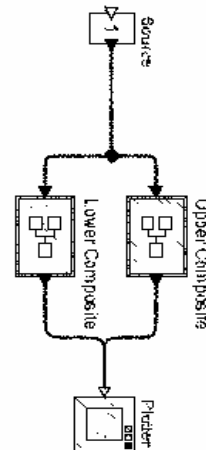


SZÁMÍTÁSI MODELLEK

*A legáltalánosabb számítási modellek
és a Ptolemy eszköz
bemutatása*

*Völgyesi Péter
volgyesi@mit.bme.hu*



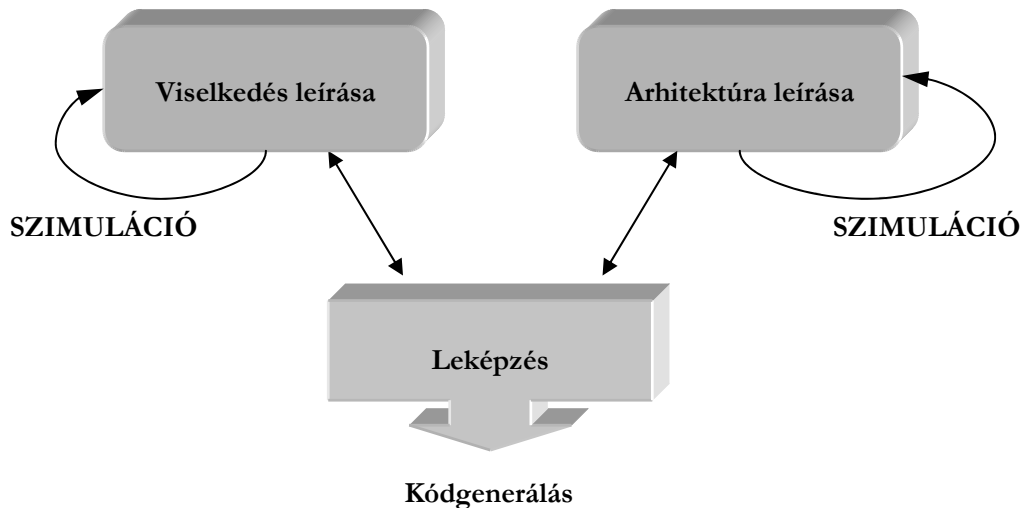
A beágyazott rendszerek jelentős része zárt, erőforrás korlátozott hardverrel rendelkezik. A szoftverfejlesztés szempontjából ez számos - a hagyományos számítógépes rendszereknél nem jelentkező – problémát vet fel. A nyomkövetés (debug), a rendszer belső állapotához való hozzáférés sokkal körülményesebb, pusztán a megfelelő beviteli és kimeneti/megjelenítési eszközök hiánya miatt is. Az előzetes szimuláció és precíz tervezés segítségével a legtöbb hiba még időben kiszűrhető.

Olyan feladatoknál, ahol a „beágyazott rendszer” nem egyetlen eszközből, hanem egymással kommunikáló autonóm elemekből áll (pl. szenzor-hálózatok) a végső rendszer tesztelése még körülményesebb, egy hibás viselkedés determinisztikus megismétlése szinte lehetetlen.

Ezen rendszereken futó alkalmazások végrehajtási logikája (esemény alapúság) és egyéb – a szoftver tervezése során figyelembe veendő – kritériumok (pl. energiafogyasztás) szintén a hagyományos szoftverfejlesztéstől való eltérést mutatják.

A fenti problémák egyik legfontosabb – és nem meglepő – tanulsága, hogy a beágyazott rendszerek tervezése során sokkal nagyobb hangsúlyt kell fektetni a befogadó fizikai környezet megértésére és a rendszer „beágyazására” ebbe a fizikai környezetbe. Célszerűnek tűnhet a fizikai környezetet és az elkészítendő rendszert együtt modellezni. Így egyrészt pontosabb képet kaphatunk a környezet és rendszerünk kapcsolatáról, és sok esetben rendszerünk belső modelljét is célszerűbben alakíthatjuk ki.

Számos modellezési eszköz áll ma már rendelkezésünkre, melyek támogatják fizikai rendszerek ill. számítógépes rendszerek (szoftver – esetleg hardver – architektúrák) modellezését. Jóval kevesebb azon modellezési eszközök száma, melyekkel ezt a két világot egyszerre lehet modellezni. A modellezési módszerek tárgyalása során a Berkeley Egyetemen kifejlesztett *Ptolemy* (<http://ptolemy.eecs.berkeley.edu/>) nevű eszközt fogjuk használni, mivel a köré szerveződött kutatócsapat – a Ptolemy szoftver elkészítése mellett – a beágyazott rendszerek modellezésének területén jelentős elméleti eredményeket is produkál.



