School-Planner

Raphael Ritter / Roman Stoffel

Semesterarbeit

Version 1.0

School-Planner 

Projektleiter	18.09.2009	Raphael Ritter / Roman Stoffel
Betreuer	18.09.2009	Josef Joller / HSR

Dokumentenverwaltung

Dokumenthistorie

Version	Status	Datum	Verantwortlicher	Änderungsgrund
1.0	In Bearbeitung	22.09.09	R. Ritter	Projektbeginn
1.1	Überarbeitet	22.09.09	Ritter & Stoffel	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	Layouting

Inhaltsverzeichnis

1 Einführung

1.1 Ziele

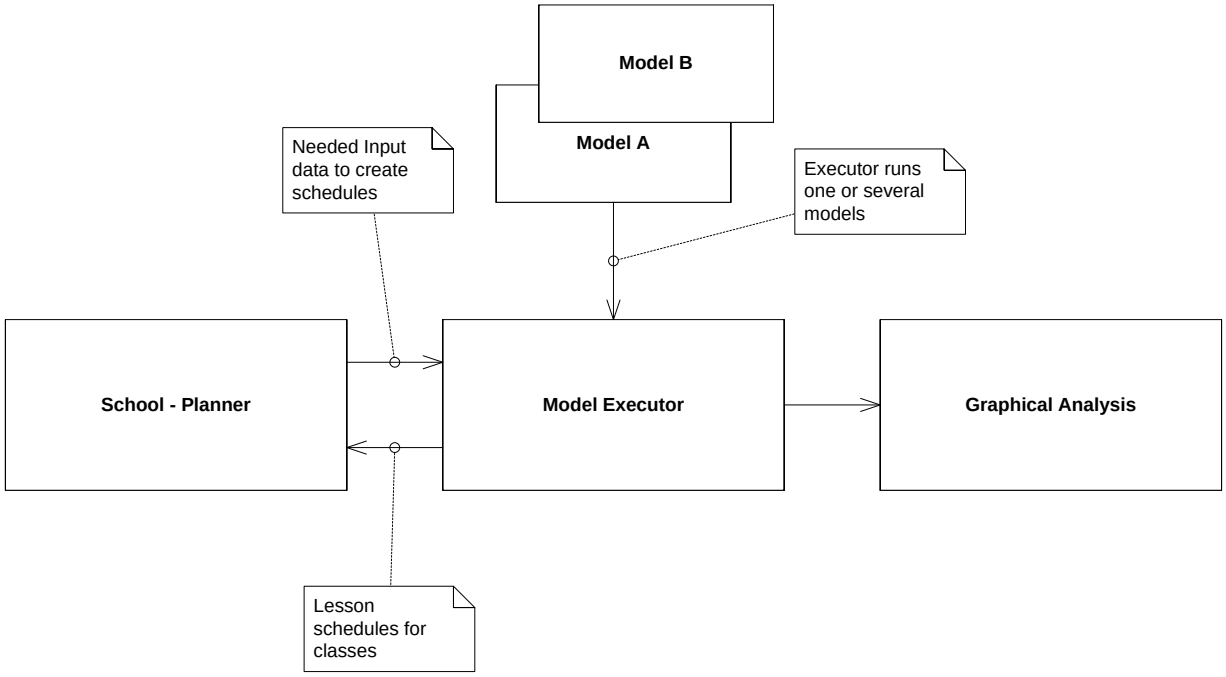
Ziel der Diplomarbeit war es ein Framework zu implementieren in welches verschiedene Algorithmen hineingehängt werden um Stundenpläne vollautomatisch zu generieren. Ein weiteres Ziel war es einen eigenen Algorithmus zu implementieren. Dieser dient dazu Anforderung an die Schnittstelle heraus zu finden. Der Algorithmus soll ebenfalls ein sinnvolles Ergebnis liefern. Zusätzlich wird ein Bewertungssystem gebaut um verschiedene Algorithmen zu bewerten um sie anschliessend miteinander vergleichen zu können.

Das Framework selbst enthält das gesamte Datenmodell, einen Basis Algorithmus, sowie die grafische Oberfläche für das Eingeben von Stammdaten. Zudem werden die generierten Stundenpläne angezeigt und persistiert.

Das Einfügen von weiteren Algorithmen in das Framework ist mit wenig Aufwand erreichbar. Wie man einen neuen Algorithmus Einfügt wird in Form eines „HowTo“- Manuals dokumentiert.

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 das Modell benötigt können hier erstellt werden.
- Der Output des Modells kann hier angesehen werden.

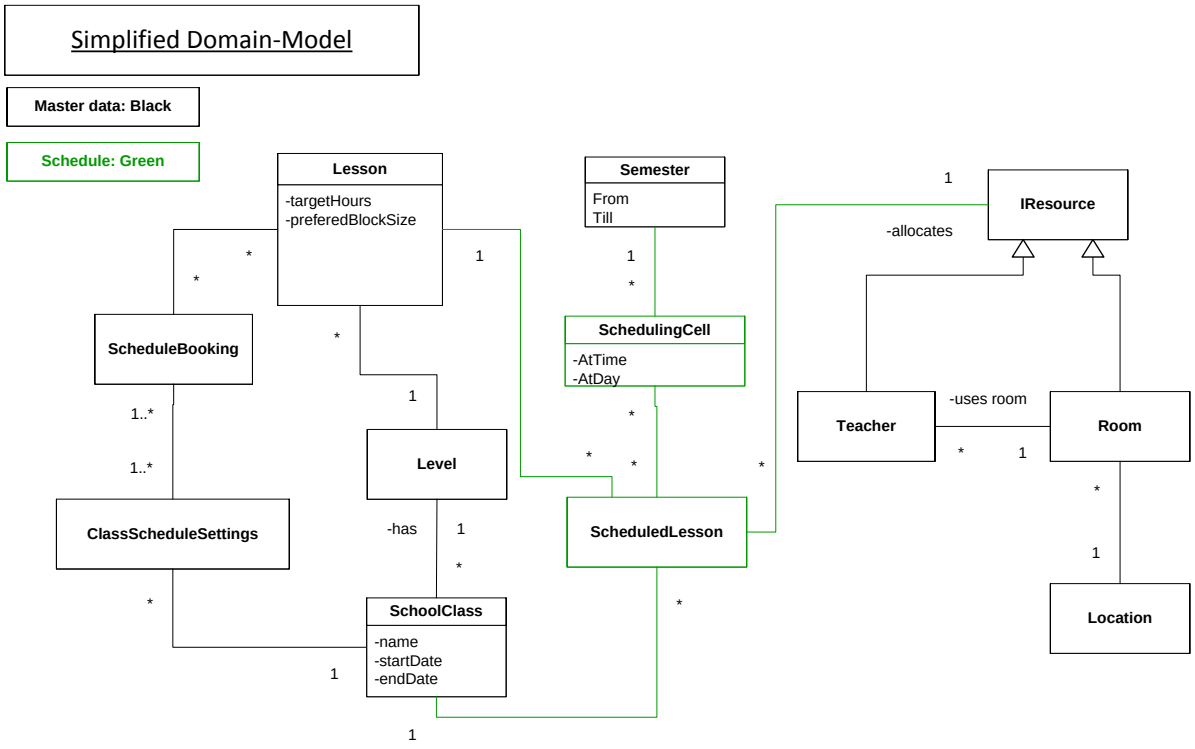
2.1.2 Model Executor

- Führt verschiedene Stundenplan-Erstellungs-Modelle aus.
- Ist in der Lage verschiedene Stundenplan-Erstellungs-Modelle miteinander zu kombinieren.

2.1.3 Grafische Analyse

- Es wird die Ergebnisse der automatischen Stundenplan Erstellung 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).

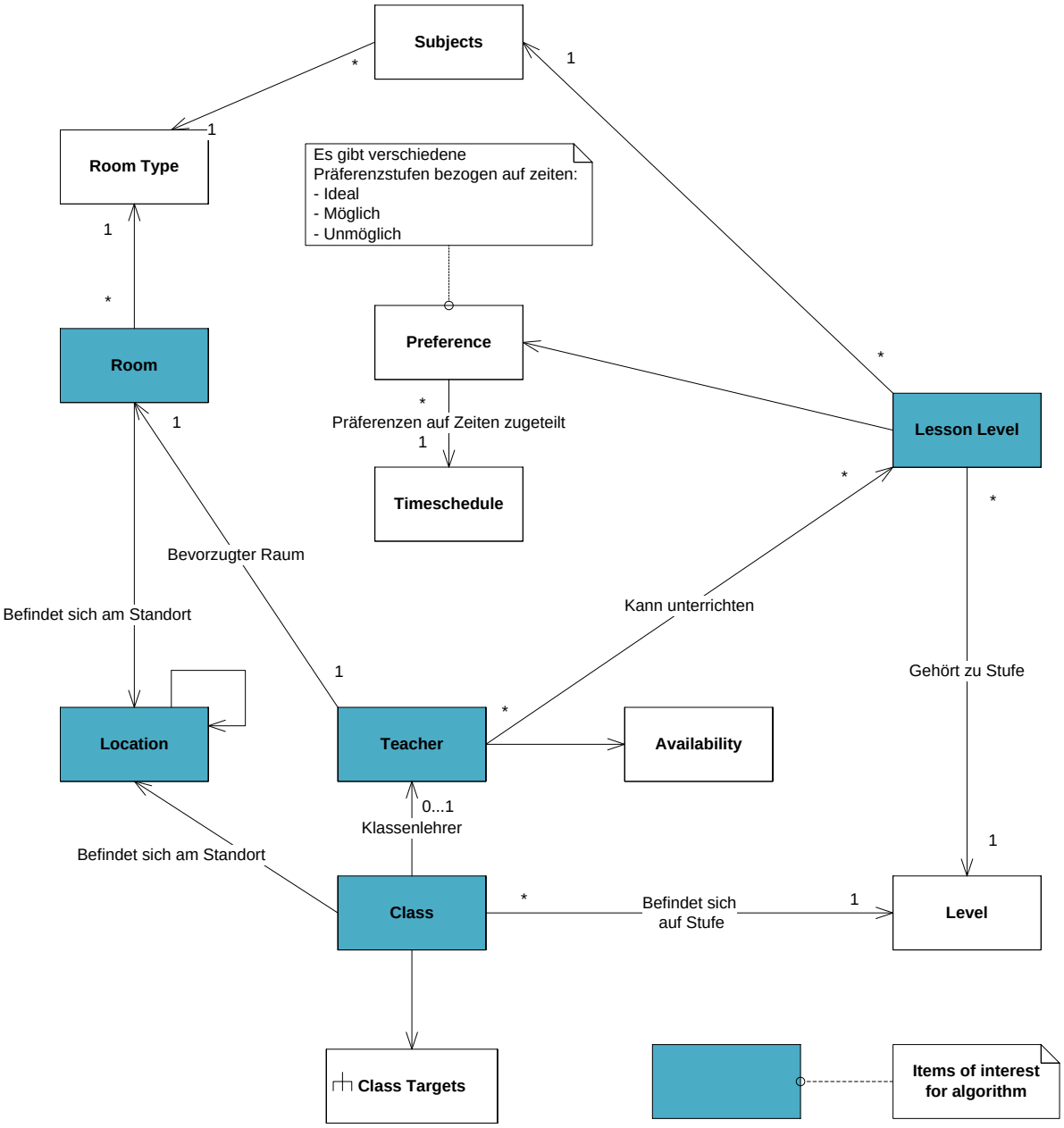
2.2 Vereinfachtes Domain Modell



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-Erstellungs-Modell 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 können diese Lehrer unterrichten?
- Wann ist der Lehrer überhaupt verfügbar um zu unterrichten?
- Wann sind die bevorzugten Unterrichtszeiten für einen Typ von Lektionen?
- Wann dürfen diese 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

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 auf einem bestimmten Niveau / Level unterrichten. Es genügt nicht einfach nur zu sagen Lehrer X unterrichtet Deutsch und Mathe. Denn 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 die man über einen Lehrer haben muss, ist sein bevorzugtes Zimmer. Gerade auf der Primar- oder Sekundarstufe ist es gewöhnlich, dass jeder Lehrer sein fixes Zimmer hat. Dieses verlässt er nur für spezielle Stunden wie Turnen, Werken oder Singen.

