Java är rätt val för kom

Passar perfekt till grafiska användargränssnitt och uppkopplade nätverk

EXPERTARTIKEL

Av Régis Latawiec, COO på IS2T

Régis Latawiec har arbetat 23 år med mjukvara och hårdvara till inbyggda system. Han började på IS2T år 2007 och är ansvarig för produktportföljen, affärsutveckling och kundvård. Tidigare har han arbetat på Alcatel med GSM-teknik och på Atmel med 32-bitars styrkretsar.

C-utvecklare brukar se tre problem med Java. Det är minnesanvändningen, exekveringshastigheten och integrationen av existerande C-kod.

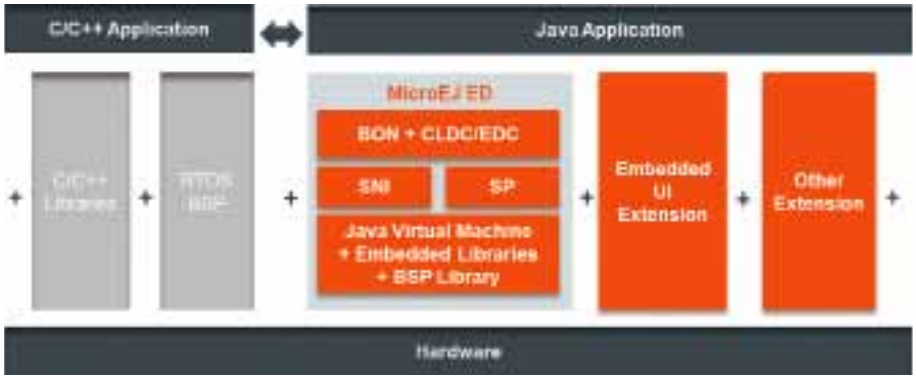
De flesta C-utvecklare arbetar med små inbyggda system där minnesanvändningen har stor betydelse eftersom målsystemet är en styrkrets med ett litet integrerat flashminne eller om systemet är lite vassare är det redan nedlusat med komplex programvara.

Nya tekniker för att implementera Java minskar minnesbehovet. Som exempel tar vi en Cortex-M3/M4. Javaplattformen har en full implementation av grafikbiblioteken till MicroUI och MWT för att bygga grafiska användargränssnitt (GUI) till displayer med matriser av färgpunkter (ESR002 och ESR011 refererar till ESR-konsortiet för API-specifikationer www.e-s-r.net).

EFTERSOM VI KAN ANSE att fotavtrycket för Javabiblioteken är ungefär desamma som deras motsvarigheter i C-världen är den faktiska kostnaden för tillgången till Java bara 28 kbyte (den virtuella Javamaskinen).

Exekveringshastigheten bestäms av den virtuella Javamaskinens prestanda men också av det tillhörande C-biblioteket. Kom ihåg tumregeln som säger att vanligen spenderas 80 procent av tiden av 20 procent av koden, normalt lågnivåbiblioteken.

MicroEJ:s plattformar har en så enkel virtuell Javamaskin och högt optimerade bibliotek att den genomsnittliga exekveringshastigheten jämfört med funktionsmässigt identisk C-kod i normalfallet inte är mer än 20 procent långsammare. Om C bjöd på samma runtimefunktioner som Java – automatisk minneshantering, kontroll av minnesaccess, och så vidare enligt ovan – skulle förstås Javas exekveringshastighet vara mycket högre än C.



Figur: Typiskt minnesbehov för en Javaplattform.

Integrationen i Java av äldre och stabil kod (C/assembler) måste vara enkel för att garantera utvecklarna en enkel och säker väg. Existerande drivrutiner och processorspecifika uppgifter måste vara tillgängliga för den nya Javakoden samtidigt som man måste hålla i minnet att overhead för anrop liksom allokering av minnes- och CPU-resurser måste vara effektiva.