1. ábra A beágyazott rendszer logikájának és architektúrájának modellezése

A modellezés hasznos lehet pusztán a feladat megértése és a tervezés szempontjából, a modellezési környezet azonban segíthet a feladat implementációs szakaszában is: a kód automatikus szintézise a modell alapján – bár korántsem triviális – számos előnnyel járhat. Az egyik legfontosabb előny, hogy így garantálható a formális modell és a szoftveres megvalósítás azonossága. Ezek a módszerek ma még gyerekcipőben járnak; az egyik legnagyobb feladatot a viselkedés formális leírásához használt fogalmak leképzése jelenti a konkrét hardver/szoftver platformra (1. ábra).

Számítási modellekről általában

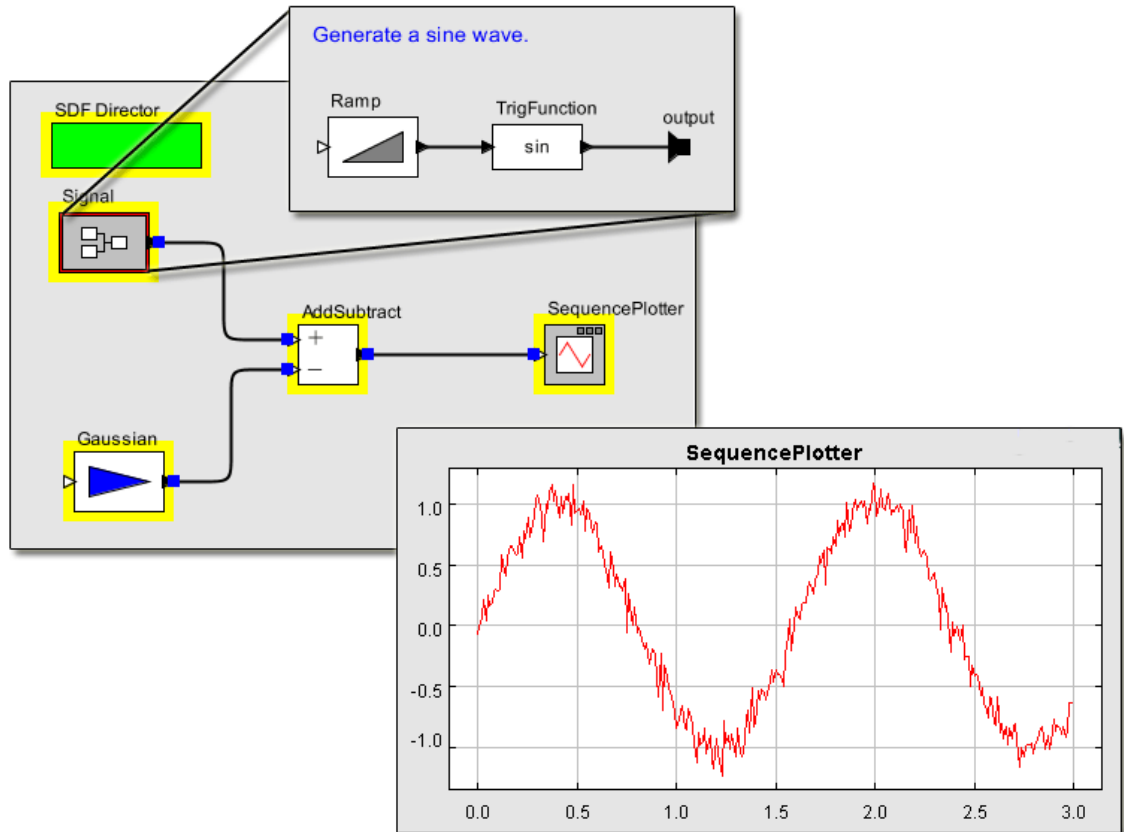
A fizikai világról alkotott ismereteink nem elegendőek ahhoz, hogy abszolút – minden szempontból hibátlan – modelleket alkothassunk róla. Ez sokszor nem is célunk, mivel az adott fizikai rendszer viselkedése csak bizonyos nézőpontból érdekel minket. A modellalkotásunk során felhasznált szabályok, jelölésrendszerünk, a modell végrehajtása (szimulációja) során használt lépések nagyon különbözőek lehetnek. A korábbi tanulmányok során is találkoztunk ezekkel „különböző modellezési megközelítésekkel”: a programozási paradigmák (gépi kód, klasszikus imperatív nyelvek, objektum orientált megközelítések, funkcionális és logikai nyelvek) ugyanarról a megoldandó feladatról alkotnak képet különböző szemszögből. A paradigmák sokszínűsége lehetővé teszi, hogy olyan leírást választhassunk, mellyel a legkönnyebben, legszemléletesebben leírható a feladat – és a megoldás.

A következőkben olyan leírási módszerekkel (*Models of Computation*) ismerkedünk meg, melyek alkalmasak lehetnek beágyazott rendszerek és környezetük modellezésére. Számos leírás ismerősnek tűnhet, mivel nagy részük nem korlátozódik a beágyazott rendszerek világára. Fontos azonban megértenünk, hogy mi teszi egyedivé az adott modellezési módszert, e nélkül ugyanis nem tudjuk, mikor érdemes az adott leírási formát választani.

