dr. Dobrowiecki Tadeusz, Mészáros Tamás

A mesterséges intelligencia új területei: intelligens ágensek

egyetemi jegyzet (tervezet)

1998.

Budapesti Muszaki Egyetem
Méréstechnika és Információs Rendszerek Tanszék

A számítógépes hálózatok széleskörü elterjedése új kihívásokkal és új eszközökkel bovítette a mesterséges intelligencia repertoárját. A kihívások között elso helyen szerepel a számítógép-hálózati rendszerekben tárolt információ felhasználásának támogatása (szolgáltatás, elérés, feldolgozás, prezentálás), illetve a hálózat megjelenésével ismét elotérbe kerültek az intelligens rendszerek környezethez történo adaptivitásának és együttmuködési képességeinek kérdéskörei. Ezen új-régi területek a mesterséges intelligenciában egy új szemléletmód, az *ágens* alapú tervezés kialakulásához vezettek.

Az intelligens ágensek (*intelligent agents*) olyan számítógép programok, melyek önállóak, reagálnak a környezetükben beállt változásokra, együttmuködnek más rendszerekkel és céljaik elérése érdekében terveik szerint önállóan kezdeményeznek. Az ágens alapú szoftver tervezési mód elsosorban szemléletmódjában jelent újdonságot a korrábi technikákhoz képest; felhasznált eszközei között a mesterséges intelligencia korábban megismert módszereit találjuk. A jegyzet ezen új tervezési mód ismertetése mellett a mesterséges intelligencia azon alapterületeit is bemutatja, melyek szerves részét képezik az ágens alapú intelligens rendszereknek. Az elméleti ismeretek mellett hangsúlyt helyeztünk a gyakorlati problémák elemzésére, illetve alkalmazási példák programokkal illusztrált bemutatására.

Az elso fejezet a számítógép-hálózatok, és különösen az internet elterjedése okozta új kihívásokat foglalja össze a mesterséges intelligencia szemszögébol. A második fejezet rövid áttekintést nyújt e két terület kialakulásáról, kapcsolódási pontjaikról. A harmadik fejezet ismerteti az intelligens ágensek fogalomkörét, típusait, belso felépítésük alapjait, fobb komponenseiket és lehetséges alkalmazási területeiket. A negyedik fejezet tér ki részletesen az ágensek belso felépítésének, komponenseinek részletes ismertetésére, a szükséges mesterséges intelligencia alapismeretek bemutatásával együtt. Az ötödik fejezet röviden ismerteti a jelenlegi ágens megvalósítási technikákat, programozási nyelveket és implementációs környezeteket.

A hatodik fejezet az ágensek közötti kommunikáció és kooperáció területeit ismerteti, kitérve az ágensek közötti kommunikációs nyelvre, protokollra, és együttmuködési sémákra. A hetedik fejezet a mobil szoftver ágensek tématerületébe nyújt bepillantást, míg a nyolcadik fejezet az ágensek megvalósításával kapcsolatos biztonsági kérdéseket tárgyalja. Néhány, a jegyzet anyagához kapcsolódó, de szorosan nem oda tartozó területet a könyv végén található függelékekben ismertetjük.

Számítógép hálózatok és a mesterséges intelligencia

A hálózati számítási modell

A számítási módszerekben a gépeke összekapcsoló hálózat megjelenése egy új fejezetet nyitott. A soros vonali terminál illesztés (dialup) továbbfejlodése gépek közötti pont-pont adatkapcsolat kialakítását tette lehetové. Megjelent a kliens-szerver modell, mely szolgáltatás-alapon különítette el a felhasználót (kliens) és a szolgáltatót (szerver). A fizikai hálózati közeg felett kommunikációs protokollok (standardok) publikusak voltak, így a különbözo gyártók adott rendszerkörnyezetben kompatibilis implementációkat készítettek (Ethernet, TCP/IP, stb).

A hálózati protokollok általános szabványa az ISO/OSI hét rétegu hálózati modell.

Az Internet

Az internet a TCP/IP-re alapuló hálózatok összekapcsolt világhálózata. Alapvetoen a UNIX operációs rendszer mentén alakult ki, de ma már minden, hálózati szolgáltatást nyújtó operációs rendszer támogatja. A számítógépes világhálózatba bekapcsolt gépek száma rohamosan bovül, lényegében félévente megduplázódik (1997 nyarán kb. 20 millió gép volt bekapcsolva a világhálózatba). Egyre nagyobb teret kap az üzleti élet is, jelenleg foleg cégek közötti és cégeken belüli információ áramlást támogatja, de már kialakulóban van az eletronikus üzleti tranzakciók, sot az internetes fizetés rendszere is. A számítógépes hálózatok és az internet infrastruktúrális alapjain privát hálózatok (intranet, extranet) is kialakultak.

Nem csak a hálózatba kapcsolt gépek száma no rohamosan, de a gépek által nyújtott szolgáltatások fajtái és foleg a mennyisége is. Az interneten tárolt információ szinte egyeduralkodó hozzáférési formája a **World Wide Web (röviden: Web)**. Rohamosan no a Web szerverek száma, illetve az azokban tárolt anyagok mennyisége is (1997 nyarán körülbelül 650 ezer szerver, és 320 millió dokumentum volt). A Web egyszeru publikálási technikája és az internet felhasználók számának rohamos növekedése rengeteg céget, szervezetet és magánembert anyagok közlésére ösztönöz. Az exponenciális növekedés mindezen dimenziókban, amellett hogy egyre több anyagot tesz elérhetove egyre több ember számára, tovább nehezíti az internet használatát, a kívánt információ elérését.

Az internet magyarországon a kilencvenes évek elején kezdett kiépülni az akadémiai szférában, mára már az üzleti életben is jelentossé vált, illetve terjed a hétköznapi életben is. Idehaza is megjelentek az internet szolgáltatók, a tartalom szolgáltatók, egyre több cég, szervezet, állami intézmény helyezi el bemutatkozó anyagait, bonyolítja napi levelezését az interneten.

Az internet használatának egyelore csak az alapkultúrája kezd kialakulni, a mindennapos gyakorlata még nem. Még tapasztalt, szakmabeli emberek számára is nehezen használható szolgáltatásai vannak. A jelenlegi programok, interfészek a világhálózathoz a felhasználót adaptálják magukhoz, és csak kevéssé alkalmazkodnak a felhasználókhoz. A jelenlegi internetes alkalmazások - bár vannak ilyen irányú kisérletek - nem tudnak megbírkózni az óriási és dinamikusan növekvo információ mennyiséggel.

Mesterséges intelligencia

A **mesterséges intelligencia (artificial intellgience - AI)** az 1960-as években indult, az emberi agy funckionalitásának utánzását tuzte ki célul. Azóta egyes alkalmazási területeken már komoly ipari eredményei vannak (gépi látás, robotika, beszédfeldolgozás, mintafelismerés, szakértoi rendszerek, stb).

A számítógépes hálózatok elterjedésével a mesterséges intelligencián belül egyrészt megjelent egy új eszköztár, másrészt új kihívásokkal is szolgáltak e terület számára. Az új eszköztár megjelenése erosítette az elosztott, illetve az egymással kommunikáló intelligens rendszerek területeinek kutatását, illetve ilyen alkalmazások készítését. A kihívások elsosorban az internetet használók támogatásában jelentkeztek a következo fobb területeken: segíto az interneten, interfész internetes rendszerekhez, tanító rendszerek, probléma megoldás, információs infrastruktúra támogatás és virtuális projektek, szervezetek, rendszerek támogatása.

1. Agensek - hol és minek ?

Bár az ágensrendszer gondolata több területen is bevezetheto és sikerrel alkalmazható, a jelenkegi fo alkalmazási területük a globális számítógépes hálózatok környezete. Szoftver ágensek ma már szép számban léteznek, az igazi felvirágzásuk a globális számítógépes hálózatok következo fejlodési szakaszában várható igazán. Meg kell még jegyezni, hogy a mélytengeri, világur, stb. kutatásban alkalmazandó rendszefejlesztési technológiák a késobbiekben valószinukeg erosen meríteni fognak az ilyen tipusú rendszerek elonyös tulajdonságaiból, hiszen éppen az ilyen alkalmazásokban szükség lenne az erosen autonom, folyamatos és rugalmas muködésu rendszerekre.

A globális hálózatok komplexitásának és méreteinek elvárt drasztikus novekedése megköveteli, hogy az információs eroforrásokhoz való hozzáférés minél egyszerubb és hatékonyabb legyen. Az alakuló információs infrastruktúrák (pl. Egyesült Államokban az un. National Information Infrastructure - NII) eloreláthatóan a hálózatba kapcsolt információs adatbázisok millióit fogják tartalmazni. Az alapveto igény a gyors hozzáférés lesz az igen bonyolult és terjedelmes információhoz, pl. az orvosi képinformációhoz, az interaktív szimulációs lehetoségekhez, a számítógépes (digitális) könyvtárokhoz, a multimedia tananyagokhoz, és a hasonló közérdeku információs tárakhoz.

Becslések szerint a központi szerep az intelligens interfejszeknek fog jutni. Az ilyen, egyszeru és hatékony hozzáférést biztosító rendszerkomponensekre szükség lesz a tudósítástárok, orvosi képtárok, kormányinformáció, elektronikus kereskedelem, elektronikus bankok, multimedia könyvtárok és oktató rendszerek, zene, film, interaktív szórakozás, tudományos termékek, on-line adatbázisok és más hasonló rendszer esetén. Az intelligens interfejszektol elvárjuk, hogy legyenek:

?? multimodális felépítésuek (ablakok, grafika, beszéd, gesztusok, mozgó film, ...),
?? célorientált muködésuek, és
?? beágyazottak (3 D + virtual reality).

Az intelligens interfészek széles köru és minoségileg új információs szolgáltatásokat fogják biztosítani, így pl.:

?? az adat és tudásmenedzsmentet (új indexelési sémákat),
?? integráló és fordító szolgáltatásokat,
?? tudásfelfedezo (knowledge discovery) szolgáltatásokat, és
?? hálózatbiztonsági szolgáltatásokat (személyi, kereskedelmi információk, ...).

Az igazi technikai kihívást a kivánt egyszeru használat megvalósítása jelenti. Manapság még mindig a felhasználó az, aki adaptálódik a számítógéphez, és nem megfordítva. Egy NII interfejsztol viszont elvárjuk, hogy a felhasználót minél eroteljesebben támogassa és legyen természetes, segítokész es eltakaró. Az ilyen interfejszhez vezeto természetes architektúra és számítási paradigma éppen egy intelligens ágens (lehet), amely rugalmas, együttmuködo és adaptálódó.

Az NII-be beágyazott intelligens rendszer alapveto 'muködési módjaát' a következoképpen el lehet képzelni:

?? a felhasználói célok szempontjából relevans információ megtalálása és
?? a megfelelo hallgatóság megtalálása a felhasználó által eloállított információhoz.

Ennek a megvalósítása az un. ‘narrow casting' (a MI egyik újabb kihívása), avagy az embereknek és az ágenseknek ama szük csoportjának az identifikálása, akik a hirdetett információban vagy szolgáltatásban érdekeltek, vagy megfordítva, egy ilyen szolgáltatást képesek nyújtani.

Az intelligens ágensekre jellemzo autonom, hosszabb ideju és valós-ideju viselkedés igen alkalmas eme feladatok megoldására egy információgazdag környezetben. A mai ágensrendszerekhez képest talán a legnagyobb és meghatározó különbség lesz az, hogy az információfeldolgozó aktivítás zöme nem ember-ágens között, hanem ágens-ágens között zajlik majd, így az intellligens interfejszeken tulmenoen fontos területté novi ki magát a multiágens kommunikáció és együttmuködés.

Érdemes végig tekinteni a már létezo, alakuló, illetve a csak jósolt alkalmazások körét. Intranet környezetben leglényegesebb a papir alapú információátvitel kiváltásával járó cégi információ köröztetése, a hozzáférések (körözvények elsajátitásának) figyelése (‘vajon legalább egyszer olvasta el a körlevelemet ?’), a virtuális szervezetek és cégek létesítése, az elosztott/ táv szimuláció vagy számítások, a távoktatás, a termékmenedzsment, az intelligens (célvezérelt) keresés (browsing) a cégi anyagokban, és sok más.

A Cyberspace-ben a lehetoségek még gazdagabbak. Beszélhetünk információt gyujto alkuszágensekrol (broker) (passzív indexelési technikák kiváltása aktív adatbrokerekkel), személyes, megbizható alkuszágensekrol (payment and delivery of services), Yellow-pages és Consumer report ágensekrol szukebb területek folyamatos monitorozására, kereso rendszerekrol, WWW robotokrol, információs szurokrol, Usenet News szurokrol, elektronikus levelezés szurokrol, információs szolgáltatásokról, mint pl. elektronikus könyvtárosokról és adatbáziskezelokrol, naplót figyelo operációs rendszer ágensekrol, elektronikus titkárokról (ld. pl. a Fehér Ház Cyberspace felületét), személyes titkárokról, FTP ágensekrol, stb.

Igen érdekes és fontos alkalmazás az un. legacy systems, avagy a korábban telepített rendszerek ‘ágensített’ változatai: pl. távközlési hálózatok menedzsmentje, számítógéppel vezérelt gyártás, szállítás szervezés, légi irányítás, orvosi rendszerek, betegfelügyelet létezo és kritikus feladatait végzo rendszerek területén. A változó környezeti/üzleti feltételek miatt ezeket a rendszereket rendszeresen frissíteni kellene. Az átirásuk azonban tulságosan költséges. Megoldás jelenthet egy 'agent wrapper' csomagolás, mely segítségével a régi rendszer az új rendszerekkel képes együttmuködni.

Digitális könyvtárak esetén az intelligens ágensek egész közössége fogalmazható meg. Tartoznak ide a 'user-interface' ágensek, a 'user profile' ágensek, a tervkészíto ágensek, adatbázis (információgyujto hely) interfejsz ágensek, az információt szolgáltató ágensek, a thesaurus ágensek, a taxonómia rendszer kezelését segíto ágensek, és az új, nem szokványos könyvtárszeru szolgáltatásokat megvalósító ágensek, mint pl. a dokumentum-belüli keresés és indexelés, a bonyolultabb információ lekérdezése (képanyag, szöveg, ...), a dokumentumok felkérésre történo összegzése, a több dokumentumegyüttes összegzése, a multimedia keresés és indexelés, a hozzáférés 'élo', vagy 'közel-élo' információhoz (folyóirat olvasó), stb.

Belépo kategoria a ‘mérnöki’ ágensek, pl. ágensalapú mérorendszerek, ágensalapú méréstechnikai szakértelem szerverek, ágensalapú gyári rendszerek. A mesterséges intelligencia szerepét a XXI sz.-ra megjósoló ARPA riport igen sok muszaki jellegu alkalmazást is sorol fel, pl. az:

- intelligens szimulációk,
- kiképzés (training),
- tanítás,
- ipari, kereskedelmi és katonai rendszerek,
- szórakozás,
- intelligens információs eroforrások,
- hálózati multimedia,
- technológiai adatbázisok,
- intelligens projekt vezetok
- tervezo teampartnerek (design associates),
- robot teamek,
- ember-gép kommunikáció több modálitásban.

Bizonyos témákhoz tartozó eredmények már megvannak, csupán a kedvezo fogadtatás kialakulására kell várni. Vannak itt azonban olyan mesterséges intelligencia alapkutatási feladatok is, melyek sikeres megoldására még sokáig kell várni.

3. Mesterséges intelligencia (MI) áttekintése

Mérnöki alkalmazások szempontjából legfontosabb a mesterséges intelligencia ama felismerése, hogy az intelligens (magyarán) racionális emberi viselkedés egy részét lehet megfelelo számítási modellekkel leírni és igy számítógépes rendszerekben reprodukálni. Az emberi intelligencia persze ennel több. Amit sikerrel gépi (az un. szakérto) rendszerekkel meg lehet valósítani, ez tipikusan a logikai érvelésen alapuló problémamegoldási készségünk. Érzékszerveink révén, szociális kölcsönhatásaink révén szerzett tudásunk, fizikai manipulálási készségünk reprodukálásában a mesterséges intellligencia még alig lépett elore, bár bizonyos konnekcionista kisérletek igen kecsegtetok.

Az emberi problémamegoldási készség elemzésénél fontos észrevenni, hogy az igazi problémát azok a feladatok jelentenek, amiknek bizonyitottan nincs vagy még nem ismert a polinomiális bonyolultságu algoritmikus megoldásuk. Az ilyen (exponenciálisan nehéz, NP-teljes) problémáknál a rendelkezésre álló szoftver és hardver fejlodési üteme nem tud általában lépést tartani a probléma felskálázásával, így ezek a problémák elvi szinten gyakorlatilag megoldhatatlanak. Járható út az ilyen nehéz problémák olyan közelíto megoldása, amely nem minding ugyan garantáltan muködik, az esetek többségében viszont a komplexitása az exponenciálisnál kisebb. Az exponenciális jelleg gyakorlati 'letörése' csak ugy lehetséges, hogy a probléma megoldásába minél több, a problémára vonatkozó tudást 'injektáljuk be', megfelelo módon reprezentálva és manipulálva azt. A mesterséges intelligencia módszerei tehát mind tudásintenzív módszerek és a tudásábrázolás és a következtetés alapveto fontosságú az intelligens rendszerek építése szempontjából.

Mesterséges intelligencia területen több formális módszerrel, programozási paradigmával és rendszerarchitektúraval találkozhatunk. Ez arra kell, hogy késztessen minket, hogy tanuljunk az egyes módszereket és megközelítéseket mérlegelni és a várható elonyok és a vállalandó hátrányok egészséges kompromisszumára törekedni.

3.1 Intelligens rendszerek - intelligens ágensek.

Régebben az intelligens rendszerek majdnem kizárólag elszigetelt, idoben korlátos muködésu 'stand-alone' rendszerek voltak. Manapság elotérbe kerültek egy környezetbe szorosan beágyazott, huzamosabb, és foleg igen autonom muködésu rendszerek. Az ilyen rendszereket, gyujto fogalommal, ágenseknek fogjuk nevezni. Egy ágens szenzorikus apparátusával folyamatosan érzékeli a környezetét, amely részére a megoldandó problémák forrása, de a megoldásukhoz szükséges információ (tudás) forrása is egyben. A szenzorikus információt az ágens összeveti a tárolt információval és rövidebb-hosszabb következtetés után megfogalmazza a környezetével szemben kifejtendo akciót. Az akció megfogalmazásában több szempont is játszhat szerepet. Lényegesek az ágens céljai, az akciók hatékonysága, az ágens ido-, memória- és egyéb eroforráskorlátjai. Akcióival az ágens befolyásolja a környezetét és igyekszik azt úgy módosítani, hogy annak állapota egyenértéku legyen a probléma megoldásával.

Egy racionális (ésszeru) ágens azt teszi tehát, ami a problémái megoldásához szükséges. De honnan tudja, mit kellene tennie ? Hiszen a 'mindenható tudás' lehetetlen és az akciók eredményeit elore nem lehet pontosan tudni a környezet elore nem megjósolható viselkedése miatt. A racionális viselkedés megvalósításához szükséges lenne tehát a:

- hatékonyságot, sikerességet mérni,
- érzékelési szekvenciara (történelemre) emlékezni,
- környezet sajátosságairól tudni, és
- tisztában lenni az akciók választékaval.

Ideális lenne a soron következo akció valamiféle algoritmikus kiszámítása, pl. az alábbi függvény megadásával:

$$\text{akció}(t_k) = F\,[\text{környezet}(t_k), \text{környezet}(t_{k-1}), \ldots \text{akció}(t_{k-1}), \ldots \text{világmodell}, \ldots]$$

Ez természetesen általában nem lehetséges a probléma bonyolultsága és az analitikus modellek hiánya miatt.

ágens	környezet	szenzorok	beavatkozó szervek
döntéstámogató rendszerek	felhasznalok	interaktív input perifériák	interaktív output perifériák
intelligens monitorozó rendszerek	figyelt objektum es felhasznalok	adatgyujto perifériák	output perifériák
intelligens szabályozó rendszer	ipari folyamatok	adatgyujto perifériák	fizikai beavatkozó szervek
gyártócella robotok	gyartocella	fizikai szenzorok	munkadarabot mozgató szervek
önallómozgásu robotok	terep	fizikai szenzorok	fizikai mozgás szervei
szoftbotok	globalis szamitogepes halozat	hálózati, op rendszer utasítások	hálózati, op rendszer utasítások
ember …..	.....	……..	……..

3.1. táblázat. Ágens paradigmával modellezheto intelligens remdszerek

3.2 Ágensek fajtái

Továbbiakban röviden vázoljuk fel néhány intelligens rendszer kategóriát az egyre nagyobb képességu rendszerek felé haladva. Látni fogjuk az összehasonlításból, hogy egyszerubb (butább) rendszer architektúrák nem szükségképpen rosszak. Az a fontos, hogy a rendszer architektúrális megoldásaiban jól illeszkedjen a rá háruló feladat sajátosságaihoz és a problémamegoldás kivánalmaihoz.

Egyszeru reflexív ágens

Az egyszeru reflexív ágens a 'reflex mozdulatok' rendszere. Egy ilyen ágensben az érzékelés 'megcimzi' az akciót és az azonnal ('gondolkodás nélkül') végrehajtásra kerül. Az akciók választéka és az érzékelés -akció asszociaciók a rendszer fejlesztési fázisában 'elo vannak keszítve' és a rendszerbe bevasolva. A fejlesztés nehéz lehet, mert minden eshetoségre elore kell gondolni. A rendszer nem képes adaptálódni, megjavulni, viszont (nagyon) gyors muködésu. Reflexív ágens ott vizsgázik le jól, ahol a környezet lehetséges állapotai kevés számban vannak és elore megbizhatóan feltérképezhetok.

3.2. ábra. A reflexív ágens architektúrája és muködése

Emlékezo ágens

Hacsak az üres akciót külön nem definiáljuk, a reflexív ágens köteles minden érzékelési ciklusában cselekedni. Mivel nem minden megfigyelés szükségszeruen egy új környezeti állapotot jelent, sok esetben az így kikényszerített új akció fölösleges. Célszeru tehát az ágenst memóriával ellátni, amiben tárolna a környezet megelözo állapotait és a saját korabban vegrehajtott akcióit. Az ágens multja alapján is 'számított' akciói feltehetoen (bár ez a tervezéstol függ) jobb alkalmazkodó képességhez, rugalmasabb problémamegoldáshoz vezetnek.

3.3. ábra. Az emlékezo agens architektúrája és muködése

Cél-orientált ágens (deliberatív ágens)

A cél (goal) ágensek világában egy olyan 'kivánatos' helyzet, amit hasznos valamilyen módon elérni. A cél-orientált ágens az ilyen kivánatos állapotok keresésébe bonyolódik bele, esetleg az akcióinak elozetes megtervezésével. Fontos megjegyezni, hogy ehhez az egyszeru memória már nem elegendo. A mindenkori környezeti állapotban az ágensnek el kell döntenie, hogy melyik akciója visz közelebbre a célhoz. Ehhez ‘tudnia' kell az akcióinak a környezetre kifejtett hatását és a kivánt célállapot felismerh eto jegyeit. 'Mérlegelnie' kell továbbá, hogy az akcióit milyen összetételben és milyen sorrendben hajthassa végre, amihez rendelkeznie kell a megfelelo tudásanyag valamilyen reprezentációjával és következtetési lehetoségekkel. A reflexív jelzotol megkülönboztetve, az ilyen ágenst delibera tív ágensnek mondjuk.

