Közlekedési események leírása formális módszerekkel

Max Gyula*

* Automatizálási és Alkalmazott Informatikai Tanszék, Budapesti Műszaki és Gazdaságtudományi Egyetem,
H-1521, Budapest, Pf. 91., Tel: +(06-1) 463-2870, Fax: +(06-1) 463-2871,
E-Mail: max@aut-bme.hu, WWW: http://www.aut.bme.hu/~max

Abstract: – A közlekedést megfigyelő vagy irányító információs rendszerek esetében a mozgó objektumok és környezetük pontos leírása elengedhetetlen követelmény. Az eddig ismert közlekedési rendszerek szinte kizárólag csak a már megtörtént vagy a jövőben megtörténő eseményekre összpontosítottak, pedig néhány formális módszer megfelel arra, hogy a mozgó objektumokkal egyszerre, real-time módon leírja a mozgás környezetét is. Közülük azonban csak néhány próbálja meg részletezni a környezet és a mozgó objektum közötti rendszeres vagy rendszertelen kölcsönhatásokat. A cikkben a közlekedés események formális leírásával foglalkozunk, bemutatva egy helyszíneken, mozgó objektumokon és eseményeken alapuló modellt. Modellünk bemutatja a Güting-féle tér-idő módszer részletes formális leírását a Real-time Object-Z formális leírónyelv segítségével.

Kulcsszavak: közlekedési információs rendszerek, mozgó objektumok, formális specifikáció.

1. BEVEZETÉS

Többféle közlekedés megfigyelő rendszert tanulmányoztak már az irodalomban [R. Cucchiara et al., 2000] különféle közlekedési modellek segítségével. Néhányuk a mozgó tárgyak viselkedésére [Yoneyama et al., 2003], mások járművek osztályozásra [Aguilar-Ponce et al., 2005] összpontosított, miközben legtöbben bemutatták az információ továbbítás útvonalát is, az elkészített utcai felvételektől a döntéshozatalig. A közlekedésmodellezéssel azonosított adatfeldolgozás esetén egy sor kép feldolgozását végezzük el azért, hogy a céljainknak felelő információkat szerezhessünk az adott környezetben folyó eseményekről, melynek célja az, hogy létrehozzunk egy megfelelően részletes leírásmodellt, amely magas szinten írja le az alacsonyabb szintekről érkező eredményeket [Max, 2008]. Egy helyszín és egy objektum tér-időbeli leírási módszereit keressük miközben az eseményeket, és kölcsönhatásukat vizsgáljuk.

Cikkünk ennek a módszernek a strukturális leírását mutatja be, megadva azokat a lehetőségeket, amellyel strukturálhatjuk a részletes leírásokat, vagy olyan osztályokat képezhetünk a Real-time Object-Z segítségével, amelyek származtatott elemeket és az osztályok párhuzamos felhasználását teszik lehetővé, hogy sok összetevőből álló rendszereket hozhassunk létre és vizsgálhassunk meg. A származtatott osztályok és a párhuzamosan működő, egymással konkuráló osztályok használata lehetővé teszi a sok összetevőből álló rendszerek vizsgálatát. Mindemellett e két módszer összekapcsolásával egy tömörebb, általánosabb képet kaphatunk a vizsgált rendszerről.
A cikk második fejezetében egy áttekintést adunk a valós idejű Object-Z (RTOZ) tulajdonságairól, amely bemutatja, hogy hogyan használhatjuk fel a RTOZ építőelemeit. Egy egyszerű példán keresztül be is mutatjuk ezeket módszereket, amelyek magukba foglalják a hely és az időbeli változók használatát is. A harmadik fejezetben külön megmutatjuk, hogy különböző rendszerekből hogyan hozhatunk létre sok összetevőből álló rendszereket, miközben egyes tulajdonságaikat egy, már előzőleg definiált rendszerből származtatjuk vagy a már létező tulajdonságaikat módosítjuk. Következtetéseinket a negyedik fejezetben adjuk meg.

2. REAL-TIME OBJECT-Z

A Z leírónyelv egy közös matematikai jelölésrendszert határoz meg. Sémákat használhatunk arra, hogy leírjuk a rendszer részeit és a sémaszámítás megengedi a részletes leírások moduláris és hierarchikus szerkezetét.

