Object-Oriented Language Interoperability

-Beta.Net og Beta.Java

Af Peter Andersen

Baggrund

Jeg vil her forsøge at beskrive hvad de to projekter Beta.Net og Beta.Java,
som vi under ét kalder for "object-oriented language interoperability" handler om. Jeg vil på forhånd afstå fra at forsøge at oversætte denne titel til dansk ☺ .
Som projektnavnet antyder, handler det om det objekt-orienterede programmeringssprog BETA, der er udviklet gennem 25 år her på DAIMI og hos Mjølner Informatics, og som langt de fleste har stiftet bekendtskab med på deres førstedelskurser. For dem der ikke har haft denne berigende oplevelse, må jeg henvise til nogle referencer [1][2][3], men man burde nu kunne læse denne artikel uden egentlig BETA baggrund.

For BETA sproget er der gennem årene blevet udviklet compilere og runtime systemer til en hoben af forskellige hardware/operativsystem platforme. De første compilere blev lavet til Motorola 68k arkitekturen på de første Macintosher og SUN-2'er, og på nogle maskiner, der hed Apollo, som mange sikkert ikke har hørt om. Siden er kommet f.eks. compilere til Intel x86 arkitekturen på Windows og Linux, til MIPS arkitekturen på Silicon Graphics, til PowerPC arkitekturen på Macintosh og til SPARC arkitekturen på de nyere SUN maskiner.
Hver portering har skullet tage hensyn til instruktionssættets egenskaber og register- og memorymodel, og over årene er et rimeligt abstrakt interface til disse temmelig forskellige platforme blevet udviklet.
Parallelt hermed er der vedligeholdt et runtime system, som indeholder f.eks. de allokeringsrutiner, som compileren kalder, når objekter skal skabes, og naturligvis den garbage collector, som senere skiller sig af med ubrugte objekter igen. Garbage collectoren er skrevet i C, og er rimelig porterbar, mens allokeringsrutiner for nogle platformes vedkommende er skrevet i assembler kode, og er "træls" at portere.

Som det vil være mange bekendt fremkom Java [4] i 90'erne og fik hurtigt stor udbredelse. Blandt andet fordi det var lavet meget porterbart og umiddelbart var tilgængelig til alle de væsentlige hardware/operativsystem platforme.

Senest har Microsoft så lanceret deres .NET platform, som på mange områder er meget inspireret af Java, men fra Microsofts side primært er tiltænkt en rolle på Windows, selvom nogle af de grundlæggende teknologier er standardiseret i ECMA, og på vej under ISO, og selvom der findes nogle open-source projekter, der er i gang med at portere dele af .NET til f.eks. Linux [6][7][8].
Én væsentlig ting adskiller dog .NET fra Java: Hvor Java er tænkt som "Skriv din kode i ét sprog – kør den alle vegne", så er .NET fra start designet til at skulle understøtte mindst de for Windows platformen udbredte implementeringssprog Visual Basic (VB), JScript, C++ og deres egen variant af Java: J++. .NET indeholder således en "common language runtime" (CLR), som understøtter tilpassede varianter af VB, C++ og J++ (nu kaldet J#), JScript og et helt nyt temmelig Java lignende sprog kaldet C#.

Microsoft har fra starten gjort sig stor umage med at få flest mulige programmeringssprog til at køre på denne platform, både – som nævnt – de store main-stream sprog, men også en lang række akademiske sprog, se [9].
SUNs har så vidt vides aldrig fokuseret på at bruge JVM som platform for andre sprog end Java. Der findes dog en lang række forskningsprojekter om dette, se [12]. Et resultat vi håber kan komme ud af vores projekt er en form for brobygning mellem Java og .NET, idet vi satser på at opnå en identisk interoperabilitet mellem hhv. BETA og .NET samt mellem BETA og Java.

