RAPPORT 84-28

Ontwikkelingen rond Ada

door

J. van Katwijk

Rapporten van de Onderafdeling der Wiskunde en Informatica

Reports of the Department of Mathematics and Informatics

Onderafdeling der Wiskunde en Informatica
Julianalaan 132, 2628 BL Delft
Postbus 356, 2600 AJ Delft

Department of Mathematics and Informatics
Julianalaan 132, 2628 BL Delft
P.O. Box 356, 2600 AJ Delft


mieke '82

CONTENTS

Ada is een gedeponeerd handelsmerk van DoD OUSDRE-AJPO

ABSTRACT

In dit rapport, samengesteld naar aanleiding van een voordracht van de auteur op 14 maart 1984 op het informatica colloquium van de VU te Amsterdam, wordt ingegaan op een drietal ontwikkelingen rond de programmeertaal Ada. Eerst komen wereldwijde ontwikkelingen rond de programmeertaal Ada ter sprake, vervolgens wordt een overzicht gegeven van de activiteiten op dit gebied ontplooid aan de Technische Hogeschool te Delft en tenslotte wordt, in een meer technisch gedeelte, een aspect behandeld van de implementatie van de Delftse Ada Subset, DAS.

1. Inleiding.

Dit rapport bestaat uit drie gedeelten. In het eerste deel zal worden ingegaan op de wereldwijde ontwikkelingen rond Ada. Het tweede deel geeft een korte impressie van de Ada activiteiten aan de TH Delft. Het derde deel is meer technisch van aard en geeft een inleiding in de structurering en beschrijving van data structuren in de Ada subset implementatie die aan de TH Delft wordt ontwikkeld.

Er wordt vanuit gegaan dat de lezer/toehoorder op de hoogte is van het bestaan van de programmeertaal Ada en van de oorspronkelijke achtergronden die tot de ontwikkeling van de taal hebben geleid.

Het is niet de bedoeling uitgebreid in te gaan op de al dan niet vermeende kwaliteiten van Ada. Ada zal zich zelf, tenminste in de toepassingsgebieden waarvoor de taal bruikbaar wordt verondersteld, moeten bewijzen. Het heeft geen zin daarnaast veel uitlatingen te doen over het al dan niet mooi zijn van bepaalde constructies of combinaties van zulke constructies.

2. Status van Ada.

Toen de "Green" language in 1979 als DoD standaard werd geaccepteerd en de naam "Ada" kreeg, verwachtte de wereld op korte termijn programmatuur in Ada te kunnen draaien. Weliswaar was aan de dikte van het reference manual te zien dat de definitie van de taal langer was dan de definitie van bijvoorbeeld PASCAL, toch zijn alle problemen behoorlijk onderschat. Nu, vijf jaar nadat Wichmann de historische woorden "Ada is green" [Wichmann] schreef is er slechts een handje vol gevalideerde[2] implementaties. Deze zijn, volgens [AJPOnews] (juli 1984):

- De NYU Ada/Ed implementatie. Een in SETL geschreven "semantisch model", bestaande uit een kleine 20k SETL regels. De implementatie draait op diverse hostsystemen, een VAX onder VMS en onder UNIX en een Amdahl onder UTS. G. Fisher een van de makers heeft (regelmatig) de woorden "If it moves it's fast enough" uitgesproken, een indicatie over de snelheid van de implementatie. Na de validatie van de implementatie zijn overigens nog enkele tientallen fouten in de implementatie aan het licht gekomen. Geconstateerd kan worden dat het niet triviaal is om te verifieren of een implementatie een gegeven taal werkelijk implementeert.
  Validatie datum was 11 april 1983.

- ROLM Corporation/Data General Corporation implementatie. ROLM Ada Compiler version 4.52 Host and targets: Data General MV4000, MV6000, MV8000, MV8000-II, MV10000 en ROLM MSE-800.
  De datum van validatie was 13 juni 1983.

- Gensoft Corporation (voorheen Systems Technology Center, Western Digital Corporation) implementatie.
  Host en target machines: Western Digital WD1600 Series Micro Engine
  Validatie datum was 9 augustus 1983.

- Telesoft Inc. compiler. Deze compiler is gevalideerd draaiend onder 68000 ROS operating systeem op een O-bus systeem en een LABTEK systeem.
  Datum van validatie was 21 mei 1984.

De Ada taal is, sinds zijn ontstaan in 1979, in aanzienlijke mate gewijzigd. In 1980 kwam het eerste "groene" document, waarna de "canvass" procedure werd aangespannen om tot een ANSI standaard te komen. Tijdens deze procedure werden meer dan 5000 commentaren op de taal gegeven.
De coordinatie van de ontwikkelingen rond het hele Ada gebeuren werd in handen gelegd van een buro, het Ada Joint Program Office, AJPO.
Op 18 februari 1983 kwam er een door ANSI gestandaardiseerde versie

2. Ada is pas Ada als DoD er Ada op gezet heeft. Een implementatie mag pas een Ada implementatie heten als ze gevalideerd is onder toezicht van DoD.

[LRM] uit. Een eerste poging om de ANSI standaard tot ISO "workitem" te verheffen is vertraagd door de zg. trademark affaire. Sommige ISO leden, waaronder Zwitserland, hebben problemen met het feit dat DoD zou moeten kunnen bepalen of een implementatie aan een door ISO gedefinieerde standaard zou voldoen of niet.
Toch is sinds april 1984 de internationale standaardisatie van Ada in volle gang, een ISO werkgroep, TC97/SC5/WG14, is belast met de voorbereiding van de standaard.
Ook is een nieuwe rationale in ontwikkeling. Daarin zouden een aantal, voor de buitenwereld onduidelijke, beslissingen verklaard kunnen worden.

3. Ada in Europa.

In het midden van de 70'er jaren werd een ESL (European System Language) studie begonnen onder auspicien van de Commissie van de Europese Gemeenschappen (CEC), kortweg de Commissie. De studie kende een resultaat dat nogal leek op het toenmalige ontwerp van de "green" language. Dit is niet zo verwonderlijk als men de "requirements" nagaat voor de ESL:

+ portabiliteit van software,
+ competitie tussen leveranciers,
+ verbeterde kwaliteit van software,
+ verbetering van software engineering methoden.

Gelet op de overeenkomsten tussen de requirements van de ESL en de DoD taal [en waarschijnlijk ook het feit dat het door DoD uitverkoren ontwerp, Green, van europese origine was] werd door de Commissie de ondersteuning aan de ESL ontwikkelingen gestaakt en support voor Ada gegeven.
Dit uiteraard zeer naar de zin van DoD, binnen DoD is vanaf het begin een sterke wil aanwezig geweest om van Ada een, met recht internationale, standaard te maken.

De huidige ondersteuning van Ada onder het zg. "multi annual data processing programme" is gebaseerd op een tweetal punten:

+ Ada voldoet in grote mate aan de requirements voor de ESL,
+ voor producten, "software componenten" is er na een internationale standaardisatie een grotere markt dan voor op ESL gebaseerde componenten. Door de Commissie wordt gedacht aan een sterkere eigen europese software componenten industrie. Momenteel is de invoer van soft- en hardware componenten binnen de EG een veelvoud van de bijdrage van de europese industrie aan de thuismarkt.

3.1 Feitelijke ontwikkelingen binnen de E.G.

Onder het huidige multi-annual programme ondersteunt de Commissie een aantal projecten, waarvan de belangrijkste zijn:

1. De portable Ada Compiler family, een europese "root" compiler ontwikkeld door een consortium bestaande uit CII/Honeywell Bull, Alsys (de firma van Ichbiah) en Siemens,
2. De Portable Ada Programming Support Environment, PAPS, een europese implementatie gebaseerd op de Stoneman requirements, uitgevoerd door een consortium bestaande uit Olivetti, Dansk Datamatik Center en Christian Rovsing

[LRM] uit. Een eerste poging om de ANSI standaard tot ISO "workitem" te verheffen is vertraagd door de zg. trademark affaire. Sommige ISO leden, waaronder Zwitserland, hebben problemen met het feit dat DoD zou moeten kunnen bepalen of een implementatie aan een door ISO gedefinieerde standaard zou voldoen of niet.
Toch is sinds april 1984 de internationale standaardisatie van Ada in volle gang, een ISO werkgroep, TC97/SC5/WG14, is belast met de voorbereiding van de standaard.
Ook is een nieuwe rationale in ontwikkeling. Daarin zouden een aantal, voor de buitenwereld onduidelijke, beslissingen verklaard kunnen worden.

Andere, door de Commissie ondersteunde projecten, zijn hoofdzakelijk voorstudies:

1. life cycle support voor Ada projecten (SDL en TECSI),
2. Ada en een multi-microprocessoir omgeving (SPL, TXT en CISE),
3. conversie van RTL/2 naar Ada (SPL en HSA),
4. grote numerieke bibliotheken in Ada (NPL en CWI).

Voor meer details wordt verwezen naar het bureau van de CEC[3].

3.2 Ada Europe.

Om binnen Europa in een iets breder verband te kunnen discussieren over de diverse aspecten van de ontwikkelingen rond Ada, is er al in 1980 een Ada Europe groep gevormd. Een van de initiatiefnemers was Brian Wichmann, tevens een van de leden van het ontwerpteam van de taal.
De groep is gevormd door personen en organisaties die, beroepshalve, geintereseerd zijn in de implementatie en het gebruik van Ada. Om zo'n groep werkbaar te maken zijn er subgroepen gevormd waarin de diverse aspecten van de taal of haar gebruik aan de orde komen. Momenteel zijn er subgroepen voor o.a.:

+ Education,
+ Formal Application semantics,
+ Environments,
+ Portability,
+ Numerics,
+ Implementation,
+ Validation.

De huidige vice voorzitter van Ada Europe is een nederlander, M. Boasson van HSA. De huidige voorzitter is K. Ripken van TECSI. Overigens zijn niet alle subgroepen even actief.
Ada Europe heeft momenteel een 100 tal leden, verspreid over geheel Europa. Een van de belangrijkste activiteiten is de jaarlijks terugkerende "joint Ada Europe/ AdaTec" meeting, dit jaar in juni, in

3. Dr. R. W. Meyer,
Commission of the European Communities,
DG-III
Rue de loi 200 Brussels, B-1049

Brussel.
De Commissie ondersteunt het werk van Ada Europe door reis en verblijfkosten van de leden voor de diverse bijeenkomsten te vergoeden.

