Das heisst beim ganzen Vorgang werden alle möglichen Kombinationen durchprobiert, bis man eine Lösung erhält, die die untere

Barriere erreicht oder unterschreitet. Daher konvergiert der Algorithmus gerade bei Schulen (z.B. Oberstufe) die ein sehr hohes Konfliktpotential aufweisen sehr schlecht, respektive benötigt sehr viel Zeit eine akzeptierte Lösung zu erreichen.

Man kann die benötigte Zeit jedoch reduzieren in dem man mit einer geringeren maximalen Auslastung der Ressourcen arbeitet (erweitert dadurch den Lösungsraum).

6.3.1 Restriktionen von Bookings

Restriktionen von Bookings werden von diesem Algorithmus ignoriert. Dazu gehören paralleles abhalten von Lektionen, sowie Lektionen die von mehreren Klassen gleichzeitig besucht werden.

6.3.2 Abfolge der Stunden

Es können für Fächer bestimmte Nachfolgestunden erlaubt oder verboten werden. Dieses Kriterium wird von diesem Algorithmus ignoriert und wird daher nie erfüllt.

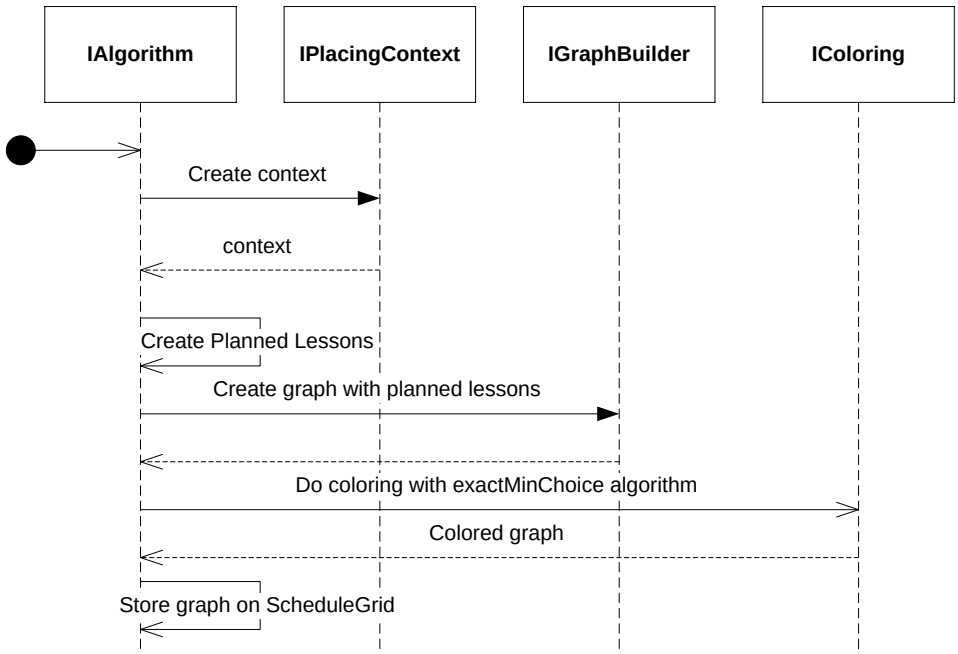
6.3.3 Minimale Stundenausnutzung

Da dieser Algorithmus bestrebt ist immer lokale Minima zu finden, wird die Lastverteilung unter den Tagen sehr schlecht sein. Denn es wird immer Tag für Tag aufgefüllt. Dies bedeutet dass die Stunden sich am Wochenanfang ballen, wohingegen am Ende der Woche keine oder fast keine Stunden mehr vorhanden sein werden.

6.3.4 Verteilung der Stunden

Die Verteilung von gleichen Lektionen auf möglichst viele verschiedene Tage wird von diesem Algorithmus nicht garantiert. Ganz im Gegenteil wird dieser Aspekt komplett ignoriert und es muss damit gerechnet werden, dass alle Stunden eines Faches direkt nacheinander stattfinden.

6.4 Implementation



6.4.1 IGraphBuilder

Der ‚IGraphBuilder' ist dafür verantwortlich aus den einzelnen ‚PlannedLesson' Knoten zu machen und diese untereinander zu einem Graphen zu verbinden. Wichtig ist dabei zu wissen, dass es möglich ist, dass dabei mehrere Graphen entstehen können. Dies ist nämlich dann der Fall, falls es zwischen einzelnen Teilgraphen keine Verbindungen gibt. Nachher kann jeder Graph für sich weiter prozessiert und gefärbt werden. Das Färben

eines einzelnen Graphen findet in einem eigenen Thread statt, das heisst mehrere Graphen zu bekommen wäre der Idealfall, da dann ein paralleles Färben der Graphen stattfinden kann.

Eine Verbindung zwischen zwei Knoten ist immer Bidirektional, da diese Verbindung ja lediglich einen Konflikt darstellt zwischen zwei Knoten egal in welche Richtung.

6.4.2 IColoring

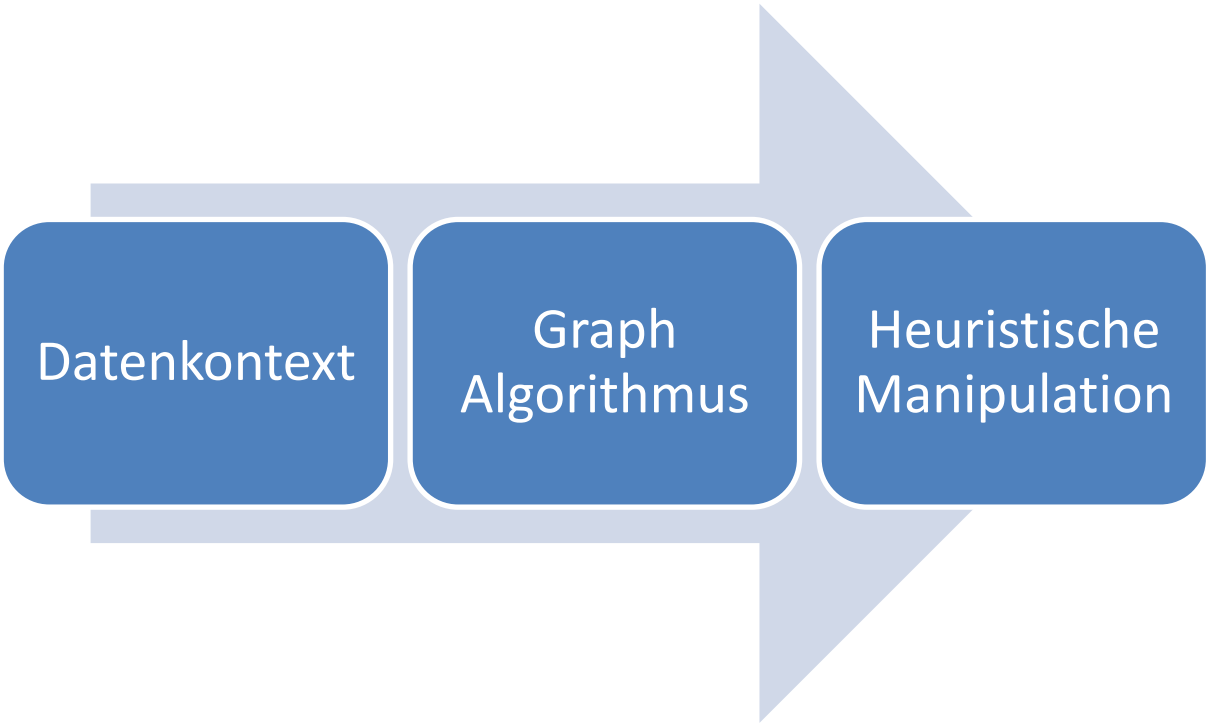
Hier werden die Graphen gefärbt. Für das Färben wird das ‚exactMinChoice' Verfahren gewählt, welches bereits im Kapitel Färbung genauer erklärt wurde.

7 Graphentheorie basierter Algorithmus kombiniert mit heuristischem Algorithmus

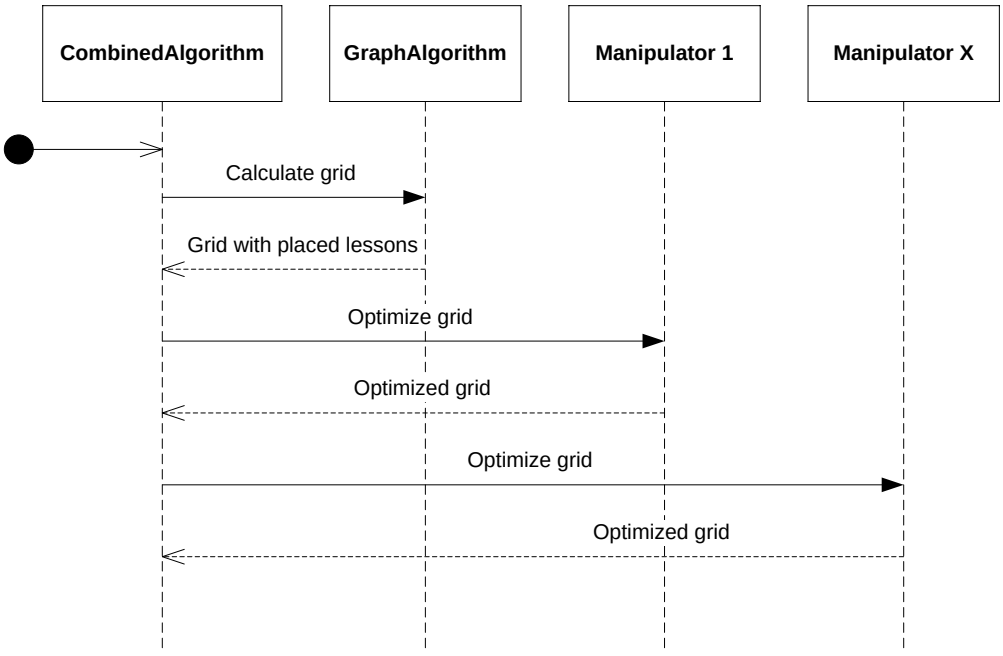
7.1 Idee

Der Algorithmus basierend auf der Graphentheorie ist vor allem darin stark die Stunden konfliktfrei zu platzieren. Er vernachlässigt aber viele andere Aspekte, die einen guten, brauchbaren Stundenplan ausmachen.

Mit diesem kombinierten Algorithmus wird versucht genau diese Mängel etwas auszumerzen. Dazu wird zuerst eine Initiale Platzierung durch den ursprünglichen graphentheorie basierten Algorithmus vorgenommen. Anschliessend wird diese Lösung mit Hilfe der Manipulationsstrategien des heuristischen Algorithmus verbessert (siehe Kapitel 4.8).



7.2 Implementation



Es wird eigentlich der erste Teil des heuristischen Algorithmus (der Teil der das platzieren der Stunden vornimmt) ersetzt durch den graphentheorie basierten Algorithmus. Zusätzlich wird auch auf ein iteratives vorgehen verzichtet, da der graphentheorie Algorithmus immer auf dieselbe Lösung kommen wird. Um ein wirklich brauchbares Iteratives Verfahren zu erhalten, müsste die Platzierungsstrategien des Heuristischen Algorithmus zurückgegriffen werden. Jedoch gäbe es dann ab der zweiten Iteration Lösungen die komplett vom Heuristischen Algorithmus generiert worden sind. Dies ist jedoch nicht das Ziel einer Kombination von zwei Algorithmen, deswegen läuft dieser Algorithmus nicht iterativ ab.

8 Auswertung der Algorithmen

8.1 Testfälle

Damit die Algorithmen auf verschiedene Fälle getestet werden können, wurden vier Testdatenbanken erstellt. Für die Datenbank ‚Unterstufe' werden die Stundenpläne der Primarschule Chur Rheinau verwendet. Die Datenbank ‚Oberstufe' basiert auf den Oberstufen Stundenplänen der Oberstufe Thusis. Der Testfall ‚Unterstufe mit Ortsverteilung' ist frei erfunden. Die Ressourcen-Knappheit Datenbank basiert auf der Testdatenbank ‚Unterstufe' wobei weniger Räume für den Unterricht zur Verfügung stehen.

8.1.1 Unterstufe

In dieser Datenbank gibt es insgesamt 12 Primarklassen. Dies heisst für jede Stufe gibt es zwei Klassen. Die meisten Stunden ausser gewissen Spezialstunden wie beispielsweise Turnen werden vom Klassenlehrer gehalten. Die Komplexität dieser Testdatenbank ist sehr gering, da es keine grossen Konfliktpotentiale zwischen den Klassen gibt. Denn 80% – 90% des Unterrichts finden beim gleichen Lehrer im gleichen Zimmer statt.

Die Algorithmen sollten bei dieser geringen Komplexität keine grösseren Probleme haben.

8.1.2 Unterstufe mit Ortsverteilung

Diese Datenbank besteht im wesentlichen aus der Datenbank ‚Unterstufe' wobei zusätzlich sechs weitere Primarklassen hinzugefügt wurden. Es gibt nun also für jede Primarstufe drei Klassen. Um die Komplexität etwas zu erhöhen, gibt es drei verschiedene Standorte, die jeweils eine Primarklasse von jeder Stufe beherbergen. Damit nun auch Lokalitätswechsel stattfinden hat ein Standort keine Turnhalle und ein Standort keine Werkräume / Handarbeitszimmer. Es kann hier also erwartet werden, dass die Bewertung für Lokalitätswechsel niemals 100% erreichen wird, da Lokalitätswechsel gezwungenermassen stattfinden müssen.

Trotz dieser zusätzlichen Schwierigkeit kann auch diese Datenbank als nur leicht komplexer als die Datenbank ‚Unterstufe' angesehen werden. Denn die Klassen sind nur in einem sehr geringen Ausmass einem Konkurrenzkampf um Ressourcen ausgesetzt.

8.1.3 Oberstufe

Es gibt in dieser Datenbank lediglich sechs Klassen. Es hat drei Real Klassen und drei Sekundar Klassen. Brauchbare Stundenpläne für diesen Fall zu erstellen ist als deutlich komplexer als auf der Primarstufe. Denn die Klassen beeinflussen sich gegenseitig und es herrscht ein Kampf um Ressourcen.

Jeder Lehrer hat sein eigenes Zimmer und gibt jeder Klasse einzelne Fächer. Die Klassen konkurrenzieren sich also gegenseitig bei der Lehrer und Raum Belegung. Damit ist die Schwierigkeit einen konfliktfreien, ausgeglichenen Stundenplan zu erhalten sehr hoch. Die Komplexität steigt zusätzlich noch durch Restriktionen. So müssen Werken und Handarbeit Stunden parallel stattfinden. Auf weitere Restriktionen wurden verzichtet, obwohl diese in einem realen Umfeld vorhanden wären (Religion, Turnen).

Bei dieser Datenbank werden die schlechtesten Ergebnisse erwartet, da die beiden anderen nicht annähernd an eine solche Komplexität erreichen.

8.1.4 Spezial Szenario, Ressourcen-Knappheit

Dieses Szenario soll aufzeigen, wie gut der Algorithmus zurechtkommt, wenn eine Ressourcenknappheit herrscht. Das Szenario basiert auf dem Unterstufen Testfall, wobei die Raum-Ressourcen auf ein Minimum Reduziert sind. Z.B. muss die Turnhalle bis auf zwei Lektionen voll besetzt werden, damit überhaupt alle Klassen Turnen können. Desweiteren gibt es weniger Klassen-Zimmer, was bedeutet, dass die Zimmer besser ausgelastet werden müssen.

8.2 Erwartungen an Algorithmen

8.2.1 Heuristischer Algorithmus

Für unsere Testfälle erwarten wir das der heuristische Algorithmus am besten Abschneidet. Erstens hat der Heuristische Ansatz viel spezifisches Wissen über genau diese Problemstellungen eingebaut. Das heisst, es hat viel spezielle Logik darin, der für ganz spezifische Szenarien zuständig ist. Dadurch können diese Spezialfälle gut abgedeckt werden.

Desweiteren ist in diesen Algorithmus am meisten Optimierungs- und Tuning aufwand investiert worden. Dadurch ist er bereits sehr gut justiert auf diese Aufgaben.

Trotzdem hat dieser Algorithmus einen gewichtigen Nachteil. Weil er auf puren, heuristischen Ansätzen basiert, ist er nicht allgemein verwendbar. Bei komplett neuen Test-Szenarien wird er vermutlich versagen.

8.2.2 Graph-Algorithmus

Der Graphen-Algorithmus kann eigentlich nur eine Aufgabe perfekt. Er kann alle Stunden 100% konfliktfrei platzieren. Er wird niemals einen Ressourcenkonflikt erzeugen.

Auf der anderen Seite kennt er keine weiteren Kriterien. Er wird alle Stunden extrem kompakt und Konfliktfrei platzieren. Ein solch kompakter Stundenplan ist jedoch nicht Praxis tauglich.

8.2.3 Kombinierter Algorithmus

Dieser Algorithmus ist eine Kombination des Graphen Algorithmus mit Heuristischen Ansätzen. Mit den Heuristiken werden die grössten Schwächen des Graphen-Algorithmus behoben. Daher erwarten wir bessere Ergebnisse als mit dem Graphen-Algorithmus.

8.2.4 Genetischer Algorithmus

Der Genetische Algorithmus besitzt keine richtige ‚Koordination'. Ein brauchbares Ergebnis ist abhängig von guten Bewertungen und vielen anderen Faktoren. Im innersten Kern basiert der Algorithmus auf ansammeln glücklicher Zufälle. Es wird darauf gehofft, dass man durch Akkumulierung guter Zufälle auf ein gutes Gesamtresultat kommt.

Wir erwarten kein gutes Ergebnis von diesem Ansatz. Der Lösungsraum ist extrem gross, die Fitness-Funktion ist nicht perfekt und man hat nicht unendlich Zeit. Wir vermuten daher, dass man keine brauchbare Lösung im Lösungsraum innerhalb nützlicher Zeit finden wird.