I slutningen af 2001 henvendte Microsoft Danmark sig til Ole Lehrmann Madsen og spurgte om vi kunne tænke os at portere BETA til .NET platformen – de ville gerne sponsorere ☺.
Omtrent samtidig havde SUN doneret en stak SunRay maskiner til DAIMI og med disse to private donationer i ryggen havde vi basis for at definere et forskningsprojekt, som altså blandt andet handler om at portere BETA til at køre ovenpå Java's virtuelle maskine (JVM) og ovenpå .NETs Common Language Runtime (CLR), og det har Ole og jeg så arbejdet med i perioder de seneste par år.

Sprog "mapping"

For at kunne "mappe" BETA til at passe ind i JVM og CLR verdenerne må vi naturligvis først kigge en del på dem. Det viser sig dog, at der er meget store fællestræk – Microsoft har tydeligvis fundet en del inspiration fra Java verdenen. Meget kort summeret kan sprogmodellerne i JVM og CLR udtrykkes således:

- Et program er en samling af klasser
- En klasse definerer
    - Data-felter og attributter med typer
    - Metoder med argumenter og muligvis én retur parameter
- Klasser kan indlejres ("nested")
    - JVM – ægte indre klasser; attributer i omkringliggende klasser kan tilgås
    - CLR – indre klasser er kun en navnescope mekanisme
- Metoder kan ikke indlejres i andre metoder
- Dynamisk exception mekanisme
- Concurrency i form af threads

Hvad der sprogligt gør BETA til en udfordring for denne model er bl.a.

- Klasser og metoder er forenet i begrebet "pattern"
- Der er generel indlejring ("nesting") af patterns
- `inner` mekanisme i stedet for `super`
- Generisk parametrisering håndteres med virtuelle patterns
- Multiple return parametre
- Aktive objecter - korutiner
- Ingen constructors
- Ingen dynamiske exceptions (endnu)
- Enter-do-exit semantik
- Leave/restart ud af procedure aktiveringer
- Pattern variable

Vi kan ikke her komme ind på hvorledes alt dette håndteres, men nedenfor forklares lidt af det.

Håndtering af patterns, enter-do-exit og generel indlejring:
Da et pattern i BETA kan bruges både som klasse og som metode er vi for et givet pattern nødt til at generere et antal klasser svarende til de forskellige måder man kan bruge dette pattern. Illustreret ved et eksempel:

```
myClass:
  (# i: @integer;
     R: ^someClass;
     foo1: (# a,b,c: @integer enter(a,b,c) do ... exit c #);
     foo2:< (# n,m: @integer enter (n,m) do ... exit n #);
  #);
```

Udtrykt i en Java lignende syntaks bliver dette til:

```
class myClass extends Object {
   Object origin;
   int i;
   someClass R;

   public myClass(Object org) { // constructor
     origin = org;
   }

   public int foo1(int a, int b, int c) {
       // generate an instance of foo1;
       // execute this instance and return the result
   }

   public myClass$foo1 new$foo1() { return new foo1(this); }

   public int foo2(int n, int m) {
       // generate an instance of foo2;
       // execute this instance and return the result
   }

   public myClass$foo2 new$foo2() { return new foo2(this); }
}

class myClass$foo1 extends Object {
   myClass origin;
   int a,b,c;

   public myClass$foo1(myClass org) { // constructor
     origin = org;
   }
   // methods for enter- do- exit-part of foo1
}
```

```
class myClass$foo2 extends Object {
   myClass origin;
   int n,m;

   public myClass$foo2(myClass org) {
     origin = org;
   }
   // methods for enter- do- exit-part of foo1
}
```

