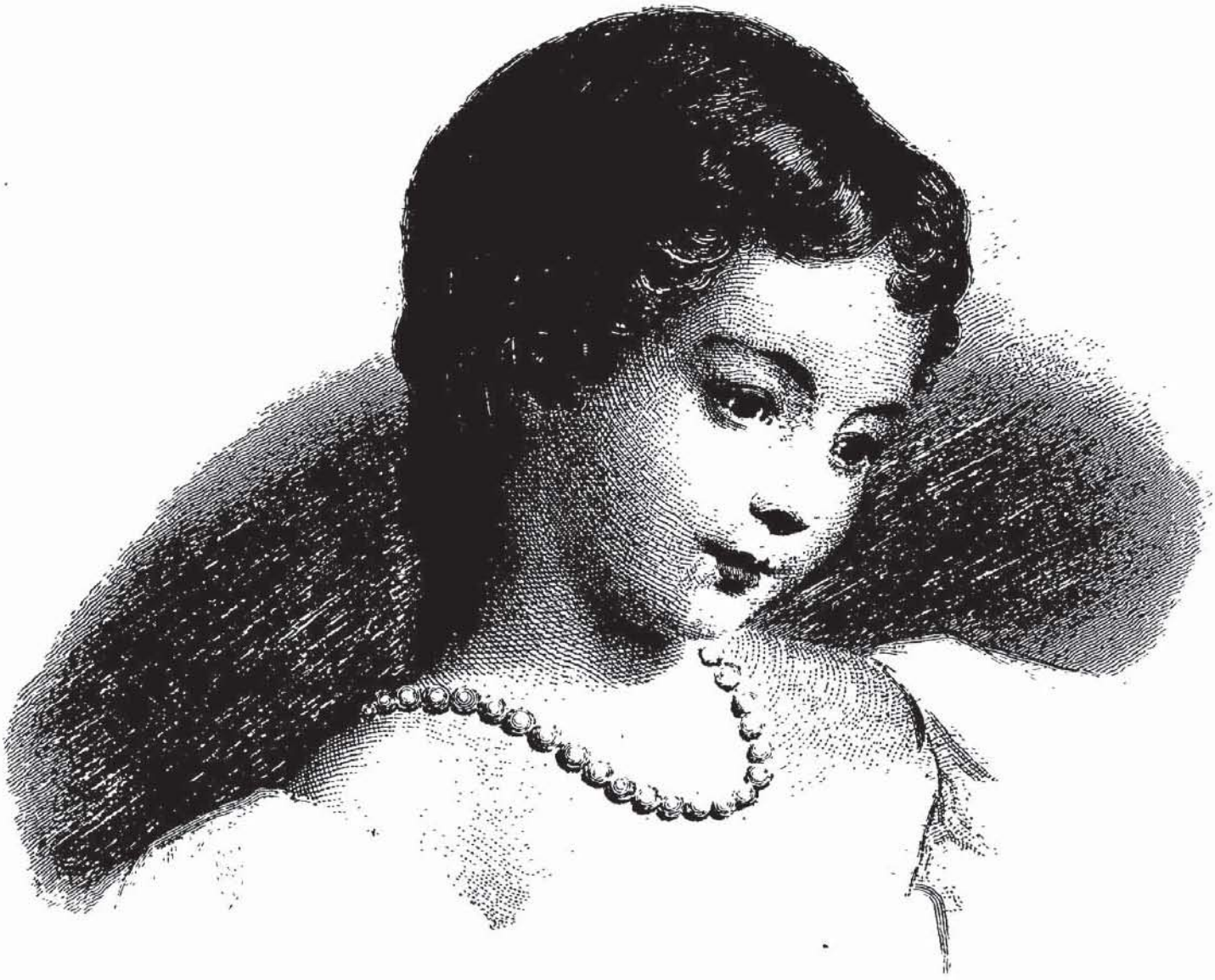
ADA



Nach ihr wurde die Sprache Ada benannt: Ada Augusta Countess (1815...1852) of Lovelace, Tochter des Lord Byron. Sie war Gefährtin und Mitarbeiterin des Mathematikers Charles Babbage, der einen mechanischen programmierbaren Computer entwickelte, die sogenannte "difference engine". Ada schrieb für diese difference engine Programme und gilt daher als die erste Programmiererin der Welt. Der Stich, von dem diese Reproduktion stammt, ist in der Sammlung "Preußischer Kulturbesitz" in Berlin zu finden. Der Stich zeigt Ada im jugendlichen Alter von 14 Jahren.

