A ℜeALKB tudástár metamodell-vezérelt megvalósítása

Kilián Imre[1], Alberti Gábor[2]

[1]PTE TTK Informatika Tanszék
[2]PTE BTK Általános Nyelvészeti Tanszék
7624 Pécs, Ifjúság útja 6.
kilian@gamma.ttk.pte.hu[1], alberti.gabor@pte.hu[2]

Kivonat: Az elmúlt évben a ℜeALIS természetes nyelvi elemző és értelmező rendszer[1] [1] tudáskezelő rendszerével kapcsolatos elméleti meggondolásokról számoltunk be [2]. Az elméleti meggondolások mellett egy sor deszkamodell-szerű tesztprogram futtatása engedte meg a derűlátó előrejelzéseket. A deszkamodellek integrációja megkezdődött: a jelen írás ennek előrehaladásáról számol be. A választott megoldás két szempontból is érdekes. Egyrészt a szoftver felületei révén programozható, és a Szemantikus Web projektum OWL ontológialeíró nyelvével [3] felülről kompatibilis, vagyis kész OWL ontológiák betölthetők. Másrészt a tudástár hátterében annak különválasztott metamodellje áll, és a programozható felületen keresztül a tudáselemek metamodell-vezérelt módon hozhatók létre és kérdezhetők le. A következtetések szemszögéből nem cél a teljesség. Egyes korai következtetések betöltési időben belefordíthatók a tudásbázis Prolog tárgymodelljébe, más következtetések későiek, vagyis ha a Prolog saját következtetési mechanizmusa nem lenne elegendő, akkor metaintepreterrel megvalósíthatók.

Logikai programozás és metaszintek

Alapfeltételezés, hogy logikai eszközök kezelését csakis logikai programozási nyelven: a gyakorlatban a Prolog valamelyik dialektusában érdemes megvalósítani. A Prolog következtetési képességei azonban elégtelenek – azok kiterjesztésére mindenképpen szükség van.

Egy logikai következtető rendszer a konkrét alkalmazói adatok kezelése mellett azok modelljét is adatként kezeli, folytonosan módosítja, fejleszti. Ezért a tudástárban legalábbis a modell modelljét, a metamodellt kell beprogramozni. A metamodell már elszakad az alkalmazói világtól, és a modell logikai szerkezetét, a modellelemek összefüggéseit írja le.

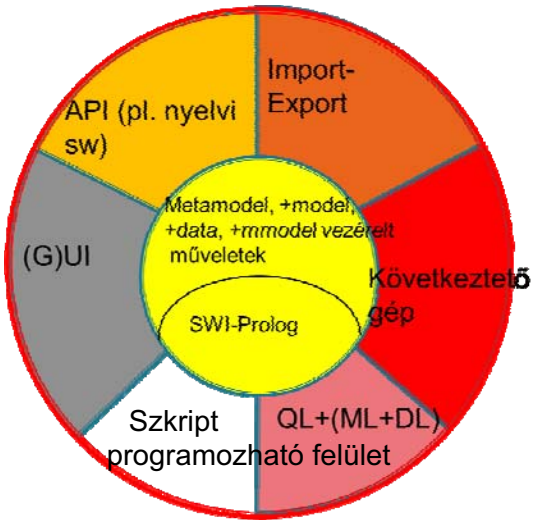
Azért, hogy a logika szerkezet maga is kellően rugalmasan fejlődhessen, célszerű magát a metamodellt sem rögzítetten beprogramozni, hanem legalábbis elválasztva,

[1] A szerzőket e cikk alapjait jelentő kutatásaikban és a konferencia-részvételben a TÁMOP 4.2.2.C-11/1/KONV-2012-0005 (Jól-lét az információs társadalomban) kutatási projektum támogatta.

adatszerűen leírni, és rajta általános algoritmusokat kidolgozni. Erre a megközelítésre szintén a Prolog nyelv a legjobb választás.

