# Automatische code generatie met de TimesTool

Bachelorscriptie
Laura Nij Bijvank 0213934
Radboud Universiteit Nijmegen
28-01-2008

# Inhoudsopgave

# 1 Inleiding

Er bestaan een aantal tools, zoals UML tools als Telelogic Rhapsody [1] en Rational Rose [2] om automatisch code te genereren. Het voordeel van het gebruik van zulke tools is dat er gewerkt wordt met een hoger niveau van abstractie, zodat er meer overzicht is. Ook is het handiger voor onderhoud en kan het model en de documentatie hergebruikt worden. Deze tools houden echter over het algemeen geen rekening met timing. In onderzoekstools als UPPAAL [3.2] valt timing echter wel goed te modelleren en te analyseren, maar deze kan dan weer geen code genereren. De TimesTool [4] zou beide moeten kunnen volgens de ontwerpers. Met de Timestool kan een model gemaakt worden van een systeem, er kan schedulability analyse mee op het model uitgevoerd worden en het kan code genereren uit het model. Het is een academische tool, voor onderzoek, en de code generatie is gericht op de LEGO RCX (zie 2.1) Het is bedoeld voor het modeleren en implementeren van embedded systemen die beschreven kunnen worden als een verzameling taken die gescheduled moeten worden. Maar bied het gebruik van de automatische code generatie nu ook voordelen boven ‘gewoon’ programmeren? Deze vraag hebben we onderzocht en het resultaat wordt beschreven in deze scriptie.

## *1.1 Probleemstelling*

We zullen nu de centrale probleemstelling van ons onderwerp formuleren.

**Onderzoeksvraag**

Wat zijn de voor- en nadelen van het genereren van code door de TimesTool voor de RCX ten opzichte van het zelf programmeren bij een relatief klein voorbeeld?

## *1.2 Deelvragen*

Om de onderzoeksvraag te kunnen beantwoorden is het handig om eerst een antwoord krijgen op een aantal deelvragen. Deze zijn:

**Deelvraag 1:**

Hoe lang duurt het om de TimesTool te leren?

**Deelvraag 2:**

Waar wordt de computation time, die nodig is in de TimesTool, voor de verschillende taken berekend?

**Deelvraag 3:**

Doet de gegenereerde code wat je verwacht?

**Deelvraag 4:**

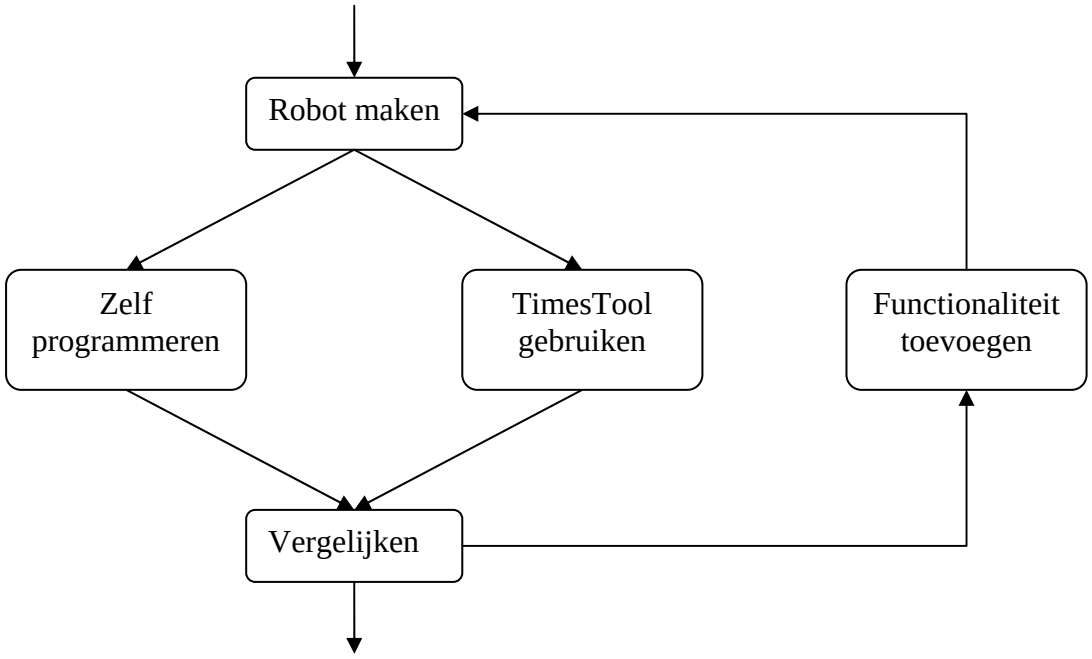
In hoeverre moet je de omgeving modelleren in de tool?

**Deelvraag 5:**

Gebruikt het gegenereerde programma meer/minder geheugen dan een niet gegenereerd programma?

# 2 Aanpak

We gaan de deelvragen beantwoorden aan de hand van een klein voorbeeld voor de RCX. Dit gaan we dan zowel gewoon programmeren als door de TimesTool laten genereren. Dat kunnen we dan vergelijken. Omdat de TimesTool C-code genereert voor het brickOS [5], een besturingssysteem voor de LEGO RCX, gaan we zelf ook C-code programmeren voor de brickOS om een goede vergelijking te kunnen maken. We beginnen met een klein voorbeeld en kunnen later functionaliteit toevoegen. De aanpak is schematisch weergegeven in onderstaand figuur.




Het maken van de robot zullen we doen in paragraaf 2.2 en 2.3. Vervolgens zullen we hier zelf code voor schrijven in hoofdstuk 3. Deze zelfgeschreven code staat in Appendix B. We beschrijven de TimesTool in hoofdstuk 4 en gebruiken deze in hoofdstuk 5. Een voorbeeld van door de TimesTool gegenereerde code staat in Appendix C. We zullen de twee gebruikte methodes vergelijken in hoofdstuk 6 en hier een conclusie en antwoord op onze onderzoeksvraag geven.

## *2.1 Hardware*

De TimesTool kan (momenteel) alleen code genereren voor brickOS [5]. brickOS is een open source embedded besturingssysteem voor de RCX [6]. De RCX is een autonome LEGO microcomputer. Het ziet er uit als een groot geel LEGOblok.

Op de RCX zitten 3 inputpoorten en 3 outputpoorten. Op de inputpoorten kun je bijvoorbeeld tastsensoren en een lichtsensor aansluiten. Op de outputpoorten kun je motoren aansluiten. Met wat extra LEGOblokjes kun je een robot bouwen die bijvoorbeeld kan rijden, een lijn kan volgen en/of obstakels kan ontwijken. Programma's voor brickOS worden in C of C++ geschreven op een pc. De code wordt

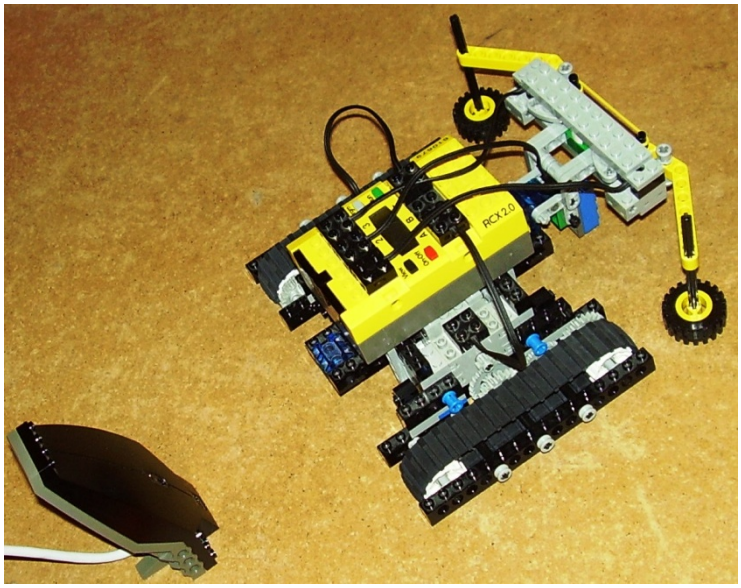
gecompileerd en vervolgens met behulp van een infrarood toren naar de RCX getransporteerd. De RCX kan het programma geheel zelfstandig uitvoeren.

## 2.2 Specificatie

Omdat we eenvoudig willen starten, beginnen we met slechts twee taken te maken. We maken een taak om willekeurig rond te rijden, de zogenaamde “wander”-taak. Omdat de robot bij het rondrijden hier en daar obstakels kan tegenkomen, is het handig als hij daar op reageert. Dus maken we een taak om obstakels te ontwijken, de zogenaamde “avoid”-taak. Hij zal het grootste gedeelte van de tijd alleen met de wandertaak bezig zijn. Als er een obstakel ontweken moet worden moet de wandertaak stop gezet worden. Dit is in het opzicht van scheduling niet erg interessant. Daarom maken we nog een derde taak die muziek afspeelt, zodat de robot telkens met minstens twee taken “tegelijk” bezig is.

## 2.3 De robot

De Robot is relatief eenvoudig. Het ontwerp is gebaseerd op de Roverbot uit [7] met rupsbanden en een dubbele bumper. De bumper hebben we iets uitgebreid zodat hij ook lage objecten kan detecteren. Misschien willen we in de toekomst een lijn kunnen volgen (als extra taak), dus we bouwen ook alvast een lichtsensor in, zodat we later de robot niet hoeven te verbouwen. De lichtsensor komt onder aan de bumper, zodat deze niet in de weg zit voor het detecteren van obstakels. Om de communicatie met de infrarood toren goed te laten verlopen, moet het infrarood signaal niet versperd worden door andere LEGOblokjes. We zetten dus het RCXblok achterstevoren op de robot, zodat de infraroodpoort vrij aan de achterkant zit.



# 3 Zelf code schrijven

In dit hoofdstuk bespreken we hoe we zelf code hebben geschreven voor de robot.
We hebben 3 taken (wander, avoid en music), die we in afzonderlijke threads laten draaien. We hebben voor iedere thread een functie nodig waar het betreffende gedrag in staat. We beginnen met de wandertaak, daarna de avoidtaak en ten slotte de musictaak.

## *3.1 De wandertaak*

Om de robot rond te kunnen laten rijden, moeten we de motoren aansturen. Dit doen we in 5 afzonderlijke functies, voor elke richting (vooruit, achteruit, rechtsaf en linksaf) eentje en een functie voor stoppen. Deze simpele functies zien er als volgt uit

```
void turn_left()
{
     motor_a_dir(fwd);
     motor_c_dir(rev);
     motor_a_speed(MAX_SPEED);
     motor_c_speed(MAX_SPEED);
}
```

Motor_a_dir(richting) geeft aan welke kant de betreffende motor uit moet draaien. Motor_a_speed(snelheid) geeft aan hoe snel de motoren moeten draaien. Zo hebben we 5 functies voor beweging: forward(), back(), turn_left, turn_right() en stop().
Om aan te geven hoelang de robot een bepaalde richting uit moet rijden, maken we 4 extra functies waar een tijd aan meegegeven kan worden. Na deze tijd wordt er gestopt met de beweging.

```
/*****bewegingsfucties voor een bepaalde tijd*****/
void forward_t (int time) //tijd in msec
{
    forward();
    msleep(time);
    stop();
}
```

Zo hebben we 4 bewegingsfuncties waar een tijd aan meegegeven kan worden: voor- en achteruit, links- en rechtsaf.
Het wandergedrag kan nu gemaakt worden, door eerst een tijdje vooruit te rijden en vervolgens willekeurig links of rechtsom te draaien, waarbij de draaihoek ook willekeurig is tussen bepaalde minimum en maximum waarden. Door dit te herhalen rijd de robot willekeurig rond.

## *3.2 De avoidtaak*

Voor het avoid gedrag, maken we gebruik van een wakeup functie (voor meer informatie zie [5.1]). Deze functie geeft de kant waarmee de robot gebotst is terug. Als er nog niet gebotst is, geeft deze functie nul terug.

```
wakeup_t bumper_poller ()
{
     int bump = 0;
     if(RIGHT_BUMPER!=0)
     {
          bump = RIGHT;
     }
     if(LEFT_BUMPER!=0)
     {
          bump = LEFT;
     }
     return bump;
}
```

Door met een wait_event [5.2] te wachten tot er geen nul meer teruggeven wordt, weet de avoidtaak wanneer de robot ergens tegenaan is gebotst en actie moet ondernemen. Deze actie bestaat eerst uit het stoppen van de wandertaak, want als er ergens tegenaan is gebotst hoeft er niet meer willekeurig worden rond gereden, maar moet het obstakel ontweken worden. Vervolgens rijd de robot een eindje achteruit, van het obstakel vandaan. Daarna draait de robot rechtsom als hij aan de linkerkant (dus met de linkerbumper) ergens tegenaan is gereden en linksom als hij aan de rechterkant ergens tegenaan is gereden. Vervolgens kan het wandergedrag weer gestart worden en wordt weer opnieuw gewacht tot de wakeupfunctie geen nul meer teruggeeft. Het starten van de wandertaak gebeurt met execi(), dit is een functie voor het starten van taken.

```
void avoid()
{
     int bump = 0;
     while(!shutdown_requested())
     {
          bump = wait_event(bumper_poller,0);  //wacht tot bumper ingedrukt wordt
          kill(wand); //stop het wandergedrag
          back_t(REVERSE_TIME);
          if(bump==RIGHT)
          {
               turn_left_t(AVOID_TURN_TIME);
          }
          else //bump==LEFT
          {
               turn_right_t(AVOID_TURN_TIME);
          }
          wand    =    execi(&wander,0,NULL,5,DEFAULT_STACK_SIZE);    //start    het
wandergedrag weer
          bump=0;
     }
}
```

## 3.3 De musictaak

In de musictaak maken we gebruik van dsound_play(). Dit is een functie die een liedje bestaand uit een rijtje noten kan afspelen. Omdat het niet uitmaakt welk liedje de robot speelt, laat ik de robot een liedje dat “devil” [8] heet spelen.

```
void music()
{

     dsound_set_duration(40); //om ervoor te zorgen dat het liedje een beetje
sneller gaat
     while (!shutdown_requested())
     {
          if (wait_event(dsound_finished,0) != 0)
            {
               dsound_play(devil);
          }
     }
}
```

## 3.4 De mainfunctie

Nu we alle taken gemaakt hebben, kunnen we de taken aan een thread hangen en de threads starten. We maken eerst voor elke taak een thread waar we naar kunnen verwijzen door middel van een thread-id.

```
/******************de threads******************/
tid_t av;
tid_t wand;
```

```
tid_t sound;
/*********************************************/
```

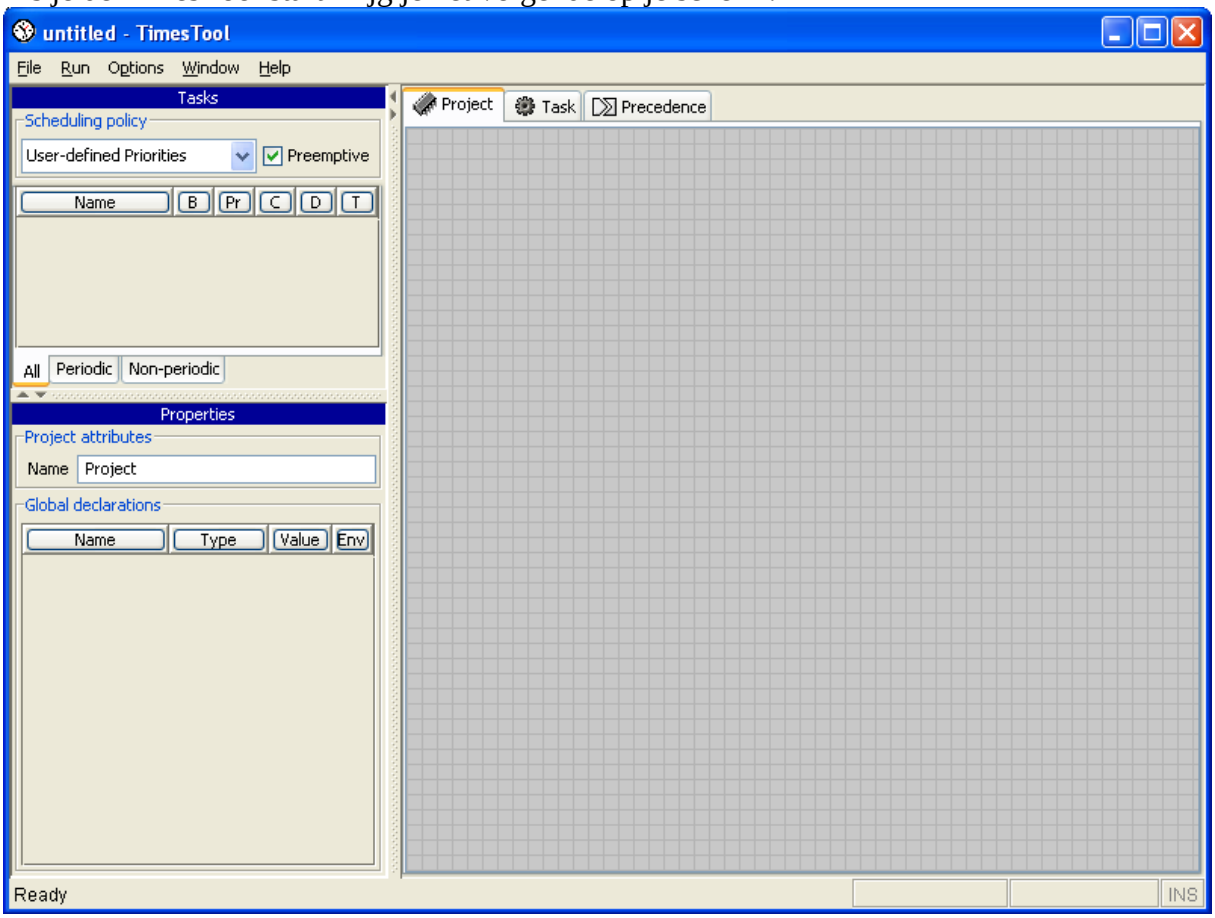
Deze threads starten we in de mainfunctie, hier krijgen ze ook een prioriteit. De avoid- en wanderthread krijgen een prioriteit van 5. We vinden de musictaak minder belangrijk dus de soundthread krijgt een iets lagere prioriteit van 4.

```
void main ()
{
     srandom(300); //seed de random functie
     av = execi(&avoid,0,NULL,5,DEFAULT_STACK_SIZE);
     wand = execi(&wander,0,NULL,5,DEFAULT_STACK_SIZE);
     sound = execi(&music,0,NULL,4,DEFAULT_STACK_SIZE);
}
```

BrickOS zorgt nu zelf voor het schedulen van de taken. Voor meer informatie over het schedulen door brickOS zie [9]. De complete zelfgeschreven code staat in appendix B.
De code doet wat ervan verwacht wordt, de robot rijdt namelijk wat willekeurig rond en speelt een muziekje; als er tegen een obstakel gebotst wordt, wordt er een klein stukje achteruit gereden en van het obstakel af gedraaid.