VON DR. H. HUMMEL, E. PLÖDEREDER.

Die Programmiersprache Ada Hintergründe, Entwicklung und Sprachkonzepte

Anfang der 70er Jahre war das Wort "Softwarekrise" in aller Munde. Die EDV-Hersteller und -Großanwender sahen sich mit explosionsartig steigenden Softwarekosten konfrontiert. Die mangelnde Qualität der Softwarekosten war im Begriff, den EDV-Einsatz gänzlich zu diskreditieren. So ermittelten z. B. Untersuchungen aus dem Jahre 1975 im amerikanischen Verteidigungsministerium (US DoD) – einem der größten EDV-Anwender der Welt – direkte Softwarekosten von über 3 Milliarden Dollar pro Jahr mit einer erschreckend hohen Steigerungsrate. Dabei liegt der Anteil für Wartung und Pflege existierender Software bei rund 70%. Die Auswirkungen von Softwaremängeln verursachten immense Folgekosten: So verlor NASA eine Mariner-Sonde durch einen Programmfehler; das amerikanische Frühwarnsystem wurde von Fehlalarmen geplagt. Die Wartung großer in Assembler programmierter Systeme nach mehreren Jahren des Einsatzes erwies sich als kaum noch durchführbar.

Als Ursache dieser Software Krise wurde primär der Mangel an modernen Konzepten und die fehlende Standardisierung im Bereich der verwendeten Programmiersprachen identifiziert. Die Untersuchungen zeigten, daß über 100 verschiedene Programmiersprachen eingesetzt wurden, die zudem meist weder sicherheits-, noch wartungsfreundlich waren. Aufgrund der fehlenden Standardisierung und der Maschinenabhängigkeiten der verwendeten Sprachen war die entwickelte Software nicht portabel und damit nicht wiederverwendbar. Dies führte zu kostspieligen Parallelentwicklungen und Neuimplementierungen, die mit portabler Software vermieden worden wären.

Um diese Softwarekrise in den Griff zu bekommen, wurde vom US DoD das "Software Management Project" ins Leben gerufen. Eines der Hauptziele bestand in der Festlegung einer Standardsprache für den gesamten amerikanischen Verteidigungsbereich, die moderne Spracheinrichtungen zur Qualitätssicherung und Wartungsfreundlichkeit der Software besitzen sollte.

Als eine der ersten Maßnahmen wurde die Zahl zugelassener Sprachen auf sieben, ausschließlich höhere Programmiersprachen beschränkt. Die gleichzeitig dazu eingeleitete Suche nach einer Standardsprache konzentrierte sich zunächst auf bereits existierende Sprachen. Die Feststellung, daß keine dieser Sprachen den Forderungen eines modernen "Software Engineering" vollendes entsprach, stieß dann aber die Entwicklung einer neuen Programmiersprache an, wie im folgenden noch näher beschrieben wird.
Die missliche Lage auch der industriellen Anwender führte dazu, daß sich schon bald von dieser Seite ebenfalls ein ausgeprägtes Interesse an der neuen Sprache zeigte. Dies schlug sich in einer regen Beteiligung der Industrie an der Entwicklung von Ada nieder. Heute, nach Abschluss der Entwicklung von Ada, wird weltweit von vielen Experten die begründete Hoffnung gehegt, daß Ada mit der Breite seiner Anwendungsmöglichkeiten erheblich zur Milderung der Softwaremisere im industriellen, militärischen und akademischen Bereich beitragen wird.