Lesson

Lektionen sind grundsätzlich immer von einem speziellen Typ (Subject). Ein Subject in unserem Fall wäre zum Beispiel Deutsch, Mathematik usw. Diese Information alleine genügt jedoch nicht. Man muss auch noch wissen für welche Stufe dieses Fach dann eigentlich vorgesehen ist. Das ist notwendig weil sich Anforderungen an die Lehrer dadurch ändern. Stufen wären beispielsweise Erste Primar, Zweite Primar… Ausserdem wird hier auch festgelegt wie viele Stunden in der Woche von diesem Subject für den ausgewählten Level gehalten werden und wie die Blockgrösse ist. Blockgrösse bedeutet wie viele Stunden von dieser Lektion nacheinander gehalten werden. Zum Beispiel wird Deutsch bevorzugt in Doppel-Stunden unterrichtet. Der Normalfall wird sein, dass man in vielen Fächern Doppelstunden haben möchten, falls dies möglich ist.

Preferences

An vielen Schulen ist es üblich, dass es für Lektionen bevorzugte Zeiten gibt wann diese abgehalten werden sollten. So ist es beispielsweise üblich anspruchsvolle Stunden wie Deutsch, Mathematik am Morgen abzuhalten. Denn am Morgen ist die Konzentration der Schüler eher 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. Dieses Ergebnis kann ebenfalls über die Preferences erreicht werden, in dem man für alle Stunden den Mittwochnachmittag als gesperrt markiert.

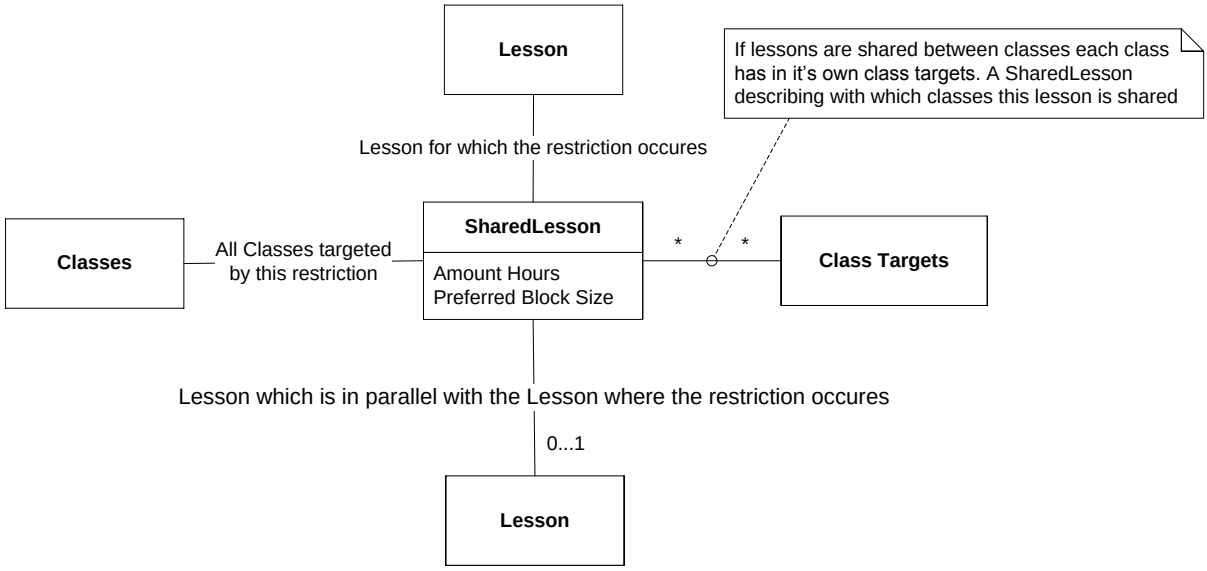
Class

Eine Klasse befindet sich jeweils auf einem bestimmten Level (z.B. Zweite Primarstufe). Anhand dieses Levels wird der dazugehörige Level der verschiedenen Fächer ausgewählt. Beispielsweise hat eine zweite Primar-Klasse vier Deutsch-Stunden. Anhand des gesetzten Levels ist nun ersichtlich dass diese vier Deutsch-Stunden für die zweite Primarstufe gehalten wird und nicht vier Deutschstunden für eine Sekundarklasse darstellen. Ausserdem hat eine Klasse noch einen Klassenlehrer. Dieser übernimmt normalerweise in der Primarstufe den grössten Teil aller Fächer.

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.

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 ein Booking parallel zu anderen Bookings läuft. Es wird generell aber nicht der Normalfall sein, das Stunden parallel ablaufen.

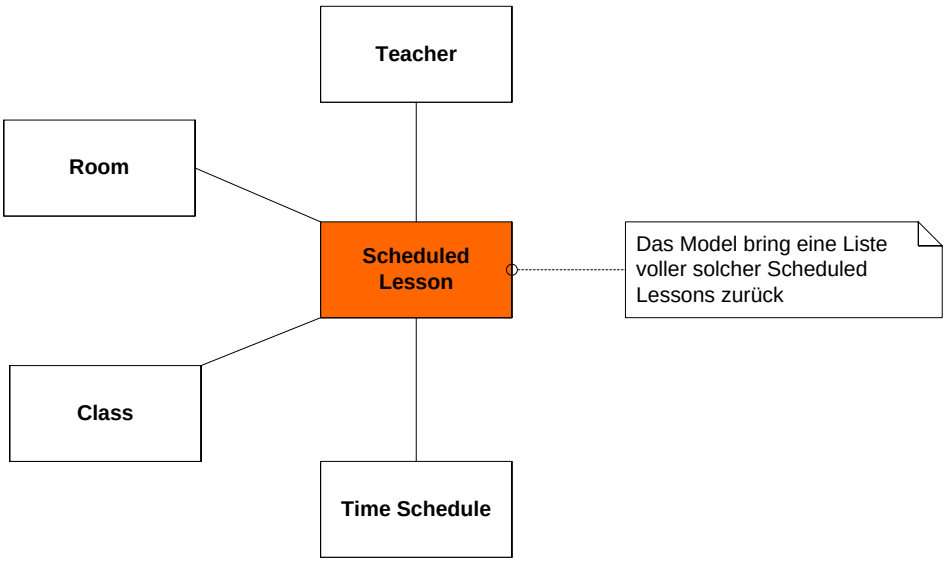
Ein weiterer Fall ist das verschiedene Klassen zusammen genommen werden, z.B. bei Fächern wie Turnen, Singen, Werken. Dass heisst es findet eigentlich nur eine Singstunde statt, aber mehrere Klassen besuchen diese gleichzeitig. Diese Situation wird aufgelöst in dem Bookings geteilt werden. So ist es möglich das Klasse A, B und C für den Singunterricht dasselbe Booking 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 diesem Booking zum selben Zeitpunkt statt.

Location

Viele Schulen haben ihre Unterrichtsgebäude auf verschiedene Standorte verteilt. Dies hat einen entscheidenden Einfluss, denn man will häufigen Standortwechsel der Klasse vermeiden. Das heisst man muss für die Klasse und für die Räume festlegen, wo sie sich befinden. Zusätzlich muss auch eine Standortabfolge angegeben werden. Das 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.

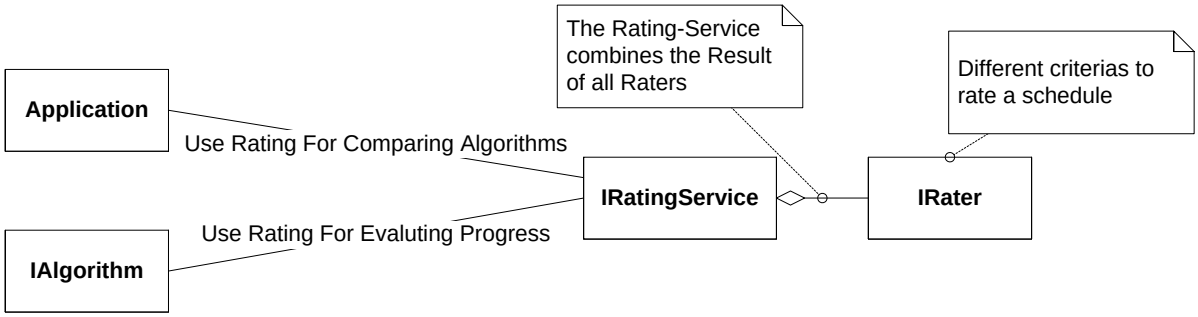
2.3.2 Modelle Output



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

- Klasse zu der die gebucht Lektion gehört.
- Raum in welchem die Lektion stattfindet.
- Lehrer der die Lektion abhält.
- Zeitfenster 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, muss man gewisse Bewertung-Kriterien erstellen können.

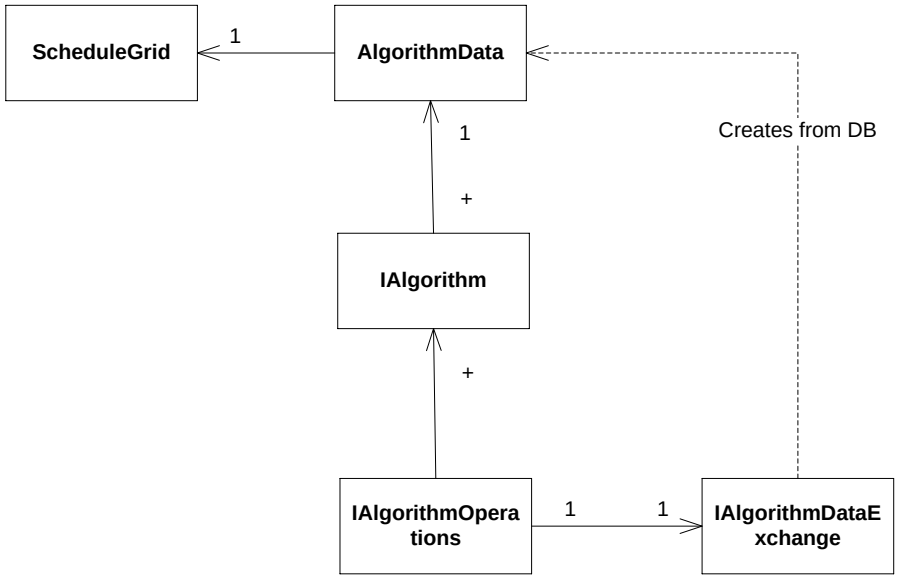
Es gibt extrem viele Aspekte die bewertet werden können. Daher gibt es ein kleines Bewertungs-Framework. Dieses erlaubt es beliebig viele Bewertungs-Kriterien einzubinden.

Es gibt zwei Bewertung-Arten.

- Tagwertung: 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. Diese wären alle Lehrer, Räume, Klassen etc. Es gibt zusätzliche, nützliche Funktionen um beispielsweise alle Räume für einen bestimmten Raum typ zu bekommen. Alle Daten in der AlgorithmData Klasse sind readonly. Diese Daten stammen von der School-Planner Applikation und stellen den Algorithmus-Input dar.

3.1.2 IAlgorithm

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

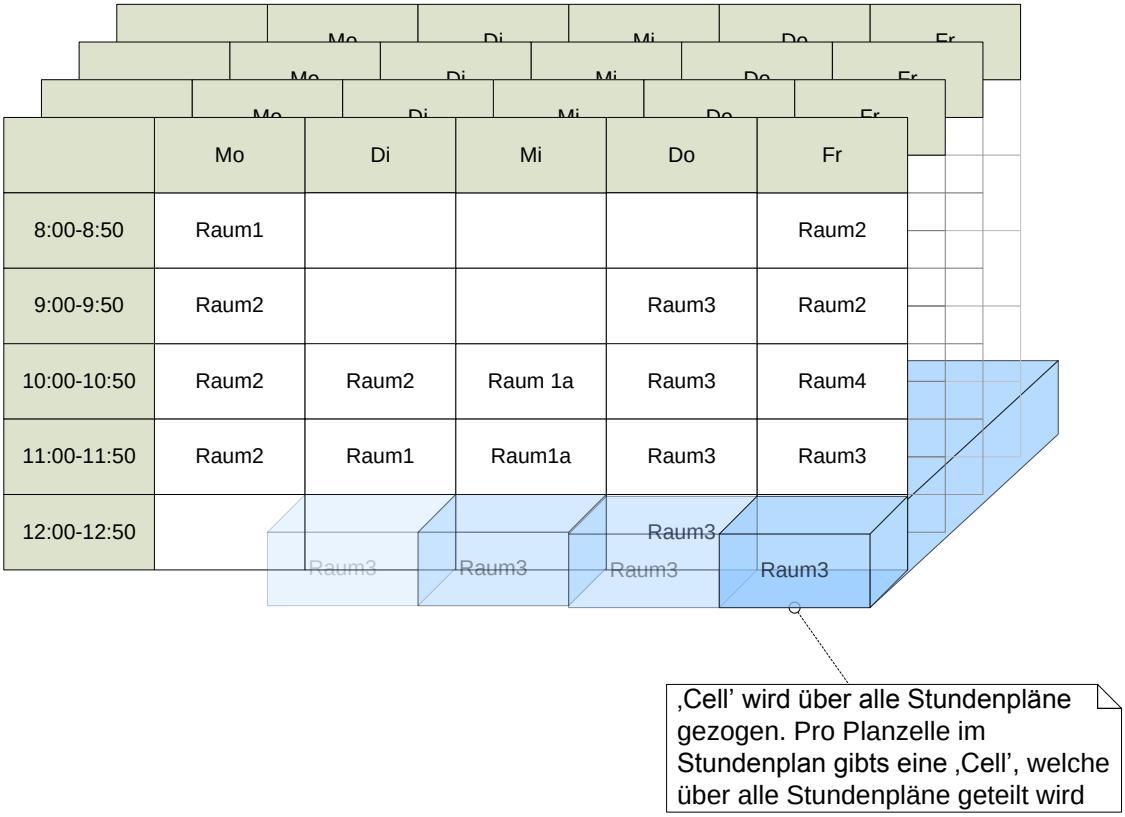
3.1.3 IAlgorithmOperations

Die Facade für die ganze Algorithmen Logik. Wird von der School-Planner Applikation verwendet um das vom User ausgelöste automatische Erstellen des Stundenplans auszulösen.