3.3 Contacten Europa en DoD.

Een Ada Liason Organisation (ALO) is gevormd om de AJPO in alles wat met Ada betrekking heeft te adviseren en om een liason te implementeren tussen AJPO en andere belanghebbenden, onder andere in Europa.

3.3.1 Interessante europese implementaties. Binnen en buiten Europa wordt op verschillende plaatsten gewerkt aan implementaties van de taal Ada of subsets daarvan. De meest bekende europese implementatie pogingen zijn:

- De Karlsruhe implementatie. Een groep onder leiding van Prof. G. Goos op de universiteit van Karlsruhe heeft een volledige implementatie van Ada-80 gemaakt. Deze implementatie draait momenteel op grotere Siemens systemen, het is de bedoeling te bootstrappen naar o.a. 68000 gebaseerde systemen.
  De implementatiegroep is nu overgegaan van de universiteit van Karlsruhe naar het GMD[4] research lab Karlsruhe.

- De York implementatie. Een Ada implementatie is ontwikkeld door de groep van Prof. I. Pyle aan de universiteit van York in Engeland. Deze implementatie draait op VAX systemen en zal, als alles volgens de plannen verloopt, dit jaar ter validatie worden aangeboden.

Een zogenaamde matrix van implementaties wordt regelmatig gepubliceerd in Ada Letters, het blad van Ada-Tec. Een copie van zo'n matrix is als appendix opgenomen.

4. GMD: Gesellschaft fur Mathematik und Datenverarbeitung

4. Ada Programming Support Environments.

4.1 Requirements voor een APSE.

Een ontwikkeling die tenminste van even groot belang is als de ontwikkeling van de programmeertaal Ada, is de ontwikkeling van een **program development system** voor Ada: de Ada Programming Support Environment, kortweg de APSE.
Parallel aan de latere ontwikkelingen van de taal is gewerkt aan het opstellen van requirements voor zo'n Ada omgeving. Het meest bekende document waarin requirements zijn beschreven is het zg "Stoneman" document [Stoneman].

Karakteristiek voor de toepassingsgebieden van Ada zijn:

- grootschaligheid, veel ontwerpers, veel programmeurs en veel onderhoudsprogrammeurs,
- lange levensduur, programmatuur met een levensduur van 20 jaar is geen uitzondering (F104 starfighter bijvoorbeeld).

Het uitgangspunt voor een Ada Programming Support Environment is de ondersteuning die zo'n omgeving moet kunnen bieden gedurende alle fasen in de lifecycle van een project. Dit betekent dat een APSE over 20 jaar ondersteuning moet bieden aan een project waarin gebruikt gemaakt wordt van programmatuur die nu ontwikkeld wordt.
Bij alle discussies over Ada en APSE's moet zowel de schaal van de software als de lifetime van de te ontwikkelen programmatuur in beschouwing worden genomen. Vergelijk dat met de "wegwerp" filosofie die veelal bij UNIX[5] wordt gehanteerd [UNIX].

De problemen die uit de grootschaligheid voortvloeien zijn te lijf te gaan met:

- project management,
- software management.

Denk bij dit laatste aan ondersteuning op het gebied van version control en configuration management.

De problemen die uit de langere lifetime van de programmatuur voortvloeien (regenereerbaarheid van oude(re) versies) zijn op te lossen door:

- software management,
- portabiliteit.

Portabiliteit is een sleutelwoord, denk aan een stuk programmatuur dat nu wordt ontwikkeld op systeem X voor systeem Y en waarin de klant na

5. UNIX is een handelsmerk van Bell Labs

20 jaar iets gewijzigd wil hebben. Dat betekent dat in het ergste geval de gehele ontwikkelomgeving portable moet zijn, tenzij de maker van de software bereid is gedurende de gehele lifetime van die software een systeem van merk X in huis te hebben. Als vergelijking kunnen we denken aan de computersystemen van 20 jaar geleden. De 360 serie was net geannonceerd, de IBM 7090 en de TR4 waren toen moderne systemen.

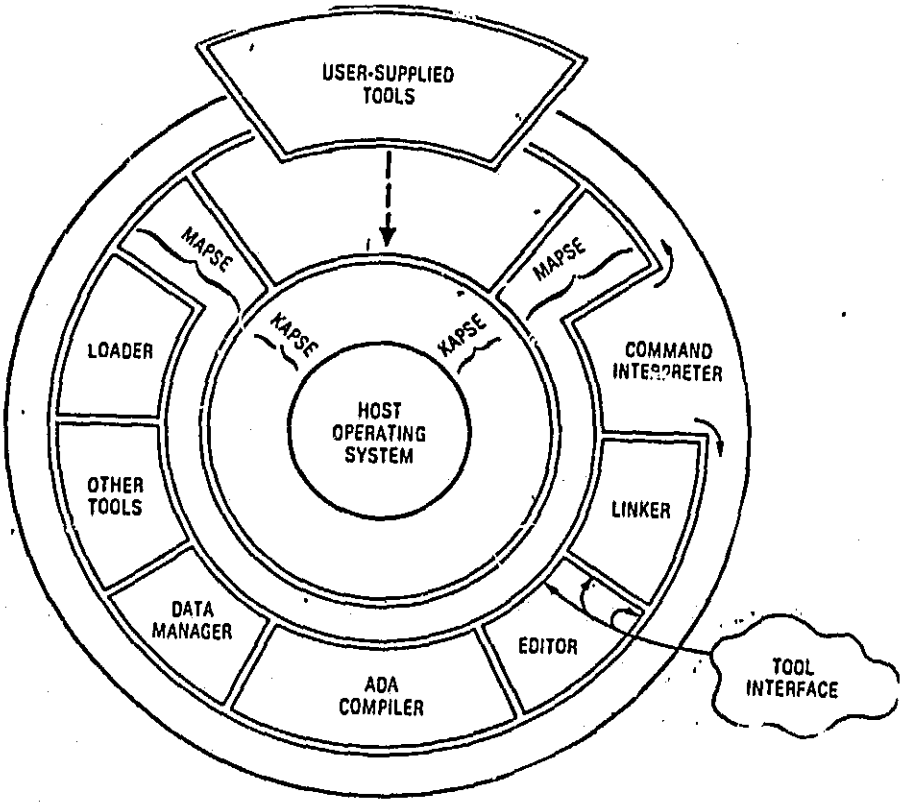
Bij de discussies over de APSE komt naar voren dat zo'n APSE ondersteuning moet bieden voor:

+ portabiliteit,
+ software management.

Voor zover mogelijk moet een APSE ook tools kunnen ondersteunen voor

+ project management.

De architectuur van APSE's kent, om de diverse aspecten van portabiliteit te ondersteunen, een tweetal nivo's, de KAPSE en de MAPSE, zoals uit het volgende schematisch overzicht blijkt.



Architectuur van een APSE

Ada is primair ontwikkeld voor de implementatie van embedded systems.

Het systeem waarop de programmatuur wordt ontwikkeld is niet noodzakelijk het systeem waarop de ontwikkelde programmatuur wordt gedraaid. (Host/target approach). We hebben dan te maken met een drietal portabiliteits problemen:

- portabiliteit van de ontwikkelomgeving, rehosting.
- portabiliteit van toolsets op een specifieke ontwikkelomgeving,
- portabiliteit van de ontwikkelde programmatuur voor andere targets, re-targeting.

Het **KAPSE** nivo is het best vergelijkbaar met het nivo dat geboden wordt door de standard I/O library onder UNIX. De KAPSE biedt een interface naar functies die niet in Ada worden uitgedrukt en biedt het runtime support voor Ada programmatuur.

Het KAPSE nivo staat centraal bij het porteren van de gehele omgeving, en het overdragen van losse tools.
Net als in UNIX voor een programma dat alleen gebruik maakt van functies die door de standard I/O library geboden worden, is in Ada een programma dat alleen gebruik maakt van KAPSE functies in principe overdraagbaar.

Het **MAPSE** nivo is het nivo van de programmeursportabiliteit. Het wordt gevormd door een Minimale Toolset waarmee de MAPSE zelf kan worden onderhouden. Door de open endness-heid van de omgeving is elke gebruikersgroep van zo'n MAPSE in staat een omgeving te creeeren die meer is gericht op de aard van het project en de werkwijze van de gebruikers. Als een standaard MAPSE als uitgangspunt genomen wordt kan zo'n ontwikkelde omgeving in zijn geheel getransporteerd worden naar een andere host die dezelfde MAPSE ondersteunt.
De ondersteuning die op dit nivo geboden wordt voor project management en de eerste (ontwerp) fasen in de lifecycle van de programmatuur bestaat uit niet meer dan een text editor.

4.2 De database.

Buiten de ondersteuning die Ada zelf vraagt, bijvoorbeeld voor de ondersteuning van separate compilatie, biedt UNIX veel van wat voor een APSE nodig is. Een van de min of meer karakteristieke verschillen treedt op bij de database (UNIX: filesysteem).
Terwille van de ondersteuning van de grootschaligheid en de lange levensduur voor de ontwikkelde programmatuur wordt:

- configuration management, en
- history en version control,

belangrijk.
Anders dan bij het UNIX filesysteem zijn de objecten in de database op meerdere wijzen met elkaar verbonden, getypeerd en is het bestaan van meerdere versies van een object niet uitgesloten.

Met name "history control" legt zware eisen op aan de database, van elk object moet, volgens Stoneman, de hele history bewaard blijven. Dat wil zeggen dat te achterhalen moet zijn:

- welke versies van welke modules gebruikt zijn bij het genereren van programma Y voor target Z,
- welke editor was gebruikt bij het invoeren van deze modules en welke text- en commando regels waren gebruikt,
- welke compiler was gebruikt met welke command options,
- (transitieve afsluiting:) welke compilerversie was gebruikt om de compilerversie te genereren waarmee module A voor target Z gecompileerd is.

Zo'n database wordt snel groot!

Voor uitgebreidere informatie wordt de lezer verwezen naar [Stoneman], [Computer].

Samenvattend:
De APSE heeft veel van UNIX weg:

- open ended,
- een KAPSE laag als kernel extensie van Ada programmatuur,
- combinatie van tools door middel van piping alsmede redirection van input en output zijn mogelijk.

