Proiectarea Sistemelor Software Complexe

Curs 12 –Proiectarea Modulelor Software de Timp Real Utilizând HTL și Exotasks-HTL

12.1 Context

În ultimele două decenii aplicaţiile de timp real au cunoscut o dezvoltareputernică, astfel în prezent ele pot fi întâlnite în numeroase domenii: de lacontrolul direcţiei pentru un autoturism, la controlul unui avion şi de la consolelede jocuri video, la sistemele informatice care asigură disponibilitatea informaţiilorbursiere în timp real. Creşterea în complexitate a aplicaţiilor de timp real adeterminat, cum era şi firesc, evoluţia tehnicilor de programare folosite pentrudezvoltarea acestor aplicaţii. Astfel, dacă la început aplicaţiile de timp real eraurealizate în limbaje secvenţiale (în cel mai bun caz de nivel mediu),comportamentul temporal fiind dependent de timpul de execuţie al fiecăreiinstrucţiuni, s-a ajuns ca în prezent funcţionalitatea şi comportamentul temporalsă fie exprimate în limbaje distincte care asigură portabilitatea unei aplicaţii nudoar din punct de vedere funcţional ci şi din punctul de vedere alcomportamentului temporal.

O categorie importantă de aplicaţii de timp real este reprezentată de aplicaţiile de control de timp real. Aceste aplicaţii pot conţine atât taskuri periodicecât şi taskuri sporadice şi aperiodice. În continuare accentul se pune peaplicaţiile de control care conţin doar taskuri periodice. O aplicaţie de controlconstă în general din repetarea următoarelor acţiuni: citirea unor senzori,realizarea unor prelucrări asupra datelor citite de la senzori şi trimiterea decomenzi care să asigure comportamentul dorit pentru procesul condus. Multe dinaplicaţiile de control constau din multiple moduri de operare, fiecare mod fiindfolosit pentru o anumită evoluţie a procesului condus. Astfel, este evident că oaplicaţie de control va conţine diferite comportamente temporale, câte unulpentru fiecare mod de operare. În aceste condiţii implementareacomportamentului temporal într-un limbaj de uz general (ex.: C, Java etc.) estefoarte dificil de realizat, în plus codul sursă rezultat va fi complicat, greu deîntreţinut şi aproape imposibil de a verifica faptul că respectă anumite cerinţe de timp real (ex.: este dispecerizabil pe o anumită platformă).

Până în prezent au fost dezvoltate patru modele de programare aaplicaţiilor în timp real: physical-execution-time (PET), bounded-execution-time(BET), zero-execution-time (ZET) şi logical-execution-time (LET). PET presupuneutilizarea unui limbaj secvenţial pentru programarea aplicaţiilor de timp real,comportamentul temporal al aplicaţiei bazându-se pe cunoaşterea exactă atimpului de execuţie pentru fiecare instrucţiune folosită. BET foloseşteprogramarea concurentă, astfel se folosesc limbaje de programare asociate cumecanisme de programare de timp real, incluse într-un executiv de timp real sauun sistem de operare. ZET presupune faptul ca hardware-rul este suficient de rapidastfel încât operaţiile se realizează instantaneu; majoritatea limbajelor bazate peZET permit folosirea verificării formale petru a verifica anumite proprietăţi. LETpresupune faptul că taskurile nu se execută instantaneu ca şi în cazul ZET, cifiecare task dispune de o anumită fereastră de timp delimitată clar de momentulîn care intrările taskului trebuie citite şi momentul in care ieşirile taskului trebuiescrise, chiar dacă execuţia

taskului se termină mai repede ieşirile nu se voractualiza decât la momentul la care acestea trebuie scrise.