Die Entwicklung von Ada

Die Entwicklung von Ada verlief in fünf Hauptabschnitten mit jeweils genau umrissenen Zielvorstellungen.
Ziel des ersten Abschnitts (Requirements) war 1975/76 die Ermittlung der Anforderungen der zukünftigen Benutzer an eine für ihr Anwendungsgebiet geeignete Programmiersprache. Eines der Hauptergebnisse dieses Abschnitts war die Erstellung von Forderungskatalogen für eine derartige Programmiersprache in drei Iterationsschritten, genannt STRAWMAN, WOODENMAN und TINMAN. Diese Forderungskataloge wurden jeweils von Fachleuten aus dem militärischen Bereich sowie aus Industrie und Forschung begutachtet und kritisiert.
In der zweiten Phase (Base Language Evaluation) wurden sechs amerikanische Software-Häuser beauftragt, vierzehn höhere Programmiersprachen auf ihre Eignung anhand der im TINMAN-Dokument aufgeführten Spezifikationen zu untersuchen und zu bewerten. Neben diesen sechs formell beauftragten Gutachtern meldeten sich weitere neun Firmen und Institute aus den USA und Europa zu Wort und brachten zusätzliche neun Sprachen ins Gespräch. Darunter befand sich auch die deutsche Echtzeitsprache PEARL. Als Erkenntnis aus diesen Untersuchungen ergaben sich die folgenden Punkte:

- Keine der untersuchten Sprachen erfüllt die TINMAN-Anforderungen ausreichend.
- Keine der TINMAN-Anforderungen ist grundsätzlich neu. Einzelne Anforderungen werden von der einen oder anderen der untersuchten Sprachen erfüllt.
- Der Entwurf der angestrebten Sprache soll von einer sorgfältig gewählten Basissprache ausgehen. Als Basissprache kommen in Betracht: PASCAL, PL/I und ALGOL 68.

Weiter wurden aus dem detaillierten Forderungskatalog TINMAN die Sprachspezifikationsdokumente IRONMAN und IRONMAN REVISED abgeleitet. Zum Abschluß dieser Phase, die sich bis etwa Mitte 1977 erstreckte, wurde vom DoD die Definition einer den IRONMAN- Spezifikationen genügenden Programiersprache international ausgeschrieben. Achtzehn Firmen bewarben sich um diese Aufgabe, darunter Firmen aus Deutschland, England und Frankreich.
Im dritten Abschnitt (Design-Phase 1) wurden vier dieser Firmen ausgewählt und beauftragt, konkurrierende Sprachentwürfe einzureichen, die eine der drei empfohlenen IRONMAN REVISED genügen sollten. Die Projekte erhielten die Bezeichnungen GREEN, RED, YELLOW und BLUE, damit vor den Beurteilern ihre Herkunft verborgen blieb und so einer Beeinflußung vorgebeugt wurde. Ein erstes interessantes Ergebnis war, daß alle vier Sprachentwickler PASCAL als Ausgangssprache wählten. Zwischenzeitlich wurde das Sprachspezifikations-Dokument IRONMAN REVISED einer neuerlichen Überarbeitung unterzogen und erhielt den Namen STEELMAN.
Die vier erstellten Sprachentwürfe wurden im Februar 1978 an über zweihundert Teams zur Begutachtung verteilt. Obwohl die Zeit für diese vergleichende Bewertung äußerst knapp bemessen war, nahmen mehr als achtzig Teams in den USA und in Europa an der Begutachtung teil. Auch von deutscher Seite wurde ein Team von Fachleuten zusammengestellt, das ein umfangreiches Gutachten erarbeitete. Aufgrund der Ergebnisse dieser Sprachevaluationen wurde je ein Auftrag an die Entwickler der Sprachen GREEN (Cii-Honeywell) und RED (Intermetrics) vergeben, in einer zweiten Entwurfsphase (Design-Phase 2) vollständige Detailentwürfe und Testimplementationen ihrer Vorschläge durchzuführen.
Der vierte Abschnitt (Design-Phase 2) war bestimmt von einer Reihe von Design-Reviews, die die detaillierten Sprachentwürfe begutachteten. Anläßlich der ersten öffentlichen Vorstellung der beiden Sprachentwürfe im November 1978 wurde auch der Name der neuen Programmiersprache bekanntgegeben. Zu Ehren von Augusta Ada Byron, die Charles Babbage's analytische Maschine programmierte und damit die erste Programmiererin der Geschichte war, wurde die Sprache Ada genannt.
Im März 1979 wurden die entgültigen Sprachentwürfe für GREEN und RED veröffentlicht und anschließend wieder einem öffentlichen Beurteilungsverfahren unterzogen. Fast vierzig Teams beteiligten sich diesmal an der Begutachtung. Mitte April fand eine viertägige technische Diskussion mit den beiden Sprachdesignern statt, bei der fast alle Begutachterteams vertreten waren. Auch diesmal wurde von deutscher Seite wieder ein umfangreiches Gutachten erstellt. Am 2. 5. 1979 fiel die Entscheidung, den Sprachentwurf von Cii-Honey well, GREEN, als alleinigen Kanditat für Ada weiterzuentwickeln.
Im fünften Abschnitt (Design-Phase 3) wurde die endgültige Form der neuen Programmiersprache Ada festgelegt - ausgehend von der seit der Design-Phase 2 vorliegenden Version von GREEN. Von Mai bis Dezember 1979 lief eine Test- und Beurteilungsphase (Test & Evaluation Phase), in der Anwender eigeladen waren, Probleme in Ada zu codieren und ihre Erfahrungen dem Beurteilungskoordinator mitzuteilen. Sie erhielten auch Gelegenheit, mit dem Sprachentwickler zu diskutieren. Diese öffentliche T & E-Phase mit einer äußerst regen Beteiligung von Fachleuten aus USA, Europa und Japan zeigte eine erstaunlich starke Reso-

