Ausarbeitung : Objective-C

Günther Noack

18. März 2005

Inhaltsverzeichnis

1 Motivation zur Entwicklung der Sprache

1.1 Geschichtliches

Objective-C wurde in den frühen 80er Jahren des 20. Jahrhunderts von *Brad Cox* für die Firma *Stepstone* geschaffen. Es handelt sich bei der Sprache um eine objektorientierte Erweiterung zur Programmiersprache C, welche für ihre Geschwindigkeit und Hardwarenähe bekannt ist.

1.1.1 Geschichtliche Verbindungen zu NeXTStep und Apple

Da die Geschichte der Programmiersprache Objective-C sehr stark mit der Geschichte der NeXTStep- und Apple-Betriebssysteme verwoben ist, werden im folgenden die wichtigsten Punkte in der Entwicklung dieser Systeme aufgeführt.

Im Jahre 1985 verließ *Steve Jobs* den Computer- und Betriebssystemhersteller *Apple*, um die Firma *NeXT* zu gründen. Ziel der Firmenneugründung war die Entwicklung von *NEXTSTEP*, einem UNIX-basierten Betriebssystem mit *grafischer Benutzeroberfläche*, dessen API in Objective-C geschrieben war. Mit der durchgängigen Verwendung objektorientierter Konzepte und dem Vorhandensein eines fortschrittlichen *Interface-Builders* war NEXTSTEP vielen anderen Systemen dieser Zeit weit voraus.

1994 wird von *NeXT Computer, Inc* und *Sun* der *OPENSTEP* Standard verabschiedet, welcher eine betriebssystemunabhängige Schnittstelle zur Objective-C Programmierung definiert, welche auf der NEXTSTEP-API basiert.

Im Frühjahr 1997 wird NeXT von Apple aufgekauft, da das neue Apple-Betriebssystem *MacOS-X* auf NEXTSTEP basieren soll. Die meisten Objective-C Programme werden heute für die Apple-Plattform geschrieben.

1.2 Ziele von Objective-C

Die Ziele der Entwicklung von Objective-C durch Brad Cox sind:

- Nahtlose Einbindung in bestehende Softwaresysteme
- Wiederverwendbarkeit
- „Evolution statt Revolution"

Das Motto „*Evolution statt Revolution*" umschreibt hierbei das Prinzip, auf bewährte Technologien aufzusetzen, anstatt das Rad neu zu erfinden. Im Detail wird dies in [2] erläutert.

Speziell entwickelte sich daraus die Idee, die neue Sprache auf einer bereits existierenden und bewährten Sprache aufsetzen zu lassen - nämlich der Sprache C. Auf diese Weise konnten die für C vorhandenen Compiler sogar unverändert für die Übersetzung von Objective-C-Programmen verwendet werden, indem der Objective-C-Code vorerst von einem einfachen Umwandlungsprogramm in C-Code umgeschrieben wurde.

Die Wiederverwendbarkeit ist eine der grundlegenden Ideen hinter der objektorientierten Programmierung. Durch die Gruppierung von Methoden und Daten in *Objekte* kann die logische Struktur von Programmen besser ausgedrückt werden. Programmierer können sich so leichter in fremde Programmteile einarbeiten.

Da Objective-C direkt von C abgeleitet ist, kann man Objective-C-Programme direkt mit vorhandenem C-Code mischen und somit bereits vorhandene Bibliotheken und Programmteile weiterhin verwenden. Auch ist der Lernaufwand für einen C-Programmierer sehr gering und die vorhandenen Compiler sehr ausgereift.

1.3 Objective-C und Smalltalk

Als Vorbild für das objektorientierte Konzept von Objective-C diente die Sprache Smalltalk, welche am Xerox PARC entwickelt wurde. Smalltalk gilt als die objektorientierte Sprache schlechthin und war zu dieser Zeit besonders für die Einfachheit berühmt, mit der man damit grafische Benutzerschnittstellen realisieren konnte.

Smalltalk ist eine Sprache, welche die Objektorientierung bis zum Extremum treibt, so dass beispielsweise sogar Zahlenliterale bereits als Objekte gelten. Zugleich ist Smalltalk vollständig dynamisch typisiert. Daraus resultierend werden Methodenaufrufe prinzipiell dynamisch gebunden.

Mit Objektorientierung, mitgeliefertem grafischem Toolkit, standardmäßiger IDE, Garbage Collector und der Möglichkeit zur Serialisierung von Objekten ohne Mehraufwand gehörte das 1971 entstandene Smalltalk zur Programmiersprachen-Avantgarde. In der Verbreitung steht es zwar weit hinter C++ oder Java, jedoch haben viele andere erfolgreiche Sprachen sich der Konzepte von Smalltalk bedient.

