Implementierungsdokument

Hanselmann, Hecht, Klein, Schnell, Stapelbroek, Wohnig

13. Februar 2017
v0.2

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Dieses Dokument beschreibt die Implementierungsphase einer Praxis der Softwareentwicklungsgruppe am Karlsruher Institut für Technologie. Der Titel der Gruppenaufgabe lautet: *Entwicklung eines Werkzeugs zur Analyse formaler Eigenschaften von Wahlverfahren.*
Diese Dokument stellt die in dieser Phase entstandenen Unterschiede zu den vorherigen Phasen (Pflichtenheft und Entwurf) dar und erklärt, warum diese notwendig wurden.
Weiterhin wird die zeitliche sowie die personelle Aufteilung der Implementierung vorgestellt.

1.1 Ziel des Programmes

Ziel des Programmes ist es, eine Lösung zur Analyse von formalen Eigenschaften von Wahlverfahren zu präsentieren. Zur Analyse der Eigenschaften wird Bounded Model Checking (Glossareintrag) verwendet. Der verwendete Bounded Model Checker ist CBMC (Glossareintrag). Das Programm soll folgende Module bereitstellen:

- Eine Möglichkeit zur Beschreibung eines Wahlverfahrens in der Programmiersprache C.
- Eine Möglichkeit zur Beschreibung von Eigenschaften, auf die das Wahlverfahren geprüft werden soll. Die Beschreibung erfolgt in einer Makrosprache (Glossareintrag).
- Eine Möglichkeit zum Angeben der Parameter für welche das angegebenen Wahlverfahren analysiert werden soll (Anzahl Wähler, Anzahl Kandidaten, Anzahl Sitze).
- Eine Möglichkeit, die Analyse auszuführen.
- Eine Ausgabe des Ergebnisses der Analyse: Eine Erfolgsmeldung falls alle Eigenschaften erfüllt werden und Präsentation eines Gegenbeispiels sonst.

2 Unterschiede zu den im Pflichtenheft gestellten Kriterien

2.1 Kannkriterien

2.1.1 Betrieb auf dem Betriebssytem MacOS

Da wir keinen MAC zum Testen hatten, haben wir keine Implementierung für MacOS vorgenommen. Da unser Entwurf Erweiterungen aber leicht zulässt, ist es für Benutzer, die dieses Betriebssystem besitzen mit einigen Programmierkenntnisse möglich sich diese Option selbst einzubauen. Dazu müsste die Funktionalität erstellt werden, Prozesse auf diesem Betriebssystem von Java aus zu starten. Mit dieser müssten dann CMBC sowie GCC angesprochen und deren Ausgabe zurückgegeben werden. Alle anderen Systeme sind unabhängig vom Betriebssystem.

3 Änderungen am Entwurf

3.1 Package Highlevel

Das Paket `highlevel` bildet den Kern von BEAST. Es enthält die `MainClass`, über die BEAST gestartet wird. Weiterhin enthält es Interfaces zu den anderen Paketen, wodurch diese Pakete unabhängiger von einander sind und damit leichter auszutauschen. Der `CentralObjectProvider` generiert Instanzen dieser Interfaces und stellt sie dem `BEASTCommunicator` bereit.

3.1.1 CentralObjectProvider

`AbstractFactory` in `highlevel` ist jetzt keine Abstrakte Fabrik mehr.
Dieses Entwurfsmuster konnte nicht verwendet werden, da die zu erstellenden Objekte teilweise voneinander abhängig sind. Weiterhin gibt es Objekte, die mehrere Rollen einnehmen, d.h. sie implementieren unterschiedliche Interfaces (wie etwa der Parametereditor, der sowohl `ParameterSource`, als auch `ProjectSource` und `MainNotifier` implementiert).
Deshalb wurde sie durch das Interface `CentralObjectProvider` und die Klasse `PSECentralObjectProvider`, die dieses implementiert, ersetzt. `CentralObjectProvider` verwirklicht die ursprüngliche Funktion der Abstrakten Fabrik, unabhängig von konkreten Implementierungen zu sein.
`PSECentralObjectProvider` erzeugt die konkreten Objekte für unsere Implementierung der `highlevel`-Interfaces und stellt diese dem `BEASTCommunicator` zur Verfügung. Dieser muss weiterhin nur von den Interfaces wissen.

3.1.2 DisplaysStringsToUser

Es wurde das Interface `DisplaysStringsToUser` hinzugefügt. Es wird von allen Elementen, die dem Nutzer Text anzeigen, implementiert. Damit wird die Einbindung anderer Sprachen vereinfacht.