# 4 De TimesTool

De TimesTool is ontwikkeld door het DARTS-team [3.1] van de Uppsala universiteit [3] in 2001, die ook UPPAAL mee ontwikkeld hebben.
Als je de TimesTool start krijg je het volgende op je scherm:



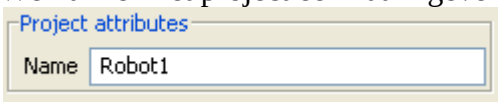

Als je niet weet hoe de TimesTool werkt, kun je de handleiding raadplegen. Er is een online help, die te vinden is onder Help->Online Help, er wordt dan een webpagina geopend met online documentatie [4.1]. Tevens is daar een pdf-versie te downloaden. De handleiding heet Documentation (1.0 beta). Dus de documentatie is van een oudere versie van de TimesTool. We gebruiken momenteel versie 1.3 beta.
De handleiding begint (na het uitleggen van de installatie) met een vijftal tutorials voor de verschillende onderdelen van de tool, namelijk: editing, simulation, schedulability analysis, verification en code sysnthesis. Deze tutorials laten stap voor stap zien hoe je een project maakt aan de hand van een voorbeeldproject “Simple”. Er wordt echter veel niet uitgelegd zoals waar de verschillende stappen voor dienen, wat precies een taak, is of wat het doel van constanten of parameters. Na de tutorials weet je nog niet welke onderdelen er nu in je eigen project moeten. Gelukkig word er na de tutorials ook nog uitleg gegeven over de verschillende onderdelen van de tool. Verder zijn er een aantal wetenschappelijke artikelen over de TimesTool geschreven zoals: “TIMES: a Tool for Schedulability Analysis and Code Generation of Real-Time Systems” [10] en “Times - A Tool for Modeling and Implementation of Embedded Systems” [17]. In “Code Synthesis for Timed Automata” [12] wordt het model van een ProductionCell beschreven, deze kunnen we als voorbeeld gebruiken.

## 4.1 Editing

Voordat we een systeem kunnen analyseren of er code voor kunnen genereren moeten we het eerst modelleren. Dit doen we in het editing onderdeel. Informatie over het maken en bewerken van modellen is te vinden in de “Editing tutorial” en in het “Editer” hoofdstuk van de help.

### 4.1.1 Een project maken

We beginnen met een nieuw project door de TimesTool te starten of op File->New te klikken. We kunnen het project een naam geven.



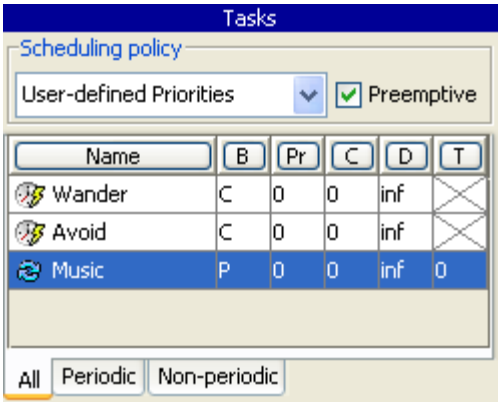

We kunnen nu het project opslaan, de bestandsnaam kan hetzelfde zijn als de projectnaam. Het project wordt opgeslagen als xml-bestand. Als je een project opslaat met dezelfde bestandsnaam als een eerder project, wordt er niet eerst gevraagd of je zeker weet dat je het eerdere project wilt overschrijven. Je bent dan het eerdere project gewoon kwijt.

### 4.1.2 Taken

Het volgende onderdeel is taken toevoegen en configureren. Maar wat zijn taken eigenlijk? Uit het Editor hoofdstuk lezen we:

> “A *task* is a piece of code, as in general real-time scheduling theory. It has properties such as worst case execution time, deadline, and possibly others.”

In ons geval lijkt het er dus op dat we 3 taken hebben: Wander, Avoid en Music. We hebben inderdaad stukken code voor deze 3 taken. Deze taken hebben ook een worst case execution time en deadline, al zijn die nu nog niet bekend, maar kunnen dit waarschijnlijk wel berekenen. We kunnen deze taken toevoegen door in het Tasks-veld rechts te klikken en “Add periodic task” of “Add controlled task” te kiezen. Periodic of controlled is de “behavior” (B) van een taak, deze kunnen we later ook nog veranderen. Verder heeft elke taak een naam (Name), een prioriteit (Pr), een execution time (C), een deadline (D) en een periode (T). Deze eigenschappen hebben allemaal met scheduling te maken en worden in de volgende paragrafen besproken.




#### *4.1.2.1 Behavior*

We lezen in het editor hoofdstuk van de help:

> “The *task behavior* defines how a task is released: periodically with the fixed period, by entering a control automaton state or periodically with the given minimal inter-arrival time (sporadically).”

C staat voor controlled, P voor periodic en S voor sporadic.

#### *4.1.2.2 Prioriteiten*

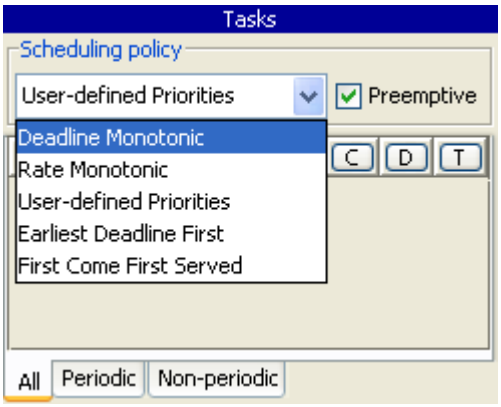
We kunnen kiezen uit vijf verschillende scheduling policies die de prioriteit van de verschillende taken bepalen:

**Deadline Monotonic**

Bij deadline monotonic scheduling krijgen de taken met de vroegste deadline de hoogste prioriteit. De prioriteiten worden van te voren bepaald.

**Rate Monotonic**

Bij Rate Monotonic scheduling krijgen de taken met de kortste periode de hoogste prioriteit.




**User-defined Priorities**

Hierbij kan de gebruiker zelf de prioriteit van de taken aangeven.

**Earliest Deadline First**

Bij Earliest Deadline First schedulingen krijgen de taken met de op dat moment vroegste deadline de hoogste prioriteit, de prioriteiten van de taken veranderen dus tijdens run-time.

**First Come First Served**

First Come First Served (FCFS) staat ook bekend als First in First out (FIFO). Hierbij worden taken die het eerst aangeroepen worden ook het eerst afgehandeld, pas daarna komt de volgende taak.

**Preemptive**

Naast de dropdown box voor de verschillende scheduling policies staat een selectievakje voor Preemtive. Bij preemtive kan een taak onderbroken worden om een taak met een hogere prioriteit aan de beurt te laten. Als we dit dus uit zetten kunnen taken niet zomaar onderbroken worden. Als we dit wel toestaan kan het zijn dat het geheel beter gescheduled kan worden, maar we moeten dus ook rekening houden met het feit dat taken onderbroken kunnen worden door andere taken.

### *4.1.2.3 Execution Time*

De execution time geeft aan hoelang taken erover doen om uitgevoerd te worden. De scheduler weet zelf niet hoelang een taak duurt, dus moet je dit zelf opgeven. Voor de code generatie wordt dit niet gebruikt.

### *4.1.2.4 Deadline*

De deadline is om aan te geven hoelang de tijd de scheduler mag nemen om de taak uit te voeren. De deadline moet dus minstens even lang zijn als de execution time, anders wordt de deadline nooit gehaald. In ons geval zal voor de wandertaak de deadline niet echt uitmaken. Er wordt immers willekeurig rondgereden, dus als de robot iets langer een richting uitrijdt dan de willekeurige tijd die berekend was, zal dit niet uitmaken. De avoid moet echter wel voor een bepaalde tijd klaar zijn, anders is de robot al ergens tegenaan gebotst zonder dat de avoid taak het door heeft. De music taak moet ook voor een bepaalde tijd een noot hebben gespeeld, anders klinkt het liedje niet goed.

### *4.1.2.5 Periode*

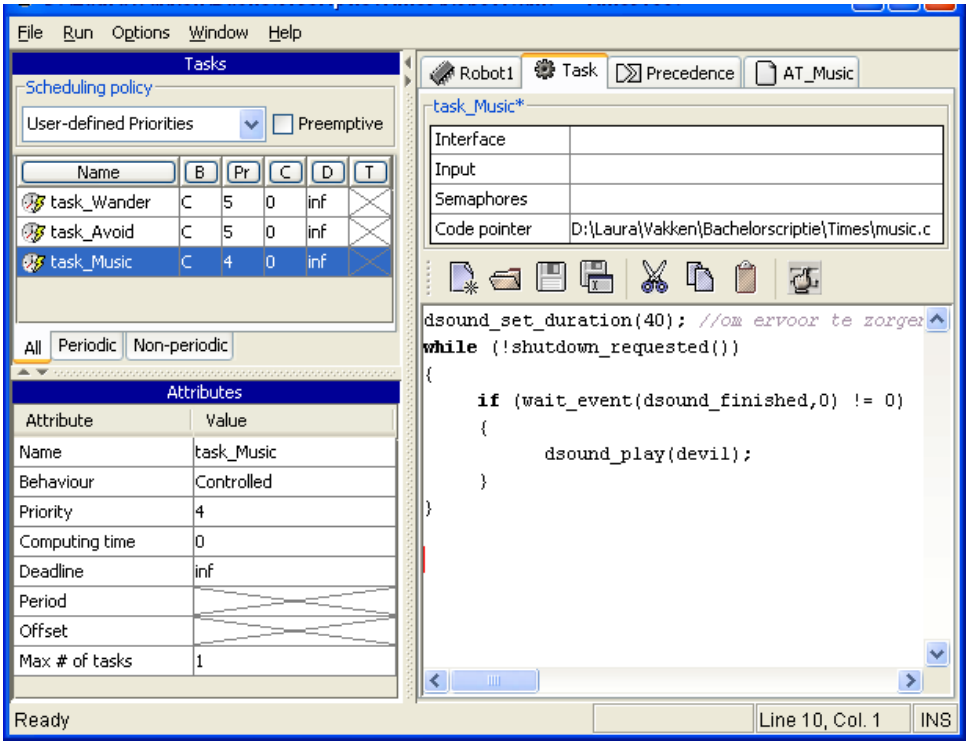
Alleen periodic taken hebben een periode. Na deze tijd moet de taak weer opnieuw uitgevoerd worden.

### *4.1.2.6 Overige eigenschappen*

In de Tasks-tabblad kunnen we ook nog het aantal instanties van een taak die gelijktijdig kunnen draaien veranderen. In ons geval willen we telkens maar één instantie van één taak tegelijk, we laten we deze waarde dus op 1 staan.

### *4.1.2.7 Task code*

We hebben nu wel de taken aangemaakt, maar het programma weet nog niet wat de taken inhouden. Dat moeten we dus nog invullen. Dit kan in het Task-tabblad. We selecteren nu bijvoorbeeld de task_Music en vullen in grote witte vak de code in die we al eerder hadden voor de musictaak. Vervolgens moeten we dit nog opslaan in een aparte file, dit doen we in music.c. We hadden ook eerst een file kunnen maken met de code erin en die dan hier openen.

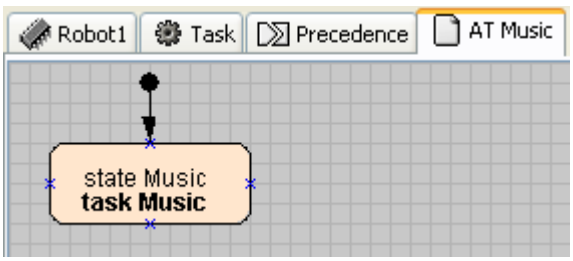



Als je bij een taak dubbelklikt op de Code pointer, dan kun je deze veranderen. Je kunt de Code pointer een relatief adres geven door er .\music.c, .\avoid etc. van te maken, zodat je de map eenvoudig kunt verplaatsen/kopiëren zonder telkens de code pointer te hoeven veranderen.

## 4.1.3 Automaton templates

Het volgende wat we moeten maken is één of meerdere automaten, die aangeven wanneer controlled taken uitgevoerd moeten worden:

> "An automaton template is an uninstantiated timed automaton with tasks and possibly some parameters. States can be associated with tasks and have invariants whereas transitions have guards, synchronization and assignments as arguments."




Elke controlled taak moet ergens in een automaat voorkomen om uitgevoerd te worden. Als je een periodic taak in een automaat zet, wordt deze vanzelf controlled.

In Code Synthesis for Timed Automata [12] staat op blz. 11 dat in een automaat eerst zoveel mogelijk transities worden genomen; als dat niet meer mogelijk is wordt de taak uitgevoerd van de toestand waar hij op dat moment is. Een taak wordt dus niet altijd uitgevoerd als de automaat in de betreffende toestand komt. Hij wordt alleen uitgevoerd als er op dat moment geen transities meer mogelijk zijn.

Als er vanuit een toestand op een moment gekozen kan worden uit twee transities, dan wordt er in de simulator willekeurig gekozen tussen deze twee transities. De volgende keer als de automaat weer in dezelfde toestand is, wordt er weer willekeurig gekozen. De ene keer zal er dus de ene transitie en een andere keer een andere transitie gekozen worden. Als er echter code gegenereerd wordt en uitgevoerd, merken we dat altijd dezelfde transitie gekozen wordt. In [10] staat bij code generatie:

> Deterministic semantics A model can exhibit two types of non-determinism: time non-determinism, i.e. that enabled transition can be taken at any time point within the time-zone, and external non-determinism i.e. that several actions may be simultaneously present from the environment. To overcome the problems introduced by this we adopt a deterministic semantics that define a subset of the behaviour. External non-determinism is resolved by defining priorities for action transitions in the controller. If several transitions are enabled in a state the one with the highest priority is taken. Time non-determinism is resolved by adopting the so-called maximal-progress assumption [11]. Maximal-progress means that the controller should take all enabled transitions until the system stabilises, i.e. no more action transitions are enabled.

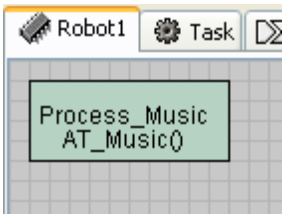
Dus een non-deterministische keuze tussen twee transities wordt deterministisch gemaakt door prioriteiten aan de transities toe te voegen. Hierdoor wordt altijd dezelfde transitie gekozen. Hier komen de simulatie en het uiteindelijke gedrag na code generatie dus niet overeen.

Twee verschillende automaten kunnen met elkaar synchroniseren door middel van een channel. Bij een urgent channel mag er geen tijd verstrijken voordat deze transitie genomen wordt.

### 4.1.4 Processen

Het laatste onderdeel van het modeleren van het systeem is het creëren van processen.

