Integrering av Java i TACOMA

Petter Reinholdtsen
<pere@td.org.uit.no>

27. mai 1996

Sammendrag

Denne rapporten beskriver en enkel integrering av programmeringsspråket Java i TACOMA, en infrastructur for internet-agenter. Integreringen gjøres som bindinger til C-funksjoner.

1 Innledning

TACOMA, Tromsø And COrnell Moving Agents [1], er en infrastruktur for flyttbare agenter. En slik infrastruktur muliggjør automatisk flytting og utføring av program-biter mellom maskiner i et netverk. TACOMA tar sikte på å tilby en abstraksjon som er uavhengig av programmeringsspråk og maskinarkitektur. Denne rapporten beskriver integreringen av programmeringsspråket Java [3] i TACOMA.

Da jeg startet med å integrere Java i TACOMA var det kun et språk, Tcl, som kunne brukes sammen med TACOMA. Tcl virker rotete og tregt, så jeg ønsket et alternativ. Da jeg ønsket meg mer erfaring med Java, valgte jeg å integrere dette som alternativt språk.

Java ble i utgangspunktet utviklet av SUN for å ha et deterministisk og arkitekturuavhengig språk å programmere konsument elektronikk. Det er multi-treaded og objektorientert, og ligner endel på C++. I tillegg tilbyr det en API som er generell på tvers av OS og arkitektur. Det siste året har det fått gjennomslag som w3s "applet"-språk. Det muliggjør utvidelser i w3-framviserne. "The NPAC Visible Human Viewer" [5] og "The Improved Sorting Demo" [4] er gode eksempel på det siste.

Java og agenter er et aktivt område for tiden. Det jobbes med integrasjon av Java i TACOMA ved Computer Science Department på Cornell University [2]. Her har de valgt å konvertere hele TACOMA til Java. Jeg har ikke sett på deres løsninger.

I tillegg finnes for eksempel "Personal Media Arbitration Protocol" [6] (PMAP) som jobber med å lage en lignende infrastruktur med helt andre abstraksjoner. PMAP skal kunne støtte flere ulike språk, deriblant Java.

2 Hva er Java

Programmeringsspråket Java ble lansert av SUN ved hjelp av deres webbrowser HotJava. Denne framviseren er skrevet i Java som et eksempel på anvendelsesområdene til språket. Denne har mulighet for å kjøre applets, nedlastede programmer, lokalt på egen maskin. Slike applets skal kjøre i en sikker omgivelse som ikke tillater ondsinnede programmer å ødelegge for brukeren. Dette har vist seg å ikke holde vann, da det er oppdaget flere brudd på den lovede sikkerheten det siste halve året..

Java er et flertrådbasert og objektorientert språk med enkel arv mellom klassene. Hukommelseshåndteringen er basert på garbage collection, i motsetning til C++. Det har et kraftig systembibliotek med billedbehandlingrutiner og abstrakte datatyper i tillegg tilde vanlige systemrutinene.

De første implementasjonene av Java oversatte til en mellomkode (Bytecode) som ble kjørt på virtuelle Java-implementasjoner. Denne mellomkoden er felles for alle arkitekturene Java kjører på, og kompilerte programmer kan dermed brukes uavhengig av prosessortype. Det finnes i tillegg kompilatorer for å oversette Java til maskinkode.

Systembibliotekene skal være felles for alle arkitekturene. Dette skal gi samme utseende og oppførsel både under X og Windows.

3 Java i TACOMA

Å integrere et nytt språk i TACOMA er en todelt oppgave. Først må det lages et bibliotek med APIen for å understøtte agent-abstraksjonen. Deretter må det lages en agent for å ta imot og starte opp agenter skrevet i det nye språket. Jeg valgte å kun ta imot bytecode for å unngå problemer med kompilering og skrivefeil i Java-koden.

TACOMA er i utgangspunktet ikke objektorientert. For å få en sømløs integrasjon av Java i TACOMA bør det lages et objektorientert design for å identifisere og spesifisere objekt-metoder. Dette innebærer i hovedsak å endre notasjonen for APIen. Jeg har valgt en annen innfallsvinkel, der gjeldende APIen blir operasjoner på ett enkelt TACOMA-object i Java.

Java-agentene får inn to parametre (`TACOMA` og `briefcase`) der første parameter er TACOMA-objektet med den tilgjengelige APIen.

Koden blir dermed slik (Uten exeptions etc):

```
import TACOMA;
class TACOMA_AGENT
{
    public void Main(TACOMA world, briefcase bc)
    {
        String text = world.folder_shift(bc, "DATA", world.RIGHT, 1, "");
        world.folder_store(bc, "DATA", "No message of the day at host");
        world.meet("ag_echo", bc, 0);
    }
}
```

En del av tanken bak bruken av agenter er at de skal fungere i en mobil omgivelse. Da mobile maskiner gjerne har færre ressurser enn stasjonære maskiner, vil lagrings og hukommelseskravene i TACOMA ha vesentlig betydning for anvendelsen. Lagringskravene for Java er rundt 3 MB for systembibliotek og kjørefiler. Dette burde ikke legge store begrensninger på anvendelsen til Java. Jeg har ikke undersøkt hukommelseskravene til Java.