Bemærk: Det er kun for læselighedens skyld at vi her udtrykker det genererede i "Java". For både JVM og CLR gælder, at vi genererer bytekoden direkte; for JVMs vedkommende direkte som binære .class filer, og for CLRs vedkommende som tekstuelle assembler filer.
Læg mærke til at der for det indre pattern `foo1` genereres både en såkaldt *call-method* `foo1`, der svarer til anvendelsen af `foo1` som procedure, samt en såkaldt *new-method* `new$foo1`, der bruges til instantiering ved anvendelsen af `foo1` som pattern. Tilsvarende for `foo2`.
Læg også mærke til at de forskellige constructors har eksplicitte `origin` argumenter, der bruges til at sætte statisk link til omkringliggende objekt op. Dette var det, der, som tidligere nævnt, er nødvendigt vi selv gør, da .NET ikke understøtter indre klasser fuldt ud.
Genereringen af *new*- og *call-methods* kunne naturligvis undlades, men de er medtaget for at gøre mapningen "pæn", så brug af BETA-genererede klasser ser nogenlunde "pæn" ud set fra en anvendelse i et andet sprog.
Dette generelle mønster for oversættelse af patterns giver naturligvis anledning til en stor mængde klasser. For at muliggøre reduktion af dette, har vi indført to nye midlertidige keywords i BETA: `proc` og `class`. Hvis man prefikser sit pattern med disse, signalerer man til compileren, at man kun ønsker kode til denne anvendelse genereret.

Som nævnt kan vi ikke her komme ind på hvorledes alt andet er håndteret, men af de mere specielle tilfælde kan det ganske kort nævnes, at korutiner i BETA er implementeret vha. Threads der eksplicit startes/stoppes; at leave/restart er implementeret ved at kaste en speciel exception ved leave/restart statementet og fange denne ved den tilsvarende label; og at pattern-variable er implementeret vha. de refleksive biblioteker i hhv. Java og .NET.

Compiler og runtime ændringer

Som nævnt i indledningen har vi gennem årene opbygget et ganske pænt abstrakt interface til maskinkodegenerering i BETA compileren. Dette har vi imidlertid måttet lave betragteligt om for at portere til disse to nye platforme!
De væsentligste forskelle mellem de traditionelle platforme og JVM/CLR er:

1. De traditionelle platforme er registerbaserede – JVM/CLR er stakbaseret
2. De traditionelle platforme har global adresserbar hukommelse – JVM/CLR har kun stakken (incl. lokale variable)
3. De traditionelle platformes maskinkode er i det store hele uden typeinformation – JVM/CLR kræver typeinformation for verifikation

Ikke mindst pkt. 3 har krævet et meget stort arbejde for at få type information med helt ud i alle hjørner af kodegeneratoren.
Som nævnt, er det sædvanligvis BETAs runtime system, der er "den anden store hovedpine" ved en portering af BETA. Denne hovedpine er vi imidlertid helt sluppet for her, da JVM/CLR automatisk stiller allokeringsrutiner (i form af allokerings bytekoder) og garbage collector til rådighed ☺

Library portering

En væsentlig opgave ved en portering af systemet er også altid at få porteret de mange BETA libraries. Hvor disse normalt ender med at kalde diverse operativ system rutiner, skal man på bytekode platformene i stedet kalde disses ”class libraries”. Så når vi i BETAs `File` pattern vil implementere `openRead` operationen, så skal vi ud i Javas/.NETs klasse biblioteker og finde nogle operationer, som vi kan kalde. Og disse kald skal vi så kunne udtrykke i BETA.
Som et første skridt på vejen for at opnå dette, har vi indført et prædefineret pattern `ExternalClass`, som man bruger til i BETA at beskrive hvad f.eks. Javas `class File` har af metoder og felter. Og for ikke at skulle sidde og gøre dette manuelt for de mange eksterne klasser (det gjorde vi manuelt i starten ☹), så har vi lavet et tool, der kan åbne en ekstern klasse og automatisk spytte den tilsvarende BETA kode ud. Dette har accellereret porteringen af libraries betragteligt.
Under gennemgangen af vores gamle BETA libraries har vi gentagne gange måttet bide i det sure æble og *rydde op* (!), for at kunne portere koden. F.eks. har vi i stor stil elimineret brugen af fy-fy operatoren `@@` (”address of”), der generelt ikke kan implementeres (verificerbart) på disse bytekode platforme. En hård men naturligvis meget sund process!