Real-time Object-Z [Smith et al., 1999] a Z nyelv kibővítése az objektumok használatával és az idő egyre finomabb megadásának és számításának lehetőségével [Smith, 2000]. A két feladat ellátására kialakított bővítmény szintén egy Z nyelven alapuló jelölésrendszer, amely segítségével lehetőségünk van a valós idejű rendszerek részletes leírására

és finomítására. A Z nyelvet ehhez, egy egyszerű halmazelméleti jelölés rendszerrel egészítették ki, amely segítségével tömören adhatjuk meg az objektumok leírását és az időintervallumok hosszát, végpontjait és a rajtuk értelmezett műveleteket.

Egyesítve ezeket az összetevőket, egy olyan környezetfüggetlen ismeretábrázolási nyelvet kapunk eredményül, amely egyrészt támogatja az idő változós régiók használatát, másrészt egy közérthető matematikai formalizmus segítségével adható meg. Szintén átvesszük annak a jelölésrendszernek is az egyszerűsített részhalmazát, amely lehetőséget nyújt minimális általános halmazelméleti számításokhoz. Ez a megadás mód különbözik attól a megközelítéstől, amely a folytonos és valós időjű rendszerek részleteit külön írja le Object-Z-ben, mivel mi csak szabványos Object-Z jelöléseket akarunk használni, de az időzítés számítást [Mahony et al., 1998] további jelöléseket kénytelen behozni az Object-Z osztály sémáiba, mivel csak ezek segítségével tudjuk fenntartani az Object-Z részletes - mind funkcionális, mind valós idejű tulajdonságait - megadó leírási technikáját. Értelemszerűen modellezni kell tudnunk még az idő múlását is, amelyet egy ún. Tick művelettel adhatunk meg az egyes osztályokban.
Mindezekkel az a célunk, hogy pontosan le tudjuk írni egy ismert helyszínen felvett képsorozatból az ott történt eseményeket illetve, az ott lévő objektumok pontos viselkedését. Céljaink eléréséhez persze néhány képtisztítási módszert is fel kell használni. A forgalom színhelyének alaptérképét primitív térbeli elemekből, ún. amiket levélrégiókból állítjuk össze. A levélrégiók a helyszínen adott, egymást átfedhető régiók határainak halmazát jelenti. Az összetett régiók olyan levélrégiókból állnak, amelyeket az unióképzés szabályai szerint illesztettünk össze. Ugyanaz a levélrégió több összetett régióhoz is tartozhat. Ezzel a módszerrel egy hierarchikus térbeli adatbázist alakítunk ki, kifejezve ezzel a színhely különböző jellemzőit, olyan régiókat is beleértve, amelyek a várt járműtípust, járműviselkedést, irányinformációkat vagy egyéb kapcsolódásokat jeleznek. Ennek a hierarchikus rendszernek a felépítéséhez változókat, típusokat, funkciókat, osztályokat és a rajtuk elvégezhető különböző műveleteket kell megadnunk.

2.1 Változók

Az RTOZ néhány különleges változót használ az ismert változó típusokból.

Környezeti változók: Ezek a változók a környezettel kommunikálnak.

$$\mid \text{NewPosition?} : \mathbb{T} \rightarrow \mathbb{R} \times \mathbb{R} \qquad - \text{pozíció}$$

A ? utótag a változók névének végén olyan környezeti változókat definiál, amely inputként csatlakozik rendszerhez. Hasonlóan, a ! a változók nevének végén outputot jelöl meg a rendszerben.

2.1.1 Állapotváltozók

A rendszerünket digitális rendszerként specifikáltunk, tehát a változók változásai csak diszkrét időpillanatokban történhetnek. Magukat a változókat lokális vagy globális állapotváltozókként deklaráljuk. Mindazonáltal a folytonosan változó változók az adott környezetben hatnak egymásra. Ezért ezeket a változókat (esetleg differenciálható) időfüggvényekként definiáljuk. Minden osztályban van egy kiemelt fontosságú hallgatólagos állapotváltozója is, a $\tau : \mathbb{T}$, ami az aktuális időt jelöli. Minden államváltozót egyben időzített nyomkövető változóként is értelmezünk az $x: \mathbb{T} \rightarrow X$ időzítési tartományban.

2.1.2 Időzített nyomkövetési változók

Egy állapotváltozó időzített nyomkövetési értéke egy változó értékébe képzett lenyomat az adott időpillanatról:

$$\text{Időlenyomat} == \mathbb{T} \rightarrow \text{érték}$$

Időzített nyomkövetési értéket minden környezeti és minden egyes állapotváltozóhoz hozzárendelünk. Ezeket az információkat a változók három részhalmazára adhatjuk meg: a környezeti változókként kezelt inputokra, az ugyancsak környezeti változókként megadott outputokra, és a lokális változókként definiált állapotváltozókra.