Giotto este primul limbaj destinat exclusiv specificării comportamentuluitemporal al unei aplicaţii. Modelul de programare folosit în Giotto este LET. Odescriere Giotto constă dintr-o serie de moduri, care conţin unul sau mai multetaskuri periodice a căror perioadă este armonică cu perioada modului din care facparte. Într-o descriere Giotto modurile sunt compuse secvenţial. După Giotto aufost dezvoltate o serie de alte limbaje destinate specificării comportamentuluitemporal al unei aplicaţii, toate folosind ca şi model de programare modelul LET.Timing Definition Language (TDL) extinde limbajul Giotto cu noţiunea decompunere în paralel. Timed Multitasking (TM) şi xGiotto folosesc modelul LET,dar spre deosebire de Giotto sunt două limbaje "event-triggered" şi nu "timed-triggered"aşa cum este Giotto. Timing Specification Language (TSL) este un altlimbaj care extinde limbajul Giotto. TSL relaxează LET-ul unui task în sensul căacesta nu mai este definit implicit de perioada taskului aşa cum se întâmpla încazul lui Giotto, ci de perioada taskului, offsetul şi durata taskului. TSL introducede asemenea şi posibilitatea de comunicare directă intre taskurile care au aceeaşiperioadă. Hierarchical Timing Language (HTL) este cel mai nou limbaj carefoloseşte noţiunea de LET. HTL suportă compunerea în paralel la fel ca şi TDL,relaxează LET-ul unui task şi suportă comunicarea directă între taskurile care auaceeaşi perioadă, la fel ca şi TSL.În plus HTL introduce conceptul de rafinare alunui task şi noţiunea de comunicator.

Având în vedere creşterea constantă în complexitate a aplicaţiilor de timpreal, în ultimul deceniu s-a manifestat un interes deosebit pentru a face din Javaun limbaj care să poată fi folosit pentru programarea aplicaţiilor de timp real. Pânăîn 2000 când au fost publicate specificaţiile de timp real pentru Java (RTSJ) nuexista o direcţie clară în ceea ce priveşte cercetarea din acest domeniu. RTSJ afixat următoarele obiective: compatibilitatea cu aplicaţiile implementate în Java,dar care nu sunt de timp real, respectarea principiului "Write Once, RunAnywhere", nu trebuie să fie extinsă sintaxa limbajului Java etc. Una din cele maiimportante surse de nedeterminism într-un program Java este reprezentată deGarbage Collector (GC).În prezent însă această sursă de nedeterminism a fostîndepărtată întrucât au fost propuse mai multe variante de GC care să satisfacăcerinţele unei aplicaţii de timp real. Totuşi s-a constatat faptul că dezvoltarea deaplicaţii care necesită rularea unor taskuri la frecvenţe de peste 1Khz, esteimposibil de realizat datorită întârzierilor introduse de procesul de colectare amemoriei. Pentru a rezolva această problemă au fost propuse o serie de soluţii,majoritatea soluțiilor se bazează pe ideea utilizării unor taskuri care folosesc ozonă de memorie privată şi care să nu acceseze zona globală de memorie a uneiaplicaţii, în acest fel taskurile vor putea întrerupe execuţia GC-ului, astfel căîntârzierile introduse de GC nu vor mai fi resimţite. Exotask este una dintre celemai noi tehnologii de programare a aplicaţiilor în timp real care foloseşte Java.

12.2 HTL

HTL este un limbaj care permite definirea comportamentului temporal al unei aplicaţii, funcţionalitatea aplicaţiei nu poate fi exprimată în HTL, ea trebuie implementată într-un alt limbaj (C, Java etc.). Un task în HTL reprezintă un bloc de cod secvenţial, care nu conţine elemente de sincronizare; taskurile HTL sunt preemptive. Modelul de programare folosit de HTL este modelul LET. Astfel orice task HTL este caracterizat de un timp logic de execuţie definit ca fiind intervalul de timp dintre cel mai târziu moment de timp la care taskul trebuie să citească o intrare şi cel mai devreme moment de timp la care taskul trebuie să scrie o ieşire. Între cele două momente de timp taskul trebuie să se execute. Execuția fizică a unui task nu se poate realiza în afara ferestrei LET, și anume taskul nu poate fi executat nici mai devreme, nici mai târziu decât această fereastră. Începutul ferestrei LET definește momentul de timp când taskul trebuie lansat în execuție, iar sfârșitul ferestrei marchează momentul de timp până la care

execuția taskului să se finalizeze. O caracteristică importantă a modelului de programare LET este acela că ieșirea unui task nu poate fi citită mai repede de sfârșitul ferestrei LET corespunzătoare taskului. În Fig. 10.1 este prezentat un task *t* care citește la momentele de timp *t1* si *t2* (*t1<t2*) intrările *i1* și respectiv *i2*, iar la momentele *t3* și *t4* (*t3<t4*) scrie ieșirile *o1* și *o2*, pentru acest task fereastra LET este reprezentată de intervalul de timp dintre momentul *t2* (când este citită ultima intrare) și respectiv momentul de timp t3 când trebuie actualizată prima ieșire. La momentul de timp *t2* taskul *t* trebuie să fie lansat în execuție, iar la momentul de timp *t3* execuția taskului trebuie să se finalizeze. Între aceste momente de timp taskul poate fi întrerupt de mai multe ori, acest lucru însă nu contează, ceea ce contează este ca până la momentul de timp *t3* taskul trebuie să se execute în întregime.