Deliberatív ágensekre jellemzo a lassúság, de nagy fokú rugalmassag is egyben, hiszen következtetési képességük révén az új helyzetekhez is tudnak alkalmazkodni.

3.4. ábra. A cél-orientált ágens architektúrája és muködése

Hatékonyság-orientált ágens

Az igazán intelligens ágensnek több célja is lehet, amit egymás után, felváltva, egyidoben, vagy akármilyen más módon probálja megvalósítani. Tekintettel, hogy a celokat több akciószekvenciával el lehet ált alában érni az ágensnek döntenie kell, hogy az egyes célokat melyik úton/módon érdemes megközelítenie és melyik céllal érdemes fogalalkozni eloszor. Ez felveti a hatékonyság és feladatdekompozició kérdését.

3.5. ábra. A hatékonyság-orientált ágens architektúrája és muködése

A feladat dekomponálása azt jelenti, hogy az ágens a feladatát kisebb porciókra bontja és azokat igyekszik külön-külön megvalósítani. A várható nyereséget legegyszerûbben úgy illusztrálhatjuk, hogy feltételezzük, hogy a probléma exponenciálisan nehéz, így az N nagyságú problémát 2^N idõt, memóriát, illetve más erõforrást igénylõ módszerrel meg lehet oldani. Ha a problémát M részre dekomponáljuk, M darab N/M nagyságú problémát kell megoldanunk, mindegyiket azonban csak a $2^{(N/M)}$ komplexitási szinten. A nyereség így 2^N helyett $M*2^{(N/M)}$, ami konzervatív számadatok mellett is óriási egyszerûsítést jelent. A feladatot tehát mindenképpen érdemes dekomponálni, ha erre lehetõség adódik, mert erõforrást nyerünk. A helyzet egyszeru, ha a feladat, jellegénél fogva, természetes módon 'esik szét' kisebb darabokra. Az esetek többségében a tökéletesen dekomponálható feladat nem létezik, aminek következménye, hogy az egyes részek megoldásából a teljes feladat megoldása nem all össze. Ennek tipikus oka, hogy a feladatbeli kölcsönhatások vagy szükos eroforrások miatt, a késobb megoldott részfeladat elrontja valamelyik korábban már megoldott részfeladat megoldását. A dekompoziciót tehát el is lehet rontani.

Gyakorlati szempontból fontos az un. majdnem dekomponálható problémák osztálya, ahol a dekompozicióból nyert részmegoldások a teljes megoldássá ugyan nem állnak össze, de az utólagos feldolgozásuk (‘összeragasztásuk’) és a teljes megoldás kialakítása így még mindig erõforrásban olcsóbb, mintha a problémát annak teljes nagyságában meg probálnánk oldani.

A fentiekbol látszik, hogy a hatékonyságának mérése segíti az ágenst abban, hogy elemezze és döntse, milyen módon végzett problémamegoldás számára a legelonyosebb. A saját hatékonyságra (sikerességre) emlékezni és a célokat az akciókkal asszo ciálni - út a tanulás és még jobb muködés felé, hiszen a mindenkori tanulás feltétele hatékonyságukban megkülönböztetni a megoldásokat.

3.3 Problémakörnyezet tulajdonságai és azok következményei

Szó volt arról, hogy az ágens által megoldandó problémák forrása maga a környezete (pl. úgy, hogy alapvetoen a pillanatnyi állapota nem az, amit az ágens kivánatosnak tart). A környezetének tulajdonságai közvetlenül kihatnak a problémák tulajdonságaira és befolyásoljak ily módon a problémamegoldás alakulását.

A környezet alapvetõ tulajdonsága (az elõbb megismert dekomponálhatóságon tulmenõen) a hozzáférhetõség. Hozzáférhetõ környezet megbizható és hiánytalan információt biztosít, ami akkor különösen fontos, ha a deliberatív következtetések érdekében az alapvetõen numerikus információt szimbólikus formába át kell alakítani (numerikus-szimbólikus konverzió), ami úgyis információvesztéssel jár együtt. Hozzáférhetõségi problémák bizonytalan és/vagy hiányos tudáshoz vezetnek. Ilyenkor speciális technikákkal ábrázolni kell az ágensen belül a tudás bizonytalanságát és ezt a fejlesztésnél, illetve a muködés közbeni következtetésekben figyelembe kell venni.

Fontos megjegyezeni, hogy a környezet hozzáférhetetlenségért maga az ágens is lehet felelos. Az ágens szenzorai meghibásodhatnak, elromolhatnak, a beavatkozó szervei szintén muködhetnek pontatlanul. Mindez ahhoz vezethet, hogy az ágens bizonytalan lesz a környezet valódi, vagy a beavatkozás utáni állapotát illetoen. Megoldása a problémának lehet redundans szenzorok kiépítése, az érzékelt adatok ellenorzése, illetve a saját akcióinak követése és a valódi hatásuk verifikálása. Mindez természetesen bonyolíthatja az ágens tervezését és növeli az eroforrások kihasználását problémamegoldás közben.

A környezetben lejátszódó folyamatok lehetnek determinisztikusak vagy véletlen kimeneteluek. Ilyenkor a környezeti tudás leírására probabilisztikus módszereket be kell vetni. A helyzet valamivel jobb, ha a

véletlen környezeti folyamatok megjósolhatók és véges állapotúak. A nem megjósolható helyzetekben a legigényesebben kialakított deliberatív készseg sem segít.

Epizód alapúnak nevezzük ezt a problémakörnyezetet, ahol a megoldandó probléma nem folyamatosan áll fenn, hanem független 'porciókban' jelentkezik. Epizód alapú az orvosi diagnosztizáló rendszer, esetleg a gyártó cella robotja, nem epizód alapúak pl. az intelligens riasztó rendszerek vagy egy ismeretlen terepen navigáló robotok. Folyamatos muködés több akció, állapot tárolásával terheli meg az ágens eroforrásait, várhatóan igényesebb tudásreprezentációt is kell alkalmazni.

Fontos tulajdonsága a környezetnek, hogy statikus, illetve dinamikus. Dinamikus környezet tulajdonságai es így maguk a problémák idoben változnak, megkövetelve, hogy a problémamegoldás tartson velük lépést. A véges ido okozta problémákról késobb lesz szó. Utolsó említendo szempont a környezet diszkrét, illetve folytonos volta. Folytonos környezet esetén a tudáskezelés alapveto en nehezebb, a folytonos lefolyású jelenségek modellezése és szimulálása miatt.

3.4 Reális megoldás - hibrid és hierarchikus ágens

Bizonyos problémakörben reflexív ágens jól vizsgázik, de más problémák esetén ugyanaz mondható más ágens tipusokról is. Azonban ha a probléma kellõen bonyolult, akkor tulajdonképpen ágensek ‘keverékére’ lenne szükség, mert majdnem mindig bizonyos beavatkozásokat reflexív módon érdemes lenne megoldani, az is biztos azonban, hogy lesznek olyan beavatkozások, amikhez a reflexív mûködés kevés.

3.6. ábra. A hibrid ágens architektúrája és muködése

Az igényesebb architektúra szükségszerûen egy hibrid hierarchikus ágens. A reflexív akciók végrehajtása mellett az érzékeléseket és a megtett akciókat el kell tárolni és elemzésnek kitenni. Fontos az érzékelésekbõl kiolvasni, hogy az eddigi akciók mennyire sikeresek és hogy a helyzet mennyire felel meg az ágens fejlesztésénél feltételezett helyzetnek. A reflexív ágensre ráépülõ deliberatív komponens feladata megvizsgálni, hogy a reflexív módon megvalósított akciók elegendõek-e a helyzet lekezeléséhez, hogy nem alakul-e a környezetben egy olyan trend, amely az eddigi érzékelés-reflexív akció asszociációk érvényességét veszélyeztetne és hogy nem kellene-e a célok eléréséhez a reflexív akciókat kibõvíteni új megtervezett akciósorozatokkal, vagy netán új reflexív akciókat megtervezni. A hibrid ágens reflexív komponense biztosítja annak viszonylag gyors mûködését olyan helyzetekben, amikor az akciókon ‘gondolkodni’ felesleges lett volna. Deliberatív komponense viszont lassabban, hosszú távon biztosítja az ágens adaptációs képességét a változó környezeti feltételekhez.

3.5 Reális megoldás - véges eroforrásokkal rendelkezo ágens

Az ágensek egy részénél fontos probléma az, hogy a mûködésükhez valamilyen fogyó erõforrás (pl. energia) szükséges, amivel gazdálkodni kell és aminek hiányát tudni kell fedezni. Az elemzés kedvéért legyen az ilyen erõforrás jelképe a villamos energia. Az ágens szenzorikus apparátusát ki kell bõvíteni a saját energiaszintet és a tölto állomás hozzáférhetoségét jelzo eroforrás szenzorokkal. Az ágensnek tehát nemcsak a környezetét kell figyelnie, hanem saját magáról is tudnia kell képet alkotnia, hogy mennyire képes (lesz) a problémával foglalkozni. Az 'én' érzékelése, az 'én' modell sokkal nehezebb probléma, mint az ágensen kivül létezõ környezeté. A nyereség viszont a még rugalmasabb mûködés, mert az érzékelt energiahiány sulyossága fontos tervezési tenyezõ, ld. pl. az alábbi általános mûködési szabályokat:

```
IF        szint1 < energiaszint < szint2
   THEN  legfeljebb 100 akció lehetséges még és lassan a töltõhely felé kell
           haladni,
```

de közben egyéb hasznos dolgot még meg lehet tenni.

```
IF        energiaszint < szint1
THEN  egyéb akciókra nincs energia és azonnal töltõhely
           felé kell sietni.
```

Az ‘én’ tipusú szenzorok problémája azok hatékony érzékelése. Az ágens erõforrásállapota általában nem gyors változású. Felesleges lenne tehát az erõforrások szintjét minden egyes mûködési ciklusban megvizsgálni. A túl ritka ellenõrzés viszont azt a veszélyt rejti magában, hogy az ágens elnézi az erõforrások hiányáról idejében tanuskodó jelzést és feleslegesen gyakran kritikus helyzetekbe kerül, amikor is az eddigi problémamegoldást azonnal meg kell szakítani és az erõforrások felfrissítésének utána nézni.

Az erõforrás szenzor kialakítása lehet passzív vagy aktív. Passzív szenzor az erõforrás megapadásánál automatikusan jelzést (logikai megszakítást) küld, amit az agens észrevesz és ennek megfelelõen reagál. Passzív szenzorral az az alapvetõ probléma, hogy a csak egyfajta erõforrásszintrõl riasztó szenzor tulságosan rugalmatlan mûködéshez vezet, több szenzor hozzárendelése több lehetséges erõforrásállapothoz mûszakilag nem gazdaságos megoldás. Aktív szenzorra az agensnek idõnként magától rá kell néznie. Ily módon az erõforrás állapotáról folyamatosan szerezhet információt és így sokkal rugalmasabban kapcsolhatja össze a cél-orientált mûködést a hiányfedezésével. Probléma, hogy a szenzorra 'ránézni' egy külön akció, amit a többi akció sorozatába be kell helyezni.

3.6 Problémamegoldó stratégiák és architektúrák

Továbbiakban az ágens deliberatív komponensének kialakításával foglalkozunk. E célból megvizsgáljuk, hogy a problémamegoldás milyen általános stratégiái ismertek és azok milyen architekturális ötletekkel valósíthatók meg. A lehetséges stratégiák közül három legfontosabbat emelünk ki. Az elsõ kettõ gyakorlatilag elterjedt és sok változatban termékként kapható (szakértõ shell rendszerek) már, a harmadik az elõnyõs tulajdonságaiban az elsõ kettõn tultesz, de még kisérletinek mondható.

Tekintettel, hogy a mesterséges intelligencia területén a közönséges értelemben nem algoritmizálható, illetve exponenciálisan nehéz problémákkal van általában dolgunk, az alapvetõ problémamegoldási stratégia keresés lesz a probléma lehetséges megnyilvánulásait tartalmazó problématérben. Az ilyen keresés alapvetõen exponenciális komplexitású, hacsak vezérlésébe a problémára jellemzõ tudást be nem visszük. A keresést végzõ architektúra az un. szabályalapú (szakértõ-, produkciós) rendszer, amit több változatban, a konkrét tudástól kibélezve (shell system), kész termékként meg lehet vásárolni.

Táblaarchitektúra (blackboard rendszer, BB rendszer) több problémamegoldó komponens összehozása a közös problémamegoldás érdekében. Táblaarchitektúra egy elosztott, heterogén és elvileg parallelizálható rendszer jelent. Történelmileg ez volt az elsõ lépés az elosztott problémamegoldás felé. Táblaarchitektúrának több verzióját fejlesztették ki és az alaparchitektúrához kész programok is állnak rendelkezésre.

A taszkalapú megközelítés alapja az emberi problémamegoldás közben végrehajtott (szak)tevékenységek strukturája. Problémamegoldást itt a taszkstruktúra (taszkhierarchia) modellez, amely megadja az alapvetõ (generic) tevékenységek készletét és az egyes tevékenységek közötti relációkat. A taszkkal szervesen kapcsolódik a modell fogalma. Egy-egy taszk specifikus tudásanyagból (modellbol) indul ki és úgyanugy egy specifikus tudásanyagot eredményez, így a megközelítés alapvetõen tudásban heterogén, viselkedés- és modell-orientált. A taszkalapú megküzelítes univerzális problémamegoldási paradigma, de különösen fontos a nagy bonyolultságú problémák esetén, ahol az elõbbi tradicionális megközelítések csõdöt mondanak. Alapvetõ probléma azonban, hogy a taszkdekompozició módszertana nem teljesen formalizált és a taszkalapú architekturák terén nincsenek még általánosan elfogadott megoldások.

3.7 Keresés és a szabályalapú rendszerek

A hagyományosan szimbólikus (nem konnekcionista) mesterséges intelligencia alapveto megközelítése, hogy csak olyan 'intelligens viselkedéssel' foglalkozunk, ill. érdemes foglalkozni, amire ráhuzható a 'problémamegoldás' jellege. A problémamegoldás elemzése elvezetett egy félig-meddig formális és igen absztrakt felfogásig, amelybol viszont sikerült származtatni egy teljesen új számítási modellt és számítógépes rendszerarchitektúrát, amely alkalmas lett a problémamegoldás jellegu feladatok modellezésére.

Milyen alapveto felismerések születtek meg az emberi problémamegoldás terén és segítséget nyujtottak a problémamegoldási készség gépi megfogalmazásában ? Mindenképpen az, hogy:

?? a megfeleloen választott absztrakciós szinten maga a probléma, a kivánt megoldása, valamint a megoldás során keletkezo "félig megoldott" problémák azonos apparátussal irhatók le, és hogy
?? az igazán bonyolúlt problémáknál (a megfelelo absztrakciós szinten) a problémamegoldás "kisebb", "jól definiált" lépések (muveletek) sorozataként áll elo, amitól a probléma folyamatosan a megoldásba alakul át. (pl. poziciós játékok, diagnózis, matematikai bizonyítások, stb. de úgyszolván minden eloirás, muködési szabályzat, útmutató, amit a 'kevésbé okosok' részére készítik, hogy elosegítsék részükre a problémák megoldását).

A problémamegoldás formális leírása tehát a következoképpen fog kinézni:

?? **problématér** (keresési tér, problem space, search space),
elemei: - **kezdeti állapot** (a probléma),
- **végállapot** (célállapot, egy megoldás), és
- egyéb (a részben megoldott probléma),
?? **operátorok** (muveletek, szabályok (rules, productions)).

Vegyük észre, hogy a problémamegoldás ténye azonos azzal, hogy a problématérben kimutatjuk, létezik egy (feltehetoen optimális) pálya - szabály szekvencia - amely a kezdeti és a vég problémaállapotot összeköti. Egy ilyen megoldást az ágens fizikailag is végrehajthatja.

3.7. ábra. Problématér és a szabály felápítése.

A problématérben szabályok segítségével megoldást kereso rendszer az un. produkciós rendszer (szabályalapú rendszer, production system, rule-based system). Jellegzetes architektúrával és muködési ciklusával rendelkezik,

3.8. ábra. Produkciós rendszer architektúrája és muködési ciklusa.

Érdekes tulajdonsága e architektúrának és a megközelítés lényege, hogy architektúrálisan elkülönül a:
(1) konkrét esetre vonatkozó tudás (munka memória), az

(2) adott tipusú problémák megoldására vonatkozó tudás (tudásbázis), és a

(3) konkrét problémától független, valamiféle általános 'vezérlo, keresési'" tudás.

Ezek a komponensek, hagyományos algoritmikus feladatoknál mindig integráltan, összemosódva jelentkeztek. Fontos azonban, hogy:

?? a következteto gép felépítheto és tesztelheto, anélkül, hogy akármilyen probléma megoldására alkalmaznánk is a rendszert (egy ilyen termék az un. keretrendszer, vagy 'shell');

?? a szabályhalmaz (tudásbázis) lecserélheto, amivel egy adott feladatra megtervezett rendszer, ujjátervezés nélkül egy másik feladat elvégzésere is lesz alkalmas;

?? a rendszer muködésbe hozható hiányos tudásbázis mellett is. Tudásbázis muködés közben bovítheto (hiszen ez a következteto gép helyes muködését nem befolyásolja), így egy gyors prototipus késobbi használat közben 'magától áttervezodik';

?? a megközelítés szigorúan vett matematikai modellje egy (probléma)gráf és az elei mentén történo mozgás, azonban ezen az absztrakciós szinten majdnem minden gráfelméleti módszer, amit be lehetne vetni, NP-teljes, tehát exponenciálisan nehéz és nehezen skálázható fel. Magyarán, a tiszta matematika keveset segít;

?? a rendszer muködése lényegében keresés a problématérben, így érdemes az általános keresési algoritmusokkal is foglalkozni. Ha azok nem vezetnek eredményre, csakis egy heurisztikus keresés(vezérlés) marad;

?? a keresés mértéke, problémamegoldás során, egyenes kapcsolatba hozható a rendszer 'ügyességével', vagy 'butaságával'. Ügyes rendszer kevesebbet keres, de a jó heurisztikát (az esetek többségében helyes, de formálisan mégsem igazolható vezérlési tudást) nehéz megtalálni.

Produkciós rendszer muködése ciklikus. A probléma aktuális állapotában a rendszer megvizsgálja, hogy mely szabályok alkalmazhatók (szabályillesztés, rule matching). Ha egy alkalmazható szabály sincs, a problémamegoldás leáll, hacsak a rendszer nem képes visszalépni (backtracking) és valamelyik, korábban nem annyira kecsegtetonek itélt és nem kiaknázott alternatívával folytatni. Ha a szabályillesztés egyetlen egy szabályt talál, ezt a szabályt kell alkalmazni és tovább lépni. Tipikus helyzet azonban, hogy mindegyik problémaállapotban több szabály is alkalmazható (konfliktushalmaz), és így egy konfliktus helyzet áll elo (melyik szabályt válasszunk), amit valamilyen módon (konfliktus feloldási stratégiával) fel kell tudni oldani. Konflitusfeloldás eredménye az alkalmazásra kiválasztott, legjobbnak itélt szabály, amit a következokben az új problémaállapot kiszámításához fel kell használni (szablyelsutés, rule firing). Kérdés, hogy az ujannan kapott állapot célállapot-e már, mert akkor a problémamegoldást befejeztük. Ha mégsem, a muködést az új állapotban folytatni kell.

Az elobb említett elonyös tulajdonságai mellett produkciós rendszer architektúrának számos problémája is van, amire gondosan kell ügyelni. Szabályillesztés pl. egyfajta kompromisszum az igényes szabályábrázolás és a gyors szabályillesztés között. Ha a szabály feltételrészében egyszeru szimbólikus kifejezések szerepelnek, a kiértékelés gyors. Ha a tudás tömörebb kifejezésére a szabályokban változókat is vezetjük be, vagy netán a bizonytalan tudással is foglalkozunk, a feltételek illesztése komolyabb feladat és külön kell ügyelni a következmények bizonytalanságának propagálására is.

Szabálybázis lefordítása

Szabályillesztés fázisában a rendszer végig nézi a tudásbázisban lévo szabályok feltételrészeit. A szabálybázis végignézésére szükséges ido arányos a szabályok számával (ami komplex problémáknál igen nagy lehet) és az illesztési algoritmus bonyolultságával (ld. elobb). Bizonyos benchmark tipusú futtatások azt mutatják, hogy a szabályalapú rendszer idejének majdnem 90%-át ebben a fázisban tölti. Gyorsabb hardver, ügyesebb szoftver egyideig segít, az exponenciális jellegu problémák miatt hardver és szoftver nemigen tud lépést tartani a megnövekedo szabályszámtól származó lassítással. Az idoveszteség oka a bizonyos muveletek felesleges többszörös redundans elvégzése. Egyrészt a szabályok feltételrészeiben sok részfeltétel közös (miért ?), másrészt, tekintettel, hogy a rendszer kis lépésekben halad elore a problémamegoldásban, a szabály elsütésével kapott új problémaállapotban a korábbi lépésben a konflikushalmazba kigyujtett szabályok nagy része most is alkalmazható.

Bizonyos mértékben segít a szabálybázis indexelése. Ez azt jelenti, hogy a munkamemóriában és a szabályok feltételeiben eloforduló tényekhez kódokat rendelünk hozzá és belülük származtatjuk a szabályfeltételek kódjait is. Szabályillesztés fázisában a következteto gép, szimbólikus illesztés helyett, kiszámítja a munkamemória tartalmának pillanatnyi 'kódját' és ezzel megcímzi a tudásbázist. A módszer csak az egyszeru felépítésu szabályok esetén alkalmazható eredménnyel. Igazi átüto megoldás a szabálybazis un. 'fordítása' (RETE háló), ahol a szabálybázis szabályainak feltételrészeit és a munkamemóriát összevonjuk és redundánciamentes formába alakítjuk át. RETE háló egy olyan hálóstruktúra, melynek csomópontjai a szabályok feltételeiben eloforduló feltételvizsgálatok (de egy adott feltétel a hálóban csak egyszer szerepel), az élek viszont az információterjedési útak. A munkamemória megváltozását jelento adatok (korábbi következtetési ciklusból, vagy a rendszer környezetébol) a hálóba kerólnek, ahol addig terjednek, ameddig képesek teljesíteni a soron következo feltételvizsgálatokat. Ha valahol egy alternatív uton befutandó információ még hiányzik, az információ pillanatnyilag eltárolódik és a terjedés felfüggeszkedik. A hálón végig vezeto pálya egy adott szabály feltételrészének felel meg és az adotok e pályán történo végigfutása a szabály alkalmazhatóságát jelenti. A háló kimeneti pontjai az elsütendo szabályokat azonosítják. RETE háló alkalmazása nagyban felgyorsítja a szabályalapú rendszer muködését, elvész azonban a szabályok a futási idoben is elvileg elvégezheto módosíthatósága, elvész a munkamemória önálló volta és a következtetés is csak elore módon (ld. késobb) végezheto el. A RETE háló bevezetését tehát gondosan mérlegelni kell és csak akkor alkalmazni, ha a gyorsításból származó elony a részletezett hátrányoknál nagyobb.