2.1.3 Megfigyelhető változók

Egy rendszernek ezeket a változóit teljes függvényként modellezik az időtartomány függvényektől a típus leíró függvényekig, azaz képviseltetünk benne minden értéknek a készletét, amit a változó feltehet.
RTOZ-ben csak a műveleteket illetve azok inputjait vagy outputjait tekintjük megfigyelhetőnek. Egy specifikáció egyéb állapotai, mint pl. egy időzítés logikai tulajdonságai, belső és egyúttal nem megfigyelhető állapotok maradnak.

2.2. Műveletek időben nem változó változókon

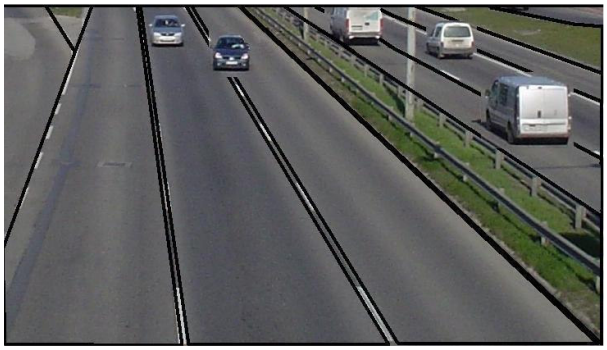
Mint azt már korábban elmondtuk, a célunk az, hogy pontosan le tudjuk írni egy ismert helyszínen felvett képsorozatból az ott történt eseményeket illetve, az ott lévő objektumok pontos viselkedését.
Tekintsünk egy járművet, amely egy adott területen áll egy adott időpillanatban. Erre a területre, régióra nézve próbáljuk meg értelmezni azt a kifejezést, hogy az adott jármű benn van-e a vizsgált régióban vagy sem. Ebben az esetben a régió fogalma nagyon fontossá válik, mert arra használhatjuk, hogy jelentős mennyiségű információra következtethetünk belőle. Azt is megtudhatjuk ebből, hogy mely járművek vannak ugyanazon úton, vagy következtetéseket vonhatunk le arra nézve, hogy egy jármű térbeli útvonala lehetséges-e, megmérhetjük a távolságot a járművek között. A mozgó objektumok állapotaiban történt változásokat a régióhatárokon történő áthaladások jelzik.

A közúti forgalom-tartomány modelljének ezenkívül még képesnek kell lennie arra is, hogy felületén mozgó vagy álló objektumok és helyszínek jellemző tulajdonságait is leírja. Ráadásul meg kell tudni adnia az adott tartományhoz kapcsolódó tudást, mint például a forgalmi szabályokat, amelyik lényegében engedélyezett vagy várt viselkedés mutatják be. Végezetül értelmezni kell tudnia olyan jelrendszereket is az adott forgalmi tartományba, mint például a jelzőtáblák vagy az útburkolati jelek. Az irodalomból vett példák azt illusztrálják, hogy a régió alapú modellezés teljesíti ezeket a célokat. Ezáltal egy közlekedési esemény modellezhetővé válik, már csak az esemény szimbolikus leírása jelenti a problémát.

Tekintsük az 1. ábrán bemutatott egyszerű forgalmi helyszínt. A helyszínhelyen öt járműveket halad. Kettő a bal oldali, három pedig a jobb oldali sávok egyikében. Ezeket a sávokat régióknak tudjuk feltüntetni a helyszíni térképen. Az út azon régióit, amelyeken a tárgyak mozognak, a mozdulatlan régiók összegének tüntethetjük fel (háttérazonosítás). Ezt a hátteret két háromsávos régióra, egy felhajtósávra és két fűvel fedett mezőre oszthatjuk.

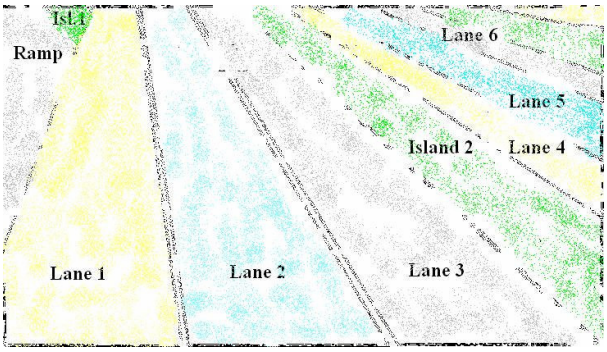
1. ábra: Az eredeti háromsávos modell

Egy füves zöld terület határolja el egymástól a két a háromsávos régiót. Füves területet találunk még a terület bal felső sarkában is, amelynek aljához egy felhajtó sáv csatlakozik. A háromsávos régiókat terelővonalak osztják fel az útfelületen. A sávok részhalmazát és az egyéb említett régiók mindegyikét a 2. ábra mutatja.