3.1.3 start-/stopReacting

Allen Interfaces zu Paketen mit GUI wurden die Methoden `stopReacting` und `resumeReacting` hinzugefügt. Diese verhindern, dass der Nutzer während einer laufenden Analyse Änderungen

an dafür benötigten Daten vornimmt.

3.1.4 ProjectSource

Es wurde das Interface `ProjectSource` hinzugefügt. Es wird von `ParameterEditor` implementiert. Damit wird es möglich, das Speichern und Laden von ganzen Projekten in zukünftigen Versionen leichter einem anderen Fenster als dem Parametereditor zu überlassen.

3.1.5 isCorrect

Interfaces zu Paketen, die Daten für die Analyse bereitstellen, wurde die Methode `isCorrect` hinzugefügt. Damit kann vor Start einer Analyse überprüft werden, ob die bereitgestellten Daten frei von Fehlern sind, die die Analyse beeinträchtigen würden.

3.2 CodeGenerierung

Die Klasse `CBMCCodeGeneration` ist nicht mehr statisch.
Sie wird in der Implementierung von der Klasse `CBMCProcessFactory` instantiiert.
So wird für jedes erzeugte C-Tempfile (Glossareintrag) eine neue Instanz erstellt. Das ist sinnvoll, da es genau von den Parametern abhängt.
Jede Instanz der Klasse `CBMCCodeGeneration` erstellt eine Instanz der Klasse `CBMCCodeGenerationVisitor`.
Diese besitzen zwei neue Methoden, die einstellen, ob er zur Codegenerierung einer Vor- oder Nachbedingungen eines Wahlverfahrens verwendet wird.
(Verändertes Klassendiagramm hier)

3.3 UserActions

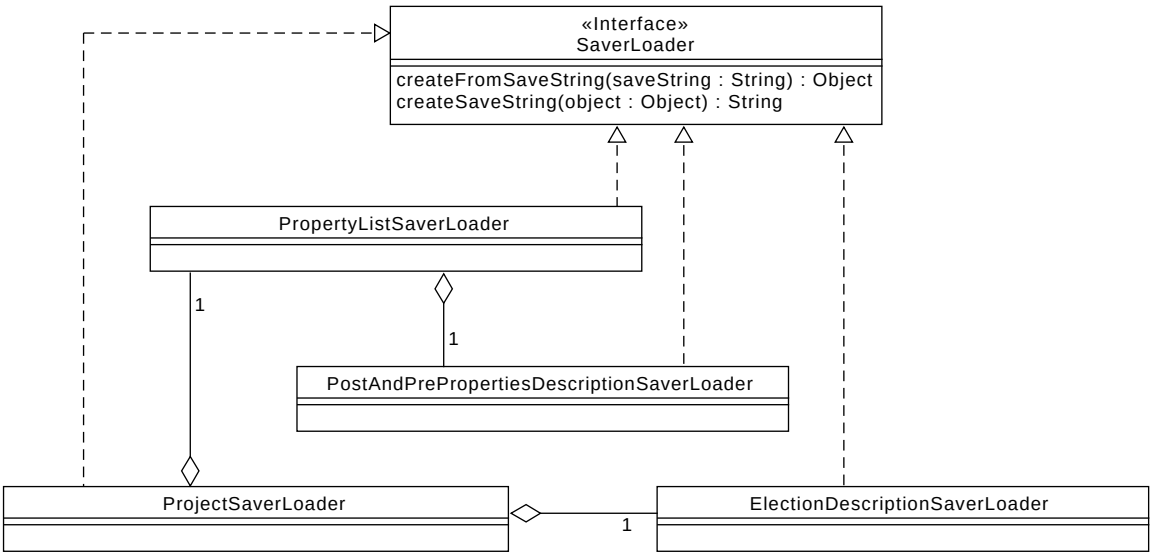
Alle `UserActions` der vier GUIs haben jetzt nur noch einen Verweis auf den ihnen zugehörigen Controller, und holen sich von diesem mit Gettern die von ihnen gebrauchten Klassen (`FileChooser`, `SaveBeforeChangeHandler`...). Beispielhaft am `BooleanExpEditor` gezeigt:

(Diagramm folgt)

3.4 SaverLoader

`PostAndPrePropertiesDescriptionSaverLoader`, `ElectionDescriptionSaverLoader`, `ElectionCheckParameterSaverLoader` und `ProjectSaverLoader` implementieren nun

das Interface `SaverLoader` mit den dargestellten Methoden. Dies ermöglicht es der Klasse `FileChooser`, polymorph gegebene DatenTypen abzuspeichern und gegebene Dateien zu laden. Alle anderen `SaverLoader`-Klassen haben nur statische Methoden. Zudem gibt es noch eine `StringSaverLoader` Klasse, die mit `createSaveString` aus allen vom Nutzer editierbaren Strings alle Vorkommen von »"durch »¿ërsetzt, beziehungsweise dies mit !`createFromSaveString` rückgängig macht. Dies verhindert die Erstellung von nicht ladbaren Dateien trotz valider Nutzer-Eingaben.

