Studienarbeit
Objective C und iOs Programmierung
ExpressionCalc

Christoph Cwelich Inf5

18. Dezember 2012

Inhaltsverzeichnis

1. Kurzanleitung

Teil I.

Dokumentation

2. Einleitung

Die vorliegende Dokumentation beschreibt den Programmablauf sowie die Klassenstruktur. Bei der App handelt es sich um einen Taschenrechner, der komplexe Ausdrücke nach den gängigen Regeln berechnen kann und diese in einer ansehnlichen mathematischen Form (mathml) auf das Display ausgibt. Ein solcher Ausdruck kann mit einer Grammatik beschrieben werden.

Folgende erweiterte Backus-Naur-Form bildet die Grundlage für das vorliegende Programm:

```
// basic definitions
<Statement>    ::= 'var' <Whitespace> <Variable> '=' ] <Expression>
<Expression>   ::=  [ <opSum> ] <Product> ( <opSum> <Product> )*
<Product>      ::= <Power> ( <opProduct> <Power> )*
<Power>        ::= <Literal> ( <opExponent> <Literal> )*
<Literal>      ::= <Number> | <Identifier>
                | '(' <Expression ')' | <Function>
<Identifier>   ::= <Constant> | <Variable>
<Constant>     ::= <pi> | <e>
<Variable>     ::= ANS | X0 | ... |  X9
<Function>     ::= <FunctionName> <Literal>
<FunctionName> ::= 'cos' | 'sin' | 'tan'
                 | 'acos' | 'asin' | 'atan'
        | <sqrt> | 'exp' | 'ln'

//tokentype: operator
<opSum>        ::=  '+' | '-'
<opProduct>    ::= '*' | '/'
<opExponent>   ::= '^'
//tokentype: number
<Number>       ::= <Decimal> [ 'E' ( <Digit> )+ ]
<Decimal>      ::= ( ( <Digit> )+ [ '.' ( <Digit> )* ] )
                  | ( ( <Digit> )* '.' ( <Digit> )+ )
<Digit>        ::= '0' | '1'| ... | '9'
//special
<Whitespace>   ::= '␣'
```

Die Grammatik gibt dabei an, wie der Syntaxbaum im Rechner aufgebaut wird. Gleichzeitig wird damit die Bindungsstärke der Operatoren geregelt, nämlich von außen die Addition/Subtraktion (schwache Bindung) bis nach innen der Exponentialoperator (starke Bindung).

2.1. Umsetzung für iOs

Normalerweise dient eine Grammatik dazu, Text von der Tastatur oder aus Textfiles zu parsen. Für den Taschenrechner auf iOs-Basis bringt dies aber Nachteile mit sich:

- Es wird nur eine beschränkte Anzahl von Zeichen und Symbolen benötigt, wobei die benötigten Symbole überwiegend mathematisch sind
- Beim Navigieren über den Ausdruck (und Löschen von Elementen) gibt es Schwierigkeiten, schließlich hat mein keinen Texteditor vor sich.
- Eine "schöne" mathematische Darstellung ist kaum möglich, es müsste eine Art Programmiersprachensyntax verwendet werden, oder die Ausgabe im getrennten Display erfolgen (Platzmangel)
- Es können durch falsche Eingaben viele Fehler auftreten → Benutzerunfreundliches debugging.

Stattdessen wird folgende Vorgehensweise forciert:

- Die Symbole aus der Grammatik werden in eine Klassenstruktur umgewandelt.
- Der "Objekt-Syntaxbaum" baut sich dann praktisch selbst auf.
- Die "Zeichen" die hinzugefügt werden können, werden direkt als Methode an den Syntaxelementen definiert. Jedes Syntaxelement kann dann abhängig von seinem Kontext entscheiden, wie es auf die Eingabe reagieren soll.
- die Eingabemethoden werden dann über den ViewController mit Buttons in der View verknüpft.
- Jedes Element erzeugt seine Ausgabe selbst als mathml. Damit wird der Objektbaum in eine HTML/XML-Struktur transformiert. Das damit aufgebaute HTML kann dann in einer UIWebView ausgegeben werden.

Dieses Vorgehen bietet folgende Vorteile:

- Eingabefehler werden minimiert, da die (Syntax-) Elemente nur definierte Eingaben akzeptieren. Unpassendes kann einfach ignoriert werden.
- Eingabe erfolgt nur über Buttons, die Ausgabe "live" über HTML.
- Änderungen und die Navigation kann direkt über die Syntaxbaumelemente erfolgen, statt über einzelne Zeichen.