A Neumann elvű számítógépek sikere, de a körülvevő élő világ is alátámaszthatja: a metaszintjeiket átmetsző rendszerek különleges fejlődési képesség lehetőségét zárják magukba. Megvizsgáljuk ezért azt, hogy a tudástár esetében a metaszinteket hol lehetséges és célszerű átvágni.

A tudástár felépítése és felületei



1. ábra: A szoftver felületei

A szoftver magja a Prolog nyelven megvalósított tudástár, amely az ANSI Prolog szabványhoz közelálló dialektusban, az SWI-Prolog rendszeren készült. A rendszer a külvilággal az egyes felületein keresztül érintkezik. A tervezett (és részben megvalósított) felületek a következők:

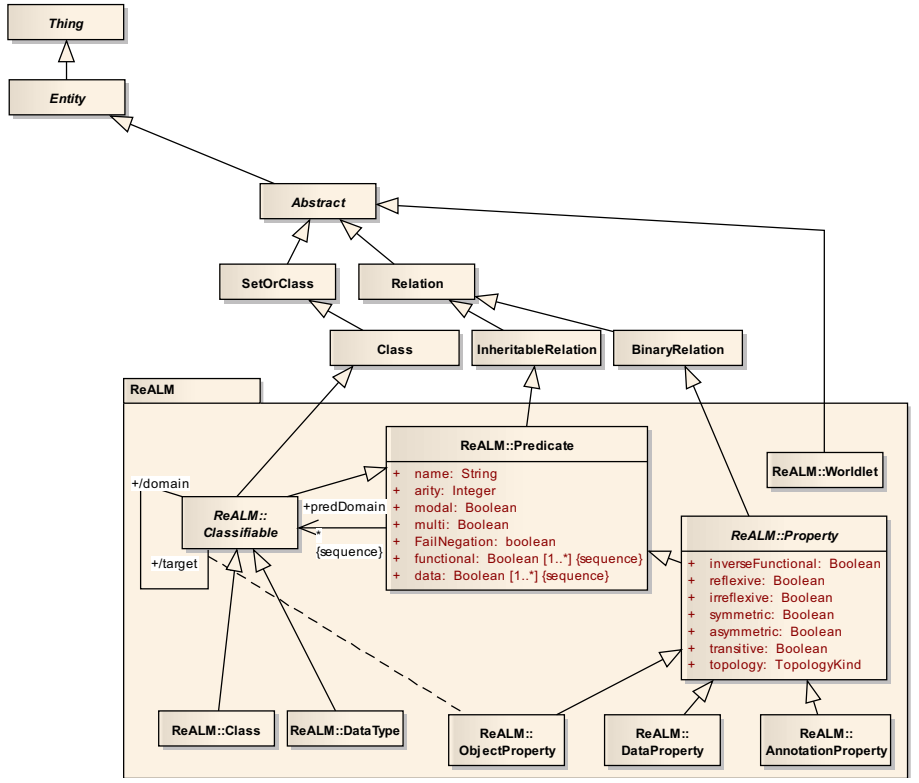
- a tudástárnak rögzített programozható felülete (API) van. Ehhez férhetnek hozzá a nyelvi feldolgozó szoftverek, pl. a ℜeALIS elemző, de a kezelői felület szintén ide kapcsolódik. A felület Prolog nyelvű, amit a megvalósítás adta módon lehet hagyományos programnyelvből meghívni. Jelenleg a Java kapcsolódás van használatban.
- a tudástárhoz egy Java Swing GUI felületet csatoltunk. Ez lehetőséget ad a tudástár adatszerkezeteinek (világmodell, adatréteg, modellréteg) a grafikus böngészésére és módosítására, valamint tesztelési célra egy közvetlen Prolog ablakot is biztosít. A Swing felület monolit felépítménnyel egyrészt könnyen programozható, másrészt a Java alaptechnológia lehetővé teszi a Swing felület lecserélését pl. Java Beans, JSP vagy más rokon technológiára akkor, ha ügyfél-kiszolgáló megoldás szükséges.
- a már meglevő tudástárak anyagának újrafelhasználása érdekében a rendszer OWL ontológiák beolvasására és mentésére is képes lesz. Ezekből egyelőre a beolvasás van az SWI-Prolog alatt használatos Thea csomag [6] segítségével, de offline módon megvalósítva. A Thea közvetlen Prolog formátumra fordít, amit jelenleg a Prolog `consult/1` műveletével tudunk beolvasni.
- a tárolt adatok lekérdezésére egy lekérdező nyelvi felület megvalósítása szükséges. Evégett az Object Query Language (OQL) nyelvet [7], mint az SQL objek-