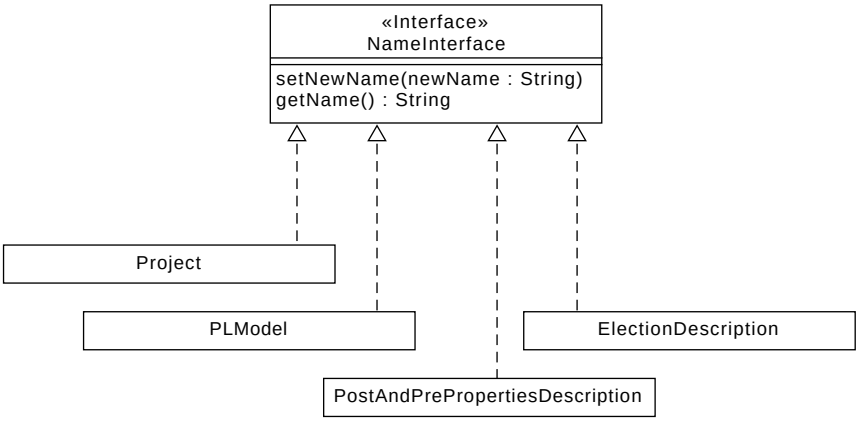


3.5 FileChooser

Diese Klasse kümmert sich um das Laden und Speichern der speicherbaren Datentypen. (Diagramm folgt)

3.6 DataTypes

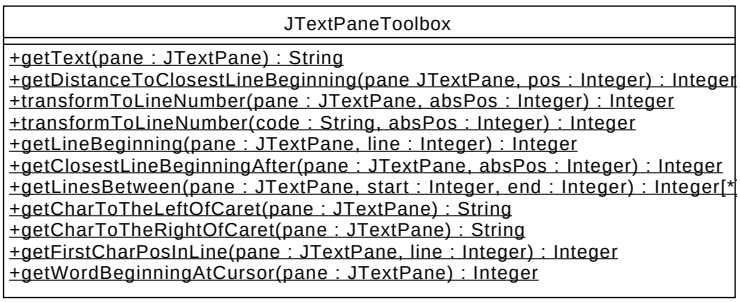
Die als Datei abspeicherbaren Datentypen implementieren nun alle das Interface `ChangeNameInterface`, das es dem `FileChooser` ermöglicht das name-Attribut dieser Klassen polymorph zu verändern.



3.7 Codearea

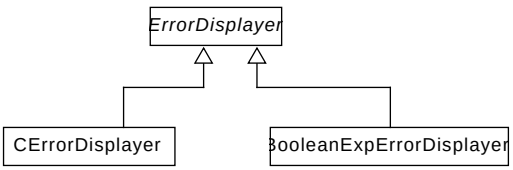
3.7.1 JTextPaneToolbox

Klasse `JTextPaneToolbox` wurde hinzugefügt. Diese enthält einige statische Methoden, welche oft benötigte, aber nicht zusammengehörige Funktionalität für `JTextPane` liefern. Dazu gehört unter anderem das Umwandeln absoluter Positionen in Zeilennummern.



3.7.2 Errordisplayer

`Errordisplayer` ist nun abstrakt. Von den erbenden Klassen müssen Fehlermeldungen generiert werden.



3.7.3 SaveTextBeforeRemove

DieKlasse `SaveTextBeforeRemove` wurde hinzugefügt. Diese speichert den Text einer `JTextPane`, sobald Text daraus entfernt wird. Dies ist nötig, da das `RemovedUpdate` des Styleddocuments keinen Zugriff auf den entfernten Text gewährt. Dieser wird jedoch benötigt, um Aktionen rückgängig zu machen. Implementiert wird die Funktionalität durch hören auf Keyevents.

3.7.4 TextLineNumber

Klasse `TextLineNumber` wurde hinzugefügt, welche die Zeilennummer anzeigt. Diese Klasse wurde direkt aus
https://tips4java.wordpress.com/2009/05/23/text-component-line-number/ übernommen.