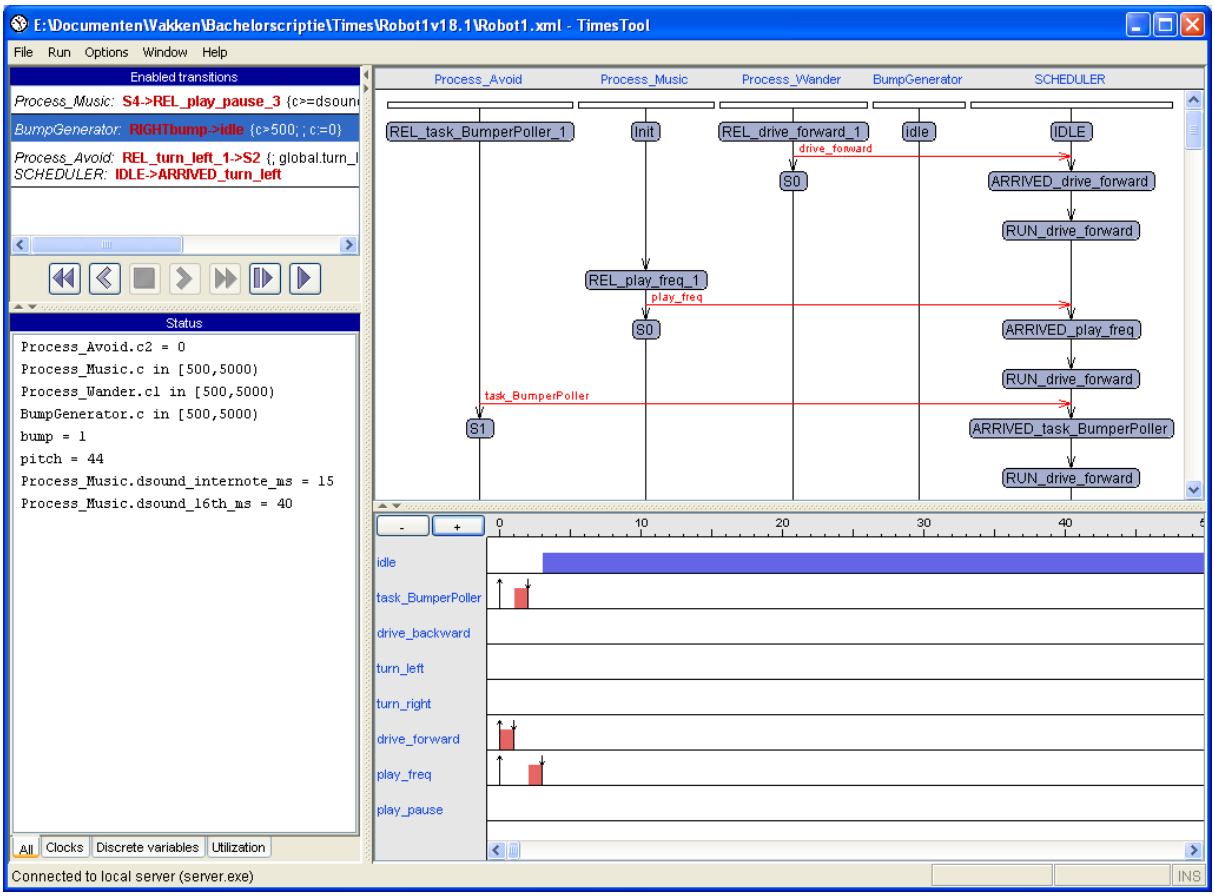
> "A process is a building block of the system, an automaton template instantiated with the given arguments. When the process instance enters a state with a task assotiated, an instance of that task is released for execution."




We kunnen dus voor iedere automaat een proces maken en eventueel argumenten meegeven.

## *4.2 Analyse*

Je kunt de werking van het gemodelleerde systeem op drie manieren analyseren. Als eerste door een simulatie uit te voeren. Je kunt dan zien welke taken op welk moment gescheduled kunnen worden en wat de waarden van variabelen op bepaalde tijdstippen zijn. Hieronder staat een screenshot van de simulator.




Ten tweede kun je TimesTool een schedulability analyse uit laten voeren. TimesTool berekent dan of je systeem schedulebaar is met de opgegeven schedule policy en waarden.
Als laatste kun je ook nog bepaalde eigenschappen van het systeem verifiëren. De verifier is gebaseerd op de verifier van UPPAAL [3.2].

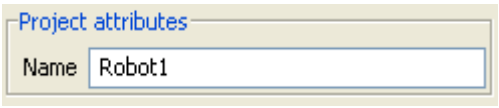
## 4.3 Code Generatie

Als we een model gemaakt hebben, kunnen we hier code voor genereren. Dit kan simpel door op Run->Code synthesis te klikken. Er worden dan een aantal bestanden gegenereerd. De bestanden die beginnen met ‘brickos’ zijn standaardbestanden en bij elke codegeneratie hetzelfde. De bestanden die beginnen met de projectnaam, zijn de bestanden waar het model in staat. In de meeste staat echter niet zoveel interessants, behalve in <projectnaam>.c. Hierin worden het hele model beschreven. De bestanden met de taakcode worden hier ingevoegd met behulp van “include”.
Deze code zou nu gecompileerd en uitgevoerd kunnen worden. Dat dit echter niet altijd meteen goed werkt, zien we in het volgende hoofdstuk, waarin we zelf code gaan genereren.

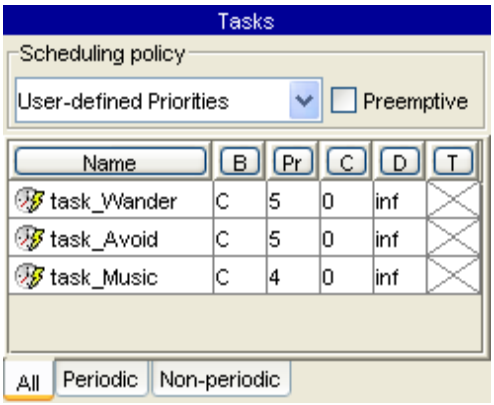
# 5 Code Generatie

In dit hoofdstuk gaan we proberen het systeem dat we in hoofdstuk 3 hebben gemaakt met TimesTool te modelleren en hiermee code te genereren. We bespreken een aantal versies die we achtereenvolgens hebben gemaakt en de problemen die we daarbij tegen kwamen. Hoe je de diverse onderdelen die nodig zijn voor het genereren en uivoeren van de code installeert en configureert staat in appendix A

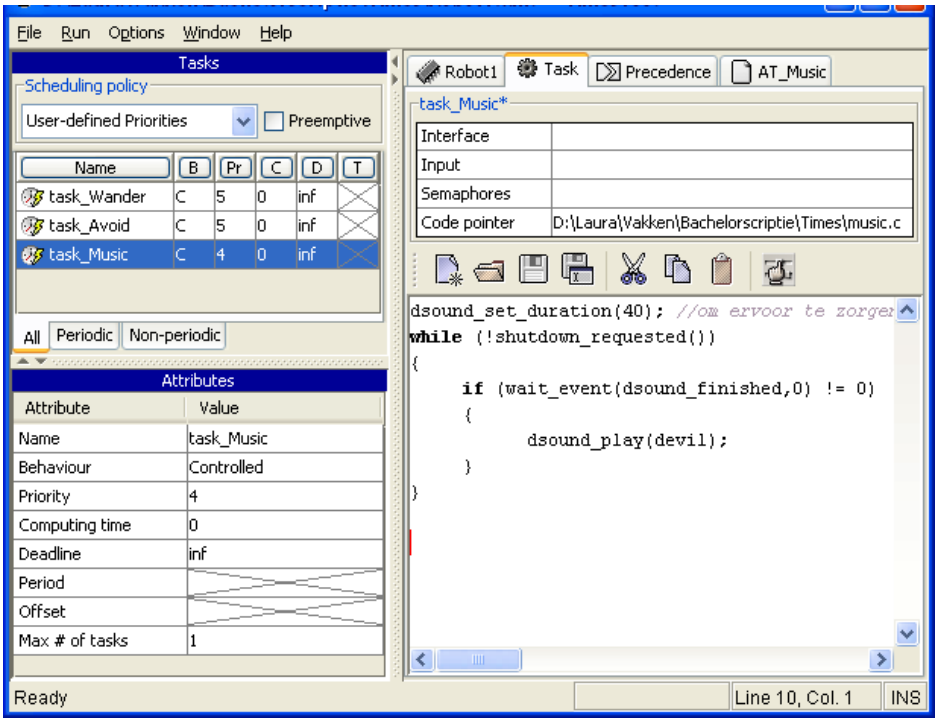
## *5.1 De eerste versie met de TimesTool*

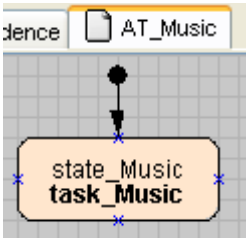



We beginnen met een nieuw TimesTool project, genaamd: “Robot1
We hebben in 2.2 gezien dat we willen dat de robot drie taken uitvoert: Wander, Avoid en Music. Het lijkt logisch om dit ook taken in TimesTool te maken. Omdat we voor elk van deze taken een automaat willen maken, worden ze alle drie controlled. We voegen dus drie periodic taken toe aan het lijstje van taken en noemen ze task_Wander, task_Avoid en task_Music.

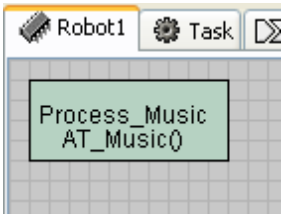



Aangezien we in hoofdstuk 3 al prioriteiten hebben bepaald, zullen we de optie User-defined priorities gebruiken. Op deze manier kunnen we het beste vergelijken tussen de zelfgeschreven code en de gegenereerde code Om het ons zelf vooralsnog niet al te moeilijk te maken zetten we de preemtive optie maar uit, zolang dat geen moeilijkheden voor de scheduler oplevert.
We weten op dit moment nog niet hoelang de taken erover doen om uitgevoerd te worden. We laten de execution time, periode en deadline dus nog maar even op de standaard waarden staan en zien wel of we er later nog achter komen.
We gaan nu eerst alleen verder met de musictaak, om te kijken of alles werkt. We selecteren nu de task_Music en vullen in grote witte vak de code in die we al eerder hadden voor de musictaak. Vervolgens moeten we dit nog opslaan in een aparte file, dit doen we in music.c.




We maken nu een automaat voor de music taak. We maken één toestand die we state_Music noemen en koppelen die aan de task_Music.
Vervolgens maken we een process block voor de zojuist gemaakte Automaton Template music. De musicautomaat heeft geen argumenten nodig.




Nu we de musictaak gemodelleerd hebben, kunnen we code gaan genereren en testen. Hierbij lopen we al meteen tegen een aantal problemen aan. Het code genereren zelf gaat nog goed en we krijgen netjes een melding welke bestanden er gegenereerd zijn. In robot1.c staat het model, omgezet in code, de overige documenten lijken vrij standaard te zijn. Als we de compileerfunctie van de TimesTool proberen, krijgen we geen reactie. We gebruiken dus vanaf nu maar BricxCC om te compileren en te uploaden. Deze geeft netjes aan of het compileren gelukt is of waarom niet.
Het volgende probleem was een aantal foutmeldingen van de compiler: “stray ‘\’ in program”. Dit bleek te liggen aan het feit dat een aantal van de door de TimesTool gegenereerde files in DOS-file formaat is opgeslagen. Door de files alsnog handmatig in UNIX-file formaat op te slaan, is dit probleem op te lossen.
Vervolgens is de “transition table” en de “location list” niet gedefinieerd. Dit blijkt te liggen aan het feit dat ze gedefinieerd worden met hoofdletters (in robot1.c) en op de plek waar ze gebruikt worden met kleine letters worden geschreven. Als we dus de definitie in robot1.c handmatig met kleine letters schrijven, lost dit het probleem op. We hebben echter überhaupt geen transition table in robot1.c staan. Dit lijkt te liggen aan het feit dat we nog geen transities hebben gedefinieerd. Het is dan wel erg slordig van de TimesTool om zelfs geen lege tabel te genereren, als deze wel nodig is voor het goed functioneren van het programma. We maken deze dus zelf maar (met kleine letters dus).
Nadat al deze problemen goed waren opgelost, kon het programma eindelijk uitgevoerd worden. De robot deed nu wat er verwacht was, namelijk het liedje spelen.

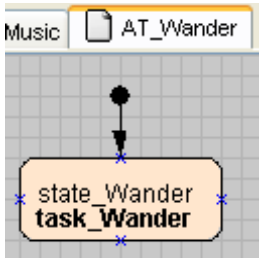
We gaan nu meer taken toevoegen. We beginnen met de wandertaak. We zetten de code die we zelf al gemaakt hadden in hoofdstuk 3 in het code-gedeelte van de task_Wander in de TimesTool:

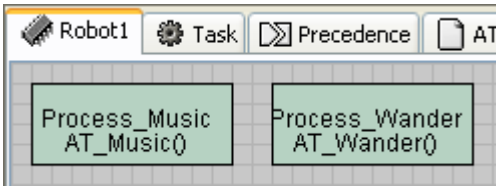
```
int rand (int min, int max)
{
     long int rand_nr = random();
     rand_nr = (rand_nr%(max-min))+min;
     return rand_nr;
}

int i;
int forward_time;
while (!shutdown_requested())
{
     forward_time = rand(MIN_MOVE_TIME, MAX_MOVE_TIME);
     lcd_int(forward_time);
     forward_t(forward_time);

     i = rand(0,2);  //random links of rechts draaien
     if (i==0)
     {
          lcd_int(i);
          turn_left_t(rand(MIN_TURN_TIME,
MAX_TURN_TIME));
     }
     else
     {
          lcd_int(i);
          turn_right_t(rand(MIN_TURN_TIME,
MAX_TURN_TIME));
     }
}
```

Ook moeten de constanten en de bewegingsfuncties (zie hoofdstuk 3) aan de code van de task_Wander toegevoegd worden.
Vervolgens maken we een automaton template voor de wandertaak. Hierin zetten we een toestand waarin de task_Wander uitgevoerd wordt.




We maken nu nog een Process naast het Process_Music. Dit process is voor de wanderautomaat.

Het model is nu uitgebreid met wanderfunctionaliteit. Als we hiermee nu code genereren en uitvoeren, dan rijdt de robot ook (en maakt nog steeds muziek), maar hij rijdt alleen maar in een rondje. De random functie werkt dus niet goed. Dit is vreemd, want er wordt gebruik gemaakt van een standaard randomfunctie uit brickOS. Voor de taken is het op dit moment niet zo belangrijk of de robot willekeurig of in rondjes rondrijdt, dus laten we het maar zo.

Tot zover ging het nog allemaal redelijk goed. Als we echter de avoidtaak willen toevoegen merken we dat deze manier van taken maken toch niet zo goed werkt.
De volgende code kopiëren we uit de zelfgeschreven code naar de avoidtaak in de TimesTool:

```
wakeup_t bumper_poller ()
{
     int bump = 0;
     if(RIGHT_BUMPER!=0)
     {
          bump = RIGHT;
     }
     if(LEFT_BUMPER!=0)
     {
          bump = LEFT;
     }
     return bump;
```

```
}

int bump = 0;
while(!shutdown_requested())
{
     bump = wait_event(bumper_poller,0);  //wacht tot bumper ingedrukt wordt
//     kill(wand); //stop het wandergedrag
     back_t(REVERSE_TIME);
     if(bump==RIGHT)
     {
          cputs("b1");
          turn_left_t(AVOID_TURN_TIME);
     }
     else //bump==LEFT
     {
          cputs("b3");
          turn_right_t(AVOID_TURN_TIME);
     }
//     wand = execi(&wander,0,NULL,5,DEFAULT_STACK_SIZE); //start het wander gedrag
weer
     bump=0;
}
```

We moeten nu ook alle standaard bewegingsfuncties en de constanten weer bij de avoidcode zetten, anders werkt het niet. Dit geeft al aan dat de indeling in taken toch anders moet omdat je anders geen functies kunt maken die niet specifiek bij een taak horen, zonder deze in elke taak te definiëren.
Een probleem is nu het stoppen en opnieuw opstarten van het wandergedrag. Je kunt niet zomaar in de code zeggen dat je een ander process wilt afkappen. Daar heb je een tid_t voor nodig en die hebben we nu niet. Zonder het afkappen van het wanderproces vertoont de robot wel iets van avoidgedrag, maar er wordt al snel weer willekeurig rondgereden. De oorzaak ligt bij de sleep in de bewegingsfuncties: In de avoidtaak staat back_t(REVERSE_TIME), die er als volgt uit ziet:

```
void back_t (int time) //tijd in msec
{
    back();
    msleep(time);
    stop();
}
```

Tijdens de msleep(time) wordt er van taak gewisseld en gaat de robot vrolijk verder met wanderen. We kunnen hieruit opmaken dat de indeling van de taken zo niet goed is en we de taken moeten opsplitsen in meerdere kleine taken.

Als we met de simulator dit model analyseren, is deze erg saai. Alle taken worden opgestart en vervolgens komt het systeem in een deadlock. Er zijn immers geen transities meer die genomen kunnen worden. De simulator kijkt niet naar de code die in de taken uitgevoerd worden. Dat de uitgevoerde taken ondertussen nog wel wat doen, weet de simulator dus niet.

**Samenvatting**

We kwamen bij het maken van een simpel model van ons systeem de volgende problemen tegen:

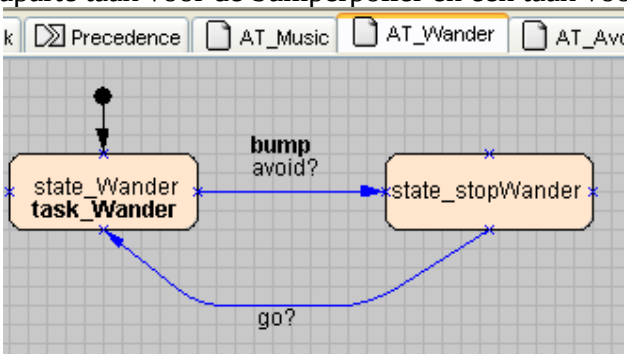
- De gegenereerde code is niet zomaar te compileren
- We moeten sommige functies aan meerder taken toevoegen, dit komt omdat er geen algemeen code gedeelte is wat door alle taken kan worden gebruikt. Later kwamen we er achter dat dit opgelost kon worden door de taken verder op te splitsen en deze functies taken te maken.
- De randomfunctie werkt niet.