3. Programmablauf

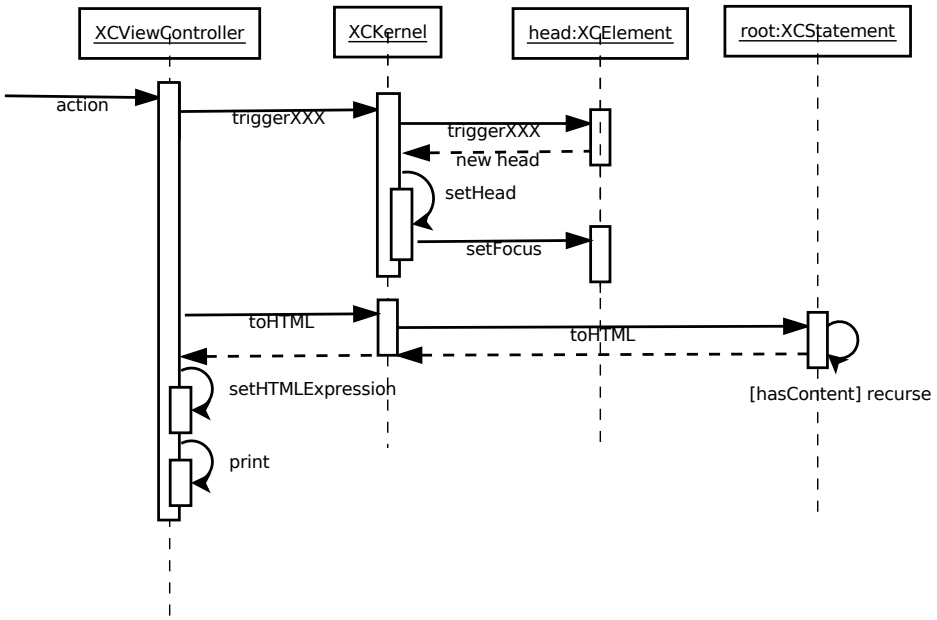

Abbildung 3.1.: Sequenzdiagramm: Trigger Allgemein

In diesem Kapitel soll der der allgemeine Programmablauf dargestellt werden. Einzelne Klassen werden hier schon mal kurz angesprochen, aber erst im nächsten Kapitel eingehender behandelt.

3.1. Eingabe von Symbolen

Dieser Abschnitt zeigt den generellen Ablauf bei Eingaben durch den Benutzer. Anschließend wird noch ein ausgewähltes Beispiel vorgestellt werden.

3.1.1. Allgemein

Diagramm 3 zeigt den allgemeinen Ablauf, wenn der Benutzer einen Button auf der View betätigt.

1. Durch Betätigung des Buttons wird im ViewController eine Action ausgelöst.

2. Im VC wird eine entsprechende `trigger`-Nachricht an den Kernel geschickt. Triggernachrichten haben immer die Form:

   ```
   hasTriggers triggerXXX [ args ]
   ```

   - `hasTriggers` ist dabei der Rückgabewert, also ein Element welches wieder Trigger besitzt. Tatsächlich implementieren alle Elemente des Syntaxbaums sowie der Kernel das Protocol XCHasTriggers.
   - `triggerXXX` ist der Methodenname, wobei "trigger" immer als Präfix steht und XXX das eigentliche "Zeichen" ist, welches eingegeben werden soll.
   - `[ args ]` bei manchen Triggermethoden müssen noch Argumente mitgegeben werden.

3. Der Kernel gibt den Triggeraufruf an das Headelement weiter. Head ist praktisch das Element im Syntaxbaum, auf dem der Focus steht. Dieses wird dann im HTML-output hervorgehoben.

4. Das Element entscheidet nun wie mit dem Triggeraufruf zu verfahren ist. Dabei gibt es in der Regel folgende Möglichkeiten:
   - Je nach Element und Kontext können dabei neue Elemente erzeugt und im Baum eingehängt werden. Dabei wird auch meist eine neuer Platzhalter erzeugt, der zurückgegeben wird.
   - Das Element gibt den Aufruf an sein Elternelement im Baum weiter, z.B. bei Operatoren
   - Im Zweifelsfall gibt sich das Element selbst zurück, ignoriert also den Knopfdruck des Benutzers.

5. Der Kernel setzt nun den head neu, löscht den Focus (Bitflag) vom ehemaligen Head und setzt den Focus beim neuen Head

6. Anschließend ruft der der ViewController die `toHTML`-Methode im Kernel auf, welcher diesen Aufruf an das Root-element des Baums weitergibt. Das HTML wird nun rekursiv erzeugt und nach Fertigstellung in ein Template eingesetzt und in einer UIWebView dargestellt.