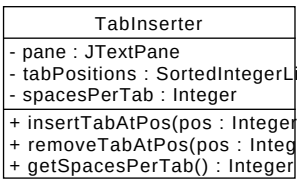
3.7.5 SquigglePainter

Klasse `SquigglePainter` wurde hinzugefügt. Diese unterstreicht Text in der `JTextPane` gezackt. Dies wird verwendet, um Fehler im Code anzuzeigen. Übernommen von https://tips4java.wordpress
painter/

3.7.6 JTextPaneToolbox

3.7.7 Tabinserter

`Tabinserter` wurde hinzugefügt. Dieser fügt Tabs in Form von Leerzeichen ein.



3.7.8 LineBeginningTabsHandler

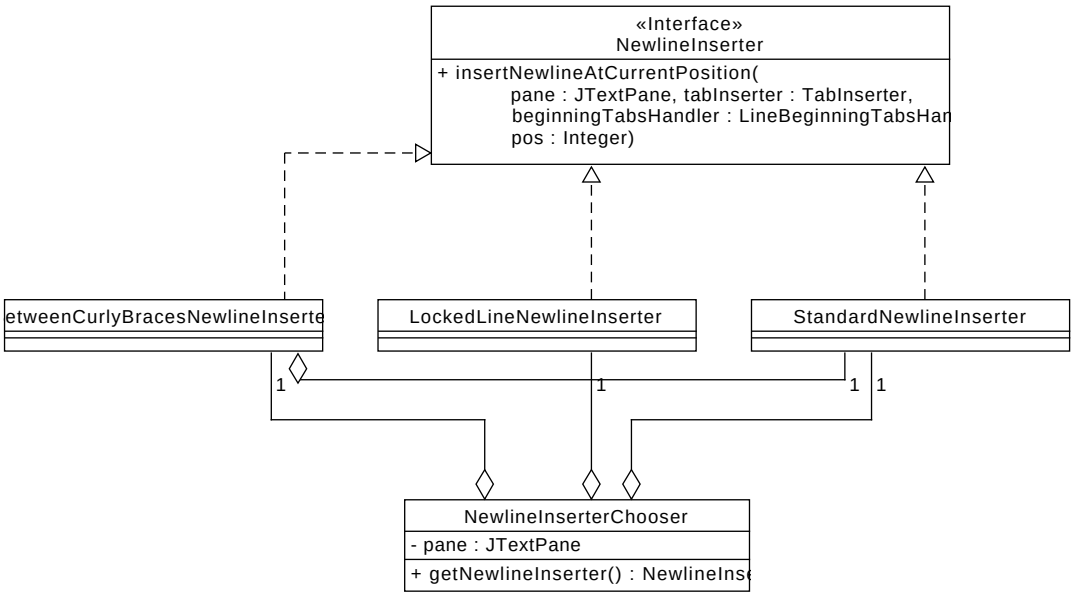
Interface `LineBeginningTabsHandler` und Implementierung `CurlyBracesLineBeginningTabHandler` wurden hinzugefügt. `LineBeginningTabsHandler` berechnet die benötigte Anzahl Tabs zu Beginn einer gegebenen Zeile. `CurlyBracesLineBeginningTabHandler` errechnet dies anhand der Anzahl geöffneter curly-Braces in vorangehenden Zeilen minus die Anzahl schließender curly-Braces am Ende der gegebenen Zeile

3.7.9 UserActions

Es gibt nun spezielle `UserActions` für Kopieren, Ausschneiden und Einfügen. Dies ist nötig um sicherzustellen, dass nicht editierbare Zeilen nicht durch diese Aktionen verändert werden.

3.7.10 NewlineInserter

`NewlineInserter` ist nun ein eigenes Paket. `BetweenCurlyBracesNewlineInserter` verwendet nun `StandardNewlineInserter`.