nanz. Während dieser Zeit gingen über 600 Lir's (Language Issu Reports") von Institutionen und Einzelpersonen und knapp 100 Test- und Evaluationsreports ein.
Die Erfahrungen dieser Testphase fanden Eingang in die endgültige Sprachspezifikation, die vom Sprachentwurfsteam unter Beratung und Mitwirkung der sogenannten "Distinguished Reviewers" (DR) erstellt wurde. (Die DR, eine kleine Gruppe von Experten aus USA, England und Deutschland, war vom DoD als Fachgremium für die Sprachentwicklung eingesetzt worden.) In dieser letzten Phase wurden in intensiven Diskussionen, die sich in mehr als 150 "Language Study Notes" dokumentierten, die endgültigen Sprachfestlegungen getroffen. Am 4. und 5. September 1980 wurde Ada offiziell vom Pentagon in Washington vorgestellt.
Die aufgezeigten Phasen der Ada-Entwicklung beleuchten die beispiellose Vorarbeit, die zur endgültigen Festlegung von Ada führte. Die äußerst intensive Interaktion zwischen zukünftigen Benutzern, Sprachexperten und dem Entwicklungsteam hat stark dazu beigetragen, daß Ada dem derzeitigen Wissenstand im Bereich der Programmiersprachen entspricht und gleichzeitig den Bedürfnissen des Anwenders optimal gerecht wird.
Die Tatsache, daß die Festlegung der Sprache nicht von Anfang bis Ende in der Abgeschlossenheit eines Forschungslabors durchgeführt wurde, trägt zweifellos zum Vertrauen auf den Erfolg von Ada bei. Als erste Auswirkung hat z. B. die US Army inzwischen für alle Software-Projekte, die ab 1983 anlaufen, die Sprache Ada zwingend vorgeschrieben. Obgleich heute noch keine Compiler in Produktionsqualität zur Verfügung stehen, setzen eine Reihe von Software-Entwicklern im In- und Ausland Ada bereits jetzt als firmen-interne Spezifikations- und Dokumentaionssprache ein.

Die Grundkonzepte von Ada

Ada wurde ausgehend von PASCAL entwickelt, was sich darin niedergeschlagen hat, daß der Kern von Ada relativ PASCAL-ähnlich ist. Ein dominantes Merkmal ist die Deklarationspflicht für alle verwendeten Größen und die strenge Typbindung. Ein Typ ist charakterisiert durch eine Menge von Werten und eine Menge von Operationen, die (nur) auf diese Werte anwendbar sind und nicht aus der Wertemenge hinausführen. Ada kennt als konkrete Datentypen

- skalare Typen (Boolesche Werte, Zeichen, ganze Zahlen, Festpunkt- und Gleitpunktzahlen, Aufzählungen),
- zusammengesetzte Typen (Arrays, Records),
- referierende Typen (Pointer) und
- abgeleitete Typen

