Das EVA Paradigma

Inhaltsverzeichnis

1 Definition

Das EVA-Paradigma wurde von Nikolaus Wirth (dem Erfinder der Sprachen Pascal, Modula-2 und Oberon) aufgestellt. Ein Paradigma liefert einen Referenzpunkt für Überlegungen bzw. Faustregeln des Handelns – Paradigmen sind aber keine unumstößlichen Lehrsätze.

Die Abkürzungen “EVA” steht für die 3 Phasen, in die sich (nach Wirth) die meisten datenverarbeitenden Prozesse gliedern lassen sollten: Eingabe, Verarbeitung und Ausgabe.

2 EVA im Softwareentwurf

Der Gebrauch des EVA-Paradigmas beim Software-Entwurf bedeutet vor allem, dass die zeitliche und prozedurale Trennung dieser Phasen angestrebt wird. Programme, welche entsprechend dem EVA-Paradigma strukturiert sind, lesen zuerst die Eingaben vollständig in Datenstrukturen ein, welche sie im Speicher halten, verarbeiten diese Datenstrukturen dann zu anderen, die sie ebenso vollständig im Speicher halten, und schreiben – unter Verwendung der eben erzeugten Strukturen – die Ausgabe. Ein- und Ausgabe sind dabei meist einfache „lineare“ Prozesse, während die Verarbeitung algorithmisch anspruchsvoll sein kann.

3 Vorteile

Der Vorteil der Anwendung von EVA im Softwareentwurf besteht darin, dass damit Fragen des Kontroll- und Datenflusses des verarbeitenden Algorithmus vollkommen abgetrennt werden von der Frage, in welcher Reihenfolge und in welchem Format die Eingabedaten vorliegen. Eine solche Trennung liegt im Interesse der Modularität.

Man beachte, dass es hierbei vor allem um eine zeitliche und logische Trennung der einzelnen Programmphasen geht. Dies hat nichts oder nur wenig damit zu tun, in welchen Modulen die betreffenden Prozeduren liegen: Auch unter dem EVA-Paradigma ist es sinnvoll, Prozeduren, welche auf derselben Datenstruktur (bzw. Datenstrukturen des gleichen Typs) operieren, in dasselbe Modul zu packen.

Dagegen ist es meistens falsch, wenn die Module (ausschließlich) den Verarbeitungsphasen entsprechen.

4 Nachteile

Es sei nicht verschwiegen, dass der Nachteil des EVA-Paradigmas darin besteht, dass es – entsprechende große Datenmengen in der Eingabe vorausgesetzt – zu speicherineffizienten Programmen führen kann, da ja alle Daten während der Verarbeitungsphase im Speicher gehalten werden.

Dies ist in der Übungsaufgabe sowieso kein Problem, weswegen ich dazu rate, den Disassembler nach dem EVA-Paradigma zu strukturieren. Ich möchte aber dennoch kurz auf das angebliche Problem der Speicherineffizienz eingehen.

5 Mangelnde Speicher-Effizienz

Nicht wenige Programmierer implementieren aus Furcht vor Speicherineffizienz vorsorglich in ein Verarbeitungsmuster, bei dem das Programm in einer Schleife wiederholt ein kleines Stück Eingabe liest, verarbeitet, und dann ein Stück Ausgabe schreibt. Das setzt natürlich voraus, dass die Eingabe auch in einer geeigneten Reihenfolge vorliegt und der Verarbeitungsprozess nur einen sequentiellen Zugriff auf die Eingaben benötigt (eine Iteration).

Ändern sich nun die Anforderungen auch nur leicht, und wird ein wahlfreier Zugriff auf die Eingaben notwendig, muss das Programm vollständig umstrukturiert werden. Das ist nicht im Sinne der Wartbarkeit.

Übrigens ist in den meisten Fällen die Angst vor der mangelnden Speicher-Effizienz sowieso übertrieben: Viele Programm mit begrenzter Laufzeit haben klare obere Grenzen für die Menge der Eingabedaten, aus denen sich Obergrenzen für den benötigten Speicher ableiten lassen. In vielen anderen Fällen kann man sich darauf verlassen, dass die virtuelle Speicherverwaltung des Betriebssystems die größten Probleme lindern wird.

Schließlich und endlich gehören auch Speicherfragen in die Kategorie der Effizienzüberlegungen und sollten den Entwurf nur dann beeinflussen, wenn es wirklich zwingend erforderlich ist. Betrachtet man nämlich die Frage der Speichereffizienz getrennt vom Entwurf, können auch sinnvollere und dem Algorithmus angepasstere Verbesserungen vorgenommen werden – etwa eine Virtualisierung der Eingabedaten, so dass diese „on demand“ eingelesen werden – als die gerade angesprochene Ad-Hoc-Lösung, die nur für Algorithmen mit einer simplen Iteration taugt.

Der obige Text entspricht weitgehend dem Originaltext des Handouts, das für die Übungen zusammen mit Blatt 5 (Entwurf eines Disassemblers) ausgegeben wurde. Ich bin mir nicht besonders sicher, dass mein Anliegen in obigem Text wirklich verständlich war: Nur sehr wenige der abgegebenen Entwürfe haben etwas mit den gegebenen Hinweisen anfangen können. Da dies thematisch zum vorliegenden Text gehört, sei kurz in Stichworten erläutert, um was es eigentlich ging:

- *Gute modulare Software ist aus abstrakten Datentypen komponiert.*
- *Dies erfordert unter anderem eine Abstrahierung der Eingabedaten, d. h. die programmexterne konkrete Darstellung der Eingabedaten muss als unabhängig von ihrem logischen Gehalt für das konkrete Problem gesehen werden.*
- *Dem wiederum ist die Anwendung des EVA-Paradigmas förderlich: Die Eingabedaten werden aus ihrer programmexternen Darstellung in die programminterne (die betreffenden ADTs) gelesen, zu anderen abstrakten Datentypen verarbeitet und diese schließlich über ein abstraktes Ausgabegerät in eine programmexterne Darstellung (oder bei Steueranlagen in einen programmexternen Effekt) überführt.*

Der Aufbau des Disassemblers in Lösungskizze 5 (TeachSWT@Tü, 2002) in **Eingabe-Tier**, **Verarbeitungs-Tier**, *und* **Ausgabe-Tier** *ist hierfür ein gutes Beipiel.*

Literatur

[TeachSWT@Tü 2002] LEYPOLD, M E.: Lösungsskizze 5: Entwurf eines Disassembler. Sand 13, D 72076 Tübingen, 2002 (Materialien zur Softwaretechnik). – `handouts/loesung-05.vdm`