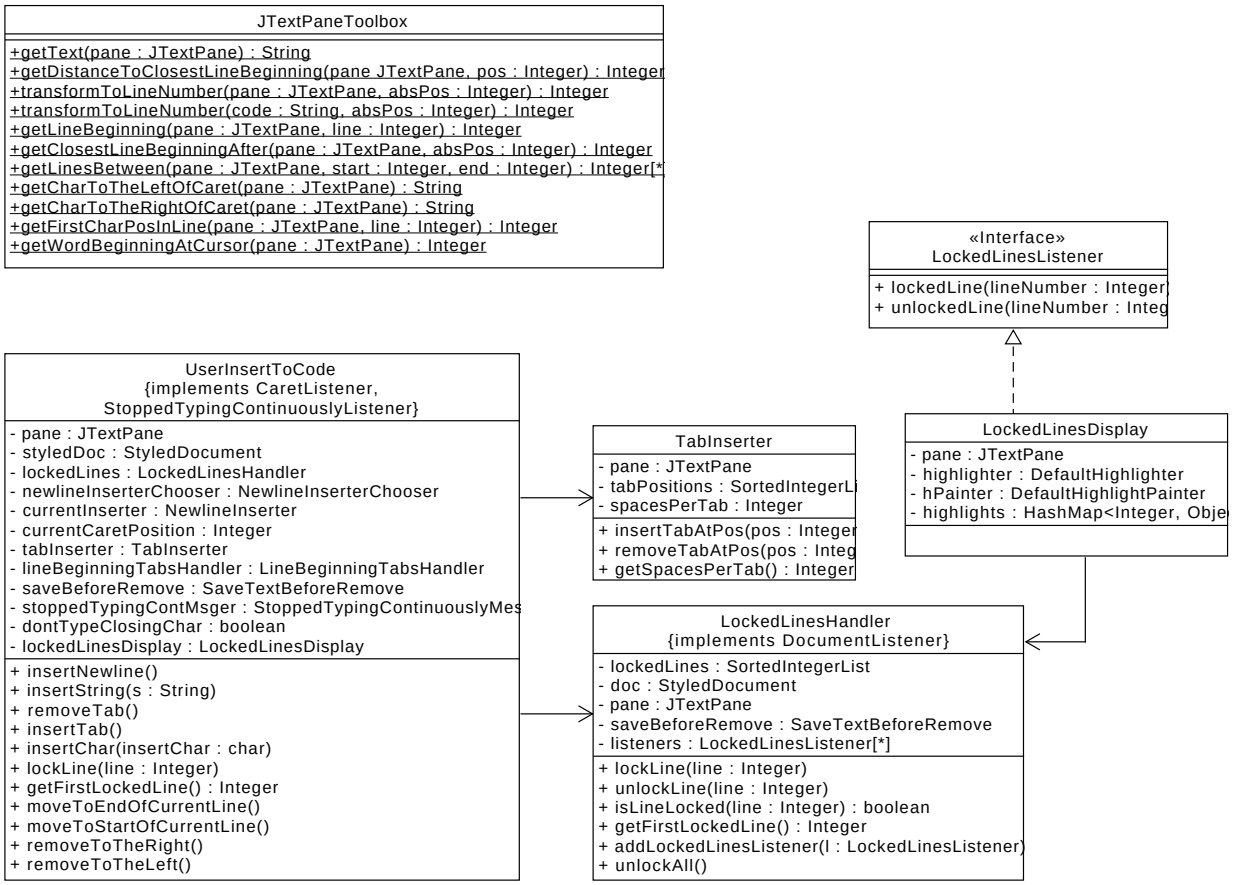
3.7.11 UserInsertToCode

Hinzu kommen folgende Funktionen:

- insertTab fügt an der Momentanen Position ein Tab ein
- insertChar Fügt das gegebene Zeichen an der momentanen Position ein
- getFirstLockedLine gibt die erste nicht editierbare Zeilennummer
- moveToEndOfCurrentLine Bewegt den Caret ans Ende der momentanen Zeile
- moveToStartOfCurrentLine Bewegt den Caret an den Start der momentanen Zeile
- removeToTheRight Entfernt das Zeichen rechts vom Caret
- removeToTheLeft Entfernt das Zeichen links vom Caret

Entfernt wurden:

- msgLockedLinesListeners: wird nun von `LockedLineHandler` übernommen



3.8 BooleanExpEditor

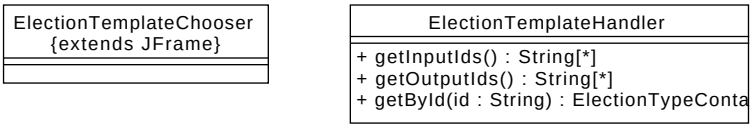
Der `BooleanExpEditor` besitzt jetzt eine Referenz auf die `CElectonDescriptionEditor`-Instanz, da dies zur Fehlerfindung durch den `BooleanExpEditorVariableErrorFinder` nötig ist.
Er bekommt vom Builder nun eine Referenz auf die `PropertyList`-Instanz, da so neu erstellte Eigenschaften in der Liste gespeichert werden können.
(Diagramm folgt)

3.9 CElectionDescriptionEditor

- Die `ChangeElectionType UserAction` und der entsprechende Menüpunkt "Wahlart ändern"wurden entfernt.