A Ptolemy modellezési és szimulációs környezet

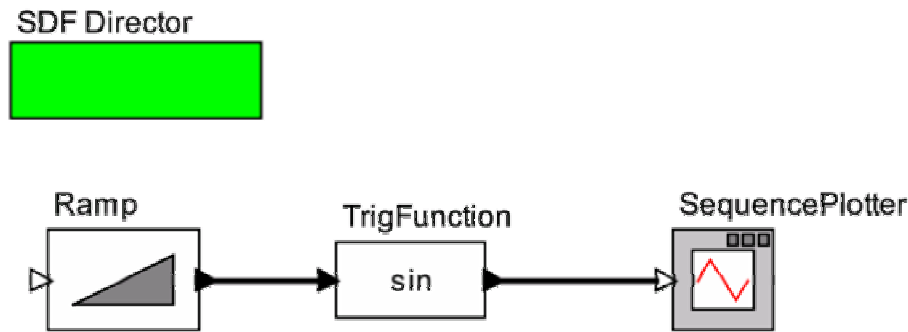
Mivel a modellezési nyelvek bemutatása során a Ptolemy eszközre hagyatkozom, elkerülhetetlen, hogy ne ismerkedjünk meg felületesen ezzel a szoftverrel. A környezet több oldalról biztosít hozzáférést modelljeinkhez: grafikus felületen keresztül szerkeszthetjük modelljeinket, ezekhez Java nyelven írt programokból hozzáférhetünk, valamint lehetőségünk van XML formában tárolni, szerkeszteni a modelladatbázist.

Az eszköz egyik erőssége, hogy a legelterjedtebb számítási modelleket ismeri. A felépített modelleket nem csak szintaktikailag képes kezelni és ellenőrizni, de képes azokat értelmezni, végrehajtani (szimulálni) is. A másik fontos tulajdonsága a szoftvernek, hogy támogatja – sőt bátorítja – a különböző számítási modellek együttes használatát, keverését egy projekten belül. A modell megfelelő részei így azon a nyelven írhatók le, mely a legközelebb áll az adott területhez (pl. fizikai rendszert egy folytonos idejű nyelven, míg a beágyazott rendszert egy diszkrét idejű számítási modellben írhatunk le). Bár az eszköz támogatja a modellekből történő kódszintézist, ez a lehetőség még nem kiforrott, így nem fogunk foglalkozni vele.



2. ábra A Ptolemy modellezési és szimulációs környezet

A modellek szinte kivétel nélkül hierarchikusak: adott szinten szereplő elemek részletesebb kifejtését adhatjuk meg az alsóbb szinteken. A legfontosabb építőelemek az úgynevezett *actor*-ok. Ezek vagy összetettek (ilyenkor fejthetjük ki a belső szerkezetet), vagy egyszerű elemek (ekkor egy Java osztály írja le az actor viselkedését). Az építőkövek ki- és bemeneti portokkal rendelkeznek, mely portokon keresztül az actorok között kapcsolatok (*relation*) létesíthetők. Minden modell rendelkezik egy ún. irányítóval (*director*), mely meghatározza, hogy milyen számítási modellt használunk. Ez az irányító tartalmazza azokat az – elsősorban végrehajtási – szabályokat, melyek az adott számítási modellre jellemzőek. Az egyik alapgondolata a Ptolemy projektnek, hogy az actorok nem feltétlenül leírás- (paradigma-) függők, vagyis függetlenül a végrehajtási logikától, bizonyos funkciók általánosan leírhatók. Sok esetben azonban bizonyos actor-director kombinációknak nincs értelme, ahogy azt később látni fogjuk. Ha egy hierarchikus modell különböző szintjein eltérő directorokat használunk, heterogén rendszereket kapunk.



3. ábra Egyszerű Ptolemy modell: szinusz jel generálása

A 3. ábra egy szinusz jelet előállító Ptolemy modellt mutat. A zöld doboz a director, mely jelen esetben egy szinkron adatfolyam számítási modellt valósít meg. A három actort portokon keresztül kötöttük össze. Az első actor egy folyamatosan növekvő értéket állít elő a kimenetén, melynek a szinusz értékét állítja elő a középső doboz, majd végül az így kiszámított értéket a plotter fog kirajzolni a szimulációs környezetben.

Számítási modellek bemutatása

Mielőtt rátérnénk a konkrét számítási modellek ismertetésére, tekintsük át azokat a legfontosabb szempontokat, amelyek alapján vizsgálni szeretnénk ezeket a nyelveket. Az egyes számítási modellek közötti különbségeket ezeken a szempontokon keresztül fogjuk látni a legszembetűnőbben.

Párhuzamos végrehajtás. Itt azt vizsgáljuk, hogy az adott számítási modell milyen mértékben támogatja a párhuzamos ill. elosztott megvalósítást. Például egy erősen szinkron rendszerben, ahol az egyes elemek (actorok) egymásra várnak a párhuzamosság szintje alacsony.

