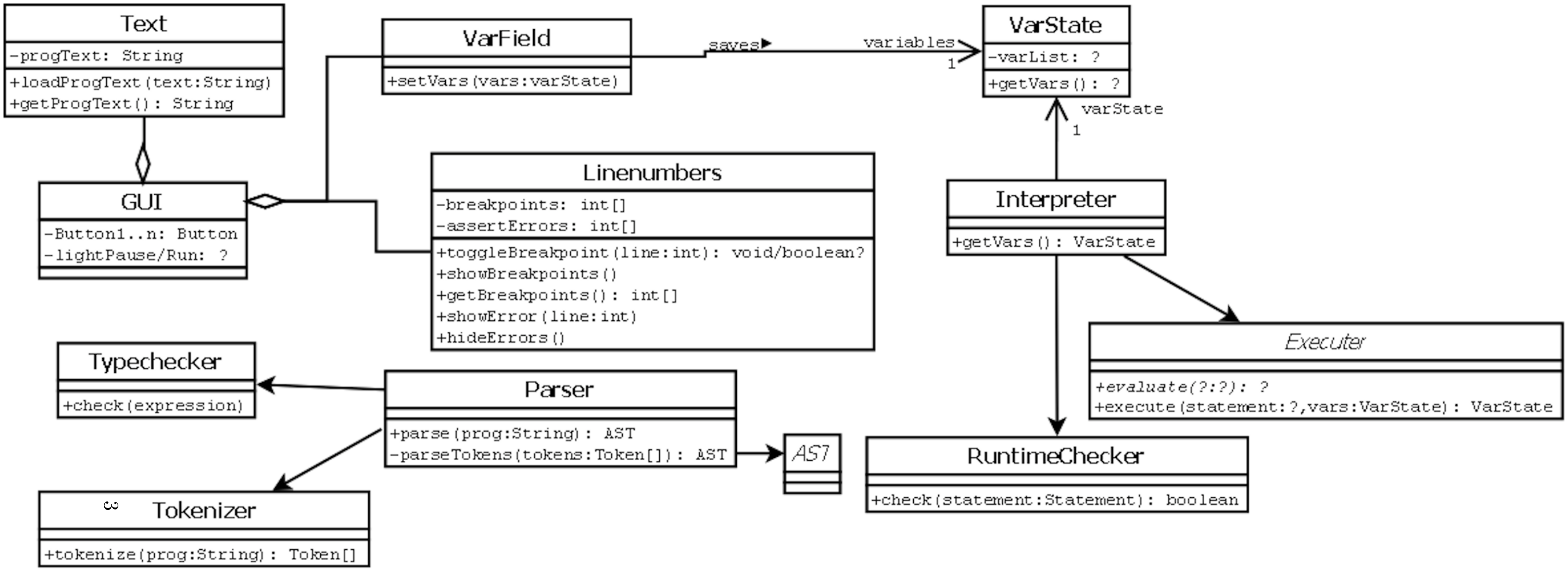
Entwurf

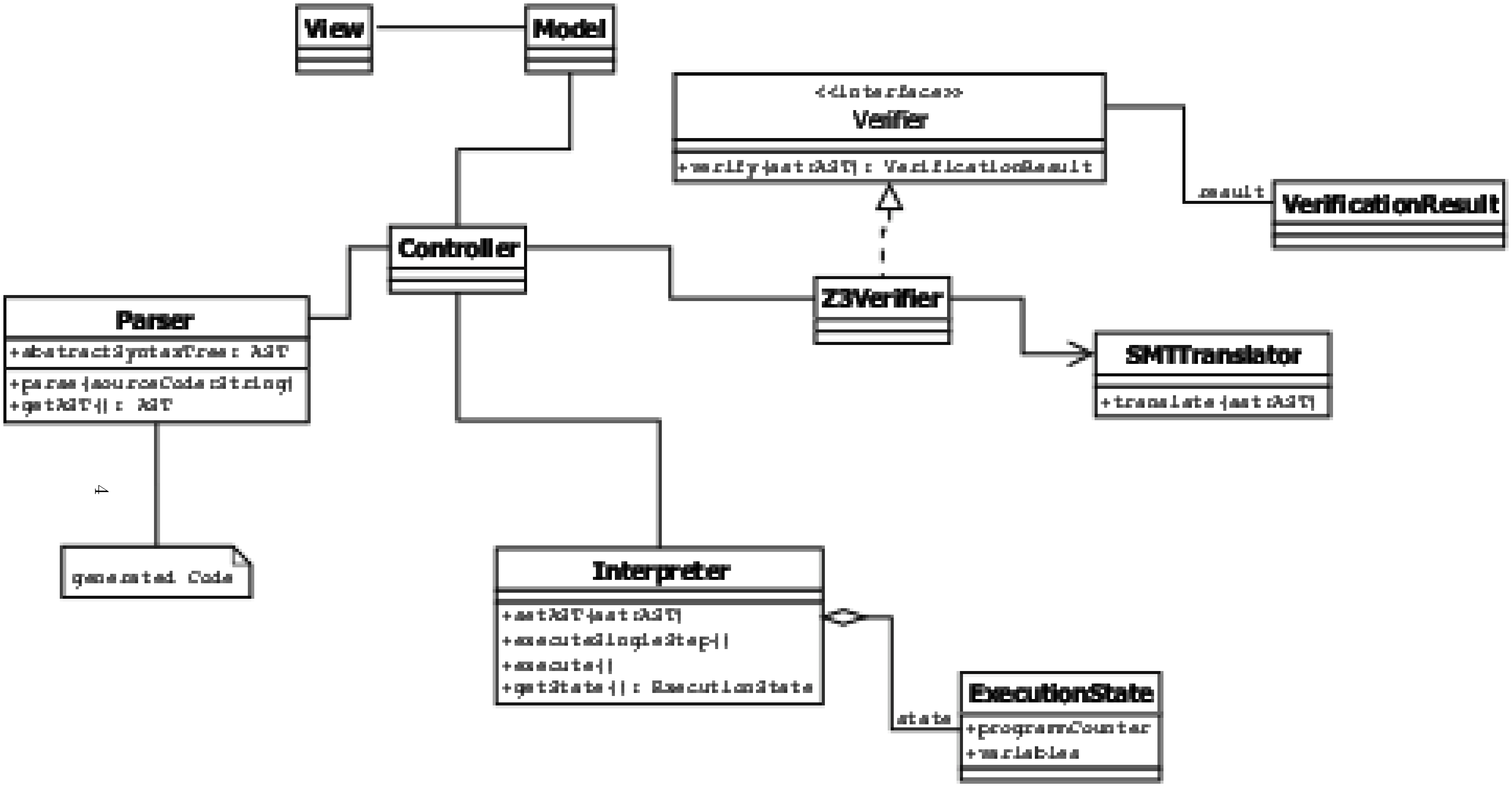
Simon Bischof Jan Haag Adrian Herrmann Lin Jin Tobias Schlumberger
Matthias Schnetz