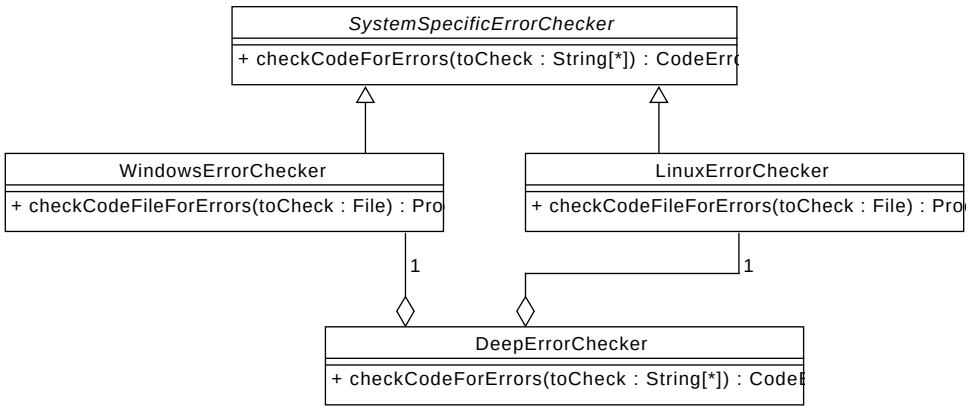
- Das neue Paket `ElectionTemplates` kam hinzu. Dieses enthält folgende Klassen:

- `ElectionTemplateHandler`: Gibt alle Election Input und Output Datentypen und deren ids aus.
- `ElectionTemplateChooser`: Zeigt dem Benutzer einen Dialog, welcher es ermöglicht Input und Result eines neuen Wahlverfahrens zu wählen.



3.9.1 ErrorHandling

Die C-Fehlerfindung findet nun ausschließlich über Aufruf eines externen Compilers statt. Dieser Aufruf geschieht und das Parsen seiner Rückgabe findet in den Klassen `DeepErrorChecker`, `LinuxErrorChecker`, `WindowsErrorChecker` und `SystemSpecificErrorChecker`



statt.

3.10 PropertyChecker

Beim `PropertyChecker` hat sich folgendes im Vergleich zum Entwurf verändert:

- Die Klasse `CBMCResult` besitzt nun die Methode "createFailureExample" samt zugehöriger Untermethoden, welche zur Erstellung des Failurexamples genutzt werden. Deshalb besitzt die Klasse Checker diese Fähigkeit nicht mehr.
- Es wurden drei neue Klassen [1] erstellt, welche beim parsen der Ergebnisse von CBMC mithelfen. Sie werden während des parsens der Rückgabe von CBMC ver-

[1] CBMCResultWrapper/long/singleArray/multiArray

wendet um die Teilergebnisse in Listen zu speichern und sie dann am Ende als Array ausgeben zu können. Dies wurde auf diese Weise implementiert, da am Anfang des parsens nicht bekannt sein kann, wie groß die Datentypen bei der Rückgabe werden würden und es die eigentliche Methode "createFailureExample" deutlich verkürzen konnte.

- Die Klasse `CheckerFactory` besitzt nun zwei neue Methoden:
  "getNewInstance(...)" wird dazu verwendet eine neue Instanz einer `CheckerFactory` zu erstellen, damit die `CheckerFactoryFactory` neue `CheckerFactorys` erstellen kann.
  Außerdem gibt es nun die Methode "getMatchingResult(int amount)", welche die gewünschte Anzahl an checkerspezifischen `Result` Objekten zurückgibt, sodass die `CheckerFactoryFactory` von diesen dann auf Wunsch so viele wie nötig erstellen kann.

3.11 PropertyList

Das Paket `PropertyList` hat sich im Unterpaket `Controller` geändert. Durch die Anbindung an die `highlevel`-Interfaces wurde es nötig, dass das Model der `PropertyList` kein Einzelstück mehr ist. Der Controller benötigt deshalb eine eigene Referenz auf das Model, weil er nicht auf die einzelne Instanz zugreifen kann. Eine zentrale Controllerklasse übernimmt nun die Steuerung, anstatt wie vorgesehen einzelne Klassen.

Die Methoden für Controller und Model wurden außerdem in eigenen Interfaces beschrieben, sodass ein schneller Überblick über die Methoden gegeben ist.

3.11.1 Subpackage Model

Die Klasse `PropertyList` wurde zu `PLModel` umbenannt. Sie hat keine Anbindung nach außerhalb des Pakets mehr. Das Interface `PostAndPrePropertiesDescriptionSource` wird nun vom Controller implementiert.

Die Klasse `PropertyItem` hält nun Daten zum Ergebnis der Analyse (zusätzlich zu `PropertiesDescription` und Teststatus).

Das Interface `PLModelInterface` beschreibt alle Manipulationen der Eigenschaftenliste.