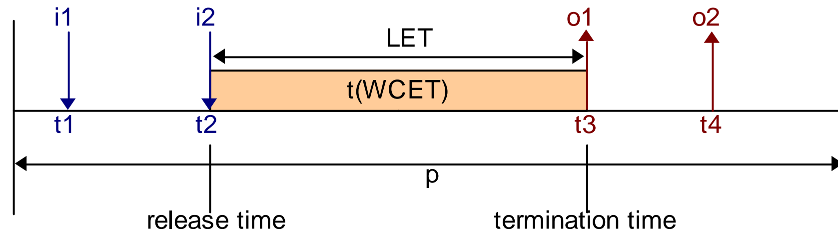

Fig. 10.1. Modelul de programare LET (logical execution time).

Folosirea modelului LET permite limbajului HTL să asigure compunerea secvenţială a două sau mai multe seturi de taskuri şi compunerea în paralel a două sau mai multe seturi de taskuri, fără ca aplicaţia să fie afectat în ceea ce privește comportamentul temporal. O altă caracteristică importantă a limbajului HTL este rafinarea, astfel în HTL pot fi definite taskuri abstracte şi taskuri concrete. Taskurile abstracte pot fi rafinate de alte taskuri abstracte sau concrete.În acest fel se poate crea o structură ierarhică de taskuri. Această structură ierarhică de taskuri este utilă mai ales atunci când trebuie verificat dacă aplicaţia este dispecerizabilă pe o anumită platformă, astfel nu trebuie verificate toate combinaţiile de taskuri care pot să apară, ci este suficientsă se analizeze combinaţiile între taskurile de pe primul nivel ierarhic, toate celelalte nivele respectând constrângerile impuse de acest nivel. HTL oferă şi suport pentru distribuirea unei aplicaţii pe mai multe echipamente hardware care pot comunica între ele.

12.2.1 Sintaxa Limbajului HTL

Principalele elemente care alcătuiesc o descriere HTL sunt reprezentate de taskuri şi comunicatori. Taskurile reprezintă funcţionalitatea aplicaţiei, în timp cecomunicatorii reprezintă un mecanism de comunicare între taskuri. Un task în HTLconstă dintr-un nume, care este folosit pentru a referi taskul în descrierea HTL, unnume de funcţie, care implementează funcţionalitatea taskului, un set de porturide intrare, un set de porturi de stare şi un set de porturi de ieşire. Doar porturilede intrare şi ieşire sunt accesibile din exterior, ele reprezentând interfaţa princare un task comunică cu alte taskuri; porturile de stare sunt accesibile doartaskului, ele fiind folosite pentru stocarea de informaţii care trebuie reţinute de lao execuţie la alta a respectivuluitask. Comunicatorii sunt variabile care au asociat untip de dată, dar care nu pot fi accesate decât la anumite momente de timp binespecificate. Un comunicator constă dintr-un nume, care este folosit pentru a refericomunicatorul în descrierea HTL, un tip de date, valoarea de iniţializare şiperioada, care defineşte momentele de timp în care un comunicator poate să fieaccesat (scris sau citit). Taskurile care au aceeaşi perioadă pot fi grupate într-unmod. Taskurile din două moduri diferite sunt compuse secvenţial. Un mod constădintr-un nume, care este folosit pentru a referi modul în descrierea HTL, operioadă, care defineşte perioada de execuţie a tuturor taskurilor dintr-un mod,un set de invocări de taskuri, care identifică taskurile invocate atunci cand moduleste activ şi un set de schimbări de mod (*mode switches*), care identifică tranziţiile posibile dintr-unmod. Taskurile invocate în