30. November 2011

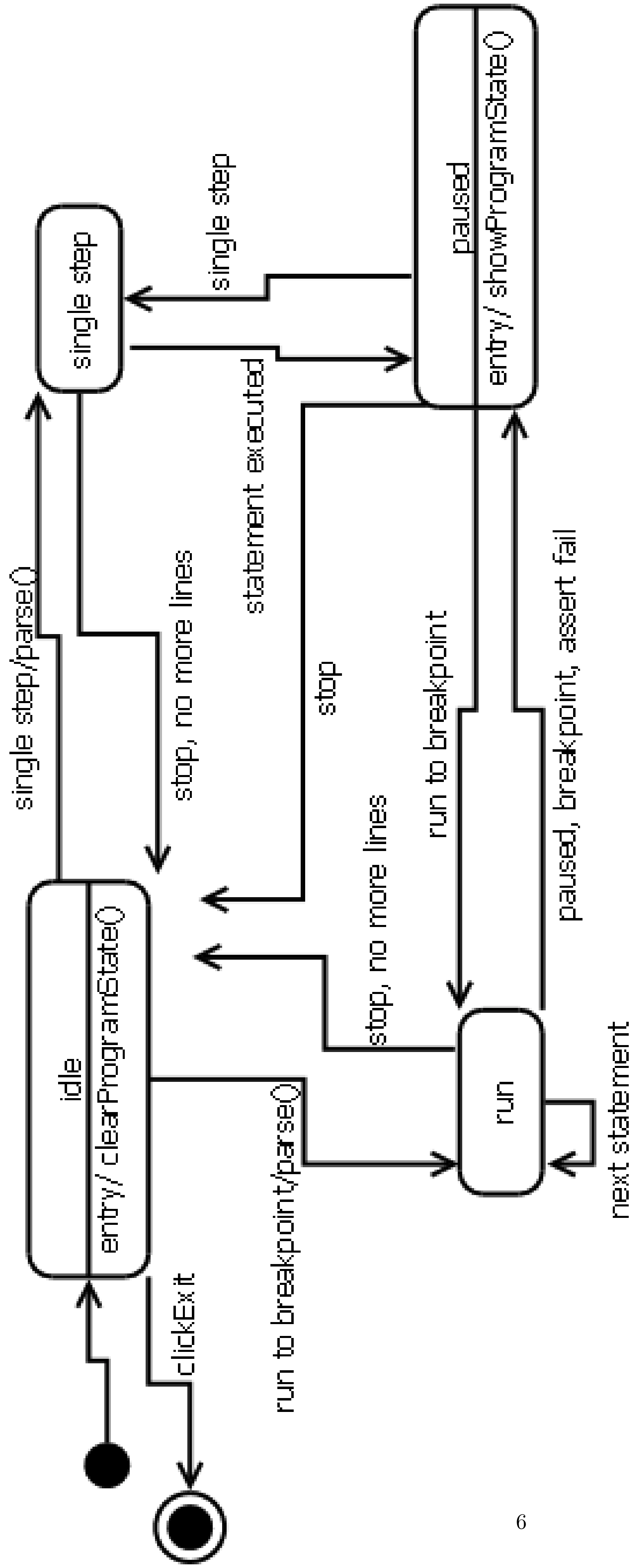
Inhaltsverzeichnis

1 Klassendiagramme

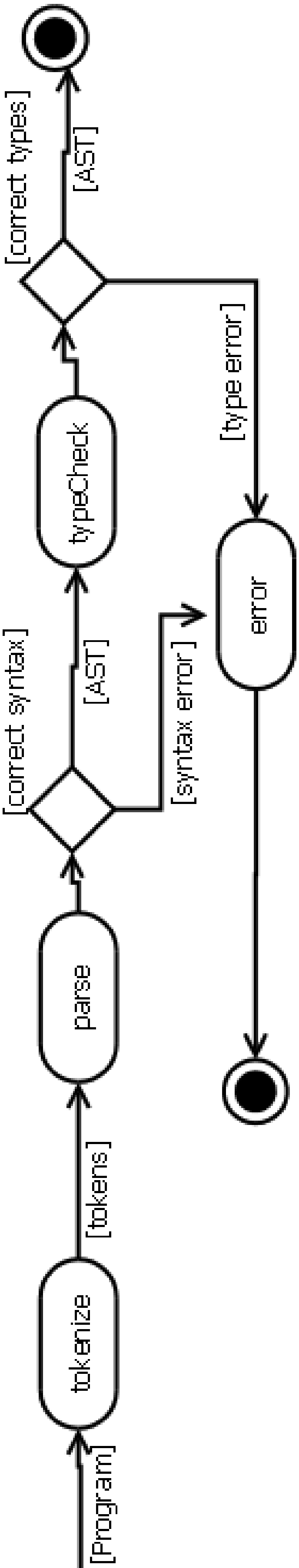

Text
-progText: String
+loadProgText(text:String)
+getProgText(): String
GUI
-Button1..n: Button
-lightPause/Run: ?
VarField
+setVars(vars:varState)
saves
variables
1
VarState
-varList: ?
+getVars(): ?
varState
1
Linenumbers
-breakpoints: int[]
-assertErrors: int[]
+toggleBreakpoint(line:int): void/boolean?
+showBreakpoints()
+getBreakpoints(): int[]
+showError(line:int)
+hideErrors()
Interpreter
+getVars(): VarState
Executer
+evaluate(?:?): ?
+execute(statement:?,vars:VarState): VarState
Typechecker
+check(expression)
Parser
+parse(prog:String): AST
-parseTokens(tokens:Token[]): AST
AST
RuntimeChecker
+check(statement:Statement): boolean
Tokenizer
+tokenize(prog:String): Token[]