3.1.2. Beispiel XCSpacer-triggerNum

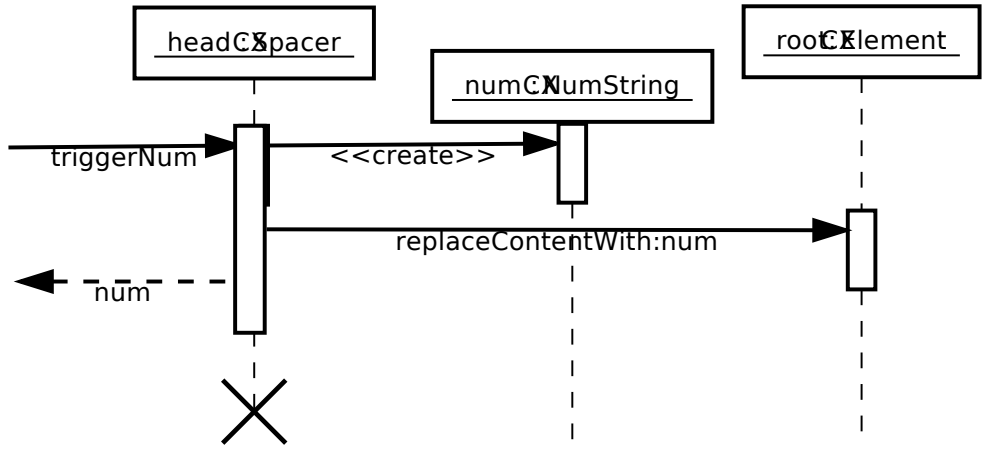

Abbildung 3.2.: Sequenzdiagrammfragment: triggerNum auf XCSpacer

Der Interne Ablauf im Baum bei einem Triggeraufruf soll nun anhand eines Beispiels gezeigt werden. Das Diagrammfragment 3.1.2 zeigt den Aufruf von triggerNum auf einem XCSpacer-objekt.

XCSpacer ist das Analogon zum Literal in der Grammatik und fungiert als Platzhalter für Zahlen, Funktionen, Konstanten, Variablen oder geschachtelte Ausdrücke. Der Spacer wird immer dann eingesetzt, wenn auf die Eingabe von Literale "gewartet" wird, also nach der Eingabe von Operatoren oder am Anfang, wenn ein leerer Ausdruck dasteht.

Nun zum Beispiel:

1. die triggerNum-Methode von XCSpacer erzeugt eine neue `XCNumString`-Instanz. Dies ist ein Stringbuffer für Dezimalzahlen.

2. XCSpacer ruft sein Elternelement auf und gibt ihm die Nachricht den aktuellen Inhalt (also sich selbst) mit der `XCNumString`-instanz zu ersetzen.

3. zuletzt wird das neue XCNumString-objekt zurückgegeben und wie oben beschrieben weiterverfahren (Head neu gesetzt, HTML ausgegeben...).

4. Nachdem der Spacer weder im Kernel noch im Syntaxbaum referenziert ist, wird er zerstört.

3.2. Berechnung von Ausdrücken

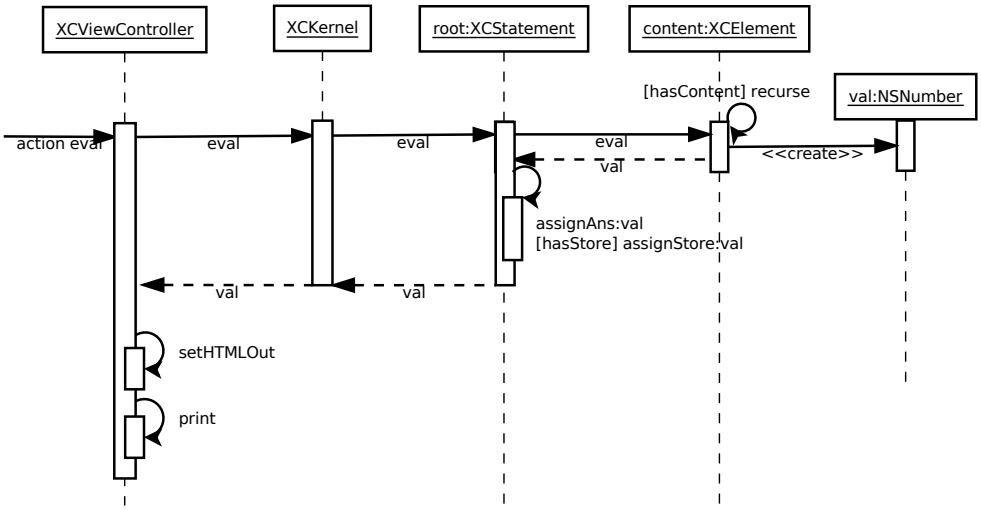

Abbildung 3.3.: Sequenzdiagramm: eval

Die Berechnung von Ausdrücken anhand des erzeugten Syntaxbaums ist nun denkbar einfach:

1. Nach dem Drücken des "="-Buttons wird nun den Syntaxbaum absteigend die `eval`-Methode an allen Elementen aufgerufen. Dabei wird Bottom-Up der Ausdruck ausgewertet (evtl. Fehlerflags gesetzt) und die Zwischenergebnisse in Form vonNSNumber objekten an den Aufrufer zurückgegeben.

2. im obersten Element des Baums (immer XCStatement) wird schließlich noch der Wert in der ANS-Variable gesetzt. Eine eventuell mit "STO" gesetzte Variable wird auch noch überschrieben.

3. das zurückgegebene Ergebnis wird im ViewController in der WebView dargestellt.

4. Klassen

In diesem Kapitel wird näher auf die wichtigsten Klassen und die Architektur eingegangen. Das Programm ist aufgeteilt in den XCKernel-Bereich, welcher den inner Zustand des Taschenrechners verwaltet und XCViewController-Bereich, in dem die View gesteuert wird. Der XCViewController benutzt dabei den Kernel.

4.1. XCViewController

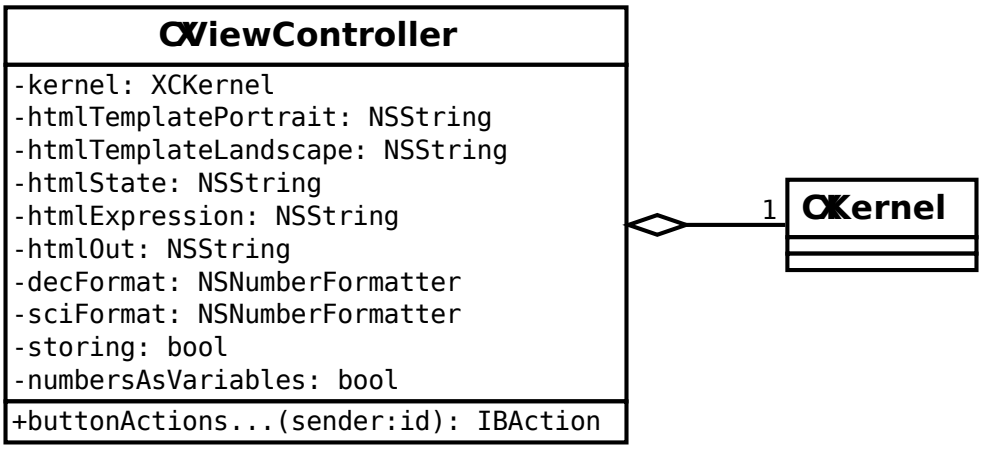

Abbildung 4.1.: Klassendiagramm

Der ViewController verwaltet seinen Zustand den er zum Darstellen des Displays und zum Ansteuern der Trigger im Kernel benötigt. Außerdem wird geregelt, wie die Tasten dargestellt werden sollen (z.B. Zahlen $\Longleftrightarrow$ Variablen, DEG $\Longleftrightarrow$ RAD). Der Kernel (s.u.) übernimmt die eigentliche Aufgabe der Eingabe neuer Zeichen und der Berechnung.

4.2. Kernel

(siehe auch extra Klassendiagramm)

4.2.1. Schnittstellen (Protocols)

Viele Schnittstellen von häufig verwendeten Methoden(-gruppen) werden in folgenden Protocolszusammengefasst:

XCHasHtmlOutput Die Implementierung kann ihren Inhalt in (mathml)-HTML ausgeben.

XCHasTriggers Die Implementierung hat Trigger (siehe oben)

XCEvaluable Die Implementierung kann ihren Inhalt als NSNumber (berechnen und) zurückgeben.

4.2.2. XCKernel

Der Kernel verwaltet das Root-Element (XCStatement) des Syntaxtrees und das Head-Element, auf welchem der Focus steht. Außerdem enthält er einen Buffer (XCStatementBuffer), der die letzten Eingaben in einem Ringpuffer (XCCircBuf) abrufbar verwaltet.

