Iedereen heeft er in zijn jeugd vast wel eens mee gespeeld: de racebaan. Met een controller kon je het raceautootje sneller en langzamer laten rijden door de knop verder in te drukken of los te laten. Op de rechte stukken ging je voluit, maar pas op die bochten. Voor je het weet ligt je racewagen naast de baan.

Let's race with real-time Java: RTSJ

JavaOne Slotcar Challenge

Deze belevenis kwam weer helemaal terug tijdens JavaOne 2006. Greg Bollella, Distinguished Engineer bij Sun Microsystems had de JavaOne Slotcar Challenge opgezet. Het doel van deze wedstrijd was om te laten zijn hoe eenvoudig het programmeren met real-time Java is. Om de race goed te kunnen begrijpen, kijken we eerst naar de betekenis van 'real-time'. Veel mensen verwarren 'real-time' met 'heel snel'' omdat men denkt dat dit equivalenten van elkaar zijn. De wetenschappelijke definitie van 'real-time' betekent: 'De mogelijkheid om betrouwbaar en voorspelbaar om te gaan met en het regelen van tijdsafhankelijke programmalogica'. Veel is afhankelijk van de eisen die je stelt aan de te halen deadlines. Als je een programma schrijft dat als gevolg van een toetsaanslag een letter op het beeldscherm toont, en dit moet binnen zes dagen gebeuren, dan is dat haalbaar. Het wordt een stuk interessanter als deze actie binnen enkele milliseconden moet gebeuren.

Sensoren De racebaan die gebruikt werd, was uitgerust met 82 sensoren. Deze sensoren gaven een signaal af, op het moment dat het raceautootje passeerde. Om deze sensoren uit te lezen, heb je geen zwaar systeem nodig. Maar als op één of andere manier de applicatie vertraagd is (bijvoorbeeld omdat je systeem even andere dingen aan het doen is), dan gaat het mis. Je signaal wordt misschien wel verzonden, maar niet tijdig (of helemaal niet) door het systeem verwerkt. Een bepaalde activiteit (in dit geval het uitlezen van de sensor en een handeling uitvoeren) moet uitgevoerd kunnen worden in een relatief korte deadline. In dit geval was dat een deadline van vijf milliseconden.

Java-technologie werd nog niet veel gebruikt in real-time systemen, omdat Java niet in staat was om aan de eisen van de real-time wereld kon voldoen. Redenen hiervoor waren onder andere de onvoorspelbare vertragingen van de JVM die veroorzaakt worden door de Garbage Collector, onvoldoende mogelijkheden voor scheduling en zeer grove timer support.

> *"I want to get the message across to the developer community: Real-time Java isn't rocket science. It's more like bicycle science." Greg Bollella, Distinguished Engineer bij Sun Microsystems*

Het programmeren van real-time systemen kan redelijk primitief zijn, in vergelijking met het gemak waarmee nu ontwikkeld kan worden in Java. Om gebruik te kunnen maken van Java in real-time systemen werd JSR-1 (Java Specification Request)[1] binnen de JCP (Java Community Process[2]) opgestart. Hierbinnen werkte een team van experts uit de real-time wereld aan een specificatie voor real-time Java. Deze JSR is opgestart in 1998 en was beschikbaar in 2002. Tijdens de slotcar challenge is gebruikt gemaakt van de Sun-implementatie van RTSJ. Deze implementatie bevat onder andere een tweetal nieuwe threading-mogelijkheden, realtime-thread en no-heap real-time thread. De laatste