Determinisztikus viselkedés. A szimuláció szempontjából fontos kérdés, hogy pontosan ugyanúgy megismételhető-e újra és újra a szimuláció. Ez a szempont sok esetben összefügg az előzővel: minél magasabb a párhuzamosság/elosztottság foka a rendszerben, annál kevésbé várható, hogy pontosan reprodukálható viselkedést fog mutatni.

Az idő fogalma. Ez a kérdés több részből áll: foglalkozik-e a számítási modell – és a végrehajtás során a szimulációs környezet – valamilyen idő jellegű információval, ha igen, akkor ez hogyan viszonyul a valós (a szimuláción kívüli) időhöz, és végül elosztott vagy globális idővel (órákkal) operál-e az adott számítási modell.

Speciális végrehajtó egységek. Szó esett arról, hogy a Ptolemy környezetben nagyon sok actor számítási modelltől független. Bizonyos számítási modellek azonban igényelnek speciális actorokat is, melyek az adott számítási modell sajátosságait használják ki. Ezek a speciális végrehajtó elemek megismerése segíthet a számítási modell megértésében.

Ismerkedjünk meg most a legelterjedtebb számítási modellekkel. Szinte kivétel nélkül minden számítási modellhez példát adok. A jegyzet mellett ezek a példamodellek is letölthetők a tárgy web oldaláról, így pontosabban megvizsgálhatók és futtathatók bárki által a szimulációs környezetben.

Diszkrét események – Discrete Events

A diszkrét események számítási modellben a végrehajtó egységek a kimenetükön időbélyeggel ellátott eseményeket generálnak. Ezek az események egy közös (globális) listába kerülnek idő szerint rendezett módon. A végrehajtás során – mely iterációkra épül - a director előrelépteti a globális órát (nem folytonosan) a lista első tokenjének időbélyegére, majd a tokent a megfelelő egység bemenetére teszi. Egy iterációs lépés végrehajtási atomi abból a szempontból, hogy a lépés végrehajtása alatt a globális óra be van fagyasztva. A lépés végrehajtása során az összes olyan esemény feldolgozásra kerül, melynek azonos az időbélyege a listában szereplő első tokenével. Egy actor többször is meghívódhat az atomi lépés során.

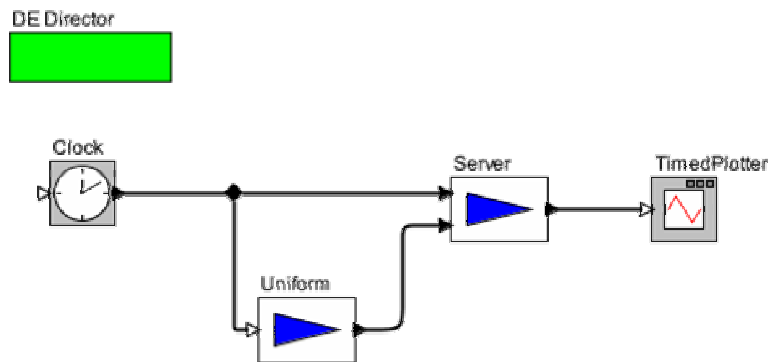
Ezt a számítási modellt előszeretettel használják beágyazott alkalmazások szimulációjára, a TinyOS rendszer szimulációs eszköze is ezt a számítási modellt használja.

Párhuzamos végrehajtás. A párhuzamosság foka alacsony, mivel az események szigorúan rendezettek, az ütemezési logika szekvenciális. A számítási modell rendkívül jól megvalósítható egyetlen végrehajtási szál segítségével. A párhuzamosság érzetét az iterációs lépésben befagyasztott óra kelti.

Determinisztikus viselkedés. Teljesen determinisztikus viselkedést mutat a rendszer, ezért is szeretik szimulációs eszközökben.

Az idő fogalma. A számítási modell erősen épít az idő fogalmára, mely globális a rendszerben, és nem folytonosan változik. Ennek előnye, hogy a szimuláció futtatásához szükséges (valós) idő nem a szimulálandó idő, hanem a szimulálandó események számával arányos - így hatalmas üresjáratú időszakok ugorhatók át szimuláció közben. Az ára ennek, hogy a szimuláció és a külvilág között nem teremthető elő kapcsolat (pl. futás alatti adatbevitel), mivel a belső időnek semmi köze a valós időhöz és annak múlásához.

Speciális végrehajtó egységek. Mivel az általános actorok tokenjeiket 0 időkéstelletéssel állítják elő, szükség van olyan elemekre, melyek képesek a kimenetükön késleltetést okozni.

