Documentnumber L
Distribution

EWD254 - 0

Hoger orde adressering.

Als men programmaonderdelen of bibliotheekprocedures onafhankelijk wil vertalen impliceert dit de introductie van locale terminologieen (immers: zonder locale terminologieen waren de vertalingen niet onafhankelijk!).

Wie dergelijke onafhankelijk vertaalde stukken wil samenbreien, verplicht zich tot de introductie van een interlocale terminologie, waarin elk object in het hele breiwerk uniek benoemd is.

Om te vermijden, dat teksten in dit samenbreien extensief worden bijgewerkt, voert men tabelletjes in, die steeds een locale terminologie in de interlocale overvoeren. We bereiken hiermee hetvolgende
1) de samenbreibeslissingen (inbedbeslissingen) zijn in de tabellen gelocaliseerd, wat voor wijzigen bv. belangrijk is.
2) als eenzelfde text in verschillende breisels voorkomt (denk aan bibliotheekprocedures), dan hoeven we slechts het tabelletje in veelvoud in te voeren.
3) als de diepte van adressering niet in de programmatext maar in de entry van de tabelletjes te vinden is, dan hebben we een flexibel middel ter versnelling (indien nodig).
4) elke locale terminologie kan een gesloten adressering zijn, met enige voorzorgen kunnen we de interlocale terminologie ook (grotendeels) gesloten houden.

Je eerste idee is, om elk vertaald stuk te voorzien van een catalogus, waaruit bij samenbreien het locale tabelletje kan worden samengesteld. (Dit zal "naamtechnisch" wel op contractie neerkomen!) Een programma, dat uit vijf onafhankelijk vertaalde stukken wordt opgebouwd bevat dan zes tabellen: eentje voor elke locale terminologie (dat zijn er al vijf) en eentje voor de interlocale terminologie (waarin we voor elk object uniek bv. kunnen bijhouden, waar het zich bevindt).

We doen er goed aan te beseffen, dat we met de introductie van deze zes tabellen al aan het implementeren geslagen zijn! In het programma zijn het de vijf locale terminologieen, die een duidelijk gescheiden identiteit hebben: wat we vragen is een of ander vertaalmechanisme van locale naar interlocale naam (of zo).

Ik wil voorlopig proberen de vijf locale tabellen en de ene interlocale zo gelijk mogelijk te beschouwen en wel om de volgende redenen:
1) als ons breiwerk uit één onafhankelijk vertaald stuk bestaat -stel dat dat denkbaar is- dan kunnen locale en interlocale terminologie samenvallen.
2) als we "interlocale terminologie" denken aan de nomenclatuur voor segmenten, dan moeten we niet schrikken als sommige entries van de interlocale tabel doorverwijzen naar een met andere programma's gedeeld universum: ik denk aan de procedurebibliotheek.
3) ik zie niet in, waarom bv. aan-afwezigheidsadministratie van een segment, dat slechts in één locale terminologie benoemd is, niet in de bijbehorende locale tabel gevoerd zou kunnen worden. Dit zou efficienter kunnen zijn.

niet gecontroleerd

Zolang men zich beperkt tot vertaal- en linkingproblemen maakt men het zich echter heel makkelijk. Het punt is nl. dat op elk moment dat een (nieuwe) interlocale terminologie moet worden ingevoerd "elk segment er al is". Je kan die segmenten op een rijtje zetten en dan nummeren. De te identificeren populatie is bekend op het moment dat de identificatoren moeten worden ingevoerd. Wie zich tot deze constellatie beperkt, dreigt te ontgaan dat elk mechanisme, volgens welke we objecten aan een reeds bestaande (en intern geidentificeerde) populatie kunnen toevoegen of er van kunnen afnemen, achter de schermen een algorithme impliceert voor de toekenning van de nieuwe identificator.

Ter onderscheiding van wat we besproken hebben en wat nu komt noem ik wat we gehad hebben statische (locale en interlocale) adressen en wat nu komt dynamische (interlocale en interlocale) adressen. Ik zal ook wel spreken van "statische context" en "dynamische context".