tum orientált kiterjesztését, valamint az OWL ontológiák lekérdezésére létrehozott SparQL nyelvet [3] célszerű megvalósítani. Jelenleg csupán a Prolog saját eszközeit használhatjuk.

- az egyes részműveletek egymás utáni megvalósítására és gyors, dinamikus programozására valamilyen szkript programíró környezet használható. Jelenleg ez a lehetőség is csupán a Prolog saját eszközeit jelenti.
- Végül, de nem utolsósorban összetettebb következtetések elvégzésére következtető csomag csatolása is szükséges. Itt számba jöhetnek Interneten elérhető következtető csomagok, esetleg Prolog nyelven megvalósított csomagok, és a Prolog sajátmaga is, olyan feladatokra, amelyekre a szegényes képességei elegendőek.

Metamodell

Az import/export műveletek miatt igyekeztünk valamiféle szabványos ontológialeíró nyelvhez illeszkedő megoldást választani. Ezért a Szemantikus Web projektum OWL ontológialeíró nyelvét tiszteletben tartó, de azt bővítő metamodellt határoztunk meg:



2. ábra: A metamodell illesztése a SUMO155 ontológiához

- Megőriztük a 'Class' (osztály) fogalmat és a kétoldalú relációkat magukba foglaló 'Property' (tulajdonság) fogalmat, valamint a tulajdonságok felbontását annotációkra, amelyek String típusúak és megjegyzés-jellegű értékűek, valamint adattulajdonságokra, amelyek skaláris értékűek, és általános objektumtulajdonságokra.
- A tulajdonságok az OWL-ban rögzített metatulajdonságokat kapják.

A bővítés a következő újdonságok bevezetését jelentette:

- Bár kétoldalúval tetszőleges reláció is leírható, és az OWL ontológiai tervminták között is fellelhető hasonló célú minta, az osztályok és a tulajdonságok általánosításaként felvettük a tetszőleges argumentumszámú predikátum fogalmát.
- Tetszőleges argumentumszámú relációkra viszont a `domain` metatulajdonság értéke vektoros: minden argumentumsorszámhoz megadja az argumentum típusát. Minthogy a relációk esetén kitüntetett érték-argumentum (`range`) nincs, így nem is bonthatók annotációs, adat- és objektumrelációkra. Az argumentumokhoz viszont alaptípusok (`String`, `Integer`, stb.) is rendelhetők.
- Ugyanígy, általános relációk esetében a kétoldalúakra vonatkozó egyes metatulajdonságok rögzítése is értelmetlen. Kivétel a függvényszerű működés, amit a `functional` metatulajdonsággal, de annak vektoros értékével adhatunk meg, és azt fejezi ki: a többi argumentum értékének rögzítése esetén az adott argumentum értéke egyértelmű.