2. ábra: A mozgó objektumok részére három részre bontott háromsávos autópálya részlet

Kiegészítésként még felhívjuk a figyelmet arra, hogy a mozdulatlan három sávos régiók egyikéhez sem tartozik parkolóterület és hogy a maximális sebesség 70 km/h-ban maximált.
A régiók alakját világosabban láthatjuk a 3. ábrán. Még egy olyan egyszerű szegmentálással is, mint amit most mi használunk, egy egészen tisztességes modellt kaphatunk a helyszín régióiról.



3. ábra: Az elkülönített régiók

Tudjuk például, hogy egyetlen jármű sem hajthat be a régiókba legálisan, 70 km/órás sebességnél gyorsabban, mivel a vizsgált sávok mozgási felületeit sebességkorlátozó régiókkal is lefedtük.

Általában a vizsgált közúti felület objektumokból és az ott történő eseményekből áll. Az út felszínén megkülönböztetünk mozgó és mozdulatlan objektumokat. A mozgó objektumok mozdulatlanná válhatnak vagy fordítva. A környezet leírását a kontextus régiók segítségével adjuk meg. Ezek a régiók, mint egy tudásbázis, a mozgó vagy mozdulatlan tárgyakkal vannak kapcsolatban, és arra használjuk őket, hogy megadják például a jelzőtáblák jelentéstartalmát vagy az egyes régiók közötti kapcsolatrendszer leírását.
Az eseményeket többféleképpen meghatározhatjuk. Például egy balesetet le lehet úgy írni, mint két régió kereszteződését, amikor megtörténik az összeütközés a két jármű között. Mindemellett, a kontextus régiókat is kétféleképpen használhatjuk: limitálhatjuk a kontextus régióba érkező objektumok tulajdonság értékeit (pl. sebességkorlátozás) vagy megadhatjuk az erre a területre érvényes szokások adatait (pl. a felhajtósáv hossza 40 méter). Ezzel a módszerrel korlátozhatjuk a sebességet, vagy forgalmi jelzőtáblákat (pl.: elsőbbségadás) tudunk használni. De a kontextus régiókat használhatjuk akár a mozgó objektumok mozgáspályáinak kiszámítására is, vagy egyéb kölcsönhatások, vezetői szándékok megjósolására is.

2.3 Műveletek időben változó változókon

Ahogy azt az előzőekben láttuk, van néhány eszköz a kezünkben, hogy tanulmányozzuk és reagáljunk az egyes eseményekre. Mindehhez azonban időre van szükségünk, mivel az események időben történnek. Modellünkben meg kell határozni az időt. $\mathbb{T}$, az abszolút időt reprezentálja

számunkra valós számok formájában, milliszekundumos egységekben. Ezek a rendszer megfigyelhető változóit függvényként értelmezik úgy, hogy az időtartományt leképezik a valós számok halmazára. A függvény értékkészlete minden olyan értéket képvisel, amit a változó felvehet. Egy rendszer időbeli tulajdonságait definiálhatjuk a rendszert leíró megfigyelhető vagy állapotváltozóihoz tartozó időintervallumok segítségével. Például a következő kifejezés azt mutatja, hogy egy megfigyelhető változó $b : \mathbb{T} \rightarrow \mathbb{R}$, 0.1 másodpercen belül egyenlővé válik a: $\mathbb{T} \rightarrow \mathbb{R}$ megfigyelhető változóval minden olyan időintervallumban, ahol a > 50.

$$\langle a > 50 \rangle \subseteq \langle \delta = 100 \rangle ; \langle b = a \rangle \quad (1)$$

A zárójelek < > arra használják, hogy specifikáljuk az időintervallumok halmazát. A fenti kifejezés bal oldala (1) minden olyan időintervallumot képvisel, ahol az a(t) értéke 50-nél nagyobb.

