

Implementierungsdokument

Hanselmann, Hecht, Klein, Schnell, Stapelbroek, Wohnig

9. Februar 2017

Inhaltsverzeichnis

1	Einleitung	4
1.1	Ziel des Programmes	4
2	Unterschiede zu den im Pflichtenheft gestellten Kriterien	5
2.1	Musskriterien	5
2.2	Sollkriterien	5
2.3	Kannkriterien	5
3	Änderungen am Entwurf	6
3.1	CodeGenerierung	6
3.2	UserActions	6
3.3	SaverLoader	7
3.4	FileChooser	7
3.5	DataTypes	7
3.6	Codearea	7
3.6.1	UserInsertToCode	8
3.7	BooleanExpEditor	8
3.8	C-Editor	8
4	Zeitablauf Implementierungsphase	9
4.1	Geplanter Ablauf	9
4.2	Eigentlicher Ablauf	9

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

Eventuell auch Trennung nach GUI-Fenstern, obwohl das vermutlich weniger sinnvoll ist.

2.1 Musskriterien

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

2.2 Sollkriterien

2.3 Kannkriterien

Falls Ein Kriterium nicht implementiert wurde, begründen warum nicht.

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. MenubarHandler ist nun abstrakt. Dies ist weil nur konkrete Subklassen wissen welche Stringressource von StringResourceLoaderInterface benötigt wird.

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 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 »_ersetzt, bzw. dies mit !createFromSaveString rückgängig macht. Dies verhindert die Möglichkeit zur Erstellung von nicht ladbaren Dateien trotz validen Nutzer-Eingaben.

(Diagramm folgt)

3.4 FileChooser

Diese Klasse kümmert sich um das Laden und Speichern von den

(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.

3.6 Codearea

- Errordisplayer ist nun abstrakt. Von den erbbenden Klassen müssen Fehlermeldungen generiert werden.
- Klasse 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 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.

- Tabinsserter wurde hinzugefügt. Dieser fügt Tabs in Form von Leerzeichen ein.
- Inteface LineBeginningTabsHandler und Implementierung CurlyBracesLineBeginningTabHandler 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

3.7 BooleanExpEditor

Besitzt jetzt eine FileChooser-Instanz.
(Diagramm folgt)

3.8 C-Editor

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

4 Zeitablauf Implementierungsphase

4.1 Geplanter Ablauf

4.2 Eigentlicher Ablauf

Lines of Code

Graphiken dazu, wer was gemacht hat. Vielleicht eine Graphik über die Commits.