3.1.4 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.

3.1.5 Schedule Grid

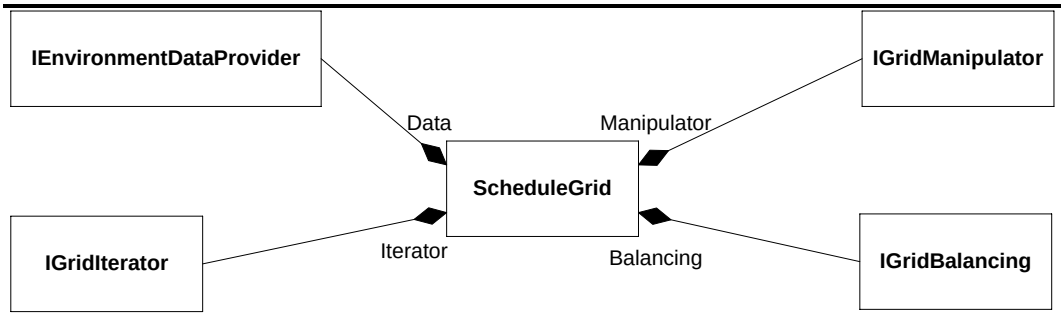


Dieses zweidimensionale Grid (Tage, Zeiten) beinhaltet Zellen. Auf diesen Zellen können dann Stunden für eine Klasse gebucht werden. Ausserdem bietet die Zelle auch die Möglichkeit abzufragen ob gewisse Ressourcen bereits verwendet werden und daher ein Konflikt entstehen würde oder nicht.

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

Der 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 das ermöglichen. Das Blackboard stellt diese Helfer jedem Client zu Verfügung.



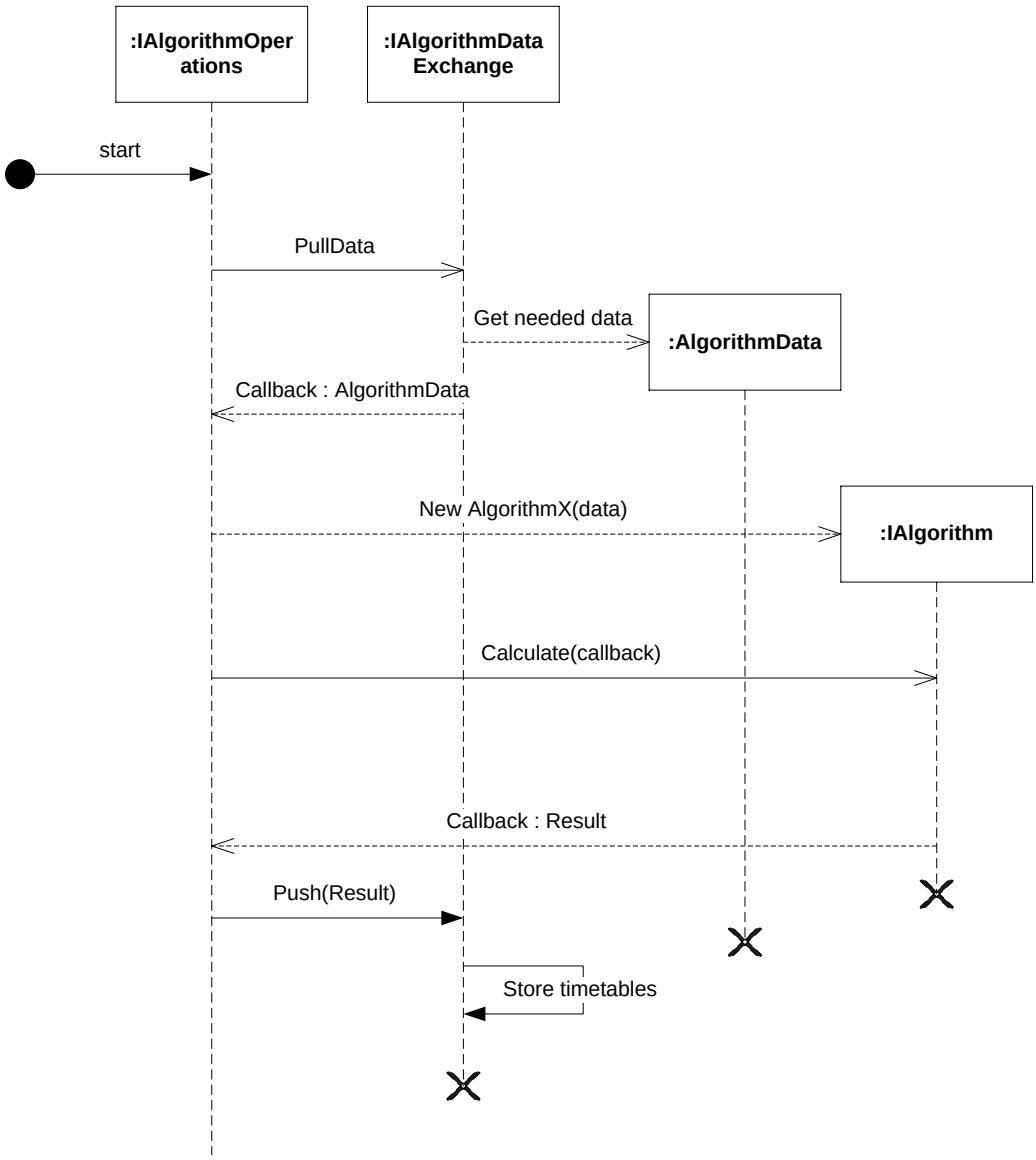
Der **IEnvironmentDataProvider** ist dazu da das abfragen von Stammdaten zu ermöglichen. Dieses Interface stellt die Funktionalität zur Verfügung, um mögliche Räume für eine bestimmte Lektion abzufragen etc.

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 direkt herauszufinden wie viele Stunden pro Tag eine Klasse hat und was für Lektionen das sind, respektive von wann bis wann die gehen.

3.2 Dynamic



Um einen Algorithmus verwenden zu können, muss der User zuerst alle benötigten Stammdaten erfassen. Ist dies getan wählt der User im Menu den Algorithmus aus, denn er benutzen will um die Stundenpläne generieren zu lassen. Sobald der User eine dort ein Algorithmus angewählt hat, ruft die Applikation die Facade des Algorithmus-Projekts 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 zurück zu schreiben in die Datenbank in Form von ausgefüllten Stundenplänen.

4 „Pseudo“ genetischer Algorithmus

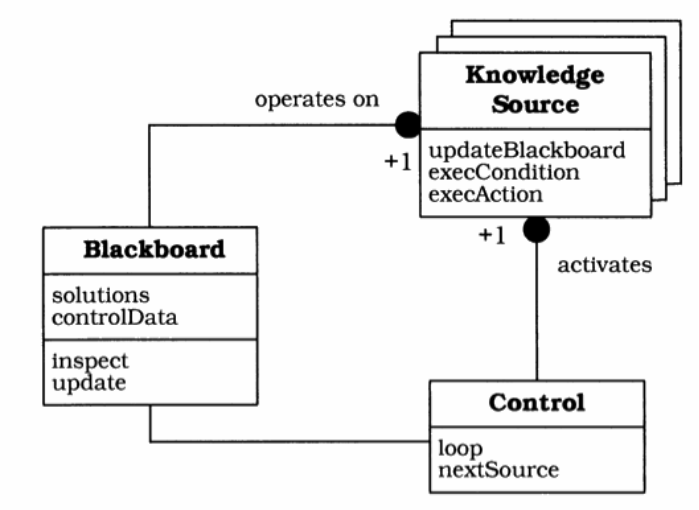
Bei der Implementation haben wir uns vom Blackboard Pattern (POSA 1) inspirieren lassen. Deswegen wird im folgenden Abschnitt dieses Pattern kurz erläutert. Bei der anschliessenden Erklärung unseres Algorithmus werden Referenzen zu diesem Pattern gemacht. Dadurch kann deren Funktion sehr gut beschrieben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

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



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

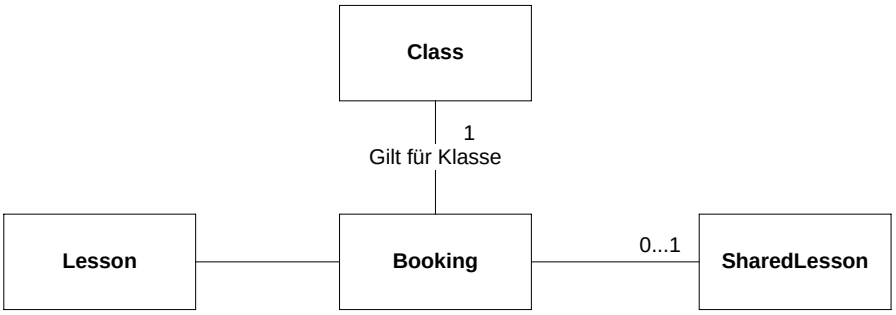
Eine Placing Strategie bekommt Bookings und die AlgorithmData. Sie versucht anschliessend so viele Booking wie ihr möglich sind sinnvoll zu platzieren und gibt alle unplatzierten Bookings zurück.

4.2.2 GeneticAlgorithm

Der Algorithmus ist nichts anderes als ein Kontroller der verschiedene Placing Strategien laufen lässt. Wann er welche Strategie anwirft kann verschieden Realisiert sein.

- Strikte Reihenfolge durch Prioritäten
- Strategie selbst gibt Input ob es sinnvoll wäre sie aufzurufen

4.3 Bookings

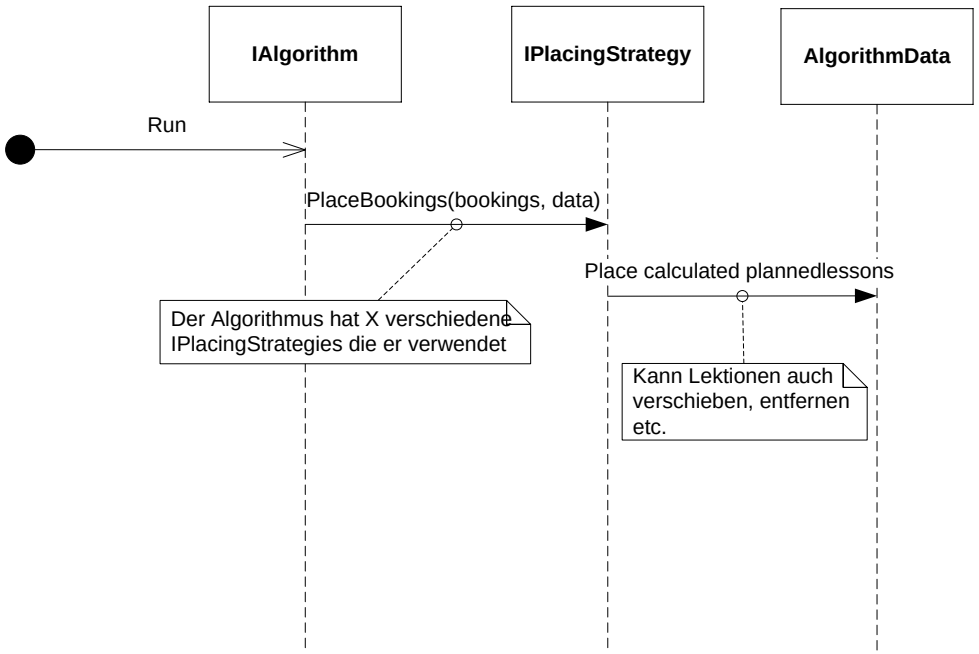


Jede Klasse hat eine Liste von Bookings die aus den Stammdaten berechnet werden können. Die Bookings bestehen einerseits aus der Anzahl Stunden für ein Fach. Diese sind bei den Lessons definiert sind und werden zusätzlich ergänzt durch die ClassTargets. Dort werden weitere Restriktionen festgehalten, 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 etc. Die berechneten Lösungen werden im ScheduleGrid abgespeichert. Dort werden dann auch Lösungen abgeändert oder verworfen.