In Objective-C wurden hier die dynamische Bindung, die Syntax der Methodennamen und die dynamisch typisierten Objektreferenzen übernommen. Inzwischen kann man bei Objective-C jedoch auch statisch getypte Objektreferenzen verwenden.

2 Kurzüberblick über Syntax und Semantik

Da Objective-C auf C aufsetzt [1], lässt es sich nicht vermeiden, dass hier auch die Sprache C kurz umrissen wird. Ich werde hier jedoch nur diese Spracheigenschaften von C beschreiben, die für das Verständnis der Objective-C-Erweiterungen erforderlich sind. Für eine vollständige Beschreibung der C-Programmiersprache möchte ich darum an dieser Stelle auf [7] verweisen.

2.1 Kurzeinführung in die Programmiersprache C

C ist eine sehr hardwarenahe Sprache. Der Programmierer hat in C die Möglichkeit, auf den kompletten virtuellen Speicher zuzugreifen, der dem jeweiligen Prozess vom Betriebssystem zugeordnet ist. Er muss sich ebenfalls um die Speicherverwaltung kümmern, was zwar viele Möglichkeiten eröffnet, jedoch auch schnell zu kritischen Fehlern führen kann.

C ist eine prozedurale Sprache, welche keine objektorientierten Konzepte kennt. Ein C-Programm besteht im Wesentlichen aus einer Menge von Datentypvereinbarungen (Strukturen, `typedef`s) und einer Menge von sogenannten *Funktionen*. Funktionen entsprechen Java-Methoden, die als „`static`" deklariert sind, operieren also nicht auf Objekten.

2.1.1 Datentypen

Objective-C verfügt im Wesentlichen über die selben Basisdatentypen wie Java. Auch Arrays lassen sich analog verwenden.

Es ist weiterhin möglich, einen *Zeiger* auf einen Wert eines beliebigen Datentypen zu vereinbaren, indem man dem gewünschten Datentypen einen Stern anhängt. Ein Zeiger ist eine Variable, in der eine Speicheradresse abgelegt ist. An dieser Speicheradresse befindet sich üblicherweise ein Objekt oder ein Wert desjenigen Types, welcher bei der Deklaration des Zeigers spezifiziert wurde. Eine Zeigervariable des Typs `int*` enthält beispielsweise die Adresse eines Wertes vom Typ `int`.

[1] Neuere Entwicklungen des GCC-Compilerzweiges von Apple ermöglichen es auch, C++ als „Basissprache" für Objective-C zu verwenden („Objective-C++").

Weiterhin gibt es `struct` und `enum` Datentypen, die aber zum Verständnis von Objective-C nicht beitragen, so dass ich sie hier nicht weiter betrachte.

2.1.2 Funktionen

Ein Beispiel für eine Funktionsdefinition ist:

```
1 int main ( int argc, char** argv )
2 {
3   puts("Hello, world!");
4 }
```

Die Definition einer Funktion in C erfolgt analog zur Definition einer Methode in Java. Im Unterschied zu Java kennt C jedoch nicht die Schlüsselwörter `public`, `protected` und `private`. Information Hiding wird hier stattdessen mittels eines gezielten Auslassens der entsprechenden Elemente in den Header-Dateien erreicht, welche in Abschnitt 2.1.3 erläutert werden.

Funktionen werden in C nicht in Klassen oder vergleichbare Konstrukte geschachtelt.

Die erlaubten Anweisungen im Funktionsrumpf sind im Wesentlichen identisch mit den Anweisungskonstrukten anderer C-ähnlicher Sprachen: Es gibt `while`- und `for`-Schleifen, `switch`- und `if`-Blöcke.

2.1.3 Header-Dateien

Es ist üblich, C-Programme in mehrere kleine Module aufzuteilen. Diese werden zuerst einzeln in Objektdateien übersetzt, so dass man sie später mittels eines *Linkers* zu einer ausführbaren Datei zusammenfügen kann.

Um ein Modul im eigenen Quelltext zu benutzen, muss man jedoch zuerst die Deklarationen der darin enthaltenen Dinge kennen. Diese werden vom C-Compiler jedoch nicht aus der Objektdatei des Moduls extrahiert. Per Konvention legt man aus diesem Grund für jede C-Datei (`.c`) auch eine sogenannte *Header-Datei* (`.h`) an, in welche man diese Deklarationen einträgt.

Werden nun die Deklarationen eines Moduls von einem Quelltext benötigt, so kann man den Compiler mittels der `#include`-Direktive anweisen, die entsprechende Header-Datei am Anfang des Quelltextes einzufügen.