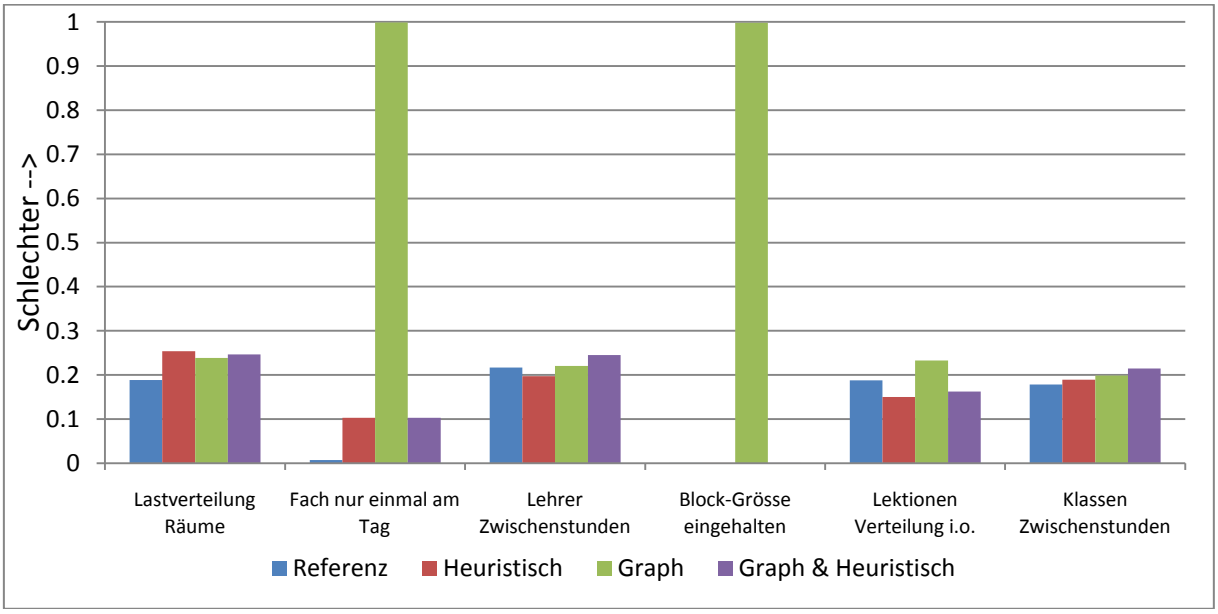
Natürlich ist dieser Algorithmus sehr stark von den Parametern abhängig. Man kann das Ergebnis stark verbessern durch weiteres Optimieren der Parameter. Allerdings ist dies nicht im Umfang dieser Arbeit enthalten und wurde nur zu einem gewissen Grad gemacht.

8.3 Vergleich der Algorithmen

Bei diesem Vergleich werden nur die Bewertungen gezeigt, welche Unterschiede Zwischen den Algorithmen aufzeigen. Die Diagramme zeigen die Problemstellen. Ein hoher Wert ist also eine Indikation für eine Disziplin welche der Algorithmus nicht beherrscht.

Ein Wert von nahe bei eins bedeutet, dass der Algorithmus diese Disziplin komplett ignoriert und nicht beherrscht. Ein Wert von null bedeutet, dass dieses Kriterium perfekt erfüllt ist. Ein Wert zwischen 0 und 0.2 ist ein guter Wert.

8.3.1 Vergleich Unterstufe



Heuristischer Algorithmus

Wie man in der Grafik sieht, schneidet er Heuristische Algorithmus extrem gut ab. Der Algorithmus hat keine Bewertungsdisziplin in der er komplett versagt. In den meisten Bewertungen ist er geringfügig schlechter als die Referenz.

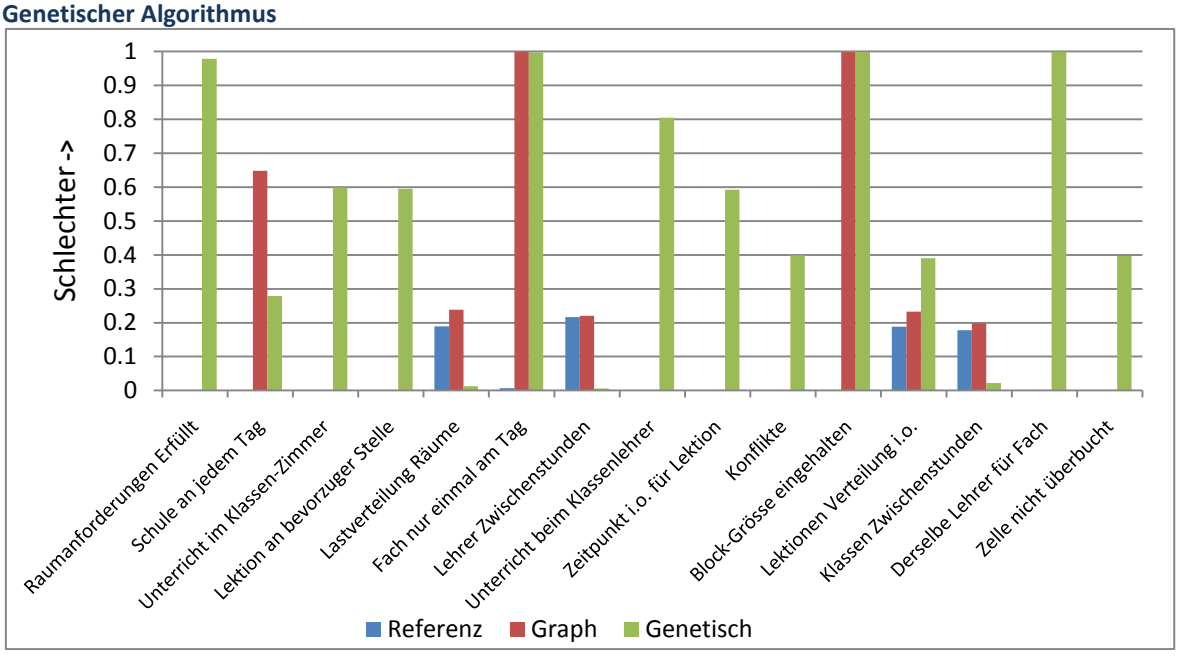
Graph-Algorithmus

Es ist klar ersichtlich, dass der Graph-Algorithmus zwei grosse Schwächen hat. Er schneidet extrem schlecht ab bei der ‚Fach nur einmal am Tag' sowie bei ‚Block-Grösse eingehalten'. Der Grund ist, dass der Graph-Algorithmus kein Konzept zur Verteilung der Stunden hat. Er platziert alle Stunden Konfliktfrei, ignoriert dabei aber eine regelmässige Verteilung der Stunden über die gesamte Woche.

Graph und Heuristisch Kombiniert

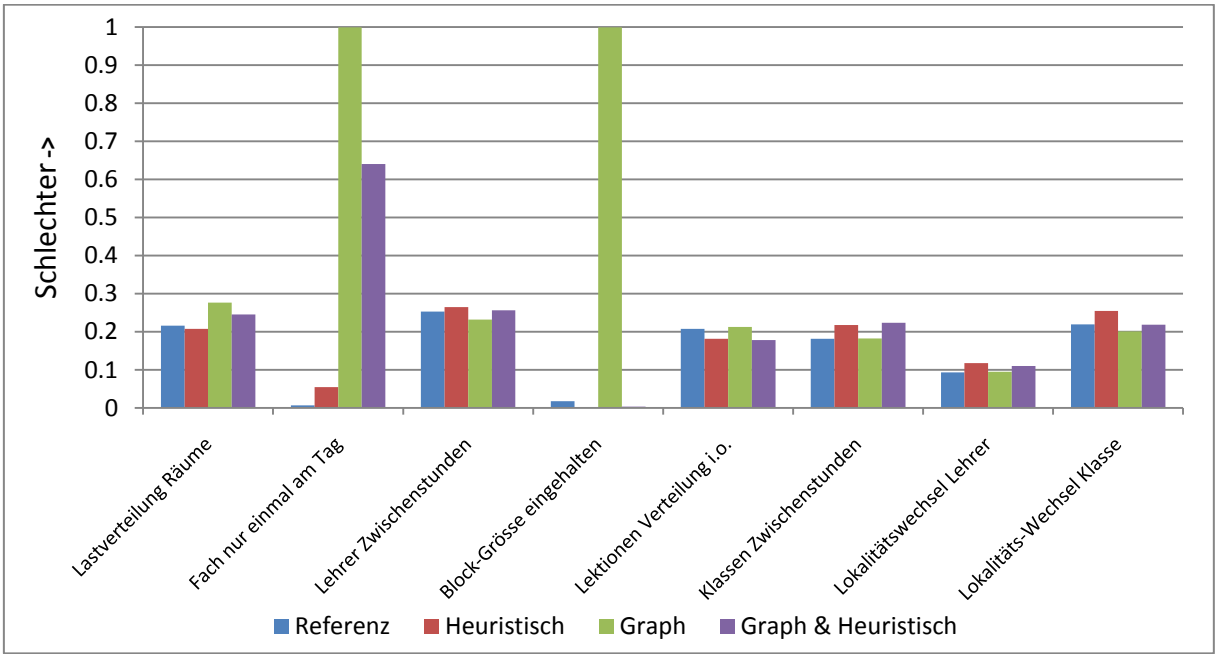
Man sieht gut, dass sobald man heuristische Verfahren zum Graph-Algorithmus hinzufügt, der grösste Teil seiner schwächen Behoben wird. Überraschend ist, wie gut diese Kombination funktioniert. Die Kombination ist etwa gleicht gut wie reine heuristische Ansätze.

Genetischer Algorithmus



Unser Ansatz eines genetischen Algorithmus funktioniert überhaupt nicht. Der Vergleich mit der Referenz und dem Graphen-Algorithmus zeigt, dass der Genetische Algorithmus die meisten Disziplinen nicht handhaben kann. Es gibt ein paar wenige Stellen, in der er besser als die Referenz ist, z.B. die Lastverteilung und die Zwischen-Stunden. Aber das muss man relativieren, da viele Grundsätzliche Kriterien wie Konflikt-Freiheit, Raumanforderungen etc. überhaupt nicht erfüllt sind.

8.3.2 Vergleich Unterstufe mit Ortsverteilung



Heuristischer Algorithmus

Der Heuristische Algorithmus schneidet wieder gut ab. Die zusätzlichen Lokalitätswechsel stellen kein Problem dar. Gut zu sehen ist, dass durch die zusätzliche Komplexität einige Bewertungen nicht mehr so perfekt sind, wie beim vorhergehenden Vergleich.

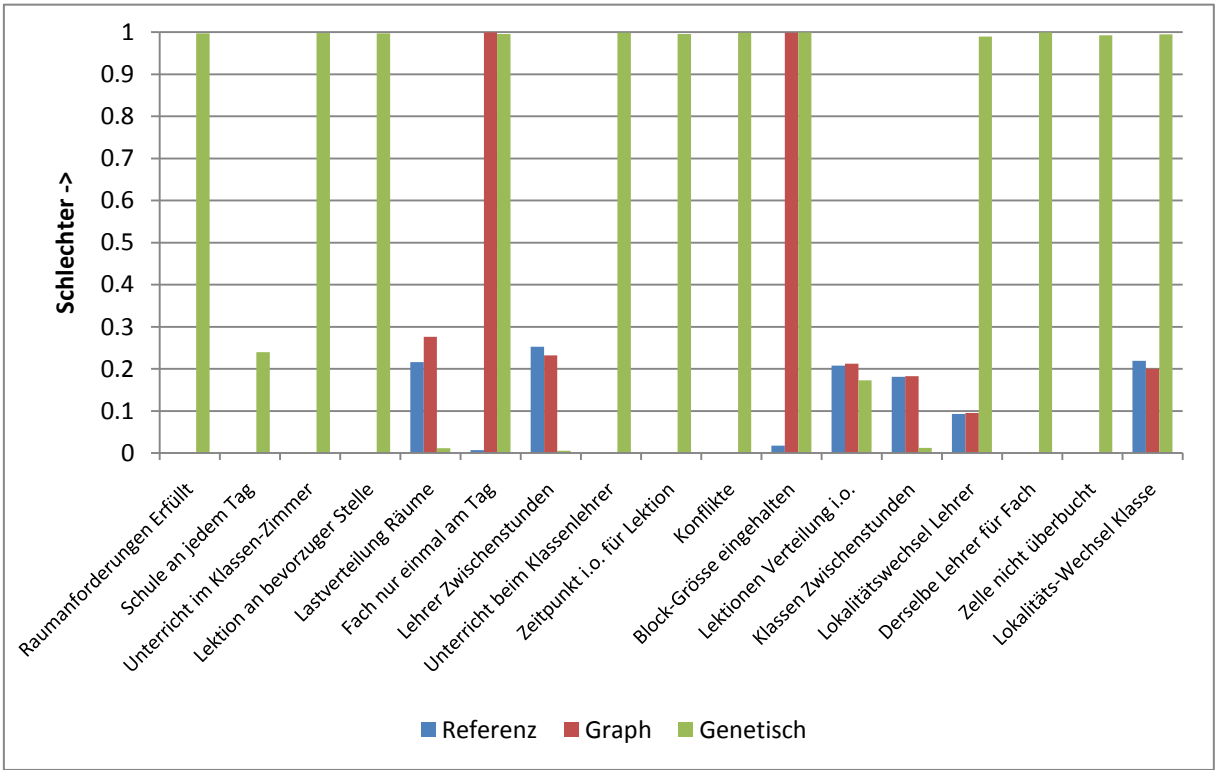
Graph-Algorithmus

Wieder sieht man, dass der Graphen-Algorithmus die Lektionen nicht verteilen kann. Ansonsten schneidet er ebenfalls gut ab. Die verschiedenen Lokalitäten machen keine Probleme. Der Grund dafür liegt darin, dass diese Komplexität bereits beim erstellen des Datenkontextes berücksichtigt wird.

Graph und Heuristisch Kombiniert

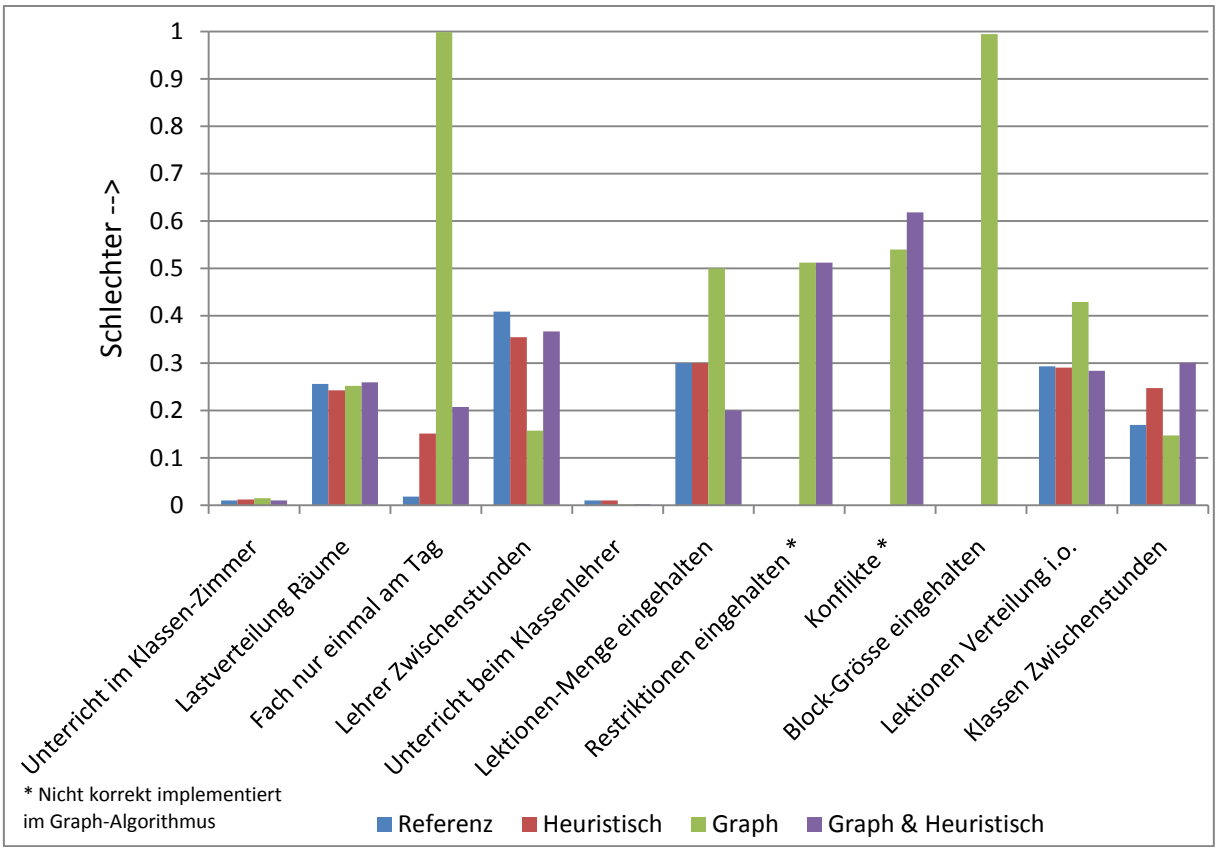
Auch hier können die zusätzlichen Heuristiken das Ergebnis stark verbessern. Allerdings ist die Verbesserung bei der Bewertung ‚Fach nur einmal am Tag' nur sehr gering. Es zeigt sich, dass die nachträglichen Heuristiken die ungünstige Vorlage des Graphen-Algorithmus nicht immer beheben können.

Genetischer Algorithmus



Der Genetische Algorithmus schneidet extrem schlecht ab. Fast alle Bewertungs-Kriterien werden nicht annähernd erfüllt.

8.3.3 Vergleich Oberstufe



Heuristischer Algorithmus

Auch beim komplexeren Oberstufen-Szenario kann der Heuristische Algorithmus durchaus mit dem Referenz-Stundenplan mithalten und ist nur geringfügig schlechter.