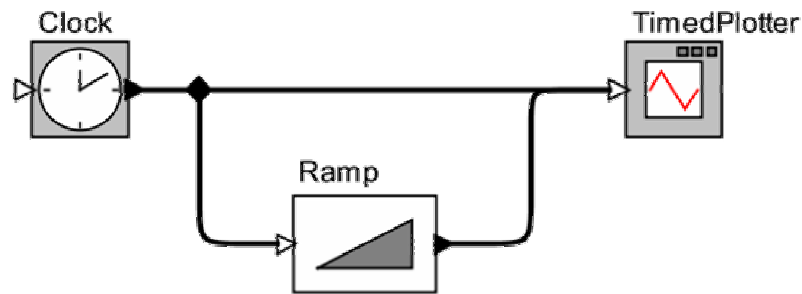


4. ábra Kiszolgálási modell

A 4. ábra egy egyszerű kiszolgáló modelljét mutatja. A Server bemenetére érkező felső inputot a kimenetén késleltetve teszi ki a kiszolgáló. A késleltetés idejét a Uniform nevű egyenletes eloszlás szerinti véletlen számot generáló egység határozza meg a Server alsó bemenetén.

Az iterációs lépésben az azonos időbélyegű eseményeket egymás után hajtja végre a szimuláció. Az 5. ábra egy olyan konfigurációt mutat, ahol „intuitív” módon el tudjuk dönteni, hogy a Clock kimenetét először a Ramp majd a Plotter egység kéne, hogy feldolgozza, azonban ez nem triviális a szimulációnak. Ptolemyben a szimuláció megkezdése előtt a feldolgozó program topológiai

rendezést készít (függőségi viszonyok alapján), melyet figyelembe fog venni az azonos időbélyegű tokenek feldolgozási sorrendjénél.



5. ábra Ütemezés topológiai rendezéssel

A topológiai rendezést „zavarba hozhatja”, ha hurkokat alakítunk ki a modellben. Ekkor mindig kell, hogy legyen egy nem nulla késleltetési actor a hurokban.

Folytonos idejű rendszerek – Continuous Time

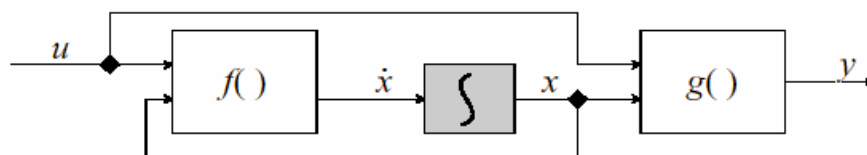
A folytonos idejű leírási módot elsősorban a beágyazott rendszert befogadó fizikai környezet leírására használjuk. A modell kimenetén és bemenetén folytonos idejű jelek vannak, melyek között differenciál egyenletek teremtenek kapcsolatot. A folytonos idejű rendszer általános matematikai modellje:

$$\begin{aligned}\dot{x} &= f(x, u, t) \\ y &= g(x, u, t), \\ x(t_0) &= x_0\end{aligned}$$

ahol x a rendszer belső állapota, u a rendszer bemeneti gerjesztése (input), y pedig a rendszer válasza (output).

Alapvetően két módon szokás egy folytonos idejű fizikai rendszert modellezni. Az első megközelítés a rendszer fizikai modelljét rögzíti (pl. áramköri modell), melyet bizonyos megmaradási törvényekkel (pl. Kirchoff egyenletek) tesz végrehajthatóvá ill. kiszámíthatóvá. Ennek a megközelítésnek az előnye, hogy a fizikai rendszerből könnyen – automatikusan – előállítható, azonban a matematikai modell nem látszik belőle közvetlenül.

A másik megközelítés a jelfolyam gráfok használata. Ez egy absztraktabb modellje a fizikai rendszernek, így az előállítása nehezekebb, viszont sokkal könnyebben kapcsolható a többi számítási modellhez és a matematikai leírás könnyen előállítható. Jelfolyam gráf esetén a bemenő jel valamilyen függvényértéket vagy a jel integrálját adják az actorok a kimenetükön. Az általános matematikai modell jelfolyam gráf alakját mutatja a 6. ábra.



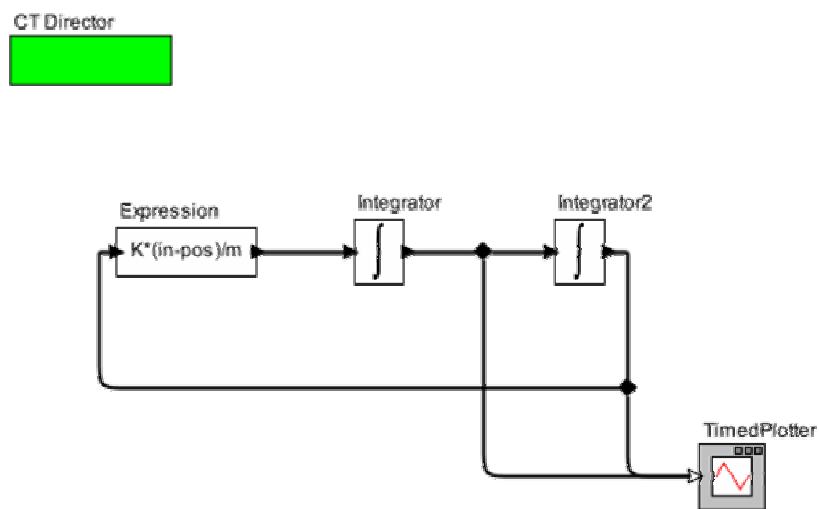
6. ábra Folytonos idejű rendszer általános jelfolyam gráfja