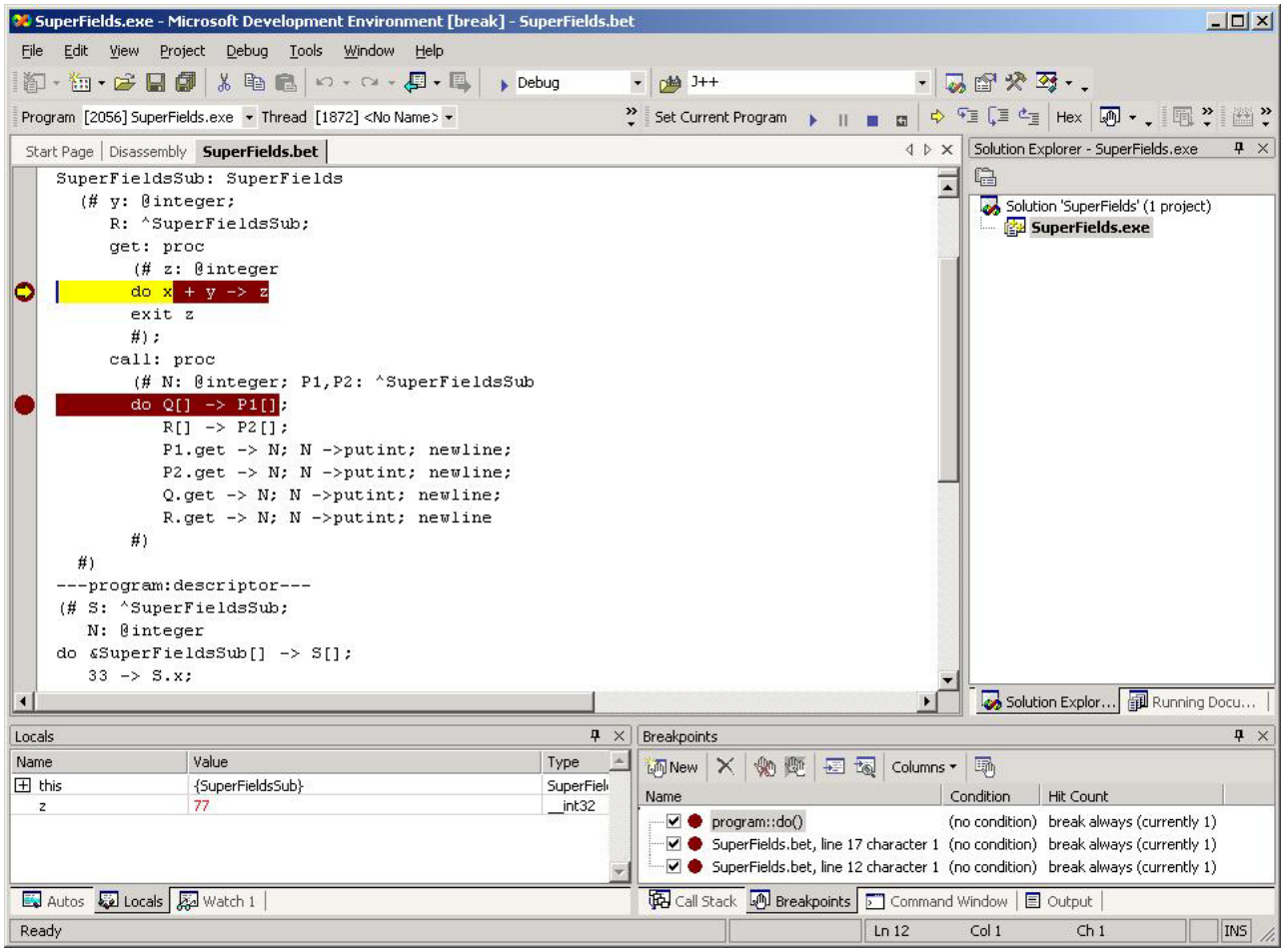
Interoperability

Den overordnede målsætning for dette projekt er som tidligere nævnt ”language interoperability” mellem BETA og JVM/CLR. Hvad mener vi så egentlig med dette?
I traditionelle platforme har man altid benyttet sig af at kunne kalde procedurer skrevet i andre sprog. I det nævnte eksempel ovenfor, hvor man i BETAs `File.openRead` skal kalde et f.eks. UNIXs `open()` procedure, er denne med stor sandsynlighed skrevet i C. Dette har kunnet lade sig gøre vha. traditionelle API’er hvor man har kunnet håndtere en vis mængde data typer som parametre. Microsofts COM og OMGs CORBA gik et skridt videre ved at specificere hvordan snart sagt enhver data type kunne ”serialiseres” og derved udveksles som parametre i procedurekald.
.NET og (viser det sig) Java er så gået endnu et skridt videre ved at lægge op til at man skal kunne definere klasser og metoder i ét sprog, og ikke bare benytte som parametre, men *nedarve og specialisere* i andre sprog.
Dette har vi gjort meget ud af at leve op til i vores BETA portering, og vi har fuldt ud implementeret dette: Hvis man i BETA har erklæret en `ExternalClass` (eller har genereret en vha. ovennævnte tool), kan man uden videre specialisere denne i et sub-pattern i BETA og bruge som ethvert andet BETA pattern. Compileren holder så styr på at kalde videre ud i de eksterne klasser, når dette er påkrævet.
Omvendt kan ethvert BETA pattern bruges som superklasse i Java/C# når først det er oversat med BETA compileren.
Der er naturligvis nogle detaljer der spiller ind når man bruger disse ”cross-language” features. F.eks. er det ikke oplagt hvorledes BETAs `inner` mekanisme (der virker ”nedad”) skal kombineres generelt med JVM/CLRs `super` mekanisme (der virker ”opad”).
Og behovet for et eksplicit omgivende objekt `origin` ved instantieringen af et BETA baseret objekt gør også bruges af BETA klasser en anelse mere besværligt fra andre sprog. Men det virker forbløffende godt. Og vi blev faktisk lidt overrasket, da det viste sig at dette virker præcis lige så godt på Java platformen, som på .NET, selvom dette ikke er et erklæret mål med Java.

Integration i udviklingsomgivelser