Graph-Algorithmus

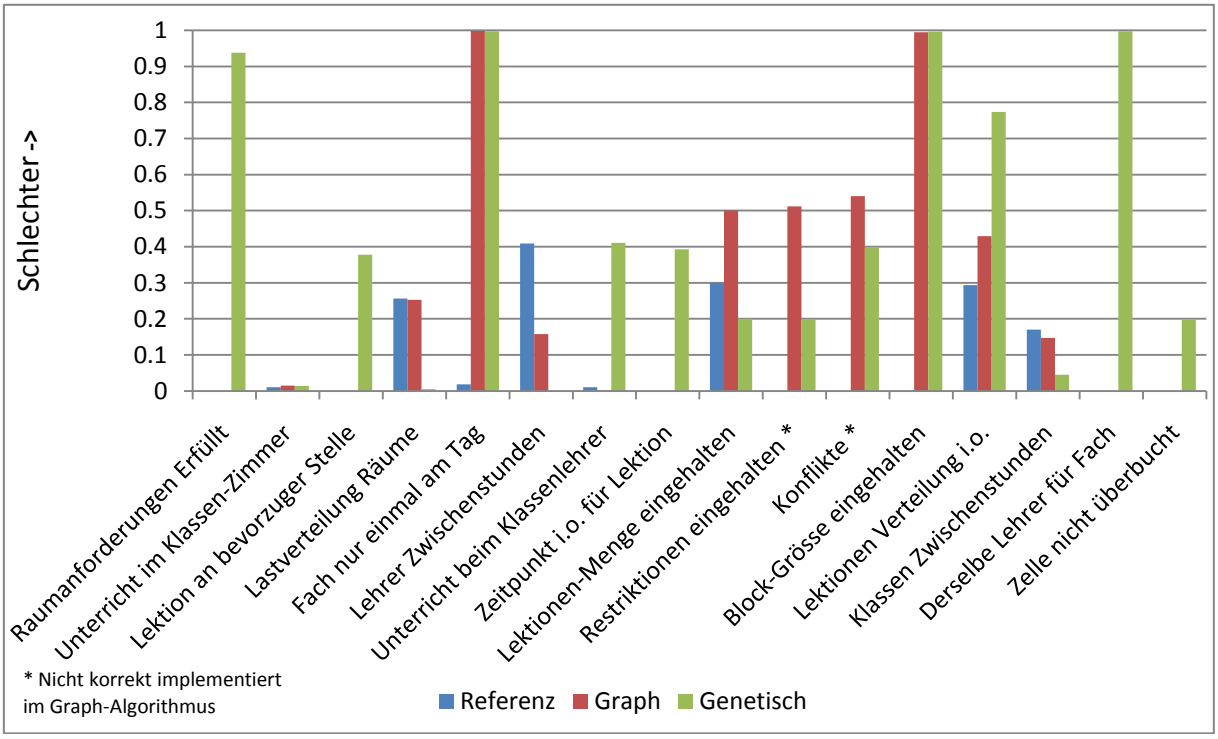
Wir sehen wieder die grosse Schwäche des Graphen-Algorithmus bei ‚Fach nur einmal am Tag', ‚Lektionen-Menge eingehalten' und ‚Block-Grösse eingehalten'. Der Algorithmus verteilt die Lektionen nicht gleichmässig. Dafür macht er weniger Zwischen-Stunden, sowohl für die ‚Lehrer Zwischenstunden' als auch die ‚Klassen Zwischenstunden'.

Ebenfalls gut ersichtlich ist, dass der Graph-Algorithmus Konflikte erstellt für dieses Szenario. Das ist allerdings eine Implementationsschwäche, weil unsere Implementation Restriktionen nicht korrekt behandelt. Wenn man dies korrekt implementiert, dann wird der Graph-Algorithmus auch in dieser Bewertung perfekt abschneiden.

Graph und Heuristisch Kombiniert

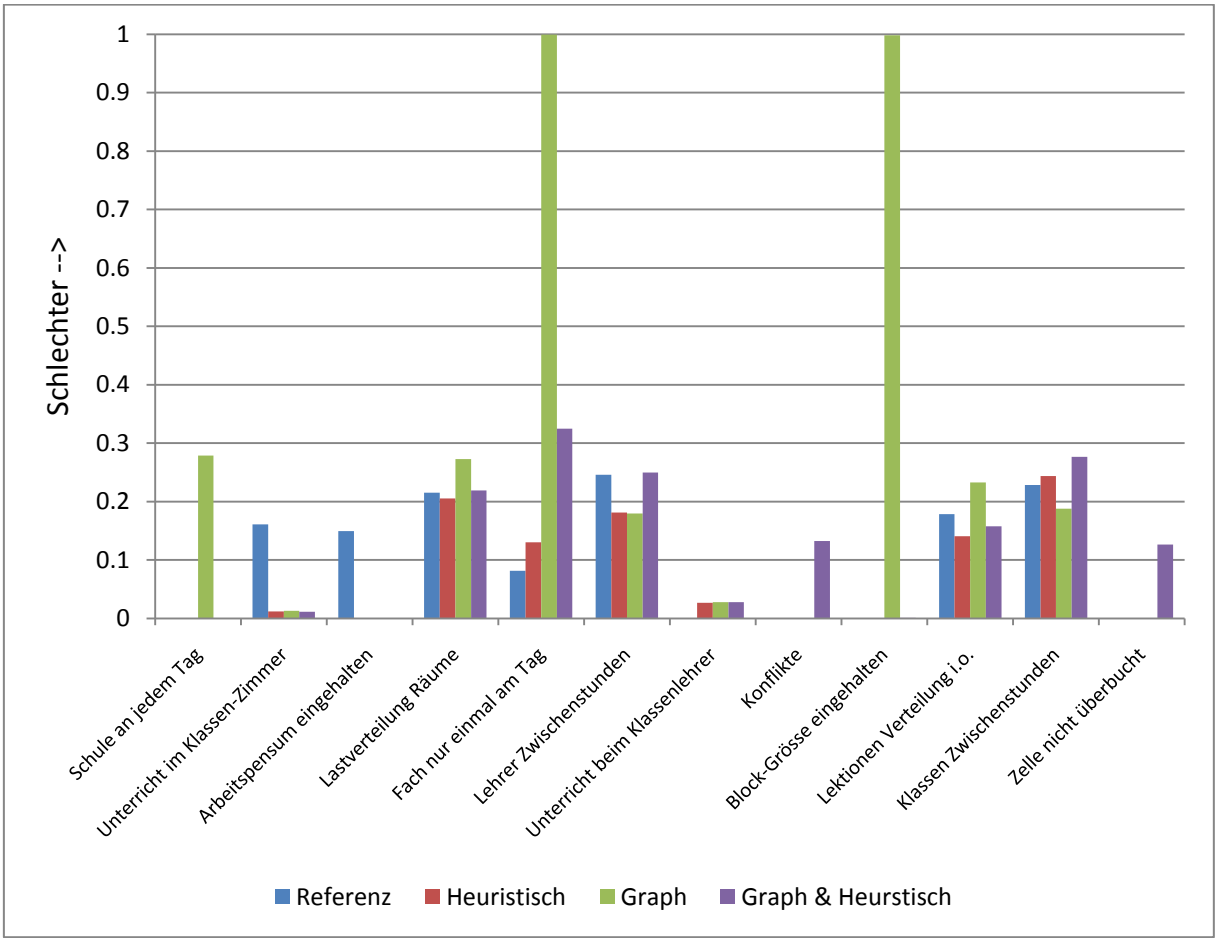
Wieder sieht man, wie die zusätzlichen Heuristischen Verfahren die Schwächen des Graph-Algorithmus beheben.

Genetischer Algorithmus



Auch hier schneidet der Genetische Algorithmus schlecht ab. Allerdings nicht so schlecht wie in den anderen Szenarien.

8.3.4 Vergleich Spezial Szenario, Ressourcen-Knappheit



Referenz

Die Referenz ist in den Bewertungen ‚Unterricht im Klassen Zimmer' und ‚Arbeitspensum eingehalten' deutlich schlechter als die Algorithmen. Das heisst die manuelle Erstellung nicht genügend Sorgfältig war.

Heuristisch

Der Heuristische Algorithmus erstellt akzeptable Stundenpläne auch unter Ressourcenknappheit.

Graph

Der Graph-Algorithmus hat die bekannte Schwäche bei dem Verteilen der Lektionen. In diesem Szenario lässt er sogar Tage für manche Klassen komplett schulfrei.

Graph und Heuristisch Kombiniert

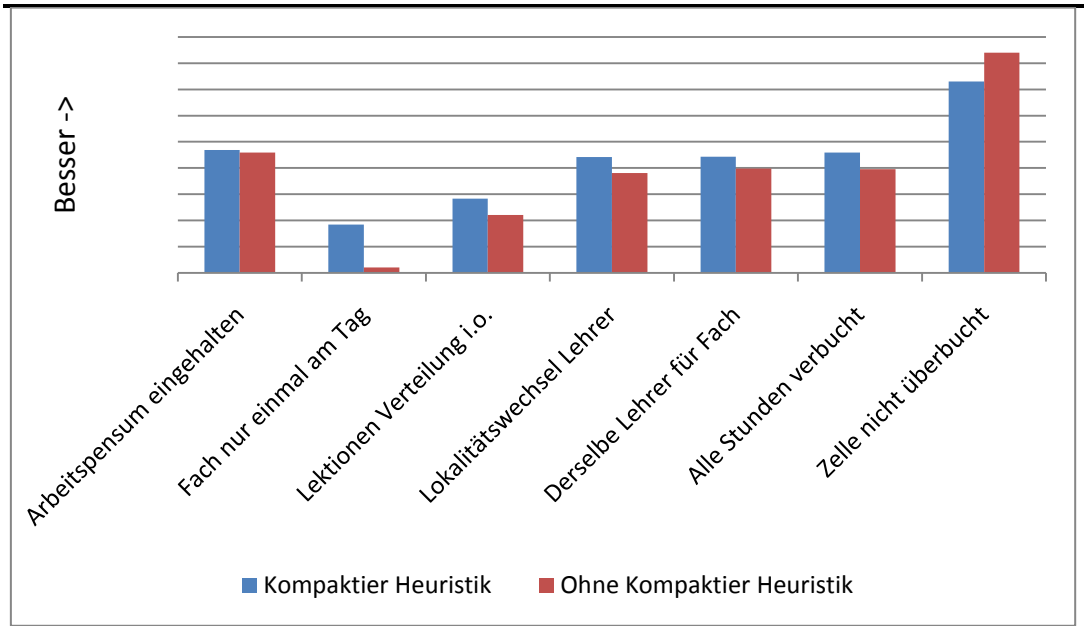
Wieder verbessert die zusätzlichen Heuristiken das Resultat des Graphen Algorithmus. Allerding auf Kosten von neuen Konflikten.

8.4 Details zu einzelnen Algorithmen

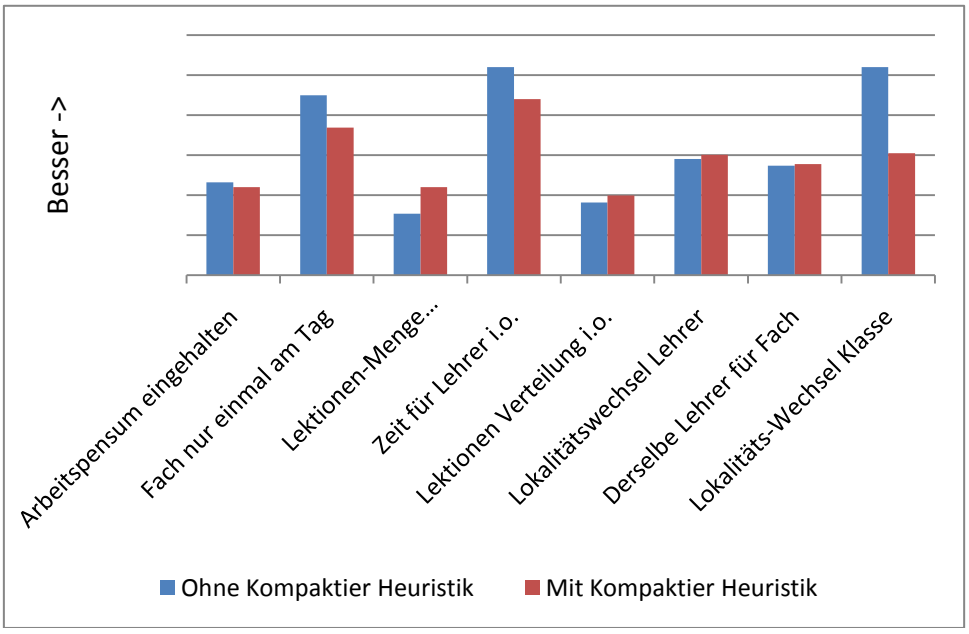
8.4.1 Heuristischer Algorithmus

Empfindlichkeit von Heuristiken

Die Heuristiken funktionieren sehr gut, solange diese gut auf das Szenario angepasst sind. Es ist jedoch schnell der Fall, dass eine Heuristik für gewisse Szenarien Verbesserung bringt und für andere eine Verschlechterung. Als Beispiel nehmen wir unsere Kompaktier-Heuristik (‚DayCompacter'), welche versucht Stunden näher zusammen zu rücken. Zuerst ein Vergleich bei dem Unterstufen mit Ortsverteilung Szenario:



Wir sehen, dass für die meisten Punkte die zusätzliche Heuristik eine kleine Verbesserung der Bewertung bringt. Somit ist diese Heuristik sinnvoll für dieses Szenario. Nun das gleiche beim Oberstufen-Szenario:

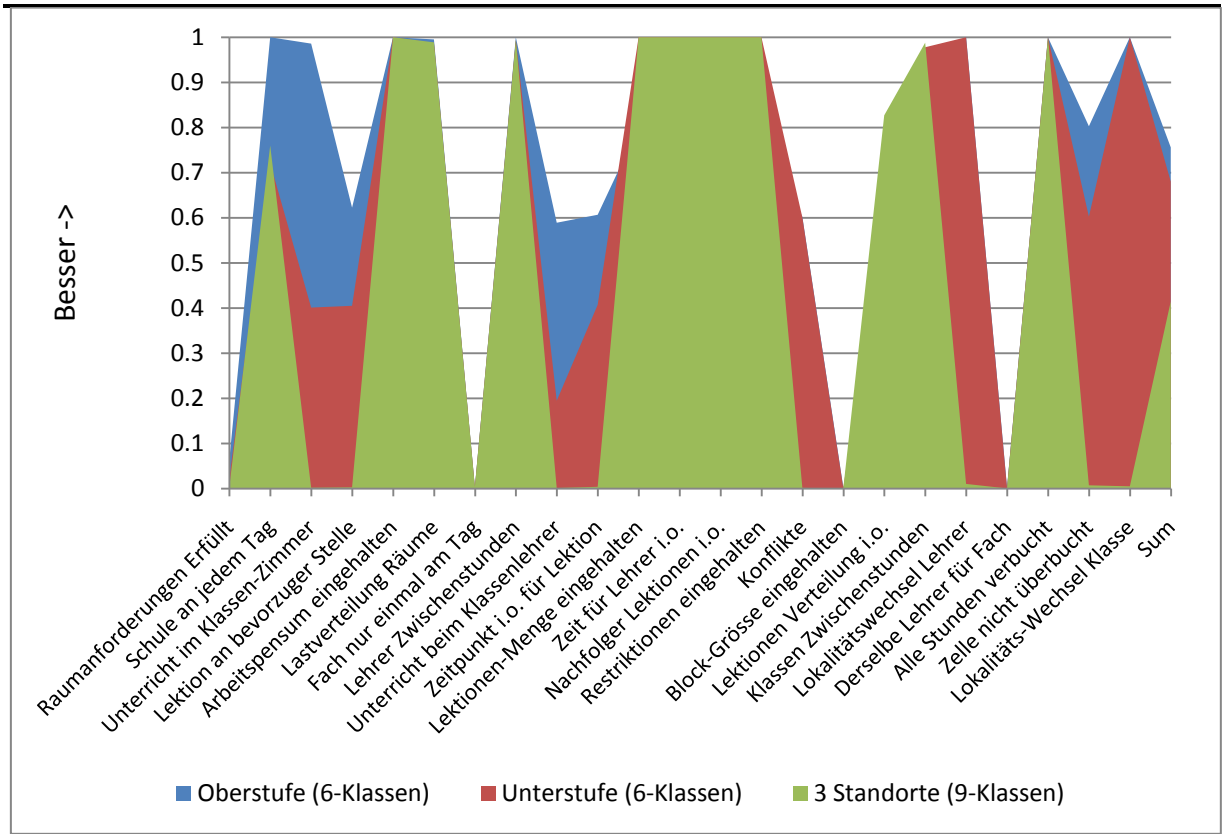


Hier sieht man das Gegenteil. Die Kompaktier-Heuristik verschlechtert das Ergebnis. An diesem Beispiel erkennt man gut die Schwäche von Heuristiken. Eine Heuristik ist meistens für ein spezifisches Szenario ausgelegt. Sie ist daher oft nicht geeignet für eine breite Palette an Szenarien.

8.4.2 Genetischer Algorithmus

Selbstvergleich

Wir haben gesehen dass der Genetische Algorithmus generell sehr schlecht abschneidet. Dennoch ist es interessant diesen Algorithmus weiter zu analysieren. Z.B. wenn man die Resultate der drei verschiedenen Szenarien mit einander vergleicht.



In diesen Graphen sind höhere Werte ein besseres Resultat. Man sieht, dass der Algorithmus für das Szenario ‚Oberstufe' am besten abschneidet. Der Algorithmus schneidet also bei einem komplexeren Szenario besser ab als beispielsweise beim einfacheren Unterstufen- und Standorte-Szenario. Wie kann das sein?