Felvettünk egy sor új metatulajdonságot, amelyekkel osztályok és tulajdonságok is jellemezhetők:

- `modal`: modálisan értelmezendő relációk kifejezésére.
- `negation`: a reláció felett explicit negációt használunk, mert nem elegendő a Prolog rendszerekből ismert kudarcalapú negáció (Negation As Failure) alkalmazása
- `multi`: a reláció nem tiszta kétértékű logikában értelmezhető

A metamodellt a metaszint-átvágás végett célszerű az alkalmazói modellbe, illetőleg a csúcsontológiába beilleszteni. Ezt a műveletet fogalomkonszolidációnak is nevezhetjük: a csúcsontológia fogalom- ill. tulajdonság-taxonómiájából levezetjük a metamodell fogalmait. Ha csúcsontológiának a SUMO155 szabadon elérhető ontológiát választjuk [3], akkor az illesztés az alábbi kapcsolatok létrehozását jelenti.

- a világocskakapcsolatok leírására a `subWorldlet` relációt használjuk.

`subWorldlet` $\subseteq$ **`abstractProperty`**

- relációszerkezetek leírására a `subPredicate` relációt használjuk.

`subPredicate` $\subseteq$ `collectionRelation`

- ennek részrelációi a relációleszármazást leíró `subProperty`, valamint az osztályleszármazást leíró `subClass` tulajdonságok.

`subProperty` ⊆ `subPredicate`, ill. `subClass` ⊆ `subPredicate`

- A `ReALM:Classifiable` egy közvetlen példány nélküli, ún. absztrakt osztály, amely a SUMO155 osztályfogalmából van levezetve, és magában foglalja a ℜeALM osztály-, valamint skaláris alap-adattípus fogalmát.

`ReALM:Classifiable` ⊆ `Class`,
`ReALM:Class` ⊆ `ReALM:Classifiable`,
`ReALM:Datatype` ⊆ `ReALM:Classifiable`

- A `ReALM:Predicate` fogalom a SUMO `InheritableRelation` fogalmának kiterjesztése.

`ReALM:Predicate` ⊆ `InheritableRelation`

- A `ReALM:Property` fogalom sajátmaga Predicate fogalmából, ill. a SUMO `BinaryRelation` fogalmából van levezetve.

`ReALM:Property` ⊆ `ReALM:Predicate`
`ReALM:Property` ⊆ `BinaryRelation`

- Végül pedig: a `ReALM:Worldlet` fogalom közvetlenül a SUMO `Abstract` fogalmának kiterjesztése.

`ReALM:Worldlet`⊆ `Abstract`

- Adatszinten egyetlenegy objektumpéldány, a gyökérvilágocska megadása szükséges.

```
Worldlet(root).
```

Prolog futási modell

A metamodelljével rögzített logikai nyelv alapvetően egy Prolog kóddá van leképezve, amelynek a formátumát a megfelelő futási modell rögzíti. Ez a magasabb rendű vagy nem klasszikus logikai szerkezeteket elsőrendű logikába képezi le. Ennek megfelelően a következő átalakítások történnek:

- A közismert logikai alapműveletek Prologban közvetlenül is ábrázolhatók.
- Az egyes logikai metaszintek számára külön Prolog modulokat használunk (`model`: a modelszint, `ontology` az adatszint, és `metamodel` a metamodell számára).
- A modalitást egy külön Prolog argumentumban ábrázoljuk [4, 5]. Az itt ábrázolt modális címke szintaxisa magában foglalja a multimodális logikai szerkezet összes vonását. Vagyis a modális címke szintaxisa lehetőséget ad temporálisan, episztemikusan és deontikusan is modális állítások kifejezésére.

- A diszkrét vagy többértékű logikai értékeket egy külön Prolog argumentumban ábrázoljuk. A megfelelő Prolog hívás hamis jellege jelzi a teljesen lehetetlen eseményt, minden egyéb esetén a Prolog argumentum értéke jelzi a lehetségesség, ill. a bizonyosság mértékét.

A rögzített Prolog futási modell előnye az is, hogy a használt ontológiát is végső soron a Prolog `consult/1` műveletével töltjük be. Az ontológiára épülő esettanulmányok és egyéb példák szintén ugyanígy, Prolog formátumban készíthetők el és tölthetők be.