4.4 Dynamics

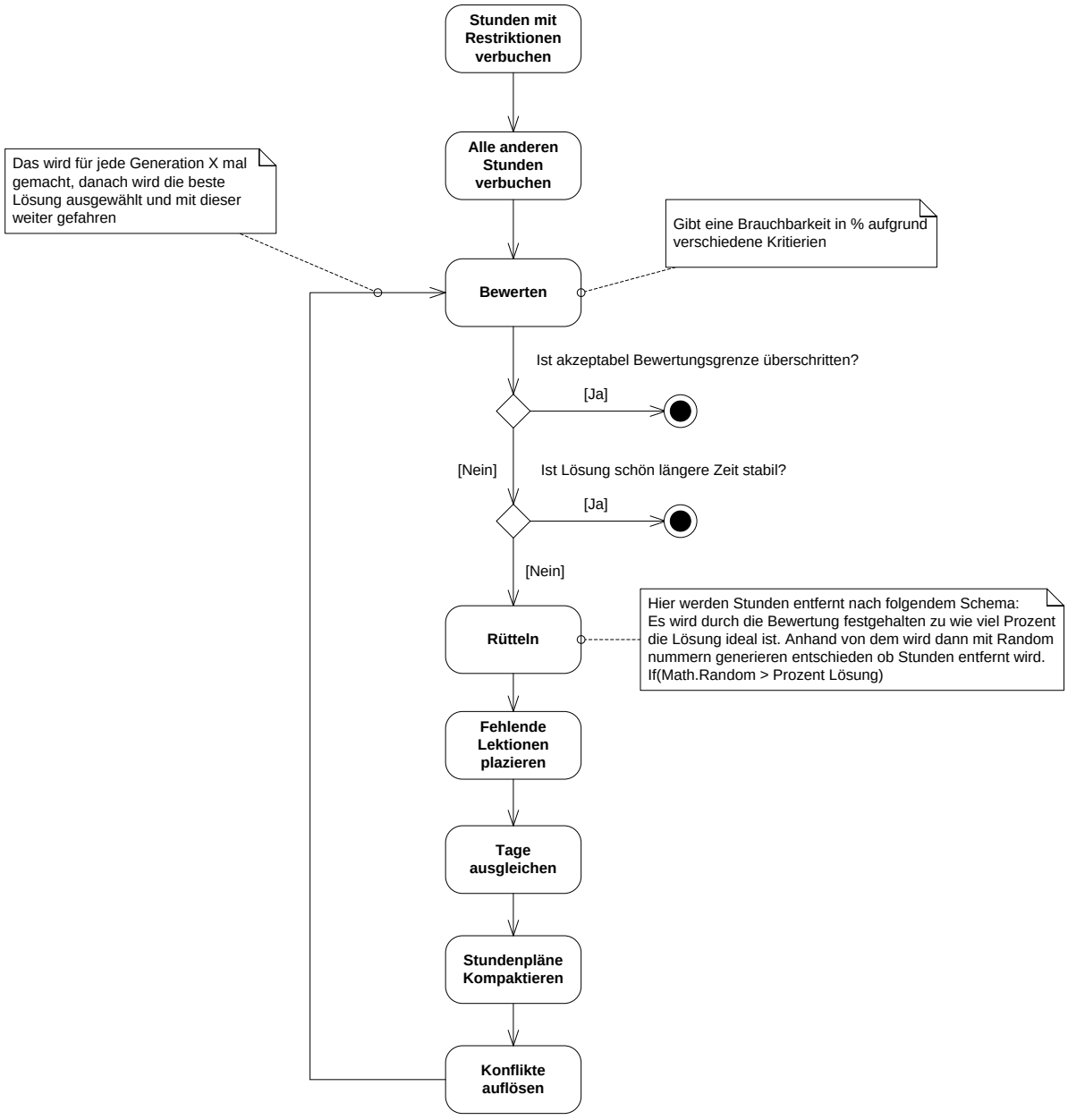


Dieses Diagramm zeigt die Umsetzung. Es entspricht ziemlich genau dem allgemeinen Ablauf der im vorhergehenden Kapitel beschrieben ist. Darum wird hier der Zusammenhang zwischen Blackbord-Pattern und unserer Implementation aufgezeigt.

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

Eine IPlacingStrategy entspricht einer Knowledge Source. Sie versucht gewisse Bookings anhand von definierten Kriterien auf dem Blackboard zu positionieren. Es ist grundsätzlich möglich dass eine Knowledge Source auftretende Konflikte ignoriert. Diese werden dann von einer anderen Knowledge Source aufgelöst. D.h. die Knowledge Sources können sehr flexibel aufgesetzt werden.

4.5 Allgemeiner Ablauf



Der Algorithmus platziert in einem aller ersten Schritt 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 vom gesamten restlichen Algorithmus ausgeschlossen, das heisst diese Stunden sind fix im Stundenplan platziert.

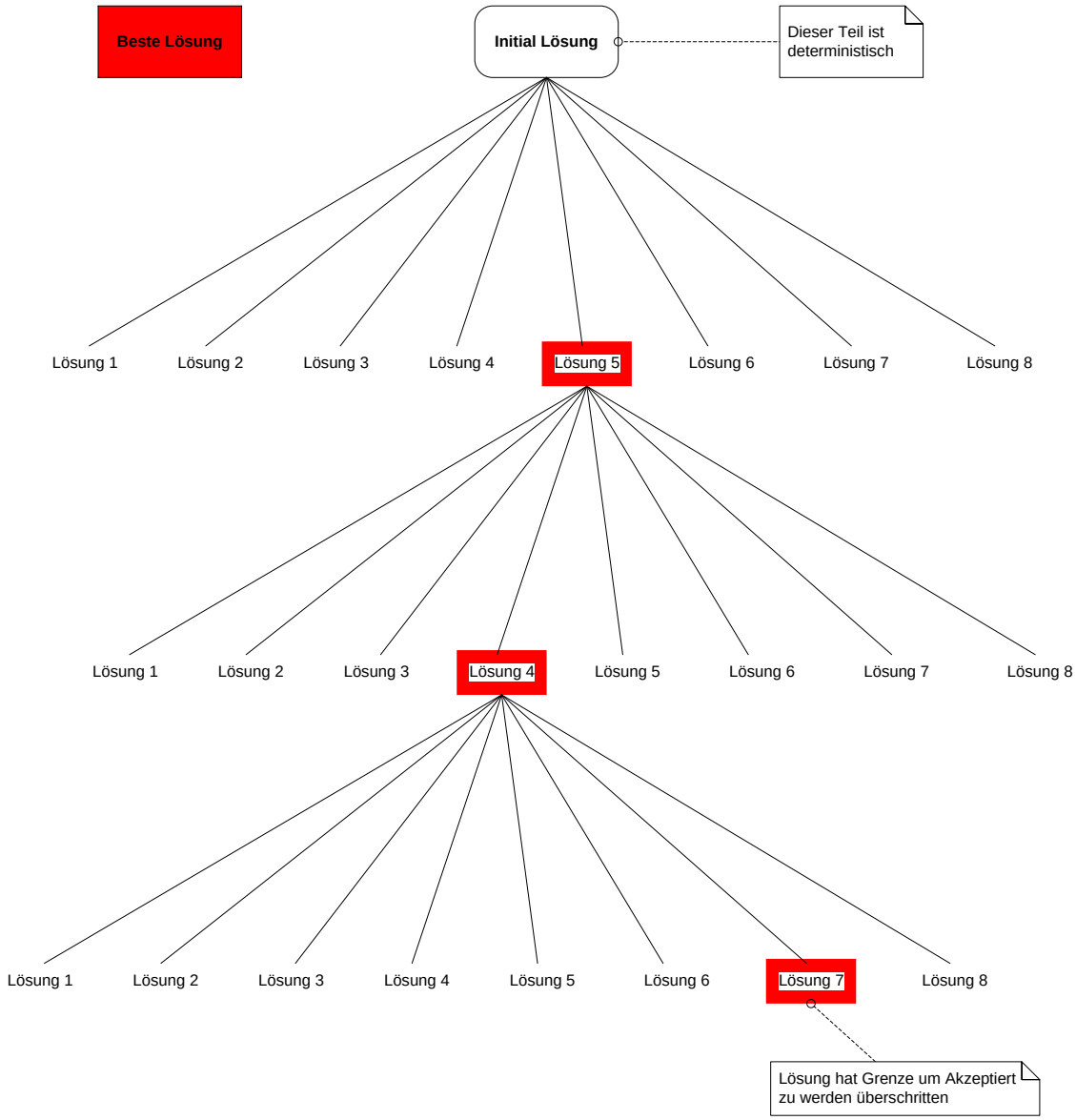
Nach diesem ersten Schritt wird ein pseudo genetischer Ansatz verwendet um die restlichen Stunden möglichst optimal zu platzieren. Es wird nun also eine Generation gebildet die aus einer fixen Anzahl von Stundenplänen besteht. Für jeden Stundenplan werden nun folgende Schritte durchgeführt:

- Es werden Stunden anhand der Bewertung des momentanen Stundenplans entfernt (beim allerersten Stundenplan, der noch leer ist, passiert in diesem Schritt nichts). Das heisst es wird für jeden Tag des Stundenplans der Bewertungsfaktor in Prozent genommen. Dieser Faktor dient als Schwelle ob eine Stunden entfernt wird oder nicht. Es wird nun durch alle Stunden iteriert und Random Prozentzahlen generiert. Ist die Zufallszahl grösser als die Schwelle so wird die Stunde aus dem Stundenplan entfernt und das Booking in die Liste der zu platzierenden Bookings verschoben. Das bedeutet je schlechter ein Tag bewertet wird umso mehr Stunden werden von ihm entfernt.

- Alle zu platzierenden Bookings werden nun möglichst Konfliktfrei in den Stundenplan eingetragen.
- Um zu verhindern, dass gewisse Tage fast keine Lektionen haben und andere wiederum fast überbucht sind, werden die Anzahl Lektionen pro Tag angeglichen.
- Da sich während der vorhergehenden Manipulationen möglicherweise Lücken gebildet haben, wird nun versucht möglichst viele Lücken durch herum schieben von Stunden innerhalb eines Tages diese zu schliessen.
- Es werden als letztes nun alle Konflikte aufgelöst, falls dies möglich ist.
- Der durch diesen Ablauf erstellte Stundenplan wird vom Bewertungssystem bewertet.

Sind nun alle Stundenpläne für eine Generation generiert worden, so wird die beste Lösung ausgewählt um fort zu fahren. Falls der Stundeplan bereits über der Bewertungsschwelle liegt bei der eine Lösung als brauchbar gilt, so wird der Algorithmus hier abgebrochen, ansonsten wird eine nächste Generation berechnet basierend auf der ausgewählten Lösung der jetzigen Generation.

Der Algorithmus baut also einen Generationenbaum in dieser Art auf:



4.5.1 Lektionen platzieren

Das eigene Modell ist relativ einfach gestrickt. Es beruht darauf, dass eine Priorisierung anhand von harten Kriterien durchgeführt wird. Innerhalb dieser Kriterien wird noch zusätzlich eine Priorisierung des möglichen Konfliktpotentials vorgenommen. Sobald diese Priorisierung vorgenommen ist, wird Fach für Fach abgearbeitet bis keine weiteren Stunden mehr zu verteilen sind.