Het wezenlijke verschil tussen een APSE en UNIX zit in de grootschaligheid:

- UNIX is, in het bijzonder wat betreft de functionele ondersteuning op kernel nivo, gericht op de individuele C programmeur, voor kortlopende projecten. Er zijn ad hoc tools, in het bijzonder Make en SCCS die kunnen bijdragen in het software management,
- APSE's zijn, in het bijzonder wat betreft de functionele ondersteuning op kernel nivo, gericht op de groep van Ada programmeurs, bezig met langlopende projecten. (uitgebreide faciliteiten voor history control en configuration management.)

4.3 APSE's in ontwikkeling.

Op een aantal plaatsen wordt aandacht besteed aan de feitelijke ontwikkeling van APSE's. De "grote" ontwikkelingen zijn die welke ondersteund worden door sponsors:

- ALS en de AIE in de V.S,

+ PAPS in Europa.

Buiten deze zijn nog een aantal andere pogingen onderweg. We zullen ons hier echter beperken tot de genoemde drie.

4.3.1 De **ALS.** Het Ada Language System, ALS in de wandeling, is een ontwikkeling van Softech (de indiener van het Blue language voorstel) in opdracht van de VS "Army". Host is een VAX/VMS systeem, initieel zijn er diverse targets, w.o. de host zelf.

Omdat de ALS het eerste werkende systeem is dreigt het een de-facto standaard binnen DoD te gaan worden. De "Army" heeft in elk geval zelf al de uitspraak gedaan dat voor ontwikkelingen ten behoeve van dit krijgsmachtonderdeel de nieuwe versie van de ALS, de ALS/N, gebruikt moet worden zodra die leverbaar is. Indien nodig kunnen de diverse program offices zelf de rehosting en/of retargeting van de ALS/N autoriseren.
Ook de "Navy" lijkt voor de ALS/N overstag gegaan te zijn, de "Airforce" heeft nog geen beslissing genomen.
Ten behoeve van rehosting van de huidige versie van de ALS is deze voor derden beschikbaar. Overigens is de huidige versie van de ALS nog niet volledig, ook is de Ada compiler nog niet klaar. Verwacht wordt dat de ALS/N in 85/86 beschikbaar komt.
Een globale indruk van de ALS, op basis van de beschikbare documenten, toont dat het ontwerp sterk beinvloed is door UNIX. De belangrijkste tools zijn:

1. Command Language Interpreter,
2. DBMS,
3. Configuration management tools,
4. Ada Compiler met diverse back-ends
5. assemblers, linkers en loaders,
6. stub generatoren,
7. pretty printers.

Text editoren en formatters worden voorlopig van het hostsysteem, VAX/VMS, geleend.

4.3.2 De **AIE.** De AIE, de Ada Integrated Environment, is een ontwikkeling van Intermetrics (Red language) in opdracht van de "Airforce". De AIE zou moeten werken op IBM 4341 en PE 8/32 computersystemen van de "Airforce". Om aan te geven hoe de opdrachtgevers zelf over de belasting van zo'n systeem denken, zijn de volgende requirements indicatief:

- De Ada compiler/linker mag voor het compileren van een willekeurig Ada programma niet meer dan 512 kb main memory gebruiken.

- Het residente gedeelte van de KAPSE mag niet groter dan 256 Kb zijn,

- Elapsed time voor de compilatie en linkage van een enkel 1000 regelig Ada programma mag niet groter zijn dan 4 minuten,

- CPU time voor deze activiteit mag niet groter zijn dan 1 minuut.

Deze laatste twee eisen onder de volgende voorwaarden: Een IBM 4341 met 4Mb, vier gebruikers ingelogged waarvan er een de Ada compiler gebruikt en de andere drie of de CLI of de editor.

De huidige status van de AIE is tamelijk onduidelijk.

4.3.3 De **PAPS.** De CEC financiert twee grote projecten in het Ada traject, de ene is de europese rootcompiler, de andere is de PAPS, de Portable Ada Programming Support Environment.
De PAPS wordt ontworpen en gebouwd door een consortium bestaande uit Olivetti, Dansk Datamatik Center en Christian Rovsing. Het is de bedoeling om een MAPSE te ontwerpen en te implementeren op o.a. mini computers. De opzet was om de PAPS april 1984 klaar te hebben. Die datum is niet gehaald, volgens de laatste berichten wordt de oplevering nu september/oktober verwacht.

4.4 **De CAIS ontwikkelingen.**

Initieel zijn binnen de DoD organisaties een tweetal, grote, contracten gegund voor de ontwikkeling van een APSE (ALS, AIE). Al snel bleek dat de verschillende ontwerpen divergeerden en dat er dus van de zo fel begeerde portabiliteit tussen APSE's weinig terecht zou komen. In 1981 is daarom binnen DoD een team geformeerd, onder auspicien van de "Navy", om bepaalde portabiliteitsaspecten te onderzoeken. Dit team luistert naar de naam KIT, Kapse Interface Team. Het wordt bijgestaan door een ander, niet DoD, team, KITIA. Het achtervoegsel "IA" staat hier voor "Industry and Academia".

Het KIT met ondersteuning van KITIA moet het probleem van "transportability" en "interoperability" onderzoeken. De problemen die gepaard gaan met het gebruik van tools op meerdere APSE's en, veel moeilijker, uitwisselbaarheid van database objecten worden door deze teams geinventariseerd en zoveel mogelijk opgelost.

Eind 1982, begin 1983, bracht dit team een informele definitie uit van de SIS, de "Standard Interface Set". In september 1983 kwam een eerste versie van de CAIS, de "Common APSE Interface Set" [CAIS] uit. Daarop zijn alweer een aantal wijzigingen.

De CAIS is gedefinieerd in de vorm van een aantal packages. Elk Ada programma dat zich aan de door de CAIS geboden definitie

conformeert, en verder geen host of target afhankelijkheden bevat, moet overgedragen kunnen worden tussen alle DoD APSE's.

5. Ada in Nederland.

Na het verschijnen van de preliminary versie van het Ada reference manual [SIGPLAN], werd in april 1980 een eendaags symposium over Ada gehouden, georganiseerd door de sectie SP van het NGI. De belangstelling was zo overweldigend dat door de sectie voorzichtige pogingen in het werk zijn gesteld om een werkgroep over/voor Ada in te stellen.
Na enige vertraging werd zo'n werkgroep inderdaad opgericht, op 2 april 1982 hield de "Ada werkgroep Nederland", werkgroep van de sectie SP van het NGI, de oprichtingsbijeenkomst.
De doelstellingen van de werkgroep waren, en zijn:

+ het vormen van een forum voor discussies over alle aspecten van de taal en haar omgeving,

+ het ontsluiten van documentatie over Ada.

De tweede doelstelling is zeker bereikt. Door de werkgroep zijn contacten gelegd met informatie (data) producerende instanties zoals:

+ de sectie SP van het NGI,

+ NNI/SC5,

+ Ada Europe,

+ DoD via mailing lists,

+ KITIA.

Bovendien zijn contacten gelegd met de "industrie".

De eerste doelstelling blijkt nog niet geheel opportuun te zijn, het aantal mensen dat daadwerkelijk met Ada bezig is is tamelijk beperkt in Nederland.

Geinteresseerden kunnen zich wenden tot de auteur, voorzitter van de werkgroep, of tot de secretaris[6] van de werkgroep.

---

6. J van Someren
p/a ACE bv
Nz Voorburgwal 314.
1012 RV Amsterdam.

6. Ada in Delft.

6.1 Inleiding.

Aan de TH te Delft wordt al geruime tijd aandacht besteed aan de ontwikkelingen rond Ada. Dit heeft zich o.a. geuit in een Ada cursus en inbedding van Ada in colleges over programmeertalen en software engineering. Bovendien wordt, meer in het kader van onderzoek dan onderwijs, aandacht geschonken aan implementatie aspecten van Ada, instructieset architecturen, compiler structuren, en aan diverse aspecten van Ada Programming Support Environments.

Ook bij het IBBC-TNO bestaat veel belangstelling voor Ada, bij TNO lopen twee projecten die aan Ada gerelateerde activiteiten met zich meebrengen.

Momenteel wordt, gezamenlijk door TH Delft en IBBC-TNO, gewerkt aan:

- de ontwikkeling van een eenvoudige doch doelmatige programming support environment.
- een portable en retargetable compiler voor DAS, de Delft Ada[7] Subset.

Een tamelijk grote handicap bij deze activiteiten is de omvang van het werk. Een "production quality" Ada compiler wordt geschat op een inspanning van enkele tientallen manjaren, manjaren die wij niet beschikbaar hebben.
Het werk aan alle deelprojecten wordt momenteel uitgevoerd door een vijftal afstudeerstudenten, K. Dijkstra, E. van Konijnenburg, M. de Niet, R. van Liere en H. Toetenel, een onderzoeker bij IBBC-TNO, C. Barkey, een TNO staflid, R. Westermann, en twee medewerkers van de vakgroep Informatica, A. Hartveld en de auteur.

Een belangrijke doelstelling bij deze activiteiten is het ontwikkelen van een bruikbare subset compiler, tesamen met een subset van de CAIS, een subset die voldoende functionaliteit biedt om een goede DAS omgeving te kunnen realiseren.

6.2 De DAPSE.

Een eenvoudige programming support environment, de **Delft Ada Program Support Environment**, DAPSE, is in ontwikkeling. Een pre-release die momenteel in gebruik is, is noodgedwongen ontstaan uit de noodzaak een program library te gebruiken voor de ondersteuning van separate compilation.

De huidige versie van de DAPSE is een gesloten omgeving waarbij de gemanipuleerde objecten compilation units of text units zijn. Afhankelijk van indicatoren die bij de naam van zo'n object wordt meegegeven wordt de source versie, een of andere intermediaire versie,

7. Ada laat overigens geen subsetting toe.

een object module of een executable bedoeld.
De relaties die er bestaan tussen de verschillende compilatie units en de verschillende representaties van dezelfde compilatie unit worden bijgehouden in de DAPSE.