A szimulációs környezet többféle numerikus differenciál egyenlet megoldó algoritmust használhat. Ezek közös jellemzője, hogy az időben kis lépésekkel haladunk előre és differencia egyenletekkel közelítjük a megoldást. Az egyes megoldó algoritmusok különböznek abban, hogy képesek adaptívan változtatni a lépésközt az időben vagy annak nagysága rögzített. Az actorok között a jel lépcsősen (nulladrendű tartó) közelíti a folytonos jelet.

Determinisztikus viselkedés. A folytonos idejű rendszerek a Ptolemy környezetben determinisztikus viselkedést mutatnak.

Az idő fogalma. A modell végrehajtása valós időben történik, a rendszer globális órával rendelkezik.

Speciális végrehajtó egységek. A legfontosabb speciális actor az integrátor, e mellett a többi számítási modellel történő ötvöztést segítik a folytonos idejű rendszer határán elhelyezhető jelgenerátorok (n-edrendű tartók) és esemény generátorok (pl. null-átmenet érzékelők).



7. ábra Rugós rendszer leírása folytonos időben

A 7. ábra egy tipikus folytonos idejű rendszert mutat: egy rugóra (K – rugóállandó) kötött m tömegű test mozgását láthatjuk a Plotteren. Az első integrátor kimenetén a test sebessége, a másodikon annak pozíciója jelenik meg, melyet felhasználunk az elmozdulás és a testre ható erő közötti összefüggésben.

Szinkron adatfolyam gráfok – Synchronous Dataflow

Az adatfolyam gráfokat elsősorban jelfeldolgozási feladatok leírására használjuk. A szinkron adatfolyam gráf esetén minden iterációs lépésben minden actor pontosan egy tokenet fogyaszt el a bemenetén és egyet produkál a kimenetén, így mindegyik pontosan egyszer lesz végrehajtva az iterációs lépésben. A topológiai viszonyok alapján a director a szimuláció megkezdése előtt el tudja dönteni – és el is dönti – az actorok végrehajtási sorrendjét. A szimuláció megkezdése előtt felismerhető a deadlock. Pontosán ezért hurkokat csak késleltető elem közbeiktatásával hozhatunk létre. (A késleltetés itt iterációs lépéseket jelent, nem időt).

Egy kiterjesztése a fenti definíciónak a nem homogén adatfolyam gráf: itt egy végrehajtó egység több tokenet is fogyaszthat vagy állíthat elő egy iterációs lépésben. A 8. ábra ilyen gráfot mutat. A helyes ütemezési sorrend ebbe az esetben: AABCC.

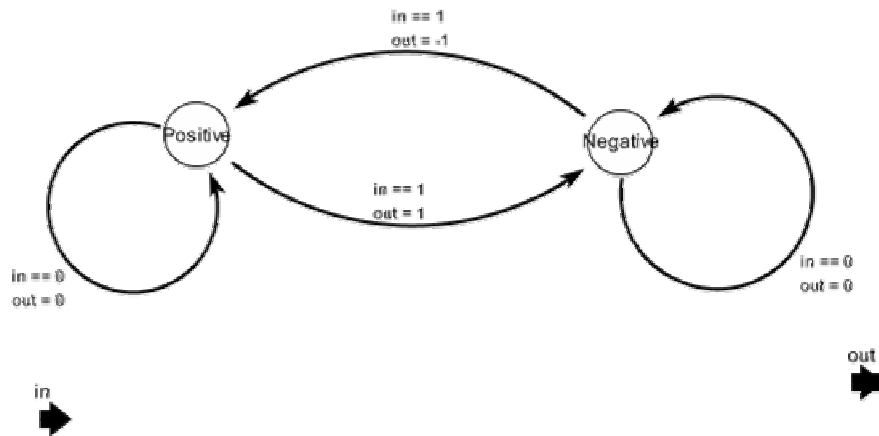
A 9. ábrán látható modell bal oldala az *Input* jelet a *Waveform* nevű vivőjelre ülteti, jobb oldala pedig előállítja az így keletkezett jel Fourier transzformáltját. Ezt valamilyen plotteren megnézve szépen látszik a vivő és a vitt jel elkülönülő frekvenciája.

Állapotautomaták – Finite State Machines

Az állapotautomaták egy kicsit kilógnak a számítási modellek sorából: a „dobozok” nem kommunikálnak egymással, hanem állapotokat írnak le. Az állapotok között átmeneteket definiálhatunk feltételekkel és valamilyen mellékhatás, művelet végrehajtásával.