Die harten Kriterien, nach denen eine Einteilung gemacht wird wären:

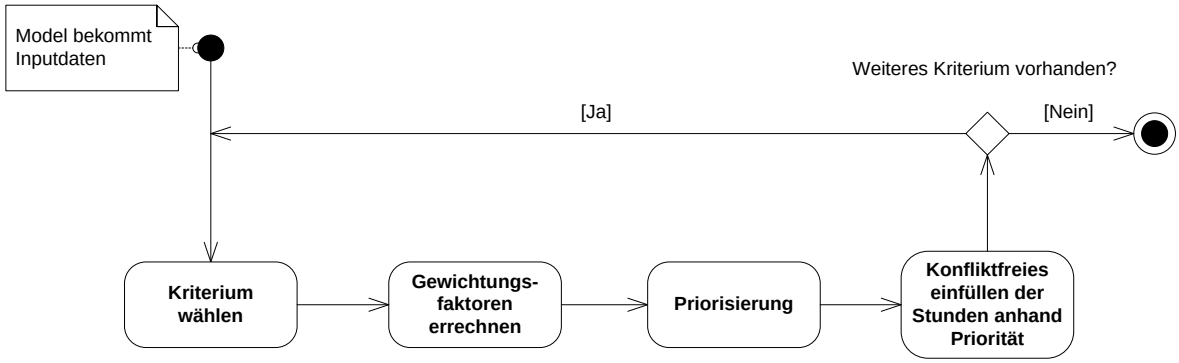
- Stunden mit speziellen Restriktionen
- Stunden die nicht vom Klassenlehrer gehalten werden können
- Stunden die der Klassenlehrer halten kann

Der Grund für diese übergeordnete Priorisierung ist, dass Fächer mit speziellen Restriktionen immer das grösste Konfliktpotential aufweisen. Denn diese sind oft Klassenübergreifen oder sind Parallelstunden. Stunden die nicht vom Klassenlehrer gehalten werden können sind insofern Kritisch als das die Ressourcen Lehrer, Raum, Klasse gleichzeitig verfügbar sein müssen. Alle Stunden die beim Klassenlehrer stattfinden sind relativ unproblematisch, da dort bereits alles vorgegeben ist (Lehrer hat oft auch ein Zimmer zugeteilt). Dass heisst es geht dort nur noch darum mögliche Lücken zu finden und dann die Lektion dort zu positionieren.

Das mögliche Konfliktpotential wird mit folgender Formel berechnet:

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

In einem ersten Ansatz wird auf das anschliessende Verschieben von Stunden bei Platzmangel verzichtet. Das heisst beim ersten Ansatz ist es möglich das Deadlocks entstehen wenn keine Freie Stunde mehr gefunden werden kann. Weil der Algorithmus dann nicht anfängt Stunden zu verschieben. Der Workflow dieses Modells sieht so aus:



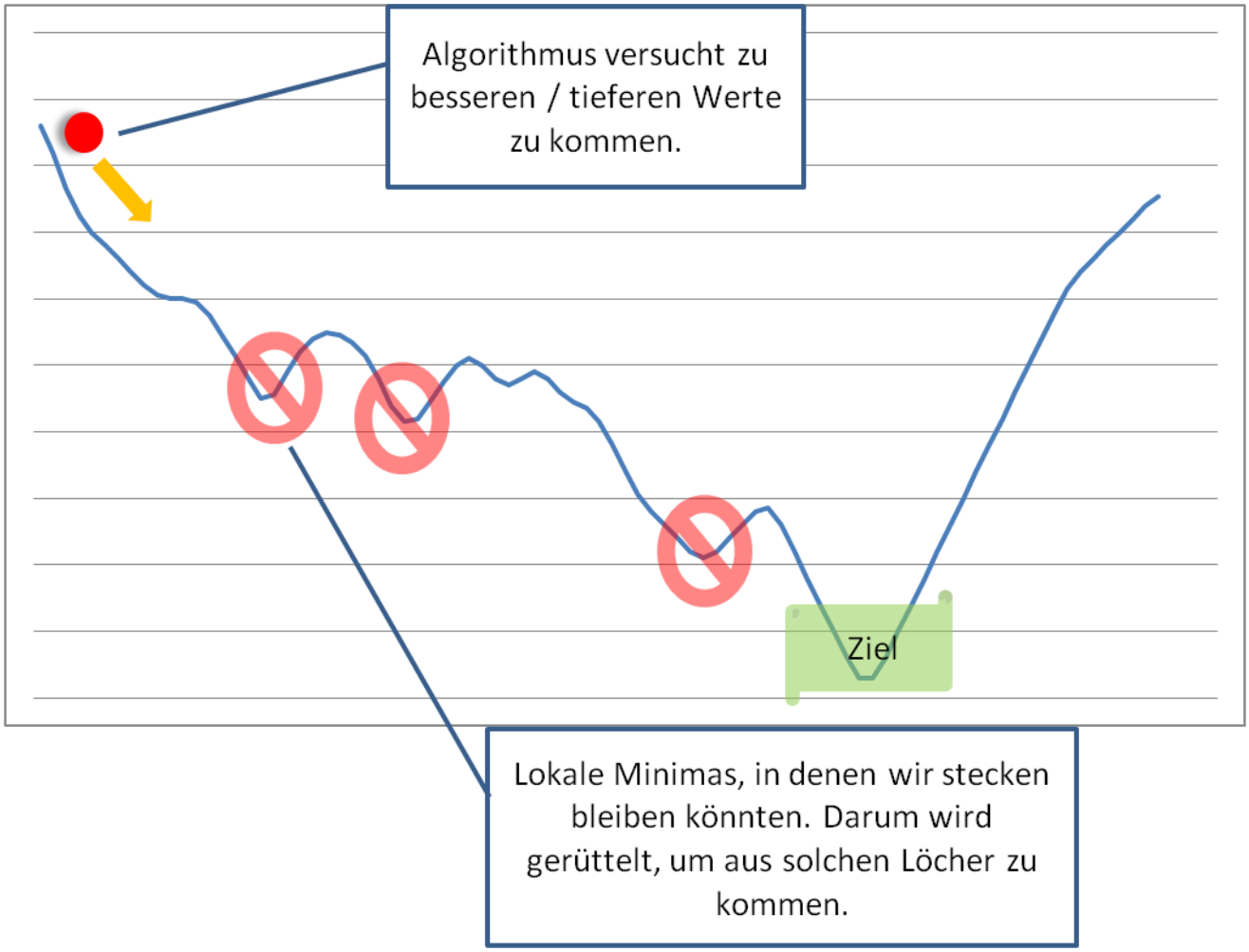
Wichtige Einflussfaktoren:

- Prefrences
- Target Hours
- Blocksize
- RoomType
- Location
- SchoolSubject & Level

Booking Positionierung:

- Lehrer-Ressource finden für die alle Stunden halten kann
- Schauen ob die ausgewählte Stunden alles im Zimmer des Lehrers gehalten werden kann
- Wenn nötig Ersatz Zimmer finden
- Stunden buchen

4.6 Lokale Minima vermeiden



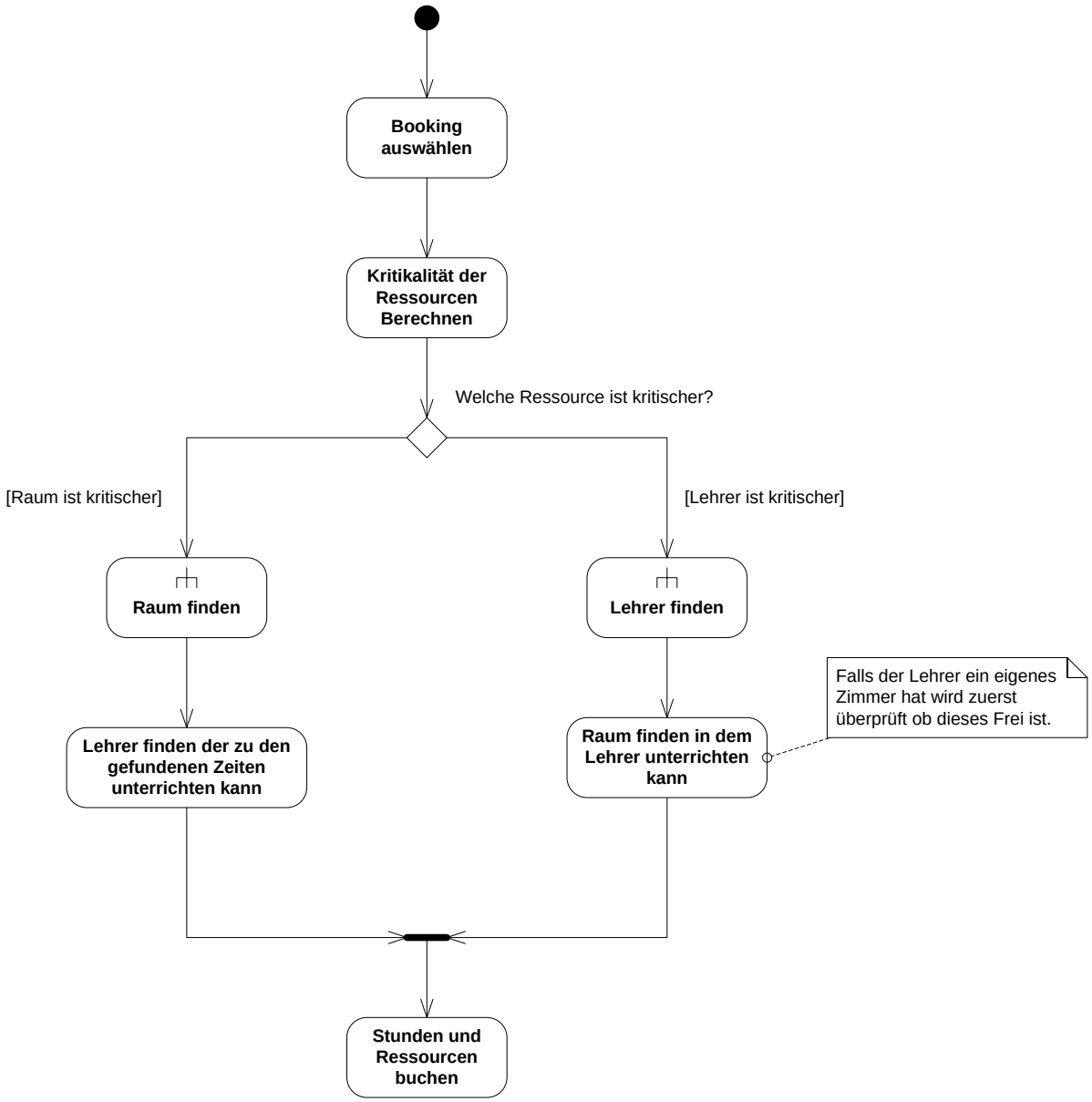
Wir implementieren einen nicht-terministischen, heuristischen Algorithmus. Dieser versucht nach und nach Verbesserungen zu finden, indem er die Lösungen mit besserer Bewertung nimmt. In solchen Art von Algorithmen kann es sein, das wir in einem lokalen Minimum sind. Dass heisst, alle Änderungen führen zu Verschlechterungen. Dabei wäre eine viel bessere Lösung ausserhalb dieses Minimums.

