Entwurf

Überdurchschnittliche Gruppe

12. Dezember 2016

1 Übersicht

1.1 Einleitung

Dieses Dokument beschreibt den Entwurf des Softwaresystemes welches in unserem Pflichtenheft beschrieben wurde. Es verwendet eine objektorientierte Architektur nach dem Model-View-Controller Prinzip. Zusammengehörige Funktionalität wird in Klassen gekapselt. Module sollen leicht austauschbar sein und klar definierte Schnittstellen bieten. Der Entwurf strebt an, modular und damit leicht veränder- und erweiterbar zu sein.

1.1.1 abstrakteste Sichtweise

Die allgemeinste Sicht auf unser Softwaresystem ist Folgende: Wir benötigen

- Eine “Quelle“ für eine Beschreibung eines Wahlverfahrens
- Eine “Quelle“ für eine Beschreibung der formalen Eigenschaften, welche diese Wahlverfahren erfüllen soll
- Eine Überprüfungsinstanz welche Wahlverfahren und Eigenschaften entgegennimmt das Ergebnis der Überprüfung zurückgibt
- Eine Komponente welche diese Ergebnisse darstellen kann
- Ein Koordinator, welcher die Koordination zwischen obigen Komponenten übernimmt

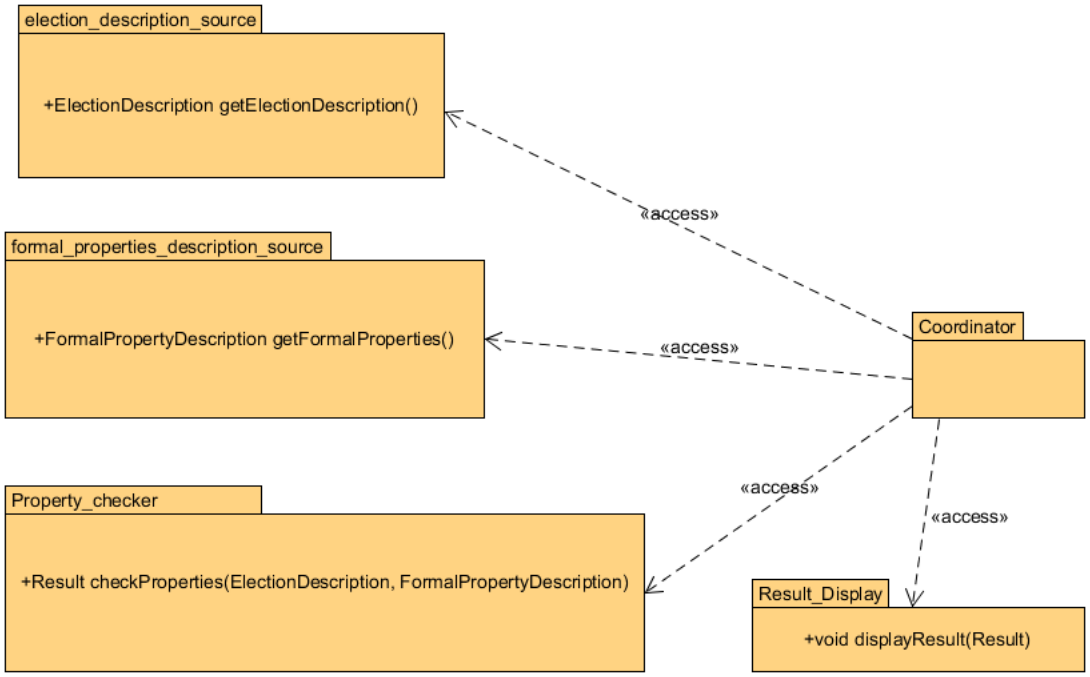


Abbildung 1.1: allgemeinste Sicht auf die Architektur von BEAST

1.1.2 Konkrete Sichtweise

Der Vorteil der abstrakten Beschreibung ist, dass sie quasi unbegrenzte Möglichkeiten liefert und es sehr klar macht, an welchen Stellen man das Programm fundamental ändern kann. Wir werden die Beschreibung des Wahlverfahrens als C-Code implementieren, und die Quelle dafür als C-Editor. Es ist jedoch denkbar, dies in Zukunft durch einen graphischen Editor zu ersetzen, welcher zum Beispiel Flowcharts verwendet. Die formalen Eigenschaften wird in der im Pflichtenheft beschriebenen Syntax gegeben. Als Quelle dient die Eigenschaftenliste welche wiederum den Eigenschafteneditor verwendet. Die Koordination wird von dem Parametereditor übernommen. Zur Überprüfung wird ein bounded model checker verwendet, in unserem Fall speziell der C bounded model checker. Darstellen der Ergebnisse geschieht in der Eigenschaftenliste.