3.9. ábra. Egy kis szabálybázis és a RETE hálója.

Keret probléma

Muködése közben szabályalapú rendszer sorra állítja elo a probléma újabb és újabb állapotait. Fontos eldöntendo kérdés a problémaállapotok ábrázolása. Amivel meg kell birkozni az un. keret probléma (frame problem), avagy mit csináljunk a lépésrol-lépésre (szabályról-szabályra) változatlanul meghagyott tudásanyaggal ? Ha egy állapot leírása mindig egy teljes leírás, a kívánt célállapotot könnyu felismerni (célállapot teszt egyszeru), az egyre növekvo információt azonban nehéz adminisztrálni. Tekintettel, hogy szabályról-szabályra az állapotleírás csak kisebb mértékben változik, alternatív megoldásnak az un. inkrementális állapotleírás kínálkozik, azaz csupán az állapotok közötti különbségek feljegyzése. A kisebb infomációtömeget könnyebb rendszerszinten kezelni, az ára a célállapotteszt sokkal nehezebb megvalósítása.

A keresés iránya

A 3.8. ábrán mutatott muködési ciklus az un. elorecsatolt, adatvezérelt (FW, forward chaining, data driven) muködés, aminek lényege, hogy a szabályalapú rendszer a probléma megfogalmazásától (kezdeti állapot) a megoldás felé (célállapot) halad. Szabályillesztésnél a szabályok feltételrésze kerül kiértékelésre, szabályelsütésnél viszont a szabály akciórésze írja elo, hogy mi lesz a következo problémaállapot.

A szabályalapú rendszer muködése megoldható másképpen is. Mivel egy konkrét probléma megoldása a probléma kiinduló megfogalmazását a megoldással összeköto szabályszekvencia (megoldási pálya) megtalálása a problématérben, elindulhatunk ezen a pályán visszafelé. Ez az un. hátra csatolt, célvezérelt (BW, backward chaining, goal driven) muködés. Most a kiindulási állapotnak a probléma feltételezett megoldását választjuk, a célállapotnak viszont a probléma eredeti leírását. Szabályillesztésnél azokat a szabályokat gyujtjük ki, amelyek az aktuális állapotot eredményeznek, avagy a szabályok akciórészei szerint tájékozódunk. Szabályelsütés a megelözo állapot visszaállítása a szabály feltételrésze alapján. Eltéro értelmezést kap a célállapotteszt is. A mindenkori új számított állapotot a bemeneti adatokkal (probléma megfogalmazása) össze kell hasonlítani. Amennyiben valamelyik lépésnél a soron lévo visszaállított állapot maga az eredeti probléma, a megoldási pályát megtaláltuk és a feltételezett megoldás biztosnak veheto. Ha a megoldást a bemeneti adatokig nem sikerül redukálni, a kiinduló feltételezett megoldás hamis és más megoldással kell probálkozni (ilyen sémát követ a reductio ad absurdum logikai bizonyítás).

3.10. ábra: Az elore- és hátracsatolt muködés.

Ha a szabályok feltétel- és akciórészükben azonos tudáselemek szerepelnek (pl. logikai állítások ellenörzése és logikai kijelentések) a rendszer muködése mindkét irányba képzelheto el. Ilyenkor dönto szempont, hogy az elore- illetve a hátrafelé megoldott problémának kisebb-e a komplexitása (problémagráf elágazási tényezoje, célállapotok számossága, stb.). Vannak természetesen tipikus FW (intelligens riasztás, monitorozás), illetve BW problémák (hibadiagnózis). A keresési irány megválasztását azonban döntoen az befolyásolja, hogy a szabályok akciórészei tartalmaznak-e mellékhatásokat (a munkamemória írásától különbözo utasításokat):

```
IF logikai feltétel
        THEN (logikai kijelentés + mellékhatás)
```

t.i. mellékhatások célvezérelten nem értékelhetok és így ilyen szabályokból a megelozo állapotok nem állíthatók vissza..

Problémamegoldás ügyességét fokozni lehet, ha valamelyik 'szélso' állapottól való kezdés helyett, sikerül a problématér közepében megtalálni a megoldási pálya valamelyik központi lépését és onnan kezdve a megoldást mindkét irányba folytatni. Ez az un. Means-Ends analizis, amihez azonban szükséges, hogy leírhassuk az egyes problémaállapotok közötti különbségeket és az ilyen különbségek és a megoldó szabályok közötti relációkat. A probléma megoldás menete ezek után a kezdeti és a célállapot közötti különbségek fokozatos eliminálása megfelelo szabályok hozzáadásával.

Konfliktusfeloldási stratégiák:

Hatékony problémamegoldás szempontjából a következteto gépen belül kúlcsszerepe van a konfliktusfeloldási stratégiáknak, azaz annak, hogy a mindenkori konfliktushalmazból milyen elvek alapján válasszuk ki az elsütésre szánt szabályt. A stratégia kialakítása függ a problémára vonatkozó tudás mennyiségétol. Beszélhetünk konkrét objektumfüggo, vagy állapotfüggo stratégiáról, amely a megoldási pálya mentén is tud változni. Stratégia lehet esetfüggo is, problémafüggo, végül akár problémafüggetlen is.

Érdekes nem triviális eset a problémafüggetlen konfliktusfeloldási stratégia. Legyen a munkamemória pillanatnyi tartalma (A, B, C, D) és legyen a tudásbázisban, többek közt az alábbi három szabály:

```
(1)  IF (A^B)   THEN H1
(2)  IF (A^B^D) THEN H2
(3)  IF (A^B^C^D)    THEN H3
```

Természetesen mind a három szabály bele fog kerülni a konfliktushalmazba, kérdés, hogy melyiket süssünk el ? A válasz a (3) szabály, mert annak feltételrésze legjobban illeszkedik a pillanatnyi állapot tudásanyagához, így a problémafüggetlen kiválasztási elv a 'leginkább specifikus' szabály eloször. Fontos megjegyezni, hogy BW tipusú muködésnél éppen az ellenkezo elv a nyero. Az elágazások kézbentartása végett a legáltalánosabb (legkisebb feltételrészu) szabályt ki kell választani.

Magyarázat generálás

Magyarázat generálás célja megkonstruálni a rendszer következtetésének a történetét, felsorolva a használt koncepciókat és érvelési lépéseket, hogy a felhasználó bepillantást nyerjen a rendszer muködésébe és ily módon készségesebb legyen akceptálni a rendszer döntéseit vagy ajánlásait..

Magyarázat generálás alapvetoen a 'miért ?' és a 'hogyan ?'-ra szeretne megadni a választ. Szabályalapú rendszerben a használt szabályok eltárolása ez viszonylag könnyen meg is valósítható. Elegendo csupán

valamiféle 'szabályformalizmus-olvasható természetes nyelvi forma' fordítón a használt szabálylánc ot 'visszapergetni' és a felhasználói felületen megjeleníteni. Legyen pl.

```
IF A THEN B
IF (B^C) THEN D
és
        A:   eroforrás rendelkézésre áll
        B:   taszk elvégezheto
        C:   taszk elvégzésére felkérés érkezett
        D:   ágens a taszkot végre fogja hajtani
```

a használt szabálylánc és tudáselemek fordítása, akkor a D eredmény generált magyarázata:

'Az ágens azért végre fogja hajtani a taszkot, mert az eroforrás rendelkezésre áll, így a taszk végrehajtható, abból és ebbol, hogy a taszkra felkérést is kapott, következik a taszk végrehajtása.'

Természetesen ez nem egy 'mély' magyarázat, amely egy szakértot is kielégítene. Igényesebb megyarázat generálás céljából a rendszert több tudással és jobb érvelési technikákkal (a problémakörben érvényes modellek, elméletek, törvényszeuségek, stb., ld. Késobb) ki kell egészíteni, ami elvezet az un. 2-ik generációs szakérto rendszerekhez.

A szabálybázis struktúrája

Alap kiépítésben szabálybázis egy olyan szabálytár, ahol az egyes szabályok egymással nincsenek kapcsolatban és a közöttük lévo összefüggéseket csakis a következteto gép tárja fel futás közben. Megfelelo ismeretek birtokában a szabálybázis még fejlesztés közben particionálható a problémamegoldás egyes fázisaihoz jobban illeszkedo porciókra. Ez természetesen megkönnyíti a szabályillesztés megvaósítását. Szabálybázis particionálása átviheto futási idobe is. E célból a szabályok közé az un. metaszabályokat be kell vezetni, melyek szerepe, elsütésükkor, nem a munkamemória tartalmán változtatni, hanem bizonyos szabálycsoportokat a további vizsgálat alól kivonni, vagy éppenségben az ilyen pillanatnyilag deaktívált szabályokat a muködésbe visszacsatolni. Metaszabályokkal oldható meg a konfliktusfeloldási stratégia futásideju módosítása is.

Megoldási pálya tervezése - kereso algoritmusok

Ha a konfliktusfeloldási stratégia eset-, vagy problémafüggetlen, ez azt jelenti, hogy az elsütendõ szabály (keresési irány) megválasztása ugyanilyen módon történik a problémamegoldás egész ideje alatt. Ez nem más, mint a gráfelméletbõl ismert gráfban keresõ algoritmus és így a gráfelméletbõl további ötleteket lehet meríteni. Keresés megvalósításánál két kérdésre kell választ adni:

?? hol tartunk, milyen áron jutottunk idáig, és
?? merrefelé a legjobb ?

Gráfelméleti keresési módszereknek két csoportját meg lehet különböztetni. 'Gyenge' módszerek (weak methods) olyan keresési eljárások, amelyek idoben vagy térben (memóriában) exponenciális komplexitásúak. Ide tartozik alapvetoen a véletlen (random), szélességi (breadth-first) és mélységi (depth-first) keresés. Iteratív mélyíto (iterative deepening) keresés igyekszik kiaknázni a szélességi és a mélységi keresés elonyeit, minimalizálva e módszerek hátrányait. Egyenletes költségu (uniform cost) keresésében figyelembe lehet venni az akciók, szabályok kiértékelésének költségét, amivel hatásosan modellezn i lehet az intelligens rendszer eroforrásigényét.

Heurisztikus keresési eljárások (heuristic search) numerikus távolsági fogalomra építenek. Ha az aktuális problémaállapotból nyíló haladási irányokat aszerint tudnánk minosíteni, hogy melyik van legközelebb a célhoz és így a keresést vezérelni, idot, memóriát és egyéb eroforrást sporolnánk, ha a legkecsegteto irányba folytatnánk a megoldás keresését. A lokálisan legjobb irányba halad a hegymászó (hill-climbing) módszer, de áldozata lehet a lokális minimumoknak és plateau-oknak. Az összes eddigi félretett alternatívára (globálisan) emlékszik és választ közülük a legjobb-eloszor (best-first) keresés, illetve ennek az un. A^*, illetve iterative mélyíto ITA^* változata. A^* keresés az un. totális pályaköltséget használ a problémaállapotok megitélésénél:

$$f' = h' + ? g_I$$

ahol
?
? g_I hatékonysági jellemzo (mennyibe került a megoldás)
h' a cél közölségére jellemzo (mennyire messze vagyunk)
f' a vizsgált állapoton átvezeto totális pálya becsült költésge.

Több keresési algoritmus létezik, azonban fontos megjegyzeni, hogy a keresõ algoritmus mindig összefügg a problémagráf tipusával. Az elõbb felsorolt algoritmusok mind az un. VAGY-gráfban keresnek, ahol az adott problémaállapotból kiinduló irányok értelmezése VAGY megoldási alternatívák. Vannak más tipusú gráfok is, pl. ÉS-VAGY gráfok, ahol egészen más elvek szerint kell keresni, bár ilyen gráfok megjelenése konkrét problémáknál ritkább. További fontos megjegyzés, hogy e módszerek, komplexitásukban, nagyon érzékenyek a h' távolság hibájára, ami gyakorlatban mindig van jelen.

Megoldási pálya tervezése - tervkészítés

A problémamegoldó szabályszekvencia az un. tervezéssel (planning) is megkereshetõ. A megközelítés lényege, hogy nem a problématérben keresünk a problémaállapotok között, hanem a tervek (szabályszekvenciák) terében. Egy 'üres' tervbõl indulunk ki, amely csak egy fiktív kezdeti és véglépést tartalmazza, ebbe épül be az eredeti probléma és a célállapot leírása. A tervet az akciók hozzáadásával folyamatosan bõvítjük, helyettesítve velük a tervben elõforduló még nem teljesített feltételeket. A módszer véget ér, ha a célállapotot leíró minden feltétel és a módszer futása közben bevezetett egyéb feltétel alkalmas akcióval le lesz fedve. Bõvebben ld. Tervkészítés fejezet.

3.8 Táblaarchitektúra és a problémadekomponálása

Táblaarchitektúra (blackboard system, BB system) a dekomponált problémák elosztott, de kommunikációmentes megoldásának egyik eszköze. Bár a módszer általános problémák esetére fogalmazták meg, igazából a hierarchikusan dekomponálható problémákra vizsgázik le jól (egy problémát hierarchikusan dekomponálhatónak mondjuk, ha a probléma több absztrakciós szinten megfogalmazható és az egyes szintekhez tartozó megoldások egymásra épülnek). A módszer jellegzetessége, hogy az elosztott komponensekre semmilyen tekintetben elõirást nem tesz, ami igen rugalmas, heterogén architektúrákhoz vezet.

Honnan kapta a módszer a nevét ? Tételezzük fel, hogy csapatosan egy kirakós játékot ki szeretnénk rakni. Egy nagy teremben vagyunk, a terem falán egy nagy tábla, minden bentülo látja. A tábla felülete ragadós, hogy a játékdarabok maradjanak fel rajta. Minden játékos kap a képdarabokból és mindenki ismeri a kirakandó eredeti képet. Tegyük fel, hogy önkényesen egy kiinduló képdarab kerül fel a táblára. Mindenki csendben nézi a táblát, nézi a saját képdarabkészletét és ha egy illeszkedot talál, kimegy és táblára felragasztja. Fontos észrevenni, hogy:

- a teremben ülo szakértok (akik ismerik a képet) a tábla állapota alapján aktivizálódnak,
- senkivel nem kell "beszélgessenek" (nem is teszik), az 'információt' (részmegoldásokat) csak a táblán keresztül cserélik,
- a megoldás a táblán lépésrol-lépésre alakul ki - rendezetlenül (!).

Táblaarchitektúra komponensei

Tábla (BB, blackboard):

Egy globális adatbázis, amely tipikusan hierarchikusan van elrendezve és mindenféle adat tárolására alkalmas. A megoldó rendszerkomponensek közötti információcserét tesz lehetové, kommunikáció nélkül. BB-nek több dimenziója lehet, tipikusan azok az ido lefolyása, a vizsgált koncepciók absztrakciós szintje és a vizsgált koncepciók bizonytalansága.

Tudásforrások (knowledge sources, KS)

Tudásforrások a szakértok. Céljük közremuködni a megoldásban, úgy hogy a BB megfigyelt állapota és a saját tudásúk alapján ujabb BB elemeket képezzenek. Tudásforrásoknak képesnek kell lennük arra, hogy felismerjék azt a szituációt (a BB tartalmát), amelyben közre tudnak muködni. Ennek érdekében a tudásforrás tudásanyaga, tartalmát tekintve, egy óriási szabály mintájára, feltételtudásból és akciótudásból áll. Gyakorlati megvalósításnál feltételtudás egy un. stimulus frame, azaz az a hipotézishalmaz, amelynek megjelenése a BB-n a tudásforrást aktivizálja. Párja a response frame, ami egy stilizált leírása annak, hogy az illeto KS hogyan módosítaná a BB-t, ha történetesen o lenne az erre kiválasztott szakérto. Stimulus frame és response frame szerepet játszik a táblaarchitektúra muködési ciklusában.

Vezérlo (controller)

Az alap módszerbe elobb-utóbb egy vezérlo/monitorozó komponenst is be kell vezetni, amely a táblához való hozzáférést irányítani fogja. Vezérlo megkérdezi, ki tud hozzájárulni a megoldáshoz, begyujti a jelentkezéseket és kiválasztja, ki legyen az, aki elhelyezheti a BB-n a nála lévo képdarabot. A vezérlési

jellegu információ a rendszer figyelemösszpontosítására (focus of attention) szolgál és a vezérlo általában igen eros befolyással van a problémamegoldás folyamatára.

3.11. ábra. Táblaarchitektúra elemei.

Táblaarchitektúra muködési ciklusa

Legegyszeubben megfogalmazva a BB rendszer muködési ciklusa igen hasonlít a szabályalapú rendszer muködésére:
(1) az összes KS olvassa a BB-n megtörtént változásokat és értékeli, miben tudna hozzájárulni,
(2) a vezérlo eldönti, hogy melyik KS érdemes arra, hogy a BB-t módosítsa,
(3) a kijelölt KS dolgozik a BB-n.

BB architektúra, ciklusából adódóan, igen alkalmas valódi parhuzamosításra (1. lépésben) és a KS-eket terhelo feladatok bovítésére (2. és 3. lépésben). A BB-t nem módosító KS-oknak szabad idejuk van a tudásanyaguk karbantartására, tanulásra, stb.

Táblaarchitektúra számos elönnyel rendelkezik. Az alap ötletnél fogva modulárizált, dinamikus vezérlésu, adaptivítása egyszeruen biztosítható, és természetes módon konkurens. Nagyon fontos, hogy igen eltéro filozófiájú tudásanyagok integrálását tudja támogatni, u.a érvényes adatokra is, és támogatja az applikációs területen természetes hierarchiának megfelelo dekompoziciót.

3.9 Taszk és modell alapú megközelítés - 2-ik generációs szakértoi rendszerek

Szabályalapú rendszerek jól mûködnek, ha a vizsgált probléma a szabályokba zárt tudásanyaggal jól lefedhetõ. Ilyen rendszerek mûködése nehézségekbe ütközik és a produkált megoldás minõsége romlik, ha a probléma a tudásanyaghoz képest perifériális, ha az igényes problémamegoldáshoz nemcsak sok, hanem sokféle tudásra van szükség, vagy pedig ha a problémamegoldás során a szabályokba zárt sématikus tudás mellett az un. ‘mély’ fizikai, kémiai, biológiai, stb. elméleteket is kellene alkalmazni.

Ilyen nehéz problémák körében javasolt a tudásanyag réteges szervezése (KADS ajánlás):
?? domain réteg: problématerület relacióinak, objektumainak és koncepcióinak a leírása,
?? következtetési réteg: domain réteg objektumain elvégezheto következtetések,
?? taszk réteg: a következtetések szervezése és vegrehajtása, végül
?? stratégiai réteg: a különbözo taszkok összekapcsolása, sikertelen taszkvégrehajtás menedzselése.

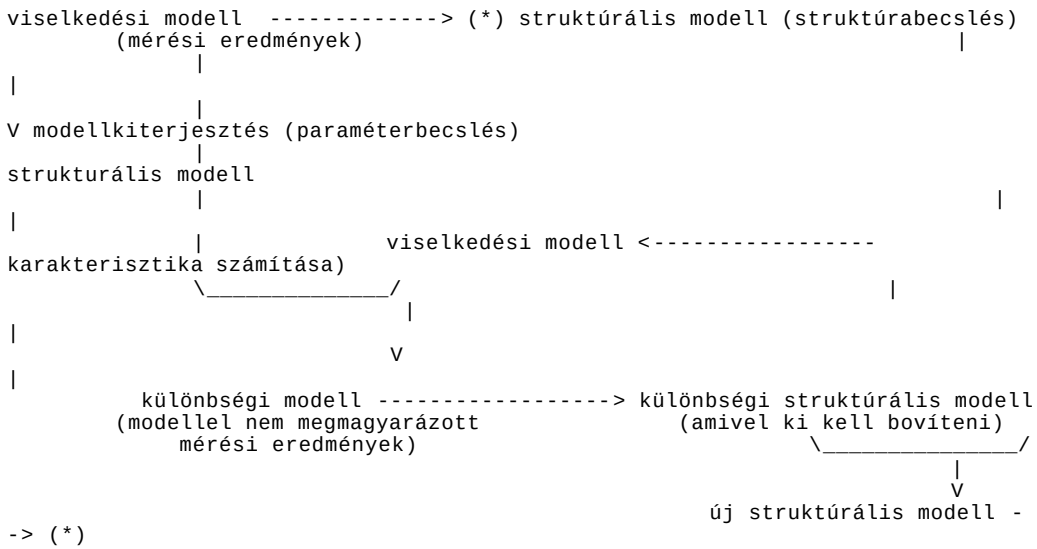
Ebben az új megközelítésben a központi fogalom a taszk. A feladat taszk szintu elemzésének eredménye a problémamegoldás lényegét kifejezo taszkstruktúra, melynek elemei a probléma-megoldásban eloforduló, különbözo absztrakciós szintu, un. elvonatkoztatott (generic) taszkok. Egy-egy taszk leírásához rögzíteni kell a taszk sikeres végrehajtásához szükséges domain tudás struktúráját, a bemeneti/ kimeneti információ tipusát (ezek az un. modellek), a taszkhoz tartozó következtetési módszereket és a vezérlési rezsimet.

A taszkstruktúra (hierarchia) származtatásának nincsen még lefektetett formális és a konkrét probléma domaintol független módszertana. Fontos, hogy a taszkok granuláltsága, a taszkstruktúra gazdagsága jól fejezze ki a humán szinten jelentkezo problémamegoldási stratégia összes lényeges mozzanatát. A taszkstruktúra származtatásának alapveto ‘trükkjei’ a magasabb absztrakciós szintu taszkok dekompoziciója alacsonyabb absztrakciós szintu taszkokra és a tipikus taszkok és tipikus tudásmodellek humán problémamegoldás szintjén megfigyelt párosítása:

```
specifikus kimeneti modell = taszk (specifikus bemeneti modell)
specifikus kimeneti modell = taszk (‘laza’ modellbe még nem szervezodo tudás)

pl.
funkcionális modell = magyarázatadás (viselkedési modell)
struktúrális modell = tervezés (funkcionális modell)
struktúrális modellek különbsége = diagnózis (viselkedési modellek különbsége)
viselkedési modell = predikció (funkcionális modell)
funkcionális modell = specifikálás ( )
összekapcsolási modell = konfigurálás ( )
temporális modell = tervkészítés ( )
```

Egy taszkon belül annak tudásanyagához illeszkedo következtetési sémát kell alkalmazni, a taszkstruktúrán belül azonban az elsodleges következtetési séma a tipikus taszkok révén megvalósuló modelltranszformáció. Vegyük példának a rendszer identifikációt (amely, mint probléma tulságosan összetett, minthogy ezt sikerrel lehetne szabályalapú vagy táblaarchitektúrával megoldani).