acelaşi mod pot comunica între ele fie direct fieindirect prin intermediul comunicatorilor. Dacă două taskuri comunică directatunci între ele se va stabili o relaţie de dependenţă, în sensul că taskul careciteşte ieşirea altui task nu se poate executa înaintea acestuia. Unul sau maimulte moduri formează un modul. Modurile din acelaşi modul sunt compusesecvenţial în timp ce modurile din module diferite sunt compuse în paralel. Unmodul constă dintr-un nume, care este folosit pentru a referi modulul îndescrierea HTL, un nume de mod, care reprezintă modul care va fi activ atuncicând modulul este executat pentru prima data şi un set de moduri. Unul sau maimulte module formează un program. Un program conţine un nume, care estefolosit pentru a referi programul în descrierea HTL, un set de module şi un set decomunicatori, care pot fi folosiţi pentru a comunica între taskurile invocate înmodulele din program. Programul reprezintă elementul prin intermediul căruia sepoate construi structura ierarhică a unei descrieri HTL. O descriere HTL poate săconţină oricâte programe, dar nu poate să conţină mai mult de un programrădăcină. Programul rădăcină în cazul unei descrieri HTL definește comportamentul temporal abstract al unui program HTL, care este rafinat în comportamente temporale concrete de restul programelor definite în aceeași specificație HTL.

În Fig. 10.2 este prezentat un exemplu de descriere HTL care specifică comportamentul temporal pentru o aplicație ce realizează incrementarea/decrementarea unui contor. Cuvintele cheie sunt marcate cu albastru. Programul rădăcină este *P_inc_dec*, acesta constă din două module care se execută în paralel. Modulul *M_read_write* realizează partea de comunicare cu exteriorul; acest modul conține doar taskuri concrete. Modulul *M_inc_dec* specifică comportamentul temporal pentru funcționalitatea de incrementare, respectiv decrementare. Primul aspect care trebuie remarcat vis-a-vis de cele două operații este acela că ele se execută odată la fiecare secundă. Doar una din operațiile de incrementare și decrementare se execută la un moment dat (ele fiind invocate în moduri diferite care pot comuta intre ele). Execuția programului începe cu operația de incrementare. Incrementarea se realizează până când contorul a ajuns la valoarea 100, moment în care se comută la modul de decrementare. Decrementarea se realizează până când contorul a ajuns la valoarea zero, moment în care se comută înapoi la modul de incrementare. Operația de incrementare specificată în modulul *M_inc_dec* este una abstractă (task-ul *t_inc* nu are specificată funcționalitate), ea fiind rafinată de programul P_inc, care definește trei tipuri de incrementări, și anume: incrementare cu 1, incrementare cu 5 și incrementare cu 10.

```
program P_inc_dec{
communicator
    c_int counter period 100 init c_zero;
    c_int ref period 100 init c_zero;

module M_read_write start m_read_write{

task t_read input() state() output(c_int p_counter) function f_read;
task t_ref input() state() output(c_int p_ref) function f_ref;
task t_write input(c_int p_counter) state() output() function f_write;

mode m_read_write period 1000{
invoke t_read input() output((counter,1));
invoke t_ref input() output((ref,1));
invoke t_write input((counter,2)) output();
    }
  }

module M_inc_dec start m_inc{

task t_inc input(c_int p_counter_in) state() output(c_int p_counter_out);
task t_dec input(c_int p_counter_in) state() output(c_int p_counter_out) function f_dec;

mode m_inc period 1000 program P_INC{
invoke t_inc input((counter,1)) output((counter,2));
switch(inc_to_dec(counter)) m_dec;
    }
```

```
mode m_dec period 1000{
invoke t_dec input((counter,1)) output((counter,2));
switch(inc_to_dec(counter)) m_inc;
    }
  }
}

program P_inc{

module M_inc start m_inc1{

task t_inc1 input(c_int p_counter_in) state() output(c_int p_counter_out) function f_inc1;
task t_inc5 input(c_int p_counter_in) state() output(c_int p_counter_out) function f_inc5;
task t_inc10 input(c_int p_counter_in) state() output(c_int p_counter_out) function f_inc10;

mode m_inc1 period 1000{
invoke t_inc1 input((counter,1)) output((counter,2)) parent t_inc;
switch(inc1_to_inc5(counter)) m_inc5;
    }

mode m_inc5 period 1000{
invoke t_inc5 input((counter,1)) output((counter,2)) parent t_inc;
switch(inc5_to_inc10(counter)) m_inc10;
    }

mode m_inc10 period 1000{
invoke t_inc10 input((counter,1)) output((counter,2)) parent t_inc;
    }
  }
}
```

Fig. 10.2. Exemplu de descriere HTL.