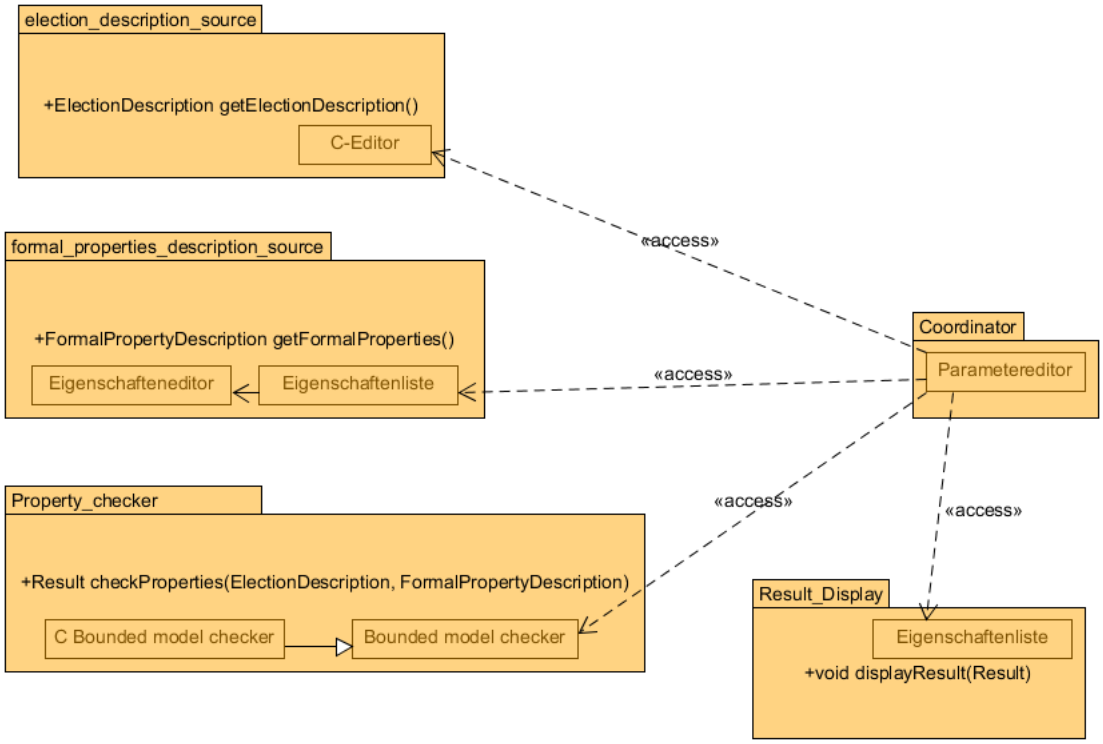


Abbildung 1.2: Konkrete Sicht auf die Architektur von BEAST

1.1.3 Komplikationen im konkreten Fall

Eine Trennung wie sie in 1.1.1 und 1.1.2 dargestellt wird ist in unserem konkreten Fall nicht zu hundert Prozent praktikabel. Dies liegt daran, dass die Komponenten ansonsten einige Funktionalität duplizieren würden. Alle Komponenten benötigen Funktionalität zum Laden und Speichern von Dateien. C-Editor und Eigenschafteneditor haben beide Textpanels welche Code darstellen. Dieser muss in beiden Fällen auf bestimmte Art und Weiße formatiert werden und auf Fehler untersucht werden. Gefundene Fehler müssen in beiden Fällen im Code markiert werden. All diese mehrmals aufkommende Funktionalität wird in eigene Klassen gekapselt werden.

1.2 Überblick über einzelne Komponenten

Hier wird ein Überblick über die einzelnen Komponenten gegeben. Wie bereits erwähnt werden alle Komponenten nach dem Model-View-Controller Prinzip in Funktionalität untergliedert. Dies bewirkt eine Trennung der internen und für den Benutzer sichtbaren Darstellung der verschiedenen Daten. Darauf wird im Folgenden der Schwerpunkt gelegt.

Darauf, wie die Funktionalität zwischen den Komponenten geteilt wird, wird noch nicht eingegangen.