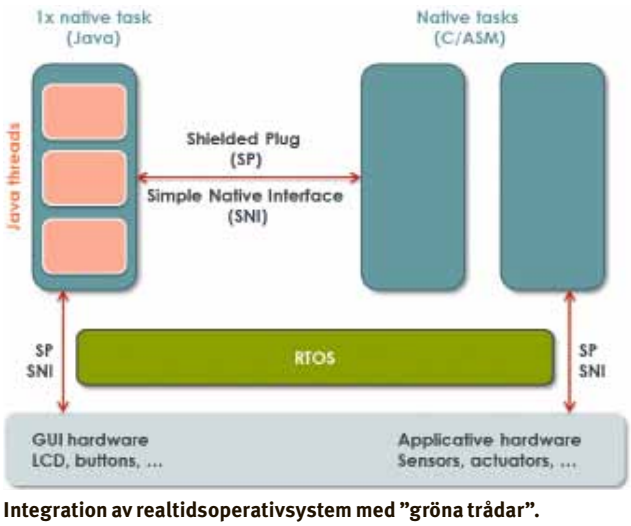
MicroEJ:s utvecklingspaket är byggt på den populära utvecklingsmiljön Eclipse. Den ger en komplett utvecklingsmiljö för att skapa Javaplattformar och Javaapplikationer för inbyggnadstillämpningar.

Utvecklarna kan skapa prototyper, utveckla och testa sina Javaapplikationer i simulerade Javaplattformar innan de sjösätts i hårdvara.

DET KRÄVS TVÅ SAKER för att integrera en Javaplattform i ett existerande och C-baserat system. För det första måste man definiera en konfigurationsprofil för Javaplattformen och för det andra ett gränssnitt till hårdvaran och den befintliga koden (C/assembler) mot Javabiblioteket och applikationslagren.

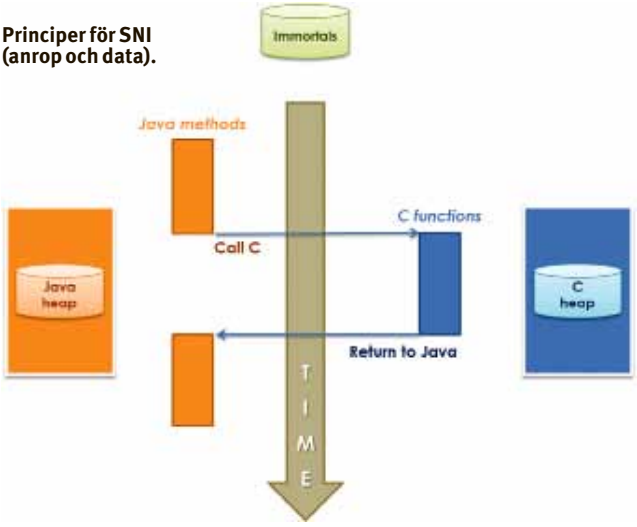
För att kunna behålla den existerande C-applikationen intakt, måste man behålla den existerande utvecklingsmiljön oförändrad. MicroEJ stödjer ARM Keil uVision, IAR Workbench och GNU gcc men även andra varianter kan användas med MicroEJ.

I utvecklingsmiljön måste man börja med att välja arkitektur för Javaplattformen (Cortex-M, Renesas RX, osv...) liksom verktygskedja för C. Sen kan den grundläggande Javaplattformen utökas med bibliotek som Embedded UI, processorspecifik C/Java-koppling, seriella gränssnitt, et cetera.



Integration av realtidsoperativsystem med ”gröna trådar”.

plexa applikationer



Principer för SNI (anrop och data).

```
package GPIO;

public class Main {
    public static native void toggle();

    public static void main(String[] a) throws InterruptedException
{
        while(true){
            toggle();
            Thread.sleep(10);
        }
    }
}
```

```
#include <sni.h>
#include "gpio.h"

void Java_GPIO_Main_toggle(){
    GPIOE->ODR ^= GPIO_Pin_2 ;
}
```

Här är ett exempel på hur Javakod kan anropa C-kod.

När biblioteken är valda och konfigurerade (antal bitar per pixel i användargränssnitten, antal seriella COM-portar, med mera) kan man bygga plattformen för att sedan skapa en uppsättning ELF-filer som är färdiga att användas med C-utvecklingsmiljön.

NÄSTA STEG ÄR ATT importera projektet för Javaplattformen i C-utvecklingsmiljön och koppla ihop den med de unika delarna i ditt Board Support Package inklusive realtidsoperativsystem och skärmdrivrutiner till Javaplattformens lågnivå API:er. Andra delar av din existerande applikation i C-kod kan också adderas.

Att lägga till fler mjukvarufunktioner till en existerande applikation brukar kräva att man adderar ett enkelt realtidsoperativsystem (vanligen ett pre-emtive) eftersom applikationen behöver kunna svara på många interaktioner (vanligen från användaren eller nätverket) även om uppgifter körs i bakgrunden.

