Entwicklung von Ada 1980 – 2010

Eine Zusammenfassung der historischen Entwicklung der Ada-Norm

Christoph Grein

März 2011

Die Programmiersprache Ada existiert seit nunmehr drei Jahrzehnten. Sie wurde entworfen mit vier bestimmenden Anliegen: einerseits Verlässlichkeit, Wartbarkeit, Effizienz der mit ihr geschriebenen Programme sicherzustellen und andererseits Programmieren als menschliche Tätigkeit möglichst einfach zu gestalten.

Ihre wichtigsten Merkmale sind eine fehlerrobuste Syntax, strenge Typbindung, Modularität, Geheimnisprinzip (*information hiding*), Portabilität, Echtzeitfähigkeit, Nebenläufigkeit, Ausnahmebehandlung und Objektorientiertheit. Zusätzlich hat Ada als einzige Sprache standardisierte Schnittstellen zu C, C++, Cobol und Fortran.

Einzigartig unter allen Sprachen ist, dass Ada mit *ISO/IEC 18009:1999 – Ada: Conformity Assessment of a Language Processor* einen Standard besitzt, der sicherstellt, dass eine Implementierung die Vorgaben des Sprachstandards einhält.

Zur Nomenklatur: Es gibt nur eine Sprache Ada (und kein Ada 83, 95 usw.), und das ist immer die des neuesten ISO-Standards; frühere Standards verlieren mit Erscheinen einer neuen Version ihre Gültigkeit. Nur zur Unterscheidung der einzelnen Generationen wird der informelle Name mit dem Jahreszahlanhang verwendet.

Ada 80: Der erste Standard *ANSI/MIL-STD-1815* von 1980 enthielt noch viele Unklarheiten und Fehler, die bei der Entwicklung der ersten Übersetzer zu Tage traten; er ist nur von historischem Interesse.

Ada 83: Der erste industriereife Standard *ANSI/MIL-STD-1815A-1983, Reference Manual for the Ada Programming Language*, wurde 1987 unverändert als *ISO 8652:1987* übernommen.

Ada war eine der ersten weitverbreiteten Sprachen, die über Sprachkonstrukte verfügte für Abstraktion (Pakete), abstrakte Datentypen (private Typen), nebenläufige Prozesse (*tasks*), generische Schablonen, Ausnahmebehandlung, getrennte Übersetzung unter strenger Typprüfung, offenen Unterprogrammeinbau (*inlining*) und vieles weitere. Damit war die Sprache ihrer Zeit weit voraus und wurde demgemäß als übermäßig komplex gescholten. Seither ist sie jedoch, was Komplexität betrifft, von anderen Sprachen weit überholt worden, insbesondere von C++, während ihre Qualitäten in der Kombination von Sicherheit, Effizienz und Echtzeitkontolle keinerlei Entsprechung gefunden haben.

Die neueren Sprachgenerationen bauen auf den vorhandenen Strukturen auf und fügen Funktionalität hinzu, die sich im Laufe der fortschreitenden Softwareentwicklung als nützlich erwiesen hat, ohne die ursprüngliche Betonung von Zuverlässigkeit, Wartbarkeit und Effizienz zu gefährden. Hintergründe für die Weiterentwicklung der Sprache findet man auch im Vorwort des jeweiligen Standards.

Eine der wesentlichen Eigenschaften Adas ist die strikte Trennung der Spezifikation und der Implementierung von Typen und Operationen. Typen können als *privat* gekennzeichnet werden, ihre innere Struktur ist dann nicht sichtbar; werden sie als *limitiert privat* gekennzeichnet, können auch keine Kopien von Objekten dieses Typs angelegt werden.

Ada 95: Diese Sprachüberarbeitung ermöglichte größere Flexibilität der Sprachkonstrukte. Auch das Referenzmanual wurde neu strukturiert: *ISO/IEC 8652:1995 Ada Reference Manual*.

Zur Erleichterung des Übergangs zum neuen Standard enthält das *Annotated Language Reference Manual* (eine Version des Standards mit zusätzlichen inoffiziellen Anmerkungen zur Sprache als Hintergrundinformation) eine Auflistung aller Änderungen unter den Stichworten *Inconsistencies With Ada 83*, *Incompatibilities With Ada 83*, *Extensions to Ada 83*, *Wording Changes from Ada 83*.