Korai következtetések

Korai következtetésnek azokat a következtetéseket nevezzük, amelyek valamiféle általános következtetési szabály (pl. öröklődés) közvetlen alkalmazásával keletkeznek. A korai következtetések tekinthetők az interpretálás helyett a tudásbázisba közvetlenül belefordított következtetésnek is. A korai következtetések általában a választott magasabb rendű logikai rendszer axiómáiból állnak elő. A ℜealKB rendszerben a következő korai következtetéseket valósítjuk meg:

- A modális világ szuperindividuális régiójában a világocskák felett öröklődés érvényes. Ez minden egyes predikátumhoz (vagyis többparaméteres relációhoz, tulajdonsághoz és osztályhoz) hozzávesz egy új szabályt, miszerint minden olyan dolog igaz, ami az ősvilágocskában igaz.
- A gyökérvilágocska felett található a mód nélküli világ, ahonnan gyökérvilágocska minden állítását örökli.
- Ha egy objektum egy osztály példánya, akkor példánya az ősosztályénak is. Az öröklődés ilyen megfogalmazása igaz modálisan és mód nélkül is.
- Ha egy példánypár vagy példány n-es példánya egy tulajdonságnak vagy egy predikátumnak is, akkor példánya az őstulajdonságnak, ill. az őspredikátumnak is.

Metamodell-vezéreltség

A modellvezéreltség azt jelenti: olyan műveleteket valósítunk meg, amelyek bár az alkalmazói példányokon dolgoznak, de paraméterként megkapják azokat az alkalmazói modellelemeket is, amelyeknek a példányai. Vagyis a megvalósított műveletek nemcsak az adattartalomtól függetlenek (mint minden tisztességes szoftver esetén), de a konkrét modelltől is. Így, ha a modell változtatása szükséges, akkor a futó kódok nem muszáj újraírni, a módosítást elegendő csupán a modellben megtenni.

Metamodell-vezérelt lehet egy modelltárház szoftver akkor, ha vagy egy általános modelltárházat tervezünk, amely egyidejűleg esetleg több metamodellű rendszert is kezelni kíván, vagy a metamodell maga sem rögzített, ezért a fejlesztés során metamodell-módosításokat is tekintetbe kell vennünk.

A ℜeALKB tudástár ilyen értelemben metamodell vezérelt, és a megfelelő metamodell-, ill. modellelemmel paraméterezett CRUD (Create, Read, Update, Delete) műveleteket valósít meg.

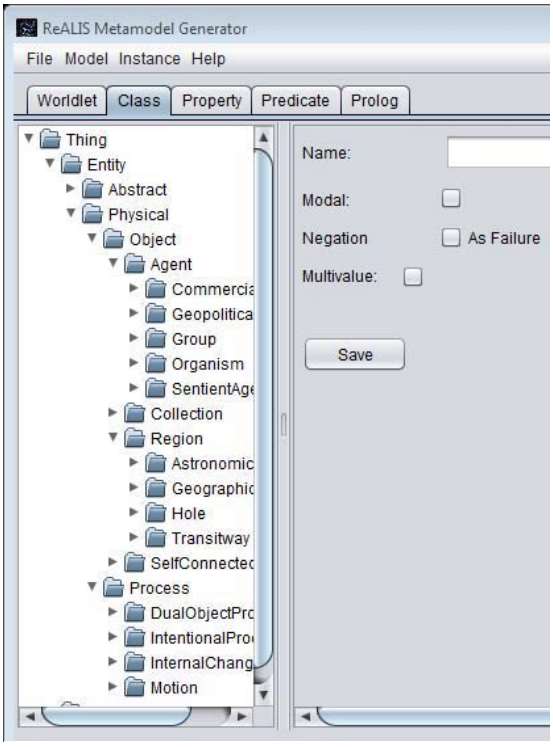
A Java Swing kezelői felület használata

eALKB rendszer lehetőségeit használja ki így a ℜeALM nyelv modellelemeinek, és a modellelemeknek megfelelő példányelemek kezelésére alkalmas. Ez a modell- és példányelemek létrehozását, módosítását, törlését valamint ellenőrzését jelenti.