3.11.2 Subpackage Controller

Die zentrale Klasse des Controllers heißt `PropertyList` und implementiert alle nötigen Interfaces nach außen. Das sind:

- `ResultPresenter` (im Entwurf nur vom View implementiert)
- `PostAndPrePropertiesDescriptionSource` (eigentlich im Entwurf im Model implementiert)
- `Runnable` (Klasse startet den View)

- `DisplaysStringsToUser` (zur Entgegennahme der Strings für den View)
- `PLControllerInterface` (alle möglichen Befehle für die `PropertyList`)

Die `Action-` und `ChangeListener` wurden aus dem Controller rausgezogen und direkt im View implementiert.

Die Klassen, die `ListChangeCommand` erweiterten (`ChangeDescription`, `ChangeDescriptionName`, `AddStandardDescription`, `AddNewDescription`, `ChangeTestedStatus`, `Redo`, `Undo`), sind nur noch Methoden im Interface des Controllers (void changeName(PropertyItem prop, String newName); usw.). Das geschah, weil sie so kleine Änderungen an der `PropertyList` bedeuten, dass sie nicht rückgängig gemacht werden müssen. Stattdessen kann der Nutzer neu erstellte Eigenschaften mit dem Klick auf den entsprechenden Button wieder löschen. Lediglich `DeleteDescriptionAction` ist den Undo wert und wurde deshalb in einer eigenen Klasse gekapselt.

Der `SaveBeforeChangeHandler` war im Entwurf noch nicht beschrieben.

3.11.3 Subpackage View

Die Elemente des View sind großteils gleich geblieben. Sie wurden allerdings nicht mit einem GUI-Builder erstellt, weil dann nicht dynamisch Komponenten hinzugefügt werden könnten (Formulare mit fester Anzahl von Komponenten).

Im Entwurf war angedacht, dass das Model die View direkt von Änderungen benachrichtigt. Nun wird das Beobachtermuster benutzt, um dem View Änderungen im Model mitzuteilen. In diesem Falle wird die Liste der Eigenschaften (ArrayList¡ListItem¿) neu aufgebaut.

Neu hinzugekommen ist die Klasse `ResultPresenterWindow`, die die Swing-Klasse `JFrame` erweitert. Dadurch war es nicht möglich, sie als Kinder des Hauptfensters zu deklarieren. Aber durch die Entscheidung gegen eine `JTextPane` konnte das Layout aus dem Pflichtenheft besser dargestellt werden.

3.12 Package Parametereditor

Das Paket `parametereditor` umfasst die Klassen, die das Hauptfenster von BEAST (das Parametereditorfenster) aufbauen und die dortigen Eingaben des Nutzers verarbeiten. In diesem Fenster können Parameter für die Analyse des Wahlverfahrens (Anzahl von Wählern, Kandidaten und Stimmen, sowie maximale Anzahl von Prozessen und maximale Dauer der Analyse) angegeben werden. Außerdem können hier benutzerdefinierte Argumente für CBMC angegeben und die Analyse gestartet und gestoppt werden.

3.12.1 Class ParameterEditor

Die Klasse `ParameterEditor` implementiert jetzt nicht mehr `BEASTCommunicator` aus `highlevel`. Damit wird `ParameterEditor` klar von der Kommunikation zwischen den einzelnen Teilen von BEAST getrennt.

3.12.2 UserActions

Es wurden neue `UserActions` hinzugefügt:

- `NewProjectUserAction`, um neben Laden und Speichern auch das Erstellen eines neuen Projekts zu ermöglichen.
- `OptionsUserAction`, um es dem Nutzer zu ermöglichen, Einstellungen wie etwa die Sprache zu ändern.
- `ShowHideBooleanExpEditor`, `ShowHideCElectionEditor` und `ShowHidePropertyList`, um die anderen GUI-Fenster vom Parametereditor aus öffnen und schließen zu können.

3.12.3 Handlers

Es wurden neue Handler hinzugefügt:

- `ArgumentHandler`, um die benutzerdefinierten Argumente für CBMC zu verarbeiten.
- `SaveBeforeChangeHandler`, um sicherzustellen, dass der Nutzer nicht versehentlich durch Laden oder Erzeugen eines neuen Projekts ein ungespeichertes Projekt verwirft.