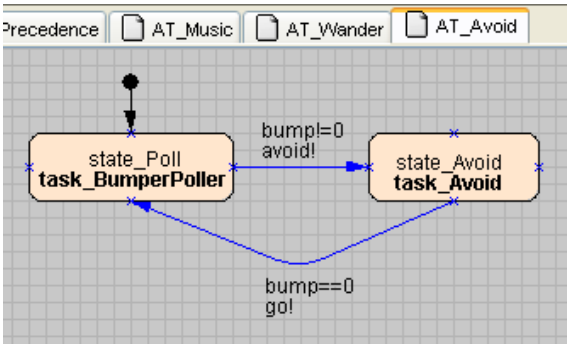
- Je kunt niet in het ene proces een ander proces killen.
- Simulatie representeert niet de echte werking van het systeem.

We kunnen nu dus concluderen dat we de taken moeten opsplitsen in kleinere taken, en meer van de functionaliteit in de automaten moeten stoppen, zodat het systeem ook beter geanalyseerd kan worden.

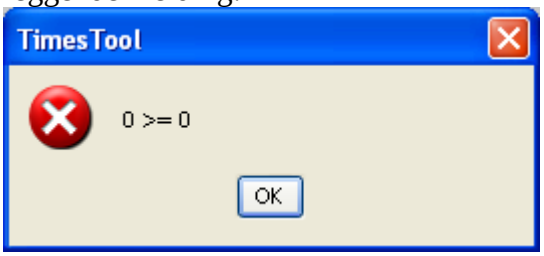
## 5.2 Opsplitsing van taken

Omdat we niet in het code gedeelte van een taak, een andere taak kunnen stoppen of starten, moeten we het wisselen in de desbetreffende automaten modelleren. Hiervoor maken we een aparte taak voor de bumperpoller en een taak voor de reactie op het botsen.

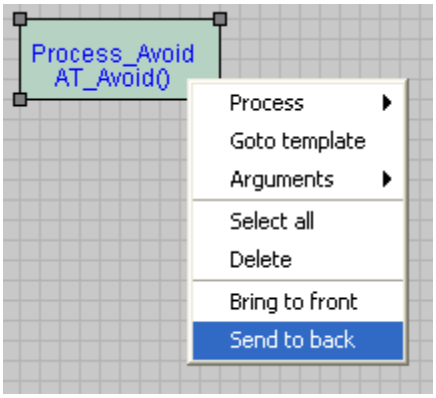



De automaat AT_Wander (het linkerplaatje) begint in de toestand state_Wander met de taak task_Wander (het “willekeurig” rondrijden). De automaat AT_Avoid (het rechterplaatje) begint in de toestand state_Poll met de taak task_Bumperpoller. Als de task_BumperPoller een botsing detecteert, zet hij de variabele ‘bump’ op 1 of 3 (afhankelijk of er met de linker of de rechterbumper gebotst is). De AT_Avoid kan nu naar de state_Avoid. Als hij dit doet, verzend hij over een urgent channel ‘avoid’. Daarop gaat de AT_Wander naar de toestand state_stopWander, waar niets gebeurd, maar de wandertaak wordt nu niet meer uitgevoerd. Aan het einde van de task_Avoid wordt bump weer op 0 gezet, en kunnen beide automaten weer terug naar hun begintoestand. We hebben dus wat functionaliteit uit de code gehaald en toegevoegd aan de automaten. Als we nu code genereren, krijgen we de volgende weinig zeggende melding:




Na lang zoeken komen we er achter dat dit opgelost kan worden door “Sent to back” te doen op het Process_Avoid. Dan wordt er wel code gegenereerd zonder foutmelding. Send to back is bedoeld om procesblokken die grafisch voor een ander procesblok staan, naar achteren te verplaatsen. Waarom dit echter van invloed is op de code generatie, is niet duidelijk en het lijkt op een bug.




Het volgende wat we ontdekken is dat code die in de taak staat, niet bij de locale variabelen kan die in de automaat staan. Dus we hebben in de AT_Avoid een variabele “bump” gedeclareerd, om te kunnen kijken of de robot al ergens tegenaan is gebotst, maar

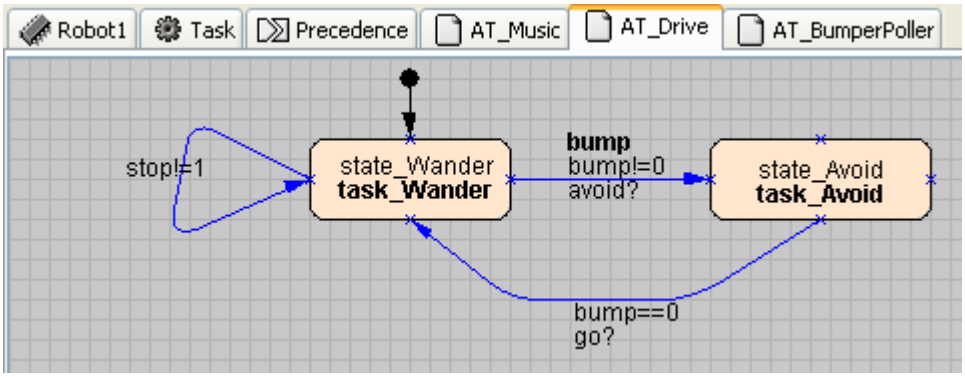
kunnen die niet gebruiken in de task_Bumperpoller code. In een voorbeeld bestand (die van de ProductionCell [12]) zien we wel een zelfde variabele in de code gebruikt worden als in het model. Dit is echter wel een globale variabele (dus niet van een specifieke automaat). We maken dus de 'int bump' ook globaal. Dit werkt beter, maar het lijkt minder logisch om de variabelen globaal te hebben.
Later vinden we in de handleiding [4.1]:

> There is also *task interface* field, where an expression should be entered showing how the task code influence internal model variables by the end of it's execution.

Dit lijkt echter alleen van iets uit te maken voor de analyse, maar niet voor de gegenereerde code.

Dit model is net als de vorige versie (paragraaf 5.1) ook niet geschikt voor analyse van het systeem. Als je de simulator draait, voert hij alle taken één keer uit en komt dan in een deadlock (wat ook logisch is als je het zelf analyseert).

Als we nu gegenereerde code proberen te compileren, komen we er achter dat de lijst met transities die een channel gebruiken (die we nu voor het eerst nodig hadden), in de gegenereerde code CHANUSAGE heet, terwijl hij wordt aangeroepen met chanusage. Dit was hetzelfde probleem als met trans en loc (zie paragraaf 5.1), en kan dus ook op dezelfde wijze worden opgelost.
Als de robot de code uitvoert, lijkt hij nu iets van avoid gedrag te vertonen. Wander en avoid lopen echter nog een beetje door elkaar. De oorzaak hiervan is dat op het moment dat de automaat naar de state_Avoid gaat, de task_Wander gewoon door gaat. We moeten de while(!shutdown_requested()) dus uit de wandertaak halen (en ook uit de andere taken), om ervoor te zorgen dat de wandertaak niet eeuwig (of totdat de robot uit wordt gezet) door blijft lopen. Als we dat aanpassen wordt de wandercode eenmaal uitgevoerd en is dan afgelopen. We willen echter dat de wandertaak als hij afgelopen is weer opnieuw begint. Hiervoor maken we een transitie van de wandertaak naar zichzelf.




We moeten echter nog wel kunnen stoppen als shutdown_requested. We maken dus een nieuwe taak die op shutdown_requested controleerd en een globale variabele "stop" op 1 zet als er gestopt moet worden. We maken dit een periodieke taak, zodat we er geen triviaal automaatje bij hoeven te tekenen. We kunnen nu in AT_Drive controleren op de globale variabel 'stop'.
Deze methode werkt echter niet omdat in een automaat eerst zoveel mogelijk transities worden genomen, als dat niet meer mogelijk is wordt de taak uitgevoerd van de toestand waar hij op dat moment is. Dit betekent dus dat de AT_Drive de hele tijd de transitie naar zichzelf blijft nemen (totdat je hem uitzet). De task_Wander wordt op deze manier dus nooit uitgevoerd.

**Samenvatting**

We kwamen bij deze iteratie de volgende problemen tegen:

- ‘Send to back’ is nodig om de foutmelding ‘0>=0’ op te lossen.
- Voor communicatie over en weer tussen de taakcode en de automaat zijn globale variabelen nodig in plaats van locale variabelen.
- De taken zijn wel wat uitgesplitst, maar het model is nog steeds niet geschikt voor realistische analyse
- Het is niet mogelijk een taak weer van voor af aan te beginnen zolang de automaat nog in dezelfde toestand is.

De taken moeten nu dus nog verder uitgewerkt. Alles wat met timing te maken heeft moet in het model en niet in de taakcode.

## 5.3 Verdere opsplitsing van taken

We willen de taken die we nu hebben opsplitsen in kleine taakjes. We beginnen met de wandertaak. En zullen daarna ook de avoidtaak verder opsplitsen. De musictaak splitsen we pas in paragraaf 5.5 op

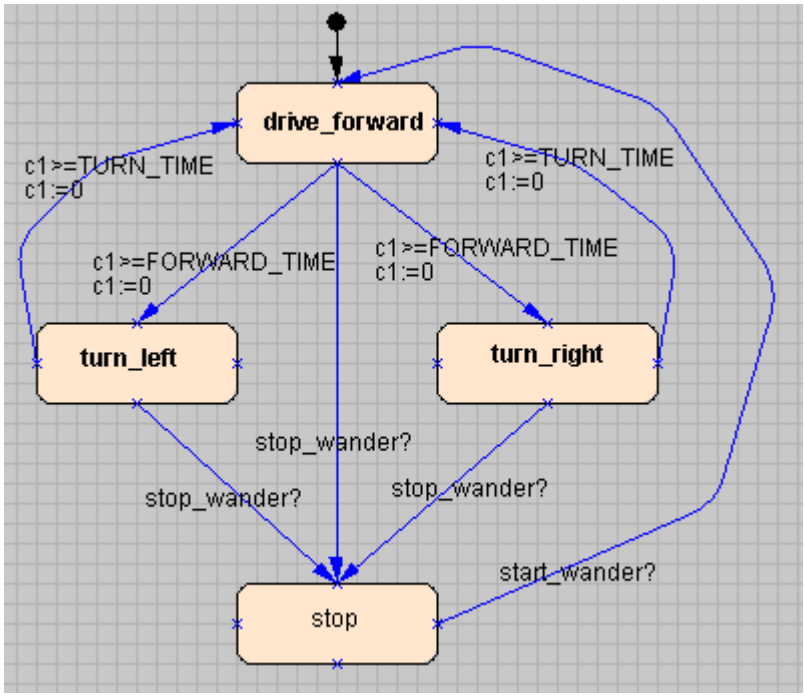
### 5.3.1 De wanderautomaat

We maken nu een taak om rechtdoor te rijden: drive_forward

```
motor_a_dir(fwd);
motor_c_dir(fwd);
motor_a_speed(MAX_SPEED);
motor_c_speed(MAX_SPEED);
```

| | | | | | |
|---|---|---|---|---|---|
| drive_forward | C | 4 | 0 | inf | |
| turn_left | C | 4 | 0 | inf | |
| turn_right | C | 4 | 0 | inf | |
| drive_backward | C | 4 | 0 | inf | |

Op dezelfde wijze maken we een taak voor rechtsaf, turn_right, en een taak voor linksaf, turn_left.




De wanderautomaat die we vervolgens modelleren is hiernaast te zien.
Hierbij is c1 een clock-variabele. De avoid-taak/automaat moet het wander gedrag kunnen stoppen en starten. Hiervoor is de stop-locatie waarin de wanderautomaat niets doet. Naar deze stop-locatie gaat een urgent channel “stop wander” vanuit elke andere locatie van de automaat heen. Er gaat vanuit daar ook weer een transitie met een channel “start_wander” terug naar het begin van de automaat zodat het wander gedrag ook weer gestart kan worden. Het idee is dat er vanaf de drive_forward-locatie willekeurig gekozen kan worden tussen links of rechtsom draaien. Als we de simulatie draaien wordt er inderdaad soms voor turn_left en soms voor turn_right gekozen. Als de code wordt uitgevoerd, wordt er

echter altijd eenzelfde kant uit gedraaid. Dit klopt dus niet met de simulatie, zoals opgemerkt in hoofdstuk 4.

Om erachter te komen wat goede waarden voor FORWARD_TIME en TURN_TIME zijn, moeten we weten met welke tijdseenheid de clocks werken. Dit is niet in de handleiding te vinden, dus moeten we het zelf maar achterhalen. De clocks werken met get_system_up_time(), die volgens [5.3] iets van het type time_t teruggeeft. Hier staat ook niet welke eenheid dit heeft. Er staat echter wel als bug bij:

> time_t is a 32 bit value which will overflow after 49.7 days of continuous operation.

We kunnen nu dus berekenen wat de eenheid is. Met 32 bits kun je 4294967296 verschillende waarden maken. Na 49,7 dagen is de maximumwaarde bereikt. 49.7 dagen is 4294080000 milliseconden. Dit komt erg dicht in de buurt van de maximum waarde van time_t. Dus de eenheid van time_t ligt dicht bij of is milliseconden. De enige andere mogelijkheid zou nog clockticks kunnen zijn. In time.h [5.3] zien we dat een tick precies 1 milliseconde duurt:

```
#define TICK_IN_MS    1
#define TICKS_PER_SEC    1000
```

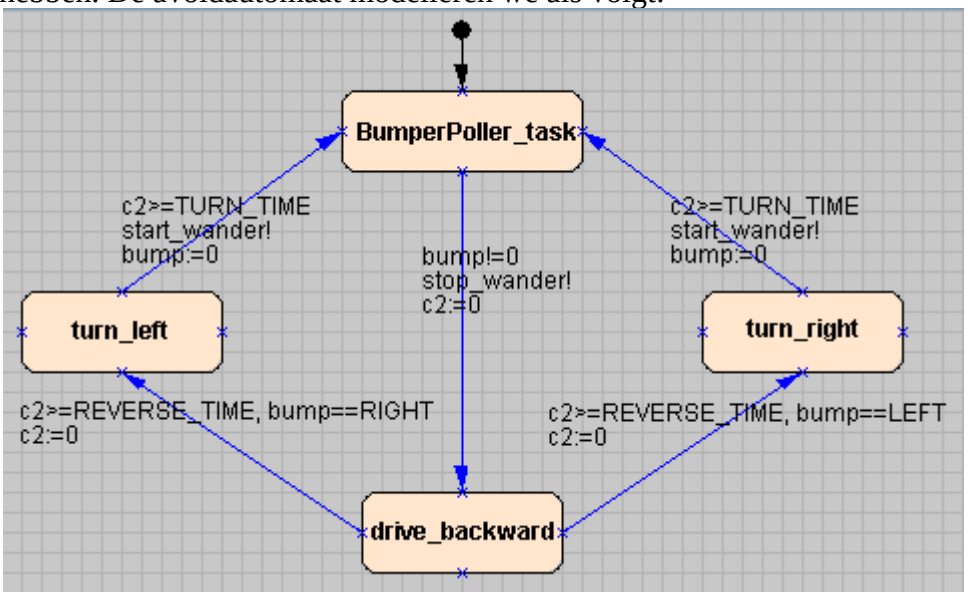
De eenheid van een clock in de TimesTool is dus milliseconden of ticks (die beide even lang zijn) Als we nu de robot 1 seconde in een richting willen laten rijden, moeten we dus de FORWARD_TIME gelijkmaken aan 1000.

### 5.3.2 De avoidautomaat

Voor de het avoidgedrag zijn we ook nog een taak voor achteruit nodig: drive_backward

```
motor_a_dir(rev);
motor_c_dir(rev);
motor_a_speed(MAX_SPEED);
motor_c_speed(MAX_SPEED);
```

Een voordeel van het opsplitsen is, dat we de code voor het rijden maar één keer hoeven te definiëren, en dan zowel in de wanderautomaat als in de avoidautomaat kunnen gebruiken. In paragraaf 5.1 zagen we dat we de code twee keer moesten definiëren. We kunnen nu in de avoidautomaat weer de “turn”-taken gebruiken die we in de wanderautomaat ook gebruikt hebben. De avoidautomaat modelleren we als volgt:




Als de BumperPoller een botsing detecteert, gaat de automaat naar de drive_backward toestand en stopt het wandergedrag door communicatie via urgent channel “stop_wander”. Als er minstens REVERSE_TIME tijd achteruit is gereden, wordt naar de toestand turn_left

of turn_right gegaan. Als de robot links ergens tegenaan is gebotst, moet rechtsom worden gedraaid, als de robot rechts is gebotst, moet linksom worden gedraaid. Na TURN_TIME tijd te hebben gedraaid, heeft de robot het obstakel ontweken en gaat de automaat weer bezig met het pollen van de bumpers. Tevens wordt de wandertaak weer gestart.