2.2 Die Objective-C Erweiterung

Objective-C erweitert C um die folgenden Konzepte und Konstrukte:

- Objekte und Klassen
- Vererbung und Subtyping
- Methoden und Nachrichten (Messages)
- Kategorien
- Protokolle [2]
- Erweiterte Datentypen und Konstanten
- Mehr Präprozessordirektiven

[2] Hierbei handelt es sich um das selbe Konzept, welches in anderen Programmiersprachen (z.B. Java) auch als „Interfaces" bekannt ist. Das Wort „Interfaces" ist jedoch bei Objective-C bereits anderweitig belegt.

2.3 Objekte und Klassen

Eine Klasse wird wie folgt definiert:

```
@implementation KLASSENNAME : SUPERKLASSE (KATEGORIE) <PROTOKOLLE>
{
  VARIABLENDEFINITIONEN
}
METHODENDEFINITIONEN
@end
```

Diese Definition wird in der Regel in eine Objective-C Quellcodedatei mit der Endung „`.m`" geschrieben. Variablendefinitionen und Methodendefinitionen werden in den Abschnitten 2.3.1 und 2.5 erklärt.

Damit andere Module die Klasse von außen ansprechen können, muss auch eine Schnittstelle zu dieser angegeben werden. Dies ist jedoch *nicht* optional, sondern muss prinzipiell erfolgen. Der folgende Quelltext soll hierfür als Beispiel dienen:

```
@interface KLASSENNAME : SUPERKLASSE (KATEGORIE) <PROTOKOLLE>
{
  VARIABLENDEKLARATIONEN
}
METHODENDEKLARATIONEN
@end
```

Die Kategorie- und Protokollangaben können im Moment ignoriert werden, und werden hier nur der Vollständigkeit halber aufgeführt. Die meisten Klassen benötigen keine Kategorien oder Protokolle, so dass die `@interface`-Zeile meist nur wie folgt aussieht:

```
@interface KLASSENNAME : SUPERKLASSE
```

Diese Deklaration wird in der Regel in eine „`.h`"-Datei geschrieben, manchmal kommt sie aber auch in einer „`.m`"-Datei vor, so dass die entsprechende Klasse nach aussen überhaupt nicht sichtbar ist.

Es ist hierbei zu beachten, dass es sich *nicht* um Interfaces im Java-Sinne handelt[3], sondern einfach um eine Bekanntmachung der Schnittstelle der Klasse an andere Module. Dieses Interface lässt sich *nicht* verwenden, um auszudrücken, dass die enthaltenen Methoden von zwei unabhängigen Klassen implementiert werden, wie es Java-Kennern suggestiv erscheinen mag.

Es ist möglich, bei der Definition einer Klasse die meisten Angaben auszulassen, die bereits im Interface stehen. Eine Klassenimplementation benötigt nur den Klassennamen und die Methoden.

```
@implementation KLASSENNAME
METHODENDEKLARATIONEN
@end
```

Beim Schreiben einer Klassenimplementation ist darauf zu achten, dass die Klasse stets zuvor mittels `@interface` deklariert wurde. Es ist hierbei unerheblich, ob diese Deklaration in einer eingebetteten Header-Datei oder in der Quelltextdatei selbst steht.

[3] Das, was in Java „Interface" heisst, gibt es in Objective-C auch, jedoch trägt es hier den Namen „Protokoll". Protokolle werden in Abschnitt 2.7 beschrieben.

```
@interface Auto : NSObject
{
 Reifen*      reifen[4];
 int          sitzplaetze;
 NSString*    kennzeichen;
}

// Hier stehen die Methoden
@end
```

Listing 1: Deklaration von Instanzvariablen

2.3.1 Instanzvariablen

Instanzvariablen werden bei der Deklaration einer Klasse angegeben. [4] Hierzu läßt man der `@interface`-Zeile einen Abschnitt in geschweiften Klammern folgen, in welchem diese deklariert werden.

Hierbei wird die übliche C-Syntax zur Variablendeklaration verwendet, wie sie auch in Listing 1 zu sehen ist.

2.3.2 Information Hiding

Auch bietet die Sprache die Möglichkeit, den Instanzvariablen verschiedene Sichtbarkeitsbereiche zuzuordnen. Durch das Einschieben der Sichtbarkeits-Schlüsselwörter `@public`, `@protected` und `@private` zwischen die Variablendeklarationen bekommen alle folgenden Variablen die jeweilige Sichtbarkeit zugeordnet, bis zum nächsten Sichtbarkeits-Schüsselwort oder dem Ende des Instanzvariablen-Blocks. Wenn kein Sichtbarkeits-Schlüsselwort angegeben wird, so werden die folgenden Variablen als ,,protected" behandelt.