I TACOMA har man foreløbig sett helt bort fra sikkerhetsproblemer ut fra tanken om at du ikke slipper inn agenter du ikke stoler på. Dette har enklet implementasjon og integrering av nye språk. Jeg har ikke vurdert hvor store sikkerhetsproblemer Java vil bidra med i TACOMA. I og med at TACOMA ikke forsøker å gi noen garantier for sikkerheten til maskinen hvis det utføres ondskapsfulle agenter, så tror jeg ikke Java tilfører noen nye problemer.

Sikkerheten ved kjøring av Java-programmer kan muligens økes ved å fjerne deler av systembiblioteket. Ved å fjerne Nett– og filtilgangs-primitiver og sørge for at all tilgang til eksterne deler går via TACOMAs API så kan man sørge for å få samlet all tilgangskontroll på en plass. Jeg har ikke forsøkt dette.

3.1 Java API

Jeg valgte å binde Java APIen direkte opp mot APIen i C. Dette gjøres ved å lage en klasse med såkalte `native` metoder. Alle C-funksjonene får således en metode i den globale TACOMA-klassen.

Java-klassen som kompileres ser da slik ut:

```
class TACOMA
{
    native briefcase bc_create();
```

```
    [...]
    // Load library
    static {
        System.loadLibrary("TACOMA");
    }
}
```

For å få bindingen mellom Java og C brukes først stub-kompilatoren (`javah -stub`) for å lage `TACOMA.c`. Denne inneholder kall til C-funksjoner som har navn etter klasse-operasjonene. Eksempelfunksjonen blir hetende `void *TACOMA_bc_create(void *this)`. Implementasjonen er triviell:

```
void* TACOMA_bc_create(void *)
{
    return bc_to_containter(bc_create());
}
```

C-biblioteket bruker struct-pekere for å representere arkivskap og kofferter. Pekere er vanskelig å representere i Java. Jeg har derfor valgt å lagre C-pekerens verdi som en string i en intern klasse (container) i Java. Denne verdien konverteres i C-stubbene før og etter C-kallene og skal aldri brukes i Java.

Det er ikke gjort noe for å hindre feil og problemer som følge av flertråd-støtten til Java. Det kan oppstå problemer hvis flere tråder bruker C-biblioteket samtidig.

Det er ikke lagt inn korrekt håndtering av exeptions i C-rutinene. Det blir dermed ingen god håndtering av feil med denne måten å integrere Java i TACOMA.

3.2 Oppstart av Java-agenter

En egen agent (ag_java) startes opp når en Java-agent kommer inn fra nettet eller egen maskin. Denne mottar en koffert som inneholder bytecoden som skal utføres.

Når ag_java mottar kofferten ser den i mappen JAVA_BYTECODE for å finne navnene på folderne med bytecoden. Disse folderne lagres på disken med filnavn likt navnet på folderen pluss '.class'. En av folderne skal hete TACOMA_AGENT. Skråstrek ('/') tolkes som katalognavn-skille og det opprettes en underkatalog for filen. Det siste er nødvendig for å bruke Javas klassehierarki.

Når alle bytecode-filene ligger klar på disken starter ag_java opp en java-stub som laster inn klassen TACOMA_AGENT, oppretter et objekt av samme type og kaller deretter operasjonen `void Main(TACOMA, briefcase)` på objektet. Første parametret er et objekt der operasjonene er APIen til TACOMA. Det skal kun være et TACOMA-objekt per agent. Det andre parametret er kofferten som agenten mottar ved oppstart.

4 Konklusjon

Jeg har her vist hvordan man på en enkel måte kan integrere programmeringsspråket Java i TACOMA. Dette var en enkel oppgave på grunn av enkel binding mellom C og Java. Den valgte metoden for integrering bruker ikke Javas styrke med feilhåndtering (exeptions) eller utnytter de objektorienterte aspekter av språket.

5 Tilgjengelighet

Mer informasjon om og siste versjon kan finnes på TACOMAs hjemmeside [7] ved Seksjon for Informatikk.

Referanser

[1] Dag Johansen, Robbert van Renesse, and Fred B. Schneider. An introduction to the tacoma distributed system. Technical report, Seksjon for Informatikk, Universitetet i Tromsø, 1995. [http://www.cs.uit.no/Lokalt/Rapporter/Reports/9523.html].

[2] Dag Johansen <dag@cs.uit.no>. Personlig kommunikasjon.

[3] SUN microsystems. Java: Programming for the internet. Elektronisk dokument, 1996. [http://java.sun.com/].

[4] Pat Morin. The improved sorting algorithm demo. Elektronisk dokument, Mars 1996. [http://turing.scs.carleton.ca/~morin/sortalg/].

[5] The npac visible human viewer. Elektronisk dokument, 1996. [http://www.npac.syr.edu/projects/vishuman/VisibleHuman.html].

[6] Personal media arbitration protocol. Elektronisk dokument, 1996. [http://www.mirai.com/PMAP/].

[7] The tacoma project. Elektronisk dokument, Mai 1996. [http://www.cs.uit.no/DOS/Tacoma/].

Om forfatteren

Petter Reinholdtsen er en laveregradsstudent i Informatikk ved Universitetet i Tromsø. Hans hovedinteresse er internet-tjenester og Unix drift. I tillegg til studiene jobber han deltid som webmaster og utvikler ved universitetet og på Telenor Origo. Han har vært aktiv på Internet siden høsten 1992, da han samlet det første laget for å delta i konkuransen The Internet Hunt.