A felület tartalmaz egy közvetlen Prolog végrehajtást lehetővé tevő ablakot is. Külön érdekesség a kétoldalú homogén relációkra vonatkozó általános böngésző alkalmazása, amely a reláció topológiájától függően alkalmaz vezérlőelemeket (Tree, List stb.)

Az általános böngésző legfontosabb alkalmazásai a világocska-szerkezet és az osztályszerkezet feletti böngészők, de ugyanígy alkalmazható pl. földrajzi objektumok között a részterület feletti viszonyra, vagy akár híres személyek családfájára.

Eredmények, továbbfejlesztés



3. ábra: A Java Swing kezelői felület

A vázolt rendszer fejlesztés alatt áll, létezik, működik, bemutatható. Amint egy viszonylag stabil és kerek változat elkészül, nyilvánosan elérhetővé kívánjuk tenni, és felajánljuk a tudományos közösségnek használat és továbbfejlesztés céljából.

A jelenleg legfontosabb célunk egy működő és stabil változat létrehozása és közzététele. Ha ez sikerült, akkor kerülhet sor a továbbiakra…

- hiányzó modulok megvalósítása és a rendszerbe illesztése
- ügyfél-kiszolgáló felépítmény megvalósítása
- egyes tételbizonyítók és megoldók rendszerbe integrálása
- konkrét lekérdezési nyelvek megvalósítása
- a jelenlegi bővített, de alap-

jaiban kétértékű logikai modell tágítása fuzzy irányba

Itt szeretnék köszönetet mondani a ℜeALIS projektbéli munkatársaimnak, Alberti Gábornak, Kleiber Juditnak és Károly Mártonnak a nyelvészeti információk önzetlen átadásáért, és a jól célzott, és egyben megfelelően adagolt, a cikk végső példányára is kiható megjegyzéseikért.

Hivatkozások

1. Alberti G.: ℜeALIS. Interpretálók a világban, világok az interpretálóban. Akadémiai Kiadó, Budapest (2011)
2. Kilián I.: A ReALIS tudástároló és következtető alrendszere In: Tanács A., Vincze V. (szerk.): IX. Magyar Számítógépes Nyelvészeti Konferencia 2013. Szegedi Tudományegyetem, Informatikai Tanszékcsoport, Szeged (2013) 225–235
3. Niles, I., Pease, A.: Origins of the Standard Upper Merged Ontology: A Proposal for the IEEE Standard Upper Ontology. In: Proceedings of Measuring Intelligence and Performance of Intelligent Systems Conference (2001)
4. Grosof, B. N., Horrocks, I., Volz, R., Decker, S.: Description Logic Programs: Combining Logic Programs with Description Logic. In: Proceedings of the Twelfth International World Wide Web Conference, ACM (2003) 48–57
5. Ohlbach, H.J.: A Resolution Calculus for Modal Logic. FB Informatik, University of Kaiserslautern, Germany (1988) (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.51.5003, letöltve: 25-Jun-12.)
6. Vassiliadis, V., Wielemaker, J., Mungall, C.: Processing OWL2 ontologies using Thea: An application of logic programming. In: Proceedings of OWL: Experiences and Directions (OWLED), CEUR Workshop Proceedings, Vol-529. (2009) (http://www.webont.org/owled/2009, letöltve: 25-Jun-12)
7. Cattell, R. G. G., Barry, D., et al.: The Object Data Standard: ODMG 3.0 Morgan Kaufmann publishers San Francisco, USA (1999)