3.12. ábra. Rendszer identifikáció modelltranszformációi

Taszkalapú rendszer architektúrát a taszkstruktúrából kiindulva meg lehet valósítani. Elso lépés az adott problématerület taszkhierarchiának a megfogalmazása és a taszkokhoz tartozó modellek és módszerek identifikálása. Következo lépés a taszkstruktúra közvetlen reprezentálása az ágensen belül. A problémamegoldás folyamata lényegében a taszstruktúra 'végrehajtása'. Magasabb absztrakciós szinten ez tipikusan taszkdekompoziciót jelent, alacsonyabb absztrakciós szintu taszkoknál a taszkokhoz szükséges bemeneti modellek létrehozása, a taszkmódszerek lefuttatása és a taszkok által eredményezett kimeneti modellek interpretálása.

Fontos megjegyzeni, hogy a könnyebb formális modellezhetoség miatt (Petri-hálók, process algebra, stb.) a taszkstruktúra a szabályalapú rendszernél lényegesen egyszerubben verifikálható.

3.10. Tudásábrázolás

Mesterésges intelligencia alapvetõ premisszája, hogy az inteligenciát valamilyen számítási mechanizmus révén gépi úton meg lehet valósítani. Ez szükségessé teszi a mennyiségek, objektumok, azok közötti relaciók reprezentálását és a manipulálási procedurák megfogalmazását. Tudás-reprezentációknál tehát szimbólikus struktúrák létesítésérõl, módosításáról, és interpretálásáról van szó. Tudásreprezentációk tárgyalásánál választ kell adni ilyen kérdésekre, mint mi a tudás, hogyan lehet a tudást reprezentálni és hogyan lehet (kellene) a tudást manipulálni ?

A tudás

A tudás több fajtája különböztetõ meg és ez a megkülönböztetés nagyban segíti a problémák elemzését. Segítséget nyújt szintén ahhoz, hogy elkerüljük a reménytelenül nehéz problémák célül tûzését. Alapvetõen háromféle tudásról beszélhetünk: kifejezhetõ (expression) tudásról, heurisztikus tudásról (heuristics) és következtetõ tudásról (inference). Kifejezhetõ (expression) tudás az, amit viselkedéssel vagy kijelentéssel ki tudunk fejezni. Motorikus képességek (motor skills), érzékelés (perception), alak- és helyzetfelismerési tudás (gestallt) megmutatható, de nem leírható, így az ilyen tudást számítógépes rendszerekben gyakorlatilag majdnem lehetetlen reprodukálni. Egyszerûbb a helyzet a deklaratív tudással, ami tényekbõl és koncepciókból áll. Koncepciók világában rendelkezünk taxonometrrikus (osztályozó) sémákkal, ld. pl. XX. ábra, és hipotézisekkel, amik modellek, elméletek és módszerek formájában jelennek meg. Heurisztikák lehetnek probabilisztikus hátterûek, illetve tételekbe, folyamányokba foglaltak.

Nagyon fontos a következtetési tudás. Dedukció (deduction) a formális (logikai értéktartó, igazságtartó) következtetés eszköze. Olyan következmények származtatására jó, amelyek a premisszákból mindenképpen kõvetkeznek. Deduktív lépés pl. a logikában alkalmazott Modus Ponens, avagy annak belátása, hogy:

A ? B
A

B

Könnyû bebizonyosodni, hogy a lépést leíró logikai állítás: ((A ? B)^A) ? B mindig igaz, az A és B értékétõl függetlenül.

Indukció és abdukció nem formális lépések. Indukció lényege egy kijelentés megfogalmazása a példák valamilyen halmaza alapján, így a halmaz terjedelme és tulajdonságai meghatározzák az indukció tipusát. Gépi szinten a leggyakoribb eset a véges számú példa általánosítása, mint pl. egy ágensközösségben az ágenseket leíró:

Deliberatív (Ágens-1)

Deliberatív (Ágens-2)

Deliberatív (Ágens-3)

.....

Deliberatív (Ágens-k)

.....

Deliberatív (Ágens-N)

állításhalmaz helyettesítése a konkrét probléma megoldása kedvéért egyetlenegy:

? x . Ágensrendszer (x) ? Deliberatív (x)

állítással.

Az indukció fontos, mert sok tény helyett egyetlen egy tényt fogalmaz meg, amiból kifolyólag a probléma mérete csökken, és az exponenciális jelleg kevésbé zavaro, de a lépés formálisan természetesen nem igaz.

Sem formális, viszont nagyon fontos az abdukciós következtetés, amely formailag a deduktív Modus Ponens ‘megfordítása:

```
A -> B
     B
---------
A
```

A lépés nem formális voltát könnyu belátni, hacsak arra gondolunk, hogy tipikusan több implikációnak úgyanez lehet a konkluziója (A_1 ? B , A_2 ? B), de miért lenne ez a lépés annyira fontos ? A valós világ logikával való leírásánál logikai implikációval célszeru kauzális kapcsolatokat rögzíteni, máskülönben a tudásanyag ellenorzése, konzisztens megfogalmazása szenvedne csorbát. Tegyük fel, hogy egy fizikai jelenséget irunk le, természetes ekkor a:

```
'a rendszer X-állapotban van'
        ? ‘a rendszer megfigyelheto viselkedése Y'
```

megfogalmazás. De az életben zömében a megfigyelésekre vagyunk utalva, problémát jelent viszont egy-egy rendszer esetén, annak a belso állapotának a kiderítése. Ez a háttere a mindennemu diagnózis feladatnak ! A diagnosztikai problémamegoldás modellje tehát az alábbi abduktív lépés:

```
'a rendszer X-állapotban van'
        ? ‘a rendszer megfigyelheto viselkedése Y'
‘a rendszer megfigyelheto viselkedése Y'
-------------------------------------------------------
'a rendszer X-állapotban van'
```

Fontos még említeni, hogy egy intelligens rendszer tudása az un. zárt vagy nyított világot képezhet. Zárt világ esetén a rendszer minden lényeges tudással rendelkezik. Problémamegoldás egy rámutatás, hogy melyik, eleve ismert megoldás, érvényesül as adott pillanatban. Nyitott világ nem tartalmaz mindent, ami a probléma megoldásához szükséges. A rendszernek, feladatmegoldás közben, tanulnia kell, új koncepciókat is tudnia kell kialakítani.

Tudásreprezentációk

Tudásreprezentáció nem más, mint valamilyen matematikai (formális) apparátusnak a tudás leírására való alkalmazása. Formális apparátus kikötése nagyon fontos, hiszen másképpen a tudást nem lehetne a számítógépes rendszeren belül reprodukálni (beprogramozni, kódolni és a rendszer futásával a tudást manipulálni). Kapcsolódik ide egy érdekes hipotézis is, az un. fizikai szimbólumrendszer hipotézise (Simon,

Newell). E hipotézis szerint a szimbólikus struktúrák tárolása, módosítása olyan képességek, amelyek az intelligens viselkedés szükségszeru feltételei. Egy fizikai rendszer, amely azokkal rendelkezik (ilyen az ember, de ilyen a számítógépes rendszer is), eleve képes intelligens viselkedést felmutatni, hacsak a rendszer bonyolultsága megfelelo szintet elér. Célszeru tehát az egyre bonyolultabb gépi rendszerekkel kisérletezni (és így a mesterséges intelligencia kisérleti számítógépes tudománynak is fogható fel).

Tudásreprezentáció világában lényeges kérdések, hogy a reprezentáció kiterjeszthető-e könnyen az adott és/vagy más problématerületre, hol vannak a reprezentáció korlátjai, tárolja-e hatékonyan a szükséges információt, emberi szemmel vizsgálható-e könnyen, integrálható-e jól más információforrásokkal, gazdaságos-e a használata, támogat-e hatékony következtetési sémákat, stb. ?

A tudásreprezentáció megválasztásánál meghatározó a reprezentációs primitívek a megválasztása. Reprezentációs primitívek az alapveto épito elemek, minden más tudás azok összekapcsolásából nyert struktúrákkal lesz leírható. Ha a primitívek tulságosan elemiek, a tudás nagyobb részének a reprezentációja bonyolult lesz, ha viszont a primitívek tulságosan összetettek, a tudás ábrázolása hiányos és rugalmatlan lesz. Tudásreprezentációk világában alapvetoen két megközelítés ismert. Tényszeru tudáselemekre épülnek az un. logikai jellegu reprezentációk, az objektumok közötti relaciók, objektumtulajdonságok explicit kifejezésére (taxonometrikus tudás) épülnek az un. struktúrális reprezentációk.

3.13. ábra. Tudásreprezentációk áttekintése.

A logikai jellegu tudásreprezentációk sokasága azzal magyarázható, hogy a két logikai érték és a hagyományos matematikai logikák (itélet- és predikátumkalkulus) un. monoton volta (az egyszer igaznak bizonyított állítást továbbiakban mindig igazként kell kezelni) tulságosan nagy megkötés a valós problémák modellezésénél. Modális logikák az ido és lehetoség koncepcióját vezetik be, de igen általános szinten.

A logikai ábrázolással az az alapveto baj, hogy a valós probléma tulajdonságait leíró állításhalmaz axiómahalmaz szerepét tölti a tételbizonyítás szempontjából. Így minden változás a problémaleírásában (amely pl. a probléma idofüggo voltából, vagy a hiányzó tudásból ered) az axiómahalmazt és a belole korábban levezetett állításokat érinti. A pillanatnyilag igaz állításhalmaz felfrissítése brute force módszerrel (az összes állítás kitörölni, axiómákat felfrissteni és az új igaz tételeket levezetni), a logikai bizonyítás exponenciális jellege miatt, nem megy. Az igazi kihívás az, a RETE hálónál elhangzottakhoz hasonlóan, hogy ha a probléma idofüggése mérsékelt, egyszerre csak néhány axióma vált értéket és így a levezetett tételek nagyobb része tovább is érvényes. Probléma azonban, hogy a hagyományos logika a függoségeket nem adminisztrálja és a tételek ilyen szétválasztása lehetetlen, hacsak speciális állításadminisztrációt (Truth Maintenance System, TMS) nem vezetünk be.

Több értéku módszerek a bizonytalan tudás ábrázolására születtek. A spektrum lényegében a valószinuségszámítás, illetve vele rokon fogalmakra épülo módszerek és ujabban a fuzzy logika (Vita folyik arról, hogy a bizonytalanság ábrázolása a valószinuségen kivül igényel-e más módszert, vagy sem. Valószinuleg a legcélravezetobb megközelítés azt belátni, hogy hol a probabilisztikus, hol a fuzzy alapú megközelítés lehet az adott probléma esetén gazdaságosabb, illetve rugalmasabb). A probabilisztikus fogalmak körébol mesterséges intelligenciában legfontosabb a feltételes valószinuség és a Bayes-tétel. Feltételes valószinuség (logikai implikáció mintájára) megragadja a problémában rejlo bizonytalan kauzális kapcsolatokat, Bayes-tétel viszont megfordítja a feltételes valószinuségekben a feltételek szerepét és az abdukcióra hasonlóan segít a rendszer viselkedésébol következtetni a rendszer állapotára:

```
P (állapot ? viselkedés) = P (viselkedés ? állapot) * P (állapot) / P
(viselkedés)
```

A P (viselkedés ? állapot) és a P (állapot) valószinuségeket viszonylag könnyu a problémaéeírásából kinyerni, a tiszta valószinuségszámítást mégsem használják, mert a P (viselkedés) valószinuségnek nemigen van fizikai jelentése. P (viselkedés) helyettesítheto úgyan az elobbi mennyiségekkel (a teljes eseménytér felbontása), de ennek formális feltételei általában nem reálisak. A többi módszer a valószinuségszámítás egy-egy adaptációja, ahol az eredeti apparátus axiómáiból egyre többet forogunk le, cserében egyre kevesebb alkalmazható tételt, de 'élethubb' apparátust kapunk. Bayes-hálók csak az objektumok ismert kapcsolatai mentén számítják ki a feltételes valószinuségeket, Dempster-Shaffer elméletben pontszeru valószinuség már nincs is, hanem egyfajta ?min P, max P? intervallum és a valószinuségi eloszlásra emlékezteto tömegeloszlás nem eseményeken, hanem eseményhalmazokon van definiálva. Végül, történelmileg elso, MYCIN-féle bizonytalansági faktorok (certainty factor) a valószinuségre már alig hasonlítanak.

Struktúrális tudásreprezentációk körében tulajdonképpen egyetlenegy módszerrol, szemantikus hálókról (semantic nets) beszélhetünk. Szemantikus háló csomópontjai a probléma objektumai, élei az ilyen objektumok közötti relációk. Szemantikus háló speciális esete a tisztán taxonometrikus háló (de gondolhatunk akár egy objektum-orientált nyelvre is), amely a szuperosztályok, osztályok, alosztályok, ..., egyedek viszonyát mutatja. Szemantikus hálónak az a legnagyobb problémája, hogy a logikával ellentétben nem rendelkezik egy általános és formálisan verifikált következtetési sémával. Egyetlen szélesebb köru lehetoség az un. öröklodésnek és a bizonyos relációk tranzivításának a kihasználása, az azonban csakis a taxonometrikus hálóban lehetséges, az általános szemantikus háló ehhez túl gazdag.

Gyakorlatilag a legfontosabb struktúrált ábrázolás egy olyan háló, ahol az adott objektumra vonatkozó tudást magába az objektumba rejtjük (az eredeti szemantikus háló csomópontja egy 'matematikai pont', az objektum minden tulajdonságát a csomópont körüli hálóval ábrázolunk), és csak a fontos globális (pl. az öröklodést biztosító) kapcsolatokat hálószeruen ábrázoljuk. Ezek az un. keretek (frames), rekeszekkel (slots), ahol az információ eltárolódik. A keretek fontos tulajdonsága, hogy az objektumban táralandó tudásra nincs semmilyen eloirás, így mint tudásreprezentáció rendkivül rugalmasak, probléma, hogy a reprezentációhoz tartozó következtetési sémákat külön ki kell alakítani. Ha egy keret rekeszeiben ábrázoljuk az események tipikus lefolyását, az eseményekben résztvevo objektumokat (tárgyak, szereplok), illetve az egyes eseménysorozatok feltételeit, akkor az un. szövegkönyveket (scripts) kapjuk.

Reprezentációs problémák

A különbözo reprezentáció problémák körébol kettot emeljük ki. Az egyik nagyon fontos megjegyzés, hogy minden reprezentáció és implementáció, az eredeti probléma megfogalmazása szempontjából, torzítás is egyben.

3.14. ábra. A tudás reprezentálásával és a reprezentáció implementálásával kapcsolatos problémák.

A megoldandó problémát elsodlegesen természetes nyelven fogalmazzuk meg, és úgyanugy értelmezzük a problémamegoldás módszereit. A természetes nyelvben kifejezett tudás akármilyen formális reprezentációban való kifejezése mindig két problémát vet fel.

(1) A reprezentáció primitív elemei és azok manipulálási szabályai a tudás csak egy részét képesek reprodukálni. Ha a tudás valamilyen lényeges része így az ábrázolásból kimarad, a formálisan ábrázolt probléma megoldása már nem bizots, hogy az igényeinknek megfelel.

(2) Egy formális reprezentáció, éppen a formális volta miatt, rendelkezhet olyan belso törvényszeruségekkel, amelyek nem biztos, hogy a valós problémának akármilyen vonatkozását is tükrözik. Mégis szerves részei a reprezentációnak és épülnek a következtetési sémákba is, potenciálisan torzítva tovább a megoldást.

A másik megemlítendo probléma a logikai bizonyítás valós problémákra való alkalmazása. Egy logikai rendszer teljes, ha biztosított benne az igaz állítások algoritmikus bizonyíthatósága, eldöntheto, ha meg lehet tenni ezt a hamis állításokkal is. Itéletkalkulus teljes és eldöntheto, de a felépítéséból adódóan az igényesebb tudás ábrázolása nagyon körülményes és bizonyos problémamegoldásokat igen nehéz modellezni benne. A függvényekkel rendelkezo predikátumkalkulus igen eroteljes modellezo rendszer, teljes is, azonban csak félig eldöntheto. Más logikáknál a helyzet még inkább romlik.

A félig eldönthetoség azt jelenti, hogy létezik egy állításbizonyító algoritmus, amely igaz állítás esetén leáll és a bizonyítás tényt közli, hamis állítás esetén viszont nem rendelkezik egy jól definiált kilépési ponttal (pontosabban megfogalmazva predikátumkalkulusban algoritmikusan kideritheto egy állításhalmaz inkonzisztenciája, abból kifolyólag (1) konkrét problémamegoldáskor egy feltehetoen igaz bizonyítandó állítást negálunk és a probléma domaint leíró, feltétlenül igaznak tartott axiómákhoz csatolunk, (2) a teljes állításhalmaz így inkonzisztenssé válik, amit ki lehet deríteni, (3) mivel csak az ujannan hozzáadott állítás a gyanus, az hamis kell, hogy legyen, de mivel negáltan adtuk hozzá, az eredeti állítás igaz).

Akármennyire az elso rendu logika (predikátumkalkulus) teljességét K. Gödel már 1930-ban belátta, az algoritmikus megoldás csak nemrég született, a sok létezo deduktív bizonyítási lépés és a predikátumkalkulus redundáns szimbólikus felépítése miatti algoritmizálási problémák miatt. Csupán az 1960 évek elején megfogalmazódott (Robinson) az egyetlenegy rezolúciós deduktív lépésre:

```
      A ? B
   ?A ? C
      ---------
      B ? C
```

alapozó rezoluciós algoritmus, amihez a logikai állításokat az un. klóz formára át kell írni, aminek jellegzetessége a minimális számú muvelet, eltüntetett kvantorok és lineáris, könnyen indexelheto forma. A legfontosabb azonban, hogy a rezoluciós lépés szimbólikusan megkülönbözteto módon (üres eredmény) és így egyszeruen implementálható módon jelzi az ellentmondás (inkonzisztencia) megtalálását:

A
?A

??

A rezoluciós bizonyítás könnyu implementálhatósága sajnálatos módon nem oldja fel a logikai bizonyítás exponenciális jellegét és természetesen a félig eldönthetoségen sem tud segíteni. Mivel az eljárásnak csak akkor van kilépési pontja, ha a vizsgált állítás A. Ha történetesen hamis állítással kisérletezünk, az eljárás magától leállni nem tud. Ez problémát jelent a gépi bizonyítás gyakorlati implementálásánál, mert kénytelenek vagyunk idokorlátot bevezetni, amelyen túlmenoen az állítást hamisnak itéljük. Nem segít a 'ha A nem megy, akkor próbálkozzunk ?A-val' taktika, mert az idokorlát erre az esetre természetesen is érvényes. A probléma feloldható: (1) a heurisztikus metalogikai tudás az alap algoritmusba való becsatolásával, vagy (2) a logikai apparátus leszükítésével, hogy a (leszükített) rendszer már teljesen eldöntheto legyen.

3. 11 Tanulás

Tanulás az intelligens rendszer alapveto eszköze, hogy adaptív lehessen. E célból a rendszer megkiséreli a tudását módosítani, illetve kibovíteni a muködése során szerzett új információ alapján, hogy a késobbi (problémamegoldó) képességei megjavuljanak. Tanulás csak a hatékonyság-orientált ágens esetén képzelheto el, hiszen ágensnek emlékeznie kell, hogy az akciói milyen eredményt hoztak.

Tanulás módja függ a rendelkezésre álló információ ás a kibovítendo tudás jellegétol. Tipikus megkülönböztetés a numerikus és a szimbólikus információ, bonyolultabb problémákban a kétféle információ azonban keverten jelenik meg. Numerikus példákat mesterséges neurális hálókkal fel lehet hatékonyan dolgozni, ehhez egy ilyen komponenst az ágensbe be kell tudni építeni. Szimbólikus tudást bovíteni lehet szimbólikusan megfogalmazott példák alapján. A neurális hálóknál kizárólagosan alkalmazott pozitív példák mellett, szimbólikus tanulás hatékonyságát negatív, illetve közel jó (near miss) példákkal is lehet serkenteni, hiszen szimbólikus anyagban ilyen példákat könnyebb értelmesebb megfogalmazni. Szimbólikus tanulás révén szeretnénk új koncepciókat kialakítani, illetve új megoldó lépéseket megtanulni.

Érdekes tanulási mechanizmusokat takar a szabályalapú architektúra. A rendszer hatékonyságát meg lehet kisérelni novelni szabálykirekesztéssel, szabályok összevonásával (általánosítás), vagy megfordítva a szabályok pontosításával. Amire feltétlenül fel kell figyelni, az un. hatékonysági probléma (utility problem) (szabályösszevonással no a tudásbázis, amitol a rendszer muködése lelassúlhat, ha tehát az új szabályokat késobbiekben nem használjuk, meg kelll azokat szüntetni) és az un. hivatkozás probléma (credit assignement problem), azaz a szabályszekvencia melyik tagja igazán felelos a problémamegoldás sikeréért vagy annak hiányáért.

Tanulás fogható fel egyfajta numerikus/szimbólikus konverziónak is, hiszen numerikusan intebzív, empirikus anyag alakul át rendszer szinten erosen szimbólikus, absztrakt modellekké, szabályokká, vagy akár teljes elméletekké. Az ‘adatokból összefüggések’ nagyon fontos kérdés, amely ujabban nemcsak speciálizált kutatási teróleteken (genetikus kutatások), hanem általánosságban adatbázisokban (data mining) is megjelenik. Ezzel a kérdéssel hibrid információ feldolgozás foglalkozik. A hibrid információ feldolgozás módszerei között találkozhatunk neurálisan tanitott szabályrendszerekkel (neuro-fuzzy rendszerek), neurális hálókból kinyert szabályokkal, szabályokkal iniciálizált neurális hálókkal, hálóba szervezett neurális hálokkal, genetikus algoritmusokkal fejlesztett neurális hálókkal és sok hasonló vegyes módszerrel.

3. 12 Beszédfelismerés és nyelvi kommunikáció

A természetes nyelv megértése ágens tipusú rendszerek esetén is nagyon fontos. Bonyolult problémák esetén az ágens környezetében (szoftver ágenseknél és a globális számítógépes hálózat környezetében különösképpen) sok olyan rendszer lesz található (emberi felhasználó, más ágens), amelyek lényeges információk vagy képességek forrásai és amikkel kommunikáni kell. Emberi felhasználóval történo kommunikációhoz a természetes nyelv valamilyen szintu való kezelése mindenképpen szükséges, de a

természetes nyelven történo kommunikáció bizonyos elemeit a tisztán ágensek közötti kommunikációban is lehet hasznosítani (l.d. késobb un. beszédaktusok, speech acts).