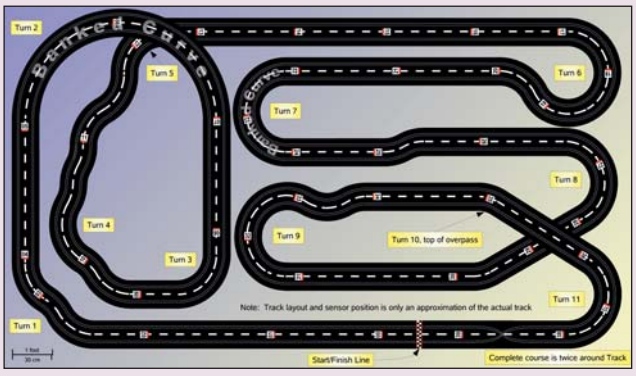
kan niet onderbroken worden door garbage collection. De threads voorzien in een preciezere scheduling dan de threads die gebruikt worden in de Java Standard Edition. Tevens zijn er uitbreidingen gedaan op het memory-management. Er zijn twee nieuwe typen aan toegevoegd. Immortal memory kan objecten bevatten die alleen zullen verdwijnen wanneer het programma eindigt. Objecten die in dit gebied worden aangemaakt, moeten dus met zorg aangemaakt en behandeld wor-

Het programmeren van real-time systemen kan redelijk primitief zijn, in vergelijking met het gemak waarmee nu ontwikkeld kan worden in Java

den. Scoped memory zorgt voor het beheer van objecten in een bepaalde sectie of scope, zoals bijvoorbeeld binnen een methode. Alle objecten worden automatisch verwijderd op het moment dat de methode wordt verlaten. Voor geen van beide geheugen-strategieën wordt garbage collection uitgevoerd. Ook zijn er aanpassingen gedaan in het timersysteem. Absolute tijd en relatieve tijd zijn binnen RTSJ op te vragen tot op de nanoseconde.

Naast een real-time implementatie voor Java is er ook gewerkt aan een plugin voor de NetBeans IDE[3]. Hierdoor kan volledig gebruik gemaakt worden van de functionaliteiten van de NetBeans IDE, waarbij ook specifiek de mogelijkheden van het RSJ platform benut kunnen worden.

Tijdens eerdere conferenties werden reeds praktijkvoorbeelden getoond. Het bekendste voorbeeld is de 'Inverted Pendulum'[4]. Hierbij moet een slinger, die kan draaien rondom één draaipunt, rechtopstaand gehouden worden door het draaipunt naar links of naar rechts te laten bewegen. Tijdens JavaOne 2005 werd een demonstratie getoond van het Scaneagle-project van Boeing. In dit onbemande verkenningsvliegtuig zorgt real-time Java voor een betere navigatie en besturing. Inmiddels is de Scaneagle al diverse malen in de praktijk ingezet.



De racebaan

JAVAONE SLOTCAR CHALLENGE Op JavaOne is een racebaan geplaatst uitgerust met 82 sensoren. Alle sensoren waren met elkaar verbonden en gaven bij het passeren van de racewagen een signaal af. Een programma kan vervolgens peilen of de racewagen zich boven een sensor bevindt. Je weet op dat moment niet, welke sensor het is. Door een eigen administratie bij te houden, weet je waar de auto zich bevindt. Afhankelijk van de plaats kan vervolgens een spanning (tussen -3,7 en 3,7 volt) naar de baan gestuurd worden. Het uiteindelijke doel is om de auto zo snel mogelijk over de racebaan te laten rijden. Natuurlijk is het nog steeds mogelijk om de auto met hoge snelheid uit de bocht te laten vliegen.

De codevoorbeelden zijn wat vereenvoudigd ter verbetering van de leesbaarheid. De twee belangrijkste klassen die gebruikt zijn, zijn ContestantRTT en ContestantNHRT. ContestantRTT zorgt voor het opstarten van de NoHeapRealtimeThread die de eigenlijke controle-logica uitvoert. Na het opstarten, leest deze thread-informatie uit en toont deze op het scherm.

```
public class ContestantRTT extends
RealtimeThread {
    public void run() {
        sp = new PriorityParameters(Priority
Scheduler.instance().getMaxPriority());
        pp = new PeriodicParameters(Clock.
getRealtimeClock().getTime().add(1000,0),
                                        new
RelativeTime((long)1,0), null, null, null,
null);
      nhrt = new ContestantNHRT(io, sp, pp);
      nhrt.start();
    }
    ......
}
```

