Implementierungsdokument

Hanselmann, Hecht, Klein, Schnell, Stapelbroek, Wohnig

9. Februar 2017

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 Musskriterien

Falls Ein Kriterium nicht implementiert wurde, ausführlich begründen warum nicht.

2.2 Sollkritierien

2.3 Kannkriterien

2.3.1 /FK1130/- Code Completion C-Editor

Das automatische Schließen von Klammern und Anführungszeichen ist implementiert. Vorschlagen von Wörtern ist weder primitiv noch intelligent implementiert, da dies zeitlich noch mindestens 8 Mannstunden in Anspruch nehmen würde, und wir diese Zeit nicht haben.

2.3.2 /FK1140/- Durch den User konfigurierbares Verhalten C-Editor

Die Interfaces um diese Anforderungen zu implementieren existieren zwar, es fehlt allerdings an der Zeit dies jetzt noch fertig zu stellen (6-8 Mannstunden).

2.3.3 /FK2140/ - Code Completion Eigenschafteneditor

Fehlt vollständig da die Zeit von mindestens 8 Mannstunden fehlt.

3 Änderungen am Entwurf

Bspw.
Änderung abstrakte Fabrik 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.
Die Lösung des Problems bietet ein so von uns genannter CentralObjectProvider der, bei seiner eigenen Konstruktion alle anderen Highlevelobjekte baut und diese dann über get zu Verfügung stellt. Somit hat der BEASTCommunicator immer noch eine abstrakte Sicht auf alle Interfaces.

3.1 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. Sinnvoll, da es es genau von den Parametern abhängt.
Jede Instanz der Klasse CBMCCodeGeneration erstellt eine Instanz eines CBMCCodeGenerationVisitor.
Dieser besitzt 2 neue Methode, die einstellen ob er zur Codegenerierung einer Vor- oder Nachbedingungen eines Wahlverfahrens verwendet wird.
(Verändertes Klassendiagramm hier)

3.2 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)

3.3 SaverLoader

PostAndPrePropertiesDescriptionSaverLoader, ElectionDescriptionSaverLoader, ParameterCheckParameterSaverLoader und ProjectSaverLoader implementieren nun das Inter-

face 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, bzw. dies mit !`createFromSaveString` rückgängig macht. Dies verhindert die Erstellung von nicht ladbaren Dateien trotz validen Nutzer-Eingaben.
(Diagramm folgt)

3.4 FileChooser

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

3.5 DataTypes

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

3.6 Codearea

- Errordisplayer ist nun abstrakt. Von den erbenden Klassen müssen Fehlermeldungen generiert werden.
- Klasse SaveTextBeforeRemove wurde hinzugefügt. Diese speichert den Text einer JTextPane sobald Text daraus entfernt wird. Dies ist nötigt da das RemovedUpdate des Styleddocuments keinen Zugriff auf den entfernten Text gewährt. Dieser ist jedoch benötigt um Aktionen rückgängig zu machen. Implementiert wird die Funktionalität durch hören auf Keyevents.
- Klasse TextLineNumber hinzugefügt, welche die Zeilennummer anzeigt. Diese Klasse wurde direkt aus https://tips4java.wordpress.com/2009/05/23/text-component-line-number/ übernommen
- Klasse SquigglePainter wurde hinzugefügt. Diese unterstreicht Text in der JTextPane gezackt. Dies wird verwenden um Fehler im Code anzuzeigen. Übernommen von https://tips4java.wordpress.com/2008/10/28/rectangle-painter/
- Klasse JTextPaneToolbox wurde hinzugefügt. Diese enthält einige statische Methoden welche oft benötigte, aber nicht zusammengehörige Funktionalität für JTextPane liefern. Dazu gehört u.a. das umwandeln absoluter Positionen in Zeilennummern.
- Tabinserter wurde hinzugefügt. Dieser fügt Tabs in Form von Leerzeichen ein.

- Inteface LineBeginningTabsHandler und Implementierung CurlyBracesLineBeginning-TabHandler wurden hinzugefügt. LineBeginningTabsHandler berechnet die benötigte Anzahl Tabs zu Beginn einer gegebenen Zeile. CurlyBracesLineBeginningTabHandler errechnet dies anhand der Anzahl '' in vorangehenden Zeilen minus die Anzahl am ende der gegebenen Zeile
- Es gibt nun spezielle Useractions für Kopieren, Ausschneiden und Einfügen. Dies ist nötig um sicher zu stellen dass nicht editierbare Zeilen nicht verändert werden durch diese Aktionen.