Bij de dynamische locale adressen hebben we er voor te zorgen, dat in eenzelfde programma hetzelfde dynamische adres in verschillende incarnaties naar verschillende variabelen kan wijzen. Zulks betekent, dat we per incarnatie een tabelletje voor de locale dynamische terminologie invoeren. Er zijn, dacht ik, een aantal redenen om statische en dynamische locale adressen naar aparte tabellen te laten verwijzen. Er is een heel prozaische reden: als we ze in één tabel willen onderbrengen, dan wil je dacht ik de dynamische adressen wel apart hebben -zeg aan het hoge eind- en dat sorteerproces compliceert je vertaalproces een beetje. Er is een tweede reden en die is practischer: als je de zaak steeds in een tabel wilt onderbrengen dan moet je bij elke procedureaanroep dat constante stuk van de statische context, dat bij assemblage is ontstaan, copieren. En als je zoekt kan je, dacht ik, ook nog wel een formulering vinden, die fundamenteler aandoet, nl. dat beide tabellen, ieder voor zich, bedoeld zijn om het systeem opgewassen te maken tegen een heel ander soort combinatorische variatie, nl. verschillende namen voor hetzelfde, versus dezelfde naam voor iets anders.

De interpretatie van locale adressen vergt nu dus de kennis van twee tabellen, de statische en de dynamische. Vanwege verschillende procedureincarnatie's moeten we de combinatie van eenzelfde statische context met vele dynamische contexten toestaan. Moet men ook omgekeerd de combinatie van eenzelfde dynamische context met verschillende statische contexten toestaan?

Ik heb het geval onderzocht, dat dit niet was toegestaan, dwz. dat elke dynamische context tijdens zijn leven altijd met een vaste statische context gecombineerd zou worden. Toen ik met die analyse klaar was, vroeg ik me echter af, wat er dan wel was overgebleven van het ideaal van "onafhankelijke vertaling van delen": een en ander lijkt er dan nl. ernstig op, dat elke procedure in zijn geheel vertaald moet worden. En aangezien ik er aan gewend ben desgewenst een programma als een procedure te beschouwen, zou elk programma in

dan combinatie v.d. dynamische context m.n. met de statisch locale terminologie

zijn geheel vertaald moeten worden. Met de restrictie "slechts een statische context per dynamische" lijkt een stukje kind met het badwater weggegooid.

Ik stel me in het volgende voor, dat we geen aparte pagina's kennen, voor segmenten enerzijds een maximale lengte accepteren, anderzijds niet opzien tegen kleine segmentjes. Er staat ons een dergelijke geheugenbespeling voor ogen, laat ons proberen het er mee te doen!

In dat geval ligt het voor de hand om te overwegen naast de statisch interlocale tabel ook een dynamisch interlocale tabel in te voeren. Het alternatief is om slechts één interlocale tabel te hebben: wat via de statisch locale tabellen gerefereerd wordt komt dan in de bodem, terwijl daarop de dynamische tabel stack-fashion (stapelsgewijze is natuurlijk goed Nederlands!) groeien en krimpen kan. Dat betekent wel zowat, dat "linking" voltooid moet zijn voordat executie begint! Ik neem dus aan, dat we een aparte dynamische interlocale tabel hebben voor de unieke identiteit van de dynamische segmenten.

Ik stel me voor dat bij blokbinnenkomst voor de locale scalairen een segment geintroduceerd wordt. De verwerking van de arraydeclaratie kan zijn dat (in dit segment) de parameters worden opgezet die subscriptiewaarden omrekenen tot dynamisch interlocale adressen en dat zoveel segmenten worden geintroduceerd als voor dit array nodig. (NB. Als we de dynamisch interlocale tabel stacksgewijze bedrijven, dan kunnen we voor een groot array opeenvolgende interlocale segmentnummers uitdelen; dan rekenen we dus aan het dynamisch-interlokale adres. Een interlocaal segmentnummer wordt dan echt een nummer!) Wat het systeem betreft zijn segmenten voor locale scalairen en voor locale arrays op practisch dezelfde leest geschoeid en dat is hoopgevend.

Nu het moeilijke artikel van de procedureaanroep. Ik moet me in dit stadium beperken tot een eerste verkenning. We zitten met:
1) het meegeven van parameters, inclusief terugkeerinformatie
2) het springen naar een ander punt en eventueel een herdefinitie van de statisbhe context
3) het creeren van een nieuwe geinitialiseerde dynamische context en het hierop overgaan.