Als we code genereren en compileren komen we erachter dat 'sys_time' niet gedeclareerd is. Sys_time wordt gebruikt voor de clocks (die we nu pas gebruiken, vandaar dat we deze melding nog niet eerder kregen) in de gegenereerde file brickos_system.h.
Via Google vind ik op LUGNET [11.1] het volgende:

> I know the value sys_time has been replaced with the function Get_sys_up_time(). You should be able to add this line at the beginning of the code:
>
> #define sys_time Get_sys_up_time() // v10 does not have sys_time
>
> Or, maybe that would do other bad things. You could just replace sys_time with Get_sys_up_time() everywhere in the code.

In de nieuwere versie van brickOS is sys_time vervangen voor get_system_up_time(). De TimesTool code generator gebruikt echter nog steeds sys_time, maar deze is dus nergens meer gedeclareerd. De oplossing aangedragen in [11.1] is ook wel een goede snellere oplossing voor de problemen met chanusage, trans en loc, dus we zetten bovenaan in Robot1.c het volgende:

```
#define sys_time        get_system_up_time()  // v10 does not have sys_time
#define CHANUSAGE       chanusage
#define TRANS           trans
#define LOC             loc
```

Dit moeten we elke keer doen wanneer we de code opnieuw genereren.

Als we nu de code uitvoeren op de robot rijdt de robot wel achteruit als hij ergens tegenaan botst, maar blijft dan ook achteruitrijden. De robot komt dus niet in een turn_left of turn_right toestand. Als we de waarde van bump op het lcdscherm printen krijgen we bij de ene bumper de waarde 257 en bij de andere bumper 259. Dit is niet gelijk aan 1 en 3 (het verschil wel), bump bevat dus verkeerde waarden, waarbij het dan niet raar is dat de robot niet naar links of rechts draait.
Om het probleem te omzeilen halen we 256 van de bump-waarde af. Als we aan het einde van de drive_backward-taak 256 van bump afhalen, krijgen we wel de juiste waarden (1 en 3) en draait de robot netjes de andere kant uit dan dat hij gebotst is. Wat de oorzaak voor het probleem is weten we echter niet. De code die nu gegenereerd is, staat in appendix C.
Als we de simulator laten draaien, zien we dat de bumperpoller en musictaak gestart worden, maar dat verder alleen het wander_process aan de beurt komt. De musictaak moeten we nog opsplitsen, zodat deze ook goed gescheduled kan worden. Dat het avoid_process verder niets meer doet, komt omdat deze nooit een bumpsignaal krijgt. In de simulator is er geen wereld met obstakels waar de gesimuleerde robot tegenaan kan botsen. We moeten dus nog een gedeelte van de omgeving modelleren om ervoor te zorgen dat de simulatie overeenkomt met de werkelijkheid. Omdat de robot in de simulator nooit een bumpsignaal krijgt, kunnen we ook niet testen met de analyse of hij daar ook de hoge waarden als signaal voor de bumpvariabele krijgt.

**Samenvatting**

We kwamen bij het verder opsplitsen van de wander- en avoidtaak de volgende problemen tegen:

- Bij non-deterministische keuzes komen de simulator en de werkelijkheid niet overeen.
- De codegenerator maakt gebruik van het oude sys_time wat niet meer bestaat.
- De bump variabele waarden zijn 256 te hoog.

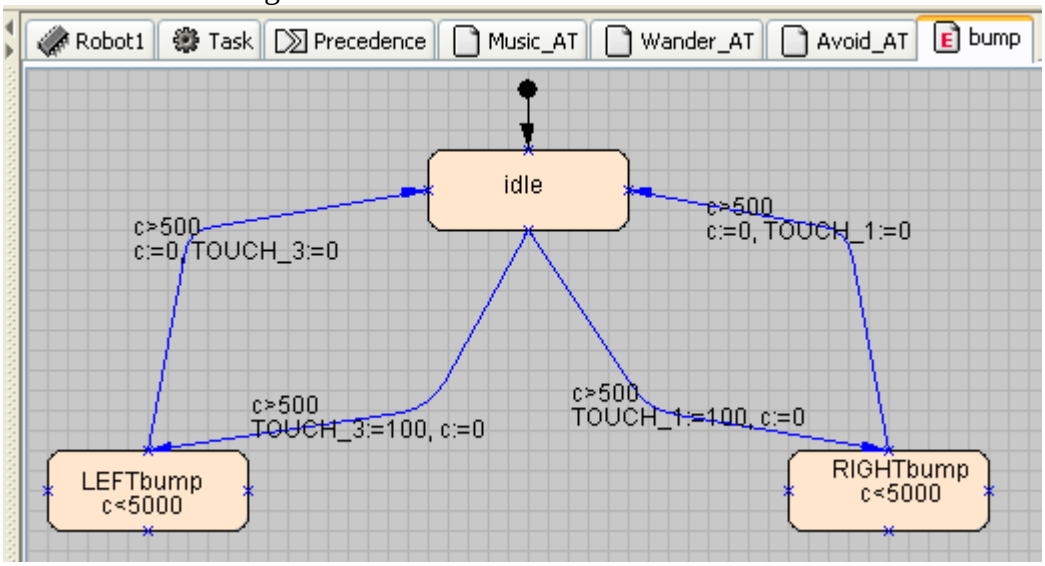
We hebben nu dus wel een versie waarbij de gegenereerde code uitvoerbaar is en ook doet wat bedoeld is, maar de analyse van het systeem komt er nog steeds niet mee overheen. Hiervoor moeten we nog de musictaak opdelen, dit doen we in paragraaf 5.5, en nog een bumpautomaat maken die er voor zorgt dat de robot in de analyse ook ergens tegenaan rijdt, dat laatste doen we in de volgende paragraaf.

## 5.4 De bumpautomaat

We moeten het model uitbreiden met iets dat botssignalen afgeeft. Over de omgeving vind ik in de handleiding [4.1] niets, maar in [12] staat over de productie cel het volgende:

> **Environment Model:** In order to analyse the behaviour of the production cell model, the abstract behaviour of its environment is modelled with the two timed automata Brick and Robot shown in Fig. 10 and Fig. 11 of Appendix A. The automaton Brick models bricks arriving on the feed belt, and Robot models the position of the robot, and how it behaves when the robot rotates. The interaction between the environment and the control program is modelled using three shared variables: DIR used to control the rotation of the robot, POS used to model the robot position, and LIGHT 2 used to sense the value of the light sensor at the start of the feed belt.

Voor de omgeving worden dus ook gewoon automaten gemaakt. We moeten dus ook een automaat maken voor het botsen. We gaan nu om een automaat maken die botsingen simuleert, zodat de analyse overeenkomt met de werkelijkheid. Er hoeft voor deze bumpautomaat echter geen code gegenereerd te worden, hij is geen onderdeel van de robot. Je kunt bij een automaat een vinkje zetten om aan te geven dat het bij de omgeving hoort. Er valt echter nergens te vinden of er dan ook geen code gegenereerd wordt.
We maken nu de volgende automaat om het botsen te simuleren:

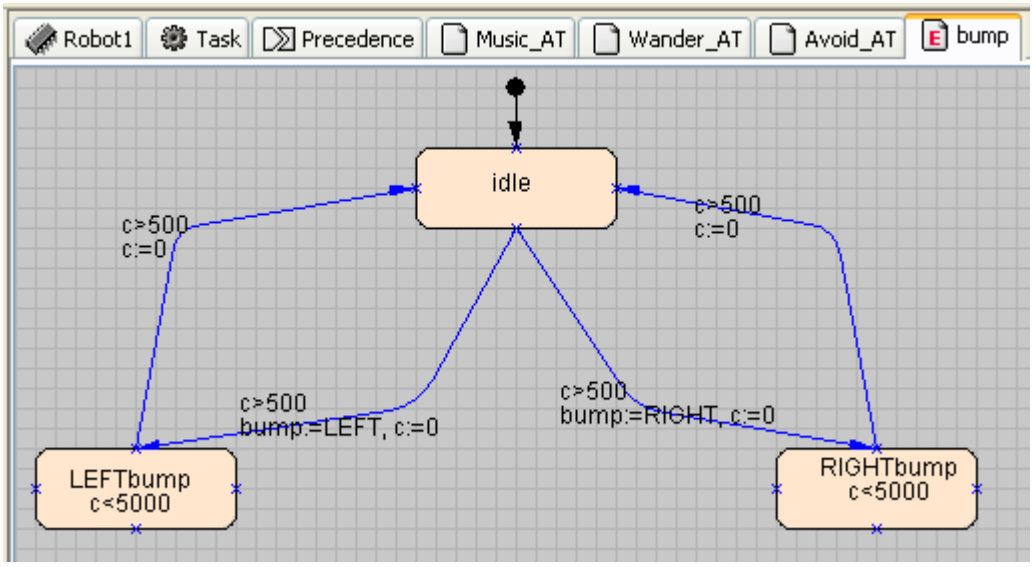



Hierin wordt ergens na 500 milliseconden willekeurig gekozen om de simulatie van of een rechtertastsensor- of een linkertastsensorsignaal (TOUCH_1 EN TOUCH_3) te genereren. Als we nu echter de simulator laten draaien, blijft de waarde van bump gelijk aan 0. Dit komt omdat we deze waarde in de code van de bumperPoller veranderen. Er wordt in de simulator niets met de code gedaan, dus het werkt zo niet. We hebben nu twee mogelijkheden om dit probleem op te lossen: ten eerste de botssimulatie-automaat aanpassen en hier direct de waarde van “bump” in aanpassen in plaats van die van TOUCH_1 en TOUCH_3, of de bumperPollertaak nog verder opdelen en niet meer met wait_event laten werken maar de functionaliteit hiervan direct in de automaat zetten. De eerste optie is waarschijnlijk het

makkelijkst, maar dan lijkt het minder op de werkelijkheid (het hele gedeelte met wait_event en wakeup_t wordt dan niet gemodelleerd in een automaat). De tweede optie is wat meer werk en heeft als nadeel dat het geheel dan minder op onze eigen geschreven code lijkt en dus minder goed vergelijkbaar is. We zullen beide opties proberen.

### 5.4.1 Optie 1

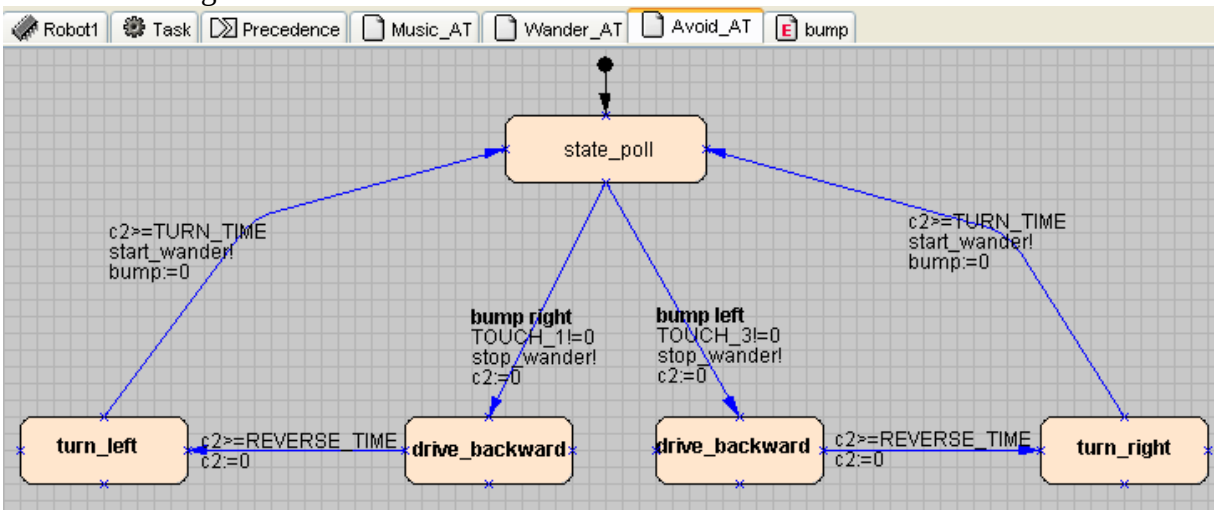
We veranderen TOUCH_3:=100 in bump:=LEFT en TOUCH_1:=100 in bump:=RIGHT. Omdat de waarde van bump in de avoidtaak weer op 0 wordt gezet, hoeven we dat hier niet te doen.




Nu worden er in de simulator ook botsingen gegenereerd. Als we de code uitvoeren vertoont de robot ook nog hetzelfde gedrag als in paragraaf 5.3.2.

### 5.4.2 Optie 2

We willen de bumperpollerfunctionaliteit uit de code halen en in een automaat zetten. We doen dat als volgt:




De bumperpollertaak bestaat nu niet meer. De functionaliteit hiervan is nu in de automaat gemodelleerd. Door in de automaat te controleren of de tastsensoren ingedrukt zijn, wordt dit in de simulatie ook gedaan. Hierdoor zijn de gegenereerde botsingen door de bumpautomaat (zie de afbeelding aan het begin van paragraaf 5.4) wel effectief. Omdat de bumpautomaat pas weer nieuwe bumpsignalen hoeft te genereren als de avoidtaak terug is in state_poll, willen

we hier eigenlijk ook over het channel start_wander synchroniseren. Er kan echter niet met meerdere automaten over hetzelfde channel gesynchroniseerd worden, de simulator kiest één van beide automaten. Er kan in één transitie ook niet over meerdere channels gesynchroniseerd worden. Als we de gegenereerde code proberen te compileren krijgen we een aantal niet direct verklaarbare foutmeldingen. Het zou te maken kunnen hebben met het toekennen van waarden aan TOUCH_1 en TOUCH_3 omdat dit interne variabelen in brickOS zijn. We kunnen achter TOUCH_1 en TOUCH_3 vinkjes zetten in de kolom "Env", om aan te geven dat ze bij de omgeving horen en niet bij de robot. Maar ze worden dan niet weggelaten uit de gegenereerde code, wat we wel verwacht hadden.

**Samenvatting**

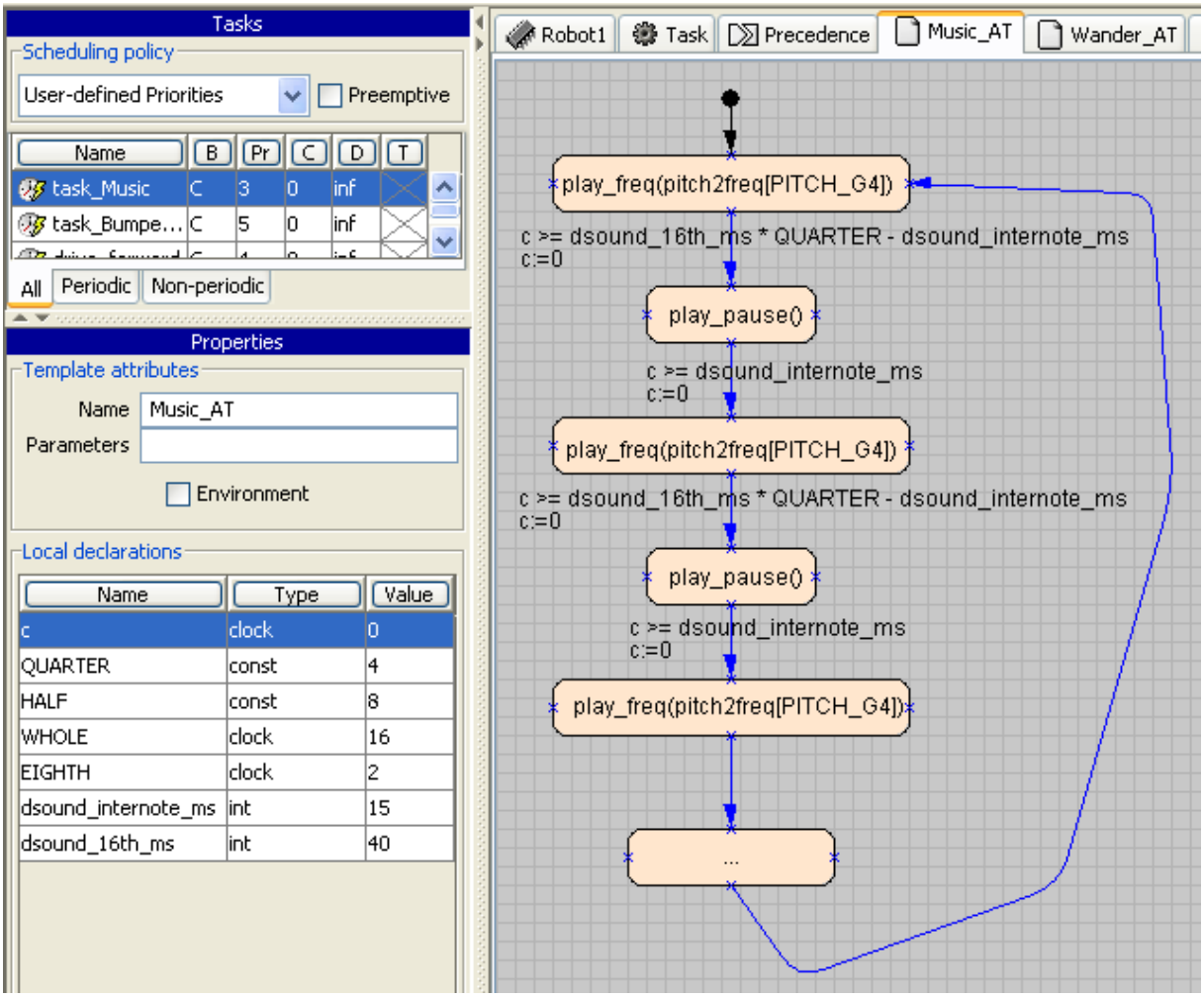
Optie 2 voor de bumpautomaat geeft teveel problemen. We gaan dus alleen verder met optie 1. Het avoidgedrag wordt nu wel goed gescheduled, maar voor een volledige analyse moeten we de musictaak ook nog opsplitsen.