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

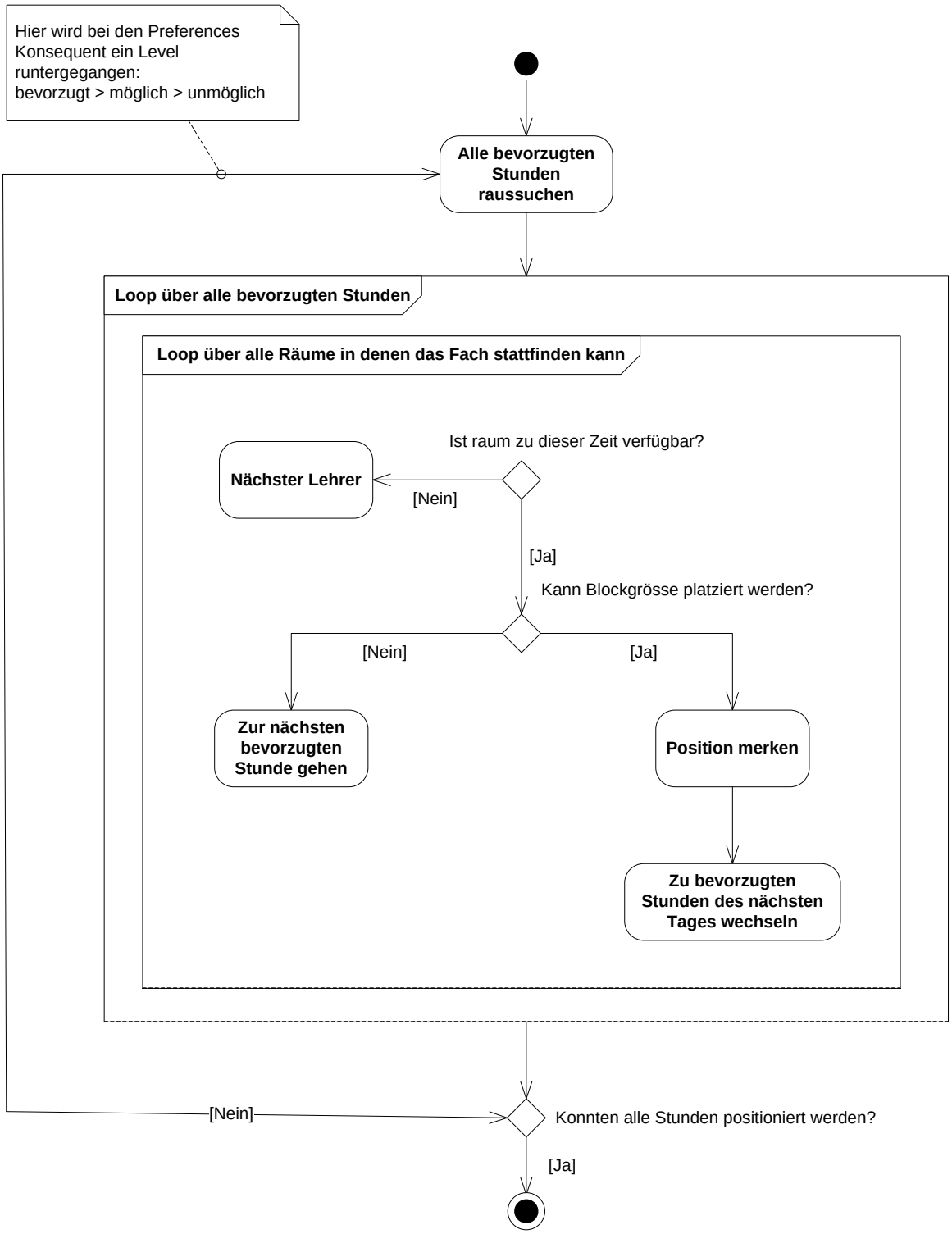
Eine der einfachsten Lösung bei solchen Problemen ist, das man ein zufälliges Rauschen beifügt. Diese Rauschen ‚rüttelt' an der Lösung, so dass der Algorithmus manchmal wieder aus den Minima heraus springt. Nun wird dieses Rauschen langsam vermindert / abgekühlt. Somit springt der Algorithmus gegen Ende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

In unserem Algorithmus werden 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.

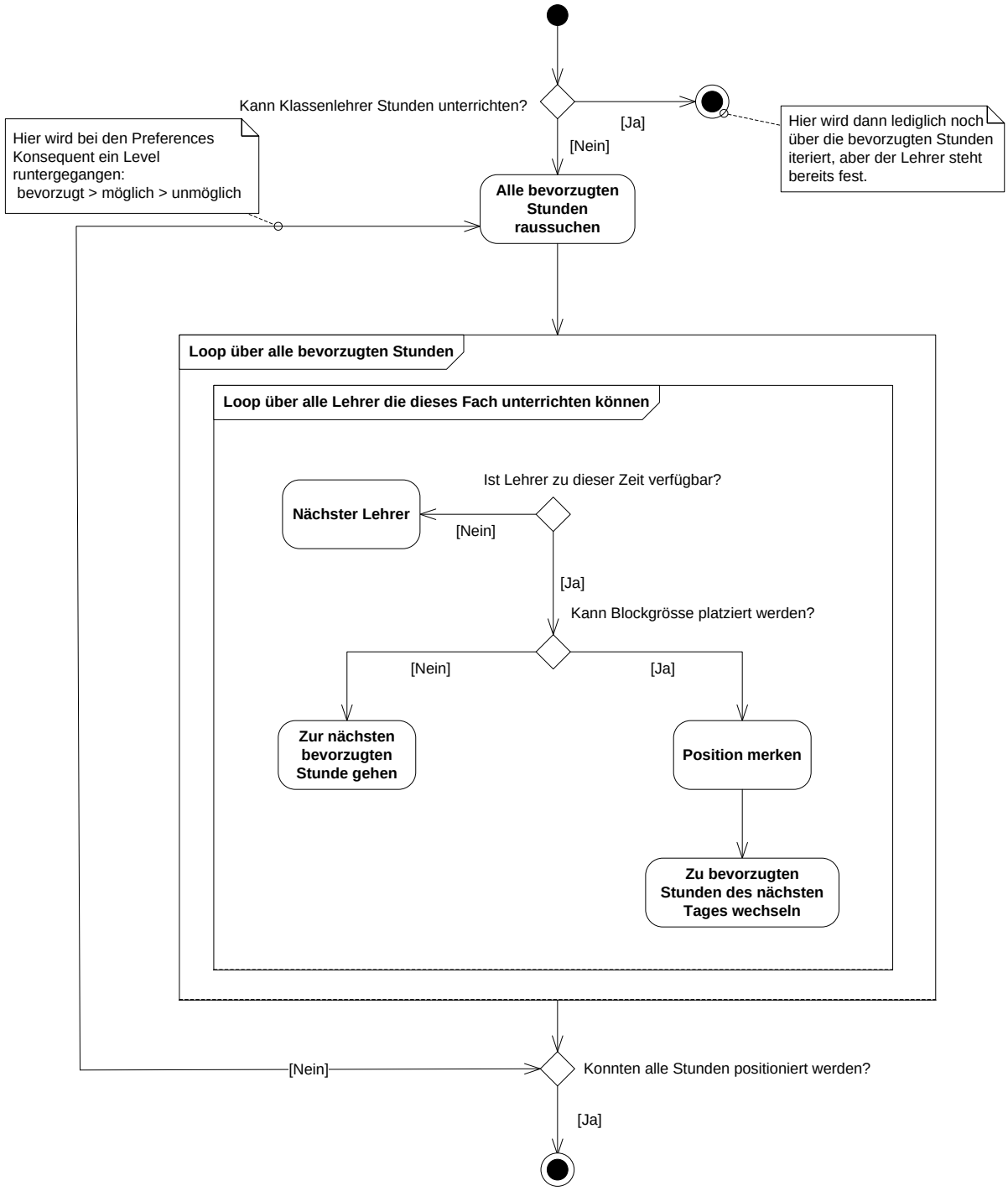
4.7 Ablauf Freie Ressourcen Finden



4.7.1 Raum finden



4.7.2 Lehrer finden



5 Neuen Algorithmus einbauen

5.1.1 Modularisierung-Konzept

5.1.2 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.

5.1.3 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 die beim Start der Applikation aufgerufen welche am Extension-Point ‚IEagerStarted' angemeldet sind.

5.1.4 Generische Factories (Providers)

Der IoC-Container stellt Generische Factories zur Verfügung. Dazu fordert man in der Client-Komponente nicht den Service direkt an, sondern ein Provider für ein Service. Danach kann man auf der Provider-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 dies sehr praktisch, dass dieses Pattern übernommen wird.

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

5.1.5 Modul

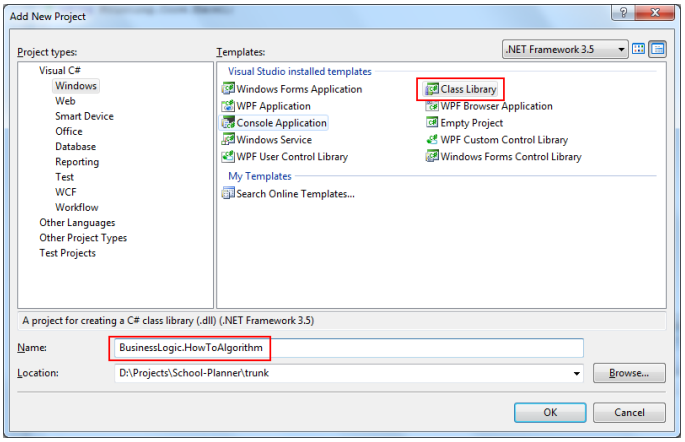
Die Applikation wird in lauter kleine Module unterteilt, die als separate Visual Studio Projekte geführt werden. Dem entsprechend werden diese auch zu separaten Assemblies kompilierte. Jede funktionale Einheit, die unabhängig von anderen Teilen der gleichen Schicht ist, wird als ein eigenes Modul geführt. 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 ein 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 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.

5.2 HowTo für eigenen Algorithmus

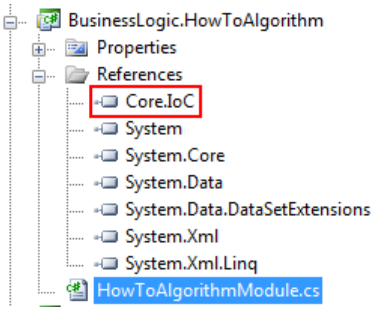
5.2.1 Projekt erstellen

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 werden deren Assembly im Ausführungspfad befindet.



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

5.2.2 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 Projekt verwendet werden können.

Der Code in der HowToAlgorithmModule Klasse sieht folgendermassen aus:

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

5.2.3 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;
        }
    }
}
```

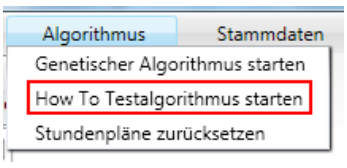
5.2.4 Anhängen zum ExtensionPoint

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
    {
        …
    }
}
```

5.2.5 Ausführen



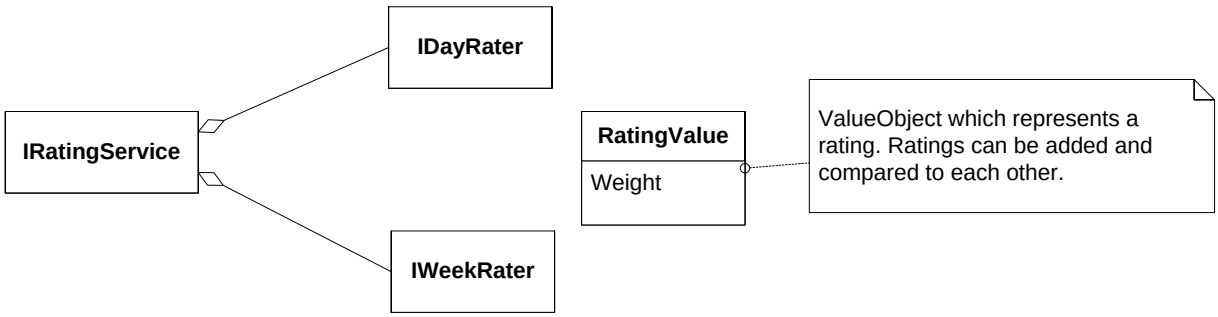
Bevor nun das ganze Ausgeführt wird, muss dieses Projekt noch dem Startup Projekt MySchoolPlanner hinzugefügt werden. Dadurch wird das Assembly unseres Projektes in den Ausführungspfad kopiert.

Nun kann die Applikation gestartet werden und wir sehen unseren Algorithmus im Menüeintrag. 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.

6 Rating-Framework

6.1 Komponenten



6.1.1 Der Rating-Service

Die Komponente ist der Einstiegpunkt für Konsumenten. Beim IRatingService kann man ein Rating für eine 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. Am Schluss sammelt er das Resultat und gibt es zurück.

6.1.2 Die Rater

Ein Rater kann ein 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.

6.1.3 RatingValue

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

6.2 Implementierte Raters

6.2.1 BlockSizeRater

Diese Bewertung schaut, ob die vorgegebene Blockgrösse eingehalten ist. Bei Verletzungen der Blockgrösse-Vorgaben gibt es Wertungs-Abzüge

6.2.2 ConflictRater

Sobald eine Ressource wie Lehrer oder Raum doppelt gebucht ist, gibt dieser Rater Abzug.

6.2.3 GapRater

Lücken zwischen zwei gleichen Lektionen geben Abzug. Z.B. Deutsch->Mathe->Deutsch ist schlechter als Deutsch->Deutsch->Mathe.

6.2.4 HasLessonForDayRater

Schaut das jeder Tag Schulstunden hat, ansonsten gibt es Abzug.

6.2.5 SameLessonTwiceRater

Schaut das die gleiche Lektion nicht mehrmals am Tag vorkommt. Ergänzung zum ‚GapRater'

6.3 Weitere Rater einbauen

Das Einbauen eines Raters beruht auf denselben Prinzipien wie das Einbauen eines neuen Algorithmus. Siehe Kapitel: Neuen Algorithmus einbauen.