Wenn man es sich das Resultat genau ansieht, sieht man das der Algorithmus für das Oberstufen Szenario am besten, für die Unterstufe am zweitbesten abschliesst und bei den verschiedenen Standorten am schlechtesten. Wenn man sich die Ausgangs-Lage und die Resultate genau ansieht, kann man folgende Zusammen-Hänge erkennen:

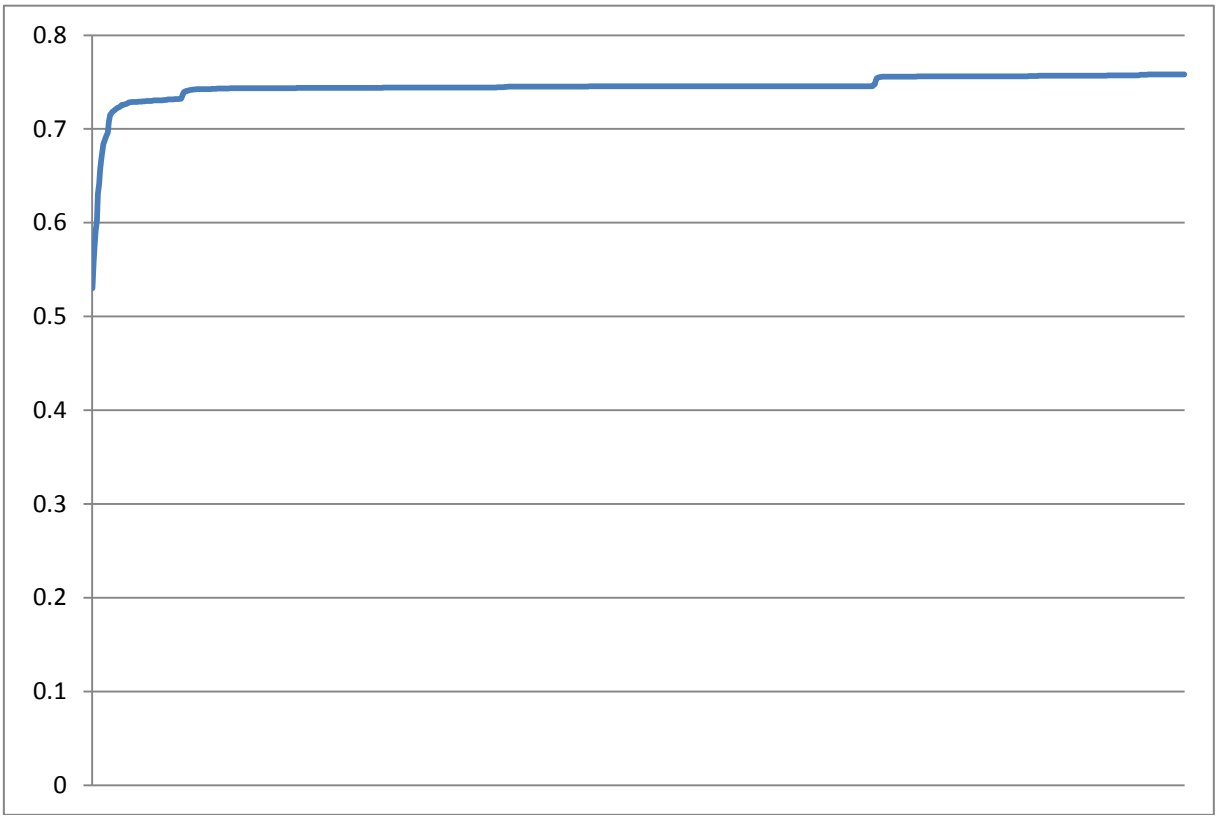
	Klassen- / Ressourcen-Anzahl	Wahrscheinlichkeit einer falschen Zuteilung Ressourcen	Ergebnis Algorithmus
Oberstufe	Wenige Klassen & Ressourcen	Klein	Schlecht
Unterstufe	Mehr Klassen & Ressourcen	Mittel	Schlechter
Locations	Viele Klassen & Ressourcen	Gross	Am Schlechtesten

Je mehr Ressourcen (Lehrer, Räume) es gibt, desto wahrscheinlicher ist es, dass die zufällige Zuteilung falsch ist. Das heisst auch wenn das Szenario einfach ist, aber viele Ressourcen hat, funktioniert unser Algorithmus schlecht.

Von Algorithmus ausgesehen sollte sich das Problem mit mehr Generationen oder besserer Parameter Wahl wahrscheinlich lösen lassen. Vorausgesetzt die Fitness-Funktion gibt ein gutes Optimierungs-Ziel für diesen Sachverhalt vor. Viel einfacher wäre es jedoch, diese Schwäche durch zusätzliche, heuristisch gesteuerte Einschränkungen zu beheben.

Verbesserungs-Graphen

Immer interessant bei genetischen Algorithmen ist, wie die Verbesserungs-Kurve verläuft. Bei unserem Experiment sehen wird, dass am Anfang eine rasante Verbesserung stattfindet. Aber das Ganze recht schnell stagniert.



8.5 Fazit

8.5.1 Heuristischer Algorithmus

Ergebnis

Der heuristische Algorithmus schneidet wie erwartet gut ab. Für die meisten Szenarien kann er akzeptable Ergebnisse liefern. Die Ergebnisse sind jeweils mit geringen manuellen Änderungen benutzbar.

Man muss aber klar sehen, dass für unsere getesteten Szenarien der Heuristische Algorithmus klar bevorteilt ist. Es wurde viel Zeit, Fach-Wissen und Energie in den Algorithmus gesteckt. Damit ist der Algorithmus gut abgestimmt auf diese Szenarien. In anderen Szenarien (z.B. Hochschule) würde dieser Algorithmus vermutlich schlecht abschneiden.

Stärken

Die Stärke des Heuristischen Algorithmus ist, dass man Fachwissen und Tricks der manuellen Stundenplanerstellung in den Algorithmus eingebaut hat. Es ist auch möglich noch weiteres Fachwissen in die Heuristiken einzubauen und somit den Algorithmus zu verbessern.

Schwächen

Der Algorithmus ist extrem spezifisch auf das Problemgebiet angepasst. Er kann nur Stundenpläne für die Grundschule gut erstellen. Der Algorithmus ist nicht auf ein breites Problemfeld anwendbar. Schon bei den angeschauten Szenarien würde es sich lohnen, gewisse Heuristiken ein oder aus zu schalten. Um ein breites Problemgebiet abzudecken, muss man viele Heuristiken erstellen und diese zusätzlich noch gut kombinieren. Das macht diesen Ansatz schlecht skalierbar.

Erfolgsversprechende Erweiterungen

Prinzipiell können weitere Heuristiken den Algorithmus verbessern oder ihn für neue Problemgebiete vorbereiten. Desweiteren ist es sehr nützlich zu wissen für welches Szenario der Algorithmus verwendet werden soll. Weiss man dies, können Heuristiken dem Szenario entsprechend ein oder ausgeschaltet werden. Damit werden immer nur geeignete Heuristiken angewandt, wodurch sich das Ergebnis zusätzlich verbessert.

Erfolgsversprechende Kombinationen

Den Graphen-Algorithmus zu verwenden für die Platzierung der Stunden ist eine gute Option. Denn der Graph-Algorithmus kann garantieren, dass die Platzierung Konfliktfrei ist.

8.5.2 Graphentheorie basierter Algorithmus

Ergebnis

Dieser Algorithmus löst eine Aufgabe perfekt. Er kann beliebig viele Lektionen garantiert und deterministisch konfliktfrei platzieren. Alles andere wird komplett ignoriert. So ist die Verteilung der Lektionen über die Woche komplett unbrauchbar. Der Verteilungsaspekt muss mittels anderer Verfahren ergänzt werden.

Stärken

Die Stärke des Algorithmus das er deterministisch die Lektionen konfliktfrei platziert. Ausserdem ist der Algorithmus extrem schnell. Wenn z.B. einen Initial konfliktfreien Stundenplan für die anschliessen manuelle Verteilung möchte, kann dieser Ansatz verwendet werden.

Schwächen

Dieser Algorithmus ignoriert Aspekte, wie Lektionen-Verteilung, Lektionen-Blöcke etc. komplett. Das heisst, er wird keine balancierten Stundenpläne erstellen. Daher sind die Stundenpläne nur mit diesem Verfahren generiert in der Praxis meistens unbrauchbar.

Erfolgsversprechende Erweiterungen

Prinzipiell lässt sich dieser Algorithmus durch den Einsatz von zusätzlichen Heuristiken stark verbessern:

- Die Farben, welche Blöcke repräsentieren, kann man heuristisch oder mit anderen Verfahren verteilen. Somit würden sich die Lektionen besser verteilen.
- Gewisse Kriterien und Spezialfälle kann man als zusätzliche Konfliktkanten implementieren
- Anstatt deterministisch die erste mögliche Lösung zu akzeptieren, kann man mehrere Lösungen der Graphen-Färbung vergleichen und auswählen.

Erfolgsversprechende Kombinationen

Kombinationen mit heuristischen Ansätzen verbessern das Resultat erheblich.

8.5.3 Kombinierter Algorithmus (Graph, Heuristiken)

Ergebnis

Zur unserer Überraschung funktioniert eine einfache Kombination des Graph- und des Heuristischen Algorithmus sehr gut. Die Schwäche bei der Lektionen-Verteilung ist fast komplett behoben. Das ist insofern überraschend, weil die Initiale Platzierung extrem schlecht ist.

Stärken

Man kann die deterministische, konfliktfreie Lösung als perfekte Ausgangs-Lage für Heuristiken verwenden. Dadurch können gewisse Heuristiken vermieden oder vereinfacht werden.

Schwächen

Die Heuristiken können die Initiale konfliktfreie Lösung zerstören, so dass der grosse Vorteil des Graphe-Algorithmus zunichte gemacht wird.

Erfolgsversprechende Erweiterungen

Erfolgversprechend ist es eventuell, gewisse Heuristiken direkt in die Graphen-Traversierung und Färbung ein zubinden.

8.5.3.1 Genetischer Algorithmus

Ergebnis

Wie wir vermutet haben schneidet dieser Algorithmus sehr schlecht ab. Der Algorithmus ist in dieser Form nicht brauchbar.

Stärken

Grundsätzlich ist der Algorithmus extrem simpel und das Prinzip bestechend einfach. Desweiteren kann man mit den Parametern wie Mutation, Generations-Grössen versuchen, den Algorithmus zu optimieren.

Schwächen

Der Algorithmus hat viele Schwächen:

- Die Fitness-Funktion ist eine der treibenden Kräfte. Doch wie weiss man, dass man eine gute Fitness-Funktion hat. Vermutlich ist diese ziemlich heuristisch Aufgebaut. Somit hat man im Prinzip heuristische Verfahren indirekt angewendet. Es ist aber viel einfacher, Heuristiken direkt an zu wenden.
- Die Parametrisierung ist extrem wichtig und zeitintensiv. Man müsste extrem viele Permutationen der Parameter laufen lassen und die Resultate vergleichen um eine optimale Parametrisierung zu erreichen.
- Der Algorithmus ist Design-bedingt extrem langsam, da man viele Generationen bilden muss, um ein brauchbares Resultat zu erreichen.

Erfolgsversprechende Erweiterungen

Es würde sich extrem lohnen gewisse Heuristiken einzubauen, was ein gültiges Gen / Mutation ist. Z.B. das der Algorithmus korrekte Räume zuordnen kann. Dadurch würde sich der Lösungs-Raum dramatisch verkleinern und somit die Lösung verbessern.

Desweiteren könnte es sich eventuell lohnen, die Gen-Codierung und Genom-Verschmelzung zu verbessern.

Erfolgsversprechende Kombinationen

Es gibt mehre Interessante Kombinationen:

- Anstatt dass man Initial zufällige Stundenpläne als Input nimmt, nimmt man viele alte Referenz-Stundepläne. Somit hat man schon einen guten Satz Genome von Anfang an.
- Man nimmt einer der anderen Algorithmen um Input-Stundenpläne zu generieren. Man schaut dann, ob man durch Mutation dieser Stundenpläne noch Verbesserungen erreichen kann.

9 Neuen Algorithmus einbauen

9.1.1 Modularisierung-Konzept

9.1.2 Inversion of Control / Dependency Injection

Um einzelne Komponenten der Applikation zu verbinden, wird das Dependency Injection Pattern angewendet (http://martinfowler.com/articles/injection.html / http://en.wikipedia.org/wiki/Dependency_Injection). Dabei wird ein klares Interface für einen Service definiert. Wenn dieser Service gebraucht wird, wird einfach dieses Interface im Konstruktor des Service-Konsumentens deklariert. Beim Start der Applikation wird automatisch dafür gesorgt, dass die Implementation dieses Interfaces instanziiert und übergeben wird. Der Konsument des Services muss sich nicht darum kümmern. Zusätzlich übernimmt die Applikation das Lebenszyklen-Management von Services, z.B. garantiert er, dass nur eine Instanz eines Service erstellt wird.

Also Dependency Injection Container wird Autofac verwendet: www.autofac.org. Es gibt zusätzlich noch Erweiterungen, um deklarativ mittels .NET-Attributen Services zu beschreiben.

9.1.3 Extension-Points

Um das Inversion Of Control Prinzip umzusetzen gibt es die sogenannten Extension-Points. Man kann Interfaces etc. als Extension-Point deklarieren. Nun können sich beliebig viele Services bei diesem Extension-Point anmelden. Am Schluss werden alle registrierten Services zum richtigen Zeitpunkt aufgerufen. So werden z.B. alle Services beim Start der Applikation aufgerufen welche am Extension-Point ‚IEagerStarted' angemeldet sind.

9.1.4 Generische Factories (Providers)

Der IoC-Container stellt generische Factories zur Verfügung. Dazu fordert man in der Client-Komponente nicht den Service direkt an, sondern eine Factory für ein Service. Danach kann man auf der Factory-Instanz beliebig neue Instanzen der Services erzeugen. Der IoC-Container übernimmt das erzeugen einer entsprechenden Factory. Dabei kann man Factories ohne oder mit Parameter erzeugen. Da einfache Factories oft vorkommen, ist s sehr praktisch, dass dieses Pattern von Container implementiert wird.

Die Factory ist zusätzlich nur ein Delegate. Daher kann man in Unit-Tests einfache Lambda-Expressions übergeben, was das ganze sehr einfach gestaltet.

9.1.5 Modul

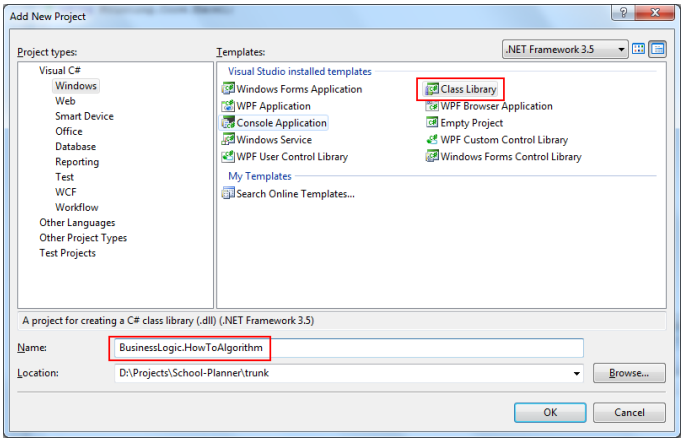
Die Applikation wird in lauter kleine Module unterteilt, die als separate Visual Studio Projekte geführt werden. Dem entsprechend werden diese auch zu separaten Assemblies kompilierte. Jede funktionale Einheit, die unabhängig von anderen Teilen der gleichen Schicht ist, wird als ein eigenes Modul geführt. Z.B. gibt es ein Modul für die Stammdaten-Erfassung und ein Modul für die Kalender-Ansicht. Diese Konvention wird auf jeder Schicht angewendet. So gibt es zum Beispiel folgende Module: View.MasterData, View.Calendar, Application. MasterData, Application.Calendar, BusinessLogic. MasterData, BusinessLogic.Calendar. So ist jedes Paket in der Logischen Architektur als eigenes Modul implementiert.

Jedes Modul hat als Einstiegspunkt eine Implementation des IModule-Interfaces zur Verfügung zu stellen. Dieses wird beim Start der Applikation aufgerufen. Es soll die Services welche dieses Modul zur Verfügung stellt beim Dependency-Injection-Container registrieren. Es darf auf keinen Fall aktiv beginnen, irgendwelche Applikations-Logik auszuführen. Im Normalfall wird eine Default-Implementierung genommen, so dass die Services automatisch gesucht werden und nicht manuelle deklariert werden müssen.

9.2 HowTo für eigenen Algorithmus

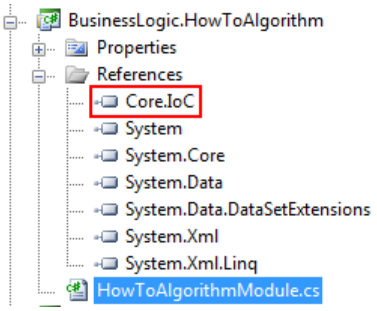
9.2.1 Projekt erstellen

Es ist vorgesehen, dass für jeden eigenen Algorithmus ein eigenes Projekt erstellt wird. Das heisst jeder Algorithmus hat sein eigenes Assembly. Dadurch kann man beim Programmstart entscheiden welche Algorithmen man alle zu Verfügung haben möchte. Denn es werden nur die Algorithmen geladen deren Assemblies sich im Ausführungspfad befinden.



Im Visual Studio kann der Solution ein neues Class Library Projekt hinzugefügt werden. Die Namenskonvention ist dabei BusinessLogic.XXX.

9.2.2 Modulklasse erstellen



Damit das Dependency Injection System funktionieren kann, muss zuerst das IModule Interface von einer Klasse implementiert werden. Dazu gibt es bereits die Default Implementation AttributeBasedModule von welcher abgeleitet wird. Das AttributeBasedModule befindet sich im Core.IoC Projekt. Daher muss dieses Projekt Referenziert werden, damit Klassen dieses Projektes verwendet werden können.

Der Code in der HowToAlgorithmModule Klasse sieht folgendermassen aus:

```
using Core.IoC;
namespace BusinessLogic.HowToAlgorithm
{
    public class HowToAlgorithmModule : AttributeBasedModule { }
}
```

9.2.3 Algorithmus Klasse erstellen

Nun möchten wir eine Implementation der IAlgorithm Klasse erstellen. Diese Klasse ist im BusinessLogic.AlgorithmInterface Projekt definiert. Folglich muss auch dieses Projekt referenziert werden um diese Klasse verwenden zu können.

Das IAlgorithm Interface deklariert zwei Funktionen:

- Name : String
- Run(ScheduleGrid) : ScheduleGrid

Der Wert des Properties ‚Name' wird im UI verwendet für den Menü Punkt dieses Algorithmus. Dieser Name sollte also möglichst vielsagend gewählt werden.

In der Run-Methode kann nun der eigene Algorithmus implementiert werden. Der Mechanismus ist folgendermassen:

Sobald im Menu der Menü Punkt dieses Algorithmus angewählt wird, wird die Run Methode aufgerufen. Dabei werden alle Daten die der Algorithmus benötigt übergeben. Der Algorithmus gibt am Schluss ein ‚ScheduleGrid' zurück. Das Resultat wird dann in die Datenbank zurück geschrieben.