De ContestantNHRT start de racewagen door de spanning 'superFastV' uit te sturen naar de baan. Vervolgens wordt een loop gestart, die afbreekt op het moment dat er binnen 3 seconden geen sensor meer is waargenomen. Zeer waarschijnlijk is de auto dan van de baan af gevlogen of staat stil. Deze implementatie kijkt wanneer de wagen een sensor passeert en zoekt in een van te voren geïnitialiseerde tabel op wat de spanning is, die op dat punt naar de baan gestuurd wordt.

```
public class ContestantNHRT extends
NoHeapRealtimeThread{
    public void run() {
```

```
        io.setVoltage(superFastV);
// Floor it!
        while (++deltaT < 3000) {
// If no sensor in 3 seconds then timeout as
// car must have gone off the track
            if (io.carOverPositionSensor())
{
                if (!overSensor) {
                    overSensor = true;
// Just registered next sensor - update speed
                    sensorNum++;
                    elapsedTime += deltaT;
                    deltaT = 0;
                    if (sensorNum >= 80) {
                        break;
// All done when we see Sensor 0 after two
laps
                    }

          io.setVoltage(Vs[sensorNum % 80]);
// Set new speed voltage
                }
            } else if (overSensor && deltaT
> 120) {
                overSensor = false;
// Reset indicator 120ms after passing the
sensor
            }
            while (!waitForNextPeriod()) {
deltaT++; }
// Update time if we miss any periods
        }
        io.setVoltage(0);
     // Stop --- we are done
        running = false;
    }
```

De implementatie is natuurlijk verre van optimaal, maar toont hoe eenvoudig de programmakant van het real-time programmeren werkt. Nu kun je de nodige natuurkundige formules loslaten op dit systeem en deze modelleren in de software. Een optimalisatie is bijvoorbeeld het acceleratie- en rempatroon. In deze implementatie wordt de spanning bij het passeren van een sensor direct aangepast. Het is mogelijk om dit door middel van timers geleidelijk te laten verlopen.

De finale van de slotcar challenge

Tijdens JavaOne konden de deelnemers via een JavaWebStart applicatie de gebouwde JAR-file naar de centrale raceserver sturen om vervolgens achter aan de rij aan te sluiten voor een test-ronde. Deze testrondes werden uitgevoerd door de baan-operators. Zij startten het geschreven programma en met eigen ogen kon je zien wat de door jou geschreven code voor resultaat had. De drie inzendingen met de snelste tijd, reden tijdens afsluitende algemene sessie tegen elkaar voor een publiek van meer dan 10.000 aanwezigen. De uiteindelijke winnaar ontving de hoofdprijs, een toegangspas voor JavaOne 2007. Enkele honderden JavaOne bezoekers hebben een gokje gewaagd en met hun programma de raceautootjes laten rijden. Het was net zo leuk als in de eerdere dagen, maar nu met een hoger *geek*-gehalte!

De setup

De slotcar challenge werd aangestuurd door een breed scala aan hardware. De sensoren waren zeer eenvoudige licht-sensoren, die aangesloten waren op het dochter-board van de Quasar Q8[5]. Dit board was geplaatst in een Solaris 10 workstation. Dit workstation zorgt voor de complete afhandeling van het proces. Via de Q8 worden de signalen gemeten en de spanning op de racebaan aangestuurd. De logica is gebouwd boven op de "Sun Java Real-Time System"[6].De hardware is afkomstig van Quasar, een leverancier gespecialiseerd in sensor- en controller-componenten. Quasar levert ook diverse kant en klare set-ups die gebruikt kunnen worden in een educatieve omgeving.

Klaasjan Tukker is voorzitter van de NL-JUG.

[1] JSR-1 Real-time Specification for Java : http://www.jcp.org/en/jsr/detail?id=1
[2] JCP: http://www.jcp.org/
[3] Java RTS NetBeans module : http://www.netbeans.org/kb/articles/java-rts.html
[4] Inverted Pendulum: http://www.quanser.com/english/html/challenges/fs_chall_linear_flash.htm
[5] Quasar Q8: http://www.quanser.com/english/html/solutions/fs_Q8.html
[6] Sun Java Real-Time System: http://java.sun.com/j2se/realtime/