Voorbeeld van een (stukje) DAPSE sessie

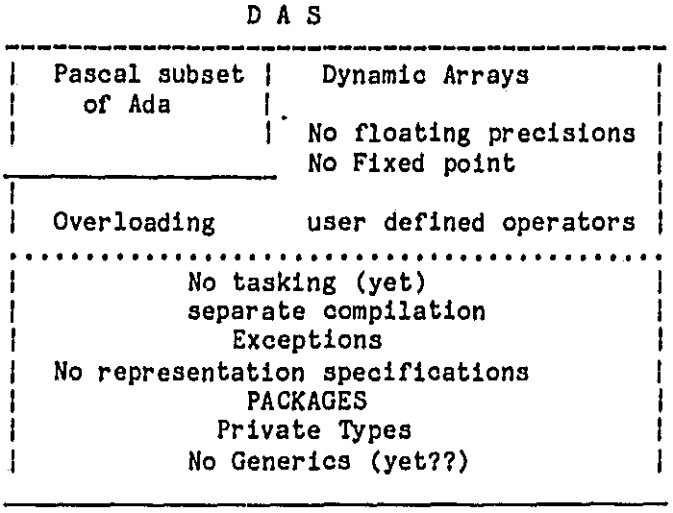
```
%dapse jan
        DELFT PROGRAMMING SUPPORT ENVIRONMENT

->edit x
?x
a
with my_unit; use my_unit;
with text_io; use text_io;

package x is
        type kleur is (rood, wit, blauw);
end;
 .
 w
 q
->compile x
```

In dit voorbeeld wordt een klein package gedefinieerd, met de naam x.
Na invoering in de DAPSE is de compilatieunit bekend onder de naam x.
De relaties tussen de unit "x" en de units "my_unit" en "text_io" worden afgeleid en bewaard.

7. Implementatie van een Ada subset.

In het kader van de Ada georienteerde activiteiten wordt aandacht besteed aan de implementatie van een Ada subset compiler. Doel van de implementatie poging is tweeledig, enerzijds levert zo'n implementatiepoging veel kennis en inzicht op op het gebied van compiler technologie, anderszijds is het resultaat een bruikbare implementatie van een systeem implementatietaal.
Het werk aan de implementatie is begonnen op een PDP-11, momenteel wordt dit werk afgerond op een Geminix[8], een M68000 gebaseerd computersysteem.

DAS, Delft Ada Subset is een subset van Ada die groot genoeg is om een compiler voor DAS en voor Ada in te schrijven. De relatie tussen PASCAL, Ada en DAS komt, globaal, naar voren uit het volgende schema:



Schematisch overzicht PASCAL, DAS en Ada

De voorvoegsels "No" in dit schema zijn van toepassing op DAS, zonder deze voorvoegsels wordt naar Ada gerefereerd.
Het zal evident zijn dat Ada een complexere taal is dan bijvoorbeeld PASCAL. Ook DAS is nog een orde complexer dan PASCAL.
Deze grotere complexiteit uit zich in:

1. Complexere analyse fases in de compiler,

2. Complexere runtime organisatie.

Bij dat laatste moeten we voorzichtig zijn, runtime efficiency is belangrijk voor een bruikbare systeemimplementatietaal.

8. Geminix is een handelsmerk van IBBC-TNO.

7.1 Compiletime aspecten.

In deze sectie geven we een kort overzicht over de verschillende fases van de DAS compiler.
Elementen die aan de orde komen zijn:

+ parsing,
+ static semantics,
+ high level intermediaire representatie,
+ low level intermediaire representatie.

De modellering van het runtime systeem wordt niet behandeld. Een detail, de modellering van data en datadescriptoren wordt later behandeld.

7.1.1 **Parsing.** Een logisch gevolg van een grote taal is een groot aantal regels bij de context vrije syntax (tenminste als we naar een redelijk nauwkeurige benadering van de taal streven). Alleen al uit het feit dat er met enige regelmaat beschrijvingen van een context vrije grammatica, liefst een LALR(1) grammatica, gepubliceerd worden, mogen we afleiden dat het formuleren van zo'n syntactische beschrijving niet geheel triviaal is.
Bij de DAS compiler worden ca 310 productieregels gebruikt om een ambigue LALR (1) [9] beschrijving te geven die door YACC [YACC] in een parser wordt omgezet. Bij de syntactische specificatie wordt gebruik gemaakt van een eenvoudige preprocessor om de relaties tussen de attributen van de diverse grammarsymbolen te specificeren [YACCPREP].

Het gebruik van een door YACC gegenereerde parser in een compiler heeft een probleem, het vermogen om te recoveren van syntactische fouten is, zacht gezegd, nogal mager.
Een betrekkelijk grote hoeveelheid werk is er ingaan zitten om de syntactische error **repair** [10] van de compiler op een redelijk nivo te brengen.

7.1.2 **Static semantics.** De analyse van de static semantics (ook wel context condities genaamd) valt globaal uiteen in drie, niet disjunkte, delen:

1. implementatie van scope en visibility regels,

---

9. Ik ben mij ervan bewust dat dit een contradictie is, ambigue grammatica's zijn niet LR. Gebruik wordt echter gemaakt van het "vermogen" van YACC om ambiguiteiten "op te lossen".

10. We streven repair na i.p.v. recovery zodat binnen de routines die door de parse worden aangestuurd (bijna) geen rekening behoeft te worden gehouden met syntactische gebruikersfouten.

2. type checking,

3. evaluatie van statische expressies.

We zullen ons beperken tot een enkele voorbeeld constructie om te tonen dat dit soort problemen bestaat.

```
package kleuren is
type t1 is (rood, wit, blauw);
        type t2 is (rood, wit, blauw);
        type t3 is (rood, wit, blauw);
        type t4 is (rood, wit, blauw);
        type t5 is (rood, wit, blauw);
        type t6 is (rood, wit, blauw);

        function "+" (a, b : t1) return t1;
        function "+" (a, b : t2) return t2;
        function "+" (a, b : t3) return t3;
        function "+" (a, b : t4) return t4;
        function "+" (a, b : t5) return t5;
        function "+" (a, b : t6) return t6;

        function "-" (a, b : t4) return t4;
        function "-" (a, b : t5) return t5;

        procedure P (a: t1);
        procedure P (a: t2);
        procedure P (a: t3);
        procedure P (a: t4);
end;
use kleuren;
        P (((rood + rood) - (rood + rood)) + (rood + rood));
```

Welke P en welke "rood"'s worden in deze procedure call gebruikt.

7.1.3 **High level intermediate representation.** Na de analyse fase van de compiler is het programma getransformeerd in een high level tree. De tree structuur lijkt nogal veel op de grote broer DIANA [DIANA]. Alvorens deze structuur verder te vertalen wordt zij eerst geattribueerd. Informatie over de adressering van runtime objecten wordt aan alle knopen, waar relevant, toegevoegd.
We zullen ons hier beperken tot een voorbeeld.

```
4    rectype                          record type definitie
       flags =>
       tag => R                       naam R
       enclunit => (   23,    1)      encoding van een pointer
       REC_alloc =>    1              descriptor globaal
       REC_flags => --V-2
       REC_offset =>   28
       REC_nform =>    2              twee discriminanten
       REC_vdsize =>   28             value descriptor grootte 28 bytes
       REC_nflds =>    9              aantal velden 9
       REC_ninits =>    0             aantal initialisaties 0
       REC_npaths =>    4             aantal layouts 4
       REC_maxpfld =>    6
       REC_vsize =>    0
           Enkele record knoop
```

Deze recordknoop is de beginknoop van het recordtype R op blz. 31. Merk trouwens op dat zo'n ASCII codering van een intermediaire representatie nogal snel in omvang toeneemt.

7.1.4 **Low level tree representatie.** Als low level tree representatie is gekozen voor de intermediaire code van de Portable C compiler [PCC]. De linearisering van de code is gebeurd, goede doelcode voor expressies moet nog worden gegenereerd. Voor dat doel zijn er diverse backends beschikbaar, o.a. voor de PDP-11, 68000, VAX, en EM.

Wat betreft ons werk zijn we dus na deze linearisering klaar, de low level bomen vormen de target code van de DAS compiler.

7.1.5 **EM.** EM code is ontwikkeld aan de VU door de groep van Prof. Tanenbaum [EM]. Als experiment wordt de DAS compiler zo geparametrizeerd dat deze in staat is, alweer via een backend van de Portable C compiler, om EM code te genereren. Deze EM code kan dan doorvertaald worden naar bijvoorbeeld 68000 code waar ook direct code voor kan worden gegenereerd. Dit experiment moet leiden tot inzichten in het gebruik van zowel EM als van de intermediaire code van de Portable C compiler als tussentaal.

7.1.6 **Status van de compiler.** De DAS compiler is initieel opgezet als een vijf pass compiler op een PDP-11. De functies van de passes waren resp. contextvrije parse, naam analyse, type checking, storage allocatie en code linearisatie. De compiler is, al geruime tijd, in staat om een subset van onze Ada subset, te vertalen. Eerst naar PDP-11 code, later naar 68000 code. De software, die initieel is ontwikkeld op een PDP-11 is overgehaald naar een 68000 gebaseerde machine en wordt aangepast aan de beschikbaarheid van een grotere adresruimte. Dit komt in feite neer op het reduceren van het aantal passes.
Het aantal codegeneratie passes[11] is tot een gereduceerd, de laatste hand word gelegd aan een front end in een enkele pass.

11. "pass" is hier een totale doorgang over de brontekst, lokaal worden veel meer scans uitgevoerd, bijvoorbeeld bij de

De eerste release van de DAS compiler met een DAPSE komt deze zomer beschikbaar. Momenteel worden de diverse delen van de compiler debugged.

7.2 Runtime aspecten.

Ook voor een Ada subset is een zekere mate van runtime efficiency gewenst. Runtime efficiency en compiletime complexiteit gaan meestal hand in hand, we zoeken naar een redelijke balancering.

Een belangrijk uitgangspunt daarbij is het definieren van een eenvoudig en compleet runtime model dat een efficiente implementatie niet uitsluit.
Het runtime model voor DAS valt uiteen in een drietal elementen:

- target instructie set,
- globale geheugen en stack organisatie,
- het data descriptie model.

Over de eerste twee aspecten zullen we kort zijn. Als instructie set wordt, zoals eerder aangegeven, de intermediaire code van de portable C compiler gebruikt.
De globale geheugenorganisatie is zoals te verwachten op een hedendaagse micro of minicomputer, een stack groeit de ene kant op, een heap de andere kant. De stack elementen zijn activeringsrecords. Daar niet alle dataobjecten statisch van lengte zijn is de organisatie van een activeringsrecord wat ingewikkelder dan in bijvoorbeeld PASCAL.
Bij de structurering van zo'n activeringsrecord is gestreefd naar een zo groot mogelijke overeenkomst met de activeringsrecords in C. Het is mogelijk om vanuit een DAS procedure een C functie aan te roepen zonder extra overhead.

Alhoewel de structurering van zo'n activeringsrecord aardige elementen bevat zullen we daar niet op ingaan. In plaats daarvan zal ingegaan worden op het model dat gekozen is voor de descriptie van data.