12.2.2 Execuția Unei Descrieri HTL

La fel ca şi în cazul limbajului Giotto, descrierile HTL nu sunt compilatedirect în cod maşină ci în aşa numitul E code, care este interpretat de o maşinăvirtuală numită E machine. Din punct de vedere structural maşinavirtuală E machine constă dintr-o listă de instrucţiuni E code, care reprezintăprogramul E code ce trebuie interpretat, registrul PC (program counter), careconţine indexul instrucţiunii ce trebuie interpretată din lista de instrucţiuni, listataskurilor care au fost eliberate spre a fi executate , o coadă de triggere, o tabelăde taskuri, o tabelă de drivere şi o tabelă de condiţii şi interpretorul de E code,care interacţionează cu toate celelalte părţi componente ale E machine. Setul deinstrucţiuni E code conţine şase instrucţiuni:

- *call* – permite invocarea unei funcţii driver;
- *release*– eliberează un task spre a fi dispecerizat de un dispecer EDF;
- *future*– adaugă un nou trigger în coada de triggere, un trigger este oasociere între un event şi o adresă E code, astfel când eventul de caredepinde trigger-ul devine activ, maşina virtuală va executa blocul de Ecode care începe la adresa specificată în trigger; un event în HTL este reprezentat fie de scurgerea unui interval de timp fie de terminarea unui task;
- *jump*– realizează un salt necondiţionat la adresa E code specificată ca şiparametru;
- *if* – realizează un salt condiţionat la adresa E code specificată ca şiparametru;
- *return* – opreşte interpretarea programului E code şi aduce maşinavirtuală într-o stare de aşteptare, din care maşina va ieşi doar atuncicând se activează un trigger în coada de triggere.

Întrucât varianta iniţială de E machine (prezentată mai sus și dezvoltată inițial pentru Giotto) nu a fost proiectată pentru asuporta structura ierarhică a unei descrieri HTL, aceasta a fost extinsă, astfel noua maşină virtuală (HE machine) esteasemănătoare ca structură cu cea iniţială, dar prezintă următoarele diferenţe:conţine trei cozi de triggere, patru registre care pot stoca referinţe către triggereşi care sunt folosite pentru a realiza diverse operaţii cu triggere şi două stive: unade adrese şi una de triggere. Setul de instrucţiuni a fost extins de la şaseinstrucţiuni la nouăsprezece instrucţiuni. Setul extins de instrucţiuni E code a fostdenumit HE code. Au fost păstrate instrucţiunile iniţiale la care s-a adăugat oinstrucţiune care realizează un apel de subrutină, un set de zece instrucţiuni carerealizează operaţii cu triggere, iar instrucţiunea future a fost înlocuită cu treiinstrucţiuni similare, fiecare specifică unei cozi de triggere. De asemenea a fostextins conceptul de trigger, astfel pe lângă informaţia despre evenimentul caredetermină activarea triggerului şi adresa asociată cu triggerul.Un trigger mai areasociat un trigger părinte şi o listă de triggere copii, acest lucru permitând refacerea structurii ierarhice a unei descrieri HTL în timpul rulării.

12.2.3 Proiectarea Aplicațiilor de Control utilizând HTL

Un avantaj important al folosirii limbajului HTL pentru dezvoltarea modulelor unui sistem software complex care necesită constrângeri de timp real severe este acela că, acest limbaj permite separarea funcționalității de comportamentul temporal, portabilitatea comportamentului temporal și separarea specificațiilor temporale de caracteristicile platformei/platformelor pe care urmează să fie executată aplicația. Separarea funcționalității de comportamentul temporal se referă la faptul că în HTL nu se descrie funcționalitatea unei aplicații de timp real ci se poate doar specifica comportamentul temporal al acesteia, funcționalitatea fiind implementată într-un limbaj clasic (C, C++, Java etc.). Portabilitatea comportamentului temporal al unei descrieri HTL este garantată de faptul că, descrierile HTL nu sunt compilate în cod mașină ci într-un program HE code, care poate fi executat pe orice platformă pentru care există o implementare a mașinii virtuale HE machine.