Zuerst wird die Klasse für den Rater erstellt. Die Raters befinden sich im , BusinessLogic.AlgorithmInterface.Rating'-namespace. Die Klasse muss entweder IDayRater, IWeekRater oder beide Interfaces implementieren. Danach muss man die Klasse an die 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);
        }
    }
}
```

7 Zusätzliche Framework Features

7.1 Zusätzliche Features

7.1.1 CRUD-Umgebung

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

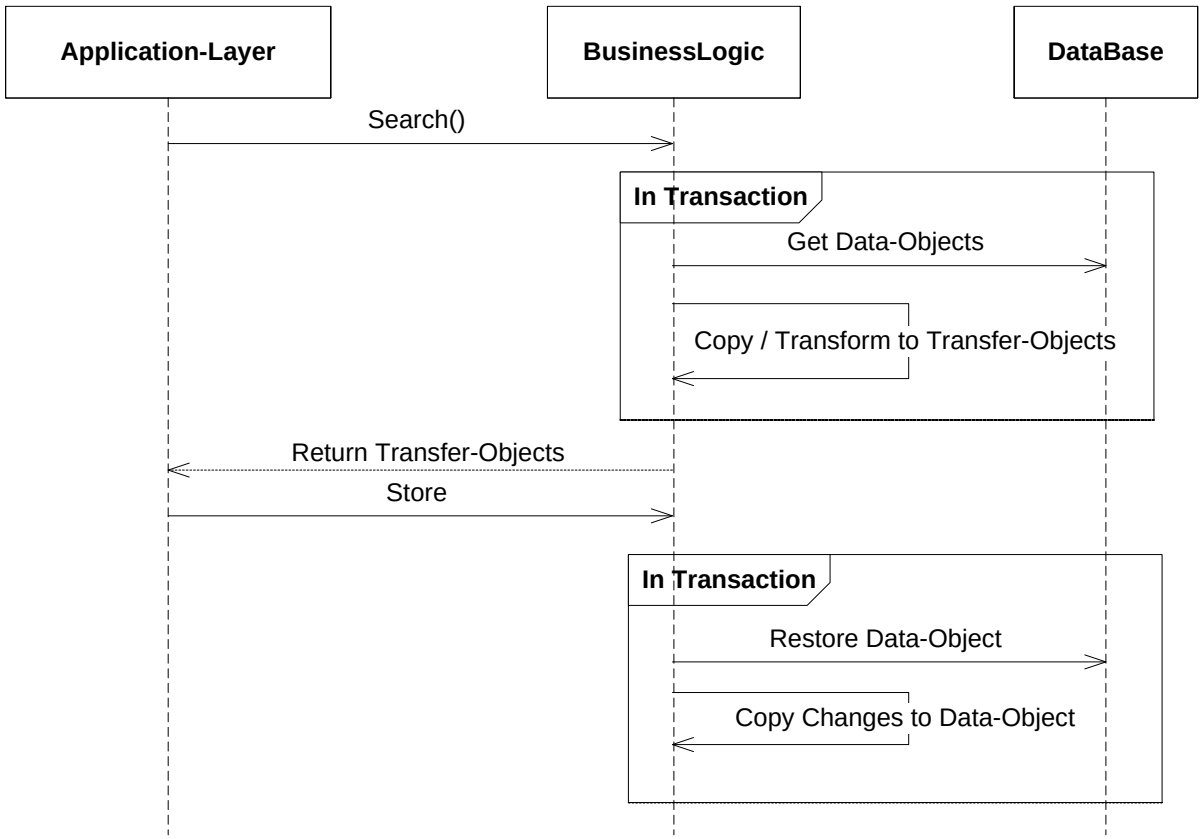
7.1.2 Aktualisierung der interaktiven Stundenplan Erstellung

Um die Algorithmen mit genügend Information zu versorgen sind viele neue Stammdaten erfasst. Diese beeinflussen die interaktive Stundenplan Erstellung. Diese beachtet ebenfalls die Eingabedaten für die Algorithmen. Damit ist eine bessere Benutzerführung realisiert, weil das System bessere Vorschläge macht.

7.2 CRUD-Design

Beim Design des CRUD-Framework 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.

7.2.1 Ablauf CRUD

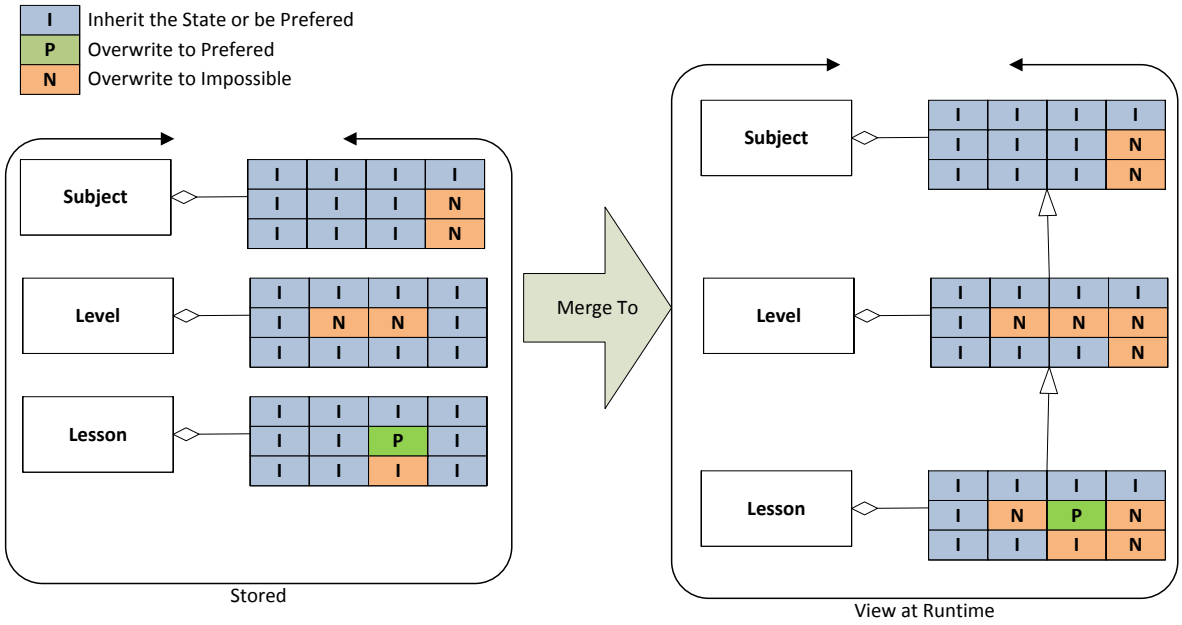


7.2.2 Implikationen diese Models

Bei Anfragen an die Business-Logik werden immer neue Transfer-Objekte erstellt. Somit ist die Identität diese Objekte wertlos. Das 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.

7.3 Vererben der Zeitausprägung

Bei Stundenplänen gibt es meistens Einschränkungen, wann gewisse Lektionen stattfinden. Diese Einschränkungen beziehen sich auf verschiedene Stammdaten. So kann sich eine Einschränkung auf einen Lektions-Typ (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 konkreten Fächer für die 1. Primar. Optional kann eine ‚vererbte' Zeitausprägung explizit wieder aufgehoben werden.

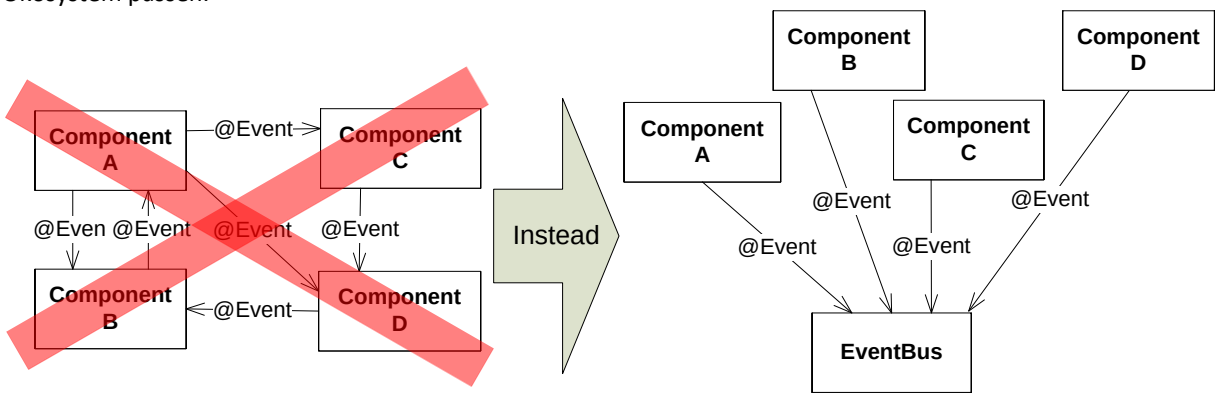


7.4 Event-Bus

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 Events-Benutzt, da sie wunderbar ins .Net-Ökosystem passen.

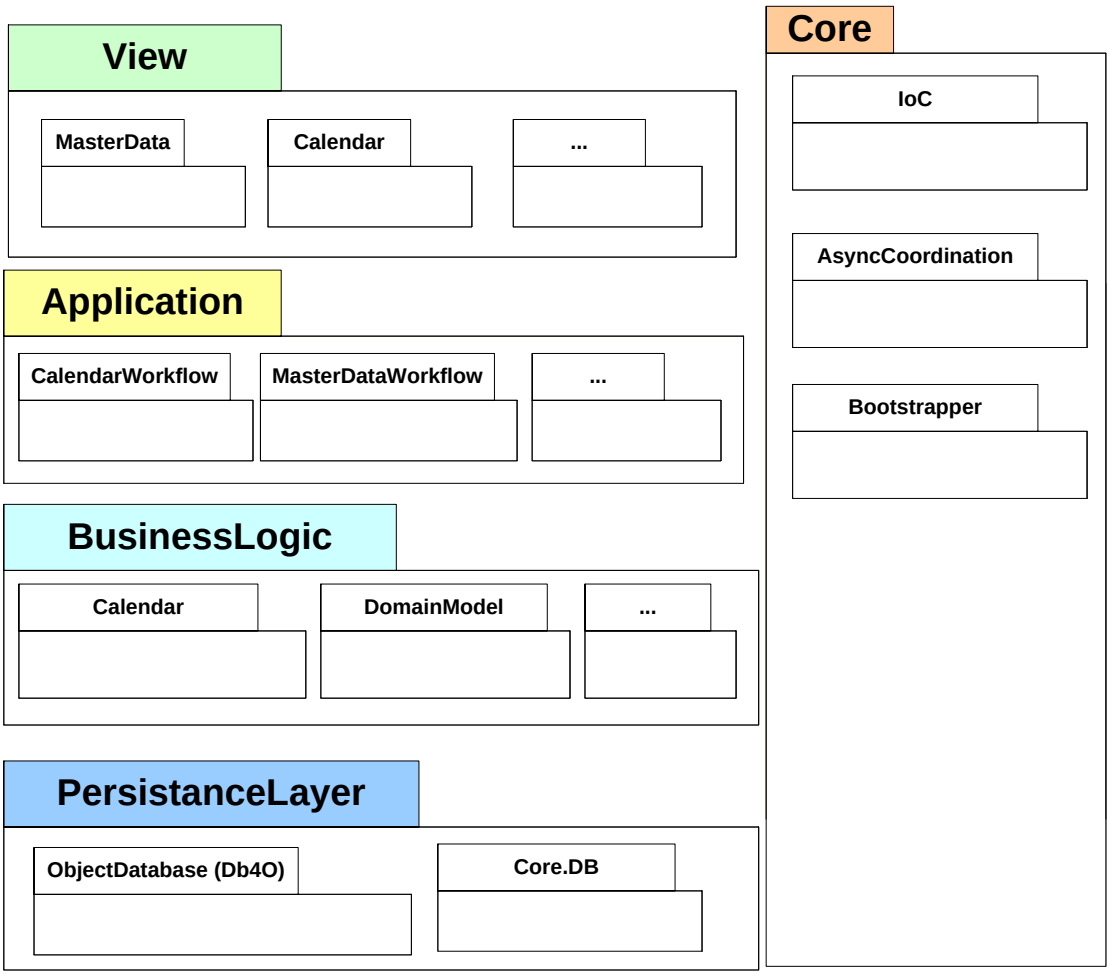


7.5 Disposer-System

Wir verwenden das Dispose-Pattern um Ressourcen frei zu geben. Ein Service kann eine solche Ressource sein. Wenn nun eine Service-Implementation weitere Services benutzt, fordert er diese über den Konstruktor an. Die Implementation soll nicht wissen wie lange der Lebenszyklus der übergebenen Services ist. Darum kann die Implementation nicht einfach alle benutzen Services ‚disposen'.

Dafür gibt es ein Disposer-System. Dieses verfolgt die Instanziierung eines Disposable-Services. Es verfolgt die Abhängigkeiten von allen Disposable-Service-Implementation. Am Schluss sorgt es dafür, das alle Services ordnungsgemäss ‚entsorgt' werden. Dieses System basiert auf den Ressourcen-Management des IoC-Containers.

7.6 Strukturierung der Applikation



7.6.1 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.

7.6.2 Application

Diese Schicht kontrolliert die Abläufe der Applikation und koordiniert die UseCases. 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)

7.6.3 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 und Stammdaten-Verwaltung sind in dieser Schicht implementiert.

7.6.4 Persistence Layer

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

7.6.5 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.

8 Environment

8.1 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.

8.1.1 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.

8.2 Issue Tracking

Die ganze Projektplanung Issue sowie Change-Management wird in unserem Issue-Tracker abgehandelt, der uns für die Semester 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.

9 Testing

9.1 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.

9.2 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.

9.3 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.

9.4 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.

10 Offene Punkte

Auszug Issue-Tracker siehe Anhang.

10.1 Stammdaten

- Klasseneigenschaften Dialog überarbeiten
- Preferences Hierarchie: Klasse -> Subject -> Lesson
- Bei platzierten Stunden soll es möglich sein Attribute (z.B. Raum) zu ändern ohne die Stunde wieder zu löschen

10.2 Rater

- Ratings aus verschiedenen Sichten: Lehrer, Schüler, Raum

10.3 Algorithm Interface