---

typechecking.

8. DAS data descriptie model.

In Ada en DAS hebben de data objecten meer vrijheidsgraden dan in bijvoorbeeld PASCAL. Bij arrays betekent dit dat de grenzen niet (noodzakelijk) meer in compiletime bekend zijn, bij records betekent dit dat de record elementen niet noodzakelijk een compiletime bekende grootte hebben. Dit tesamen met de wens om veilige variant records te hebben heeft geleid tot de ontwikkeling van een eenvoudig en doelmatig datadescriptie model.

Het model is beschreven in [DASDESC] en [DOUBLET]. Het is geimplementeerd op een VAX als deel van een interpretator voor DAS [DASINTER], en als deel van de compiler op een PDP-11 en de 68000 [BACKEND].

Alvorens in te gaan op het model voor de oplossing, eerst een paar uitgangspunten.

+ Bij de ontwikkeling van het model werd uitgegaan van een PDP-11 met, in hedendaagse termen, een beperkte adresruimte. Opslag van gegevens voor de datadescriptie moest derhalve behoorlijk efficient zijn.

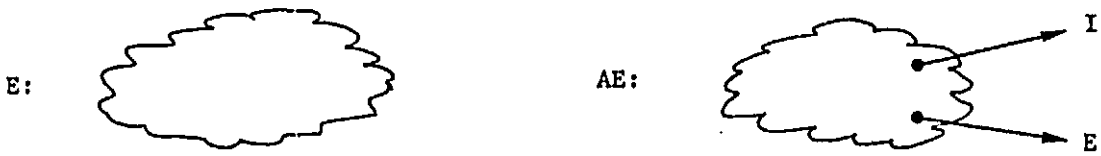
+ Platte data opslag. Het leek een redelijk uitgangspunt om alle datastructuren, hoe complex ook gedefinieerd, plat, gelineariseerd, op te slaan. Zeker voor assignments van multiple values heeft dit voordelen. Merk overigens op dat deze voordelen niet altijd aanwezig hoeven te zijn bij vergelijking van multiple values als gevolg van de alignment van componenten.

+ Automatische initialisatie en constraint checking moest mogelijk zijn. Om de hoeveelheid code te reduceren werd besloten om bepaalde "onder water handelingen", zoals bij automatische initialisatie van record componenten, over te laten aan een, lokaal zeer intelligent, runtime support systeem.

8.1 Oplossingsmodel.

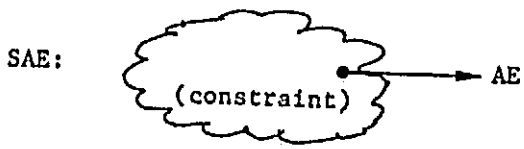
Uitgangspunt bij het model is dat voor elke elaboratie van een typedefinitie of een constraint er een descriptor is in runtime die de typedefinitie of constraint implementeert.
Als een type definitie of een constraint verwijst naar een ander type of constraint, dan komt dat in de corresponderende descriptor tot uitdrukking door een pointer van de ene descriptor naar de andere.

type E **is** ... ;



type AE **is array**(I **range** <>) **of** E;

De bijbehorende descriptor heet **type descriptor.** Als, zoals hier een typedefinitie parameters bevat is zij niet zonder meer bruikbaar voor de beschrijving van variabelen en values van het type.



subtype SAE **is** AE (constraint);

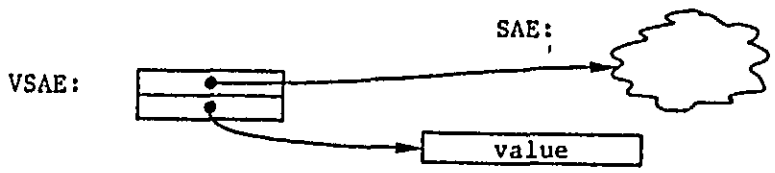
De descriptor voortkomend uit een expliciete of impliciete subtype definitie wordt **value** descriptor genoemd.

Verder is er in het model voor elke variabele, hetzij impliciet, hetzij expliciet, een **doublet** die de variabele implementeert. Dus:

VSAE: SAE;

wordt gerepresenteerd in het model als:

VSAE:

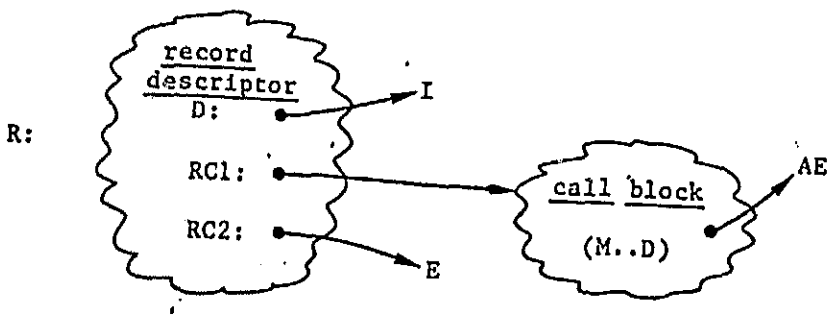


Bij een operatie als indexing op de variabele VSAE wordt een doublet gecreeerd, waarna de structuur wordt:

In het bovenstaande voorbeeld gingen we er, gemakshalve, vanuit dat we steeds als we een descriptor nodig hadden deze reeds bestond, zoals in PASCAL. Een extra complicatie wordt gevormd door de parametrizatie die o.a. bij records kan optrden. In het geval van een definitie

```
type R (D: I) is
   record
        RC1 : AE (M .. D);
        RC2 : E;
   end record;
```

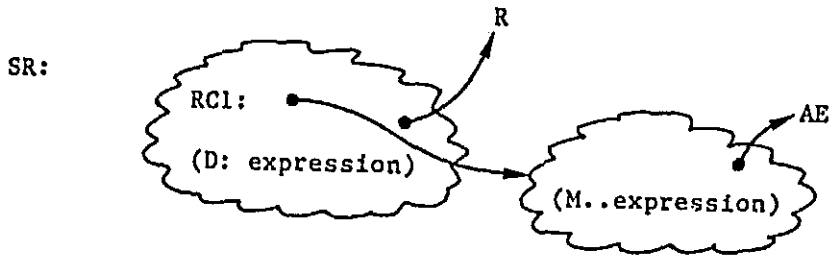
kunnen we in de descriptor die de definitie van R implementeert niet verwijzen naar de descriptor van de component RC1, de descriptor van de laatste bestaat eenvoudig nog niet.
De oplossing voor dit probleem is gegeven in [CALLBLOCK], we kunnen voor RC1 geen descriptor maken maar wel een indicatie hoe we te zijner tijd een descriptor moeten maken. We maken een structuur, een **call block**, waarin we alle relevante informatie over de te creeeren descriptor zetten.
Een descriptor voor typedefinitie R wordt:



Als we een constraint op R leggen, zoals in:

subtype SR **is** R (expressie);

wordt de waarde van de parameter D bekend en kunnen we een descriptor maken voor de component RC1. Merk op dat elke keer als zo'n constraint op R gelegd wordt, een nieuwe descriptor voor RC1 gemaakt moet worden. We moeten ons de descriptor van RC1 dan ook voorstellen lokaal ten opzichte van de descriptor voor SR.



Schematische voorstelling descriptor SR

8.2 Implementatie implicaties van het model.

In het model wordt voor elke variabele, hetzij expliciet hetzij impliciet, een doublet gemaakt. In de, half afgekomen, boom interpretator voor DAS [DASINTER], werd dit model letterlijk geimplementeerd. Elk tussenresultaat van een operatie werd ook werkelijk als een doublet geimplementeerd.

Een van de aardige aspecten van het model is dat de implementatie partieel door de code generator en partieel door het runtime support systeem plaats vindt. De codegenerator kan zijn kennis benutten om een gedeelte van de informatie die het model verschaft, in compile time te verwerken zodat de informatie in runtime niet meer nodig is.

Een van de meest voor de hand liggende voorbeelden is een "gewone" variabele. De compiler weet:

- waar de waarde van de variabele te vinden is, althans de adresfunctie is bekend,
- waar de descriptor van de variabele, of de waarde van de variabele, te vinden is, ook daarvan is tenminste een adresfunctie bekend.

Voor alle optredens van de variabele die door de compiler worden vertaald, is de compiler in staat het bestaan van het doublet te simuleren. In plaats van het doublet te genereren kunnen steeds de accessfuncties voor de waarde of voor de descriptor worden gegenereerd.
Voor "gewone" variabelen wordt nooit een runtime-doublet gemaakt.

Als voorbeeld nemen we een indexing operatie, eerst eenvoudig daarna in een gecompliceerdere context.

A (expr)

waarbij A een gewone variabele is gedefinieerd als:

```
type my_array is array (integer range <>) of integer;
A : my_array (1 .. 10);
```

In dit voorbeeld zijn de adressen van zowel de array data als de descriptor bekend, indexering is nu eenvoudig:

```
intermediaire code voor A(1) := 1

58      4                       assign
13      0       4               deref
6       24                      sub
8       24                      add
94      0       14      24      reg 0
4       64      0       4       64
11      4                       mult
4       4       0       24      4       elementsize
8       24                      sub
4       1       0       4       1
13      0       4               deref
6       24                      plus
4       0       0       24      _Mce3   descriptor adres
4       16      0       4       16
4       1       0       4       1
```

De elementgrootte is in compiletime bekend en dus inline gecodeerd. Als deze grootte niet in compiletime bekend is dan is ze uit de runtime descriptor te halen.

Als complexer voorbeeld nemen we een combinatie van een functie aanroep en een indexing operatie.

```
function moeilijk (a : integer) return My_type;
```

Eenvoudshalve nemen we even aan dat het type van de resultaat value van de functie "moeilijk" een of ander array type of subtype is.

```
moeilijk (3)(index_expressie)
```

De evaluatie in het model is als volgt:

1. evalueer de functie "moeilijk", deze functie keert terug met een doublet.

2. doe de indexing. D.w.z. maak een nieuw doublet met daarin een verwijzing naar de berekende plaats van de waarde en de bijbehorende descriptor.

3. neem de inhoud van de berekende locatie.

Bij de implementatie van het model moeten we twee gevallen onderscheiden:

+ het return type van de functie "moeilijk" verschaft informatie over de grenzen, d.w.z het is een subtype,

+ het return type van de functie is een unconstrained array type.