A természtes nyelv megértése összetett probléma. Eloször is meg kell oldani a beszédfelismerés problémáját. Zajos akusztikus jelben meg kell különböztetni a beszédet, a szüneteket. Beszédet szét kell választani olyan értelmes hangkomponensekre (fonémaszint), amit továbbiakban szimbólikusan elemezni lehet. Következik a szimbólikus információ elokészítése (morfológia szint), hogy a következo szintaktikai (nyelvtani) elemzés csakis értelmezheto szavakkal találkozzon. Nyelvtanilag helyesen elemzett mondatok jelentésével a szemantikai elemzés foglalkozik. Végül az egyes mondatokban zárt jelentést fogja össze a párbeszéd integrálás. Utolsó lépés az un. pragmatikus elemzés, melynek célja a verbális jelentés mögött rejtozködo valódi jelentés (szándék, hiedelem, stb.) azonosítása, hiszen ez az igazi információ az intelligens rendszer számára (gondoljunk csak az angol 'understatement'-re, vagy az irónikus beszédstilus szófordulataira).

A nyelvmegértés alsó szintjein uralkodó technikák a digitális jelfeldolgozás. Mesterséges intelligencia a szemantikai elemzésnél magasabb szonteken jelent segítséget, hiszen az igazi megértés az jelenti, hogy a rendszer a nyelvi közlésben talált elemeket a tudásbázisában található egyes elemekkel azonosítani tudta. Gép-gép kommunikáció szempontjából a hagyományos protokoll technikák tulajdonképpen szintaktikai szintnek felelnek meg. Magasabb protokoll szintek kiaknázása elkezdodott, ágensek világában különösen érdekes a verbális kommunikáció mögött huzódó szándék és hiedelem információ.

3.13 Egyéb (humán) percepció

Az ember által uralt percepciós lehetoségek közül a látás képessége az, ami bizonyos környezetben és bizonyos problémákkal foglalkozó ágens szempontjából lehet hasznos (gondoljunk pl. az embereket vizuálisan felimero vagy az utat vizuálisan követo rendszerre). A látás gépesítése követi az emberi vizuális információfeldolgozás feltárt, vagy vélt lépéseit, féleg azért mert annak kiderítése, hogy a gépi látást nem lenne-e érdemes más alapokra helyezni még egyáltalán nem feltárt (elképzelheto, hogy egy gépi rendszer rovar módjára sokkal hatékonyabban tudna tájékozódni, mint 'emberi szemmel' nézve a környezetét, probléma azonban, mint mindig, a szükséges tudás azonosítása és a rendszerbe való beépítése).

3.14 Elosztott rendszerek, kommunikáció és együttmuködés

A problémamegoldás képességét fokozni lehet, ha másokra hárítjuk a terhek egy részét. Ennek alapveto feltétele, hogy úgyanabban a környezetben több intelligens rendszer tudjon találkozni. Az ilyen elosztott rendszerben, amit másokkal meg lehet osztani, az: (1) nyers információ, (2) tudás, azaz a bizonyos részproblémák megoldása, illetve (3) a feladatok egy része. A megosztáshoz valamiféle kommunikációs mechanimus szükséges, amely lehet pl. egy közös memória (szoros csatolást biztosító táblaarchitektúra), vagy üzenetváltás az ágenseket összeköto hálozatban. Táblaarchitektúra nem igényel egy külön kommunikációs protokoll megfogalmazását és végrejahtását a tudásforrások között, így nem tekintheto egy igazi kommunikáló rendszernek.

Kommunikáció nem elég. Biztosítani kell, hogy az érdekek találkozzanak, hogy együttmuködés alakuljon ki. Az együttmuködést parancs szóval ki lehet kényszeríteni, ha valamelyik ágenst meghatározó fönöki szerepbe juttatjuk és ennek megfeleloen alakitjuk ki a kommunikációs protokollt. Az egyuttmuködés így is elképzelheto, hogy a kitüntetett szerepkörben lévo ágensek nem fönökök, hanem csupán 'munkaközvetítok'. Ilyenkor a többi ágensbe egyfajta 'vállalkozási kedvet' be kell építeni, máskülönben az igények és a lehetoségek nem fognak találkozni.

Ha a kitüntetett szerepköru ágensek nincsenek, a helyzet nehezebb. Félo, hogy több ágens konfliktusban van másokkal, például a közös eroforrások, vagy az igaznak tartott tudás tekintetében. Ilyen nehéz helyzetek kezelése speciális protokollokat igényel, és akkor sem biztosított, hogy a kivánt együttmuködés egyáltalán létrejön.

3.15 Véges ido problémája

A tudásábrázolással foglalkozó fejezetbol látszik, hogy a deliberatív rendszerek igazi ellensége az ido, pontosabban a problémamegoldás a megoldandó feladat által diktált sebessége. Természetes megállapítás, hogy minden probléma idoben változik, és minden problémamegoldás egy valós-ideju folyamat. Igazi

valós-ideju intelligens rendszerek azonban nem léteznek, beszélhetünk legfeljebb kelloen gyorsan muködo rendszerkrol kelloen lassan változo problémák esetén.

Idoben változó problémák egy része az un. missziókritikus problémák, ahol az intelligens rendszer feladata egy folyamat/ rendszer folyamatos figyelése, és a rendellenességek észlelésekor riasztás, vagy diagnózis. Jellemzo ilyen feladatkörben az un. kognitív túlterhelés, ami azt jelenti, hogy amig a megfigyelt jelenség 'rendesen' viselkedik, a feldolgozandó információ mennyisége kevés, a feldolgozásra rendelkezésre álló ido pedig sok. A helyzet drámaian megfordul, ha a jelenségben nem kivánatos változasok (hibákhoz vezeto trend, hibák terjedése, bekövetkezett hibás állapot) következnek be. Ilyenkor megugrik a feldolgozandó információ és a következtetéseknél használt tudás mennyisége, a rendelkezésre álló ido viszont lényegesen megrövídül.

A dinamikus problémákkal küszködo rendszerek további problémája az, hogy nem elég, hogy gyorsan muködjenek, tudniuk kell következtetni magáról az idofolyásáról is. A fizikában használt valósszám tengely idomodellt nehéz összehozni az alapvetoen szimbólikus és statikus tudásreorezentációkkal.

A rendelkezésre álló véges ido esetén a legfontosabb fejlemény az un. progresszív következtetés gondolata. A lényeg egy fokozatos problémamegoldás, amikor a rendszer gyorsan de pontatlanul igyekszik a feladatot megoldani, tudásbázisának csak egy részét kihasználva. Ha ezek után az ideje elfogy, kész megoldással rendelkezik. Ha az elso megoldási lépés után kiderül, hogy van még ideje és az elso hozzávetoleges megoldásr még nem kell alklmaznia, a rendszer igényesebb elemzésbe kezd bele, igyekezvén az eddigi megoldást megjavítani több tudásanyag és képesség bevetésével. Elofordulhat, hogy az ido közben elfogy, ekkor marad a régebbi megoldás. Nagy eséllyel viszont a rendszer jobb megoldást is tud produkálni, különösen, ha figyelembe vesszük, hogy idoközben több bemeneti információra is tudott szert tenni.

3.16 Nyelvek, eszközök

Érdemes néhány szót szánni a rendelkezésre álló eszközökre. A közhasználatú PC-k teljesítménye annyira megugrott, hogy a speciális esetektol eltekintve más hárdvert nemigen alkalmaznak. Sokkal nagyobb választék tapasztalható szoftver oldalon. A mesterséges intelligenciához tartozó nagyon sok módszer könnyuszerrel implementálható univerzális (imperatív) programozási technikákkal. Pálmát visz itt a Lisp, amely univerzális nyelv ugyan, de a listakezelo függvényei, a szimbólumok többféle kiértékelési módja, futási ideju tipusvizsgálat az ilyen feladatokra igen alkalmassá teszik. Így pl. nagyon egyszeru a keretek, szabályalapú rendszerek, táblaarchitektúra, stb. Lisp implementációja és ilyen implementációk publikusan is elérhetok.

Logikai tudásreprezentáció használatát támogatja a logikai programozási paradigma, melynek fo képviseloje a Prolog. Ujabban Prologot univerzális nyelvi elemekkel is látják el, hogy bonyolultabb problémák esetén is vizsgázzon le jól.

Keretek használatát támogatják a keretalapú nyelvek, amelyek nem mások, mint a keret tudásreprezentáció általában Lisp szintu implementációi. Utolsó és a legkomolyabb termékkategória az un. keretrendszer, avagy egy 'üres' szabályalapú rendszer, aminek tudásbázisa kitöltésre vár. Az ilyen rendszer következteto gépe természetesen készen áll, a felhasználónak így semminemu programozási munkát nem kell elvégeznie, csupán a problémájához tartozó tudásanyagot szabályokba zárni.

4. Milyen rendszer egy agens?
(Egy agens, vagy csak egy program?)

4.1 Általános meghatározások

Ahhoz, hogy jobban megértsük. hogy az ágens elnevezéssel tulajdonképpen milyen rendszerkategóriát szeretnénk szerephez juttatni, lássunk kiindulásul néhány irodalmi meghatározást:

> "Egy ágens akármi lehet, amit úgy lehet értelmezni, hogy szenszoraival a környezetét
>
> érzékeli és a beavatkozó szerveivel a környezetébe beavatkozik" (Russel, Norvig, 1995)
>
> "Autonom ágensek olyan számítási rendszerek, amelyek valamilyen komplex dinamikus
>
> környezetben tartozkodnak, érzekelnek és ebben a környezetben autonom módon cselekednek,
>
> és ily módon olyan taszkokat vagy célokat valósítanak meg, amire megtervezték oket"
>
> (Maes, 1995)
>
> stb.

Általában egy-egy fejlesztés, a pillanatnyi célok érdekében, kissé eltéro meghatározáshoz vezetett és vezet, azonban az 'ágens' szó hagyományos értelmezésében lényeges közös vonások is vannak:

?? valami, ami cselekszik, vagy képes cselekedni,

?? valami, ami mások helyett, de azok beleegyezésével cselekszik.

Autonom ágens egy lehetséges definiciója tehát valahogy így festene:

egy környezetbe beágyazott és a környezet részét képezo rendszer, amely ezt a környezetet érzékeli, és ennek függvényében, idoben folyamatosan, agendájának megfeleloen cselekszik úgy, hogy ezzel a késbob érzékelt környezet állapotát befolyásolja.

Ez persze csak egy hozzávetoleges és különben is túlságosan bo definició. A beskatulyázás veszélye a definicióba még megféro extrém esetekbol is meglátszik, pl:

?? ember, magasabb rendu állatok: többféle indíték, többféle érzékszervek, többféle lehetséges akciók, nagyon bonyolúlt vezérlési struktúrák.

?? ..... : .....

?? termosztát, bakterium: egy-két érzékszerv, egyetlenegy akció, abszurdálisan egyszeru vezérlési struktúra.

Egy ilyen definiciót ki kellene bovíteni, ami már 'használható' ágens 'altipusok'-hoz vezethet. Ne felejtsük azonban néhány lényeges pontot:

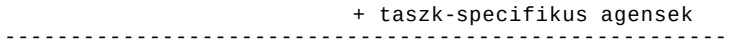
?? autonom ágens egy környezetbe van beágyazva, abból kiemelve, más környezetben (amihez az érzékszervei 'nem jók') többé nem ágens;

?? minden szoftver ágens egy program, de nem minden program egy ágens: lényegi különbség, hogy az ágens kimenetei (célorientáltan) befolyással vannak

?? a késobbi bemeneteire, és hogy a muködése idoben 'nyújtott'.

Az ágensek egy lehetséges osztályozása (agensek világa) az alábbi árán látható:

```
-------------------------------------------------------
             autonom agens:
                * biológiai agensek
                * robotikus agensek
                * számítási agensek
                     o 'artificial life' agensek
                     o szoftver agensek
                          + virusok
                          + szorakoztató agensek
```



4.1. ábra. Ágensek osztályozása

4.2 Inteligens agensek (részletesebb) elmélete

Az ágensek világában két alapveto megközelítés létezik annak tekintetében, hogy milyen ‘természetes’ tulajdonsággal fel kellene ruházni egy ágensrendszert. Az un. 'gyenge' ágens fogalma a szintén ilyen jelzot viselo 'gyenge' mesterséges intelligencia nyomán fogalmazódott meg, ahol az intelligens viselkedés csupán külso reprodukálásáról van szó. A 'gyenge' ágens alapveto tulajdonságai a következok:

?? 'kitartó' (persistent) - folyamatosan konzisztens belso állapottal rendelkezik;

?? autonom - nagy foku ellenorzést gyakorol a saját belso állapota és akciói felett;

?? önálló - direkt emberi beavatkozás nélkül muködik;

?? reaktív - érzékeli a környezetének változásait és reagál azokra;

?? szociális - kapcsolatban áll emberekkel, ill. más ágensekkel;

?? kommunikál - képes információt cserélni más rendszerekkel;

A 'magasabb intelligencia' irányában vezeto kiegészíto tulajdonságok lehetnek még:

kezdeményezés: - (pro-active, data-directed execution) cél-orientált, opportunista viselkedés (felhasználói feladat hiányában maga fogalmazza meg a feladatait);

?? mobilitás - képes a környezetében helyrol-helyre mozogni, megtartva saját belso állapotát;

?? következtetés - alapvetoen logikai tudásreprezentációval dolgozik, tehát tudnia kell logikai módon következtetni;

?? tervkészítési készség - a fentiekbol értelemszeruen következik;

?? tanulás, adaptáció;

?? párbeszéd - ahhoz, hogy belássuk, hogy agens megosztja-e a céljainkat és képes-e azokat megvalósítani, párbeszédre van szükség, amely tisztázza az intenciókat és a képességeket; a párbeszéd eredménye a megegyezés;

?? igazmondás - a környezetének szántszándékkal nem hazudik és , ha többen vannakj, segítokész,

?? jóindulat - megkisérel teljesíteni mások kéréseit;

?? együttmuködés - lényegében egy megállapodás megszületésében: a felhasználó feladatról 'beszél', az agens arról, mit tehet ennek érdekében;

?? szelektív figyelem - hatékony muködés véges (szukos) eroforrások ill. szenzorikus lehetoségek közepette;

?? robusztus - muködhet helyzetek sokosságában;

?? racionális - céljai elérésére törekszik;

?? korlátos racionálitás (szukos eroforrások optimális kihasználása), törekedhetünk ennek biztosítására:

- ?? tervezéskor,
- ?? logikai következtetés eredményeként,
- ?? adaptáció révén.

Igen fontos problémakör a rizikó és bizalom kérdése. Egy agens a döntés és a feleloség kihelyezését jelenti. A kihelyezésrol viszont nem lehet szó, ha nincs valami biztosíték, hogy aminek a hatáskörébe a feladatot kihelyezzük, képes azt elvégezni, méghozzá úgy, ahogyan mi ezt szeretnénk. Ha a feladatot nem mi végzük, rizikót vállalunk, hogy az ágens esetleg ezt a fe ladatot hibásan fogja megoldani.

A rizikó és bizalom mérlegelésénél az ágens modelljét össze kell vetni a feladat domain-jével. Itt beszélhetünk az un. alacsony rizikójú domainrol (ilyen pl. a szociális kapcsolattartás), illetve a magas

rizikójú domainrol, ahol kérdéses lehet az ágens használata. Ilyen szempontból fontos az un. fokozatos romlás (gracefull degradation) tulajdonság. Ha kommunikációs vagy domain jellegu problémák lépnek fel és ennek eredményeként az ágens nem teljesen azzal foglalkozik, amivel kellene (meg lett bizva), az eredmény ne essen azért távol az elvárttól.

A 'gyenge' definició párja az un. 'eros' ágensdefinició (szintén az un. 'eros' mesterséges intelligencia nyomán, ahol a kutatás a 'tiszta' intelligencián túlmenoen kiterjed más 'emberibb' vonások megértésére és gépesítésére is). Ágens szintjén ez azt fogja jelenteni, hogy az elobb részletezett tulajdonságokon túlmenoen, az ágens koncepciójának kidolgozása, megvalósítása olyan elvek alapján fog történni, melyeket többnyire emberek körében értelmeznek, mint: tudás, vélemény, szándék, meggyozödés, kötelességtudat. Esetenként az ágens modellezése az 'emberi érzelmekkel' is történik, ami az un. emocionalis ágenshez vezet. Ez az un. intencionális megközelítés lényege, hogy nagyon bonyolult rendszerekrol úgy is beszél(het)ünk, hogy képesek vagyunk a viselkedésüket megmagyarázni vagy megjósolni anélkül, hogy a muködésüket megértenénk (ami viszont rendszertechnikailag igen hasznos lehetoség).

4.3 Elméletek, architektúrak és nyelvek

Ahhoz, hogy megbizható módon az ágenseket tervezni tudjuk és foleg, hogy jól tervezzük meg az ágensek közötti kölcsönhatásokat, jól áttekíntheto elmélet szükséges, amiben az ágenseket, a tudásukat, a problémamegoldás folyamatát egységes módon modellezni tudjuk. Egy ágensmodell alapvetoen egy logikai modell, hiszen tulajdonságait legegyszerubben tényszeruen kifejezni. Logikai modell lehetoséget is nyújt bizonyos ágens, illetve közösségi tulajdonságokat logikai következtetéssel belátni.

Ágensmodell speciékitása, hogy a tudásreprezentációval nem csak azt kell tudni kifejezni, hogy az ágens miket tud, hanem ezt is, hogy az ágens tud valamit, és foleg, hogy tudja, mások is tudnak valamit. Ezek az un. információs attitüdök, tehát a tudás, a hiedelemek (belief) és a kölcsönös információs attitüd. Az ágens egyéb tulajdonságai az un. pro-attitüdök: kíváncsiság, intenciók, kötelesség, kötelezettség, választás képessége. és a kollektív pro-attitüdök.

Különö probléma az un. információs attitüdök (hozzáállások) modálitásainak a logikai modellben való jó kifejezése. Modálitás az elso rendu logika (predikátum kalkulus) olyan kiterjesztése, amivel le tudjuk írni az ido mulását, a hiedelmeket, a lehetoségeket, stb. Próbáljuk logikai módon kifejezni azt, hogy Julia ágens azt hiszi, hogy a vele kapcsolatban lévo emberi felhasználó egy fiú:

```
'Julia azt hiszi, hogy P egy fiú'                (*)
```

Ennek egy naiv átírása logikai nyelvre valahogy úgy festene, hogy:

```
'Hiszi(Julia,Fiú(P))'
```

ahol Hiszi és Fiú predikátumok, Julia és P viszont objektumok. Sajnos ez az állítás csak látszólag egy predikátum kalkulus-beli állítás, mert az egyik argumentum Fiú(P) már önmagában egy logikai állítás, és ez nincs megengedve. Predikátum kalkulus állításai az un. 'igazságfunkcionálok' - értékük csak a bennük szereplo termek igazságértékétol függ.

Teljesen világos, hogy ateljes (*) állítás értéke nem függ a ‘P egy fiú’ állítás logikai értékétol, így az elso rendu logikában nem modellezheto. Ha nem tudunk dolgozni az 1' rendu logikával, akkor azt ki kell terjeszteni (modális logikák, modálitások), viszont akkor annak a félig eldönthetosége el fog veszni. A probléma megoldása modális operátorokkal kiterjesztett logika, amely más vonatkozásaiban viszont leszükített, hogy a bizonyítás kezelheto legyen. Ebben a logikában ki kell tudni fejezni a pro-attitüdök modálításai, az akciók hatását és az ido reprezentációját.

Továbbiakban látni fogunk, hogyan lehet kialakítani logikai ágensmodelleket, ágens architektúrákat és ágens nyelveket. Foglalkozunk továbbá az intelligens agensek együttmuködésével, ami a :

?? kommunikáció - a tudás és az információ kicserélése (KQML)

?? kooperáció - az együttmuködés lehetséges formái, és

?? koordináció - az együttmuködés összehangolasa, tervezés.

5. Ágens programozás

Az eddigiekben áttekintettük a szoftver ágensekkel kapcsolatos alapfogalmakat, belso felépítésüket, és fobb komponenseiket. Ez a fejezet segítséget kíván nyújtani a megismert technikák gyakorlati kipróbálásához olyan technológiák, környezetek áttekintésével, melyek segítségével ágensek készíthetoek.

Általánosságban elmondható, hogy nincs kitüntetett ágens programozási nyelv és környezet. Mind általános célú programozási nyelvek (C, C++) és fejlesztési környezetek, mind a mesterséges intelligenciában már megismert fejleszto eszközök (LISP, CLIPS, Prolog), valamint speciális alkalmazói környezetek eszközei is használhatóak ágensek kifejlesztésére. Vannak ugyanakkor olyan technológiai elemek, melyek leegyszerusíthetik az elvégzendo feladatot: az objektum-orientált megközelítésmód, alkalmazások és számítógépek egymás közötti kommunikációját és együttmuködését segíto eszközök (pl. IPC: *inter-process communication*, RPC: *remote procedure call*, hálózati programozás, stb.), elosztott rendszerek készítését támogató módszerek (pl. DCOM vagy CORBA), szoftverkomponensek újrafelhasználását támogató rendszerek (pl. JavaBeans), illetve speciális alkalmazási környezetekben ágens sablonok (pl. Lotus Notes Agent Template, vagy a JAT: *Java Agent Template*). Mindezek alkalmazási környezettol függoen lerövidíthetik, megkönnyíthetik az ágensek kifejlesztését.

A legkönnyebben elkészítheto ágensek lehetnek például UNIX alatti egyszeru shell script programok, melyeket a *cron* program vagy valamely operációs rendszer esemény indít alkalmilag vagy rendszeres idoközönként. Másik általános, de szerteágazóbb lehetoségeket kínáló script környezet a VisualBasic, mellyel a Windows rendszer és alkalmazási környezetben hozhatunk létre önállóan muködo programokat. Jó segítséget nyújt ehhez az OLE (Object Linking Environment) automatizálási képessége, és a COM (Component Object Model) objektumokat összeköto szolgáltatáshalmaza. Egy szukebb alkalmazási környezetben, az adatbázis rendszereknél megismert trigger események is elindíthatnak önállóan feladatot megoldó SQL programokat. Más speciális környezetekben, például elektronikus dokumentum menedzselo rendszerekben a felhasználó egyszeruen (szinte programozás nélkül) kifejleszthet automatikusan aktiválható programokat, majd ezeket a rendszerbe illesztheti bizonyos rendszeresen felmerülo problémák önálló megoldására.

Kifejezetten az ágens megközelítésmódot támogató általános célú programozási nyelvek és környezetek is léteznek, azonban ezek inkább egy-egy kutatócsoport (általában a fejlesztoik) által használtak, nem terjedtek el széles körben. Az általános célú programozási nyelvekhez is készültek ágens fejlesztoi kiegészítések, melyek részben általános célúak, részben speciális alkalmazási környezetekben, vagy speciális célra használhatóak. Az egyik legtöbb kiegészítéssel ellátott általános célú programozási környezet a Java.

Ágensek fejlesztése Java környezetben

A Sun által kifejlesztett Java rendszer egy objektum-orientált programozási nyelvet (Java), fejlesztési (Java Development Kit, JDK) és futtató környezetet (Java Virtual Machine, JVM) foglal magába. A nyelv a C++ alapjain épült, szigorúbban eloírva az OO technika alkalmazását és szukítve a C nyelvbol örökölt lehetoségek körét. A Java nyelven írt programokat a fejlesztoi környezet fordítója ún. bájtkódra fordít, melyet a futtató környezet értelmez és hajt végre.