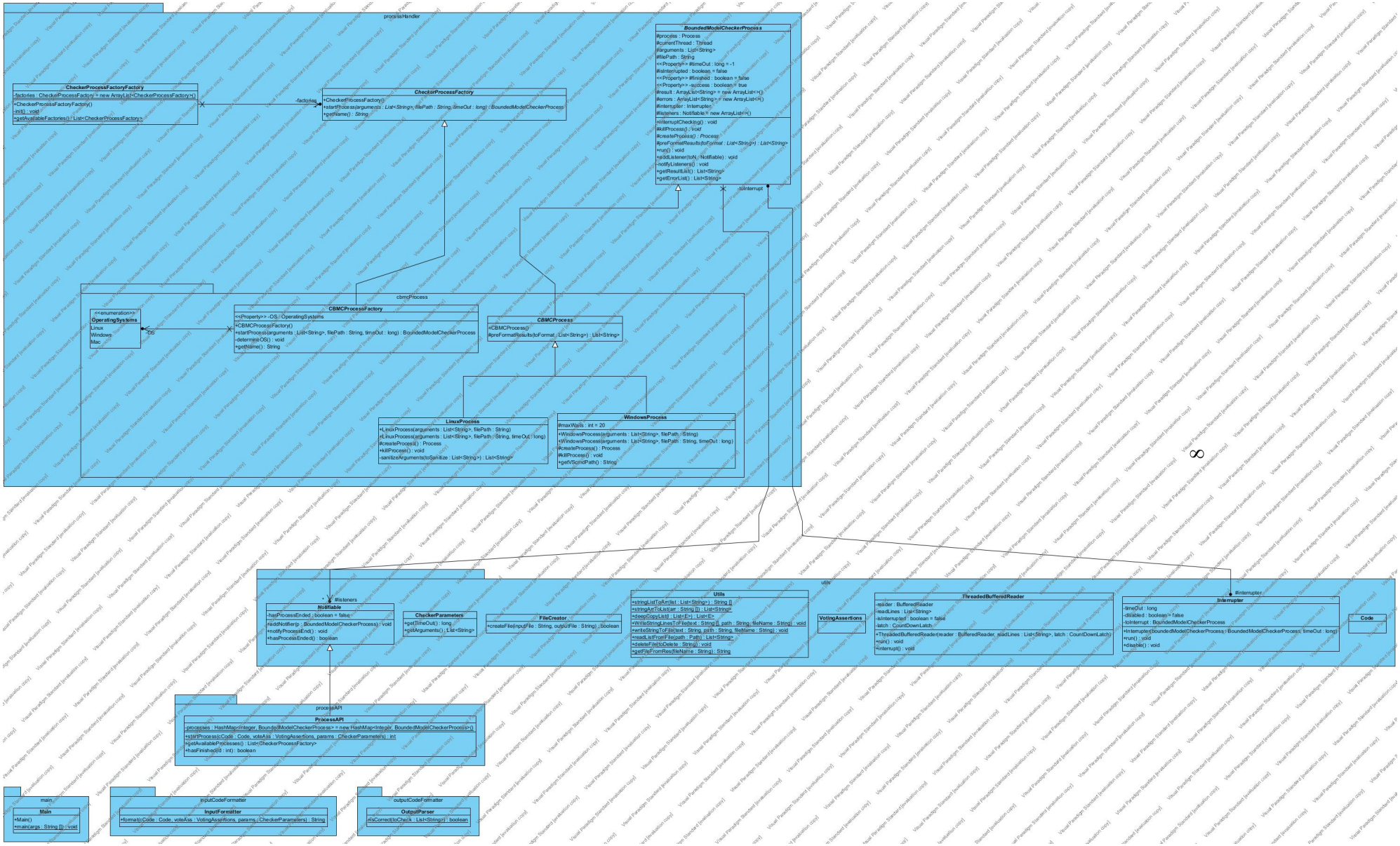
1.2.1 C-Editor

Der C-Editor hat als hauptsächliche Komplikation die Tatsache, dass sowohl die Darstellung des Code (View) als auch die Eingabe des Benutzers (Controller) als auch interne Darstellung des Codes als String (Model) zunächst durch eine Komponente - die JTextPane - übernommen werden. Dies hat den Vorteil, dass viel Funktionalität bereitgestellt wird: Kopieren, Löschen, Ausschneiden von Text. Der Nachteil ist wenig Flexibilität den Text während der Eingabe zu bearbeiten. Daher werden diese Funktionen getrennt bearbeitet werden. Die Keypress - Events werden nicht mehr direkt an die Textpane weitergeleitet sondern zunächst abgefangen. Der Code wird intern in einem speziellen Format abgespeichert. Dieses feuert ein Event wann immer der Code aktualisiert wird oder Fehler im Code festgestellt werden. Dieses Event wird von dem TextPaneDecorator abgefangen, der den Code dann darstellt und sich um das Syntax Highlighting kümmert.