For en programmør vil denne ”interoperability” først blive opfattet som fuldstændig når man vil kunne arbejde med BETA i hans/hendes foretrukne udviklingsomgivelse på platformen.

På Windows er state-of-the-art udviklingsomgivelsen Microsoft Visual Studio.NET.
Til vores store glæde kunne vi meget hurtig opnå en hel del integration i denne ved blot at generere bytekode til .NET med linienummer information i: Vi kan åbne BETA filer, sætte breakpoints direkte i denne BETA kildetekst, starte programmet, stoppe ved disse breakpoints, inspicere og rette variable på runtime, inspicere kalds kæden mm. Og vi kan steppe rundt i kildeteksterne i et program, der indeholder både BETA og anden .NET kode, f.eks. C#!



Skærmbillede fra Visual Studio.NET der viser editering og debug af et BETA program. Bemærk at den lokale variabel Z kan ses (og modificeres), og at der kan sættes breakpoints direkte i BETA koden

Fra Microsoft i Cambridge har vi siden fået doneret adgangen til brug af en kommerciel API, der muliggør en endnu tættere integration i Visual Studio.NET. Når vi får tid at gøre dette, vil vi kunne opnå syntaktisk og semantisk farvning af kildeteksten og håndtering af oversættelsen, altså hvorledes BETA compileren skal kaldes af Visual Studio.NET.
På en måde var det jo ikke så overraskende, at en sådan tæt integration i Visual Studio.NET var mulig – det var jo fra start et erklæret mål fra Microsofts side at understøtte multiple sprog.
Det var så til gengæld spændende om noget lignende kunne lade sig gøre i Java verdenen. Dette har nu vist sig at være noget sværere! Vi har undersøgt 15-20 af "de store" Java udviklingsværktøjer, og ingen af disse kunne vi få til at relatere en BETA kildetekst med en herudfra genereret klassefil, selvom vi genererer de nødvendige attributter i klassefilen, der fortæller hvad kildetekst filen hedder. Alle disse værktøjer *forventer* simpelthen at kildefilerne ender på `.java`!

Men at det er det rigtige vi genererer, og at JVM faktisk kan understøtte dette viser en enlig succes vi havde: Den tekstbaserede low-level debugger `jdb` fra SUN kan overraskende nok bruges til at debugge BETA programmer på kildetekstniveau, herunder at sætte breakpoints i kildetekten, inspicere variable osv!