Die Bedeutung der drei Sichtbarkeitsstufen ist analog zu den Java-Pendants, wobei die Objective-C-Varianten jedoch nur für Instanzvariablen anwendbar sind. Will man Methoden vor dem Benutzer einer Klasse verstecken, so kann man hierfür Kategorien verwenden, indem man in der `.m`-Datei eine Kategorie mit den zu versteckenden privaten Methoden *definiert und deklariert*. So ist es möglich, dass diese Methoden nicht in der zur Klasse gehörenden Header-Datei vorkommen müssen.

2.3.3 Erzeugen einer Objektinstanz

Da Objective-C keine Konstruktoren kennt, müssen diese mit normalen Methoden nachgebildet werden. Der Programmierer einer Wurzelklasse ist dafür verantwortlich, geeignete Konventionen hierfür zu finden.

Üblicherweise hat jede Klasse eine Methode Namens `alloc` und eine oder mehrere Methoden, die mit `init` beginnen. Mittels `alloc` wird der benötigte Speicher für das Objekt allokiert. Mit einer der `init`-Methoden wird das Objekt vor der ersten Verwendung initialisiert.

2.4 Vererbung und Subtyping

Wie die meisten anderen objektorientierten Sprachen auch, bietet Objective-C dem Programmierer die Möglichkeit, Subklassen zu bilden.

Mehrfachvererbung wird von Objective-C nicht unterstützt. Diese lässt sich jedoch analog zu Java stets über Protokolle nachmodellieren.

An dieser Stelle ist auch der Hinweis angebracht, dass man Objective-C für Objekte auch dynamisch getypt verwenden kann, indem man Objektreferenzen vom

[4] Es ist auch erlaubt, diese bei der Definition explizit anzuführen, dies ist jedoch nicht zwingend erforderlich.

Typ „`id`" verwendet. In frühen Versionen der Sprache liessen sich sogar nur solche Referenzen benutzen.

2.5 Methoden und Nachrichten

In Objective-C-Terminologie wird beim Aufruf einer Methode auf einem Objekt vom Senden von *Nachrichten* gesprochen. Sobald man eine Nachricht an ein Objekt sendet, wird zur Laufzeit bestimmt, welche Methode aufzurufen ist.

Manche Sprachen verwenden aus Geschwindigkeitsgründen hin und wieder statische Bindung bei Methodenaufrufen, indem sie mit den feststehenden Datentypen der Objektreferenzen arbeiten. Sollte man in solchen Sprachen Objekte unvorsichtig und falsch casten, kann es also passieren, dass Methoden auf Objekten aufgerufen werden, welche diese nicht unterstützen. Resultat sind unvorhergesehene Fehler, welche nur schwer zu finden sind, insbesondere wenn die falschen Speicherzugriffe nicht direkt zum Absturz führen, sondern nur nach und nach die internen Programmstrukturen zerstören.

Bei Verwendung von dynamischer Bindung hingegen würde im Falle eines falschen Casts beim ersten Versuch des Aufrufs einer nicht unterstützten Methode das Problem erkannt und im Normalfall mit einer entsprechenden Meldung quittiert.

2.5.1 Methodenbezeichner

Im Gegensatz zur Syntax von C-Funktionen, an der sich die meisten C-verwandten Sprachen orientieren, erscheint die Syntax von Methoden in Objective-C für den C-Kenner zuerst recht fremd.

Wie auch bei den C-Funktionen wird eine Methode über einen Namen identifiziert, wobei die Namen hier jedoch auch speziell Doppelpunkte enthalten können. Ein jeder Doppelpunkt ist hierbei ein Platzhalter für einen Methodenparameter.

Diese Art der Methodenbezeichnung ist für die meisten Programmierer ungewohnt. Da die Methodennamen in Objective-C tendenziell länger sind als die Namen vergleichbarer Funktionen und Methoden in üblicheren Sprachen, wirken die Quelltexte zu Beginn recht unübersichtlich. Auch kann es nützlich sein, einen speziell angepassten Quelltext-Editor zu verwenden, der in der Lage ist, nach den aufgespaltenen Methodennamen zu suchen.

Auf der anderen Seite hat es auch Vorteile im Methodennamen direkt beschreiben zu können, an welcher Stelle welche Parameter erwartet werden. So werden die Methodennamen zwar länger, aber sie sind auch sehr viel ausdrucksstärker.

Zum Beispiel lautet die C-Funktion zum Aufrufen eines Programms
„`int execve(const char *p, char *const a[], char *const e[]);`".
Nur schwer ist aus dem Namen ersichtlich, was diese Funktion macht und was die einzelnen Parameter bedeuten.