Általában, a zárójelekben valamilyen fontos tulajdonságot adunk meg, ami az időértékeket valamilyen egyéb, X változótípusba transzformálja. Ezeknek a függvényeknek az időtartományára való világos hivatkozása olyan részletes leírásokkal végződik, amik tömörebbek és olvashatóak. A fenti kifejezés jobboldala két intervallumot tartalmaz. Az első, a δ fenntartott szimbólumot használja, amely az időintervallum hosszát jelenti. Esetünkben ide tartozik minden olyan időintervallum, amelynek a hossza 100 milliszekundum. Van még két másik fenntartott szimbólum is, az α és ω, amely egy időintervallum kezdetét illetve végét jelenti.

A második intervallum az összes olyan időintervallumot jelenti, amelyben b értéke megegyezik a-val. Ezt kombináljuk az első időintervallummal, az összefűzés jelentő ";" művelet segítségével. Ez a műveket egy olyan időintervallum csatlakozást hoz létre, amelyben az egyik végpontja a másik kezdőpontjához kapcsolódik. (Egy végpontot le kell zárni ahhoz, hogy egy másikat megnyissunk). Mivel a kifejezés jobboldala megadja mindazokat az időintervallumokat, amelyekben 100 milliszekundum után, b megegyezik a-val. Az egész kifejezés tehát megadja (a $\subseteq$ szimbólumot használva) az összes olyan időintervallumot, ahol az a értéke nagyobb 50-nél, és azokat az időintervallumokat is, ahol 0.1 másodperc múlva b meg fog egyezni a-val.

2.4 Osztályok

Az osztályok, a RTOZ integrált jelölésrendszerében két részből állnak, amit egy vízszintes vonal választ el egymástól. A vonal feletti rész a szabványos RTOZ lokális definícióinak és sémáinak a helye. A vonal alatti rész az osztályokon történő további megszorítások helye, amit a pontosított időzítési jelölés rendszerben részleteznek. Ez az utóbbi rész a pontosított időzítés számításokban két részre van osztva: egy feltétel és egy következmény részre. Minden vonal alatti $x : \mathbb{T} \rightarrow X$ az időzített nyomkövetési változót a vonal feletti részben időzített nyomkövetési állapotváltozóként x : X értelmezünk.

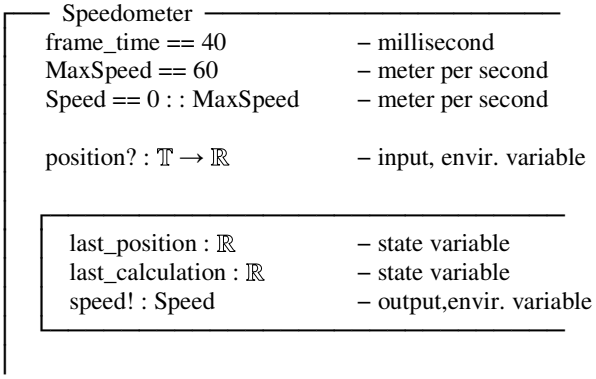
Bár minden valós idejű tulajdonságot definiálhatnánk az osztály időzített nyomkövetési részében, azért lokális állandók és $\mathbb{T}$ típusú állapotváltozók használata megengedett, valamint minden osztályban létezik egy $\delta : \mathbb{T}$ az aktuális időt jelentő állapotváltozó. Ez az értelemszerű megszorítás a $\forall t : \mathbb{T} \bullet \tau(t) = t$ kifejezést definiálja az osztály időzített nyomkövető részében.

Példaként tekintsünk olyan sebességmérőt, amely egy mozgó jármű sebességét számítja ki, miután meghatározta annak pozícióját: a sebességet úgy számítja ki, hogy elosztja a két egymást követő pozíció különbségét azzal az idővel, ami egy felvétel elkészítéséhez szükséges. Tegyük fel, hogy a maximális sebesség 60 m/s (216 km/h).

Ebben a példában, outputot (speed!) azért deklaráltuk állapotváltozóként, hogy értéke csak akkor változhasson, amikor a változó listában, (Δ-list) szereplő valamelyik művelet bekövetkezik. A művelet neve logikai-változókként jelenik meg az osztály időzített nyomkövetési részében. A változó, amit ez a művelet képvisel, igaz minden időintervallumban, ami alatt az operáció történik.

A Speedometer kiszámítja a sebességet (speed!) a mozgó jármű legutóbbi és eggyel korábbi pozíciójából. Ahhoz, hogy ezt meg tudja tenni, az utolsó sebességszámításhoz tartozó pozíció értéket a last_position állapotváltozóban meg kell tartani.