A nyelv és a környezet kialakításakor elsosorban a hálózatos és Web alkalmazási környezet igényeit vették figyelembe egy tiszta, egyszeruen megtanulható, hordozható, általános célú programozási nyelv és környezet kialakításakor. A fejlesztoi környezet olyan speciális Java alkalmazások elkészítését is támogatja, melyek közvetlenül a Web böngészokön belül futtathatók. Ezek a **Java programkák (Java applet)**. Ezen alkalmazási környezetek olyan biztonsági követelményeket támasztanak, melyeket a Java rendszer megalkotói nyelvi, fejlesztoi és futtatói szinten is igyekeztek kielégíteni. A nyelv nem tartalmaz direkt memória manipuláló elemeket (hiányoznak a C/C++ nyelvekben megszokott mutató típusok), a fordító ellenorzi és megköveteli a nyelvi konstrukciók betartását (jóval kevesebb pongyolaságot és kerüloutat enged meg, mint a C/C++ fordítók), a futtató környezet a kód értelmezése elott biztonsági ellenorzést végez a kódon. Az applet technológia további megkötésekkel él már a fejlesztés ideje alatt, melyek közül a leglényegesebb a helyi eroforrások hozzáférésének és a hálózati kapcsolat kiépítésének szigorú korlátozása (nem nyitható fájl, hálózati kapcsolat csak a forrás állomással építheto ki, stb).

A Java ágens sablon

Az alábbiakban röviden ismertetünk egy Java ágens kiterjesztést, a Java Agent Template-et (JAT, Rob Frost, CrossRoute Software Inc.). Az ismertetés során elsosorban az általános felépítésre koncentrálunk, nem térünk ki az implementált objektumok pontos használatára, szintaktikai megkötéseire, melyek az internetrol szabadon letöltheto fejlesztoi környezetben megtalálhatóak.

A rendszer alapja az `Agent` objektum, mely egy többszálú, autonóm szoftver ágenst reprezentál. Minden ágens rendelkezik egy kommunikációs interfésszel (`CommInterface`), mely aszinkron módon képes más ágensekkel üzeneteket váltani a KQML protokoll segítségével (lásd a "Multi-ágens rendszerek kommunikációja" c. fejezetet). Egy másik interfész (`MessageOutput`) az ágenst az ot kívülrol körülvevo osztállyal (például egy felhasználói interfésszel) valósít meg kommunikációt. E két interfész objektummal együtt az ágenst egy közös kontextusban kell kialakítani, mely az `AgentContext` interfészt implementálja, s így biztosítja az ágensek egységes külso felületét.

Az ágensek létrehozásukkor már ismerik a KQML kommunikációs nyelvet és az ágens ontológiát, melyek segítségével más ágensekkel kommunikálhatnak. A létrehozáshoz szükséges egy inicializáló fájl (URL), és egy tárolóhely címe (URL), melyen más ágensekkel közösen használt objektumok, nyelvek, ontológiák osztályai találhatóak. Az ágensek belso tudása úgynevezett eroforrás osztályokban tárolódik, melyekre a rendszer kölcsönös kizárást biztosít a többszálú ágensen belül, így biztosítva azok konzisztenciáját. Egy speciális eroforrás osztály (`RetrievalResource`) tartalmazza a más ágensek címeit, a tolük beszerzett nyelvi, ontológiai és egyéb osztályok referenciáit. Ezek segítségével az ágens más ágensek objektumait is felhasználhatják – a rendszer automatikusan gondoskodik azok beszerzésérol és elindításáról.

A rendszer része egy ágens név szerver (Agent Name Server, ANS), mely a rendszerbe lépo ágensek nevét és elérhetoségét tartja nyilván. Az ágensek létrehozásukkor ennek a kitüntetett ágensnek küldik el nevüket és címüket.

Az ágensek közötti üzenetek fogadására a rendszer a következo mechanizmust biztosítja. A minden ágensben megtalálható CommInterface végzi az üzenetek fogadását, elemzi oket, majd KQML formára alakítva továbbítja az üzenetértelmezonek. Az üzenetértelmezo az üzenetben megadott ontológia alapján az AgentOntology osztály vagy leszármazottjának azt a példányát hozza létre az üzenet értelmezésére, mely képes a megadott ontológia feldolgozására. Amennyiben ilyet nem talál, úgy a RetrievalResource objektum segítségével az üzenet feladójának közös eroforrás tárolóhelyérol automatikusan beszerzi a szükséges osztályt és létrehozza belole az üzenet feldolgozására képes egyedet. Minden egyes ontológián belül a nyelvének megfeloen kell a tartalmat értelmezni. Amennyiben az ágens nem rendelkezik a megfelelo nyelvi osztállyal, ismét az üzenet feladójához fordul és automatikusan beszerzi azt.

5. Ágensek és ágensrendszerek - logikai modellek

5.1 Bevezeto

Ágensek leírásában a logikai modelleknek kitüntetett helye van. Egy ágens bizonyos funkciói lehetnek ugyan reaktív kivitelezésuek is, de igazán komplex ágensnek tudnia kell akciós terveket szonie, ehhez viszont tudni kell érvelni, amihez viszont a környezetének és a saját magának logikai leírását kell használnia. Így a logikai modelleknek ezen a területen meghatározó szerepe van. Kérdés persze, hogy logikával mennyire fogunk messzire menni. Az eddigi elemzések alapján meg lehet fogalmazni az ágens logikai modelljének néhány kivánatos komponensét:

?? ágensnek legyenek explicit módon reprezentált (valamilyen logikai 'hiedelem nyelvben' kifejezett) hiedelmei,
?? ágens legyen képes kideríteni a hiedelmeinek néhány (de nem szükségképpen valamennyi) logikai következményét,
?? ágensnek rendelkeznie kell számítási eroforrásokkal a 'belso' kognitív akciói végrehajtásához (egy ágens ‘külso’ kognitív akciója az információ/tudás szerzése kommunikáció révén, ‘belso’ kognitív akciója a saját tudásbázisa alapján történo érvelés),
?? ágens más ágensek hiedelmeit befolyásolja komunikáció utján, üzenetek átadásával.

Egy több ágenses rendszerhez szükséges még egy végrehajtási (execution) modell mégadása, amely lehet szinkron (synchronous) (mindenki egyszerre cselekszik), illetve átlapolt modell (interleaved) (egyszerre legfeljebb egy ágens aktív). Ne felejtsük azonban, hogy az ágens logikai modellezésénél:

?? nem egy emberi, hanem egy gépi rendszert írunk le,
?? nem egy kanonikus modellt hozunk létre,
?? azonban egy idealizált, leegyszerusített modellt hozunk létre, amely csak a leglényegesebb vonásokat tartalmaz.

Az irodalomban négyféle logikai modell szokott elõfordulni, aszerint, hogy itéletkalkulust, illetve predikátumkalkulust és lineáris (valós számegyenes), illetve elágazó idõmodellt használunk. Az itéletlogika és a predikátumkalkulus (elsõrendû logika) közötti választás a logika kifejezõ ereje és a logika eldönthetõsége közötti kompromisszum függvénye. Az elágazó idomodell igénye abból fakad, hogy egy ágens rendszerben többféle döntés lehetséges, az adódó lehetoségek és folytatások miatt. Az elágazó idomodell lehetséges jõvõket, potenciális lehetõségeket segít formálisan ábrázolni, azonban lényegesen bonyolultabb. Az irodalomban használt elnevezésekkel a négyféle modell a következõ:

5.1. ábra. Ágens logikai modellek.

	itéletlogika	1' rendu logika
lineáris ido modell	AL	QAL
elágazó ido modell	BAL	QBAL

5.2. Intencionális rendszerekrol még néhány szót

Hiedelmek logikai ábrázolása szempontjából beszélhetünk elsorendu, illetve magasabb rendu intencionális rendszerrol. Elsorendu rendszer magukat a hiedelmeket, kivánságokat ír le, magasabb rendu intencionális rendszer kitér a hiedelmekrol, kivánságokról alkotott hiedelmekre és kivánságokra.

Az intencionális megközelítés (azaz a hiedelem, hit, szabad akarat, intenció, tudat, képesség, akarat, stb. formális ábrázolása) egy gépi rendszer esetén jól megalapozott lesz, ha ez a megközelítés ugyanazt az információt fejezi ki, mint ahogyan ezeket a fogalmakat az emberekre alkalmazzuk. A megközelítés tehát hasznos, ha a gép viselkedését megfeleloen 'megvilágítja', de ne felejtsük, hogy használata formálisan sohasem szükségszeru.

Az intencionális megközelítést egy egyszerû példával, a villanykapcsoló esetével, világítsuk meg. Villanykapcsoló mechanisztikus modellje közismert: a kapcsoló rugós szerkezete az áram útját zárja, illetve nyítja, és ily módon befolyásolni tudjuk az elektromos fogyasztó (villanykörte) állapotát. Az intencionális modell valahogy így nézne ki: a kapcsolóban rejlo agens kedvünkre tesz, ha a kapcsoló révén tudomá st szerez szándékainkról, és intézkedik az áramút zárásáról.

Mindkét modell koherens, jól magyarázó, jól jósló, de az intencionális modell nem hoz többet a mechanisztikus modellhez képest, mert alapvetoen értjük a kapcsoló muködését ! Vegyük azért észre, hogy gyerekek sokszor intencionális tipusú modellekkel kezdenek, amig nem szerzik meg a mechanisztikus tipusú modellel kifejezett tudást (pl. 'Nem tudom, hogy a számítógépem mit akar tolem' - és mint szófordulatot, késõbb, felnõtt korban is alkalmazzák). Igazán érdekes eset egy olyan rendszer, amelynek a mechanisztikus modellje kilátástalanul bonyolult, vagy a modell értékelése tulságosan eroforrás igényes (pl. egy számítógép, egy nagy elosztott rendszer, NII, egy ember, ...). Ilyenkor az intencionális modell esetleg az egyetlen konstruktív modellezési lehetõség, és ilyen aspektus miatt ez a kérdés ágensek modellezésénél is megjelenik.

5.3. Hogyan lehet érvelni intenciókról

Térjünk vissza az elobb elemzett problémához és kiséreljük meg logikai módon kifejezni azt, hogy Julia nevu ágens azt hiszi, hogy a vele kommunikáló, Dugó becenévre hallgató emberi felhasználó egy fiú:

```
'Julia azt hiszi, hogy Dugó egy fiú'                (*)
```

Ennek egy naiv logikai átírása valahogy úgy festene, hogy:

```
'Hiszi (Julia, Fiú (Dugó))'
```

ahol Hiszi és Fiú predikátumok, Julia és Dugó viszont objektumok. Sajnos ez az állítás helytelen, mert az egyik argumentum Fiú (Dugó) már önmagában is egy logikai állítás, és ez nincs megengedve. Azontul az itélet vagy a predikátum kalkulus állításai az un. 'igazságfunkcionálok' - értékük csak a bennük szereplo termek igazságértékétol függ. Pl. a 'p ^ q' értéke csakis a p és q értékétol és az állítás muveleti struktúrájától függ (igazságtábla). Értheto azonban, hogy a teljes (*) állítás értéke nem is függhet a 'P egy fiú' állítás logikai értékétol, hiszen Julia akkor is elhiheti, ha ez történetesen nem igaz. A probléma így az elso rendu logikában nem modellezheto.

Fokozhatjuk a problémát, ha figyelembe vesszük a (Dugó = Nebula-Hugó) logikai ekvivalenciát. Igaz legyen (lenne), hogy:

```
Hiszi (Julia, Fiú (Nebula-Hugó)) ???
```

Egyáltalán nem biztos, pedig logikai ekvivalenciának logikai formulába való behelyettesítése nem szabad, hogy hatással legyen az állítás értékére. A hiedelem ábrázolása tehát olyan kontextust teremt, ahol az elsõrendu logika behelyettesítései nem érvényesek ('homályos kontextus' - 'opaque context'). Ez komoly gond, mert nem támaszkodhatunk a megszokott logikában használt igazságtáblázatokra.

A hiedelem ábrázolásával tehát kétféle problémánk van - szintaktikai és szemantikai (avagy milyen legyen a hiedelem állítások külalakja, és hogyan lehessen megállapítani az értéküket). Szintaktikai problémák megoldása (tehát hogyan lehet egy jól definiált állítást megfogalmazni hiedelem esetére is) lehet:

?? modális logika: modális operátorokkal, amelyek viszont nem igazságfunkcionálok, illetve
?? metanyelv, amely egy olyan elsorendu logika, melynek termjei egy másik nyelv állításai.

A szemantikus problémák legelterjettebb megoldása (avagy hogyan lehet eldönteni, hogy egy modális állítás igaz, vagy hamis) általában az un. lehetséges világok (megadjuk, hogy egy modális állítás milyen 'világban' igaz).

5. 4. A lehetséges világok és modális logika

A lehetséges világok ötletét világítsuk meg egy példával:

> *Képzeljünk el egytetszoleges kártyajátékot. Az ágensünknek káró ásza van. Az ellenfelek minden kártyakombinációja egy-egy konkrét 'világ'.Az ágens belegfog a lehetetlen világok eliminálásába, feltéve, amit tud (pl. hogy lehetetlen az a világ, amelyben nincs neki káró ásza). Amilyen világok maradna megk, azok a 'lehetséges világok' (lehetséges állapotok, az ágens tudása szerint). Ami igaz minden lehetséges világban, azt az ágens nyilvánvalóan 'elhiszi'. Egy 'lehetséges világ' tehát meghatározza az adott hiedelem (modális állítás) érvényességi körét.*

A hiedelmek és tudás ábrázolására modális logikát fogunk alkalmazni. E célból vizsgáljuk meg, hogy a modális logika milyen apparátust jelent, és majd azzal foglalkozunk, hogy a modális logika apparátusát hogyan terjesszük ki és interpretáljuk a hiedelmek esetére.

A közönséges modális logika eredetileg egy filozófiai fejleményként indult. A modális logika kiinduló pontja az igazságok különbözo szilárdsága. Szükségszeru igazságok (pl., hogy ?(2) nem racionális szám) olyanok, hogy másképpen nem is lehetne, szükségszeru igazság, amely minden lehetséges világban igaz. A lehetséges igazságok (pl. Duna állása ma alacsony) viszont ideiglenesek, most igazok, de lehetne másképpen is. Modális logika két új, modális operátor vezet be a közönséges itélet kalkulusba:

```
□       - 'szükségszeru, hogy'
?       - 'lehetséges, hogy'
```

Az ilyen új operátorok esetén meg kell oldani az új szintaktikai szabályok megfogalmazását, avagy milyen alakú legyen egy jól definiált állítás. Komolyabb gond az új operátorokat tartalmazó jól definiált állítások logikai értékének a meghatározása, hiszen azok most már nem igazságfunkcionálok.
A probléma megoldására két általános megközelítés lehetséges: (1) az adott problémakörre vonatkozó, heurisztikus kiszámítási szabályok, és (2) általánosabb, általában absztrakt algebrai struktúrákra alapozó és több probléma körben is alkalmazható kiszámítási szabályok.

Szintaktikai eloírások (szabályok)

Legyen az atomok (elemi állítások) egy megszámlálható halmaza, Prop = {p, q,....},
1. Ha p? Prop, akkor p egy állítás;
2. Ha p, q állítások, akkor: true, ~p és p ? q szintén állítások;
3. Ha p egy állítás, akkor □p és ?p szintén állítások.
Most még meg kellene szerkeszteni a 'lehetséges világokat'. Egy-egy világ nem más, mint az ágens egy konkrét 'tudás', ill. 'hiedelem' állapota. Mivel az ágens tudása a muködése során változik, ágens folyamatosan kerül át egy 'lehetséges világból' egy másik 'lehetséges világba'. Az ágens 'alkotától', 'képességeitol' függ, hogy milyen 'lehetséges világokat' képes átjárni. A közönséges modális itélet logika modellje tehát (*W*, *R*, ?), ahol *W* a világok egy halmaza, *R* ? *W* ? *W* az ágensre jellemzo reláció világok felett (az un. ágens hozzáférési reláció - az ágens valami módon (akció, érvelés) át tud 'nyulni' egy világból egy másik világba) és ?: *W* ? powerset Prop, egy értékelo függvény, w ? *W* -hez adja meg a w világban igaz atomikus állítások halmazát.
Foglaljuk össze a logika szemantikai szabályait. Legyen

```
(M, w) |= állítás
 |  |__ egy referencia világ
 |__    az elõbb részletezett modell
```

az un. teljesülési reláció (a specifikált állítás teljesül az adott világban). Akkor az alapveto szemantikai szabályok a következok:

```
(M, w) |= true
(M, w) |= p        p ? Prop, a.cs.a, ha p ? ? (w),
(M, w) |= ?p       a.cs.a, ha (M, w) |?p,
(M, w) |= p ? q          a.cs.a, ha (M, w) |= p , vagy (M, w) |= q,
(M, w) |= □p       a.cs.a, ha ? w ? W. ha (w, w')  ? R, akkor (M ,w') |= p
                   igaz minden hozzáférheto világban,
(M,w) |= ?p        a.cs.a, ha ? w'? W . (w, w') ? R, and (M ,w') |= p
                   igaz legalább egy lehetséges világban..
```

Egy fontos összefüggés a modális operátorok dualitása:

□p ? ???p

Egy modális logikai állítás lehet:

- kielégítheto, ha igaz lesz valamelyik (M,w) párban, különben nem elégítheto ki;
- igaz egy modellben, ha kielégített a minden hozzátartozó világban;
- érvényes (valid) a modellek egy osztályában, ha minden modellre igaz;
- érvényes szimpliciter (valid simpliciter), ha igaz a modellek minden osztályában;

egy érvényes állítás jelölése: |= p (azaz az igaz értéke nem világfüggo).
A szintaktikai és a szemantikai szabályok megadása még nem elég. Logika tulajdonságait pontosítani kell annak definiálásával, hogy a szokásos és az ujannan bevezetett operátorok milyen kapcsolatban vannak egymással. Mivel a modális operátorok bevezetése heurisztikus volt, az igazság-funkcionál és modális operátorok kapcsolatát formálisan levezetni nem lehet. Egyetlen lehetoség, hogy a javasolt összefüggéseket axióma szintre emeljük és minden bizonyítás nélkül elfogadjuk. A lehetséges axiómák közül a legfontosabbak a:

Neve	axióma
K	|= □(p ? q) ? (□p ? □q)
NR	ha |= p, akkor |= □p (az un. szükségszeruségi szabály - 'necessitation rule')

További axiómak függenek az R hozzáférési reláció tulajdonságaitól:

Neve	axióma	R tulajdonságai	elsorendu logikai jellemzés
T	□p ? p	reflexív	? w ? W. (w, w) ? R
D	□p ? ?p	soros	? w ? W. ? w' ? W. (w, w) ? R
4	□p ? □□p	tranzitív	? w, w', w' ? W'. (w, w') ? R és (w', w")? R ? (w, w")? R
5	?p ? □?p	euklideszi	? w, w', w" ? W. (w, w') ? R és (w ,w") ? R ? (w', w") ? R

Az un. megfeleltetési elmélet (correspondence theory) azt állítja, hogy az R reláció tulajdonságai és a lehetséges axiómák összefüggenek. A 4 db. új axióma elvileg 16 különbözo logikai rendszerhez vezethet, de abból csak 11 az igazán különbözo logikai rendszer. Azoknak a rendszereknek a szokásos jelölése az axiómanevek felsorolásából áll (pl. 'K' logika, 'TD4' logika, stb.)

5.5. 'Tudás' kifejezése modális logikával. Episztemikus (epistemic) logika

Az ágens tudását most már könnyusze rrel ki tudjuk fejezni. E célból nevezzük át a □ modális operátort, mondjuk következoképpen:

□p - 'ismert, hogy' p (it is known that p), de lehetne úgyszintén: 'ágens tud', 'ágens hiszi', ...

Ez alapvetoen egy egyedi ágens logikája. Több ágens esetén több hozzáférési relációt kell definiálni, mindegyik ágenshez külön-külön (hiszen ágensek episztemikus tulajdonságai igen különbözoek is lehetnek). A a több ágenses modell tehát: ($W, R_1, R_2,..., R_n$, ?). Az □ operátor helyett be kell vezetni:

```
Ki - 'i-edik ágens tudja, hogy' modális operátort, amely szemantikai
meghatározása:
(M, w) |= Ki p     a.cs.a ? w'?W' . ha (w, w') ? Ri, akkor (M, w') |= p
```

Mennyire jó a közönséges modális logika a tudás és hiedelem kifejezésére ?

?? (NR) axiómából következik, hogy egy agens minden érvényes állítást tud. Nyílvánvaló, hogy ezen belül minden itélet kalkulusbeli tautológiát is. Tautológiákból viszont végtelen sok van, így az ágens tudása végtelen;

?? (K) axiómából következik, hogy az ágens tudása logikai implikációra nézve zárt. Legyen a p állítás egy ? = {p_1, ..., p_n} állításhalmaz logikai konzekvenciája. Akkor minden olyan világban, ahol ? igaz, p is igaz lesz, abból kifolyólag érvényes állítás az, hogy: $p_1 \wedge p_2 \wedge ... p_n$? p.

(NR) miatt az ágens elhiszi azt, és mivel hiedelmei implikációra zártak, ha elhiszi ? -t, el kell hinnie p-t is. Abból kifolyólag az ágens tudása logikai konzekvenciára nézve zárt. Ez viszont erosen intuicióellenes: pl. Peano-axiómák birtokában egy ágensnek tudnia kellene, hogy Fermat-féle nagytétel igaz, vagy sem, stb. Kapcsolódik ide a logikai 'mindentudás' (omniscience) problémája, avagy szabad-e/ kell-e minden érvényes állítást tudni ? Az ágens tudása/ hiedeleme zárt a logikai konzekvenciára nézve, így a logikai konzisztencia tulságosan eros megkötést jelenthet a korlátos eroforrásokkal érvelo ágensnek, ha viszont az ágens nem konzisztens, akkor (logikai törvények miatt) mindent elhisz. Gyengébb megkötés lehet a 'nem-ellentmondásosság', azaz az ágens ne higye el egyszerre a p-t és a ? p-t.

A D, T, 4 és 5 axiómák több (n) ágenses rendszer esetén is hasznosak. Legyen ilyenkor a jelölésük D_n, T_n, 4_n, és 5_n, valamint jelentse K_n azt, hogy az 'n-ik ágens tudja'. Elemezzük most e axiómák néhány tulajdonságát.

```
Dn:     Ki p ? ?Ki ?p , ha i-edik ágens a p-t tudja, akkor a ?p-t nem tudja
        (ágens tudása nem ellentmondásos);
Tn:     amit tud, az igaz  (tudás = igaz hiedelem:  i-ik ágens tudja a p-t, ha
i hiszi a p-t és p igaz);
4n:     pozitív introspekció ('önvizsgálat'),  (ágens tudja, amit tud);
5n:     negatív introspekció  (az ágens tisztában van azzal, hogy mit nem
tud);
        tehát agensnek tökéletes tudása van arról, amit tud, és amit nem tud;
```

általában a 'tudás' logika az un. $S5_n$ rendszer K, T és 5 axiómákkal, a 'hiedelem' logika viszont az un. gyenge-$S5_n$ rendszer K, T, 4 és 5 axiómákkal.