Der Code sieht nun folgendermassen aus:

```
using BusinessLogic.AlgorithmInterface;
using BusinessLogic.AlgorithmInterface.DataTypes;
namespace BusinessLogic.HowToAlgorithm
{
    public class HowToAlgorithm : IAlgorithm
    {
        public string Name
        {
            get { return "How To Testalgorithmus"; }
        }

        public ScheduleGrid Run(ScheduleGrid grid)
        {
            //do nothing and return the empty grid
            return grid;
        }
    }
}
```

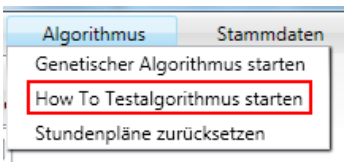
9.2.4 Anhängen zum ExtensionPoint

Damit nun der Menüeintrag und die Aufrufsemantik automatisch aufgesetzt wird, muss der Algorithmus noch dem ‚IAlgorithm' ExtensionPoint angehängt werden. Dies wird über das verwenden von Attributen erreicht.

Hier der Code:

```
using BusinessLogic.AlgorithmInterface;
using BusinessLogic.AlgorithmInterface.DataTypes;
using Core.IoC;
namespace BusinessLogic.HowToAlgorithm
{
    [Service] //Um dem DependencyInjection System anzuzeigen, dass es sich hier um einen Service
handelt der Injected werden kann
    [AddToExtensionPoint(typeof(IAlgorithm))] //Diese Implementation dem ExtensionPoint von IAlgorithm
anhängen
    public class HowToAlgorithm : IAlgorithm
    {
        …
    }
}
```

9.2.5 Ausführen



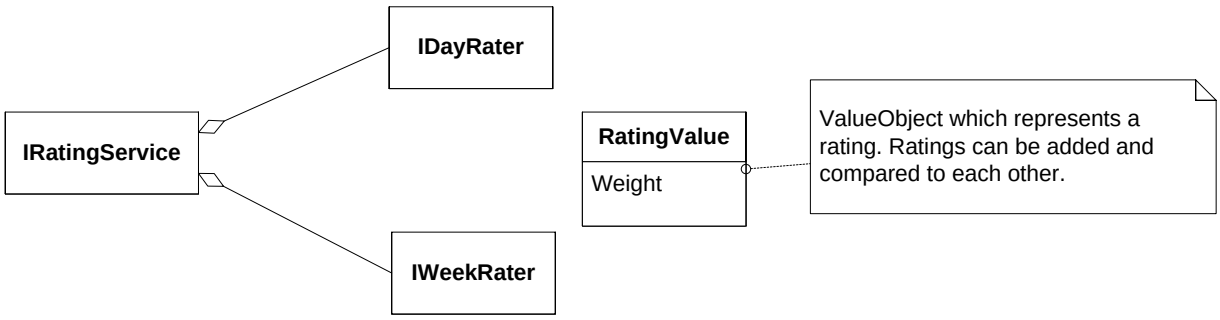
Bevor nun das ganze Ausgeführt wird, muss dieses Projekt noch dem Startup Projekt ‚MySchoolPlanner' hinzugefügt werden. Dadurch wird das Assembly unseres Projektes in den Ausführungspfad kopiert.

Nun kann die Applikation gestartet werden und wir sehen unseren Algorithmus im Algorithmen Menü. Sobald er ausgewählt wird geschehen folgende Schritte:

- Es werden alle Stammdaten aus der Datenbank gelesen und in ein ‚AlgorithmData' Objekt abgepackt.
- Der Algorithmus wird aufgerufen.
- Der Rückgabewert des Algorithmus wird in die Datenbank zurück geschrieben.

10 Rating-Framework

10.1 Komponenten



10.1.1 Der Rating-Service

Die Komponente ist der Einstiegpunkt für Konsumenten. Beim ‚IRatingService' kann man ein Rating für einen Tag oder einen ganzen Stundenplan verlangen. Dazu übergibt man dem ‚IRatingService' den aktuellen Stundenplan (eine ‚ScheduleGrid'-Instanz). Der RatingService nimmt alle bekannten Rater und übergibt den Stundenplan zur Bewertung an diese. Am Schluss sammelt er die Resultate der einzelnen Rater, berechnet die Gesamtbewertung und gibt diese zurück.

10.1.2 Die Rater

Ein Rater kann einen Tag, eine Woche oder beides bewerten. Dazu implementiert er die entsprechenden Interfaces. Er wird von Rating-Service aufgerufen mit den Daten die er zu bewerten hat. Ein Rater muss sich wie eine Mathematische Funktion verhalten. Er darf nur anhand der übergeben Parameter eine Bewertung vornehmen. Er darf kein Status behalten oder Seiteneffekte verursachen.

10.1.3 RatingValue

Ein Rating-Value repräsentiert das Resultat einer Bewertung. ‚RatingValues' können zusammengezählt und verglichen werden. Zusätzlich enthält ein ‚RatingValue' ein Gewicht. Dieses Gewicht sagt aus, wie stark sich dieser ‚RatingValue' auswirkt beim summieren mit anderen ‚RatingValues'. Mit diesem Gewicht wird der Einfluss der verschiedenen Rater reguliert.

10.2 Implementierte Raters

10.2.1 AlwaysSameTeacherUsedRater

Man will normalerweise erreichen, dass ein Fach immer beim gleichen Lehrer gehalten wird. Diese Anforderung wird von diesem Rater überprüft. Sollte dies nun nicht der Fall sein gibt es einen exponentiellen Abzug abhängig davon wie oft diese Anforderung missachtet wurde.

10.2.2 BlockSizeRater

Falls für Lektionen gewisse Blockgrössen (aneinander hängende Stunden vom selben Fach) gefordert werden, schaut sich dieser Rater an ob diese Anforderungen eingehalten sind. Falls dies nicht der Fall ist gibt es einen exponentiell grösser werdenden Bewertungsabzug.

10.2.3 ClassGapRater

Schaut die Stundenpläne der einzelnen Klassen an und überprüft diese auf mögliche Zwischenstunden (Leerstunden zwischen zwei Schulstunden). Dieser Rater gibt exponentiellen Abzug für die Anzahl Zwischenstunden, wobei eine grössere Anzahl zusammenhängender Zwischenstunden einen grösseren Abzug geben.

10.2.4 ClassLocationChangeRater

Hier wird angesehen wie oft eine Klasse die Lokalität wechseln muss. Ziel ist es möglichst immer im selben Gebäude Schule zu haben und die Ortwechsel möglichst gering zu halten. Es gibt für jeden Ortswechsel einen exponentiell grösser werdenden Bewertungsabzug.

10.2.5 ClassTeacherUsedRater

Überprüft ob für alle Stunden die der Klassenlehrer geben kann auch wirklich von diesem Übernommen werden. Für die Anzahl Stunden die der Klasselehrer geben könnte, jedoch von einem anderen Lehrer gehalten werden, gibt es einen exponentiellen Abzug.

10.2.6 ConflictRater

Sobald eine Ressource wie Lehrer oder Raum doppelt gebucht ist, gibt dieser Rater einen exponentiellen Abzug für die Anzahl Verletzungen die aufgetreten sind.

10.2.7 EverythingPlacedRater

Dieser Rater überprüft ob alle Stunden die für eine Klasse verbucht werden müssen auch wirklich verbucht worden sind. Für die Anzahl Stunden die verplant hätten werden sollen aber nicht geplant wurden gibt es einen exponentiellen Abzug.

10.2.8 HasLessonForDayRater

Überprüft ob an jedem Schultag Schulstunden stattfinden, ist dies für ein Tag nicht der Fall gibt es eine extrem schlechte Bewertung, da dann dieser Tag bei der Stundenverteilung komplett ignoriert wurde.

10.2.9 LessonFollowerRater

Überprüft ob auf eine Lektion auch nur die Lektionen folgen, welche folgen dürfen. Wenn eine Lektion einer anderen Lektion folgt, die eigentlich nicht nachfolgen dürfte, gibt es einen exponentiell grösser werdenden Abzug pro Verletzung.

10.2.10 LessonAtDayBalancingRater

Man schaut an, ob die einzelnen Schultage einer Klasse ungefähr gleich viele Schulstunden beinhalten. Falls dies nicht der Fall ist, gibt es einen Abzug bei der Bewertung. Der Abzug wird durch die durchschnittliche Abweichung über alle Tage gesehen ausgedrückt.

10.2.11 NotTooManySchoolHoursADayRater

Um eine möglichst hohe Lernkurve zu erreichen, versucht man nicht zu viele Stunden an einem Tag zu halten. Es ist momentan ein Limit von Acht Schulstunden pro Tag gesetzt. Es basiert auf der Annahme, dass dies die maximale Anzahl Schulstunden pro Tag ist bei dem einen Schüler auch am Ende des Tages noch aufnahmefähig ist. Dieser Rater gibt einen linearen Abzug für jeden Schultag einer Klasse, der nicht den Anforderungen entspricht.

10.2.12 PlacedAtPreferredLessonTimesRater

Der Rater überprüft ob die Lektionen wirklich zu den bevorzugten Zeiten stattfinden. Es gibt für jede Stunde die nicht zu einem bevorzugten Zeitpunkt stattfindet einen Abzug der sich logarithmisch verhält. Wobei eine Stunde die zu einem möglichen Zeitpunkt gehalten wird weniger Abzug gibt, als eine Stunde die zu einem gesperrten Zeitpunkt stattfindet.

10.2.13 PlacedAtPreferredTeacherTimesRater

Dieser Rater überprüft ob die Lektionen an den von Lehrer bevorzugten Zeiten stattfinden. Ansonsten verhält er sich gleich wie der ‚PlacedAtPreferredLessonTimesRater'.

10.2.14 ProhibitedPositionsRater

Rater der Abzug gibt wenn eine Stunde zu einer gesperrten Zeit stattfindet.

10.2.15 RoomOverbookedRater

Hier wird überprüft ob die Lastverteilung unter den Räumen ungefähr ausgeglichen ist. Es gibt bei unfairer Lastverteilung unter den Räumen einen Abzug in Form der durchschnittlichen Abweichung zu der durchschnittlichen Auslastung der Räume.

10.2.16 RoomRequirementRater

Der Rater überprüft ob die Anforderungen an den Raum der geplanten Stunden auch wirklich erfüllt werden. Das heisst wenn beispielsweise eine Turnstunde in einem Klassenzimmer abgehalten wird, so gibt das einen grossen Abzug in der Bewertung.

Die Bewertung wird logarithmisch vorgenommen, da bereits einmalige Missachtungen der Raumanforderungen ein grosses Problem darstellen, ob dies dann sogar mehrmals vorgekommen ist, spielt lediglich eine untergeordnete Rolle.

10.2.17 SameLessonTwiceRater

Schaut das die gleiche Lektion nicht mehrmals am Tag vorkommt. Es wird jedoch zusätzlich berücksichtigt, dass es je nach Konfiguration möglich ist, dass gewisse Stunden mehrmals an einem Tag vorkommen. Dies ist beispielsweise der Fall, wenn man Lektionen mit grösserer Sollstundenanzahl als Anzahl Schultage hat. Das heisst, die maximal mögliche Anzahl Lektionen vom gleichen Typ pro Tag werden folgendermassen berechnet:

- Ceiling(Anzahl Stunden pro Woche / Anzahl Schultage)

10.2.18 SingleCellForLesson

Überprüft, dass keine Lektionen parallel stattfinden, wenn dies nicht explizit geplant ist.

10.2.19 TeacherGapRater

Macht dasselbe wie der ‚ClassGapRater' wobei hier nicht der Klassenstundenplan auf Zwischenstunden überprüft wird sondern der Stundenplan eines Lehrers.

10.2.20 TeacherLocationChangeRater

Hier wird dasselbe gemacht wie beim ‚ClassLocationChangeRater' wobei hier die Standortwechsel eines Lehrers angesehen werden und nicht die einer Klasse.

10.2.21 TeacherUsesAssignedRoom

Der Rater schaut an, ob der Lehrer seinen bevorzugten Raum benutzt oder nicht. Einzige Ausnahme ist bei Räumen welche die Anforderungen der Lektion erfüllen. Jedes Mal wenn ein falscher Raum verwendet wird, gibt es einen exponentiell grösser werdenden Abzug.

10.2.22 TeacherWorkloadRater

Ein Lehrer ist normalerweise maximal 100% angestellt und das entspricht 30 Schul-Stunden pro Woche. Dieser Rater überprüft nun ob jeder Lehrer wirklich nicht mehr als diese 30 Schulstunden unterrichtet. Für jeden Lehrer der mehr als diese 30 Stunden unterrichtet gibt es eine exponentiell schlechter werdende Bewertung.

10.3 Weitere Rater einbauen

Das Einbauen eines Raters beruht auf denselben Prinzipien wie das Einbauen eines neuen Algorithmus. Siehe Kapitel 9.2.

Zuerst wird die Klasse für den Rater erstellt. Alle Rater befinden sich im gleichen Namespace (‚BusinessLogic.AlgorithmInterface.Rating'). Die Klasse muss entweder ‚IDayRater', ‚IWeekRater' oder beide Interfaces implementieren. Danach muss man die Klasse an den entsprechenden ‚ExtensionPoints' registrieren:

```
namespace BusinessLogic.AlgorithmInterface.Rating
{
    [Service]
    [AddToExtensionPoint(typeof(IDayRater))]
    public class ExampleRater : IDayRater
    {
        public string Name
        {
            get { return "Example-Rater"; }
        }

        public RatingValue ForDay(IEnumerable<ICell> cellsOfDay)
        {
            return RatingValue.Percent(1.0);
        }
    }
}
```

11 Zusätzliche Framework Features

11.1 Stammdaten

11.1.1 CRUD-Umgebung

Um die Stammdaten zu erfassen wurde eine CRUD-Umgebung erstellt. Dies ist extrem wichtig, damit die Daten überhaupt erfasst werden können für die Algorithmen. Für komfortables Editieren werden editierte Daten sofort übernommen in andere Ansichten.

11.1.2 Aktualisierung der interaktiven Stundenplan Erstellung

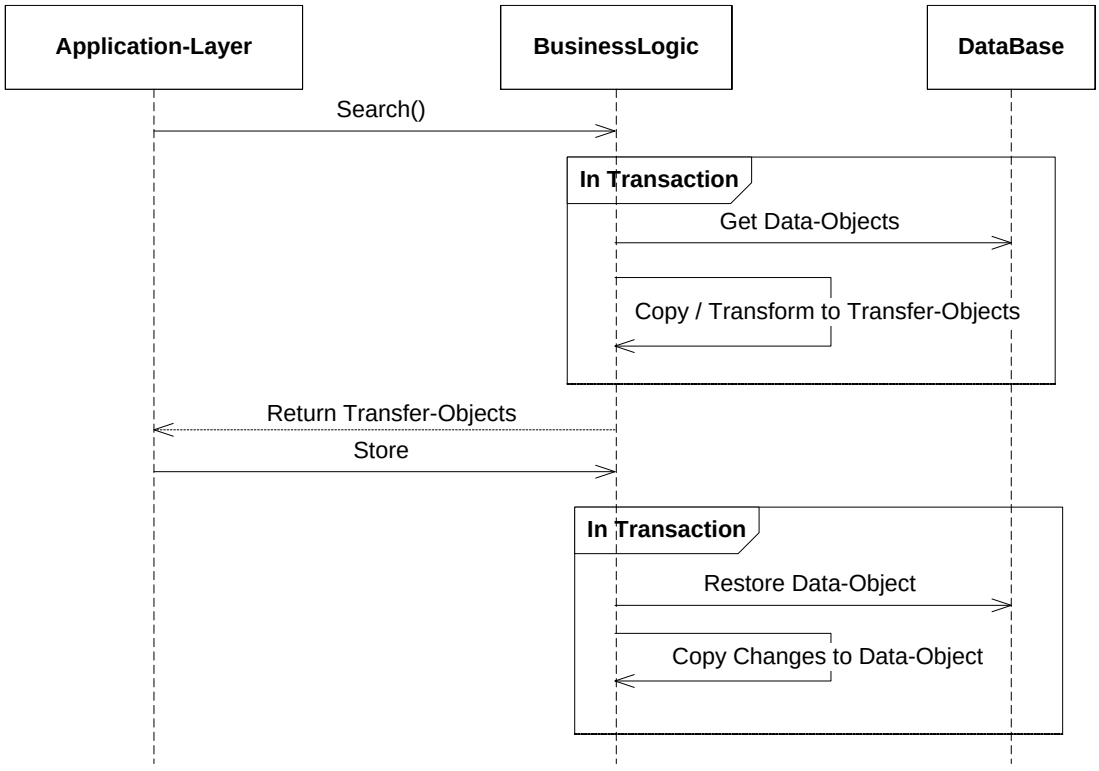
Um die Algorithmen mit genügend Information zu versorgen sind viele Stammdaten zu erfassen. Diese Daten werden ebenfalls für die manuelle Stundenplanerstellung benutzt.

11.1.3 CRUD-Design

Beim Design des CRUD-Frameworks ist wichtig, das keine Daten-Model-Klassen bei der View-/Applikations-Schicht direkt verwendet werden. Das hat drei Gründe.

- Erstens muss es möglich sein, dass sich das Daten-Model ändert ohne dass alle Views angepasst werden.
- Zweitens verhindert es, dass die Daten-Model-Objekte ausserhalb einer Transaktion manipuliert werden. Sobald man Daten-Model-Objekte ausserhalb der Business-Logik-Schicht zulässt, verliert man die Kontrolle über diese Objekte.
- Drittens ist es einfacher Unit-Tests zu fahren. Denn man muss nicht die Datenbank zur Verfügung stellen um höher Schichten zu testen.