## 5.5 Opsplitsing van de Musictaak

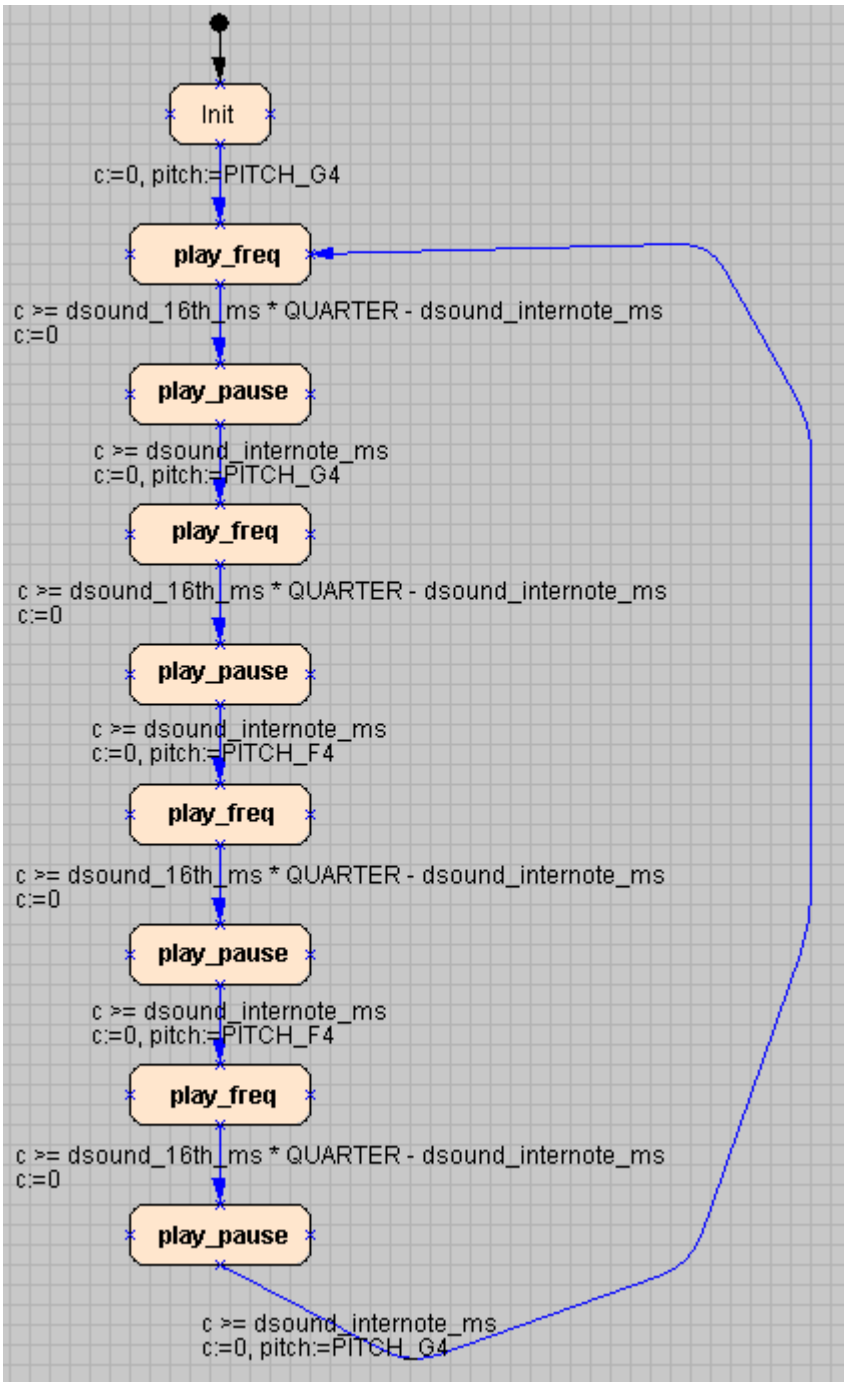
We gaan nu de musictaak ook opsplitsen, zodat we daar bij de schedulability-analyse ook iets aan kunnen zien. Omdat optie 1 van de bumpautomaat het doet, gaan we daar nu mee verder.
Tot nu toe gebruikten we dsound_play() om een rijtje noten af te spelen. We de functionaliteit nu modeleren in een automaat.
Dsound_play() wordt gedefinieerd op regel 234 van dsound.h [5.4]:

```
00234 static inline void dsound_play(const note_t *notes) {
00235   dsound_next_note=(volatile note_t*) notes;
00236   dsound_next_time=0;
00237 }
```

Een note_t is een structure bestaande uit een pitch(toonhoogte) en een length. Met d_sound_setduration() geef je aan hoeveel milliseconden een 16de noot duurt. In de zelfgeschreven code (hoofdstuk 3) duurde een $16^{de}$ noot 40 milliseconden, dus een kwart noot duurt 160 milliseconden en een halve noot duurt 320 milliseconden. De musictaak kan bij PITCH_PAUSE switchen naar een andere taak. Dit is in ons liedje na 4160 milliseconden vanaf het begin. Tussen twee opeenvolgende noten zit ook elke keer een kleine pauze, de internote-tijd, hier kan dus ook geswitcht worden van proces. Deze lengte van deze internote staat in dsound_internote_ms. Dsound_internote_ms staat standaard op 15 milliseconden (zie [5.4]). Het omzetten van een rijtje noten met pitch en length naar frequenties en tijden wordt gedaan in de functie dsound_core() (zie dsound.c [5.5]).
Hier wordt er eerst een bepaalde frequentie aan gezet voor de lengte van de noot min de internote-tijd. Dan komt de internote-tijd, waarbij het geluid uit wordt gezet. En dan weer de volgende toon. We gaan nu deze functionaliteit in een automaat zetten. Als we dit, voor een klein stukje van het liedje, in de TimesTool proberen te tekenen, lijkt het iets te worden als:




Waarbij play_freq() en play_pause() taken zijn. In brickOS zijn dit interne functies die de user niet kan aanroepen. Maar omdat we anders geen goede analyse kunnen uitvoeren, moeten we ze nu toch maar gebruiken. Op deze manier werkt het alleen niet, omdat we de pitch meegegeven moet worden aan de play_frec-taak, maar er kan niets meegeven worden in TimesTool aan taken. Een taak heeft wel een vakje voor “Input”, maar in de handleiding staat hier echter niets over. Bij de voorbeelden van de TimesTool is ook bij geen enkel voorbeeld dit vakje ingevuld. Aangezien het inputveld nergens gebruikt of uitgelegd wordt, weten we dus niet hoe het gebruikt moet worden. We moeten het dus op een andere manier oplossen. Het is niet handig om voor elke frequentie een aparte taak te maken want er zijn teveel frequenties, waardoor we veel te veel taken zouden krijgen die bijna hetzelfde zijn. Het handigst zou zijn om het hele liedje ergens op te slaan en dit dan aan de Music_AT mee te geven (bij automaten kunnen er wel parameters ingevuld worden). Er kan alleen nergens iets gecompliceerders dan een int, clock of channel worden opgeslagen, dan in de code van de taken, en daar willen we het nu juist niet hebben. We zetten nu alleen de lijst met pitch2freq maar in de play_freq taak en gebruiken de pitch-waarde als variabele genaamd “pitch”. Om ervoor te zorgen dat de play_freq taak deze variabele ook kan gebruiken, maken we hem globaal in plaats van locaal. Omdat het een erg grote automaat wordt als we het hele liedje in de automaat zetten, maken we een korter liedje met maar 4 noten, die telkens herhaald worden.

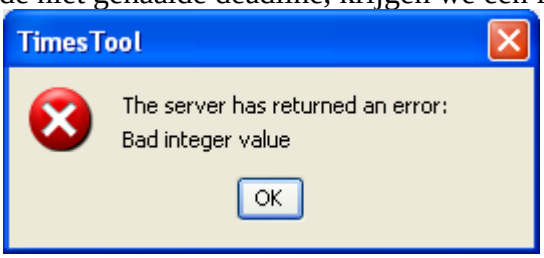


We kunnen nu eindelijk in de simulatie de hele scheduling zien.

Als we nu echter code genereren en uitvoeren, rijd de robot nu ook regelmatig achteruit zonder eerst ergens tegenaan gebotst te zijn. Dit is het werk van de bumpautomaat. Deze wordt dus gewoon mee gegenereerd, wat niet de bedoeling was. Om toch goede code te genereren halen we het proces van de bumpautomaat weg. Dit blijkt echter geen goed idee, de robot doet nu helemaal niets meer. Zelfs het programma opnieuw downloaden of de firmware downloaden lukt niet meer. De robot wil ook niet meer uit. De enige oplossing is de batterijen eruit halen en weer in doen, zodat alles gereset is. Blijkbaar is het programma teveel in de war geraakt door het ene proces weg te halen, dus zetten we dit maar weer terug.

Als we de schedulability analyse optie uitproberen, zegt deze dat het systeem niet schedulebaar is omdat de taak drive_forward zijn deadline niet haalt. Dit is vreemd, aangezien

drive_forward nog helemaal geen deadline heeft. We hadden ondertussen alle taken een computation time van 1 gegeven, maar nog geen deadlines. Hoe deze taak dan toch zijn deadline niet kan halen, is een raadsel. Als we de deadlines van alle taken op 10 zetten, is het geheel volgens de analyse nog niet te schedulebaar. Dit terwijl een deadline van 10 makkelijk te halen moet zijn aangezien alle taken maar 1 tick duren. Als we de trace willen bekijken van de niet gehaalde deadline, krijgen we een foutmelding te zien waar we niet veel mee kunnen:




We beschouwen dit als een bug.

Om toch nog uitvoerbare code te kunnen genereren halen we nu niet alleen het proces van de bumpautomaat weg, maar ook de bumpautomaat zelf. De robot kan de code nu wel goed uitvoeren, maar als de robot ergens tegenaan botst, rijd hij alleen maar achteruit zonder daarna nog iets te doen. Dit zijn we eerder tegengekomen in paragraaf 5.3.2, toen hebben we het opgelost door 256 van de bumpwaarde af te halen. We bekijken de bumpwaarde nu weer en deze blijkt '-255' te zijn in plaats van 1. We verwijderen de regel waar we 256 van de bumpwaarde afhalen dan maar weer uit de code. Hierna doet de robot wel wat we verwachten, maar het geheel blijft erg vreemd. Omdat we de bumpautomaat weg hebben gehaald, is de simulatie niet erg interessant meer, er worden immers geen botsingen meer gesimuleerd.

**Samenvatting**

Bij het opsplitsen van de musictaak kwamen we de volgende problemen tegen:

- Er kunnen geen parameters meegegeven worden aan taken.
- int, clock en channel zijn de enige type variabelen die gebruikt kunnen worden in een automaat.
- We worden gedwongen diep in de musiccode te duiken om een analyseerbaar model te maken.
- Voor environment automaten en variabelen wordt ook code gegenereerd,
- De schedulability analyse van de TimesTool werkt niet helemaal naar behoren.
- We hebben uiteindelijk twee verschillende modellen moeten maken, één voor de analyse en één voor de code generatie.

Uiteindelijk hebben we dus wel een model dat analyseerbaar is en een model dat code genereert, maar niet één model dat beide goed kan en erg gemakkelijk ging het niet.

# 6 Conclusie

We zullen in dit hoofdstuk nagaan of we antwoord kunnen geven op de deelvragen. Daarna zullen we een antwoord proberen te vinden op de hoofdvraag.

## 6.1 Deelvraag 1

De eerste deelvraag was: Hoe lang duurt het om de tool te leren?
We zien dat het niet al te lang duurt om iets in de tool te maken. Maar een model maken dat ook goed analyseerbaar is en vervolgens ook nog goede code genereert, duurt wel lang. Dit heeft vooral te maken met het gebrek aan goede documentatie. In de handleiding staat niet goed uitgelegd hoe een systeem gemodelleerd moet worden. Verder is er weinig documentatie te vinden.

## 6.2 Deelvraag 2

De tweede deelvraag was: Waar wordt de computation time voor verschillende taken berekend?
De computation time wordt niet door TimesTool berekend, deze moet dus van te voren bekend zijn of moet handmatig gemeten/berekend worden.

## 6.3 Deelvraag 3

Doet de gegenereerde code wat je verwacht?
De gegenereerde code deed niet altijd wat we verwachten. Een aantal keer lag dit aan het verkeerd modelleren, maar ook een aantal keer zijn we er niet achter gekomen wat het probleem was. Dit was bijvoorbeeld het geval bij de bumpvariabele die 256 hoger bleek dan we verwachten. Omdat we de code voor de taken zelf gemaakt hebben, deed dit wel altijd wat we verwachten. De gegenereerde code maakt alleen de taken aan en voegt de functionaliteit voor deze taken in.
De gegenereerde code deed ook niet altijd wat het model in de simulatie liet zien, dit was het geval bij een non-deterministische keuze tussen twee transities zoals we in paragraaf 5.3.1 hebben gezien, in de gegenereerde code werd hier altijd voor dezelfde transitie gekozen, terwijl de simulatie een willekeurige transitie nam.

## 6.4 Deelvraag 4

In hoeverre moet je de omgeving modelleren in de tool?
Als er interactie plaats vindt met de omgeving, moet dit ook gemodelleerd worden. Dit is vooral noodzakelijk voor de analyse. Voor code generatie kun je beter de omgeving niet modelleren, daar raakt het alleen maar van in de war. Er is geen goede mogelijkheid om aan te geven dat iets onderdeel is van de omgeving en dus niet meegenomen hoeft te worden in de code generatie

## 6.5 Deelvraag 5

Gebruikt het gegenereerde programma meer/minder geheugen dan een niet gegenereerd programma?
Het bestand dat naar de RCX geüpload wordt, is een lx-bestand. Dit is bij onze eigen geschreven code 1422 bytes groot. Bij de laatste versie van ons model wordt bij de gegenereerde code een bestand van 3018 bytes geüpload. Dit is dus meer dan twee keer zo

groot. We kunnen ook naar het aantal regels code kijken. Onze eigen geschreven code telt 260 regels. De gegenereerde Robot1.c-file telt 353 regels. Verder worden er ook nog allerlei files gegenereerd met hulpfuncties. Deze worden niet altijd allemaal gebruikt, maar worden gegenereerd om de code zo standaard mogelijk te maken. Het gegenereerde programma gebruikt dus meer geheugen dan het zelf geprogrammeerde programma. We merken dit in de praktijk vooral doordat het programma langer nodig heeft om geüpload te worden

## 6.6 Conclusie

Onze onderzoeksvraag was:
Wat zijn de voor- en nadelen van het genereren van code door de TimesTool voor de RCX ten opzichte van het zelf programmeren bij een relatief klein voorbeeld?
We kunnen nu antwoord geven op deze vraag.
Het grootste voordeel van het gebruik van de tool, is dat je schedulability analyse uit kunt voeren. Dit waren we echter bij ons kleine voorbeeld niet nodig.
Er zijn een aantal nadelen aan het gebruik van de tool. Als je voor de eerste keer de TimesTool gebruikt, duurt het redelijk lang voordat je door hebt wat de TimesTool van een model verwacht. Ten tweede moet je, als je schedulability-analyse wilt uitvoeren, de code veel verder uitsplitsen dan je bij het zelf programmeren hoeft. Dit zagen we vooral bij de musictaak (paragraaf 5.5). Een ander nadeel is dat de gegenereerde code niet in alle gevallen deed wat we verwachten. Als laatste gebruikt het gegenereerde programma meer geheugen, dan het zelf geprogrammeerde programma.
We kunnen dus concluderen dat je een eenvoudig schedulebaar systeem beter zelf kunt programmeren dan de TimesTool gebruiken. De tool zou wel potentieel interessant kunnen zijn voor toepassingen waar het schedulen niet meer goed handmatig te analyseren is.