Nem elhanyagolhatók a számítási bonyoldalmak sem. A bizonyíthatósági probléma a K_n, T_n, $S4_n$, $S5_n$ és gyenge-$S5_n$ logikákban eldöntheto, az érvenyességi és a kielégítési probléma K_n, T_n, $S4_n$ (n>=1), $S5_n$, és gyenge-$S5_n$ (n>=2) logikákban exponenciálisan nehéz.

Foglaljuk most össze az ágensmodell alapveto tulajdonságait:

?? az ágens elhisz minden érvényes állítást,

?? az ágens hiedelmei logikai konzekvenciára nézve zártak,

?? az ekvivalens kijelentések ekvivalens hiedelmeket jelentenek,

?? ha ágens inkonzisztens, akkor mindent elhisz,

?? a legrosszabb esetben (worst-case) a logikai következtetések automatizálása nehéz,
?? a lehetséges világok, a hozzáférési relációk 'nyelve' nehézkes, nincs kapcsolatban az ágens architekturális felépítésével.

A bevezetett apparátussal modellezni lehet a józán ész (common sense) és az elosztott tudást is. Legyen annak jelolése, hogy ‘mindenki tudja, hogy (everybody knows p)’ EK p. Ekkor:

$EK\ p = K_1\ p \wedge K_2\ p \wedge \ldots \wedge K_n\ p$ vagy
$(M, w) \models EK\ p$ a.cs.a $(M, w) \models K_i\ p$ minden $i = \{1, \ldots, n\}$

Legyen annak a jelolése, hogy ‘mindenki tud k-fokig’ EK_k p. Ekkor:

$EK_1\ p = EK\ p$
$EK_k\ p = EK\ (EK_{k-1}\ p)$

A józán ész ('common sense') tudás a következoképpen fejezheto ki:

$CK\ p = EK\ p \wedge EK_2\ p \wedge EK_3\ p \wedge \ldots \wedge EK_k\ p \wedge \ldots$

Az elosztott tudás az összes ágens 'közös' tudása. Például:

A1: azt tudja, hogy: p
A2: azt tudja, hogy: p -> q

akkor Modus Ponens miatt ketten már q-t is tudják, így az eloszott tudáshoz q is tartozik:

$(M, w) \models DK\ p$ a.cs.a $(M, w') \models p$ minden olyan w'-re, amire (w, w') ? $(R_1 \wedge R_2 \wedge \ldots \wedge R_n)$

A bevezetett új operátorok nem függetlenek, hierarchiába rendezhetok:

$CK\ p$? ... ? $EK_k\ p$? ... ? $EK\ p$? $DK\ p$

5.6. Kvantifikált episztomikus logika

Mi lenne a helyzet a logikai modellel, ha a nagyobb ábrázoló ero érdekében kvantorokat vezetnénk be ? A klasszikus eésorendu logikába (predikátum kalkulusba) vezessük be a $\Box$ és ? operátorokat. Így dolgozhatunk pl.

?x. $\Box$P(x) ? Q(x)
P(a) ^ $\Box$?x.P(x) ... alakú állításokkal is.

Az itéletlogikához képést az atomikus formulákban most függvények (predikátumok) is állhatnak, azokban változók szerepelnek. Az állítások logikai értéke akkor van értelmezve, ha a függvényekben a változók konstansokkal vannak helyettesítve, vagy kvantifikáltak.

Megjegyzés: a valós problémában eloforduló objektumok identitása, tulajdonságai, relációjuk más objektumokkal éppen logikai konstansokkal, objektum-objektum függvényekkel és predikátumokkal írható le, hiszen a logikai apparátus összes többi eleme formálisan definiált.

Legyenek:
Const - a konstansok halmaza,
Var - a változók halmaza, és
Pred - a predikátumok halmaza,

akkor a modell: $(W, R, D, I, ?)$
ahol:
D - a konstansok jelentését megadó domain,
I: Const ? D,
?: Pred ? W ? powerset D^n.

Az elsorendu logika bevonása miatt érvényesek az un. Barcan-féle formulák (*):

? x . $\Box$P(x) ? $\Box$? x. P(x)
$\Box$? x.P(x) ? ? x . $\Box$P(x)

A Barcan-féle formulák következménye, hogy egy 'tudó' ágens az összes lehetséges esetet egyszerre is el tudja képzelni, illetve egy ágens véghez képes vinni az univerzális kvantifikálást. A Barcan-féle formulákat a szakma tulságosan megszorítónak és intuicióellenesnek tartja.

Az elsorendu logika használatanak további következényei az alabbiak:

- konstans domain feltételezése:

D rögzített kell, hogy legyen az összes w világra, mert ha a D világonként változik, akkor (*) nem lesz igaz. Bevezethetjük az un. merev (rigid) konstansokat és a változó (fluent) kifejezéseket, pl.:

□| Futbalista (Miniszterelnök) igaz, vagy hamis ?

Az állítás értéke a logikai konstans mögött rejlo 'igazi' konsytanstól függ, tehát pl. attól függoen, hogy Miniszterelnök = Horn -Gyula vagy Orbán-Viktor.
A probléma megoldására bevezethetünk egy új (un. bullet) operátort. Ha a egy logikai konstans, akkor ?a az a mögötte rejlo igazi, jelentést adó konstans, így. □| Futbalista (?Miniszterelnök) tudás a mindenkori miniszterelnökre vonatkozik.

- problémák vannak az egzisztenciális kvantorral is, pl. 'Tudjuk, hogy egyszarvú létezik' állítást lehetne modellezni:

?x . □|Egyszarvú (x), de akkor ezzel kijelentjük, hogy legalább egy egyszarvú egyed létezik, vagy pedig

□| ?x . Egyszarvú (x), de akkor az ágens hisz az egyszarvú egyed létezésében.

5.7. Rögzített (grounded) lehetséges világok

A rögzített (grouonded) lehetséges világok a lehetséges világok precizebb megfogalmazása, esetleges konfuziók elkerülése érdekében. Egy elosztott rendszer következo egyszeru modelljébol indulunk ki. A rendszer komponensek:
- a környezet, amely E állapotok valamelyikében van,
- az n db folyamat {1,..., n}, mindegyik L 'lokális' állapotok valamelyikében lehet,
- es az egész rendszer a G globális állapotok valamelyikében lehet G = E ? L ? ... ? L
- a rendszer futása ('run') egy olyan függvény, amely egy-egy globális állapotot egy idopillanathoz rendel hozzá:
  Run = N ? G
- egy pont ('point') maga a futás az idopontjával együtt:
  Point = Run ? N

A pontok a lehetséges világok szerepét töltenek majd. A rendszer a futások halmaza:
System = powerset Run

Legyen most s és s' két globális állapot: $s = (e, l_1, ..., l_n)$ és $s' = (e', l_1', ..., l_n')$. Az i-edik folyamatra definiáljuk a következo ekvivalencia relációt:

$s \sim_i s'$ a.cs.a, ha $(l_i = l_i')$,

tehát az s és s' állapotok az i-edik folyamat szempontjából nem különböztethetok meg. A folyamat lokális állapota az o 'tudása', ha két globális állapot semmiben sem különbözik, akkor inek mindkettoben ugyanolyan tudással kell rendelkeznie. Figyelembe véve, hogy egy folyamat akciói az o lokális állapotának függvénye, mindkét állapotban a folyamat ugyanilyen akciókhoz is fog folyamodni. Használjunk logikai leíró nyelvnek a K_i logikát. Akkor a szemantikai szabályok a következok:

(M, r, u) |= állítás

M = (R, ?) - az interpretált rendszer, és (r, u) egy pont,
R - a rendszer és
p: Point ? powerset Prop

(M, r, u) |= p ahol p ? Prop, a.cs.a ha p ? ? ((r,u)),
(M, r, u) |= K_i p a.cs.a (M, r' ,u') |= p minden r' ? R és u' ? N, úgy,
hogy: r (u) $\sim_i$ r' (u')

Érdekes az elosztott tudás operátora. Legyen s ~ s' a.cs.a ? i ? {1, ..., n} . s $\sim_i$ s'

(M, r, u) |= DK p a.cs.a (M', r', u') |= p minden r' ? R, és u' ? N, hogy r (u) ~ r'(u').

5.8. Logikai 'mindentudás' elkerülése

A logikai ‘mindentudás’ nem egy kívánatos tulajdonság. Elkerülésének egyik módja a hiedelem halmazának bontása az un. explicit és implicit hiedelmekre és a lehetséges világok helyett az un. 'helyzetek' bevezetése. Egy helyzetben egy állítás lehet:

- igaz,
- hamis,
- egyik sem, illetve
- mindketto (inkonzisztens helyzet).

A hiedelmek fajtáját operátor szinten tudni kell megkülönböztetni. Legyen:

B - az explicit hiedelem operátora és
L - az implicit hiedelem operátora.

Legyen továbbá a modell: (S, *B*, T, F), ahol:

S - a helyzetek halmaza,
B ? S - az olyan helyzetek halmaza, amelyek az ágens hiedelmeivel konzisztensek,
T: Prop ? powerset S (azok a helyzetek, ahol a hiedelmek igazak), és
F: Prop ? powerset S (azok a helyzetek, ahol a hiedelmek hamisak).

(M, s) $|=_T$ alátámasztja az állítás igaz voltát,
(M, s) $|=_F$ alátámasztja az állítás hamis voltát.

(M, s) $|=_T$ p p ? Prop, a.cs.a s ? T(p),
(M, s) $|=_F$ p p v Prop, a.cs.a s ? F(p),
(M, s) $|=_T$ p ? q a.cs.a (M, s) $|=_T$ p vagy (M, w) $|=_T$ q,
(M, s) $|=_F$ p ? q a.cs.a (M, s) $|=_F$ p és (M ,w) $|=_F$ q,
(M, s) $|=_T$ ~p a.cs.a (M, s) $|=_F$ p,
(M, s) $|=_F$ ~p a.cs.a (M, s) $|=_T$ p,
(M, s) $|=_T$ Bp a.cs.a (M, s') $|=_T$ p , minden s' ? *B*,
(M, s) $|=_F$ Bp a.cs.a (M, s) $|?_T$ B p.

Legyen W(s) olyan helyzetek halmaza, amelyek megegyeznek az s helyzettel az állítások igaz/ hamis értékét illetoen (ha s helyzet inkoherens, akkor W(s) = { }). Akkor:

(M, s) $|=_T$ L p a.cs.a (M, s') $|=_T$ p minden s' ? W (*B*)
(M, s) $|=_F$ L p a.cs.a (M, s) $|?_T$ L p
és
|= B p ? L p

Az explicit hiedelem nem zárt logikai a következtetésre nézve, abból kifolyólag az agensek lehetnek inkonzisztensek, úgy is, hogy semmit sem hisznek el. A logikailag ekvivalens állítások nem ekvivalens explicit hiedelmek, azonban a hiányosságok közé tartozik, hogy a logikai modellben:

- kvantorok ninncsenek,
- hiedelem beágyazás sincsen, és különben
- a helyzetek ötlete eléggé furcsa.

5.9. Általános 'tudomásul vétel' logika

Egy másik megoldás az un. általános ‘tudomásul vétel’ logika (logic of general awareness - LGA). Itt három modális operátorral találkozunk:

L - az implicit hiedelem operátora,
A - a 'tudomása van' operátor,

B - az explicit hiedelem származtatott operátora

A logika modellje: $(S, A, B, ?)$, ahol:
S - az állapotok halmaza,
$A: S ?$ powerset LGA - azok az állítások, amikrol agensnek tudomása van s állapotban,
$B ? S ? S$ tranzitív, soros és euklideszi reláció, és
$?: S ?$ powerset Prop.

(M, s) |= L p a.cs.a (M, s') |= p, minden s' ? S, (s, s') ? *B*,
(M,s) |= A p a.cs.a p ? *A*(s), és
B p = L p ^ A p

5.1? . Hiedelmek, célok, intenciók és a racionálítás

Eddig csak a tudás vagy a hiedelem ábrázolása volt a cél, de egy igazi logikai modellben az ágens episztemikus (tudás) állapotát össze kellene kapcsolni az o intencionális állapotával is. Erre szolgál a racionális ágenslogika (logic of rational agency).

Mit jelentsen az, hogy az intenciók és célok között racionális egyensuly van ? Az autonom ágenstol elvárjuk, hogy az intencióinak megfeloen cselekedjen, és nem velük szemben. Elvárjuk, hogy egy ágens olyan intenciókat 'alakítson ki', amiket kivitelezhetonek tart, és ejtse el azokat, amelyek teljesítésére nincs esélye. Az ágens az intencióival tartson ki, de nem tulságosan sokáig, és a teljesítettnek vélt intenciókat felejtse el. Alapveto vonás, hogy az ágens változtassa meg az intencióit, ha a hiedelmei lényegesen megváltoznak. A tervkészítés során folyamatosan segéd-, helyettesíto intenciókat is fogalmazzon meg.

Az intenció elméletnek meg lehet fogalmazni néhány kivánt tulajdonságát:

- az intenciók problémákat állítsanak ágensek elé, amiket az ágenseknek sorra meg kell oldaniuk,
- az intenciók szuroként hassanak más intenciók megfogalmazásánál (ne lehessen szó ellentmondásos intenciókról),
- az ágens kövesse figyelemmel az intencióinak sikerességét, és hajlandó legyen ismételten próbálkozni, ha nem járt sikerrel,
- az ágens azt higye, hogy az intenciói lehetségesek,
- az ágens ne higye, hogy az intencióit nem lehet megvalósítani,;
- bizonyos körülmények között az ágens azt higye, hogy az intenciói megvalósulnak,
- az ágensnek nem szükséges elhinni az intencióinak összes várható mellékhatását.

Hogyan néz ki a racionális ágenslogika (logic of rational agency - LRA) ? LRA egy elsorendu, multi-modális logika, egyenloséggel. A modell modalitásainak (modál operátorainak) egy része az ágens tudásállapotára vonatkozik, maradó modalitások az idomodellt írják le. Az alapveto modalitások a következok:

(BEL x p) x ágens hiszi a p-t,
(GOAL x p) x ágens p céllal rendelkezik,
(HAPPENS a) a akció következik,
(DONE a) a akció éppen megtörtént.

Minden egyes ágens esetén a hiedelem hozzáférési relációja euklideszi, tranzitív és soros (KD45 logika), a cél hozzáférési relációja soros (KD logika), és a célok ? hiedelmek (avagy a célokat is el kell hinni !).

Bonyolultabb akciók az elemi akciókból szerkeszthetok, pl.:

a;a' - a' akció követi az a akciót,
a? - egy teszt akció.

A tisztán idomodálitások a következok:

□ - mindig (always)
? - néha (sometimes)
LATER - az un. szigorú néha

?a = ? x . (HAPPENS x;a?)
□a = ? ?? a
(LATER p) = ? p ^ ?p

fontos feltételezés, hogy a célokat később (elobb-utóbb) levesszük az agendáról:

?? (GOAL x (LATER p))

Igen fontos ha a cél tartós (persistent goal):

(P_GOAL x p) = (GOAL x (LATER p)) ^ (BEL x ? p) ^
(BEFORE ((BEL x p) ? (BEL x □? p)) ? (GOALx (LATER p)))

magyarán, egy ágensnek tartós célja van, ha:
- van p célja, amely majd igaz lesz, de most agens még nem hiszi annak,
- a cél elejtése elott az agensnek:
  - hinnie kell, hogy a p célt megvalósította(k), vagy pedig
  - hinnie kell, hogy a p célt sohasem lehet megvalósítani.

Hasonló módon modellezheto a szándék kifejezése:

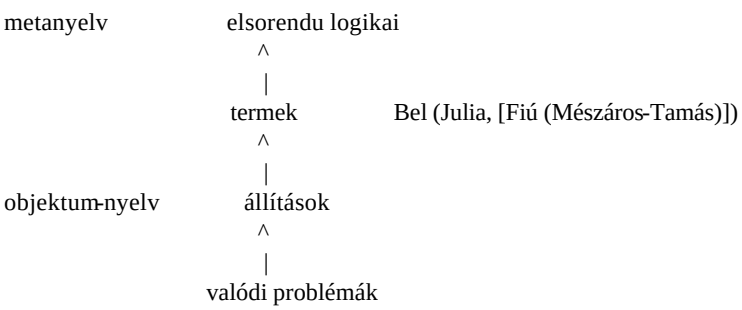
(INTEND x a) = (P_GOAL x [DONE x (BEL x (HAPPENS a))?;a])

A teljes LRA logika áeírása a függelékben található.

5.11. Metanyelvek

Emlékezzük, hogy a 'tudja, hogy' szintaktikai problémái miatt kilátásba helyeztük a metanyelv használatát. A logikai nyelv és a metanyelv viszonyát az 5.1. ábra mutatja.

5.1. ábra. Logikai metanyelv szerepe.



A metanyelv megoldásnak alapveto hiányossága, hogy az objektum-nyelv termjeire nem lehet univerzálisan kvantálni, mert a valós világra vonatkozó állítások meta-nyelv domenjében csupán egyszru objektumok (logikai konstansok). A metanyelv megoldásnak azért elönyos a tekintélyes kifejezo ereje, pl.:

?x . Bel (i, x) 'i hisz valamit'
? x . Bel (i, x) ? Bel (j, x) 'i mindazt hiszi, amit j'

állítások modális logikában lehetetlenek lennének. Egy másik elöny a számíthatóság, hiszen az elsorendu logikánál maradunk. A félig eldöntheto elsorendu logikában a rezolúciós bizonyítás köönyen algoritmizálható.

A metanyelv megközelítés azért sem mentes komoly problémáktól, amilyenek az önhivatkozások és az inkonzisztencia. Alapveto hozzáállás, hogy a'tudás = biztos hiedelem', tehát:

Know ([p]) ? Bel ([p]) ^ True ([p]),

no, de hogyan értelmezhetjük a True függvényt (igaz predikátumot metanyelv szintjén) ? A p állítás az objektum-nyelvben nyílvánvalóan vagy igaz, vagy hamis állítás, de meta-nyelvben az csak egy term, aminek önálló logikai értéke nincs ! Egy természetes axióma lehetne:

True ([p]) ? p (*)

Ez azonban azt jelenti, hogy szükségünk van egy olyan logikai nyelvre, ahol lehetséges az önhivatkozás (pl. meta-nyelv = objektum-nyelv választással), de bizonyítható, hogy minden elsorendu logika, a (*) axiómával együtt inkonzisztens rendszert alkot (Tarski). További probléma, az hogy természetesnek látszó:

Bel ([p]) ? p (axióma)
|= p -bol következik |= Bel ([p]) (következtetési szabály)

a kelloen gazdag logikai elméletben (pl. aritmetika) már inkonzisztens (a konzisztencia bizonyos módosításokkal visszanyerheto).

Önhivatkozó metanyelv alternativája egy hierarchikus metanyelv:

L_0 - L_1 - L_2 - ... - L_k - ...,

ahol L_0: egy nem önhivatkozó (közönséges elsorendu logika), és minden k-ra L_k egy olyan meta-nyelv, amelynek termjei az ‘alatta’ lévo nyelvek álltásai. Így az önhivatkozás paradoxonjai eltünnek, de problémák azért nem. Problémás pl. a [A elhiszi, hogy minden, amit B hisz, igaz] jellegu állítások modellezése (ami ágensek között azért elönyos lehetne). Az állítás felírása L_0-ban nem megy. Magasabb nyelvben lehetne:

Bel (A, [? x . Bel (B, x) ? True (x)]),

de az A és B így két külön nyelvi szinten jelenik meg, és az univerzális kvantifikálás vagy nem a B hiedelmei, vagy nem az A hiedelmei szerint történik. Egy ilyen kijelentés meta-nyelvekben nem valósítható meg jól.

5.12. Több ágenses rendszer mint számítási paradigma

Érdekes kisérlet a több ágensbol álló rendszer számítási mechanizmusként (paradigmaként) megfogalmazni. Ehhez meg kell fogalmazni az egyedi ágens modelljét és az ágenseket összkapcsoló végrehajtási modellt.

5.2. ábra. Ágens modell komponensei.

Egy ágens a valamilyen belso L (logikai) nyelven kifejezett hiedelmekból és akciókból áll. Akciói lehetnek (1) kognitív akciók, (2) kommunikációs akciók, és (3) 'fizikai' akciók. Kognitív akciók az ágens saját számítási eroforrásaira épülnek. Ezek 'privát' akciók, más ágensek által nem észlelhetok. Az ágensnek kontrollja van felettük, más ágens azokba az akciókba nem szólhat bele. Kommunikációs akciók az üzenetküldések. Ezek az akciók más ágensek kognitv állapotát befolyásolják. Az ágensnek nincs teljes kontrollja az ilyen akciók felett. Kommunikációs akciók megfelelo elvégzéséhez ágensnek tudnia kell jósolni, tervezni, és más deliberatív taszkokat elvégezni. Fizikai 'kieszközölo' akciók a 'normál' akciók. Ezekkel a számítási modellben nem foglalkozunk.

5.3. ábra. Az ágens információs modellje.

Továbbiakban adjuk meg a modell egyes elemeinek pontosabb és formálizált definicióját.

Kognitív akció az új tudás kikövetkeztetése, azaz:

hiedelmek ? episztemikus (tudás) input

Kommunikációs akció kúlcsa az ágens üzenetinterpretó képessége:

interpretálás ? [üzenet értelmezése a vevo kognitív állapotában]

minden ágensnek természetesen saját interpretációs 'képessége' lehet, az interpretálás a:

hiedelmek ? üzenetek ? episztemikus input

Az ágens információs modelljének egyik legfontosabb komponense (és egyben az ágens egyik legfontosabb képessége) a hiedelem felfrissítési képesség (belief revision):

hiedelmek ? episztemikus input ? új hiedelmek

Az ágens végrehajtási modellrol itt beszélni nem fogunk. A végrehajtási modellnél feltétlenül szükséges valamilféle vezérlési modell, amely sokféle lehet, általában bonyolúlt és valamilyen jellegu tervezésre alapoz (ld. tervkészítés). Fontosak még a standart elnevezések, amelyeknél az ágens közösségben koncenzusra kell törekedni, máskülönben a helyes interpretáció lehetetlen lesz.

Térjünk most rá az egyes komponensek pontosabb formális kifejezésére. Az ágens hiedelemei a lehetséges logikai állítások valamilyen részhalmaza:

Belset = powerset Form(L)
close(?, ?) = {p | ? |-? p} a hiedelmek deduktív lezártja (azaz ami belülük tovább kikövetkeztetheto), itt ? az állítások halmaza és ? a deduktív szabályok halmaza.

Az episztemikus bemenet: Epin = powerset Form(L)

és a hiedelem felfrissítés: Brf = Belset ? powerset Epin ? Belset

Kommunikáció leírásához meg kell adni az ágens üzenetvételi és üzenetadási képességét. Legyenek az ágensek azonosítói:

Agid = tetszoleges megszámlálható halmaz, és az ágensek rendre: i, j, k,...