Das Pendant in Objective-C hierzu ist die Methode
„`launchedTaskWithLaunchPath:arguments:`", die auf der Klasse NSTask aufgerufen wird, und ein Objekt zurückgibt, welches einen laufenden Prozess repräsentiert. Hier ist sehr viel direkter ersichtlich, was die Methode bewerkstelligt, und welche Parameter wo erwartet werden.

2.5.2 Selektoren

Da die Verarbeitung von ganzen Zeichenketten als Repräsentation für Methoden zur Laufzeit sehr aufwendig werden kann, existiert in Objective-C eine Möglichkeit, hierfür einen einfacheren Datentyp zu verwenden. Man spricht bei Variablen dieses Typs von Selektoren.

```
- (BOOL) openFile: (NSString *) fullPath;

- (void) dealloc;

- (void) tableView: (NSTableView*) tableView
    setObjectValue: anObject
    forTableColumn: (NSTableColumn*) aTableColumn
               row: (int) rowIndex;
```

Listing 2: Beispiele für Methodendeklarationen

Um einen Selektor zu einem Methodennamen zu erhalten, verwendet man einen `@selector`-Ausdruck. `@selector` wird wie eine Funktion verwendet, welche einen Methodennamen entgegennimmt, und einen Wert vom Typ `SEL` zurückgibt.

```
SEL performWithOneArgument = @selector(perform:with:);
```

Der `SEL`-Datentyp ist nicht genauer spezifiziert. Was ein Selektor intern ist und wie von Methodenbezeichnern auf Selektoren umgerechnet wird, wird vom Compilerhersteller definiert und muss vom Programmierer nicht beachtet werden.

2.5.3 Methoden schreiben

Eine Methodendefinition besteht aus einem Methodenrumpf und einem Methodenkopf. Der Rumpf ist dabei aufgebaut wie der Rumpf einer C-Funktion oder Java-Methode.

Der Methodenkopf enthält den Namen der Methode, ihren Rückgabetyp und ihre Parameter, welchen wiederum Typen zugeordnet werden können. Der Kopf beginnt mit einem Minus-Zeichen für Instanzmethoden oder mit einem Plus-Zeichen für Klassenmethoden (Factory-Methoden).

Hat eine Methode Parameter, so werden die Namen der Parametervariablen hinter den einzelnen Doppelpunkten im Methodennamen eingefügt. Optional kann man den Parametern explizit Typen zuweisen, indem man den Typen in Klammern vor den Namen des Parameters setzt. Ebenso kann man den Rückgabetyp der Methode spezifizieren, indem man den Rückgabetypen in Klammern vor den Methodennamen setzt. Läßt man eine explizite Typangabe aus, so verwendet der Compiler an dieser Stelle den Standardtyp `id`.

In der Deklaration einer Klasse werden die Blöcke der Methoden ausgelassen. Zur Abgrenzung von anderen Konstrukten enden die Methodenköpfe hier mit einem Semikolon. Gültige Methodenköpfe sind beispielsweise in Listing 2 zu sehen.

2.5.4 Das Senden von Nachrichten

Methoden werden in Objective-C aufgerufen, indem eine Nachricht an ein Objekt gesendet wird. Dieser Aufruf besteht aus einem Empfänger und einem Methodenbezeichner, die in eckige Klammern gesetzt werden. Ein Empfänger kann hierbei ein Objekt oder eine Klasse sein[5].

```
NSButton* button = [NSButton alloc];
```

Besitzt die Methode Parameter, so setzt man die gewünschten Übergabewerte hinter den Doppelpunkten im Methodenbezeichner ein:

```
button = [button initWithLabel: @"Hallo, Welt!"];
```

[5] Bei dem im untenstehenden Beispiel verwendeten Zeigerdatentypen `NSButton*` handelt es sich um einen statisch getypten Zeiger auf eine Instanz der Klasse `NSButton`.

Es ist wichtig, zu beachten, dass man in Objective-C *jedem* Objekt *jede* Nachricht senden kann. Der Compiler schränkt den Programmierer hier nicht ein. Wenn dieser eine Nachrichtensendung über eine getypte Objektreferenz findet, die er für falsch hält, gibt er lediglich eine entsprechende Warnung aus, übersetzt jedoch das Programm.

2.5.5 Klassenobjekte

Für jede Klasse existiert im Speicher ein sogenanntes Klassenobjekt. Dieses Klassenobjekt enthält eine Liste aller in dieser Klasse verfügbaren Methoden und einen Zeiger zu einer Superklasse, der den Namen `isa` trägt. Handelt es sich bei der Klasse um eine Root-Klasse, so hat der `isa`-Pointer den Wert `Nil`, was einem „nicht zugeordnet" entspricht.

Klassenobjekte werden verwendet, um Methodenaufrufe aufzulösen.