In het eerste geval kunnen we het model verder door de compiler laten implementeren dan in het tweede geval. Van elk der gevallen zullen we een globale uitwerking geven.

Als de functie "moeilijk" een array**sub**type als resultaattype heeft, dan is weliswaar in compiletime het adres van de array data niet bekend maar wel het adres van de descriptor.[12] In executietijd komt de functie terug met een pointer naar de waarde.

De compiler kan hier nog een aanzienlijk deel van het doublet simuleren.

Het meest ingewikkelde geval treedt op als de return value unconstrained is, bijvoorbeeld:

function moeilijk (a : integer) **return** AE;

In compiletime is nu noch het adres van de descriptor van het resultaat, noch het adres van de resultaat value zelf bekend. D.w.z. dat de compiler nog maar weinig van het doublet kan simuleren.
Wel kunnen we aangeven waar de losse elementen van de doubletten zich bevinden. Ook de structuur van de descriptor is bekend, d.w.z. de indexeringsoperatie kan worden uitgedrukt in termen van een indirecte referentie naar de value en een indirecte referentie naar de descriptor van de value.

De descriptor voor het resultaat van de indexering is altijd weer in compiletime bekend. D.w.z. dat het doublet voor het resultaat, tenminste voor de bewerkingen die op de descriptor plaats vinden, weer in compile time kan worden gesimuleerd. Voor de descriptor van het resultaat van de indexering hoeven we de descriptor van de arrayvalue niet te raadplegen.

8.3 Records, adresseren van componenten en variant selectie.

In Ada zijn, zoals bekend, records minder statisch van structuur dan in bijvoorbeeld PASCAL. Dit heeft zijn weerslag in de complexiteit van het access van record componenten.
Selectie van een component is niet altijd triviaal. Ook voor de wat ingewikkelder gestructureerde record objects willen we graag het eerder gegeven model hanteren, d.w.z.:

+ complete scheiding tussen data en data descriptie,

12. Merk op dat alle vormen van constraint checking van de return value van de function bij de executie van de return statement, binnen de body, worden verricht.

+ gestrekte values.

De complicaties t.o.v. PASCAL records zijn o.a.:

+ de record componenten zijn minder statisch van grootte,

+ we willen checken dat een component binnen de geldende variant is geselecteerd.

In die gevallen dat de offsets van de data van de record componenten niet statisch te bepalen zijn, voeren we een noodgedwongen indirectie in. Hetzij de typedescriptor, hetzij de valuedescriptor, wordt voorzien van een zg. offsettable, een tabel die voor elke component in het record vertelt wat de offset van de data van de component binnen de data van een object van dat type of subtype is.

We zullen de complete descriptor structuur geven voor een object van een niet triviaal record type.

```
type my_array is array (integer range <>) of integer;

type R2 (x: integer) is record
        A : my_array (e1 .. x);          -- e1, e2, e3 willekeurig
        B : my_array (e2 .. x);
        C : my_array (e3 .. x);
end record;

O2 : X2 (constraint);
```

Merk op dat de structuur en de grootte van deze descriptoren bekend is. Ook bekend is de relatieve positie van de locale descriptoren voor A, B en C binnen de descriptor van het subtype van O2.

Ook hier geldt dat, gegeven het adres van een descriptor de adressen van de component descriptoren hetzij compiletime bepaald, hetzij in runtime berekend kunnen worden.
We nemen als voorbeeld weer een functie, met als returntype R2.

```
function moeilijker (x : integer) return R2;
```

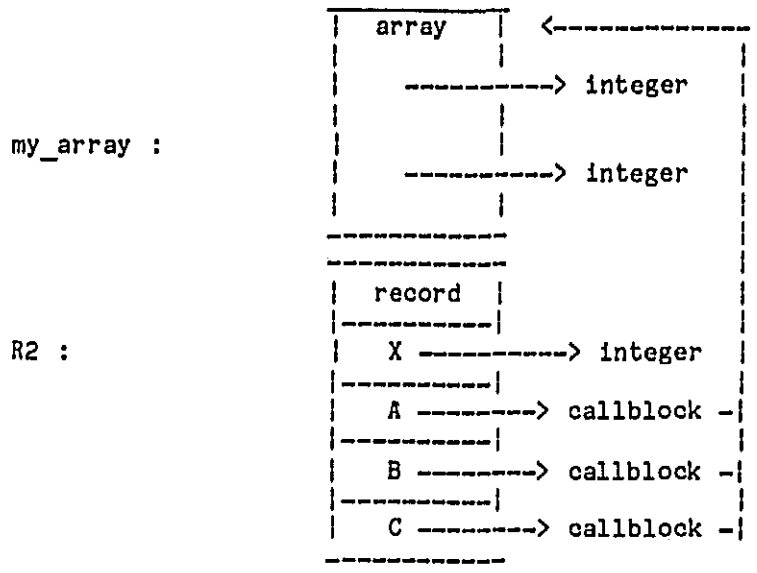
en beschouwen een "name construct"

```
...moeilijker (3). B (exp)
```

De volgende opmerkingen kunnen daarbij worden gemaakt:

1. het adres van de storage waarin B moet worden geselecteerd is het resultaat van de functie aanroep,

2. via een pseudo **out** parameter is de descriptor van dit resultaat adresseerbaar,



type descriptor structuur voor R2

3. het adres van de data van B vinden we door de inhoud van een element, met een compiletime bekende offset, uit de offset table bij het adres van de data op te tellen,

4. de descriptor van B vinden we op een, compile time, bekende offset binnen de descriptor van het functie resultaat,

5. het adres van het arrayelement vinden we op de gebruikelijke wijze binnen de data op het adres van B,

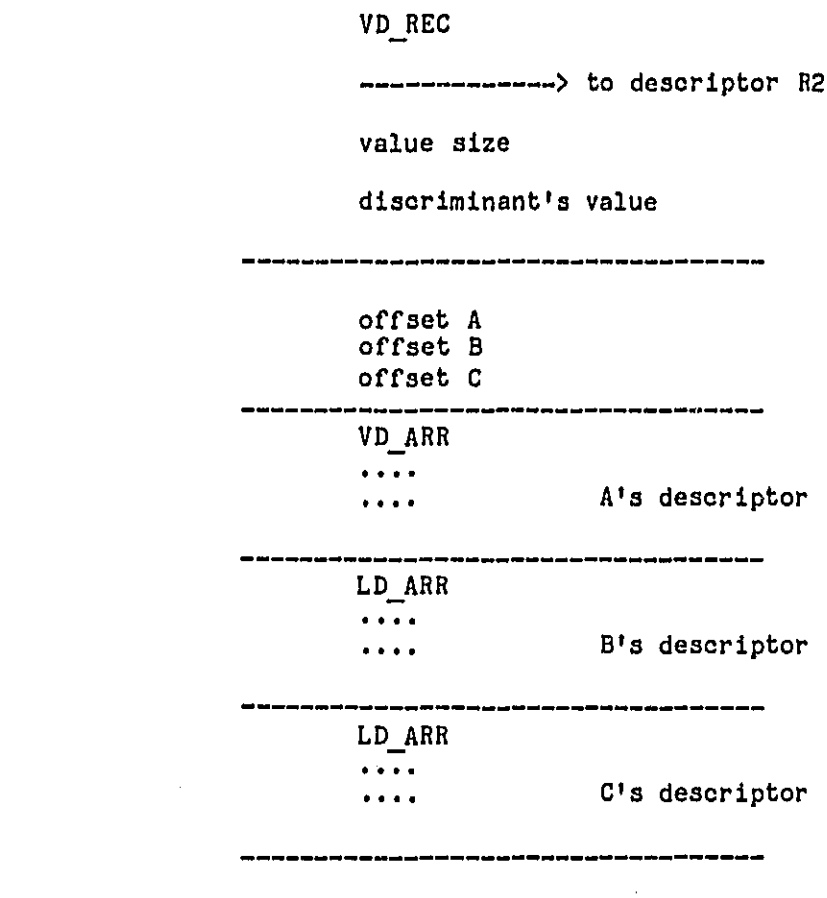
6. De descriptor van het arrayelement is "integer", die vinden we op een compiletime bekende plaats.

Het is het vermelden waard dat de selectie:

...moeilijker (3). x

anders loopt. Voor de selectie van een discriminant hebben we de descriptor van de value, in plaats van de value zelf, nodig.

De beschrijving van het data descriptiemodel wordt gecompleteerd met een beschrijving van een eenvoudig mechanisme ter ondersteuning van de zg. variant selectie, de verificatie dat een gekozen component ligt binnen een aanwezige variant.
We zullen de methode aangeven aan de hand van een voorbeeld.




```
type R (d1, d2 : integer) is
record
   f1, f2 : integer;
   case d1 is
      when -100 .. -1 =>
         f3, f4 : integer;
      when 0          =>
         f5 : integer;
         case d2 is
            when -1000 .. -1 =>
               f6 : integer;
            when others      =>
               f9, f8, f9 : character;
         end case;
      when others     =>
         null;
   end case;
end record;
```

We kunnen de paden door dit recordtype nummeren en krijgen dan een relatie tussen paden, of layouts, en componenten:

- layout nummer 1:
  D1, D2, F1, F2, F3, F4

- layout nummer 2:
  D1, D2, F1, F2, F5, F6

- layout nummer 3:
  D1, D2, F1, F2, F5, F7, F8, F9

- layout nummer 4:
  D1, D2, F1, F2

Voor elk van de componenten kunnen we een range aangeven waarbinnen het getal dat de layout aangeeft moet vallen om geselecteerd te mogen worden.