A Ptolemy környezet – a UML Statechart nyelvéhez hasonlóan – néhány fontos dologgal egészítette ki a hagyományos állapotautomata nyelvet: hierarchikus állapotokkal és más számítási modellekkel történő kombinálhatósággal. (Lásd.: hibrid rendszerek)

Szoftverkomponensek belső működésének leírására rendkívül jól használható nyelv.



10. ábra AMI kódoló állapotai

A 10. ábra egy ún. *Alternate Mark Inversion* bit kódolót mutat. Ennek a kódolásnak a sajátossága, hogy akkor vált értéket, ha 1-es bemeneti értéket kap, egyébként 0-ban marad. Vegyük észre, hogy az állapotátmenetek feltételének ill. műveletének leírásakor felhasználhatjuk az állapotgép portjain lévő jeleket.

Idővezérelt rendszerek – Time Triggered Systems

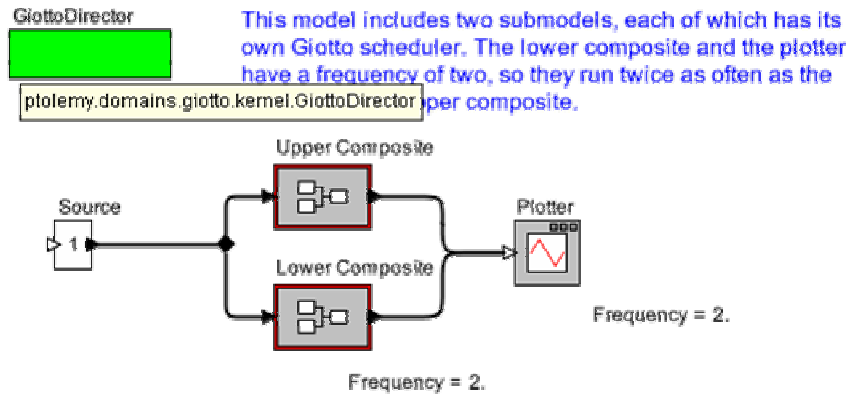
Az ismertetésre kerülő számítási modellek közül ez az egyetlen, melyben a végrehajtó elemeket nem a beérkező jelek, hanem az időzítő (director) hajt végre (előre megadott frekvenciával). Mivel minden actor rendelkezik egy „Worst Case Execution Time” paraméterrel az ütemező előre felismerheti az ütemezési problémákat.

Az actor által előállított tokenek csak a végrehajtás után érhetőek el más elemek bemenetein (pl. ugyanolyan frekvenciával működő összekötött komponensek közül a második egy ciklussal későbbi tokeneket lát)

Párhuzamos végrehajtás. Magas szintű párhuzamosság, mivel nincs az actorok között adat jellegű függés.

Determinisztikus viselkedés. A viselkedés kiszámítható, determinisztikus. Az idővezérelt rendszereket előszeretettel használják ipari alkalmazásoknál (pl. gyártósor)

Az idő fogalma. Erősen valós idejű. A rendszer egy globális órával rendelkezik.



11. ábra Különböző frekvenciával hajtott actorok

Kommunikáló folyamatok – Communicating Sequential Processes

Ebben a számítási modellben párhuzamosan futó folyamataink vannak. Külső kontroll (ütemezés) nincs a folyamatok fellett, azok maguk között próbálnak „szinkronizálni”, ha arra szükség van (adatcsere). A folyamatok között egyirányú FIFO (tároló) nélküli kommunikációs csatornák vannak, ami miatt mind a küldés mind fogadás műveletek blokkolhatják az actort. Két ilyen actor akkor tud adatot cserélni, ha ezekkel a műveleteikkel találkoznak (randevú). Az adatcsere atomi módon zajlik le, mely után mindkét actor tovább fut. Erőforrás-menedzsmenttel kapcsolatos problémák modellezésére rendkívül jól alkalmazható nyelv.

A deadlock csak futási időben ismerhető fel. Egyszerű deadlock felismerés: minden folyamat blokkolódott. (Ez természetesen nem ismer fel minden deadlock-ot)

Párhuzamos végrehajtás. Magas szintű párhozamosság, egyedül a randevú ideéig kell szinkronba kerülniük az actoroknak.

Determinisztikus viselkedés. A viselkedés a külső kontroll hiánya miatt nem determinisztikus

Az idő fogalma. Az eredeti modell az idő fogalmát nem kezeli, de létezik a nyelv idő alapú kiterjesztése.

Folyamathálózatok – Process Networks

Talán megfelelőbb lenne aszinkron adatfolyam gráfoknak hívni őket, mivel olyan adatfolyam gráfokról van szó, ahol egy actor akárhányszor meghívódhat, míg mások blokkolva vannak. A kommunikáló folyamatokhoz hasonló számítási modell, de a végrehajtó elemek között FIFO sorok vannak, így az írás (küldés) művelet nem blokkolódik. Tipikusan jelfeldolgozási problémák leírására használható. Az ütemezés előre nem kitalálható, a deadlock felismerésére ugyanaz igaz, amit a kommunikáló folyamatoknál elmondunk.