Relation til BETA.Eclipse

Som omtalt i en anden artikel i dette nummer af Daimi Posten, så er der for tiden et andet BETA relateret projekt i gang her på DAIMI. Det hedder BETA.Eclipse, se [11], og handler kort fortalt om at integrere BETA i den IBM understøttede Eclipse udviklingsplatform, på samme måde som vi forsøger at integrere BETA i Visual Studio.NET.
Eclipse er plug-in arkitektur baseret på Java – plug-ins hertil skal simpelthen være Java klassefiler. Da en integration af BETA heri vil indeholde en del eksisterende BETA kode er det oplagt at benytte resultaterne fra BETA.Java til at generere klassefilerne for disse plug-ins direkte fra denne BETA kildetekst.
Dette har vi også fået til at lykkes – undervejs har vi måttet indføre en mekanisme i BETA (for de BETA erfarne: en ny property på linie med INCLUDE etc.) til håndtering af Java packages, da Eclipse forventer at plug-ins ligger i bestemte packages.
Da noget af den oprindelige BETA kode, som skal integreres i Eclipse kalder nogle libraries, der er skrevet i C, er vi i øjeblikket i gang med at få kald af C funktioner fra Java, der er genereret ud fra BETA, til at virke. Javas mekanisme hertil hedder Java Native Interface – JNI – og er temmelig omstændeligt at arbejde med, så dette arbejde er ikke færdiggjort endnu.
Men de indledende forsøg tyder på at vi kan komme langt med denne integration vha. BETA.Java.

Status

Som det fremgår af ovenstående er vi kommet temmelig langt med projektet, men har alligevel et godt stykke endnu. Vi arbejder i øjeblikket mest med to emner: Dels som nævnt at understøtte JNI til BETA.Eclipse, og dels med at ”boote” BETA compileren på JVM platformen. Vi er her kommet til at compileren kan oversættes og starte op, men herefter crasher programmet pga. fejl i det genererede.
Udover vores understøttelse af BETA.Eclipse forestiller vi os også at kunne tage del i eventuelle DAIMI projekter baseret på den af Lars Bak m.fl. udviklede virtuelle maskine OOVM, se [13], idet vi forventer at en portering til denne Smalltalk baserede maskine vil være rimelig smertefri, når vi først er kommet igennem på JVM og CLR.
Vi forventer senere at producere nogle artikler om vores erfaringer med interoperability generelt, og specielt en sammenligning af CLR og JVM som generelle sprogplatforme.
Vores projekt har selv et par hjemmesider, se [11].

Referencer

1. BETA sprog hjemmeside: http://www.daimi.au.dk/~beta
2. Mjølner System (BETA udviklingsomgivelse) hjemmeside: http://www.mjolner.com/mjolner-system
3. O. Lehrmann Madsen, B. Møller-Pedersen, K. Nygaard
   Object Oriented Programming in the BETA Programming Language
   ISBN 0-201-62430-3, Addison Wesley, juni 1993.
   PDF: ftp://ftp.mjolner.com/BETA/BOOK/betabook.pdf
4. Java hjemmeside: http://java.sun.com
5. Microsoft .NET hjemmeside: http://www.microsoft.com/net

6. Microsoft shared source common language infrastructure (ROTOR): http://msdn.microsoft.com/net/sscli/
7. Project Mono: http://www.go-mono.com/
8. DotGNU: http://www.dotgnu.org
9. .NET languages: http://www.gotdotnet.com/team/lang/
10. Projekt websider: http://www.pervasive.dk/projects/langInter/langInter_summary.htm og http://www.daimi.au.dk/~beta/ooli
11. BETA.Eclipse websider: http://www.pervasive.dk/projects/Eclipse/Eclipse_summary.htm og http://www.daimi.au.dk/~beta/eclipse
12. Programming Languages for the Java Virtual Machine: http://www.robert-tolksdorf.de/vmlanguages
13. Object-Oriented Virtual Machines: http://www.oovm.com/