11.1.4 Ablauf CRUD

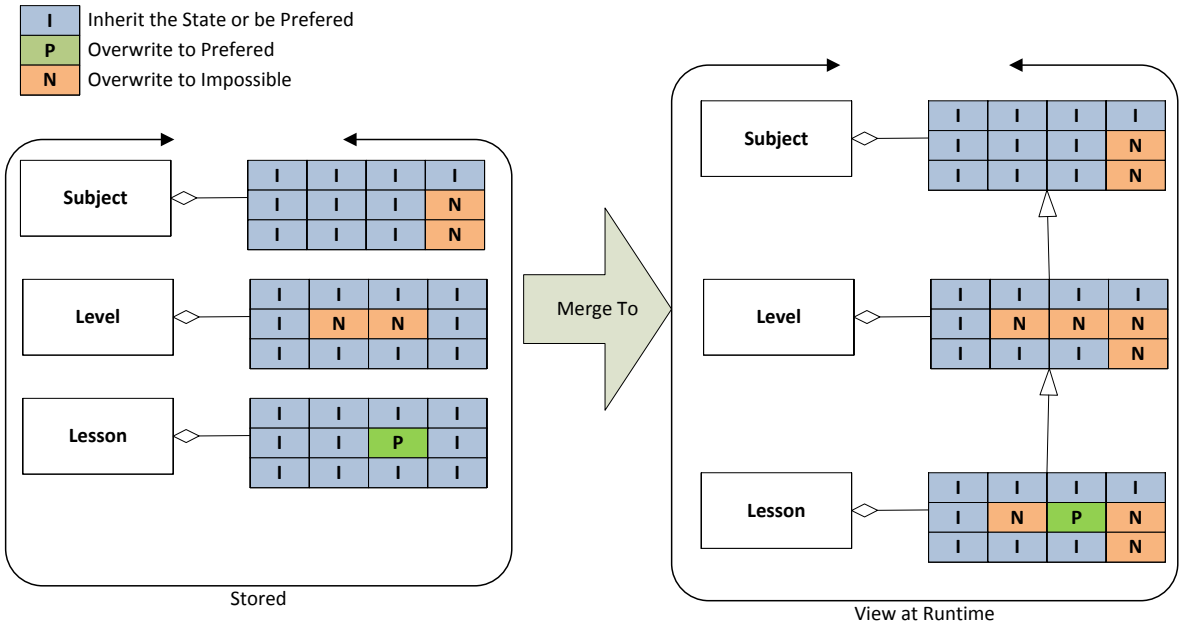


11.1.5 Implikationen diese Models

Bei Anfragen an die Business-Logik werden immer neue Transfer-Objekte erstellt. Somit ist die Identität diese Objekte wertlos. Dies hat Folgen: Man kann kein Observer-Pattern auf diese Objekte erstellen. Darum registrieren sich die Observer eine Stufe höher. Bei Änderungen wird nochmals das komplette Objekt übertragen.

11.2 Vererben der Zeitausprägung

Bei Stundenplänen gibt es meistens Einschränkungen, wann gewisse Lektionen stattfinden dürfen oder eben nicht. Diese Einschränkungen beziehen sich auf verschiedene Stammdaten. So kann sich eine Einschränkung auf ein Fach (Z.B. Deutsch), auf eine Stufe (Z.B. 1. Primar) oder auf konkrete Lektionen (Z.B. Deutsch auf 1. Primar-Stufe) beziehen. Ein typisches Beispiel für solch eine Einschränkung ist, dass in der Unterstufe keine Lektionen am Mittwochnachmittag stattfinden. Um diese Einschränkungen komfortabel zu editieren gibt es eine Art ‚Vererbung'. Wenn eine Einschränkung für die 1. Primar gilt, wird diese übernommen für alle Lektionen der ersten Primar. Optional kann eine vererbte Zeitausprägung explizit wieder aufgehoben werden.

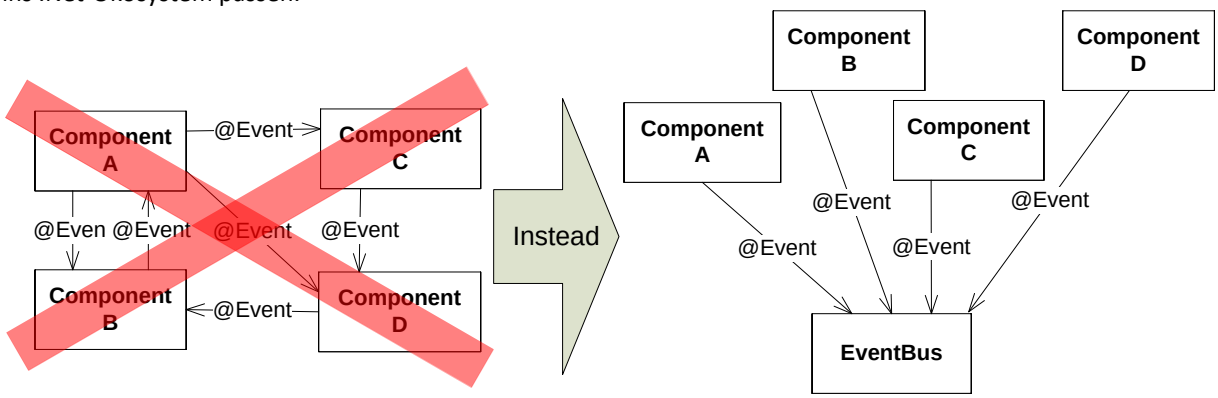


11.3 Event-Bus

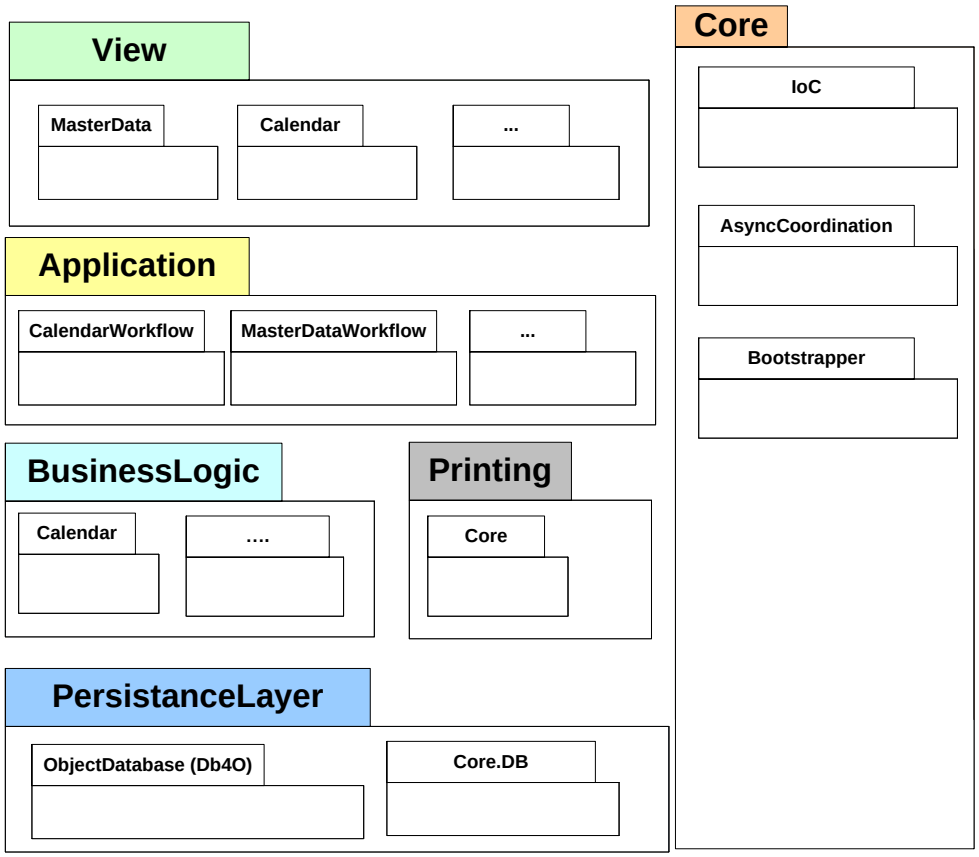
Events sind extrem wichtig in unserem Framework. Mit ihnen werden Aktualisierungen im System propagiert. Mit der zunehmenden Komplexität gibt es immer mehr Events und Event-Erzeuger. Die Event-Konsumenten müssen wissen welche Komponente im System welche Events erzeugt. Bei zunehmender Anzahl von Komponenten wird das immer problematischer. Es müssen sich immer mehr Komponenten gegenseitig kennen, nur um die Events abzufangen.

Darum haben wir uns entschieden, einen zentralen ‚Event-Bus' einzuführen (Einen Mediator für die Events). Damit gibt es eine zentrale Stelle an der man sich für Events anmelden kann.

An den Schnittstellen zur Applikations-Schicht werden weiterhin klassische Events benutzt, da sie wunderbar ins .Net-Ökosystem passen.



11.4 Strukturierung der Applikation



11.4.1 View

Diese Schicht beinhaltet alle grafischen Oberflächen. Dazu gehören alle GUI-Definitionen und GUI-Komponenten die verwendet werden. Zusätzlich enthält sie das Event-Handling, welches für die grafische Bedienung nötig ist.

11.4.2 Application

Diese Schicht kontrolliert die Abläufe der Applikation und koordiniert die Use Cases. Dies beinhaltet welche Views wann angezeigt werden, welche Daten dazugehören und was bei einer User-Aktion passieren soll. Ebenfalls verwaltet diese Schicht das View-Model (MVVM-Pattern).

11.4.3 BusinessLogic

Diese Schicht kümmert sich um das Domain-Model und alle Logischen Abläufe welche die Daten manipulieren. Das Algorithmen-Framework, die Algorithmen selbst, das Rating-System, Konflikterkennung und Stammdaten-Verwaltung sind in dieser Schicht implementiert.

11.4.4 Printing

Diese Schicht kümmert sich um das drucken von Stundenplänen und Bewertungen. Von der logischen Ebene her gesehen befindet sie sich auf derselben Stufe wie die ‚BusinessLogic'.

11.4.5 Persistence Layer

Dieser Layer kümmert sich um das Persistieren der Daten. Hier befindet sich die Objekt-Datenbank db4o sowie zusätzliche Funktionen die für das Persistieren wichtig sind.

11.4.6 Core

Hier befinden sich Features, die Elementar für die Applikation sind, aber nicht zu einer Logischen Schicht gehören. Diese Logik wird von allen Schichten benutzt.

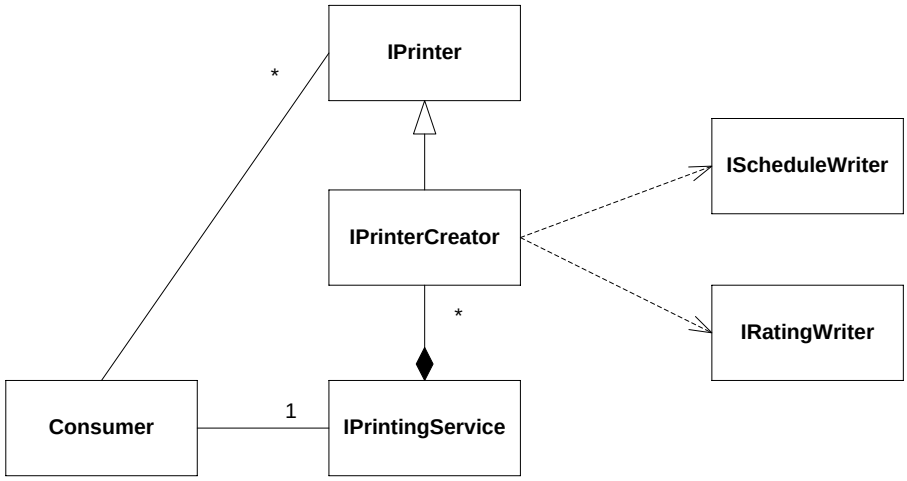
11.5 Printing System

Mit der ‚School-Planner' Software können komfortabel Stundenpläne erstellt werden. Sobald diese Erstellt worden sind, wird jedoch auch noch eine Möglichkeit benötigt die Stundenpläne in einem Format zu exportieren.

11.5.1 Formate

Das Drucksystem unterstützt im moment das Excel-Format für Excel 2000 oder neuer.

11.5.2 Design



Der ‚Consumer' kennt nur die Liste von verfügbaren Printern und den ‚Printingservice'. Der Consumer der School-Planner Applikation kreiert aus den vorhandenen Printern jeweils direkt die Menüeinträg. Jeder dieser Menüeinträge ist mit einem Printer verbunden und wenn der Benutzer einen auswählt, wird der ‚Printingservice' mit dem gewünschten Printer aufgerufen.

Der ‚PrintingService' Sucht dann den benötigten ‚IPrinterCreator' anhand der ‚IPrinter' Information, welches der ‚Consumer' mitgeliefert hat. Über den ‚IPrinterCreator' kann der gewünschte Printer geholt werden, dies kann einerseits ein Stundenplan Printer oder ein Rating Printer sein.

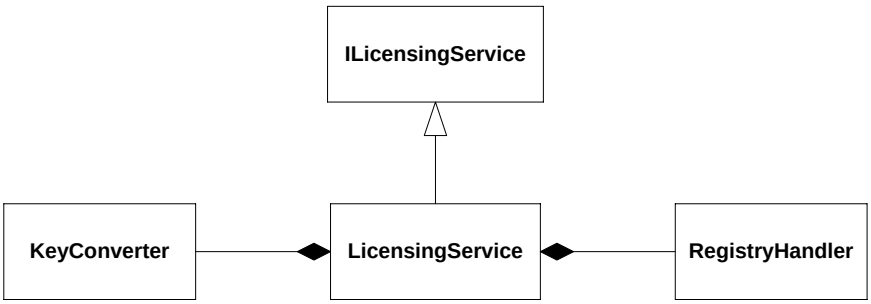
11.5.3 Neuen Printer hinzufügen

Möchte man einen neuen Printer hinzufügen, so muss lediglich eine neue implementation der Interfaces ‚IPrinterCreator', ‚IScheduleWriter' und ‚IRatingWriter' erstellt werden. Zudem muss das Ganze noch korrekt zu den jeweiligen Extensionpoints des Dependency Injection Systems angehängt werden. Hier ein kurzer Beispielcode wie das für den Excel Export gemacht ist:

```
[Service]
[AddToExtensionPoint(typeof(IPrinterCreator))]
[AddToExtensionPoint(typeof(IPrinter))]
internal class ExcelWriterCreator : IPrinterCreator
{
    private static readonly PrinterId ExcelId = PrinterId.Create("Excel", "xls");
    #region IPrinterCreator Members

    public PrinterId Id
    {
        get { return ExcelId; }
    }

    public IScheduleWriter CreateScheduleWriter(Stream writer)
    {
```

```
            return new ExcelScheduleWriter(writer);
        }

        public IRatingWriter CreateRatingWriter(Stream writer)
        {
            return new ExcelRatingWriter(writer);
        }
        #endregion
    }
```

11.6 Lizensierungssystem

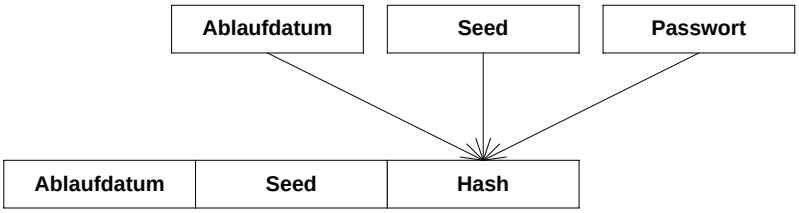


Um das Lizensierungssystem verwenden zu können wird lediglich das ‚ILicensingService' Interface verwendet. Darauf kann man alle benötigten Operationen wie, Lizenzschlüssel überprüfen, neue Lizenzschlüssel erstellen abrufen. Für das Erstellen und Überprüfen auf Gültigkeit ist der ‚KeyConverter' verantwortlich.

Zudem wird falls kein gültiger Lizenzschlüssel gefunden wurde automatisch eine Trial-Phase gestartet die fünfzehn Tage dauert. Danach ist die Applikation gesperrt, bis ein gültiger Lizenzschlüssel eingegeben wird.

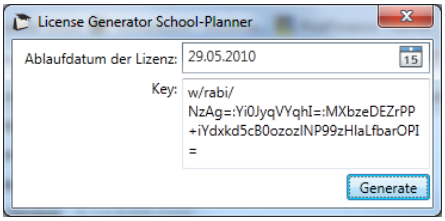
Für die Trial-Phase wird das Ablaufdatum der Trial-Phase in die Registry mit dem ‚RegistryHandler' hineingeschrieben. Mit einem Trial Schlüssel kann die volle Funktionalität des Programms verwendet werden bis auf das Drucken von Stundenplänen. Falls ein korrekter Lizenzschlüssel eingegeben wurde, wird dieser ebenfalls mittels dem ‚RegistryHandler' in der Registry gespeichert.

11.6.1 Lizenzschlüssel



Der Lizenzschlüssel besteht aus dem Ablaufdatum, einem zufällig generierten 8 Byte grossem Seed sowie einem Hash. Der Hash wird aus dem Ablaufdatum, dem Seed und einem fest einprogrammierten Passwort gebildet. Für das Erstellen des Hashes wird der SHA-256 Hashingalgorithmus verwendet.

11.6.2 Lizenzschlüssel Generierung



Damit einfach Lizenzschlüssel erstellt werden können gibt es einen Lizenzschlüssel Generator. Man muss dort lediglich das Ablaufdatum der Lizenz festlegen.

So können einfach Lizenzschlüssel für Kunden generiert und direkt per E-Mail zugesandt werden.

11.6.3 Sicherheit

Das Lizenzierungsverfahren ist bewusst relativ einfach gehalten. Es ist ebenfalls klar, dass man mit etwas Computerfachkenntnis das ganze Verfahren umgehen kann.

- So könnte man beispielsweise jedes Mal wenn die Trial-Phase abläuft in die Registry gehen und dort den Trial Eintrag löschen. Es würde dann nämlich von der Applikation her ein neuer Trial Eintrag gemacht werden, da angenommen wird, dass es sich um eine Frische Installation handelt.
- Ein etwas aufwändigerer Ansatz wäre es mit einem Debugger herauszufinden wie sich der Lizenzschlüssel zusammensetzt und das Passwort auszulesen. Mit diesen Informationen könnte man selber Lizenzschlüssel erstellen.
- Es könnte auch einfach der Codeteil ersetzt werden in welchem der Check gemacht wird, ob ein gültiger Lizenzschlüssel verwendet wird.

Alle diese Ansätze erfordern etwas Expertise. Doch diese werden wohl kaum bei einem durchschnittlichen Benutzer vorhanden sein. Ausserdem wird davon ausgegangen, dass das Programm ohne gezielte Schulung nicht verwendet werden kann. Aus diesem Grund wird die Gefahr dieser Sicherheitsmängel als gering eingeschätzt. Der Aufwand diese zu beheben, wenn man dem den erwarteten Mehranteil von zahlenden Kunden gegenüberstellt, als zu hoch bewertet.

