Manipulation der Klassenhierarchie

Die Klassenarchitektur von Object Pascal ist sehr komfortabel, erfordert sie doch auf der einen Seite ein – im Gegensatz zu C++ – relativ geringes Verständnis der für die Funktionsweise verantwortlichen Interna. Auf der anderen Seite setzt sie der Designfreiheit des Programmierers beinahe keine Grenzen.

An eine dieser Grenzen stoßen wir jedoch, wenn wir versuchen, die Klassenhierarchie nachträglich zu verändern, also beispielsweise den Vorgänger einer bereits existierenden Klasse neu zu bestimmen. Handelt es sich hierbei um unsere eigene Klassenhierarchie, so aktualisieren wir selbstverständlich unseren Quelltext dementsprechend.

Wollen wir hingegen den Vorgänger einer bestehenden Klassenhierarchie ersetzen, ohne den Quelltext verändern zu können, so müssen wir uns wohl oder übel doch ein wenig mit den Interna der Klassenarchitektur beschäftigen. Eine Klasse – der virtuelle Bauplan für ein Objekt – setzt sich aus diversen Bestandteilen zusammen, so beispielsweise aus dem *Virtual Method Table* (VMT), der als eine Art Liste von Zeigern auf virtuelle Methoden zu verstehen ist. Wird in einer Klasse eine virtuelle Methode überschrieben, so vermerkt der Compiler dies im entsprechenden VMT, so dass beim Aufruf einer virtuellen Methode nur im jeweiligen VMT nachgeschaut werden muss, um den passenden Methodenzeiger zu finden.

Für unsere Problemstellung – nämlich die Hierarchie einer Klasse verändern zu können – benötigen wir einen anderen Bestandteil, nämlich den Zeiger auf den jeweiligen Vorgänger der Klasse. Der Compiler speichert nämlich nicht für jede neue Klasse die komplette Klasseninformation, sondern nur die Neuerungen bezüglich des jeweiligen Vorgängers. Dies spart auf der einen Seite Arbeitsspeicher und ermöglicht auf der anderen Seite, die Hierarchie nachträglich zu beeinflussen.

Denn der Vorgänger einer Klasse ist in ihrer internen Struktur als Zeiger auf die vorhergehende Klasse vermerkt. Um auf diesen Zeiger zugreifen zu können, definiert Object Pascal in der verborgenen *Unit System* die Konstante *vmtParent*; addieren wir diese Konstante zu einer Klasse – denn eine Klasse ist nichts anderes als ein Zeiger –, so erhalten wir die Adresse des Zeigers auf die Vorgängerklasse. Diesen Zeiger können wir nun – mit ein wenig Aufwand, da der Compiler diesen Speicherbereich vor dem Programmierer zu schützen versucht – verändern, so dass unsere Klasse einen neuen Vorgänger erhält.

Sinnvollerweise sollte der neue Vorgänger natürlich selbst vom ursprünglichen Vorgänger abgeleitet sein, so dass er zwar als eine Art Erweiterung fungiert, jedoch die originäre Struktur nicht grundlegend abändert.

Ein konkrete Beispiel (auf Heft-CD) verändert recht praxisbezogen den Vorgänger von *TControl* – der Basisklasse aller visuellen Elemente der VCL – so, dass auf die Veränderung der zentralen Eigenschaft *Caption* automatisch reagiert werden kann. Die Beispielapplikation demonstriert dies, indem sie vordefinierte Platzhalter in der *Caption* durch feste Werte automatisch ersetzt.

Der folgende Code demonstriert auszugsweise, wie sich der Vorgänger einer Klasse verändern lässt; so ließe sich beispielsweise mit *SetClassParent(TControl, TControlHookClass)* der Vorgänger von *TControl* ändern:

```
Function IncPAddr(Const P: Pointer;
  Const Sum: Integer = 1): Pointer;
Begin
  Result := Pointer(Integer(P) + Sum);
End;

Function GetClassParent(Const AClass:
  TClass): TClass;
Begin
  Result := NIL;
  If Assigned(AClass) Then
    Result := AClass.ClassParent;
End;

Function SetClassParent(Const AClass,
  ANewParent: TClass): Boolean;
Var
  P: Pointer;
Begin
  Result := Assigned(ANewParent) And
    Assigned(GetClassParent(AClass));
  If Result Then
  Begin
    P := IncPAddr(AClass, vmtParent);
    KernelProtectPage(P, SizeOf(P),
      [KernelMemRead, KernelMemWrite]);
    Pointer(P^) := IncPAddr(ANewParent,
      vmtSelfPtr);
    KernelProtectPage(P, SizeOf(P),
      [KernelMemRead]);
  End;
End;
```

Im Grunde genommen liegt die Funktionsweise ganz einfach darin, den Zeiger an *AClass + vmtParent* auf *ANewParent + vmtSelfPtr* zeigen zu lassen, wodurch sich die interne Klassenhierarchie auch für alle von *AClass* abgeleiteten Klassen schlagartig verändert. Die Aufrufe von *KernelProtectPage* (im Quelltext der Heft-CD) sind nötig, um den Arbeitsspeicher an dieser sensitiven Stelle verändern zu können. Delphi kennt seine Grenzen, und versucht sie vor dem Programmierer zu schützen.

(Oliver Friedmann)