- F2 mag worden geselecteerd voor de layouts 1.. 4

- F5 mag geselecteerd worden voor layouts 2 .. 3

- F6 mag geselecteerd worden voor layouts 2 .. 2

We moeten ons daarbij realiseren dat voor elk subtype van dit record de layout vast ligt als de discriminantwaardes bekend zijn. De layout hoeven we dus maar eenmalig te berekenen en in de descriptor te stoppen. Staat de codering van de layout in de descriptor, dan kan het access zoals door de DAS compiler wordt gegenereerd naar bijvoorbeeld component F5 worden geformuleerd als:

```
base + (not layout in 2 .. 3 ?
        raise exception : offset_van F5
```

Om de layout te bepalen van een recordsubtype, wordt een zg padfunctie gegenereerd, gebaseerd op de structuur van het record. Voor bovenstaand record type is de gegenereerde padfunctie:

```
L5:
        tstb    sp@(-_F7-64)
        link    a6,#-_F7
        moveml  #_S7,a6@(-_F7)
        movl    a6@(8),a0
        moveq   #-100,d0
        cmpl    a0@,d0
        jgt     L10000
        cmpl    #-1,a0@
```

```
        jgt     L10000
        moveq   #1,d0
        jra     L10001
L10000:
        movl    a6@(8),a0
        tstl    a0@
        jne     L10002
        movl    #-1000,d0
        cmpl    a0@(4),d0
        jgt     L10003

        cmpl    #-1,a0@(4)
        jgt     L10003
        moveq   #2,d0
        jra     L10004
L10003:
        moveq   #3,d0
L10004:
        jra     L10005
L10002:
        moveq   #4,d0
L10005:
L10001:
        moveml  a6@(-_F7),#1028
```

9. Conclusies en verder werk.

Ondanks dat Ada niet dagelijks in de ochtendkranten wordt vermeld lijkt de opmars van de taal niet te stuiten. Dit kan o.a. geconstateerd worden uit het regelmatig verschijnen van nieuwe leerboeken over de taal.
Zeer belangrijk is het gegeven dat nu reeds vele tientallen miljoenen guldens geinvesteerd zijn in de ontwikkelingen rond Ada en dat ook buiten DoD, in het bijzonder in de VS en Europa, de taal geaccepteerd wordt.
We kunnen niet verwachten dat een programmeertaal van de ene op de andere dag ingeburgerd is bij een groot publiek. Ook voor een, momenteel, populaire taal als PASCAL heeft het 5 a 10 jaar geduurd voor de taal werkelijk populair werd.
Het is overigens niet erg waarschijnlijk dat Ada de populariteit zal halen die PASCAL momenteel heeft, Ada is een orde complexer en wordt vooralsnog gezien als taal voor grote projecten.

Programming Support Environments, programmeer omgevingen waarbij een zekere mate van integratie wordt gerealiseerd tussen de tools onderling en de tools en de te manipuleren objecten, staan de laatste jaren volop in de belangstelling. De eerste Ada Programming Support Environments zijn in aantocht, de eerste resultaten tonen dat er relatief veel computer "power" voor nodig is. Gezien de snelle ontwikkelingen van de microelectronica lijkt dit echter geen beletsel, momenteel heeft een pocket computer het vermogen van een mainframe van 20 jaar geleden.
Binnen DoD heeft men een beeld van de software engineer van de toekomst [SI]. Zo iemand is uitgerust met een eigen portable computer systeem (een Super Apse Workstation), liefst gebaseerd op het dynabook concept, met megabit verbindingen naar centrale apparatuur die de grootschaliger databases bevat.

Wat betreft ons eigen werk, een initiele versie van de DAS implementatie nadert zijn voltooing. Het nodige werk zal nog besteed moeten worden aan diverse vormen van optimalisatie. In het bijzonder high-level optimalisaties zijn nodig om de resulterende code efficient te krijgen.
Overwogen wordt om, als exercitie af te stappen van het gebruik van het back end van de PCC en een eigen codegenerator te ontwikkelen. Ook bestaan plannen om DAS uit te breiden met tasking en generics.
Een tweede versie van de DAPSE staat op het punt te verschijnen. In deze versie wordt een eenvoudige vorm van een database management systeem geboden aan de gebruiker, o.a. ten behoeve van de domain manager, de beheerder van de program libraries.

10. Bibliografie.

[1] [Wichmann]
Ada is green
B.A. Wichmann,
Computer Bulletin sept 1979

[2] [AJPOnews]
Newsletter Ada Information Clearinghouse,
Volume I, Number 4
July 1984

[3] [LRM]
Reference manual for the Ada programming language,
ANSI/MIL-STD 1815-A,
Januari 1983

[4] [Stoneman]
Requirements for Ada Programming Support Environments,
U.S. Department of Defense,
1980

[5] [UNIX]
Foreword in: Bell System Technical Journal,
juli-augustus 1978

[6] [Computer]
diverse artikelen in: Computer magazine,
juni 1981,

[7] [CAIS]
Draft specification of the Common APSE Interface Set (CAIS),
Version 1.0
26 Augustus 1983,
Uitgegeven door DoD

[8] [SIGPLAN]
Preliminary Ada Reference Manual,
Sigplan Notices juni 1979.

[9] [YACC]
YACC- yet another compiler compiler,
S.C. Johnson
CSTR 32, Bell laboratories Murray Hill
1974
included in UNIX documentation

[10] [YACCPREP]
A preprocessor for YACC or
a poor man's approach to parsing attributed grammars,
J van Katwijk,
SIGPLAN Notices, october 1983

[11] [DIANA]
DIANA reference manual,
revision 3
A. Evans, K. Butler
Tartan Laboratories Incorporated
1983.

[12] [PCC],
A tour through the Portable C compiler,
UNIX documentatie versie 7

[13] [EM]
Description of a machine architecture for use with block structured languages,
A.S. Tanenbaum, H van Staveren, E.G. Keizer, J.W. Stevenson
Rapport IR-81
Subfaculteit Wiskunde en Informatica
Vrije Universiteit Amsterdam

[14] [DASDESC]
Descriptors for DAS,
J van Katwijk en J van Someren
Rapport 82-20 Onderafdeling der Wiskunde en Informatica
Technische Hogeschool Delft,
1982.

[15] [DOUBLET]
The doublet model,
J van Katwijk and J van Someren,
SIGPLAN Notices januari 1984

[16] [DASINTER]
An interpreter for DAS,
N.C. Bogstad, A.M.J. Hartveld
Laboratorium voor schakeltechniek en techniek van de informatieverwerkende machines
rapport 051560-28(1983)16,
Afdeling der Elektrotechniek,
Technische Hogeschool Delft,
1983

[17] [CALLBLOCK]
A runtime representation for Ada variables and types,
A. Hisgen, D.A. Lamb, J Rosenberg and M. Sherman
SIGPLAN Notices 15, November 1980

[18] [SI]
Strategy for a DoD Software Initiative,
U.S. Department of Defense.
1 october 1982

[19] [BACKEND]
Using PCC's backend for generating code in the DAS compiler,

doctoraalscriptie,
C. Barkey,
Onderafdeling Wiskunde en Informatica,
Technische Hogeschool Delft.
december 1982.

MATRIX OF ADA LANGUAGE IMPLEMENTATIONS

(Updated November 1983)

EUROPEAN AND JAPANESE EFFORTS

ORGANIZATION	CONTACT	CONTRACT	SCOPE	HOST SYSTEM	TARGET SYSTEM	AVAILABILITY
SESA-Deutschland GmbH	Mr. Debest Frankfurt, W.Germ. (49) 6 11-71 72 11	German MoD (part of SPERBER)	Specification and realization of Ada I/O	Siemens 7.7xx (Ada)	Siemens 330, portable to others	Completed Dec. 1981
SPERBER Project: Univ. Karlsruhe; GPP, SESA, VERUM	MoDr G. Fickenscher (201) 488-6393 IABG (Tech. Coord) Dr. H. Hummel, (49) 89 6088-3198	German MoD	Standardized program development system for military applications	Siemens 7xxx (Ada)	Several; including Siemens 7xxx & 3xx	See individual projects
SystemKG	Georg Winterstein (49) 7215 52837		a.) Ada'80 Front End in standard Pascal b.) ANSI Ada compiler	VAX	VAX IBM-370	a.) Now b.) Mid 1983
Univ. of Hamburg	Prof. Dr. H. Nagel (49) 484-4123-4151	Deutsche Forschungsgemeinschaft	Investigate suitability of Ada for image processing systems. APSE implementation planned. Uses Karlsruhe Front End.	DEC-10 (Pascal first, Ada-0 superset later)	German minicomputers	
Univ. of Karlsruhe	Dr. Gerhard Goos Dr. Guido Persch (49) 721-608 3805	Front End by German MOD, Back End internal	Full Ada, host independent Front End, retargetable Back End	Siemens 7760	Siemens 7.000, MC68000	Front End completed in Dec. 1981, Bootstrap Back End early 1983, completed Dec. 1983
Univ. of York	Dr. Ian Wand (44) 904-59861	Science Research Council of Great Britain	Compiler for full Ada with retargetable Back End; emphasis on compile time diagnostics	VAX/UNIX (C) PERQ/UNIX (C)	Same, plus PDP-11/UNIX, bare PDP-11	Subset compiler for VAX available to UK educational establishments in Oct. 1982. Full compiler for VAX in mid-summer 1983.

III-4.157

MATRIX OF ADA LANGUAGE IMPLEMENTATIONS

(Updated November 1983)

U. S. EFFORTS

ORGANIZATION	CONTACT	CONTRACT	SCOPE	HOST SYSTEM	TARGET SYSTEM	AVAILABILITY
Alsys, Inc.	Chuck Patrick (617)890-8030	Internal	Full ANSI Ada; Joint effort with Alsys, S.A.	VAX/VMS; 68000-and 8086-based workstations	MC 68000, Intel 8086	Fall 1984
Amdahl Corp.	Robert Lent (408)746-6563	IR&D	Port NYU Ada/ED to Amdahl 470; cross-compiler for Intel 8086.	Amdahl 470 (IBM compatible)	Intel 8086	Ada/Ed: now 8086: Dec. '83 (Int. use only)
Bell Labs	Charles Wetherell (201)582-3099	Internal	Bootstrap Subset Compiler; Full Prototype Compiler	VAX (Unix)	VAX	Early '83 (Bootstrap) Limited
Bolt, Beranek and Newman	Bob Thomas Rich Shapiro Julie Sussman (617)491-1850	Internal	Partial implementation based on Pranis compiler	VAX(VMS); DEC-20	VAX, PDP-11; DEC-20	Late 1982
Burroughs Federal and Special Systems	Teri Payton. (215)648-7268	IR&D	Ada Front End to Diana using attribute grammar techniques	Burroughs large systems	Burroughs system	
Carnegie-Mellon Univ.	Mario Barbacci (412)578-2578 Barbacci@CMUA	SPICE Project	Full compiler & programming environment	DEC VAX (UNIX) and Three Rivers Computer Corp. PERQ (ACCENT)	PERQ(Also VAX UNIX with postprocessor	Subset compiler available since 1981. Full Ada compiler & programming environment in 1984. Distribution under SPICE Project Industrial Affiliates Program.
Control Data Corp.	Clyde G. Roby, Jr. (404)955-8782	Internal	Full Ada with partial environment	CYBER-170(NOS)	CYBER-170	Late 1986