A legelső esetben, amikor mozgó ármű belép a háromsávos régiók egyikébe, a last_position, a position?(τ) és speed! értékeit nullára, míg a last_calculationt értéket a frame_time-nál kisebb értékre állítjuk be. Mivel minden frame_time milliszekundumban új keretek jönnek, ez gondoskodik arról, hogy CalculateSpeed elkezdődjön. A sebesség-számítás nullára állításáról azért kell gondoskodnunk, mert nem ismerjük a régióba belépő jármű utolsó, még egy másik régióban mért pozíciója. CalculateSpeed minden alkalommal végrehajtódik, amikor a last_calculation vagy a speed! értéke megváltozik. Inicializálás alatt a position?(τ) értékét nullára vették fel, úgyhogy ez kényszeríti a CalculateSpeedet, hogy végrehajtódjon. A CalculateSpeed addig vár, amíg az aktuális idő egyezik meg n-edik frame_time idejével, hol n egy intereger. Amikor frame_time-nak vége van, egy új sebesség értéket számít ki. Következő lépések során elmentjük a szükséges változókat. Így a Last_calculation (és a speed!) változók gondoskodnak arról, hogy a CalculateSpeed újra elindulhasson.



INIT

last_position = 0
position?(τ) = 0
last_calculation < frame_time
speed! = 0

CalculateSpeed

Δ (last_calculation, speed!)

τ mod frame_time = 0
∀ t : (τ …τ'] • τ mod frame_time ≠ 0
speed!'= (position?(τ) – last_position) / frame_time
last_calculation' = τ
last_position' = position?(τ)

< τ mod frame_time = 0 > ;
< τ mod frame_time ≠ 0 > ⊆ < true >;
< CalculateSpeed > ; < true >

3. FORMÁLIS SPECIFIKÁCIÓ

Az előző részekben elmondtuk, hogy magát a forgalom tartományát hogyan modellezhetjük régiók felhasználásával, különös tekintettel az objektumokra, a kontextus régiókra és az eseményekre. Magát helyszínt az 1. ábrán, míg a helyszínen található régiókat és a mozgó objektumokat a 2. ábra mutattuk be. A 3. ábrán a képről eltávolítottuk a mozgó objektumokat, hogy láthassuk a szegmentált régiókat. A megfigyelt régiót három sávrégióba, egy felhajtósáv régióba és két füves régióba osztottuk. Megjegyezzük, hogy a Parkolni tilos tábla és a sebességet korlátozó jelzés 70 km/h jelzés látható módon nem szerepel egyik képen sem.

3.1 Régiók és objektumok

A megfigyel tartományom belül minden objektum a mozgó régiókon alapszik, mivel a mozdulatlan, álló régiókat nulla sebességgel mozogó régióknak tekinthetjük. Egy általános objektumot a következő séma segítségével definiálhatunk:

OBJECT

identifier :	STRING
region :	MREGION
speed, dir :	R
c :	POINT
c	= center(region)
(speed, dir)	= velocity(region)

Látható, hogy ugyanezzel a sémával a mozdulatlan régiók megadása is megtörténhet, csak abban az esetben a sebesség értéke nulla lesz.

CONTEXT

region :	REGION
limit :	OBJECT → BOOL
label :	STRING

Mint az látható, a kontextus régiók tartalmazhatnak kritériumokat, korlátozásokat is, amelyekkel bizonyos feltételeket fogalmazhatunk meg az adott régióra a következő formában:

limit : OBJECT → BOOL

∀ o : OBJECT • limits(o) — az o = igaz teljesülése
— esetén életbelépő
— korlátozások

A kontextus régió tartalmaz egy stringként definiált címkét, amellyel alternatív módon hivatkozhatunk ezekre a korlátozásokra.

3.2 Mozgó és álló objektumok

Egy objektum állapota a tulajdonságaihoz rendelt értékekkel adható meg. A következő példák különböző állapotokat mutatnak be:

Mozgó objektum:

Moving : OBJECT → BOOL

∀ o : OBJECT • Moving(o) = (o.speed > 0)

Álló objektum:

Stationary : OBJECT → BOOL

∀ o : OBJECT • Moving(o) = (o.speed = 0)