Weiterhin sind hiermit die Möglichkeiten zur Introspektion gegeben, wie man sie von Java kennt, welches ebenfalls Klassenobjekte besitzt.

Es ist auch möglich, Klassenobjekte zur Laufzeit zu modifizieren. So lassen sich beispielsweise der Klassenbaum verändern und Methoden ersetzen. Die in Abschnitt 2.6 beschriebenen *Kategorien* setzen ebenfalls darauf auf.

2.5.6 Forwarding

Wenn ein Objekt eine Nachricht gesendet bekommt, wird anhand des Selektors die gewünschte Methode im Baum der Klassenobjekte im Speicher gesucht. Hierzu wird der Baum von der Klasse des Empfängers bis hoch zur Wurzelklasse durchlaufen. Sobald die Methode gefunden wird, wird sie aufgerufen.

Wird die gewünschte Methode *nicht* im Klassenbaum gefunden, so wird die Nachricht „`forward::` an das Empfängerobjekt gesendet. Die Parameter sind der Selektor der ursprünglich gewünschten Methode und die Liste der Parameter, die für den misslungenen Methodenaufruf vorgesehen war. Die „`forward::`"-Methode existiert in jeder Wurzelklasse, so dass garantiert werden kann, dass die Nachricht bei jedem Objekt ankommt, dessen Klasse von einer der Standard-Wurzelklassen erbt.

Da beim Senden der `forward::`-Nachricht der Klassenbaum erneut von der Klasse des Empfängers an durchlaufen wird, um die Implementation der Methode zu finden, kann man die `forward::`-Methode beliebig überladen. Auf diese Art ist es möglich, fehlgeschlagene Nachrichtensendungen abzufangen und kontextbezogen zu behandeln. Man redet hierbei von *Forwarding*.

Forwarding wird gerne eingesetzt, um Nachrichten an andere Objekte weiterzuleiten. Zusammen mit der einfachen Serialisierung von Objekten und der Möglichkeit zur Introspektive lassen sich so beispielsweise Proxy-Klassen für Objekte *beliebigen Typs* erstellen, die auf anderen Rechnern liegen. Hierzu wird die ursprüngliche Parameterliste untersucht, serialisiert und die Nachricht dann über das Netzwerk an den fremden Rechner geschickt. Zu diesem Zweck existiert für MacOS-X die vorgefertigte *Distributed Objects*-Architektur, welche in [1] im Kapitel „The Objective-C Runtime System" beschrieben wird.

2.6 Kategorien

Über Kategorien ist es möglich, eine Klasse, deren Quelltext dem Programmierer nicht zur Verfügung steht, um eigene Methoden zu erweitern. Dabei wird das Klassenobjekt im Speicher beim Programmstart um die entsprechenden Methoden erweitert. Die Syntax der Deklaration und Definition von Kategorien ist bereits von Klassen bekannt. Der Name der Kategorie wird hierbei in runden Klammern hinter den Namen der erweiterten Klasse gesetzt. Ein Beispiel sind die Header- und

```
#import "Foundation/Foundation.h"

@interface NSTask (ProcessPriority)
-(BOOL) setPriority: (int) aPriority
@end
```

Listing 3: Header-Datei der ProcessPriority-Kategorie

```
#include <sys/time.h>
#include <sys/resource.h>
#import "ProcessPriority.h"

@implementation NSTask (ProcessPriority)
-(BOOL) setPriority: (int) aPriority
{
 // Aufruf der entsprechenden Funktion
 // _taskId : Instanzvariable, in der die Prozess-ID steht.
 return (setpriority(PRIO_PROCESS, _taskId, aPriority) == 0);
}
@end
```

Listing 4: Objective-C-Datei der ProcessPriority-Kategorie

Objective-C-Datei (Listings 3 und 4) der `ProcessPriority`-Kategorie, welche die `NSTask`-Klasse um Möglichkeiten zur Prozesspriorisierung erweitert[6].

Eine der Hauptanwendungen ist die Aufteilung der Implementierung von Klassen in mehrere Kategorien, deren Vereinigungsmenge die eigentliche Klasse bildet. Auf diese Weise lassen sich mehrere Methoden auch syntaktisch in sauber abgetrennte Gruppen zusammenfassen, so daß die eigentliche Definition der jeweiligen Klasse ungeachtet der Größe dieser Klasse dennoch übersichtlich bleiben kann.

2.7 Protokolle

Ein Protokoll ist eine Menge von Methodendeklarationen, welche einen Namen zugewiesen bekommt. Wenn eine Klasse alle Methoden eines Protokolls kennt, so läßt sich dies bei ihrer Deklaration angeben. Um dies zu bewerkstelligen wird der Protokollname in spitzen Klammern hinter den Namen der Klasse gestellt:

```
@interface KLASSENNAME <PROTOKOLLNAME>
// ...
@end
```

Wie in Java kann man Protokolle dazu verwenden, Mehrfachvererbung zu modellieren. Protokolle sind äquivalent zu den „Interfaces" in Java.