3.6.1 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

3.7 BooleanExpEditor

Besitzt jetzt eine Referenz auf die `CElectonDescriptionEditor`-Instanz, da dies zur Fehlerfindung durch den `BooleanExpEditorVariableErrorFinder` nötig ist
(Diagramm folgt)

3.8 CElectionDescriptionEditor

Die `ChangeElectionType UserAction` und der entsprechende Menüpunkt "Wahlart ändern"wurden entfernt, da es sehr wenig Sinn macht ein bestehendes Wahlverfahren grundsätzlich zu ändern anstatt einfach ein neues zu erstellen und die Implementierung dieser `UserAction` somit unnötig kompliziert wäre.
(Diagramm folgt)
Neues Package ElectionTemplates kam hinzu. Dises 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 PropertyChecker

Beim PropertyChecker haben sich folgende Sachen im Vergleich zum Entwurf verändert:

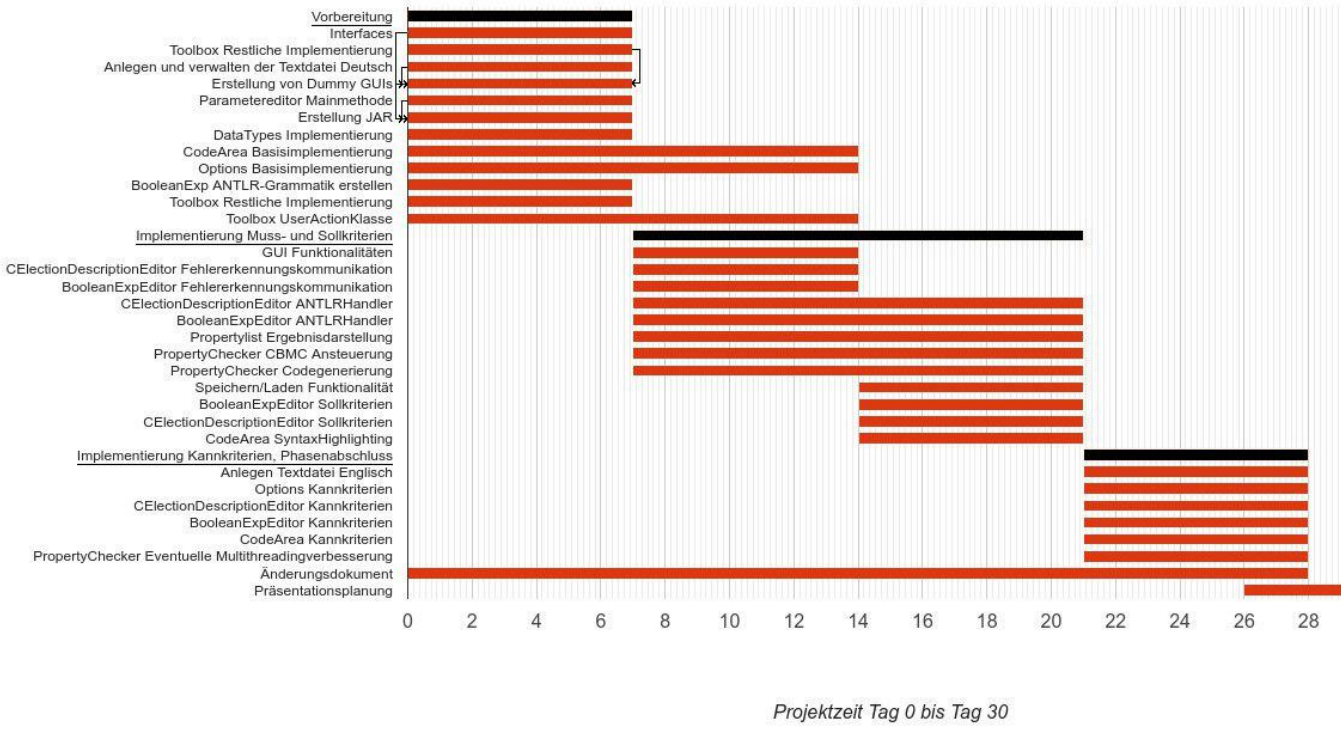
- Die Klasse CBMC_Result besitzt nun die Methode "createFailureExample" samt zugehöriger Untermethoden, welche zur Erstellung des Failurexamples genutzt werden. Deshalb besitzt die Klasse Checker diese Fähigkeit nicht mehr.

- Es wurden drei neue Klassen namens "CBMC_Result_Wrapper_/long/singleArray/multiArray" erstellt. Diese werden während des parsens der Rückgabe von CBMC verwendet 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 die zwei neue Methoden:
  "getNewInstance(...)" wird dazu verwendet eine neue Instanz einer Checkerfactory zu erstellen, damit die CheckerFactoryFactory neue CheckerFactory 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.

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)