Bei den skalaren Typen handelt es sich, mit Ausnahme der Aufzählungen, um vordefinierte Standardtypen. Weiter ist die Deklaration von statischen und dynamischen Arrays beliebiger Dimension möglich. Records dürfen mit sogenannten Diskriminanten versehen werden, die es ermöglichen, verschiedene Varianten eines Records zu schaffen. Mit referierenden Typen bietet Ada die Möglichkeit, Objekte dynamisch während der Laufzeit zu erzeugen. Abgeleitete Typen erlauben dem Benutzer die Trennung zwischen einem konzeptuellen Typ (z. B. "Währung") und seiner Darstellung (z. B. als Festpunktzahl) zu unterscheiden. An Operationen enthält Ada die Grundmenge der arithmetischen und logischen Operationen. Zusätzlich stehen noch Indizierung, Selektion und Qualifiktation zur Verfügung. Weitere Operationen können vom Benutzer definiert werden.
Zur Strukturierung und Kontrolle des Programmablaufs hat Ada im Kern die von PASCAL bekannten traditionellen Anweisungen übernommen. Zu finden sind die Zuweisung, IF- und CASE-Anweisung und eine Basisschleife mit den Möglichkeiten der FOR- und WHILE-Iteration und des kontrollierten Schleifenausgangs. Es wird auch eine Sprunganweisung angeboten, deren Freiheitsgrade jedoch stark eingeschränkt sind.
Die Programmstrukturierung, d. h. modularer Programaufbau, wird von einer Reihe von Einrichtungen unterstützt bzw. sogar erzwungen. Auf der niedrigsten Ebene kennt Ada die Blöcke, die zwar zur syntaktischen Kategorie der Anweisungen gehören, jedoch im Grunde eingebettete, sofort ausführbare Unterprogramme darstellen, die mit einem eigenen lokalen Deklarationsteil versehen werden können.
"Echte" Programmeinheiten dagegen sind die Unterprogramme, von denen es zwei Arten gibt. Zum einen sind das die Prozeduren, die eine Folge von Aktionen beschreiben, zum anderen die Funktionen, die zusätzlich einen Ergebniswert zurück liefern. Deklaration (Schnittstelle) und Rumpf (Implementierung) eines Unterprogramms kann separiert werden.
Ada erlaubt es, den gleichen Unterprogrammnamen für verschiedene Unterprogramme zu verwenden (Overloading). Entscheidungen, um welches Unterprogramm es sich bei Aufrufen jeweils handelt, werden vom Compiler aus dem Kontext getroffen. Gleiches gilt für die Möglichkeit, etwa "+" auch für andere Datentypen und Strukturen zu deklarieren, z. B. für Matrizen.
Diese bisher vorgestellten traditionellen Sprachkonstrukte bilden quasi den Kern von Ada. Jedoch gibt es noch eine ganze Reihe zusätzlicher Einrichtungen, die weit über diesen Kern hinausgehen:
Ein sehr wichtiges Mittel zur Strukturierung von Programmen bietet Ada durch das Modulkonzept