View
Model
Controller
Parser
+abstractSyntaxTree : AST
+parse(sourceCode:String)
+getAST() : AST
generated Code
<<interface>>
Verifier
+verify(ast:AST) : VerificationResult
result
VerificationResult
Z3Verifier
SMTTranslator
+translate(ast:AST)
Interpreter
+setAST(ast:AST)
+executeSingleStep()
+execute()
+getState() : ExecutionState
state
ExecutionState
+programCounter
+variables

2 Zustandsdiagramme


idle
entry/ clearProgramState()
single step
paused
entry/ showProgramState()
run
single step/parse()
stop, no more lines
single step
statement executed
stop
stop, no more lines
run to breakpoint
paused, breakpoint, assert fail
run to breakpoint/parse()
next statement
clickExit

3 Aktivtätsdiagramme


[Program]
tokenize
[tokens]
parse
[correct syntax]
[AST]
[syntax error]
typeCheck
[correct types]
[AST]
[type error]
error

4 Syntax der While-Sprache

4.1 Übersicht der Schlüsselwörter und Sonderzeichen

boolean	→ type_specifier
else	→ if_statement
false	→ logical_expression
if	→ if_statement
int	→ type_specifier
return	→ statement
true	→ logical_expression
while	→ while_statement
0..9	→ integer_literal
a..z,A..Z,_	→ identifier
&	→ logical_expression
|	→ logical_expression
!	→ logical_expression
!=	→ testing_expression
==	→ testing_expression
<	→ testing_expression
<=	→ testing_expression
>	→ testing_expression
>=	→ testing_expression
+	→ numeric_expression
-	→ numeric_expression
*	→ numeric_expression
/	→ numeric_expression
%	→ numeric_expression
,	→ arglist → parameter_list → variable_declaration → variable_initializer
;	→ statement → variable_declaration
=	→ variable_declarator
(	→ expression → if_statement → methode_declaration → while_statement
)	→ expression → if_statement → methode_declaration → while_statement
[	→ expression → type
]	→ expression → type
{	→ statement_block → variable_initializer
}	→ statement_block → variable_initializer
#	→ comment

4.2 Startsymbol

```
compilation_unit
```

4.3 Produktionsregeln

```
arglist ::=  expression { "," expression }

comment ::= "#" "... text ..."

compilation_unit ::= { field_declaration }

expression ::= numeric_expression

                | testing_expression
                | literal_expression
                | logical_expression
                | identifier
                | ( ”(ëxpression ”)”)
                | ( expression ( ( ”(”[ arglist ] ”)”)
                                  | ( ”[ëxpression ”]”) ) )


field_declaration ::= ( $[$ comment $]$ ( method_declaration

                                       | variable_declaration ) )


identifier ::= "a..z,A..Z,_" { "a..z,A..Z,_,0..9" }

if_statement ::= "if" "(" expression ")" statement_block [ "else" statement_block ]

integer_literal ::= ( "0..9" { "0..9" } )

literal_expression ::= integer_literal

logical_expression ::= ( "!" expression )

                         | ( expression ( ”&”
                                         | ”
                                         | ( ”&&”)
                                         | ( —”) ) expression )
                         | ”true”
                         | ”false”

method_declaration ::= type identifier "(" [ parameter_list ] ")" ( statement_block )

numeric_expression ::= ( ( "+"

                             | ”) expression )
                         | ( expression ( ”+”
                                          | ”
                                          | ”*”
                                          | ”/”
                                          | ”%”) expression )

parameter ::= type identifier
```

```
parameter_list ::= parameter { "," parameter }

statement ::= variable_declaration

                | ( expression ”;”)
                | ( statemen_block )
                | ( if_statement )
                | ( while_statement )
                | ( ”return”[ expression ] ”;”)
                | ( ”;”)

statement_block ::= "{" { statement } "}"

testing_expression ::= ( expression ( ">"

                                        | ≪”
                                        | ≫=”
                                        | ≪=”
                                        | -=”
                                        | ”!=”) expression )

type ::= type_specifier { "[" "]" }

type_specifier ::= "boolean"

                    | ïnt”

variable_declaration ::= type variable_declarator { "," variable_declarator } ";"

variable_declarator ::= identifier [ "=" variable_initializer ]

variable_initializer ::= expression

                          | ( {”[ variable_initializer { ”,”variable_initializer } ] ”}”)

while_statement ::= "while" "(" expression ")" statement_block
```