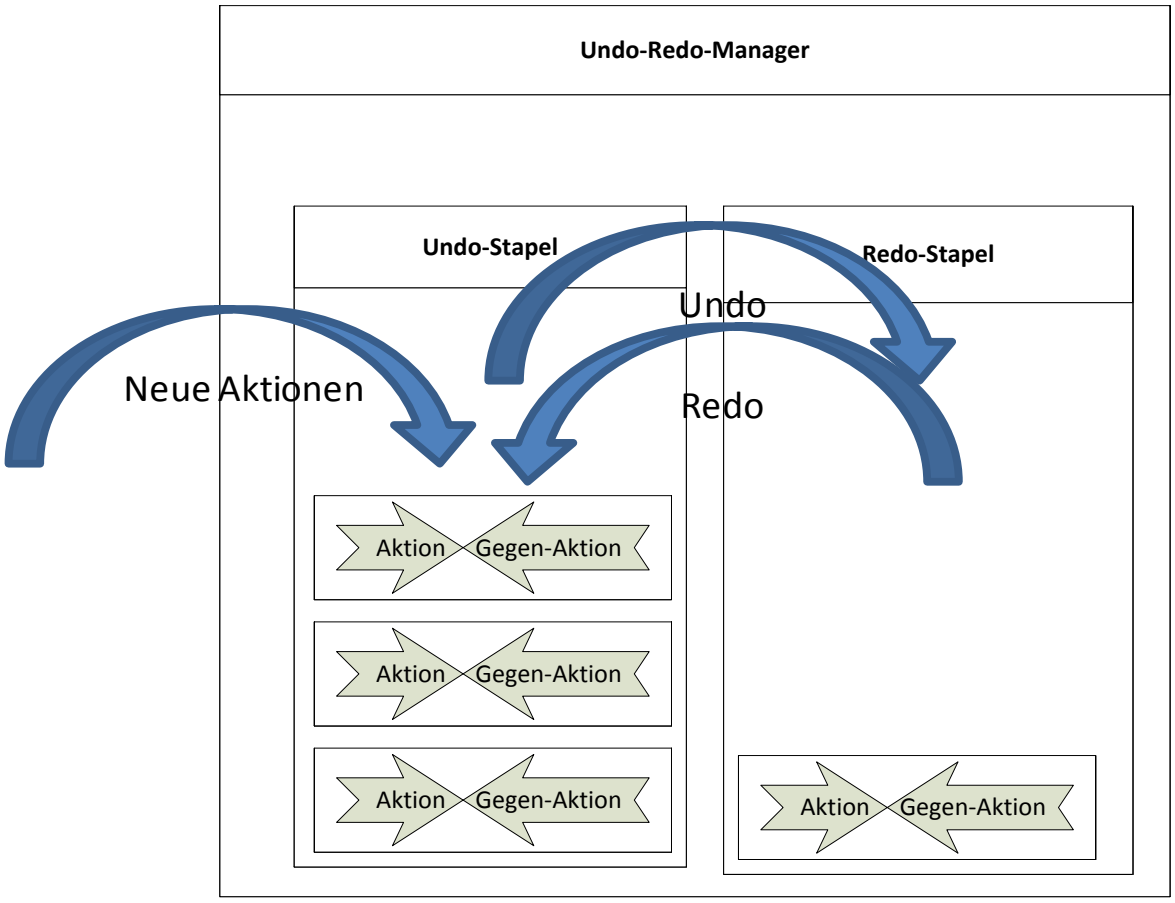
11.7 Undo & Redo Funktion

Für die Kalender ist eine Undo-Redo Funktionalität implementiert. Folgende Anforderungen müssen erfüllt sein.

- Jede Kalender-Ansicht hat einen eigenen Undo-Redo Stapel.
- Wenn eine Aktion auf einem Kalender stattfindet und diese Aktion auch andere Kalender betrifft, muss diese Aktion in den Undo-Redo Stapel dieser Kalender aufgenommen werden.
- Wenn eine Aktion stattfindet, welche eine Kalender-Ansicht nicht betrifft, so soll diese nicht im Undo-Redo Stapel aufgenommen werden.
- Ein Undo- und Redo-Aktion auf dem angezeigten Kalender funktionieren wie erwartet.
- Ein Undo- oder Redo-Aktion auf einem anderen Kalender, wird als normale Aktion erfasst.
- Eine Kalender-Ansicht kann mehrere Kalender anzeigen. Das Undo und Redo muss beide Kalender beachten.

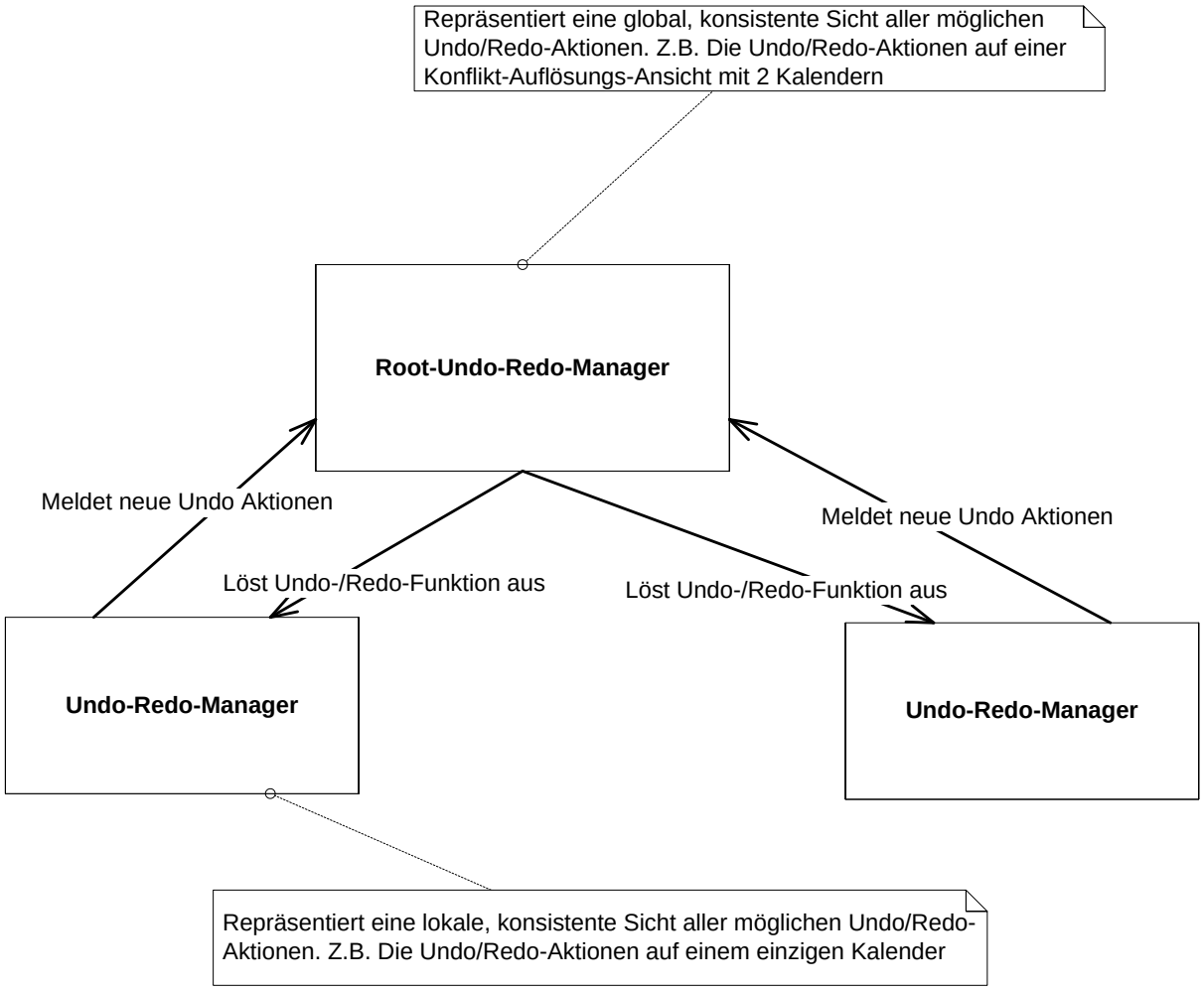
11.7.1 Ein Undo-Redo-Manger pro Kalender

Zuerst wird für jede Aktion die Gegenaktion berechnet. Z.B. für eine Einfüge-Aktion wird die rückgängig machende Lösch-Aktion berechnet. Diese Aktion und die Gegenaktion werden als Paar einem Undo-Redo-Manager übergeben. Dieser stapelt alle Aktionen und Gegenaktionen auf. Sobald etwas rückgängig gemacht wird, wird das letzte Aktion-Gegenaktion-Paar genommen und die Gegenaktion ausgeführt. Anschliessend wird dieses Paar auf den Redo-Stapel gelegt.



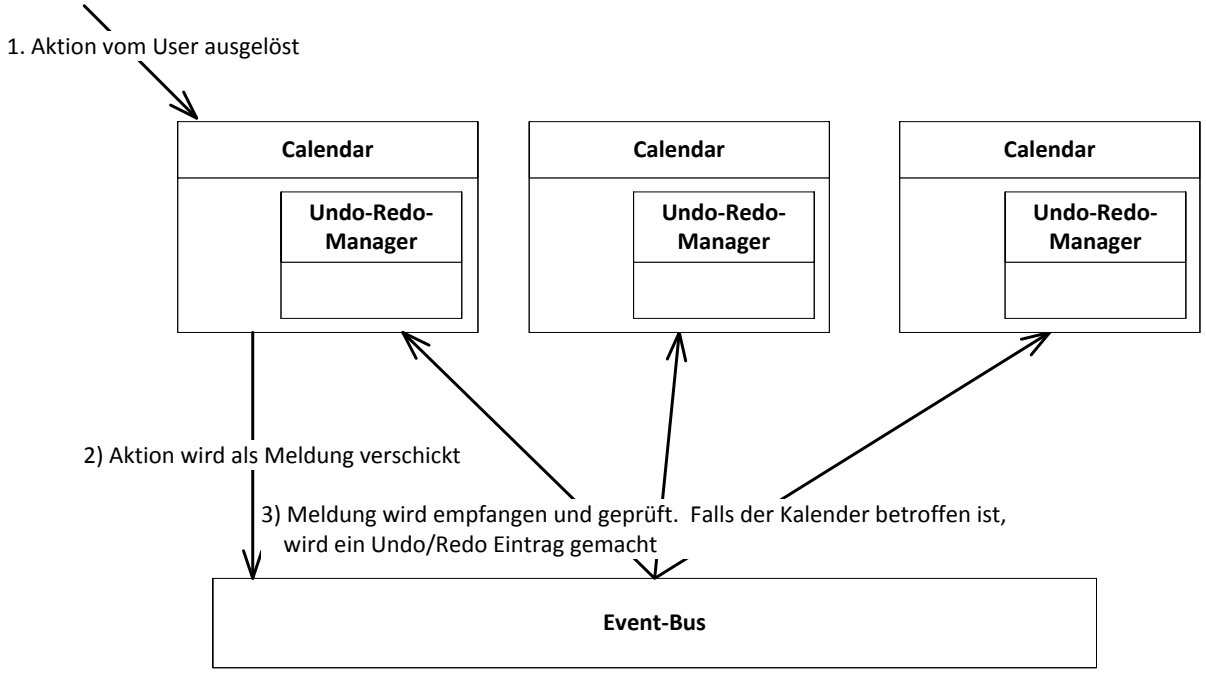
11.7.2 Hierarchische Undo-Redo-Manager

Das System besteht aus hierarchischen Undo-Redo-Managern. Der übergreifende Undo-Redo-Manager überwacht die darunterliegenden Undo-Redo-Manager. Sobald in einem darunterliegenden Manager ein neuer Undo-Schritt erfasst wird, wird dieser ebenfalls im darüber liegenden Manager erfasst. Dies löst das Problem für Ansichten, welche mehrere Kalender enthalten. So gibt es eine darüber liegenden Undo-Redo-Manager, welcher die Aktionen aller Undo-Redo-Manager überwacht.



11.7.3 Synchronisierung der Undo/Redo über die Kalender-Grenze

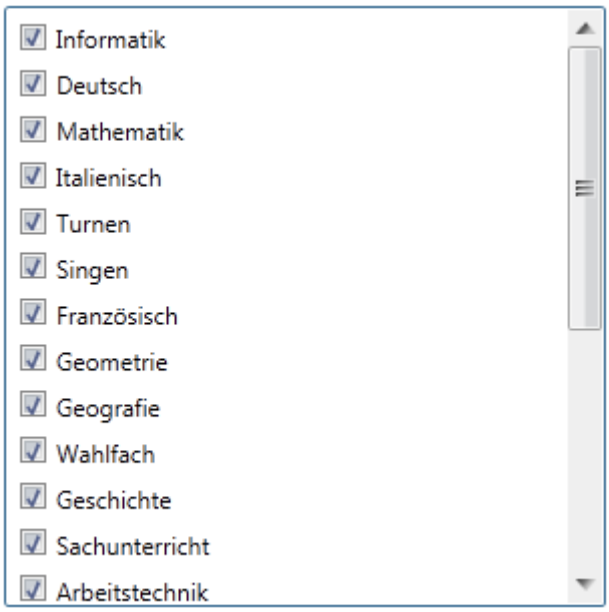
Jeder Kalender hat seinen eigenen Undo/Redo-Stapel. Aber viele Aktionen betreffen mehrere Kalender. Wie wird also sichergestellt, dass eine Aktion welche mehrere Kalender betrifft bei allen Kalendern in ihrem Undo/Redo-Stapel auftauchen? Nun dazu wird das bestehende Bus-System verwendet, siehe 11.3 Event-Bus. Jede Aktion wird als Meldung über das Bus-System verschickt. Jede Kalender-Instanz hört auf diese Meldungen. Ankommende Meldungen werden untersucht und auf Relevanz für den Kalender überprüft. Falls die Meldung relevant ist, wird die Aktion und Gegenaktion ebenfalls auf dem Stapel des Undo-Redo-Managers dieses Kalenders abgelegt.



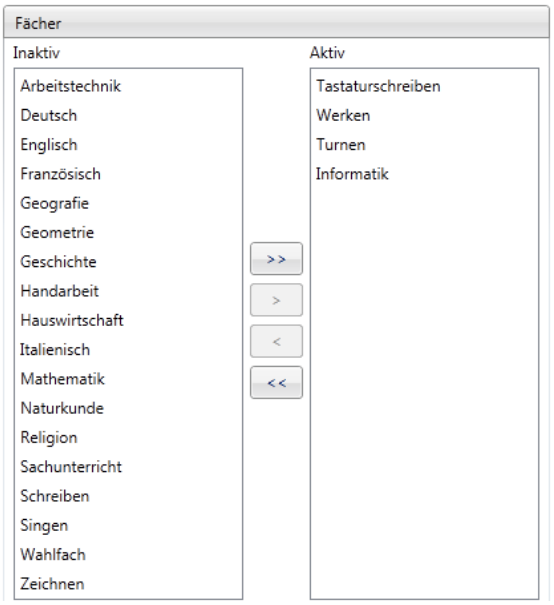
12 Userinterface

12.1 Multi Selection Control

Es gibt verschiedene Stammdaten bei denen es möglich ist, mehrere Einträge gelichzeitig auszuwählen. Um solche mehrfach Selektionen zu ermöglichen werden in den meisten Applikationen einer der zwei folgenden Ansätze verwendet.



Massenauswahl mit Checkboxen



Massenauswahl mit Schieben

Um für die School-Planner Applikation eine möglichst optimale Lösung zu finden wurde zu diesen zwei Möglichkeiten eine kleine Userumfrage durchgeführt.

12.1.1 Fragen

Welches Usercontrol würden Sie für die Massenauswahl von Elementen bevorzugen?

Warum bevorzugen Sie dieses Usercontrol und worin sehen sie dessen Vorteile gegenüber dem anderen?

Kennen sie durch bereits verwendete Programme noch andere Ansätze für die Massenauswahl von Elementen?

12.1.2 Probanden

Die Umfrage wurde mit 10 Teilnehmern durchgeführt. Die Teilnehmer sind unterschiedlich Erfahren mit der Benutzung von Computerprogrammen. Die Teilnehmer verteilen sich folgendermassen:

- Experten (3 Teilnehmer)
- Anwender (4 Teilnehmer)
- Gelegenheitsbenutzer (3 Teilnehmer)

12.1.3 Auswertung

Die Auswertung ergab, dass 9 der 10 Probanden die Massenauswahl mit Schieben bevorzugten. Der Hauptgrund war, dass man sofort auf einen Blick sieht, was eigentlich momentan ausgewählt ist. An der Massenauswahl mit Checkboxen wurde die fehlende Übersicht kritisiert, obwohl deren verhalten intuitiver ist als beim schieben.

Bei der Massenauswahl mit Checkboxen wurde zudem kritisiert, dass es mindestens noch einen Button benötigt wird um alles auszuwählen, respektive abzuwählen. Diese Funktionalität kann einfach implementiert werden. Das Problem der geringeren Übersichtlichkeit wäre aber nicht gelöst. Dieses Problem könnt man mit zusätzlichen Boxen lösen, worin alle gewählten, respektive abgewählten Items angezeigt werden.

Aufgrund dieser Umfrage wird nun Primär das Usercontrol mit Schieben zur Massenauswahl verwendet.

Interessant ist zu erwähnen, dass alle Probanden beide Formen der Massenselektion bereits in mindestens einem Programm gesehen haben und daher mit den Konzepten vertraut waren.

Ausserdem konnte niemand einen anderen Lösungsansatz nennen, den beobachtet haben.

13 Environment

13.1 Versions und Build-Management

Als Versions-Management-System wird Subversion benutzt. Die URL des Subversion -Server lautet: http://rrt.endofinternet.org/svn/

Zusätzlich wird TeamCity verwendet, um kontinuierliche Builds zu erstellen. TeamCity benachrichtigt die Projekt-Teilnehmer, wenn das Projekt momentan nicht kompiliert oder Unit-Tests fehlschlagen. Die URL des TeamCity-Servers ist: http://rrt.endofinternet.org:8888/teamcity/

Sowohl das Subversion-Repositiory als auch der TeamCity-Server laufen auf dem privaten Server bei Raphael Ritter und Roman Stoffel.