# 7 Referenties


[1] Telelogic Rhapsody: http://modeling.telelogic.com/products/rhapsody/index.cfm
[2] Rational Rose: http://www-306.ibm.com/software/awdtools/developer/rose/
[3] Uppsala University, Sweden: http://www.uu.se/
  [3.1] Darts (Design and Analysis of Real-Time Systems) team: http://www.it.uu.se/research/group/darts
  [3.2] Larsen, Kim G. Pettersson, Paul, and Yi, Wang, Uppaal in a Nutshell, *Int.Journal on Software Tools for Technology Transfer 1, 1–2 (Oct.)*, pagina's 134–152, 1997 http://www.uppaal.com
[4] TimesTool: http://www.timestool.com/
  [4.1] The TIMES Tool version 1.0 beta User Manual, http://www.timestool.com/documentation.shtml
  [4.2] Code synthesis: http://www.it.uu.se/research/group/darts/times/synthesis.shtml
[5] brickOS http://brickos.sourceforge.net/
  [5.1] wakeup: http://brickos.sourceforge.net/docs/APIs/html-c/tm_8h.html#a17
  [5.2] wait_event: http://brickos.sourceforge.net/docs/APIs/html-c/unistd_8h.html#a7
  [5.3] time.h: http://brickos.sourceforge.net/docs/APIs/html-c/time_8h.html
  [5.4] dsound.h: http://brickos.sourceforge.net/docs/APIs/html-c/dsound_8h.html
  [5.5] dsound.c: http://brickos.sourceforge.net/docs/APIs/html-kern/dsound_8c-source.html
[6] RCX: http://mindstorms.lego.com/eng/products/ris/index.asp
[7] Robotics Invention System 2.0 Constructopedia, De LEGO Groep, 1999/2000
[8] Devil with a Blue Dress: http://brickos.sourceforge.net/docs/APIs/html-c/C-Demos.html
[9] Stig Nielsson, Introduction to the legOS kernel, 2007
[10] Tobias Amnell, Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. TIMES: a Tool for Schedulability Analysis and Code Generation of Real-Time Systems. *Proceedings of the 1st International Workshop on Formal Modeling and Analysis of Timed Systems, FORMATS 2003, Marseille, France, September 6-7*, 2003
[11] LUGNET (International LEGO Users Group Network) www.lugnet.com
  [11.1] Errors compiling threads http://news.lugnet.com/robotics/rcx/legos/?n=3916&t=f&v=a
[12] Tobias Amnell, Elena Fersman, Paul Pettersson, Hongyan Sun, Wang Yi. Code Synthesis for Timed Automata. *Nordic Journal of Computing (NJC)*, volume 9, nummer 4, pagina's 269-300, 2002
[13] BricxCC: http://bricxcc.sourceforge.net/
[14] Cygwin: http://www.cygwin.com/
[15] LEGO: www.lego.com
  [15.1] MindStorms Technical Support FAQs: http://www.lego.com/eng/service/faqs.asp?section=ConsumerService-FAQ-TechSupport&tech=true&catid=87BC4CA6-D8CD-4BF1-8307-6B52AB45AF02
[16] William Stallings, *Operating Systems*, Prentice Hall, Upper Saddle River, New Jersey, 2001
[17] Tobias Amnell, Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. Times - A Tool for Modelling and Implementation of Embedded Systems. *Tools and Algorithms for the Construction and Analysis of Systems, proceedings of 8th International Conference, TACAS 2002, part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 (Grenoble, France, April 8-12, 2002),*

Lecture Notes in Computer Science, Vol.2280, pagina's 460-464, Springer-Verlag, 2002
[18] Luis Villa, LegOS HOWTO, http://legos.sourceforge.net/HOWTO/HOWTO.ps, 2000
[19] Allan Bedford, *Unofficial Lego Builder's guide*, No Starch Press, San Francisco, 2005

# Appendix A: Installatie en configuratie

We beschrijven de installatie van alle tools die nodig zijn om met TimesTool code te genereren en uit te voeren op de LEGO RCX. De beschreven methode is getest op een computer met Microsoft Windows XP SP2.
We hebben niet alleen de TimesTool zelf nodig, maar voor het uitvoeren van de gegenereerde code, moeten we deze eerst kunnen compileren en vervolgens kunnen uploaden. Voor het uitvoeren van het compileren en uploaden gebruiken we BricxCC [13]. Voor het compileren gebruiken we cygwin [14] en een Hitachi H8 crosscompiler. Verder zijn we het brickOS [5] besturingssysteem nodig voor op de RCX. Voor het uploaden moeten we de infrarood toren installeren.

## *a. BricxCC installeren*

Op de site van BricxCC kun je de laatste versie van BricxCC downloaden, wat op dit moment versie 3.3.7.18 is. Als je niet van plan bent om iets met NQC te gaan doen (wat we nu niet van plan zijn), kun je ervoor kiezen om deze componenten niet te installeren.

## *b. Cygwin en de Hitachi H8 crosscompiler installeren*

We zijn cygwin nodig om de C-code te kunnen compileren. De processor van de RCX is een Hitachi H8, hiervoor zijn we dus een crosscompiler nodig. Er is een alles-in-één installatiebestand te vinden op de site van BricxCC te vinden met de benodigde onderdelen van cygwin en kant en klare H8 tools. Bij de installatie kunnen we ervoor kiezen om de H8 Gnu Pascal Compiler niet te installeren, we gebruiken immers toch geen Pascal. Voor een goede werking is het aan te raden te installeren op C:\cygwin (wat ook de standaard installatielocatie is).

## *c. BrickOS installeren*

Op de Site van BricxCC is ook een minimale installatie van brickOS te vinden. Let op dat deze in de cygwin directory wordt geïnstalleerd. We kunnen hierbij helaas niet kiezen om het leJOS gedeelte niet te installeren.

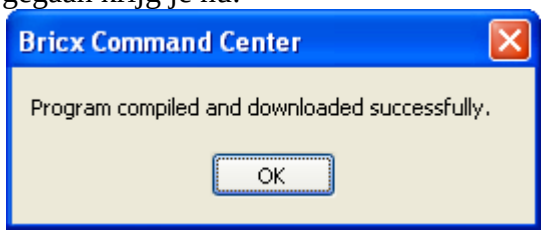
## *d. De IR-tower installeren*

Van de LEGO site, kun je op bij de customer service [15.1], drivers downloaden voor de IR-tower. Op het moment dat je deze installeert, moet je de IR-tower niet aangesloten hebben. Na de installatie van de drivers, kun je de IR-tower aansluiten op een USB-poort. Windows geeft vervolgens een melding dat hij een USB-tower heeft gevonden.

## *e. BricxCC testen*

Om te testen of alles tot nu toe goed geïnstalleerd hebben, proberen we even een demobestand te compileren en te uploaden naar de Robot. Als we BricxCC starten, krijgen we een venstertje waar we een Port (usb) en Firmware (brickOS) moeten kiezen. Als je de RCX voor de IR-tower aan hebt staan, dan kan bricxCC deze ook vinden, anders moet je voor het uploaden eerst nog even Tools->Find Brick doen. Bij Edit->Preferences zetten we een vinkje bij “Show compiler status message”, zodat we een melding krijgen of het compileren al dan niet geslaagd is.

We moeten nu eerst nog de firmware naar de RCX uploaden, dit hoeft alleen opnieuw als de batterijen van de RCX vervangen worden. We kunnen dit met BricxCC doen via tools->Download Firmware. De firmware staat in cygwin\brickos\boot. De RCX laat een geluidje horen als het laden van de firmware klaar is
In de map cygwin\brickos\demo staan een aantal demoprogramma's. Hiervan openen we er eentje. We compileren dit door Compile->Compile te doen, of op F5 te drukken. Vervolgens uploaden we het programma door Compile->Download te doen of F6. Als alles goed is gegaan krijg je nu:




## f. De TimesTool installeren

Het enige wat nu nog geïnstalleerd moet worden, is de TimesTool zelf. Op de site [4] kun je de laatste versie downloaden, wat momenteel versie 1.3 beta (build 2461) is, en deze installeren.

## g. BrickOS configureren

Op de site van de TimesTool, lezen we bij code synthesis dat we $(CMACROS) moeten toevoegen aan CFLAGS. We openen hiervoor brickos\makefile.common. Op regel 98 staat:
CFLAGS=$(COPT) $(CWARN) $(CINC) $(CDEFINES)
Dit moet dus veranderd worden in:
CFLAGS=$(COPT) $(CWARN) $(CINC) $(CDEFINES) $(CMACROS)

## h. TimesTool configureren

Het enige noodzakelijke is de TimesTool vertellen waar brickOS geïnstalleerd is. Hiervoor open je de configuratie in de TimesTool via Options->Configuration. Op het tabblad "Code generator" moet je bij "brickOS directory" C:\cygwin\brickos invullen. Voor de overige configuratiemogelijkheden kun je de Times User Manual [4.1] raadplegen. Je kunt nu de TimesTool gebruiken en code genereren, zie hoofdstuk 4.

## i. BricxCC configureren

Voordat je de code met BricxCC kunt compileren, moet je de code eerst gegenereerd hebben met de TimesTool. Bij de gegenereerde bestanden zit ook een makefile, waar onder andere iets in staat als CMACROS=-imacros <projectnaam>.h. Hierbij is <projectnaam> afhankelijk van hoe je je project in TimesTool hebt genoemd. Omdat we deze makefile niet gebruiken, maar BricxCC gebruiken om te compileren, moeten we deze regel toevoegen aan de makefile van BricxCC. De makefile van BricxCC is te vinden bij Edit->Preferences->Compiler->C/C++/Pascal. Hier voegen we dus de regel over CMACROS uit de gegenereerde makefile in.
Je kunt nu het gegenereerde bestand <projectnaam>.c openen met BricxCC. Nu moeten we nog aangeven welke bestanden nog meer tot het project behoren. Dat zijn <projectnaam>_init.c en brickos_kernel.c. Dit doen we door bij View->Project Manager deze twee bestanden toe te voegen. Alles is nu klaar om gecompileerd en geüpload te worden.

# Appendix B: Robot code

```
/*********************************************
  dit programma gaat uit van een robot met de volgende aansluitingen:
  rechter motor op output A
  linker motor op output B
  rechter bumpersensor op input 1
  linker bumpersensor op input 3
  lichtsensor op input 2
*********************************************/

/********bibliotheken die we nodig hebben********/
#include <dmotor.h>
#include <dsensor.h>
#include <dsound.h>
#include <rom/lcd.h>
#include <stdlib.h>
#include <unistd.h>
/*********************************************/

/******************Constanten******************/
#define REVERSE_TIME            500        //Tijd om achteruit te rijden bij avoid
#define AVOID_TURN_TIME         500        //Tijd om te draaien bij avoid
#define MAX_MOVE_TIME           900        //Maximale tijd om vooruit te rijden bij wander
#define MIN_MOVE_TIME           200        //Minimale tijd om vooruit te rijden bij wander
#define MAX_TURN_TIME           300        //Maximale tijd om te draaien bij wander
#define MIN_TURN_TIME           200        //Minimale tijd om te draaien bij wander
#define RIGHT                    1         //rechterbumper
#define LEFT                     3         //linkerbumper
#define RIGHT_BUMPER            TOUCH_1
#define LEFT_BUMPER             TOUCH_3
/*********************************************/

/******************de threads******************/
tid_t av;
tid_t wand;
tid_t sound;
/*********************************************/

/**********standaard bewegings functies**********/
void forward ()
{
     motor_a_dir(fwd);
     motor_c_dir(fwd);
     motor_a_speed(MAX_SPEED);
     motor_c_speed(MAX_SPEED);
}

void back ()
{
     motor_a_dir(rev);
     motor_c_dir(rev);
     motor_a_speed(MAX_SPEED);
     motor_c_speed(MAX_SPEED);
}

void turn_left()
{
     motor_a_dir(fwd);
     motor_c_dir(rev);
     motor_a_speed(MAX_SPEED);
     motor_c_speed(MAX_SPEED);
}


void turn_right()
{
     motor_a_dir(rev);
     motor_c_dir(fwd);
     motor_a_speed(MAX_SPEED);
     motor_c_speed(MAX_SPEED);
}
```

```
void stop()
{
     motor_a_dir(off);
     motor_c_dir(off);
     motor_a_speed(0);
     motor_c_speed(0);
}
/*********************************************/

/*****bewegingsfucties voor een bepaalde tijd*****/
void forward_t (int time) //tijd in msec
{
    forward();
    msleep(time);
    stop();
}

void back_t (int time) //tijd in msec
{
    back();
    msleep(time);
    stop();
}

void turn_left_t(int time) //tijd in msec
{
    turn_left();
    msleep(time);
    stop();
}

void turn_right_t(int time) //tijd in msec
{
    turn_right();
    msleep(time);
    stop();
}
/*********************************************/

/*************het wander gedeelte***************/
//returnt een random getal tussen min en max
//incl min, exl max
int rand (int min, int max)
{
     long int rand_nr = random();
     rand_nr = (rand_nr%(max-min))+min;
     return rand_nr;
}

void wander ()
{
     int i;
     int forward_time;
     while (!shutdown_requested())
     {
          forward_time = rand(MIN_MOVE_TIME, MAX_MOVE_TIME);
          lcd_int(forward_time);
          forward_t(forward_time);

          i = rand(0,2);  //random links of rechts draaien
          if (i==0)
          {
               lcd_int(i);
               turn_left_t(rand(MIN_TURN_TIME, MAX_TURN_TIME));
          }
          else
          {
               lcd_int(i);
               turn_right_t(rand(MIN_TURN_TIME, MAX_TURN_TIME));
          }
     }
}
/*********************************************/

/*************het avoid gedeelte****************/
wakeup_t bumper_poller ()
```

```
{
     int bump = 0;
     if(RIGHT_BUMPER!=0)
     {
          bump = RIGHT;
     }
     if(LEFT_BUMPER!=0)
     {
          bump = LEFT;
     }
     return bump;
}

void avoid()
{
     int bump = 0;
     while(!shutdown_requested())
     {
          bump = wait_event(bumper_poller,0);  //wacht tot bumper ingedrukt wordt
          kill(wand); //stop het wandergedrag
          back_t(REVERSE_TIME);
          if(bump==RIGHT)
          {
               cputs("b1");
               turn_left_t(AVOID_TURN_TIME);
          }
          else //bump==LEFT
          {
               cputs("b3");
               turn_right_t(AVOID_TURN_TIME);
          }
          wand = execi(&wander,0,NULL,5,DEFAULT_STACK_SIZE); //start het wander gedrag weer
          bump=0;
     }
}
/*********************************************/

/*************het sound gedeelte***************/
//een liedje
static const note_t devil[] = {
  { PITCH_G4, QUARTER },
  { PITCH_G4, QUARTER },
  { PITCH_G4, QUARTER },
  { PITCH_G4, QUARTER },
  { PITCH_G4, HALF },
  { PITCH_G4, HALF },

  { PITCH_G4, HALF },
  { PITCH_G4, HALF },
  { PITCH_G4, HALF },
  { PITCH_G4, HALF },

  { PITCH_F4, QUARTER },
  { PITCH_F4, QUARTER },
  { PITCH_F4, QUARTER },
  { PITCH_F4, QUARTER },
  { PITCH_F4, HALF },
  { PITCH_F4, HALF },

  { PITCH_F4, HALF },
  { PITCH_PAUSE, HALF },
  { PITCH_PAUSE, HALF },
  { PITCH_PAUSE, HALF },

  { PITCH_E4, QUARTER },
  { PITCH_E4, QUARTER },
  { PITCH_E4, QUARTER },
  { PITCH_E4, QUARTER },
  { PITCH_F4, HALF },
  { PITCH_F4, HALF },

  { PITCH_E4, HALF },
  { PITCH_E4, HALF },
  { PITCH_F4, HALF },
  { PITCH_F4, HALF },

  { PITCH_E4, QUARTER },
```

```
  { PITCH_E4, QUARTER },
  { PITCH_E4, QUARTER },
  { PITCH_E4, QUARTER },
  { PITCH_F4, HALF },
  { PITCH_F4, HALF },

  { PITCH_E4, HALF },
  { PITCH_PAUSE, HALF },
  { PITCH_PAUSE, HALF },
  { PITCH_PAUSE, HALF },
  { PITCH_END, 0 }
};

void music()
{

     dsound_set_duration(40); //om ervoor te zorgen dat het liedje een beetje sneller gaat
     while (!shutdown_requested())
     {
          if (wait_event(dsound_finished,0) != 0)
            {
               dsound_play(devil);
          }
     }
}
/*********************************************/

/**********de main functie*********************/
void main ()
{
     srandom(300); //seed de random functie
     av = execi(&avoid,0,NULL,5,DEFAULT_STACK_SIZE);
     wand = execi(&wander,0,NULL,5,DEFAULT_STACK_SIZE);
     sound = execi(&music,0,NULL,4,DEFAULT_STACK_SIZE);
}
/*********************************************/
```

# Appendix C: Gegenereerde code

In deze appendix staat de code zoals gegenereerd door de TimesTool. Deze code werkt wel, maar het model waar de code uit gegenereerd is, is niet geschikt voor analyse.