3.3 Események

Egy eseményt tekinthetünk úgy, mint egy vagy több objektum állapotainak időbeli átmeneteit egyik állapotukból egy másikba. Ezt a gondolatmenetet folytatva az összetett eseményeket vagy történéseket egyszerűbb épületkockákból vagy primitív eseményekből rakhatjuk össze, amiket térben vagy időben egymáshoz kapcsolódnak, és olyan egyidejű, esetleg egymással konkuráló eseményekből állnak, amiket időben szinkronizálnunk kell [Nevatia, 2004]. Egy tetszőleges kontextusban az objektum állapotát (STATE) egy logikai változóba képezzük le. Például az objektum egyik állapotjelzője lehet a mozgás vagy mozdulatlanság, de felhasználhatjuk az állapotjelzőket arra is, hogy megmutassuk, vajon egy objektum benne van-e egy adott állapotban vagy sem:

InContext : OBJECT x CONTEXT → BOOL

∀ (o : OBJECT , c: CONTEXT) • InContext(o,c)
= overlap(o, c)

Ezek után egy primitív eseményt (EVENT) egy olyan függvényként adhatunk meg, ami az objektum belső állapota és környezetének állapota között határoz meg egy átmenetet, míg az eseménysorokat (HISTORY) az egymást követő események egy véges sorozatának tekinthetjük.

EVENT : STATE × STATE → BOOL

HISTORY : EVENT1 × . . . × EVENTn → BOOL

A primitív események tehát olyan események, amelyek az objektum állapotai közötti kapcsolatokat írják le. Sok primitív eseményt közvetlen átmenetként jellemzünk egyik állapotból a másikba. Mivel ezek az események gyakran megtörténnek, a könnyebb áttekinthetőség miatt, létrehozunk egy ún. átmenet (Transition) függvényt, amely megadja ezeket az állapotátmeneteket:

Transition : STATE × STATE → BOOL

∀ state1, state2 : STATE | (∃ p1, p2: PERIODS •
p1 = deftime(o when in [state1]) ∧
p2 = deftime(o when in [state1])) •
Transition(state1, state2) = TRUE ⇔
∃ (t1 ∈ p1, t2 ∈ p2) | max(t1) = min(t2)

Informálisan az átmenetfüggvény tehát az objektum egyes állapotából a kettes állapotba történő közvetlen átmenetet írja le. Felhasználva ezt a definíciót, példaként, bemutatjuk néhány esemény leírását:

Elinduló objektum:

Start : OBJECT → BOOL

∀ o : OBJECT • Start(o) =
Transition(Stationary(o), Moving(o))

Megálló objektum:

Stop : OBJECT → BOOL

∀ o : OBJECT • Stop(o) =
Transition(Moving(o), Stationary(o))

Egy adott környezetbe belépő objektum:

EnterContext : OBJECT × CONTEXT → BOOL

∀ (o : OBJECT, c: Context) • EnterContext(o,c) =
Transition(¬InContext(o, c), InContext(o, c))

Egy adott környezetet elhagyó objektum:

LeaveContext : OBJECT × CONTEXT → BOOL

∀ (o : OBJECT, c: Context) • LeaveContext(o,c) =
Transition(InContext(o, c), ¬InContext(o, c))

3.4 A tulajdonságok örökítése

A 2. fejezetben egy sebességmérő (Speedometer) részletes leírását dolgozzuk ki. A megoldás során a problémát az jelenti, hogy ha akkor használjuk a sebességmérőt, amikor a mozgó tárgy elhagyja a megfigyelt régióinkat, és mivel ekkor a sebesség számítás művelete már nem történik meg, kimeneti sebességként mindig az utoljára kiszámolt érték kapjuk eredményül. Ahhoz, hogy ezt a problémát feloldjuk, meg kell adnunk egy olyan rendszert, amely észleli azt az eseményt, amikor a mozgó objektum belép vagy elhagyja régiónkat és a kimeneti sebességet 0-ra állítja. A problémát megoldhatjuk a szabványos RTOZ örökítési funkcióinak felhasználásával. Amikor egy RTOZ osztály megörököl egy másikat, az implicit módon tartalmazza annak állandóit, változóit, állapotsémáit és műveleteit. Ezeket a definíciókat a későbbiekben módosíthajuk is.