Ada 83 kannte nur den 7-Bit-ASCII-Zeichensatz genannt `Character`. Er ist nun auf 8 Bit erweitert. Allerdings ist es auch schon Ada83-Übersetzern aus Kompatibilitätsgründen zu Ada 95 erlaubt, den 8-Bit-Zeichensatz zu verwenden. Ein neuer Typ *Wide_Character* stellt den 16-Bit-Zeichensatz zur Verfügung.

Während Ada 83 mit seiner strengen Typbindung schon Vererbungsmechanismen enthielt, fehlten doch Typerweiterbarkeit und Laufzeitpolymorphismus. Daher wurde Ada 83 nur *objektbasiert* genannt. Mit den so genannten *tagged types* (das sind erweiterbare Verbunde) ist nun volle *Objektorientiertheit* erreicht.

Weiterhin sind nun standardisierte Pakete zur Unterstützung wichtiger Anwendungsgebiete (Systemprogrammierung, Echtzeitsysteme, verteilter Systeme, Numerik, Sicherheit) in Anhängen zum Kernstandard definiert, die optional sind (sie müssen allerdings, wenn sie ein Übersetzer anbietet, vollständig implementiert sein). Als besonders störend in Ada 83 war z.B. das Fehlen einer Bibliothek mit mathematischen Grundfunktionen (Logarithmen, Kreisfunktionen u.ä.) empfunden worden.

Zur einfacheren Synchronisation nebenläufiger Prozesse wurden *protected types* und *objects* eingeführt. Ada 83 hatte nur Rendezvous zur Synchronisation von Prozessen vorgesehen, was als umständlich angesehen wurde. (*Protected objects* ersetzen seither die so genannten *passive tasks*, für die zur Implementierung im Rahmen von Ada 83 spezielle Optimierungen entwickelt worden waren, die eine leichtgewichtige Synchronisation ermöglicht hatten ähnlich der mit den neuen *protected objects*.)

Die Grundstruktur zur Modularität in Ada ist das Paket. Bibliotheken bestehen aus Sammlungen von Paketen. In Ada 83 waren Bibliotheken flach, was die Strukturierung großer Systeme durch lange Namen umständlich machte. Ada 95 erleichtert die Modularisierung durch Einführung hierarchischer Bibliotheken mit Abkömmlingen (*child packages*) der bisherigen Pakete. Zur besseren Kontrolle der Sichtbarkeit zwischen Paketen wurden private Abkömmlinge (*private child packages*) eingeführt.

Ada 83 bot zur Echtzeitbehandlung nur relative Wartezeiten an. Ein neues Paket mit einer kontinuierlichen Uhr (die bisherige Kalenderuhr konnte beim Übergang zwischen Sommer- und Winterzeit springen) ermöglicht nun die Definition einer genau terminierten Wartezeit.

Zugriffstypen (*access types*), in anderen Sprachen gewöhnlich Zeiger (*pointer*) genannt, können jetzt sowohl auf beliebige Objekte (Variablen und Konstanten) zeigen als auch auf Unterprogramme. Anstelle des bisherigen impliziten *heaps* können dedizierte Speicherbereiche (*storage pools*) definiert werden. Zugriffstypen können auch anonym sein. Spezielle Regeln zur Lebenszeitüberwachung verhindern dabei, dass Zeiger die durch sie bezeichneten Objekte überleben (*dangling pointers*).

Neben gewöhnlichen Festpunkttypen (basierend auf einer von der Hardware vorgegebenen Basis, gewöhnlich 2 oder 16) gibt es jetzt auch dezimale Festpunkttypen, die besonders im Bankenwesen Anwendung finden.

Ada 95 Technical Corrigendum 1: *ISO/IEC 8652:1995/Cor1:2001* Diese relativ kleine Sprachrevision korrigiert Ungenauigkeiten und Fehler des Standards.