## *a. Robot1.c*

```
/* This code was AUTOMATICALLY generated.
 * --> EDIT WITH CARE! <--
 */

/**
 * @file Robot1.c
 * @author TimesTool, Version 1.3 beta, April, 2007 (build 2461) <www.timestool.com>
 * @brief Generated by: Laura
 * @date Fri Dec 28 15:06:33 CET 2007
 * - with Target:          brickos
 *
 */

#include "Robot1.h"
#include "brickos_interface.h"
#include "Robot1_global.h"
#include "Robot1_init.h"
/**
 * @name Task identifiers (tid).
 */
#define tid_offset 200
#define tid_task_BumperPoller tid_offset+0
#define tid_drive_backward tid_offset+1
#define tid_turn_left tid_offset+2
#define tid_turn_right tid_offset+3
#define tid_drive_forward tid_offset+4
#define tid_task_Music tid_offset+5
#define tid_NOP tid_offset+6

#define sys_time        get_system_up_time()  // v10 does not have sys_time
#define CHANUSAGE       chanusage
#define TRANS           trans
#define LOC             loc


/**
 * Location identifiers, also offsets into
 * location array.
 */
#define Process_Avoid_S0 0
#define Process_Avoid_S1 3
#define Process_Avoid_S2 5
#define Process_Avoid_S3 7
#define Process_Music_state_Music 9
#define Process_Wander_S0 10
#define Process_Wander_S1 14
#define Process_Wander_S2 17
#define Process_Wander_stop 20

char release_list[NB_TASK]={ 1,0,0,0,1,1};


wakeup_t task_release(wakeup_t data) {
  if(release_list[data]) {
    switch (data) {
    default: return true;
  }
  } else
  return false;
}

wakeup_t task_complete(wakeup_t tid) {
  switch(tid) {
  case 0:case 1:case 2:case 3:case 4:case 5:}
  return true;
}
```

```
/**
 * Constant values
 */
#define RIGHT 1
#define LEFT 3
#define Process_Avoid_TURN_TIME 500
#define Process_Avoid_REVERSE_TIME 500
#define Process_Wander_FORWARD_TIME 550
#define Process_Wander_TURN_TIME 150

/**
 * Clock variables
 * Ordered: global first, then local clocks for each process.
 */
time_t clock_Process_Avoid_c2;
time_t clock_Process_Wander_c1;
/**
 * Integer variables
 */
int bump=0;
int* IVARS[NB_VAR] ={&bump};

void COPY2LOCAL( int tid, int instance ) {
  switch( tid ) {
  }
};

/**
 * Channel identfiers, one each for sending
 * and receiving, also offsets into
 * chanusage[] array
 */
#define stop_wanderS 0
#define start_wanderS 2
#define stop_wanderR 5
#define start_wanderR 9
#define recv_channels stop_wanderR


/**
 * List of transitions that uses a channel.
 */
int CHANUSAGE[11] = {
  0,NB_TRANS
  ,3,4,NB_TRANS
  ,9,10,11,NB_TRANS
  ,12,NB_TRANS
  };

/**
 * Evaluate guards on transition trn.
 * @param trn Transition id
 * @return true if guard satisfied, false otherwise.
 */
bool eval_guard(int trn) {
  switch(trn) {
  case 0: return (bump!=0);
  case 1: return (rdClock(Process_Avoid_c2)>=Process_Avoid_REVERSE_TIME&&bump==RIGHT);
  case 2: return (rdClock(Process_Avoid_c2)>=Process_Avoid_REVERSE_TIME&&bump==LEFT);
  case 3: return (rdClock(Process_Avoid_c2)>=Process_Avoid_TURN_TIME);
  case 4: return (rdClock(Process_Avoid_c2)>=Process_Avoid_TURN_TIME);
  case 5: return (rdClock(Process_Wander_c1)>=Process_Wander_TURN_TIME);
  case 6: return (rdClock(Process_Wander_c1)>=Process_Wander_FORWARD_TIME);
  case 7: return (rdClock(Process_Wander_c1)>=Process_Wander_FORWARD_TIME);
  case 8: return (rdClock(Process_Wander_c1)>=Process_Wander_TURN_TIME);
  case 9:
  case 10:
  case 11:
  case 12:
    return true;
  }
  return false;
}

/**
 * Perform assignments on transition trn.
 * @param trn Transition id.
```

```
 */
void assign(int trn) {
  switch(trn) {
  case 0:
    setClock(Process_Avoid_c2,0); break;
  case 1:
    setClock(Process_Avoid_c2,0); break;
  case 2:
    setClock(Process_Avoid_c2,0); break;
  case 3:
    bump=0; break;
  case 4:
    bump=0; break;
  case 5:
    setClock(Process_Wander_c1,0); break;
  case 6:
    setClock(Process_Wander_c1,0); break;
  case 7:
    setClock(Process_Wander_c1,0); break;
  case 8:
    setClock(Process_Wander_c1,0); break;

  }
}

/**
 * Transition table.
 */
trans_t TRANS[NB_TRANS] = {
  {true,Process_Avoid_S1,Process_Avoid_S0,stop_wanderR},
  {false,Process_Avoid_S0,Process_Avoid_S2,-1},
  {false,Process_Avoid_S0,Process_Avoid_S3,-1},
  {false,Process_Avoid_S2,Process_Avoid_S1,start_wanderR},
  {false,Process_Avoid_S3,Process_Avoid_S1,start_wanderR},
  {false,Process_Wander_S1,Process_Wander_S0,-1},
  {true,Process_Wander_S0,Process_Wander_S1,-1},
  {true,Process_Wander_S0,Process_Wander_S2,-1},
  {false,Process_Wander_S2,Process_Wander_S0,-1},
  {false,Process_Wander_S2,Process_Wander_stop,stop_wanderS},
  {false,Process_Wander_S1,Process_Wander_stop,stop_wanderS},
  {true,Process_Wander_S0,Process_Wander_stop,stop_wanderS},
  {false,Process_Wander_stop,Process_Wander_S0,start_wanderS}
};

/**
 * Location list
 */
loc_t LOC[NB_TRANS+NB_LOC] = {
  1,2,tid_drive_backward/*S0*/,
  0,tid_task_BumperPoller/*S1*/,
  3,tid_turn_left/*S2*/,
  4,tid_turn_right/*S3*/,
  tid_task_Music/*state_Music*/,
  6,7,11,tid_drive_forward/*S0*/,
  5,10,tid_turn_right/*S1*/,
  8,9,tid_turn_left/*S2*/,
  12,tid_NOP/*stop*/
};

/**
 * Task bodies
 */
//
//
int task_BumperPoller() {
  TASK_BEGIN(task_BumperPoller)
#include ".\bumperPoller.c"
  TASK_END(task_BumperPoller)
}

//
//
int drive_backward() {
  TASK_BEGIN(drive_backward)
#include ".\drive_backward.c"
  TASK_END(drive_backward)
}
```

```
//
//
int turn_left() {
  TASK_BEGIN(turn_left)
#include ".\turn_left.c"
  TASK_END(turn_left)
}

//
//
int turn_right() {
  TASK_BEGIN(turn_right)
#include ".\turn_right.c"
  TASK_END(turn_right)
}

//
//
int drive_forward() {
  TASK_BEGIN(drive_forward)
#include ".\drive_forward.c"
  TASK_END(drive_forward)
}

//
//
int task_Music() {
  TASK_BEGIN(task_Music)
#include ".\music.c"
  TASK_END(task_Music)
}

int main(int argc, char **argv) {

  Robot1_init();

  execi( &task_BumperPoller, 0, NULL, 5, SMALL_STACK_SIZE);
  execi( &drive_backward, 0, NULL, 4, SMALL_STACK_SIZE);
  execi( &turn_left, 0, NULL, 4, SMALL_STACK_SIZE);
  execi( &turn_right, 0, NULL, 4, SMALL_STACK_SIZE);
  execi( &drive_forward, 0, NULL, 4, SMALL_STACK_SIZE);
  execi( &task_Music, 0, NULL, 3, SMALL_STACK_SIZE);
/*
 * Reset clock variables
 */
  setClock(Process_Avoid_c2,0);
  setClock(Process_Wander_c1,0);


  execi( &controller, 0, NULL, PRIO_HIGHEST, SMALL_STACK_SIZE);

  cputw(MAKESIG);
  return 0;
}
```

## *b. Robot1.h*

```
/**
 * @file Robot1.h
 * @author TimesTool, Version 1.3 beta, April, 2007 (build 2461) <www.timestool.com>
 *
 * @brief Automatically generated file for system Robot1 defining constants.
 *
 */

#define NB_TASK 6
#define NB_TRANS 13
#define NB_PROC 3
#define NB_LOC 9
#define NB_CLOCKS 2
const char* iv0= "bump";
#define NB_VAR 1
const char** VARNAMES[NB_VAR] = {&iv0};
```

```
#define NB_CHAN 2
#define MAKESIG 0xA5FF
```

## c. Robot1_global.h

```
/**
 * @file Robot1_global.h
 * Add any #define:s needed by the tasks below.
 */
```

## d. Robot1_init.c

```
/**
 * @file Robot1_init.c
 * Add any initialisation code that must be executed before
 * the system starts in a function named Robot1_init().*/

void Robot1_init() {

}
```

## e. Robot1_init.h

```
/**
 * @file Robot1_init.h
 *
 * Add any initialisation code that must be executed before
 * the system starts in a function named Robot1_init().*/

void Robot1_init();
```

## f. brickos_hooks.h

```
/**
 * @file   brickos_hooks.h
 * @author Tobias Amnell <tobiasa@docs.uu.se>
 * @date   Thu Nov 21 11:56:51 2002
 *
 * @brief Defines actions to take when hooks are called. Currently
 * only used together with lnp logging.
 *
 * The hooks are called by the code in (@see brickos_kernel.c) when actions are
 * taken. The currently supported hooks are:
 * - Transition taken
 * - Task released
 * - Taske begin
 * - Task end
 */

#ifdef LNP_LOGGING

#include "brickos_logging.h"
/**
 * Hook called when transition was taken
 */
#define TRANS_TAKEN_HOOK( trn ) (void)addDataI( LOG_TRANS, trn )

/**
 * Hook called when task was released
 */
#define TASK_RELEASED_HOOK( tid ) (void)addDataI( LOG_TASK_RELEASE, tid)

/**
 * Hook called when task begins executing
 */
#define TASK_BEGIN_HOOK( tid ) (void)addDataI( LOG_TASK_BEGIN, tid)

/**
```

```
 * Hook called when task begins executing
 */
#define TASK_END_HOOK( tid ) (void)addDataI( LOG_TASK_END, tid)

#else // No-action hooks

#define TRANS_TAKEN_HOOK( trn )
#define TASK_RELEASED_HOOK( tid )
#define TASK_BEGIN_HOOK( tid )
#define TASK_END_HOOK( tid )

#endif // LNP_LOGGING
```

## g. brickos_interface.h

```
/* Comments in Doxygen format (www.doxygen.org) */
/**
 * @file brickos_interface.h
 * @author Tobias Amnell <tobias.amnell@docs.uu.se>
 * @date 23 May 2002
 *
 * @brief Define interface to hybrid automata kernel/engine.
 */

#ifndef BRICKOS_INTERFACE_H
#define BRICKOS_INTERFACE_H

#include <unistd.h>
#include <dsensor.h>
#include <dmotor.h>
#include <sys/tm.h>
#include <conio.h>
#include <dsound.h>
#include <semaphore.h>

#include "brickos_system.h"

#ifdef LNP_LOGGING
#include "brickos_logging.h"
#endif

/*
 * API in brickos_kernel
 */

wakeup_t check_trans( wakeup_t data );
wakeup_t task_release(wakeup_t data);
int controller();

#endif /* BRICKOS_INTERFACE_H */
```

## h. brickos_kernel.c

```
/**
 * @file   brickos_kernel.c
 * @author Tobias Amnell <tobias.amnell@docs.uu.se>
 * @date   Thu Mar 28 14:28:23 2002
 *
 * @brief The kernel functions that executes the controller.
 */

#include "brickos_interface.h"

extern int chanusage[];
extern trans_t trans[];
extern loc_t loc[];
extern int eval_guard(int);
extern void assign(int);
//extern bool pc[];

#if NB_TASK>0
extern char release_list[NB_TASK];
```

```
#endif /* NB_TASK>0 */

#ifndef tid_offset
/**
 * Offset to separate transition ids from task ids. Transitions are
 * between 1 and tid_offset and tasks are between tid_offset and MAX_TRANS.
 */
#define tid_offset 200
#endif /* tid_offset */

#ifndef recv_channels
/**
 * Receive channels have id larger or equal to recv_channel
 */
#define recv_channels 100
#endif /* recv_channels */

/**
 * Release task with id @a tid.  Copy variables used by task to local
 * store and increase task element in release list.
 */
#if (NB_TASK>0)
#define releaseTask( tid ) do{ \
  release_list[ tid ]++; \
} while(0)
#else
#define releaseTask( tid ) do{ } while(0)
#endif /* NB_TASK>0 */

/*
 * Private functions
 */
#if NB_CHAN>0
static int check_synch( unsigned char sync );
#endif

static void clear_and_set ( unsigned char sync );

/**
 * Deterimines if it is possible to synchronise on a channel.
 * N.B. Transition id 0 is not allowed, since 0 indicates that no
 * synchronisation was possible.
 *
 * @param sync Channel id, i.e. an index into the chanusage array.
 *
 * @return 0 if synchronisation is not possible, transition id of the
 * other transition if it is.
 */
#if NB_CHAN>0
static int check_synch( unsigned char sync ) {
  while( chanusage[sync] < NB_TRANS ) {
    if( (trans[chanusage[sync]].active==true) &&
        eval_guard( chanusage[sync] ) )
      return chanusage[sync];
    sync++;
  }
  return false;
}
#endif

/**
 * Update the list of active transitions when a transition is taken,
 * and release task of target location.
 *
 * @param trn The transition taken
 */
static void clear_and_set ( unsigned char trn ) {
  int jj;
  // Clear outgoing transition from source location
  jj=trans[trn].from;
  do {
    trans[loc[jj]].active=0;
  } while(loc[++jj]<tid_offset);
  // Set outgoing transition from target location
  jj=trans[trn].to;
  while ( loc[jj]<tid_offset ) {
    trans[loc[jj]].active=1;
```

```
    jj++;
  }
  // Release task of target location
  releaseTask(loc[jj]-tid_offset);

  TASK_RELEASED_HOOK(loc[jj]-tid_offset);
}

/**
 * Is a channel @a id a sending channel?
 */
#define IS_SEND( id ) (id < recv_channels )

/**
 * Check if any of the active transitions are enabled, and if so take
 * it. Will continue until a stable state (no more enabled
 * transitions) is reached.
 *
 * @param data Unused (should be null).
 * @return false when in stable state
 */
wakeup_t check_trans( wakeup_t data ) {
  int trn;
#if NB_CHAN>0
  int compl_trn;
#endif /* NB_CHAN>0 */
  for(trn=0; trn<NB_TRANS; trn++) {
    if( trans[trn].active ) {
      if( eval_guard(trn) ) {
#if NB_CHAN>0
        if( trans[trn].sync > -1 ) {
          if( (compl_trn =
              check_synch( trans[trn].sync )) ) {
            if( IS_SEND(trans[trn].sync) ) {
              assign( trn );
              assign( compl_trn );

              TRANS_TAKEN_HOOK(trans[trn].to);
              TRANS_TAKEN_HOOK(trans[compl_trn].to);

            } else {
              assign( compl_trn );
              assign( trn );

              TRANS_TAKEN_HOOK(trans[compl_trn].to);
              TRANS_TAKEN_HOOK(trans[trn].to);

            }
            clear_and_set( trn );
            clear_and_set( compl_trn );
            trn=-1;
          }
        } else
#endif /* NB_CHAN>0 */
        {
          assign( trn );
          TRANS_TAKEN_HOOK(trans[trn].to);
          clear_and_set( trn );
          trn=-1;
        }
      }
    }
  }
  return false;
}

/**
 * Wake-up function for threads (tasks).
 * @param data Unused (should be null)
 */
#if NB_TASK>0
/* Replaced by preceedence implementation */
/* wakeup_t task_release(wakeup_t data) { */
/*     return release_list[data]; */
/* } */
#endif /* NB_TASK */
```

```
/**
 * Thread body for automata controller
 * @return 0 (allways)
 */
int controller() {
  wait_event( &check_trans, 0);
  return 0;
}
```

## i. brickos_system.h

```
/* Comments in Doxygen format (www.doxygen.org) */
/**
  * @file   brickos_system.h
  * @author Tobias Amnell <tobias.amnell@docs.uu.se>
  * @date   Mon Apr 22 10:55:25 2002
  *
  * @brief  Definitions of types and macros used by generated code.
  *
  * Define types and macros used by the atomaticatly generated code
  * from TimesTool.
  */

#ifndef BRICKOS_SYSTEM_H
#define BRICKOS_SYSTEM_H

#include "brickos_hooks.h"

/**
 * Read clock value of @a clname .
 */
#define rdClock(clname) (sys_time - clock_##clname)

/**
 * Set clock with id @a clname  to value @a value.
 */
#define setClock(clname, value) clock_##clname = sys_time-value

/**
 * Copy clock values with id @a clname2 to @a clname .
 */
#define cpClock(clname, clname2) clock_##clname = clock_##clname2

/**
 * Transition information
 */
typedef struct s_TransitionType trans_t;

/**
 * Location list type
 */
typedef int loc_t;

/**
 * Define bool type
 */
typedef int bool;

/**
 * Transition table element
 */
struct s_TransitionType {
  char active;      /**< Active transition */
  char from;        /**< Source location */
  char to;          /**< Target location id */
  signed char sync; /**< Syncronisation id */
};

#ifndef true
#define true (1==1)
#endif

#ifndef false
#define false (0==1)
#endif
```

```
#define SMALL_STACK_SIZE 128

#define TASK_BEGIN( taskname ) while(true) {\
    do { \
    wait_event( &task_release, tid_##taskname - tid_offset );\
    TASK_BEGIN_HOOK( tid_##taskname - tid_offset );\
    release_list[ tid_##taskname - tid_offset]--; }while(false); {

#define TASK_END( taskname ) do {\
    wait_event( &task_complete, tid_##taskname - tid_offset );\
    TASK_END_HOOK( tid_##taskname - tid_offset );\
    } while(false); }}\
    return 0;

#define CONTROLLER( AUT ) int AUT##_controller() {\
    while( true ) { \
        wait_event(& AUT##_events, 0);\
    }\
}

#endif /* BRICKOS_SYSTEM_H */
```