MeasuringSpeed

Speedometer − inheritance

c:	CONTEXT	− context region
StepIn :	EnterContext	− state variable
StepOut :	LeaveContext	− state variable
Inside :	InContext	− state variable

INIT

StepIn(o,c)	= false
Inside(o,c)	= false
StepOut(o,c)	= false

Measuring

Δ (StepIn, StepOut, position?(τ))

τ mod frame_time = 0
∀ t : (τ …τ'] • τ mod frame_time ≠ 0
< ∀ o : OBJECT • StepIn(o,c) >;
<Create(o); Speedometer;>
< ∀ o : OBJECT • StepOut(o,c)>; <Delete(o)>
< ∀ o : OBJECT •Inside(o,c) >; <Speedometer>

< τ mod frame_time = 0 > ;
< τ mod frame_time ≠ 0 > ⊆ < true >;
< Measuring > ; < true >

Az új rendszerünk, amelyet sebességmérésnek (MeasuringSpeed) neveztünk el, első lépésként megörökli az előzőekben elkészített sebességmérő (Speedometer) tulajdonságait, valamint deklarál és inicializál még néhány új változót. Minden egyes bejövő kép feldolgozása során a

mérés (Measuring) művelete végrehajtódik, melynek során sémánk, a vizsgált környezetbe történő belépésekor létrehoz egy új objektumot, majd kiszámítja az új vagy a már létező objektum sebességét. Amikor az objektum elhagyja a megfigyelt területet, törli azt, amely így elveszti eddig fennálló tulajdonságait.

3.5 Párhuzamos eseménynek kezelése

Azokban a rendszerekben, amelyekben több esemény is megtörténhet ugyanabban az időpillanatban, mindig gondot okoz a párhuzamosság problémájának kezelése. Hogy megoldjuk a párhuzamosság problémáját, bemutatjuk az osztályok számára kidolgozott, a párhuzamosság kezelésére szolgáló ‖ műveletet. Az elképzelésünk erről a műveletről az, hogy engedje meg az osztály bármely összetevőjének azt, hogy időben végrehajthassa megadott műveleteit, miközben szinkronizálja azokat a rendszer más elemeivel a környezeti változókon (pl., az ? és !) és közös elnevezésű műveleteiken keresztül. Mivel ebben az esetben a műveletek átlapolódhatnak, ha szükséges két vagy több művelet kölcsönös kizárása a végrehajtandó műveletek során, akkor azt explicit módon, nekünk kell megadnunk. Az egyik ilyen lehetséges mód, amivel ezt meg tudjuk adni, ha olyan monitor osztályokat definiálunk, amelyek tartalmazzák az adott folyamat összes komponensét.

4. KÖVETKEZTETÉSEK

Ebben a cikkben bemutattuk azt, hogy hogyan lehet felhasználni a Real-time Object-Z formális leírónyelvet közlekedési események megadására, miközben kitértünk az RTOZ osztályainak és öröklődési mechanizmusainak valamint a konkurens események kezelésének bemutatására is. Ezek a tulajdonságok különösen fontosak olyan rendszerek tervezése során, amelyek sok összetevőből állnak. A bemutatott példák segítséget nyújtanak mind az RTOZ könnyebb megértésében, vagy akár egy később kidolgozandó nagyobb rendszer tervezésében is.

.

IRODALOMJEGYZÉK

R. Aguilar-Ponce, A. Kumar, J.L. Tecpanecatl-Xihuitl and M. Bayoumi (2005). Autonomous Decentralized Systems Based Approach to Object Detection in Sensor Clusters. *IEICE/IEEE Joint Special Section on Autonomous Decentralized Systems*, pp. 4462 – 4469.

R. Cucchiara, M. Piccardi, and P. Mello (2000). Image analysis and rule-based reasoning for a traffic monitoring system", *in IEEE Trans. on Intelligent Transportation Systems*, **1. 2**. pp. 119 – 130.

B.P. Mahony and J.S. Dong (1998). Blending Object-Z and Timed CSP: An introduction to TCOZ. *20th International Conference on Software Engineering (ICSE'98)*, IEEE Computer Society Press, pp. 95-104.

G. Max (2008). Model of automatic recognition of traffic events. *Acta Agraria Kaposvariensis* **Vol 11 No 2**, University of Kaposvar, pp. 1 – 9.

Nevatia, R., Hobbs, J., and Bolles, B (2004). An ontology for video event representation. *IEEE Workshop on Event Detection and Recognition.*

G. Smith and I. Hayes (1999). Towards real-time Object-Z. *1st International Conference on Integrated Formal Methods*, Springer-Verlag, pp. 49 – 65, 1999.

G. Smith (2000). The Object-Z Specification Language. *Advances in Formal Methods*, Kluwer Academic Publishers.

A. Yoneyama, C.H. Yeh and C.-C. Jay Kuo (2003). Moving cast shadow elimination for robust vehicle extraction based on 2D joint vehicle/shadow models. *IEEE Proc. International Conference on Advanced Video and Signal Based Surveillance*, pp. 229 - 236 p.