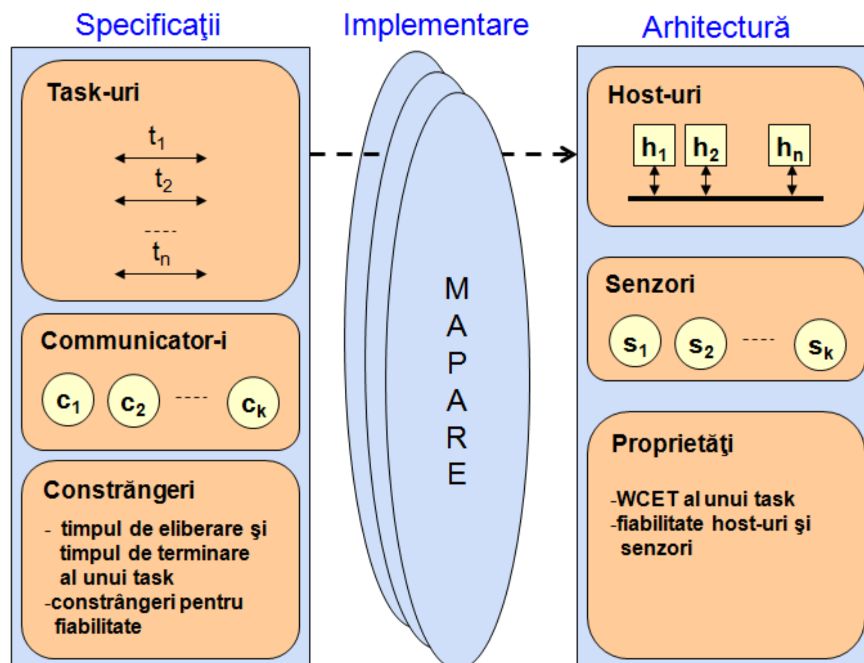

Fig. 10.3. Principiul separării specificațiilor de arhitectură.

În ceea ce privește separarea specificațiilor temporale de platforma pentru care este implementată aplicația, limbajul HTL aplică principiul cunoscut sub denumirea de *platform-based design*. În Fig. 10.3 este ilustrat grafic modul în care în HTL se realizează separarea specificațiilor temporale de platforma pentru care se realizează implementarea. Astfel, o descriere HTL specifică un set de taskuri, un set de comunicatori și un set de constrângeri (de ex.: timpul de eliberare și terminare al unui task sau constrângeri în ceea ce privește fiabilitatea). O platformă (arhitectură hardware) pentru o descriere HTL este reprezentată de un set de echipamente de calcul conectate în rețea (hosts), un set de senzori care extrag informații din procesul condus și un set de proprietăți (de ex.: timpul cel mai defavorabil de execuție al uni task (WCET) pentru un anumit host sau fiabilitatea unui host și a unui senzor). O implementare în cazul HTL este reprezentată de o mapare de taskuri specificate de descrierea HTL la echipamentele de calcul disponibile pentru platforma pentru care se realizează implementarea. Această mapare trebuie să garanteze faptul că sunt respectate toate constrângerile specificate în descrierea HTL; verificarea se face pe baza proprietăților platformei. De exemplu, trebuie să se verifice faptul ca pentru maparea realizată programul este dispecerizabil utilizând un dispecer EDF.Pentru acesta se va realiza o analiza în care pe baza WCET al unui task pentru echipamentul pe care el va fi executat se poate determina dacă implementarea este dispecerizabilă.