4.2.3. Hilfsklassen

NSNumber+XCNumber Hierbei handelt es sich um eine Obj C Category, welche NSNumber um die unten benötigten Operatoren und einige Zustandsabfragen erweitert. Außerdem wird eine interne Typ und Überlaufskontrolle durchgeführt. Dabei werden eingegebene Integerwerte erhalten und im Fall von Überläufen in double konvertiert.

Anmerkung: Eine Erkennung von ganzzahligen double-Werten ist noch nicht enthalten.

XCGlobal Dies ist zum einen eine Art global.h, welche globale symbolische Konstanten enthält, aber auch eine Singleton-Klasse, welche globale Werte verwaltet (derzeit nur die Grad/Rad Winkeleinstellung)

4.2.4. Syntax-Tree

Die Oberklasse für alle Elemente des Syntaxbaums ist die Klasse XCElement. Diese ist konform zu allen oben genannten Protokollen und NSCopying [1]. Für alle Methoden werden Standardimplementierung angeboten. Unterklassen müssen dann nur ihr eigenes Verhalten implementieren.

Desweiteren ist ein Bitflag-struct zur Speicherung des inneren Zustands nebst Hilfsmethoden enthalten, die von den Unterklassen zur Erstellung der HTMLs benötigt werden.

Außerdem ist ein root-Element vorhanden, damit ist [2] das Elternelement der Instanz gemeint. Mittels content-Methode und root ist der Syntaxbaum damit in beide Richtungen navigierbar.

Eine Ableitung von XCElement ist der bereits besprochene XCSpacer. Alle weiteren Unterklassen von XCElement werden nochmals in drei Kategorien eingeteilt:

[1] Wird beim Wiederholen aus dem Statementbuffer benötigt, damit beim Ändern keine alten Ausdrücke überschrieben werden

[2] etwas missverständlich

4.2.4.1. XCComplexElement

Die komplexen Elemente enthalten mindestens ein (oder min. zwei) weitere Elemente. Diese werden gekapselt von einer XCComplexElementContent-instanz verwaltet, welche auch den Zeiger auf das aktuelle Element beinhaltet.

Unterklassen sind:

XCExpr Die Summe. Negative Werte werden in XCNegate verpackt. Die Klasse bildet gleichzeitig die Implementierung für "Ausdrücke".

XCExpo Das Produkt. Divisionen werden in Form von XCInvert dargestellt.

XCProd Der Exponetialoperator, alle Operatoren werden der Form nach berechnet:

$$b_0^{(b_1^{(\ldots^{b_n})})}$$

4.2.4.2. XCSimpleElement

Die simplen Elemente enthalten immer nur ein weiteres Element.

Unterklassen sind:

XCStatement Analog zur Grammatik bildet das Statement immer das Root-Element eines Syntaxbaums. Es enthält einen Ausdruck, der berechnet werden kann sowie optional eine explizit zugewiesene Variable, welcher nach der Berechnung das Ergebnis eingesetzt wird. Implizit wird immer das Ergebnis der ANS-Variable zugewiesen.

XCFunction Die Funktion besteht aus einem Name und einem Ausdruck, dessen Ergebnis in die Funktion eingesetzt wird. Die Klasse verwaltet ein Dictionary von XCFuncAlg-instanzen (XCTrigoFuncAlg für Winkelfunktionen). Diese enthalten die eigentliche Funktion als Functionpointer. XCSqrt ist eine Spezialisierung von XCFunction, es wird lediglich die toHTML-Methode für Wurzeldarstellung verändert.

XCInvert Invertiert einen Wert. Wird unter anderem von XCProd, da diese Klasse nur multiplizieren kann.

XCNegate Der enthaltene Wert wird negiert. Analog zu XCInvert von XCExpr zur Darstellung von Minus benutzt.

4.2.4.3. XCTerminalElement

Wie der Name schon andeutet handelt es sich dabei um die Implementierung der Terminalsymboleaus der Grammatik. Naturgemäß enthalten diese keine weiteren Syntaxelemente und bilden die Blätterdes Syntaxbaums.

Unterklassen sind:

XCIdentifier enthält ein XCStorage-objekt, also eine Variable (XCVariable) oder eine Konstante (XCConstant).

XCNumString Diese Klasse hat lediglich einen String-Buffer mit dem Dezimalzahlen erstellt werden können (also Digits und Komma). Die Klasse ist so intelligent um falsche Eingaben, wie mehrfach Komma abzufangen. Beim Aufruf von eval wird der Buffer geparst und in ein NSNumber umgewandelt

5. Tests