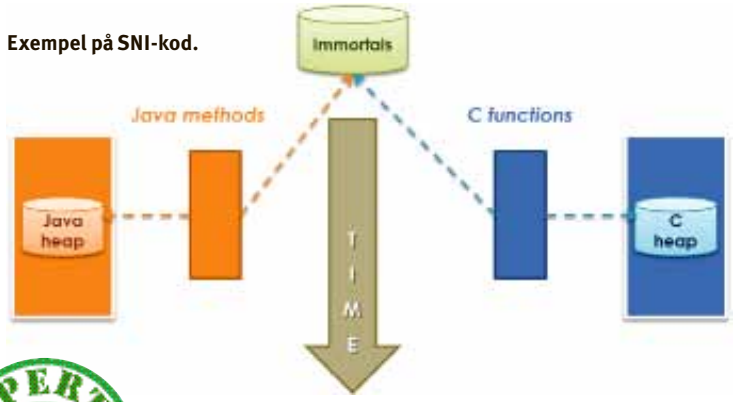
Integration av ”gröna trådar” är den vanligaste vägen att addera en virtuell Javamaskin till ett existerande mjukvarusystem. Javaplattformen definierar en multitrådad miljö för hela Javaapplikationen som körs i en enda realtidstråd. Den CPU-resurs som behövs för Java styrs av realtidstrådens konfiguration. Det här angreppssättet gör det lättare att fördela om mellan de olika uppgifterna i applikationen. Den största CPU-kraft som behövs allokeras till Javadelen kan låsas enligt antalet gröna trådar eller aktiviteterna i dessa.

Det finns flera olika sätt att koppla Java till C för att få access till drivrutiner eller äldre firmware. En implementation kallas Simple Native Interface eller SNI (ESR012). Java Native Interface är ett annat sätt att koppla in processorspecifik kod men är mycket mer komplext och har onödiga funktioner som objektmanipulering som inte är användbara för styrkretsbaserade tillämpningar.

Anrop via SNI:er är rättframt. De tillåter direkta anrop av C-funktioner från Java och av Javametoder från C. Argumenten kan vara heltal eller flyttal och anges när man överför data.

MÅNGA APPLIKATIONER har mjukvaruarkitekturer som baseras på att processorerna körs oberoende av varandra och utan interaktioner men som kräver mekanismer för att utbyta data. I sådana situationer publiceras data av ”producenter” och ”konsumenter” använder data efter det att de tecknat ett abonnemang. Den här typen av arkitektur används vanligen i industriella styrsystem, automatisk avsyning, medicinska system och på telekomområdet. ▶

Exempel på SNI-kod.




```
package com.corp.examples;
public class Hello {

   static int[] array = (int[])Immortals.setImmortal(new int[50]);
   public static native int getData(int[] array);

   public static void main(String[] args){
      int nb = getData(array);
   }
}
```

```
#include <sni.h>

jint Java_com_corp_examples_Hello_getData(jint* array){
    array[0] = 0xBEEF;
    return 1 ;
}
```

En så kallad Shielded Plug, SP, behövs när Javaapplikationen ska kopplas till äldre hårdvara. Både (i) logiken och (ii) hela apparaten utbyter data genom SP:n.

SP ger en väldefinierad avgränsning mellan producenter och konsumenter av data. Dessutom behöver dessa inte exekveras från samma värld. SP ger en komplett segregation av processerna (producenter och konsumenter) och kan skrivas i C eller Java.

SP:n talar om när data i ett block har ändrats, kan reinitialisera ett block, kan kontrollera om data är tillgängligt och har mekanismer för att göra data seriellt liksom att ta mot seriell data och göra om det till andra format. Datastrukturerna är beskrivna i ett fristående språk (XML).

Slutsats

Vi har visat att Java är den rätta lösningen för att utveckla komplexa applikationer och speciellt för produkter som behöver ett grafiskt användargränssnitt eller är uppkopplade till ett nätverk. Konceptet med ett objektorienterat språk kombinerat med en virtuell plattform erbjuder många fördelar för utvecklare samtidigt som det håller hårdvarukomplexiteten och kostnaderna på samma låga nivå som för C.

På kort sikt ökar Java produktiviteten och gör det snabbare att få fram en produkt genom att förenkla processerna för mjukvaruutveckling. Det förbättrar också värdet på produkten genom att göra det möjligt för utvecklarna att släppa loss sin kreativitet genom att använda moderna programmeringsspråk. På lite längre sikt ger Java access till den stora utvecklingsgemenskapen, förbättrad projektstyrning, att återanvända mjukvara för att kapitalisera existerande kod och enklare förvaltning tack vare Java skalbarhet och portabilitet. ■