Párhuzamos végrehajtás. Nagyon magas szintű párhozamosság (csak az olvasás blokkolhat)

Determinisztikus viselkedés. Nem determinisztikus viselkedés. Ez az egyik „leglazább” számítási modell.

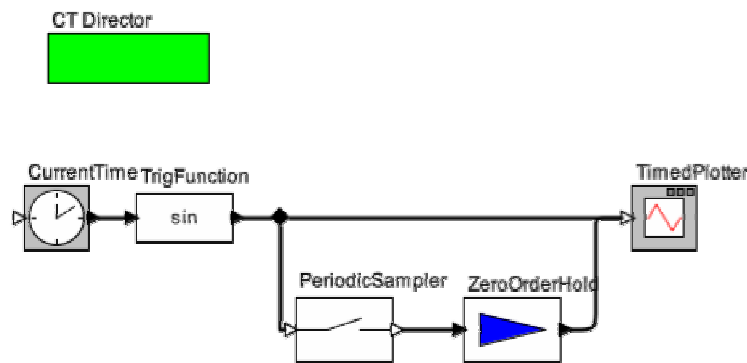
Az idő fogalma. Az eredeti modell az idő fogalmát nem kezeli, de kiterjeszhető

Számítási modellek ötvözése

A megismert számítási modellek lehetséges kombinációi közül nézzük meg a két legtipikusabb ötvözetet.

Kevert jelek – Mixed Signals

A kevert jeleket tartalmazó modellek tipikusan minden beágyazott alkalmazásban jelentkeznek: fizikai rendszer modellezése folytonos idejű jelekkel történik, amihez valamilyen diszkrét idejű számítógépes feldolgozást tervezünk. A folytonos idejű számítási modellnél megismert speciális végreható egységek segítenek a két „világ” illesztésében.



12. ábra Folytonos és diszkrét idejű jelek keverése

A 12. ábrán egy folytonos idejű szinusz jelből mintákat veszünk (diszkrét időbe térünk át), majd az így előállított jelet nulladrendű tartóval alakítjuk ismét folytonossá. A két actor közé tetszőleges diszkrét idejű számítási modellel leírt logika beilleszhető.

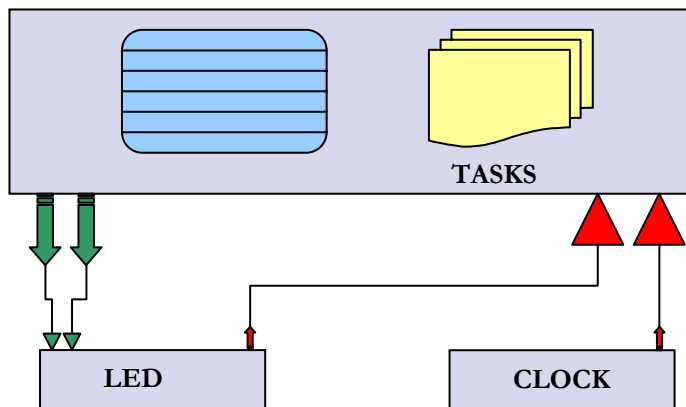
Hibrid rendszerek – Hybrid Systems

Sok esetben a beágyazott rendszer vagy maga a fizikai környezet bizonyos feltételek teljesülése esetén egy másik „viselkedési módba” kapcsol. Ha ezt a beágyazott rendszer teszi, átkonfigurálásnak hívjuk a folyamatot. Mivel állapotváltás történik, ilyenkor a már megismert állapotautomatás leírásunkat kombináljuk valamely másik számítási modellel (legtöbbször a folytonos idejű jelekkel). Ezeket a rendszereket hívjuk hibrid rendszereknek.

Egy – a példafájlok között is megtalálható – tipikus példa a hibrid rendszerekre a korábbi rugós rendszer kiterjesztése: két egymással szemben lévő rugó egy-egy testet mozgat. Ha a két tömeg pozíciója „összeér”, a rendszer megváltoztatja viselkedését: két szemben lévő rugó mozgat egy összeragadt tömeget. Amikor a két rugó elég erősen húzza ketté a tömeget, az ismét ketté válik és az első működési mód szerint rezegnek tovább.

TinyOS alkalmazás modellezése

Szintén a példa fájlok között található meg 13. ábrán látható TinyOS alkalmazás Ptolemy-ben elkészített modellje. Az alkalmazás a LED-jeit villogtatja az időzítő esemény hatására. A példában a komponenseket diszkrét eseményekkel modelleztük, de néhányuk belsejében állapotgépet kellett elhelyezni (például a LED ki- és bekapcsolásához). A példa jól mutatja a számítási modellek ötvözésének hatékony módszerét.



13. ábra A TinyOS Blink alkalmazás