Het meegeven van de parameters komt kennelijk ten laste van de calling sequence. In de daar heersende dynamische context kome een parametersegment voor, waarin de calling sequence de gegevens ad 1 invulle (wellicht uitgezonderd de terugkeerinformatie).

De karakterisering van de procedure bevatte de gegevens ad 2 (voor de sprong) en een karakterisering van de dynamische context waaruit de bodem van de te creeren dynamische context geinitialiseerd dient te worden.

Er kan nu een subroutinesprong uitgevoerd worden met als parameters
a1) de procedurekarakterisering (zie vorige alinea)
a2) de naam van het parametersegment (ik stel me voor, dat dit de interlocale naam al is, maar daarvan ben ik niet helemaal zeker)

De subroutinesprong bergt (bv. op een vaste plaats van het parametersegment) de terugkeerinformatie (in elk geval terugkeerplaats en statische context, dynamische context mag wat later) en de sprong naar de eerste opdracht van de procedure wordt uitgevoerd, terwijl de dynamische context van de calling sequence nog vigeert! Onder controle van deze text kan nu een nieuwe dynamische context gecreeerd worden. (Als voor deze creatie de lengte van de dynamische tabel nodig is en deze is een functie van de blokhoogte en van interne nesting van blokken, dan kunnen we deze gegevens, dacht ik, aan het begin van de procedure bekend veronderstellen.) Ik neem aan, dat gegevens a1 en a2 nog beschikbaar zijn. Uit a1 komt het gegeven uit welke dynamische context de nieuw geprepareerde gevuld moet worden (de proceduretext kan de blokhoogte suppleren). Vervolgens zal- en dat is eigenlijk de overgang van het parametersegment van actueel parametersegment in de calling sequence naar formeel parametersegment van de procedure- de procedure zelf een element (het volgende neem ik aan) van de dynamische tabel in preparatie initialiseren voor het parametersegment. Omdat gegeven a2 nog bekend verondersteld wordt en de procedure zelf de eigen naam van het formele parametersegment kent, zijn hiervoor alle gegevens ter beschikking. Tenslotte kan de procedure de dynamische context in opbouw uitbreiden met een segment voor de eigen locale en moet de dynamische context in opbouw de vigerende worden.

Dit hele stuk initialisatie en introductie van de nieuwe dynamische context kan de procedure dus zelf commanderen. Ik kan me voorstellen dat de ongeinitialiseerde nieuwe dynamische context meteen al "de vigerende"wordt. Initialisatie en introductie van een nieuw segment hebben dan per definitie betrekking op de vigerende dynamische context en dat XXXXX lijkt wel prettig. Je zit dan wel een tijdje met "verboden dynamische adressen" (ongedefinieerde).

Opm1. Voor apart vertaalde procedureonderdelen mag de statische context voor "externals" van het onderdeel, die verwijzen naar locale van de hele procedure bv., als entry een dynamisch adres bevatten, datper definitie verwijst naar de op dat moment vigerende dynamische context! Een dynamisch adres is statisch (bij textopbouw, maar ook bij samenbreien) heel goed manipuleerbaar. Dit geeft een belangrijk stuk vrijheid, bv. voor procedures uit de bibliotheek, die je "own's" wilt geven, geadresseerd in de bodem van de stapel van het aanroepende programma.

Opm2. Dat het parametersegment bij cell van dynamische naam verandert lijkt me uiterst gezond.

Opm3. Dat een dynamische context niet meer (zoals in mijn eerste opzet) vast aan één statische context gekoppeld is lijkt me een verbetering. Dit maakt het

nl. mogelijk statische en dynamische context onafhankelijk om te zetten, dwz. nadat de statische context is omgezet kan de proceduretext zelf beginnen met de creatie van een nieuwe eigen dynamische context. In mijn eerste opzet moest dat veel gekunstelder.

Opm4. De terugkeerinformatie stel ik me voor te bevatten
1) dynamische context (wijzer)
2) statische context (wijzer)
3) locaal terugkeeradres (in termen van 2.)