Abbildung 1.3: C-Editor overview

2 Überprüfung der Modelle

Blabla bounded model blabla cbmc


processHandler
CheckerProcessFactoryFactory
-factories : CheckerProcessFactory = new ArrayList<CheckerProcessFactory>()
+CheckerProcessFactoryFactory()
-init() : void
+getAvailableFactories() : List<CheckerProcessFactory>
-factories
CheckerProcessFactory
+CheckerProcessFactory()
+startProcess(arguments : List<String>, filePath : String, timeOut : long) : BoundedModelCheckerProcess
+getName() : String
BoundedModelCheckerProcess
#process : Process
#currentThread : Thread
#arguments : List<String>
#filePath : String
<<Property>> #timeOut : long = -1
#isInterrupted : boolean = false
<<Property>> #finished : boolean = false
<<Property>> -success : boolean = true
#result : ArrayList<String> = new ArrayList<>()
#errors : ArrayList<String> = new ArrayList<>()
#interrupter : Interrupter
#listeners : Notifiable = new ArrayList<>()
+interruptChecking() : void
#killProcess() : void
#createProcess() : Process
#preFormatResultsToFormat : List<String>) : List<String>
+run() : void
+addListener(toN : Notifiable) : void
-notifyListeners() : void
+getResultList() : List<String>
+getErrprList() : List<String>
-toInterrupt
cbmcProcess
<<enumeration>>
OperatingSystems
Linux
Windows
Mac
-OS
CBMCProcessFactory
<<Property>> -OS : OperatingSystems
+CBMCProcessFactory()
+startProcess(arguments : List<String>, filePath : String, timeOut : long) : BoundedModelCheckerProcess
-determineOS() : void
+getName() : String
CBMCProcess
+CBMCProcess()
#preFormatResultsToFormat : List<String>) : List<String>
LinuxProcess
+LinuxProcess(arguments : List<String>, filePath : String)
+LinuxProcess(arguments : List<String>, filePath : String, timeOut : long)
#createProcess() : Process
+killProcess() : void
-sanitizeArguments(toSanitize : List<String>) : List<String>
WindowsProcess
#maxWaits : int = 20
+WindowsProcess(arguments : List<String>, filePath : String)
+WindowsProcess(arguments : List<String>, filePath : String, timeOut : long)
#createProcess() : Process
#killProcess() : void
+getVScmdPath() : String
utils
#listeners
Notifiable
-hasProcessEnded : boolean = false
+addNotifierIp : BoundedModelCheckerProcess) : void
+notifyProcessEnd() : void
+hasProcessEnded() : boolean
CheckerParameters
+get(timeOut) : long
+getArguments() : List<String>
FileCreator
+createFile(inputFile : String, outputFile : String) : boolean
Utils
+stringListToArr(list : List<String>) : String []
+stringArrToList(arr : String []) : List<String>
+deepCopyList(l : List<E>) : List<E>
+WriteStringLinesToFile(text : String[], path : String, fileName : String) : void
+writeStringToFile(text : String, path : String, fileName : String) : void
+readLisfFromFile(path : Path) : List<String>
+deleteFile(toDelete : String) : void
+getFileFromRes(fileName : String) : String
VotingAssertions
ThreadedBufferedReader
-reader : BufferedReader
-readLines : List<String>
-isInterrupted : boolean = false
-latch : CountDownLatch
+ThreadedBufferedReader(reader : BufferedReader, readLines : List<String>, latch : CountDownLatch)
+run() : void
+interrupt() : void
#interrupter
Interrupter
-timeOut : long
-disabled : boolean = false
-toInterrupt : BoundedModelCheckerProcess
+Interrupter(boundedModelCheckerProcess : BoundedModelCheckerProcess, timeOut : long)
+run() : void
+disable() : void
Code
processAPI
ProcessAPI
-processes : HashMap<Integer, BoundedModelCheckerProcess> = new HashMap<Integer, BoundedModelCheckerProcess>()
+startProcess(cCode : Code, voteAss : VotingAssertions, params : CheckerParameters) : int
+getAvailableProcesses() : List<CheckerProcessFactory>
+hasFinished(id : int) : boolean
main
Main
+Main()
+main(args : String []) : void
inputCodeFormatter
InputFormatter
+format(cCode : Code, voteAss : VotingAssertions, params : CheckerParameters) : String
outputCodeFormatter
OutputParser
#isCorrect(toCheck : List<String>) : boolean