3.12.4 AboutWindow

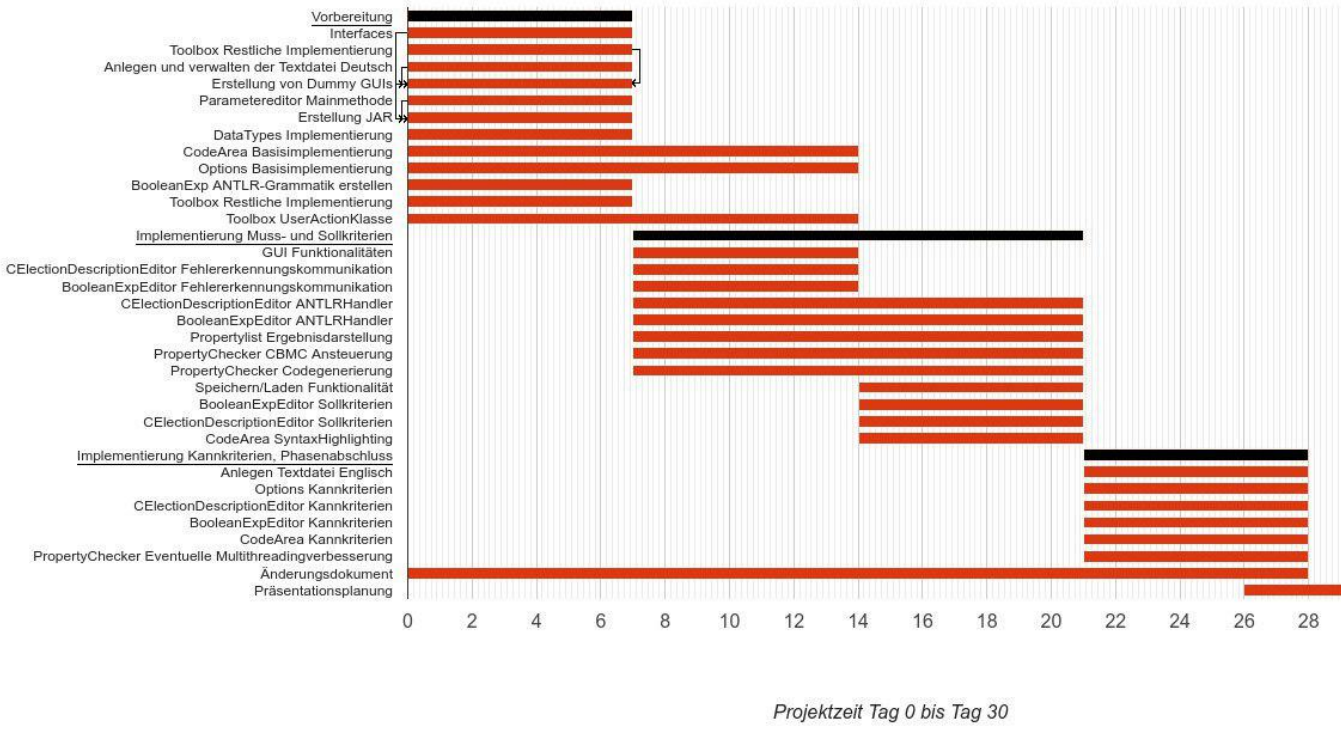
Es wurde die Klasse `AboutWindow` hinzugefügt, die dem Nutzer das Akronym BEAST erklären und die aktuelle Versionsnummer sowie das Build-Datum anzeigt.

3.13 Toolbox

Folgende Klassen kamen hinzu:
- `SortedIntegerList`: Eine einfache Wrapperklasse, welche eine stets sortierte Liste von Integern enthält. Diese wird verwendet von `LockedLinesHandler` und `TabInserter`.
- `RepaintThread`: Implementiert `Runnable`. Bekommt einen `JFrame` für welchen er 60 Mal pro Sekunde die repaint-Funktion aufruft. Dies ist benötigt, da die Frames auf Windows sonst nicht korrekt rendern, falls zuvor ein anderes Fenster vor ihnen war.
- `CCodeHelper`: Diese Klasse enthält Funktionalität um aus den internen Datentypen C Code zu erstellen. Sie wird sowohl von der Codegenerierung als auch dem C-Editor verwendet.
- `ActionIdAndListener`: Eine einfache Wrapper-Klasse welche Zugriff auf einen `ActionListener` und die dazugehörige String-Id bietet.
- `Tuple`: Eine Klasse, welche zwei verschiedene Typen hält.

4 Zeitablauf Implementierungsphase

4.1 Geplanter Ablauf



4.2 Eigentlicher Ablauf

Der ursprüngliche Plan wurde hauptsächlich eingehalten. Ein paar Komplikationen bei den Paketen `highlevel`, `parametereditor` und `propertylist` verzögerten die Fertigstellung einiger Meilensteine.

(Genauere Beschreibung von Aufgabenumverteilung bzw. Ablaufänderung folgt)

(GANTT Diagramm von tatsächlichem Ablauf folgt)