13.1.1 SVN Policy

Es wird grundsätzlich nur Code eingecheckt der auf dem Entwickler-PC problemlos kompiliert und alle Unittests erfolgreich durchlaufen hat.

Alle Dokumente die für das Projekt erstellt werden, sind im SVN abgelegt.

Jeder Projektteilnehmer darf jedes File mutieren und einchecken, wobei bei kritischen Änderungen die restlichen Projektteilnehmer informiert werden müssen.

13.2 Issue Tracking

Die ganze Projektplanung Issue sowie Change-Management wird in unserem Issue-Tracker abgehandelt, der uns für die Bachelor Arbeit gratis von der Firma Countersoft (http://www.countersoft.com/home.aspx) zur Verfügung gestellt wurde. Der Issuetracker ist erreichbar unter der URL http://rrt.endofinternet.org:8080/Gemini/. Im Issue-Tracker wird auch die gesamte Zeiterfassung durchgeführt. Damit kann Issue nachvollzogen werden, wo wie viel Zeit verbraucht wurde.

14 Testing

14.1 Performance Profiling

Im Verlauf der Implementierung der Applikation gab es immer wieder Funktionen welche zu viel Zeit in Anspruch nahmen. Diese stellen wurden dann jeweils folgendermassen untersucht:

- Suche der Zeit-Intensiven Stellen der Funktion
- Optimierung der gefundenen Performanceprobleme

Diese beiden Schritte wurden solange durchgeführt, bis eine zufriedenstellende Performance erreicht wurde.

Für das finden von möglichen Problemstellen wurde ein Profiler und zusätzlich absolute Zeitmessung im Code benutzt.

14.2 Memory Profiling

Beim Hinzufügen von neuer Funktionalität besteht immer die Gefahr, dass Memory Leaks eingebaut werden. Solche Memory Leaks entstehen dadurch, dass Objektreferenzen gehalten werden, obwohl sie nicht mehr benötigt werden. Diese Objektreferenzen müssten aber eigentlich abgehängt werden um dem Garbage Collector das Freigeben von Memory zu ermöglichen. Typischerweise entstehen solche Memory Leaks beim Event-Handling. Man meldet dort ein Objekt an einem Event an, aber nie mehr ab.

Um dem entgegen zu wirken wurde die Applikation mit einem Memory-Profiler auf solche Leaks untersucht. Dabei wurde geschaut, ob gewisse Objekte aufgeräumt werden. Falls dies nicht der Fall war, wurde untersucht von wo sie noch referenziert werden. Sobald man das Problem behoben hatte, wurde mit dem Profiler untersucht, ob sie nun frei gegeben werden. Dieser Vorgang wurde jeweils solange wiederholt, bis die Objekte wirklich freigegeben wurden.

Das Ziel ist des ganzen Memory Profilings war es die Memory Leaks auf ein akzeptables Niveau zu senken. Die Applikation sollte schlussendlich für mehrere Stunden problemlos verwendet werden können ohne den gesamten Arbeitsspeicher aufzubrauchen.

14.3 Unit-Tests

Für die komplexe Geschäfts-Logik und andere Businesskomponenten werden Unit-Tests geschrieben. Diese Tests prüfen einzelne Komponenten auf ihre Funktionalität. Mit diesen Tests sollen vor allem Regressionen der Software verhindert werden. Ausserdem wird jeder gefundene Bug mit einem Testreproduziert um sicher zu stellen, dass derselbe Bug nicht mehr auftreten kann.

14.3.1 Testabdeckung

Project	Acceptable	Unvisited SeqPts	Coverage
Unknown	95.0 %	3629	72.9 %

Modules	Acceptable	Unvisited SeqPts	Coverage
Application.Calendar.dll	95.0 %	470	55.0 %
Application.CalendarWorkflow.dll	95.0 %	220	40.5 %
Application.DataStructures.dll	95.0 %	247	47.7 %
Application.Utils.dll	95.0 %	12	57.1 %
BusinessLogic.AlgorithmContext.dll	95.0 %	9	93.7 %
BusinessLogic.AlgorithmInterface.dll	95.0 %	293	84.0 %
BusinessLogic.Basedata.dll	95.0 %	153	88.4 %
BusinessLogic.Calendar.dll	95.0 %	317	83.8 %
BusinessLogic.ConflictDetection.dll	95.0 %	157	79.2 %
BusinessLogic.DomainModel.dll	95.0 %	158	79.7 %
BusinessLogic.DomainModel.Tests.dll	95.0 %	28	94.8 %
BusinessLogic.GeneticAlgorithm.dll	95.0 %	30	90.6 %
BusinessLogic.GraphAlgorithm.dll	95.0 %	106	70.9 %
BusinessLogic.HeuristicAlgorithm.dll	95.0 %	577	57.1 %
Core.AsyncCoordination.dll	95.0 %	95	70.3 %
Core.DB.dll	95.0 %	163	74.7 %
Core.IoC.dll	95.0 %	268	61.1 %
Core.Licensing.dll	95.0 %	59	55.6 %
Printing.Core.dll	95.0 %	267	21.7 %

Die Testabdeckung auf dem Applikation Layer ist nicht so hoch, da dort auch einiges auf Userinterface ebene vorgenommen wird, was schwierig zu testen ist. Ausserdem werden dort sowieso primär lediglich die ‚ViewModels' auf ihre korrekte Funktionsweise getestet.

Beim ‚Printing.Core' wird auf exzessives Testen mit Unittests verzichtet, da man den visuellen Excel Output sofort sieht und einfach überprüfen kann. Deswegen wurde dieser Teil nur rudimentär getestet, nicht zuletzt auch aus Zeitgründen.

14.4 User Tests

Es wurden drei grössere Usertests durchgeführt, einer mit dem Stundenplanverantwortlichen der Schule Chur Rheinau, mit einem Student und einer Lehrerin. Bei den Usertests wurde jeweils mit einer leeren Datenbank gestartet. Anschliessend wurde das ganze Erfassungsprozedere durchlaufen und schlussendlich fertige Stundenpläne erstellt. Diese Usertets wurden jeweils auf Video aufgenommen für eine anschliessende Analyse.

Bei den Bemerkungen zu dem jeweiligen Usertest werden lediglich die Wichtigsten Punkte aufgezählt die bei den jeweiligen Usertests gefunden wurden.

14.4.1 User Test Schule Chur

- Stundenpläne für Räume und Lehrer werden benötigt.
- Es ist ein gutes Manual für den Beschrieb der Stammdaten nötig.
- Stammdaten müssen gelöscht werden können.
- Beim Einfügen von neuen Daten sollte es einen Autofokus geben auf den neuen Eintrag.
- Es sollten bei Komboboxen sinnvolle default Auswahlen getroffen werden anhand vorhergehender Eingaben.
- Stundenpläne sollten ausgedruckt werden können, sowohl für Lehrer, Schüler als auch Räume.
- Es sollten Ressourcen auch nachträglich noch gewechselt werden können.

14.4.2 User Test mit Informatikstudent

- Beim Einstellen von bevorzugten Stunden sollte es die Möglichkeit geben denselben Wert für eine ganze Linie.

- Die Verschiebeansicht wurde nicht komplett verstanden und muss im Manual besser dokumentiert werden.
- Die Hilfe sollte Kontextsensitiv angezeigt werden.
- Es wurden mehrere Abstürze gefunden anhand von unerwarteten Benutzereingaben.
- Zusätzlich wurden noch einige logische Bugs in der Applikation gefunden.
- Ressourcenwechsel hat nicht richitg funktioniert.
- Multiselektion in Tabellen möglich, obwohl sinnlos.

14.4.3 User Test mit Lehrerin

- Es wurden verschiedene Fokusprobleme in den Stammdatenansichten entdeckt.
- Ausserdem wurden zwei Abstürze gefunden, die durch unerwartete Useraktionen entstanden sind.
- Der Dialog der für die Erfassung der Zeiten benötigt wird, hat sich für einen Benutzer als komplett unbrauchbar herausgestellt. Durch den Input der Testperson wird dieser nun mit Textboxen die auf Korrektheit validiert werden ersetzt.
- Im Kalender erscheinen die Konfliktsymbole nicht mehr korrekt
- Das öffnen von Lehrer oder Raumansicht bevor die jeweilige Ressource gespeichert wurde führte zu Abstürzen.

14.5 System und Usability-Tests

Es wird eine Testdatenbank mit echten Daten der Schule Thusis erstellt und mit dieser Systemtests und Usabilitytests durchgeführt, wobei auf eine genaue Dokumentation dieser Tests verzichtet wird. Ausserdem wird die Software dritten gegeben um sie auf Usability zu testen.

14.6 Fortlaufende Builds

Bei jedem Check-In ins Subversion wird TeamCity automatisch das Projekt kompilieren und alle Unit-Tests durchführen. Dieser Build ist unabhängig von der Entwicklungs-Umgebung und wird auf einem „sauberen“ System durchgeführt. Damit ist sichergestellt, dass keine Entwicklungs-Umgebung spezifischen Eigenheiten sich ins Projekt einschleichen. Ein Fehlschlagen eines Builds oder eines Tests löst eine E-Mail-Nachricht an alle Projekt-Mitglieder aus.

14.7 Bugs- und Issue-Tracking

Bugs und Probleme der Software sind im Bugtracker zu erfassen. Der Bugtracker ist hier erreichbar:

http://rrt.endofinternet.org:8080/gemini/

Bugs, die man selbst entdeckt und sofort behebt, müssen nicht erfasst werden. Bugs die man später behebt oder von einem anderen Team-Mitglied behoben werden müssen, müssen im Bugtracker erfasst werden. Alle vom Kunden entdeckten Bugs sind zwingend im Bugtracker zu erfassen.

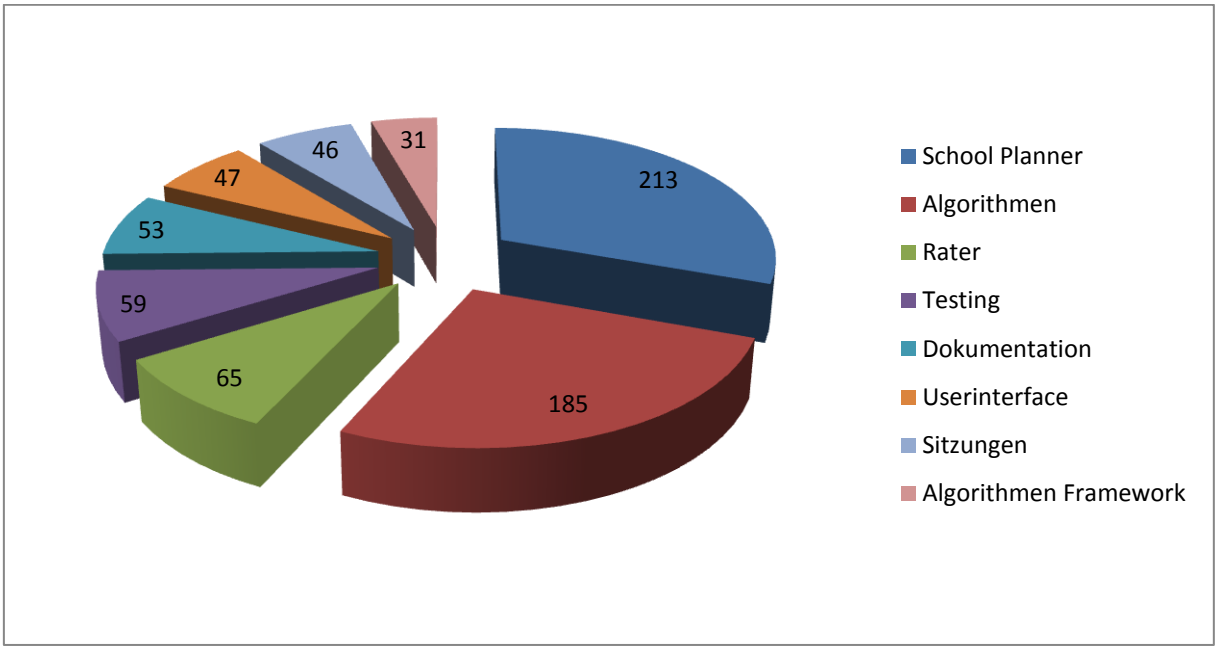
15 Anhang

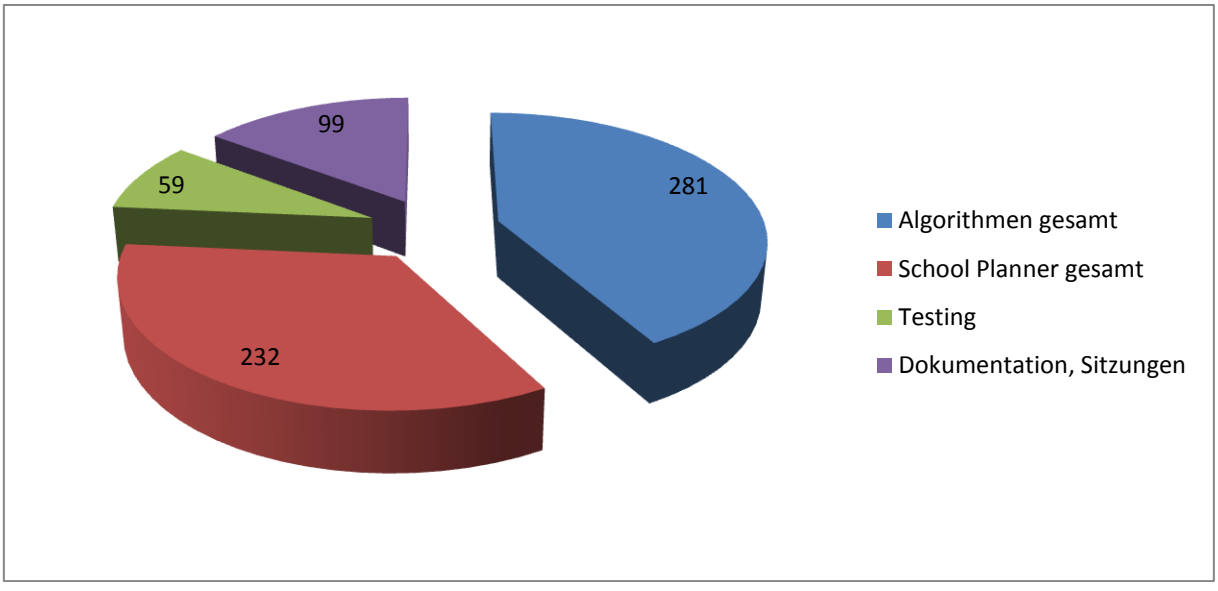
15.1 Arbeitsaufteilung

Die Arbeitsaufteilung wurde so vorgenommen:

- Raphael Ritter
  - Heuristischer Algorithmus
  - Graphen Algorithmus
  - Kombinierter Algorithmus
  - Algorithmen Framework
  - User Interface & Applikationsschicht
- Roman Stoffel
  - Genetischer Algorithmus
  - Analyse und Vergleich der Algorithmen
  - Rating Framework
  - Konflikterkennung
  - Business- & Datenbankschicht

15.2 Stundenverteilung





15.3 System-Anforderungen

Prozessor 2 GHz,

1 GB Arbeitsspeicher,
Windows 7 mit .NET 4.0 Framework
Office 2000k oder neuer für Excel-Exporte

Grundsätzlich läuft die Applikation auch jedem Windows-System, welches .NET 4.0 installiert. Doch wurde die Applikation nur auf Windows 7 getestet.

15.4 Risiken

Scheitern des Projektes	4		
Massiver Mehraufwand (< 150 Stunden)	2,8		6
Erheblicher Mehraufwand (< 100 Stunden)		3,5,7	
Mehraufwand (<50 Stunden)		1	
	Unwahrscheinlich	Möglich	Erwartet

Risiko	**Prevention**
1. Der Server fällt aus und die Projekt-Daten sind somit verloren	- Jedes Teammitglied hat eine lokale Kopie des aktuellen Entwicklungs-Standes - Der Server wird periodisch auf ein zweit-System gesichert
2. Das existierende Framework ist ungeeignet für die neuen Algorithmen.	
3. Die Analyse-Berechnungen und Messungen haben zu wenig Aussage-Kraft, um die Algorithmen zu vergleichen und verbessern	- Es werden 10 oder mehr verschieden Ansätze implementiert
4. Die Dokumentation wird als nicht befriedigend bewertet.	- Es wird von Anfang an dokumentiert - Reviews werden Durchgeführt

5. Algorithmen sind zu komplex / werden nicht verstanden	- Es wurden Algorithmen gewählt, bei denen ‚improvisieren' möglich ist
6. Algorithmen sind viel zeitaufwendiger zu implementieren als angenommen	
7. Die Testdaten haben zuviele Spezial-Fälle die nicht umgesetzt werden können	- Workarounds benutzen
8. Team-Mitglied fällt aus	-

15.4.1 Massnahmen bei Eintreffen:

Risiko	**Massnahme**
1)	- Migration auf anderen Server - Wiederherstellen der Daten von lokalen Kopien
2)	- Umschreiben / Ergänzen des Frameworks
3)	- Alternative Ansätze evaluieren - Manuelle Bewertung in Betracht ziehen
5)	- Reduzierung der Algorithmen auf ihre Grundidee - Alternative Ansätze verfolgen
6)	- Evalution ob Reduzieren des Umfangs der Arbeit
7)	- Ignorieren von Spezial-Fällen
8)	- Reduzieren des Umfangs der Arbeit

16 Referenzen

Alle Referenzen die nicht in Buchform vorgelegen haben, liegen der Arbeit bei.

Dokument über Stundenplangenerierung: , Diplomarbeit Martin Löhnertz‘

Dokument über Färbungsalgorithmen: ‚Diplomarbeit-Tobias.Baumann.pdf‘

Dokument über Lizensierungspatterns: ‚LicensingPatterns‘

Vorlesung „Wissensbasierte Systeme“ an der HSR

Buch POSA 1 (ISBN13: 978-0471958697)

Buch Apress WPF (ISBN13: 978-1-59059-955-6)