Beschreibung von Modulen in natürlicher Sprache

Inhaltsverzeichnis

1 Spezifikation von Modulen

Die Architektur eines Programms ist seine Strukturierung in Module und die Benutzungs- und Aufrufbeziehungen zwischen den Modulen.

Wie schon im Handout *Spezifikation durch Funktionen und die Behandlung von Speicher* (TeachSWT@Tü, 2002) erwähnt, werden Typen, Aufrufsyntax, und Speicherstrategien der aus Modulen exportierten Datenstrukturen und Operationen in den Softwaretechnik-Sprachen durch die Definitionsdateien der Module, in C/C++ dagegen durch Headerdateien beschrieben. Die datenverarbeitende Leistung der einzelnen Operationen ist damit noch nicht festgelegt und muss mit anderen Mitteln spezifiziert werden. Formale Spezifikationssprachen sind hier, weil entsprechend genau, natürlich der Königsweg, aber leider auch sehr aufwändig, und nicht für jeden Kreis von Rezipienten unmittelbar verständlich. In vielen Fällen genügt auch eine natürlichsprachige Beschreibung.

2 Natürlichsprachige Beschreibungen

Natürlichsprachige Beschreibungen haben unter anderem den Nachteil, dass schnell einzelne Fälle, Seiteneffekte, und Ausnahmebehandlungen vergessen werden: Sie erzwingen keine Vollständigkeit. Hier ist es nützlich, sich vom „Geist der formalen Spezifikation“ leiten zu lassen, und Prozeduren als Operationen zu begreifen, die Werte bzw. Zustände auf andere Werte bzw. Zustände abbilden.

In dieser Sichtweise gibt es keine Seiteneffekte (die nebenher erwähnt werden), sondern nur Zustände (z. B. einer Logdatei), die als Eingabe und als Ausgabe in Erscheinung treten. Was sie zu Seiteneffekten macht, ist lediglich die Tatsache,, dass diese Eingaben keine Entsprechung als Parameter in den Prozedurköpfen der Implementation haben.

Im Detail lassen sich aus der Forderung, den „Geist der formalen Spezifikation“ walten zu lassen, die folgenden Gebote ableiten:

- Gib bei (nicht-opaken) Datentypen die Strukturanforderungen (Invarianten) an.
- Werden Parameter durch Referenz auf einen Speicherplatz (bzw. eine Variable) übergeben, dann gib an, ob der Wert in diesem Speicherplatz als Eingabe dient, und ob ein Ergebnis in diesen Speicherplatz geschrieben werden wird.
- Wenn Zeiger als Parameter fungieren, gib an, ob die Werte, auf die der Zeiger verweist, kopiert werden, oder ob der Zeiger direkt weiterverwertet wird (da ist zwar schlechte Praxis, kommt aber zuweilen vor).

  Das bedeutet auch: Gib an, wo die Verantwortlichkeit für die Freigabe des dazugehörigen Speichers bleibt. Dies gilt für alle Arten von Verweisen auf anderswo gespeicherten Zustand: Auch für *Handles* und *Dateideskriptoren*.
- Gib an, welche Daten und Zustände implizit verändert werden.
- Gib Vorbedingungen an, welche auch die impliziten Eingaben mit einschließen.
- Gib Nachbedingungen an, welche die datenverarbeitende Leistung der Prozedur vollständig und eindeutig beschreiben. Diese können häufig in einen Regelfall und in Ausnahmen (Fehlerfälle) getrennt werden, was insbesondere für eine übersichtliche Beschreibung nützlich ist.

3 Ein Beispiel

Prozedurkopf: int symtable_insert(char* idp, desc* dp)

Beschreibung: Fügt das Paar *((*idp),(*dp))* aus Namen und Deskriptor in die (globale) Symboltabelle ein.

Zugriff: Der Zeiger *idp* zeigt auf einen Speicherplatz, in dem der Name als null-terminierte Folge von Zeichen abgelegt ist. Der Zeiger *dp* zeigen auf eine Speicherplatz, in dem der Deskriptor (als opaker Datentyp) gespeichert ist. Beide Speicherplätze werden nur gelesen. Die Werte in beiden Speicherplätzen werden kopiert, so dass der Speicher nach dem Aufruf der Funktion bei Bedarf freigegeben werden kann.

Modifiziert: Der Zustand der Symboltabelle wird implizit verändert.

Vorbedingung: Die Länge von *(*idp)* muss > 0 sein. Namen mit einer Länge von 0 sind nicht statthaft. Der Name *(*idp)* darf sich noch nicht in der Symboltabelle befinden.

Nachbedingung: Wenn noch Platz in der Symboltabelle ist, wird *(*idp)* nach dem Aufruf in der Tabelle und mit dem Deskriptor *(*dp)* assoziiert sein (auf diesen Deskriptor verweisen). Bei erfolgreicher Durchführung der Operation wird der Rückgabewert der Prozedur 1 sein.

Ausnahmen / Fehler: Wenn die Symboltabelle bereits voll ist (die Größe der Symboltabelle ist eine implementationsabhängige Beschränkung), wird ihr Zustand nicht verändert werden und die Prozedur wird 0 zurückgeben, um einen Fehler zu signalisieren.

Aus einer vollständigen natürlichsprachigen Beschreibung müsste sich im Prinzip die formale Spezifikation rekonstruieren lassen. Im vorliegenden Beispiel sollte sich das folgende VDM-SL-Fragment geradezu aufdrängen:

functions

1.0 $\mathit{symtable\text{-}insert}(s : \mathit{symtable}, \mathit{id} : \mathit{identifier}, \mathit{desc} : d)\ s' : \mathit{symtable}, \mathit{succ} : \mathbb{B}$
.1 pre $(\mathsf{len}\ \mathit{id}) > 0 \wedge \mathit{id} \notin (\mathsf{dom}\ s)$
.2 post $(\neg \mathit{full}(s)) \Rightarrow \mathit{succ} \wedge \mathit{id} \in (\mathsf{dom}\ s') \wedge s'(\mathit{id}) = d) \vee$
.3 $(\mathit{full}(s)) \Rightarrow (\neg \mathit{succ}) \wedge s = s')$

Literatur

[TeachSWT@Tü 2002] LEYPOLD, M E.: Spezifikation durch Funktionen und die Behandlung von Speicher. Sand 13, D 72076 Tübingen, 2002 (Materialien zur Softwaretechnik). – `handouts/functionale-spec-und-speicher.vdm`