Soll eine Objektreferenz auf ein Objekt verweisen, welches ein spezielles Protokoll implementiert, so kann man dies bei ihrem Typ mit angeben. Der Typ einer solchen Referenz ist dann `id <PROTOKOLLNAME>`.

3 Fazit

Dieser Abschnitt nennt zusammenfassend einige Vor- und Nachteile von Objective-C und zeigt auf, welche Einflüsse Objective-C auf andere Sprachen hatte.

[6] Der hier verwendete Datentyp `BOOL` ist der Boolean-Datentyp von Objective-C, welcher die Werte `YES` und `NO` annehmen kann.

3.1 Objective-C und Java

Als Java Anfang der 90er Jahre von *Sun Microsystems* entwickelt wurde, wurden viele Eigenschaften von Objective-C übernommen und einige konzeptionelle Unschönheiten, die von C stammen, entfernt.

Java übernimmt von Objective-C das Konzept der Protokolle als Ausweichmöglichkeit für die nicht vorhandene Mehrfachvererbung, benennt diese jedoch in „Interfaces" um.

Eine weitere Gemeinsamkeit der beiden Sprachen ist die Existenz von Klassenobjekten (siehe Abschnitt 2.5.5), über welche auch erst die Modifikation von Klassen und Betrachtung derselben möglich wird.

Neben einigen syntaktischen Feinheiten löst sich Java auch von der Unterscheidung zwischen Deklaration und Definition, welche von Objective-C erzwungen wird. Hierdurch fällt auch die Erstellung der Header-Dateien weg[7].

Ebenfalls verschwunden ist die heutzutage eher unübliche Namensgebung für Methoden, die in Objective-C verwendet wird. Man kann generell sagen, dass Java syntaktisch recht nahe an C++ oder C ist, semantisch jedoch eher Objective-C oder Smalltalk gleichkommt.

3.2 Nachteile

3.2.1 Geschwindigkeitsverlust gegenüber C

Objective-C ist etwas langsamer als C, da die Methodenaufrufe standardmäßig dynamisch zur Laufzeit gebunden werden. In der Praxis ist dies jedoch schon fast zu vernachlässigen, da die dafür verwandte Zeit lediglich linear zur Tiefe des Klassen-Vererbungsbaumes ist, und dieser sich in den meisten Programmen zur Laufzeit nicht mehr ändert. Darüber hinaus werden die Ergebnisse früherer Suchen im Baum im Regelfall gepuffert, so dass der Aufwand für viele Suchen sehr gering ist. In [2] wird so der Geschwindigkeitsfaktor 2,5 im Vergleich zum Aufruf von C-Funktionen begründet.[8]

Weitere Geschwindigkeitseinbußen gegenüber geschwindigkeitsoptimierten objektorientierten Sprachen wie C++ bestehen darin, dass ein jedes Objekt bei der Erzeugung allokiert und bei der Zerstörung deallokiert werden muss. In anderen objektorientierten Sprachen werden im Gegensatz hierzu Objekte teilweise auch auf dem Stack abgelegt.

Glücklicherweise ist C eine sehr hardwarenahe Sprache, für die es bereits sehr gut optimierende Compiler gibt. Da die gebräuchlichen Compiler für Objective-C auf bereits existierenden C-Compilern basieren, erbt die Sprache den größten Teil dieser Vorzüge.

3.2.2 Keine Konstruktoren und Destruktoren

Etwas schwerwiegender ist der konzeptionelle Mangel, dass Objective-C keine Konstruktoren kennt. Stattdessen werden hier gewöhnliche Methoden verwendet, welche ungetypte Objektreferenzen zurückliefern. Eine Folgerung dessen ist, dass die konstruierten Konstruktoren an alle Subklassen vererbt werden. Selbstverständlich werden viele Subklassen beim Aufruf eines solchen Konstruktors jedoch nicht korrekt initialisiert.

[7] Moderne Entwicklungsumgebungen sind inzwischen in der Lage, Header-Dateien automatisch zu generieren. Generell wird es jedoch als sinnvoller erachtet, diesen Automatismus in den Compiler zu integrieren.

[8] Für performanzkritische Anwendungen besteht auch noch die Möglichkeit, die Ergebnisse „von Hand zu cachen", indem man `IMP`-Zeiger verwendet. In diesem Fall entspricht der Aufwand dem Aufwand des Aufrufs einer C-Funktion.