12.3 Exotask-HTL

În ultimii zece ani s-a manifestat un interes deosebit în ceea ce priveştefolosirea limbajului Java pentru programarea aplicaţiilor de timp real. Apariţia unoralgoritmi predictibili de colectare a memoriei a făcut posibilă folosirea limbajuluiJava pentru dezvoltarea aplicaţiilor de timp real. Au fost propuse mai multemodele de programare care folosesc Java pentru dezvoltarea aplicaţiilor de timpreal. Unul dintre cele mai recente modele de programare, este cel propus desistemul Exotask. Dezvoltarea unei aplicaţii de control utilizând Exotask constădin definirea unui graf Exotask şi din implementarea funcţionalităţii aplicaţiei înJava. Un graf Exotask specifică comportamentul temporal al aplicaţiei. Nodurileunui graf Exotask reprezintă taskuri, iar arcele reprezintă căile de comunicaredintre taskuri. Atât nodurilor cât şi arcelor li se pot asocia informaţii ce descriucomportamentul temporal al taskurilor, respectiv al căilor de comunicare. Toateelementele care pot fi folosite pentru a descrie comportamentul temporal al unuitask sau al unei căi de comunicare reprezintă gramatica de specificare acomportamentului temporal. Deşi sistemul Exotask este distribuit împreună cu ogramatică de specificare a comportamentului temporal ce permite compunereasecvenţială a unor seturi de taskuri, el nu este limitat doar la folosirea acesteigramatici întrucât Exotask permite definirea de noi astfel de gramatici. O astfel de extensie esteExotask-HTL, care definește o gramatică de specificarea comportamentului temporal ce permite utilizarea sintaxei HTL pentru a specificacomportamentul temporal al unui graf Exotask. In Fig. 10.4 este prezentat un exemplu de graf Exotask (captura a fost realizată din editorul de grafuri Exotask disponibil ca și plug-in pentru Eclipse).

Gramatica HTL pentru Exotask conţinetoate elementele specifice sintaxei HTL: programe, module, moduri, schimbări demoduri, taskuri şi comunicatorii. Dispecerul care face posibilă executarea unei aplicaţii de timp real dezvoltată în Exotask-HTL conţine atât o implementare în Java acompilatorului HTL cât şi o implementare în Java a maşinii virtuale HE machine.Când o aplicaţie Exotask este rulată, mai întâi are loc o compilare a grafuluiExotask într-un program HE code reprezentat în Java, după care programul HEcode este interpretat de maşina virtuală. Dispecerul care a fost dezvoltat pentru apermite rularea de aplicaţii Exotask al căror comportament temporal estespecificat în HTL este un dispecer multi-threading, spre deosebire de cel care estedistribuit împreună cu platforma Exotask, care este single-threading.

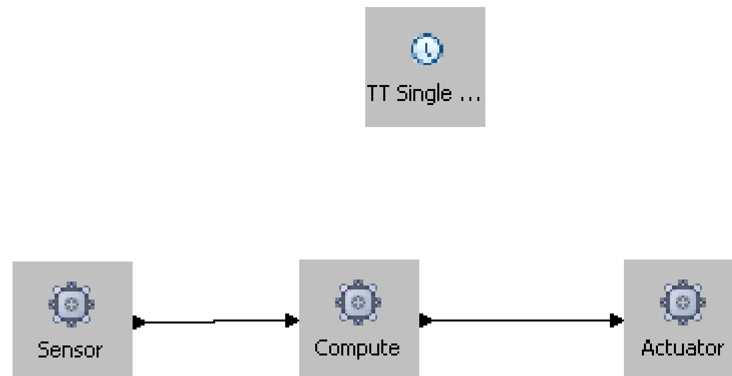

Fig. 10.4.Exemplu de graf Exotask.

Bibliografie

[1] Daniel Iercan. Contributions to the Development of Real-Time Programming Techniques and Technologies.Teză de doctorat. Editura Politehnica. 2008.

[2] C.M. Kirsch and R. Sengupta.The Evolution of Real-Time Programming.In*Handbook of Real-Time and Embedded Systems*.Chapman and Hall/CRC, 2007.

[3] A Hierarchical Coordination Language for Interacting Real-Time Tasks. A. Ghosal, T. A. Henzinger, D. Iercan, C. M.Kirsch, Al. Sangiovanni-Vincentelli. EMSOFT 2006, 6th ACM & IEEE Conference on Embedded Software, 22-25 October, 2006, Seoul, Korea, ISBN:1-59593-542-8, Pages: 132 – 141 [ACM Digital Library]

[4]Low-latency time-portable real-time programming with Exotasks. J. Auerbach, D.F. Bacon, D. Iercan, C.M. Kirsch, V.T.Rajan, H.Röck, R. Trummer. /ACM Transactions on Embedded Computing Systems (TECS), January 2009, /ISSN:1539-9087, Volume 8, Issue 2