(packages). Im einfachsten Fall stellen Moduln nur einen globalen Pool von Daten und Typdeklarationen zur Verfügung. In der Regel aber werden Moduln verwendet, um abstrakte Datentypen zu beschreiben, d. h. Typen, Objekte und Operationen zusammenzufassen.
Wie bei Unterprogrammen ist für Moduln eine Trennung zwischen Moduldeklaration (Spezifikation der Schnittstelle) und Modulrumpf (Implementierung der Dienstleistungen) zu finden. Die Leistung eines Moduls ist nur über die Spezifikation ansprechbar; die Implementation ist für Modulnbenutzer völlig verborgen. Aus der Spezifikation läßt sich entnehmen, welche Objekte und Manipulationen einem Benutzer angeboten werden.
Eine der wesentlichsten Fähigkeiten von Ada ist die Möglichkeit der seperaten Übersetzung von Programmeinheiten (Moduln, Unterprogramme), wobei das Einhalten von Schnittstellenkonventionen vom Compiler geprüft wird. Damit zusammenhängend ist die Verfügbarkeit einer adäquaten Programmbibliothek als integraler Bestandteil der Sprache. Ein Programm in Ada ist somit nicht als ein Ganzes zu sehen, sondern wird durch eine Kollektion von Kompilationseinheiten repräsentiert, wobei die einzelnen Einheiten durch verschiedene Übersetzungen in die Bibliothek eingebracht werden können. Diese Organisation unterstützt als eine Arbeitsweise, bei der man sich nach einer Art Baukastensystem für ein spezielles Problem passende Einheiten aus der Bibliothek sucht und zu einer Lösung oder Teillösung "zusammenstellt".
Ferner besitzt Ada das Konzept der Prozesse (tasks) zur Modellierung und Programmierung paralleler Aktionen. Der Aufbau von Prozessen in Ada ist völlig analog zum Aufbau von Moduln. Auch hier wird klar zwischen Deklaration und Implementierung getrennt. Dabei definieren die Prozessrümpfe die Ausführung der Prozesse der entsprechenden Prozesstypen. Prozesspezifikatioen bestehen ausschließlich aus sogenannten ENTRY-Deklarationen (Deklarationen von Eingängen), die ähnlich den Unterprogrammdeklarationen sind. Eingänge eines Prozesses, d. h. Interfaces zu "Dienstleistungen" dieses Prozesses, können von anderen Prozessen aufgerufen werden. Die Aktionen, die bei einem solchen Aufruf ausgeführt werden, sind durch korrespondierende ACCEPT-Anweisung im gerufenen Prozess festgelegt. Der Aufruf von ENTRIES und die ACCEPT-Anweisung sind somit die grundlegenden Spracheinrichtungen für Synchronisation und Kommunikation zwischen Prozessen.
Normalerweise läuft die Kommunikation zwischen Prozessen in der folgenden Art ab: ein (rufender) Prozess setzt einen ENTRY-Call an einen anderen (gerufenen) Prozess ab. Wenn der gerufene Prozess die zum Aufruf passende ACCEPT-Anweisung erreicht, werden - nach eventueller Parameterubergabe - die Statements der ACCEPT-Anweisung vom gerufenen Prozess ausgeführt. Dabei befindet sich der rufende Prozess in einem Haltezustand. (Diese Interaktion wird als Rendezvous bezeichnet.) Nach der Ausführung der ACCEPT-Anweisung laufen rufender und gerufener Prozess wieder parallel weiter.
Als eine weitere Spracheinrichtung bietet Ada das Konzept der generischen Einheiten an. Generische Einheiten sind Modelle (oder Schablonen) zur Erzeugung von Programmeinheiten (Unterprogramme und Moduln) durch Parametrisierung des Textes zur Übersetzungszeit. Dabei sind neben Variablen auch Datentypen und Operationen als Parameter erlaubt. Dieser Sprachkonstrukt bietet somit die Möglichkeit, durch die Verwendung von Modellen strukturell ähnliche Programmeinheiten automatisch zu generieren.
Mit dem Exception-Mechanismus wird ein Konzept bereitgestellt, das es erlaubt, Laufzeitfehler und Ausnahmesituationen zur Laufzeit abzufangen und gezielt darauf zu reagieren. Dem Programmierer ist hier ein Werkzeug in die Hand gegeben, das man als eine Art Software-Interrupt Mechanismus ansehen kann. Wird eine vordefinierte oder vom Benutzer deklarierte Exception wirksam, so wird der normale Programmablauf abgebrochen und als Reaktion auf die Exception die Kontrolle an eine isolierte Folge von Anweisungen (handler) übergeben. Anschließend wird das Programm in einer übergeordneten Einheit regulär wieder aufgesetzt.
Zu den bisher vorgestellten, rein auf logische Konzepte abzielenden Eigenschaften bietet Ada auf der anderen Seite gewisse Möglichkeiten, die physische Darstellung von Daten vorzugeben bzw. zu beeinflussen. Dieser Aspekt, der wesentlich ist, gerade im Hinblick auf Systemprogrammierung und Realzeitanwendungen, wird durch das Konzept der Darstellungsspezifikation abgedeckt. Es gibt dabei verschiedene Spezifikationen, die es ermöglichen, Größe und Lage eines Datenobjekts im Speicher genau vorzuschreiben.
Abschließend sei noch bemerkt, daß Ada die Möglichkeit bietet, kontrolliert Assembler einzuschieben und daß Vorkehrungen zum Anschluß von anderen Sprachen getroffen worden sind.
Die hier vorgestellten Konzepte von Ada sind natürlich nicht prinzipiell neu, was sogar explizit bei der Sprachentwicklung gefordert worden war. Neu jedoch ist, daß diese Konzepte in einer Sprache zusammengefaßt sind und daß diese Konzepte nicht nebeneinander stehen, sondern miteinander integriert (orthogonal) sind. Gerade diese Orthogonalität aber trägt wesentlich zur Mächtigkeit von Ada bei.

IA