- Das Laden der Daten von der Datenbank ist extrem inperformant und wurde bis jetzt in keinster Weise optimiert

10.4 „Pseudo“ Genetic Algorithm

- Der Algorithmus ist noch nicht wirklich Genetisch, es muss noch ein Merge zwischen den verschiedenen Ergebnissen einer Generation gemacht werden
- Intelligente Konfliktauflösung, mit Raum / Lehrer austauschen
- Das berechnen einer Generation geschieht noch seriell. Das könnte man auf parallel umstellen um die Performanz der Berechnung zu erhöhen.

10.5 Further Features

- Anzeige ob alle Stunden verbucht sind, noch zu verbuchende Stunden anzeigen
- Modernes Drag & Drop für das manuelle Einfügen von Stunden
- Möglichkeit einzelne Fächer vom Algorithmus verbuchen zu lassen, um eine halbautomatische Stundenplangenerierung zu erreichen
- Auf starten mit leerer Datenbank führt dazu, das automatisch der Semester Erstellungs-Wizard aufgerufen wird
- End-User-Dokumentation
- Grafische Auswertung einer bewerteten Lösung
- HSR-Stundenplan-Szenarien unterstützen

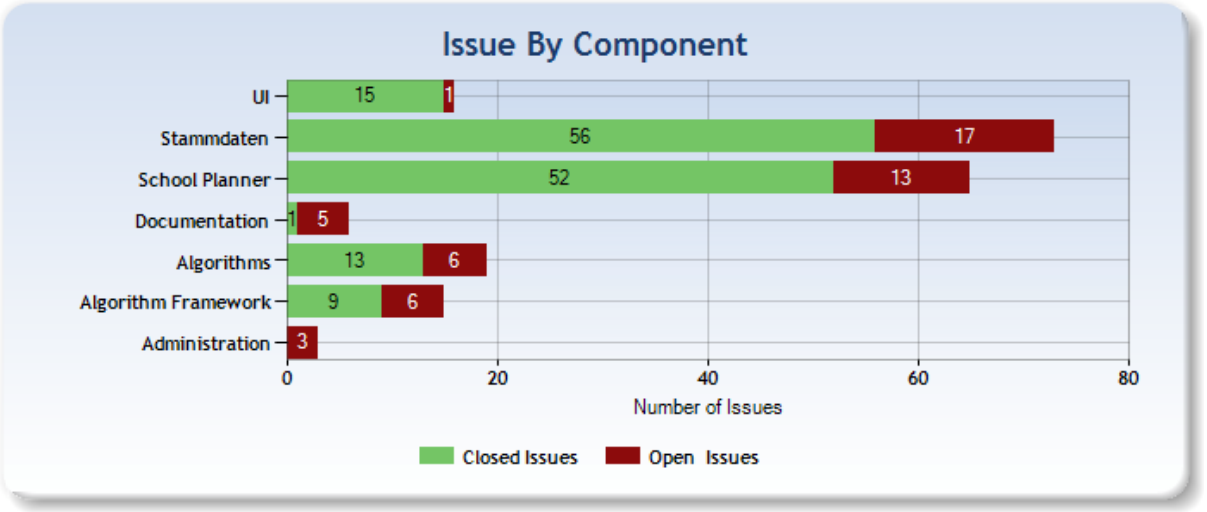
10.6 Test-Bett erstellen

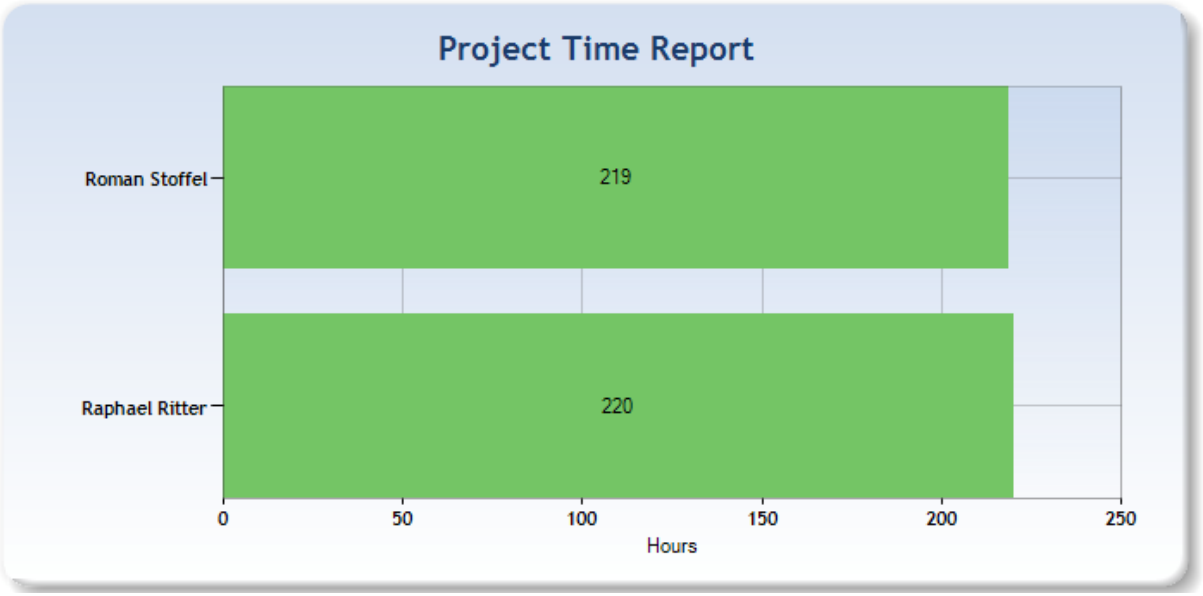
- Muster-Stundenpläne erstellen
- Laden existierender Stundenpläne in Algorithmen
- Rating auf vorhanden Stundenpläne

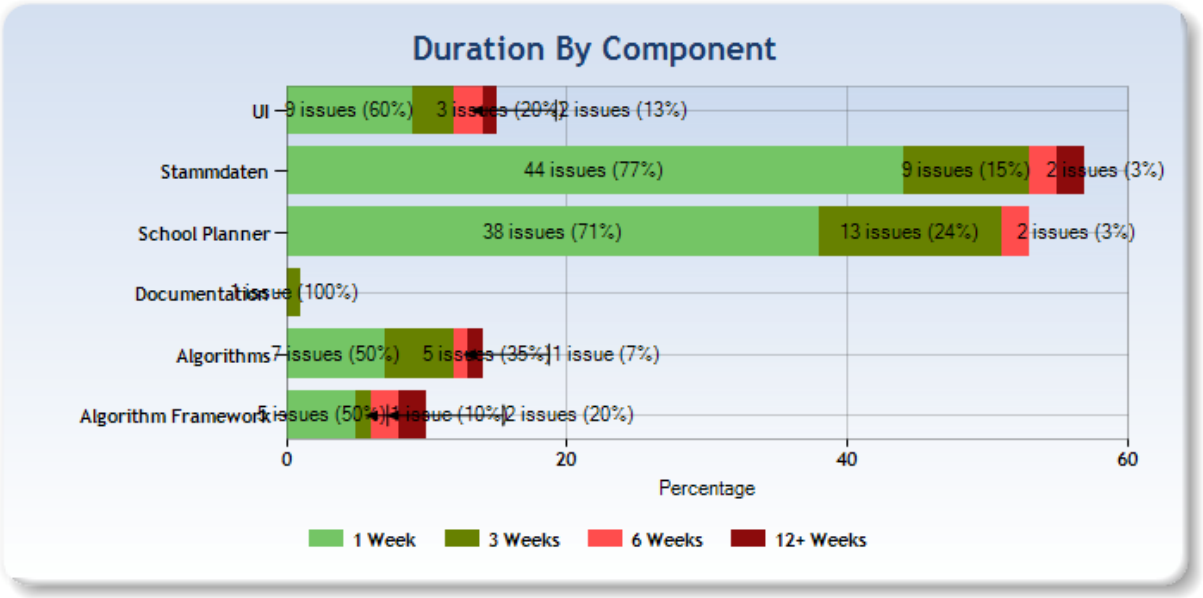
11 Anhang

11.1 Arbeitsverteilung / Stundenrapporte

Alle Daten von unserem Issue Tracking System haben den Stand vom 10.12.2009.







11.2 Offene Issues

Issue	Type	Component	Summary
SA-154	Task	Stammdaten	There is no null item for optional selections like classroom for teacher
SA-180	Task	School Planner	Improve Speed for reading / writing data
SA-194	Bug	School Planner	Database is growing and growing
SA-211	Task	School Planner	Insert-Logic in Room/ Teacher-Calendars
SA-216	Bug	Algorithm Framework	Rating is very low
SA-223	Enhancement	Algorithm Framework	Algorithm not stopped if started and window closed
SA-225	Bug	Stammdaten	Change in Subject view does not change stuff in Lesson view
SA-229	Bug	Algorithms	ConflictResolver strategy not always working
SA-162	Bug	Stammdaten	To set preferences on the teacher, tab has to be closed first

SA-232	Enhancement	Algorithms	Calculation of one generation Multi threaded
SA-119	Task	Algorithm Framework	Performance in algorithm data exchanges unacceptable
SA-109	Task	Stammdaten	Review Class Target View
SA-105	Bug	Stammdaten	Class Settings
SA-102	Bug	Stammdaten	Class Targets can't be removed
SA-200	Enhancement	Stammdaten	Overwork classtargets view
SA-118	Task	Stammdaten	Combobox and 'Deleted' Items
SA-111	Task	Stammdaten	Preferences on SchoolSubject
SA-207	Task	Stammdaten	Create preferences on Class
SA-184	Task	Stammdaten	Adding a 'Defaul'-Flag for all Types like RoomType, LevelType
SA-212	Task	Algorithm Framework	Raters should be configurable
SA-117	Task	Stammdaten	Inconsisting naming of 'Level' and 'Stufe'.
SA-224	Bug	Algorithms	Algorithm is ignoring the Active flag
SA-31	Bug	Stammdaten	Search-terms are case-sensitive
SA-176	Enhancement	Stammdaten	Search should be better
SA-228	Task	Algorithms	Create new Strategy for aglorithm to distribute lessons fair
SA-129	Task	Stammdaten	There should be a view to edit the Timeslices
SA-230	Enhancement	School Planner	ConflictResolved event very dodgy
SA-179	Task	School Planner	First startup loads semester creation wizard
SA-110	Quality Check	School Planner	Binding errors in Teacher view
SA-58	Task	School Planner	Remove IPlanningCalendar as key's in internal ui column repre-sentation
SA-79	Enhancement	Stammdaten	On editable-tables: Add new entries with 'Enter' & 'Tab'
SA-80	Enhancement	Stammdaten	On editable-tables: Add new entries with 'Enter' & 'Tab'
SA-83	Task	School Planner	Using the Application
SA-93	Task	School Planner	Disposable-Hirachie / -Chain
SA-122	Task	School Planner	Managing libraries
SA-187	Task	School Planner	On application startup with empty db
SA-96	Bug	UI	Crash if no lesson is selected in Klasseneigenschaften dialog
SA-130	Task	Algorithms	Create Printing engine
SA-149	Task	School Planner	Move Logic from application layer to Businesslayer for insertion information
SA-133	Bug	School Planner	Conflicts are not disappearing after restart of the application