III-4.152

MATRIX OF ADA LANGUAGE IMPLEMENTATIONS

(Updated November 1983)

U. S. EFFORTS

ORGANIZATION	CONTACT	CONTRACT	SCOPE	HOST SYSTEM	TARGET SYSTEM	AVAILABILITY
Digital Equipment Corporation	Margie Tagliari (603)881-2338		Full Ada	VAX/VMS	VAX/VMS	
Digicomp Research Corporation	Graham Hall (607)273-5988	Western Digital	ANSI Standard Ada	Delphi 186 with Gensoft STC Ada (validated); Telesoft-Ada	Same	Immediate
Florida State Univ.	Dr. T. P. Baker (904)644-5452 or 644-1898	USAF/ AFATL	Full Ada	CDC Cyber 70 (NOS)	Zilog Z-8000 Development Module MIL-STD-1750A	Dec. 1983
Gensoft Corporation	Pamela Buzzy (412)621-8235	1. acq'd from M.D. 2. TBM 3. In-ternal 4. In-ternal	1. ANSI Ada 2. ANSI Ada 3. 16-bit interpreter. Limited to 400 line units. Primary use is education and training. 4. 32-bit interpreter. ANSI Standard Ada	1. MD 1600 Series 2. VAX/VMS 3. VAX/VMS 4. VAX/VMS	1. Same 2. MCF 3. VAX/VMS 4. VAX/VMS	1. Now; validated Aug. 9, 1983 2. Late 1983 3. Now 4. April 1984
Honeywell Office Mgmt. Systems Division	Alan C. Lyman (617)671-2887	None	Subset Full Ada	MULTICS (PL/I) DPS6	DPS6 Same	Now TBD
Intel Corporation	Tony Anderson (503)640-7139	Intel	Full Ada, Linker, Debugger, Target Operating System except Fixed Point Types, Abort, Timed Entry, Enumeration Type I/O, some predefined pragmas.	VAX(UNIX/VMS)	Intel iAPX-432	April 1983

MATRIX OF ADA LANGUAGE IMPLEMENTATIONS

(Updated November 1983)

U. S. EFFORTS

ORGANIZATION	CONTACT	CONTRACT	SCOPE	HOST SYSTEM	TARGET SYSTEM	AVAILABILITY
Intel Corporation	Kenneth Pooper (408)987-5934	Intel	Full, validated Ada	VAX/VMS	8086 (iAPX86)	End of 1984
Intermetrics, Inc.	Peter Belmont Mike Ryer (617)661-1840 Belmont@ECL8	AJPO	Full Ada	PDP-10 (TOPS-20) (SIMULA)	TOPS-20 with extended dressing	Partial Implementation (Not supported)
Intermetrics, Inc.	Mike Ryer (617)661-1840	USAF (RADC)	Full Ada plus environment ('Stoneman')	IBM 370	Same	1984
Mills International	Mark McClees Carlton Mills (217)398-1986	Private	Full Ada	Burroughs large systems: B5900, B6900, B7900	8086, 286, host emulation	Third quarter 1984
New York University	Ed Schonberg (212)460-7482 SCHONBERG at NYU	U. S. Army (CECOM)	Full language interpreter and an operational definition of Ada.	VAX: VMS and UNIX IBM:CMS, MVS, VSI	Same	Now, VAX/VMS: from NTIS (validated 3/83) others: from NYU
NuSoft	Wayne Magor (509)525-9463	Internal, w/Hewlett Packard	Full Ada eventually	HP3000	Same	Subset late 1982
ROLM Corp.	Jim Williams (408)988-2900	Internal	Full ANSI Ada with toolset	MSE/600; Data General 32-bit ECLIPSE	MSE/600, MSE/14, 16600	Validated compiler available now
RR Software	Randall Bruhhardt (608)244-6436	Internal	Designed to run on, generate code for, current microprocessors.	8086 MS-DOS, 8086 CP/M-86	Same, and 8080/280	Partial implementation available now
Science Applications, Inc.	Bruce Lightner Lee Carver (619)456-6660	Internal w/Hewlett Packard	Full Ada eventually	HP 1000 series (Pascal)	Same	Subset late 1983

MATRIX OF ADA LANGUAGE IMPLEMENTATIONS

(Updated November 1983)

U. S. EFFORTS

ORGANIZATION	CONTACT	CONTRACT	SCOPE	HOST SYSTEM	TARGET SYSTEM	AVAILABILITY
SofTech, Inc.	George Rodriguez (617)890-6988	U.S.Army (CECOM)	Comprehensive Ada Programming Support Environment with 'Stoneman', and including production AMSI Ada compiler.	VAX/VMS	Same MCF	Host/Target - Oct. '84 MCF- Dec. '85
Stanford University	David Luckham (415)497-1242 Software Dist. Ctr. (415)497-8651	DARPA	Adam supports generics, exceptions, sep. comp., most of Ada88 tasking; Full Ada tasking in progress.	DEC-10 TOPS-20	DEC-10	Adam available now
Telesoft, Inc.	Bruce Sherman (619)457-2700 V.P., Sales & Marketing	None	High performance substantial host independent implementation Front End (AMSI Ada '83) and middle pass, easily retargeted Back End. Suitable for production projects and/or education.	VAX/VMS; VAX/UNIX; IBM/VM/S/TSO; IBM/CMS; IBM PC 68000/UNIX; 68000/ROS	Same, cross-compilation from all hosts to 68000 Bare machine for production code.	Partial AMSI Ada '83 for all above available now. Certify to pass all validation tests on at least one host/target by the end of 1983. Will be submitted for validation shortly thereafter.
Univac Defense Systems Division	Joe Cross (612)456-4929	IRAD	'Bda' subset; no tasking very small	Univac 1100	FORTRAN simulator of B-machine target	January 1981
USAF Armament Lab	Mr. A.Marlow Henne 2nd Lt. Brian K. Whatcott (904)882-8264 -2961		Full Ada Compiler	CDC Cyber 176 running NOS or NOS/BE operating system.	Z8002 Development Module, Z8002 DIS computer	Z8002 Develop Module Target 1st Q CY84 Z8002 DIS computer Target 3rd Q CY84
USC-ISI	Dan Lynch Robert Balzer (213)822-1511	DARPA	Interpreter for Formal Semantic Definition of Ada by INRIA/ALSYS	DEC-20 (TOPS-20) Berkeley VAX	Interlisp	July 1982

MATRIX OF ADA LANGUAGE IMPLEMENTATIONS

(Updated November 1983)

EUROPEAN AND JAPANESE EFFORTS

ORGANIZATION	CONTACT	CONTRACT	SCOPE	HOST SYSTEM	TARGET SYSTEM	AVAILABILITY
EACP (European Ada Compiler Project); Alsys, S.A.; CII Honeywell Bull; Siemens	J-C. Hellard Alsys La Celle St. Cloud France (33-3)918 1244 MMyers MIT-Multics	European Communities	Full Ada; portable cool; rehostable on mini- and micro-computers	C1% HB level 64, Siemens 7760	Same.	1984
Entwicklungsberro WEHM	Mr. Windauer (49)4131 53344	German MoD (part of SPERBER)	Symbolic Test/Debug System Compiled and interpreted mode; highly rehostable/ retargetable.	Siemens 7.xxx BS 2000 (Ada)	Same, and also Siemens 330/ ORG K	Mid 1984
GPP Munich	Mr. Eichenauer (49)89 481854	German MoD (part of SPERBER)	Retargetable Back End for Full Ada; input is Diana from Karlsruhe Front End	Siemens 7000 (Ada-0)	Siemens 3300 et. al.	Mid 1983
Int'l Computers Ltd.	Martyn Jordan (44)753 31 111	ICL	Systems progng. subset; excludes tasking, real types, I/O	ICL 2900	ICL 2900	
Nippon Telegraph & Telephone, Yokosuka Electrical Communication Laboratory	Ryoichi Hosoya (81)468-59-2718		Full Ada eventually	DIPS	Same	Subset late 1983
Olivetti/Danish Datamatics Centre Christian Rovsing Ltd.	Ole Oest Lyngby, Denmark (45)2-872422	European Communities	Full Ada compiler plus Stoneman environment	Christian Rovsing CR80D Olivetti S6000	Same	Early 1984
Oy Softplan Ab	Pekka Lahtinen (+358)31 37 317 TELEX: 122548 SPLAN SF	NOKIA Electronics Finland	Subset for systems software development on MPS 10	MPS 10 Bootstrap on MIKKOS	MPS 10	October 1981

Reeds eerder verschenen in deze reeks:

84-11	Z. Nowak	A quasi-Newton multigrid method for dertermining the transonic lifting flows around airfoils
84-12	A. Segal	A short note on the implementation of boundary conditions of the type u equals unknown constant in finite element codes
84-13	W. de Melo & S.J. van Strien	Diffeomorphisms on surfaces with a finite number of moduli
84-14	G.W. Decnop	In de ban van de driehoek
84-15	J. van Kan	A second-order pressure-correction method for viscous incompressible flow
84-16	P. Sonneveld	CGS, a fast Lanczos-type solver for nonsymmetric linear systems
84-17	A.H.P. van der Burgh	A simple aeroelastic oscillator as a model for conductor galloping
84-18	H.L. Claasen (red.)	Colloquium Vakgroep Algemene Wiskunde
84-19	M. Kok and F.A. Lootsma	Pairwise-comparison methods in multi-objective programming, with applications in a long-term energy-planning model
84-20	Bram van Leer and William A. Mulder	Relaxation methods for hyperbolic equations
84-21	Ben de Pagter	A functional calculus in f-algebras
84-22	M. Kok and C. Roos	On the convergence of a nonnegative square matrix
84-23	Bram van Leer	Upwind-difference methods for aerodynamic problems governed by the Euler equations
84-24	L. Fortuin and F.A. Lootsma	Future directions in operations research
84-25	P. Sonneveld and B. van Leer	A minimax problem along the imaginary axis
84-26	C.J. van Duyn and D. Hilhorst	On a doubly nonlinear diffusion equation in hydrology
84-27	G. Hooghiemstra and M. Keane	Calculation of the equilibrium distribution for a solar energy storage model
84-28	J. van Katwijk	Ontwikkelingen rond Ada

Exemplaren van deze rapporten kunnen besteld worden bij het bureau van de Onderafdeling der Wiskunde en Informatica, Julianalaan 132, 2628 BL DELFT, telefoon 015-784568.