Eine wesentliche Änderung betrifft die Aufteilung der bisherigen Repräsentationsklauseln in Operationsklauseln und (eigentliche) Repräsentationsklauseln. (Eine `Storage_Pool`-Klausel ist z.B. eine Repräsentationsklausel, eine Read-Klausel dagegen eine Operationsklausel.) Operationsklauseln und Repräsentationsklauseln zusammen heißen nun Aspektklauseln. Diese Änderung war nötig wegen Problemen bei der Vererbung von *tagged types*.

Ada 2005 Amendment 1: *ISO/IEC 8652:1995/Amd1:2007* Diese Sprachrevision fügt weitere standardisierte Pakete hinzu.

Wie schon bei Ada 95 enthält das *Annotated Language Reference Manual* eine Auflistung aller Änderungen gegenüber früheren Standards.

Ein zusätzlicher Typ stellt das gesamte 32-Bit-Zeichensatzrepertoire gemäß *ISO/IEC 10646:2003* (`Wide_Wide_Character`) zur Verfügung.

Interface-Typen stellen ein moderate Form der Vielfachvererbung nach dem Java-Modell bereit. Sie können auch auf Prozesstypen und Typen für geschützte Objekte angewendet werden und verbinden somit nebenläufige Prozesse mit Vererbung.

Zugriffstypen erlauben nun die Definition eines Null-Ausschlusses, d.h. solche Zeiger müssen stets auf ein Objekt verweisen. Anonyme Zugriffstypen können unter erweiterten Umständen verwendet werden.

Überarbeitete Sichtbarkeit zwischen Paketen erleichtert nun die Definition wechselseitiger Abhängigkeit zwischen Typen (*limited with-clauses*).

Limitierte Objekte können mittels spezieller Aggregate oder Konstruktorfunktionen initialisiert werden.

Eine Containerbibliothek ähnlich der STL von C++ stellt Strukturen für Listen, Bäume, Zuordnungen u.a. zur Verfügung.

Zwei spezialisierte Anhänge wurden beträchtlich erweitert:

Der Echtzeit-Anhang enthält nun das sogenannte Ravenscar-Profil (benannt nach dem Ort, wo es während einer Konferenz definiert wurde) für hochzuverlässige Systeme. Zusätzlich zur bisherigen präemptiven *Dispatching-Policy* sind weitere Dispatching-Policies definiert (z. B. *EDF — earliest deadline first*).

Der Numerik-Anhang stellt neben komplexer Vektorarithmetik, die zuvor in dem separaten Standard *ISO/IEC 13813:1997* definiert war, weitere Operationen der linearen Algebra bereit.

Insgesamt wurde die Zuverlässigkeit der Sprache erhöht durch neue Syntax und Zusicherungen (*assertions*).

Ada 2012 Amendment 2: Diese neueste Überarbeitung des Sprachstandards ist derzeit noch in Entwicklung. Ihr erklärtes Ziel ist, Ada einen Stil zu verpassen, der das Abstraktionsniveau des Kodes noch näher zum Problembereich bewegt: Neue Syntax für Iteratoren, Prä- und Postkonditionen, Invarianten und Prädikate.

Prä- und Postkonditionen gehören zur Spezifikation von Operationen, daher ist es nun nötig, von dem Grundsatz „Keine Unterprogrammrümpfe (Kode) in Spezifikationen“ abzugehen. Ausdrücke konnten allerdings schon immer als Konstanten in Spezifikationen definiert werden. Nun werden auch bedingte und quantifizierte Ausdrücke (*conditional* und *quantified expressions*) in so genannten *expression functions* erlaubt.

Bislang beschränkte sich die Definition des Wertebereichs von Typen auf Eigenschaften, die sich durch zusammenhängende Bereichseinschränkungen und Diskrimi-

nanten ausdrücken ließen. Invarianten erlauben, zusätzliche Eigenschaften wie die Geradzahligkeit der zugelassenen Werte zu definieren.

Zugriffe auf Elemente in Containern werden wesentlich vereinfacht durch implizite Dereferenzierung und Indizierung. Die neue Syntax erlaubt Elementzugriffe und Schleifenindizierung mit Containern, als wären sie gewöhnliche Felder.

Die Erweiterung der Containerbibliothek durch Warteschlangen stieß auf unerwartete Schwierigkeiten, an deren Lösung durch erweiterte Syntax derzeit gearbeitet wird.