Dies bedeutet für den Programmierer insbesondere, dass er sich spezielle Techniken aneignen muss, um die ererbten Konstruktoren in Subklassen möglichst geschickt zu überschreiben[9]. Wenn jedoch Sub- und Superklasse von verschiedenen Programmierern implementiert wurden, und der Programmierer der Superklasse nachträglich noch einen Konstruktor hinzufügt, so ist es für einen Benutzer der Subklasse problemlos möglich, die Klasse mit dem falschen Konstruktor zu instanziieren, und somit neu hinzugekommene Instanzvariablen uninitialisiert zu lassen.

Auch Destruktoren sind kein Sprachfeature, und müssen mit normalen Methoden nachmodelliert werden.

3.3 Vorteile

Im Folgenden werden die Vorteile der Programmiersprache Objective-C aufgezählt.

3.3.1 Kurze Einarbeitungszeit

Einer der wahrscheinlich größten Vorteile der Sprache ist ihre Einfachheit. Dies wird sowohl durch das direkte Aufsetzen der Sprache auf C erreicht, welches dem C-Programmierer die Einarbeitung sehr erleichtert, als auch durch die Beschränkung auf nur wenige neue Syntaxelemente.

Laut [1] werden die Produktionen der C-Grammatik nur an vier Stellen verändert: External declarations, Type specifiers, Type qualifiers und Primary expressions[10]. Durch diese Beschränkung gibt es nur wenige, sauber abgetrennte Stellen in einem Quelltext, an denen ein C-Programmierer sich mit der neuen Syntax auseinandersetzen muss. Beim Schreiben der Algorithmen kann er sich so sicher sein, nicht über Eigenheiten der Sprache zu stolpern, mit denen er noch nicht vertraut ist.

3.3.2 Flexibles objektorientiertes Modell

Ein weiterer Vorteil besteht in dem flexiblen objektorientierten Modell, welches in Objective-C verwendet wird. Durch die Verlagerung des Großteils der Algorithmen zur Umsetzung der objektorientierten Konzepte in die Laufzeit wird eine weitaus höhere Flexibilität im Objektmodell ermöglicht, als man es von anderen compilierten Sprachen gewohnt ist.

3.3.3 Wiederverwendbarkeit des geschriebenen Programmcodes

Das direkte Aufsetzen auf C bringt es mit sich, dass bereits geschriebener C-Quelltext unverändert in Objective-C-Programme übernommen werden kann. In den meisten Fällen ist es einfach, eine Klasse um diesen Code herum zu modellieren.

3.3.4 Hoher Abstraktionsgrad der OPENSTEP-Klassenbibliotheken

Die Programmiersprache C ist sehr hardwarenah. Das manuelle Speichermanagement kann das Schreiben von Programmen sehr kompliziert machen, und stellt eine der größten Hürden für C-Neulinge dar.

Die auf Objective-C aufbauenden OPENSTEP-Klassenbibliotheken bringen einen auf Reference-Counting basierenden Garbage-Collector mit, der sich schon mit bedeutend weniger Aufwand nutzen lässt.

[9] Das korrekte Überschreiben von Konstruktoren wird in [1] ausführlich behandelt.

[10] Diese Namen sind bewusst nicht übersetzt worden, da es sich um Bezeichner für Produktionen der C-Grammatik handelt, von der mir keine deutsche Übersetzung bekannt ist.

Anders als die Standardbibliothek von C bieten die verschiedenen OPENSTEP-Versionen viele Möglichkeiten, die weit über Dateizugriffe und mathematische Funktionen hinausgehen. Zu den Features zählen so unter anderem Zugriffe auf Webserver (mit automatischem Caching der heruntergeladenen Dateien), das Rendern von PDF-Dokumenten, OpenGL-Integration und eine Netzwerkschnittstelle.

Literatur

[1] Apple Computer, Inc. *The Objective-C Programming Language*.
http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC/ObjC.pdf

[2] Brad J. Cox und Andrew J. Novobilski.
Object-Oriented Programming - An evolutionary approach. Addison-Wesley Publishing Company, Inc., 2. ed, 1991

[3] *comp.lang.objective-c FAQ*
http://users.pandora.be/stes/faq.html

[4] *Wikipedia-Eintrag zu Objective-C*.
http://en.wikipedia.org/wiki/Objective_C

[5] *Wikipedia-Eintrag zu Smalltalk*.
http://en.wikipedia.org/wiki/Smalltalk

[6] Andrew M. Duncan. *Objective-C Pocket Reference – A guide to Language Fundamentals*. O'Reilly, 2. ed. 2003

[7] Brian W. Kernighan und Dennis M. Ritchie. *The C Programming Language*. Prentice Hall, Inc., 1988