Az üzenetek halmaza a:

Mess = Agid ? Agid ? Form(L), azaz egy üzenet ? = (küldoágens, vevoágens, tartalom)

és egy ágens lehet: sender (?), vagy recvr (?), azontúl nyílvánvaló. hogy:

? ? ? Mess . (sender (?) ? recvr (?))

Az ágens fontos képessége az üzenetinterpretálás:

Messint = Belset ? Mess ? Epint

Egy kognitív akció egy feltétel? következmény szabály. Szabály halmaz lehet 'gyengén teljes' (weakly complete), ha minden hiedelmi állapothoz létezik legalább egy akció, vagy üzenet, amit alkalmazni, ill. küldeni lehet. Egy ? akció feltétel és következmény a:

Action = Belset ? Epin
Cond = Form(L) ? {true}

és az akciók, ill. üzenetek halmaza:

Arule = Cond ? Action
Mrule = Cond ? Mess

Egy akció alkalmazható:

ac_applic ((p, ?), ?) = p ? (? ? {true}), azaz ha az ágens elhiszi annak feltételeit.

Az akciók halmaza gyenge teljes, ha:

ac_wk_complt (AR) = ? ? ? Belset . ? ar ? AR .ac_applic(ar, ?)

Egy akció legális, ha:

ac_legal (? , AR, ?) = ? (p, ? ') ? AR . (? = ? ') ^ ac_applic ((p, ? '), ?)

és hasonlóan definiálható egy legális üzenet is.

Egy üzenet öszinte:

honest (i, MR) = ? (p, m) ? MR . (sender (?) = i)

Az ágens logikai architektúrája ezek után a: (?₇, ?, ?, ?, MR, AR), ahol:
?₇ - a kezdeti hiedelmek,
? ? Drule - az L nyelv dedukciós szabályai,
? ? Brf - a 'belief revision' függvény,
?? Messint - az üzenet interpretációs függvény,
MR ? Mrule - az agens üzenetszabály halmaza,
AR ? Arule - az agens akciószabály halmaza.

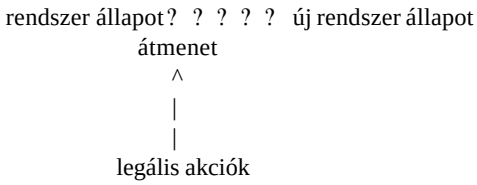
Az ágens muködése ciklikus:
(1) a vett üzenetek interpretálása;
(2) a hiedelem felfrissítés episztemikus bemenetekbol brf függvény segítségével:
(a) az elozo akciókból, és
(b) az üzenetinterpretációkból;
(3) a hiedelem halmaz deduktív lezártjának a számítása;
(4) a lehetséges küldendo üzenetek meghatározása;
(5) egy üzenet kiválasztása és elküldése;
(6) a lehetséges (belso kognitív) akciók meghatározása;
(7) egy akció kiválasztása és végrehajtása;

A több agensbol álló rendszer modellje struktúrában hasonló: (Ag, ?₇, ?, ?, ?, MR, AR), ahol:
Ag = az agensazonosítók halmaza,
?₇ = Ag ? m? Belset,
? = Ag ? m? powerset Drule,
? = Ag ? m? Brf,
?= Ag ? m? Messint,
MR = Ag ? m? powerset Mrule,
AR = Ag ? m? powerset Arule,
avagy a korábbi definiciókat az egyes agensekre külön-külön (és differenciáltan) alkalmazzuk (? m? az ágensek halmazának leképzése a jobb oldalon álló megfelelo halmazokra, azaz az elobb specifikált halmazokat most mindegyil ágenshez külön-külön rendeljük hozzá).

Kijelenthetjük pl., hogy:
? i ? Ag . honest (i, MR (i)) stb.

Az ágens közösség muködtetéséhez definiálni kell a megfelelo végrehajtási modellt, amely tipikusan lehet szinkronizált, vagy átlapolt. Egy ilyen modell 'potenciális világokat' biztosít modális tudáslogika számára.

Szinkronizált végrehajtásnál:



5.XX. ábra. A rendszerállapot változása és a rendszer akciói.

A rendszer állapota: State = Agid ? m? Belset, és
a kezdeti állapota:

init_state = System ? m? State
init_state (Sys) = {i ? m? close (? ? (i), ? (i)) | i ? Ag}.

A rendszer lépése: Move = Action ? Mess, egy legális lépése viszont:

mv_legal (?, ag, ?) = ac_legal (action (?), AR, ?) ^ ms_legal (mess (m), MR, ?)

A rendszer állapotátmenete: Trans = Agid ? m? Move, és egy legális állapotátmenete:

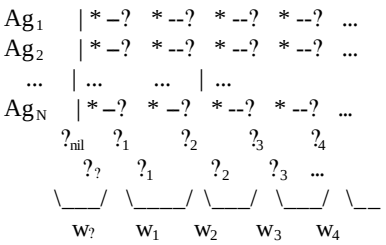
trans_legal (?, sys, ?) = ? i ? dom ? . mv_legal (? (i), agent (i, sys), ? (i)).

Az állapotátmenetkor elküldett üzenetek:

sent (?) = { mess (?) | ? ? rng ? } + Messnil,

és az állapotátmenetkor vett üzenetek

recrd (i, ?) = { ? | sent (?) ^ ? ? (recvr (?) = i)}.



5.XX. ábra. A rendszer futása (run) a világok és világszekvenciák.

A rendszerállapotátmenetkor keletkezo következo hiedelemhalmaz:

next_bel (ag, ?, ms, ?) = close (? (?, {? (?)} ? {i (?, ?) | ?? ms}), ?)

ahol a ? függvény argumentumában szereplo unió az episztemikus bemenetek uniója.

A következo rendszerállapot:

next_state (sys, ?, ?) = {i ? m? nex_bel (agent (i, sys), ? (i), recvr (i, ?), action (? (i))) | i? dom ?}

Az ágensvilág a: World = State ? Trans. és a következo világ:

next_world (w, w', sys) = trans_legal (trans (w'), sys, state (w)) ^
(state (w') = next_state (sys, state (w), trans (w')))

A kezdeti világ:

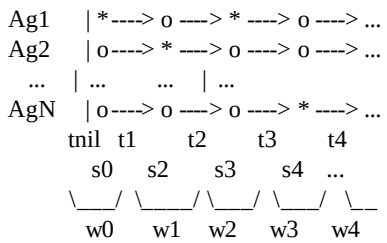
init_world (sys) = (init_state (sys), nil_trans (sys))

Végük a világok sorozata:

Worldseq = World, és a rendszer futása:

run_of (W, sys) = (W(?) = init_world (sys)) ^ ? u ? N . next_world (W(u-1), W(u), sys)

Az átlapolt végrehajtásnál (interleaved executions) legfeljebb 1 ágens muködik egyszerre egy adott pillanatban:



5.XX. ábra. A rendszer futása (run) a világok és világszekvenciák.

Mivel nem mindegyik ágens aktív lesz a mindekori világállapotátmenetkor, bizonyos elküldett üzenetek nem kerülnek azonnali feldolgozásra. El kell képezni az üzenetek készletét (message pool), amibol meríteni fog az éppen aktív ágens.

Pool = powerset Mess, és

State' = (Agid ? m? Belset) ? Pool

init_state' (sys) = (..... , {}), ahol az üres halmaz a kezdeti üres üzenetkészletet jelenti.

recvd' (i, ?) = { ? | pool (?) ^ ? ? (recvr (?) = i) }

A rendszer állapotátmenete:

? ?? Trans' . ?! i ? dom ? . (? (i) ? nil), (a ?! jelentése ‘létezik egyetlenegy’),
trans_legal' (?, sys, ?) = ...
sent' (?) = ...
next_bel_state = ...
next_pool_state = ...
next_state = ...

5.13. AL logika

Definiáljuk most az ágens részére az AL temporális tudás logikát. Logika alapelemeihez tartoznak:
- szimbólumok: {true, Bel, Send, Do},
- konstansok halmaza: Const = $Const_{Ag}$? $Const_{Ac}$ (ágensek és akciók nevei),

- az L belso nyelv minden zárt állítása (azaz minden változó helyen valamilyen konstans szerepel),
- ? , ? operátorok,
- unáris temporális operátorok { ○, ● } és bináris temporális operátorok { $\mathcal{U}$, $\mathcal{S}$ },
-), (szimbólumok.

A logika szintaktikai szabályai:

- ha i,j az ágensek azonosítói, p egy zárt állítás L-ben, ? egy akció azonosító, akkor:
  true, (Bel i p), (Send i j p), (Do i ?) állítások AL-ben,
- ha p,q állítások AL-ben, akkor: ? p, p ? q szintén állítások AL-ben,
- ha p,q állítások AL-ben, akkor: ○p, ●p, p $\mathcal{U}$ q, p $\mathcal{S}$ q szintén állítások AL-ben.

Adjuk meg az operátorok értelmezését:

(Bel i p) - i agens elhiszi a p-t,
(Send i j p) - i agens p állítást küld j agensnek,
(Do i ?) - i agens ? akciót tesz,
○p - következo p (NEXT),
●p - utóljára p (LAST), az un. 'eros' LAST, kezdetkor False,
p $\mathcal{U}$ q - p addig tart, amig nem következik q, p elotte van q-nak (p until q),
p $\mathcal{S}$ q - p q óta tart, p q után keletkezett (p since q),

A logika szemantikai szabályainak alapja a rendszer futása. A rendszer futása egy világszekvencia, ez adja meg az operátorok értelmezését:

(W, Ag, Ac, I), ahol:

W ? Worldseq,
Ag - agensek halmaza,
Ac - akciók halmaza,
I = Const ? m? (Ag ? Ac), a konstansok interpretálása.

Egy állítást igaznak vesszük egy modellben, ha: (M, u) |= p, u ? N. A bonyolultabb kifejezések értelmezése a következo:

(M,u) |= true,	
(M, u)|= (Bel i p)	a.cs.a p ? State (W(u)) I(i)), az i ágens hiedelemi halmaza a W világszekvencia u idopontjában,
(M, u) |= (Do i ?)	action (trans (W(u)) (I(i))) = I(?),
(M, u) |= (Send i j p)	(I(i), I(j), p) ? sent (trans (W(u)),
(M, u) |= ? p	(M, u) |? p,
(M, u) |= p ? q	(M, u) |= p vagy (M, u) |= q,
(M, u) |= ○p	(M, u+1) |= p,
(M, u) |= ●p	u > 0 és (M, u-1) |= p,
(M, u) |= p $\mathcal{U}$ q	? v ? N . v ? u (M, v) |= q és (M, w) |= p, ? w ? N . u ? w < v,
(M,u) |= p $\mathcal{S}$ q	? v ? {0,..., u-1} . (M, v) |= q és (M, w) |= p, ? w ? N . v < w < u.

Érvényesek természetesen a szokásos szimbólikus átirások

p ? q = ? p ? q
p ^ q = ? (? p ? ? q)
p ? ((p ? q) ^ (q ? p))

Vezessük be további temporális operátorokat a következo módon:

■ - 'gyenge' LAST, kezdetkor True,
□ - ‘always’, □p a.cs.a, ha p most igaz és minden jövoben is,
■ - ‘heretofore’, ■p a □p múltideju verziója (szigorúan vett múlt),

? - ‘sometimes’, most, vagy legalább egyszer a jövoben,
◈ - 'was', szigorúan múltideju verzió (legalább egyszer a múltban),
$\mathcal{U}$ - ‘until’, p teljesül mielott q teljesülne, de q -nak valamikor teljesülnie kell,
$\mathcal{W}$ - ‘unless’ (weak until), -//-, de q esetleg sohasem teljesül,
$\mathcal{S}$ - ‘since’, szigorúan múlt verziója az $\mathcal{U}$-nak és $\mathcal{W}$-nek,
$\mathcal{Z}$ - ‘zince’, –//–,
$\mathcal{B}$ - p $\mathcal{B}$ q, p szigorúan megelozi q-t,
init - kezdetkor True.

Az új operátorok definiciós kapcsolata korábbiakkal

●p = ? ○? p	(gyenge) utóljára p,
init = ? ○true	kezdetkor,
?p = true $\mathcal{U}$ p	néha p,
□p = ? ?? p	mindig p,
p $\mathcal{W}$ q = □p ? p $\mathcal{U}$ q	p amíg q,
◈p = true $\mathcal{S}$ p	p volt,
■p = ? ◈? p	mindig volt p,
p $\mathcal{Z}$ q = ■p ? p $\mathcal{S}$ q	q óta p,
p $\mathcal{B}$ q = ? ((? p) $\mathcal{U}$ q	p megelozi q-t,

A logika bizonyítási elmélete (kis (legkisebb)) axiómahalmazból ás (kis) következtetési szabály halmazból áll- Az axiómák tautologiák (mindig érvényes állítások), jelük:

|- p

és érvényesek továbbá az ? , ? , ?, ^ operátorok klasszikus szabályai.

A logika alapaxiómája (attachment axiom) a:

|- ((Bel i p1) ^ ... ^ (Bel i pn)) ? (Bel i p)

ahol: {p1,..., pn} $|-_{?(i)}$ p

Fontosabb axiómákhoz tartoznak:

|- ○? p ? ? ○p
|- p ? ?p
|- ○p ? ?p
|- □p ? ?p
|- p $\mathcal{U}$ q ? ?q
|- ○p ? ◈p
|- ■p ? ◈p
|- p $\mathcal{S}$ q ? ◈q
|- ?(p ?q) ? (?p? ?q)
|- □(p^q) ? (□p ^ □q)
|- □(p ? q) ? (□p ? □q)
|- □(p ? q) ? (?p ? ?q)
|- □(p ? q) ? (○p ? ○q)
|- ?p ? p Ú ○?p
|- □p ? p ^ ○ □p
stb.

A logika legfontosabb következtetési szabálya a:

(|- p ? q) és (|- p) -bol következik (|- q)

ez nem más, mint Modus Ponens, amely érvényes, mert a logikai operátorok klasszikusak.

Más következtetési szabályok tartalmazzák temporális következtetéseket is:

(|- p) -bol következik (|- ○p)
(|- ?p)
(|- □p)
(|- ●p)
(|- ◈p)
(|- ■p)

Fogalmazzük meg néhány lehetséges tételt illetve ágens axiómát:

(K) (Bel i p ? q) ? ((Bel i p) ? (Bel i q))
(D) (Bel i p) ? ? (Bel i ? p)

Tétel 5.1: Egy konzisztens kezdeti hiedelem halmazzal, jól definiált dedukciós szabályokkal és konzisztens brf függvénnyel rendelkezo ágens globálisan konzisztens.

(4) (Bel i p) ? (Bel i (Bel i p)) (pozitív introspekció)
(5) ? (Bel i p) ? (Bel i ? (Bel i p)) (negatív introspekció)

Idoben cselekvo ágensnek kell, hogy legyen tudomása az idorol (temporal awareness) és kivánatos a hiedelmeinek tartóssága (persistence of belief):

- eros 'temporal awareness': (Bel i p) ? ○(Bel i ○p)
- gyenge 'temporal awareness': ((Bel i ◈p) ? (Bel i p)) ? ○(Bel i ◈p)
- tartós hiedelmu ágens: (Bel i p) ? □(Bel i p).

Tétel 5.2: Ha egy ágens globálisan konzisztens és tartós hiedelmu, akkor:
(Bel i p) ? □? (Bel i ? p)

azonban egy ilyen ágens a véleményét megváltoztatni nem tudja. Jobb egy 'addig-ameddig vélemény':

- default persistence: ((○(Bel i p)) ^ ? (Bel i ? p)) ? (Bel i p)

Egy ágens lehet tovabbá:

- 'értékelo' (anticipating): ha tudja, mit küld és mit csinál:

(Send i j p) ? (Bel i (Send i j p))
(Do i ?) ? (Bel i (Do i ?))

- 'üzenetet értékelo' (message retentive)

(Rcv i j p) ? (Bel i (Rcv i j p))
(Rcv i j p) ? (Send j i p)

- jövobeli irányított hiedelmei lehetnek:

(Bel i ○(Do i ?)) ? ○(Do i ?)
(Bel i ○(Send i j p)) ? ○(Send i j p)
.....
stb.

5.XX. ábra. Az AL logika összefoglalása.

	agens	ido
nyelv	hiedelem:	modális temporális
logika		
	szentencionális modális operátor: (Bel I p)	múlt/jövo operátorokkal ○ ○ ● ? ◆ □ ■ U W S Z B
	akció:	
	szentencionális modális operátor: (Do i ?)	
	kommunikáció:	
	szentencionális modális operátor: (Send i j p)	
modell	hiedelem: dedukciós	
	modell	lineáris, diszkrét, véges múlt
	akciók: primitív	
	akciók	végtelen jövo
	kommunikáció:	
	iranyított üzenetek	

5.14. Hogyan tovább ?

Az ágens logikai modelljét bonyolíthatjuk (1) kvantorok bevezetésével, illetve (2) a belso nyelv gazdagabb megválasztásával. Egy ilyen kisérlet az IAL 'internal AL' = FOTL (First Order Temporal Logic), amely egy elsorendu logika (predikátum kalkulus) temporális operátorok (AL-bol) kibovítve.

pl. ? x . ○P(x) ? Q(x) (FOTL)
(Bel I P(a)) (AL)
(Bel i P(a)) ^ ? x . ○P(x) ? Q(x) (IAL)

Az elágazó idomodell bevezetésével az AL logikát fel tudjuk használn I a tudás/hiedelem kezelésére. Az ilyen BAL logikában a többi operátor mellé vezessük be Ap operátort:

Ap igaz, ha igaz az elágazó jövo mindegyikében, azaz

(M, p) |= Ap a.cs.a (M, p') |= p, minden olyan p' ? paths (W, R), hogy head p' = head p.

Származtatott operátor a: Ep = ? A? p p igaz valamilyen pályán (jövoben),
a többi operátor megegyezik az AL logika-beli definiciókkal.

Izelítonek néhány lehetséges axióma:

Ap ? p
Ep ? p ha p atomikus
p ? Ap -//-
A(p ? q)
(Ap ? Aq)
Ap ? AAp
Ep ? AEp
A ○p ? ○Ap
stb..

6. Multi-ágens rendszerek

A számítógépes hálózatok technológiájának elterjedése a mesterséges intelligencia területén is elotérbe helyezte az intelligens rendszerek összekapcsolásának kérdéskörét, illetve elosztott probléma megoldó rendszerek tervezését. Az **elosztott mesterséges intelligencia (distributed artificial intelligence, DAI)** a MI azon részterülete, mely intelligens rendszerek összekapcsolásával létrejövo "közösségek" tudásábrázolási, kommunikációs és probléma megoldó technikáit vizsgálja. Az általános cél az, hogy a lazán kapcsolt intelligens komponensek közössége, az un. **multi-ágens (MAS)** rendszer egy közös probléma olyan megoldását szolgáltassa, mely túlmutat az egyes komponensek önálló probléma megoldó képességén.

A MAS rendszerek létrehozásának motivációi a következoek:
- egy különálló ágens eroforrásain túlnövo problémák megoldásának támogatása,
- a különálló örökölt rendszerek (legacy systems) bekapcsolásának lehetosége,
- eredendoen elosztott problémák megoldásának támogatása (pl. távközlés menedzsment),
- elosztott információs forrásokkal rendelkezo problémák megoldása,
- eloszott tudást tartalmazó problémák megoldása (pl. egészségügyi rendszerek),
- a probléma megoldás sebességének, megbízhatóságának és skálázhatóságának biztosítása,
- tisztább, egyszerubb koncepcionális tervezés

A MAS rendszereket alapvetoen két nagy csoportba sorolják: kooperatív, illetve kompetitív rendszerek. A kooperativ rendszerekben a hangsúlyt az együttmuködésre helyezik: az egyes résztvevok alapveto tervezési szempontja, hogy a közös cél érdekében végzik tevékenységüket. Ezt a teljes rendszer egységes tervezésével biztosítják. A kompetitív rendszerekben az ágensek független módon tervezett, egyénileg motivált rendszerek, melyek között a versengés alakítja ki a megkívánt koordinációt. Ez utóbbi területtel részleteiben a játékelmélet foglalkozik.

A gyakorlatban megvalósított tisztán kooperativ rendszerek is tartalmaznak kompetitív elemeket, elsosorban a részvevo ágensek konfliktusban álló részcéljai miatt. Az ellentmondásos részcélok feloldásához szükséges a kommunikáció, koordináció és egyeztetés megoldása, melyek így a MAS rendszerek tervezésének alapfeladatai. A három terület közül a kommunikáció megvalósítása az elsodleges, hiszen ez szükséges a másik két problémakör kezeléséhez is.

Ágensek közötti kommunikáció

Az ágensek közötti kommunikáció alapfeltétele egy közös **ágens kommunikációs nyelv (agent communication language, ACL)** definiálása és elfogadása. Ezen nyelveknek jelenleg nem létezik egyetlen közösen elfogadott változata. Legelterjedtebb, és szabványosodó nyelv a **KQML (Knowledge Query and Manipulation Language – tudás lekérdezo és manipuláló nyelv)**. Az ágens kommunikációs nyelvek közös kiindulási alapja az ún **szólás aktusok (speech act)** elmélete, melyet a következokben foglalunk össze.

Szólás aktusok elmélete

Az elmélet a természetes nyelvek megismerésének pragmatikus megközelítése, ahol a nyelvészek a hangsúlyt annak megismerésére helyezik, hogy az emberek a köznapi nyelvi formákat, mint például a kérést, parancsot, igéretet, kívánságot, stb., hogyan használják mindennapos feladataik megoldására.

A kommunikáció céljai és formái

A kommunikációs rendszer kialakításának alapveto célja az önállóan cselekvo entitásokat tartalmazó rendszerben a koordináció és a koherencia biztosítása. Ez alapvetoen két részterületre bontható szét: az adatcsere megvalósítására és a kölcsönös megértés (mutual understanding) biztosítására. Az adatcsere az ágensek közötti adatmozgást valósítja meg az interakciós protokoll, kommunikációs nyelv és adatátviteli protokoll definiálásával. A kölcsönös megértés szükséges egyrészt egy közös tudás reprezentációs nyelv, vagy a reprezentációk közötti fordítás, illetve a tudás azonos értelmezésének biztosítása.

Általánosságban a következo kommunikációs formákat határozhatjuk meg:
- *nincs kommunikáció*: az ágensek egymás viselkedését csak közvetve befolyásolják, általában ez igaz a kompetitív alapú rendszerekre, melyekkel a játékelmélet foglalkozik,
- *jelzések*: az ágensek rövid jelzéseket küldenek egymásnak, melyek alapvetoen szinkronizációs feladatokat oldhatnak meg,

?? *üzenetküldés*: az ágensek szabályozott tartalmú üzeneteket küldhetnek egymásnak, melyek – ha szuk kontextus körben is, de - a szinkronizáción kívül már adatokat/tudást is átvihetnek,
?? *tervküldés*: az üzenetküldés egy speciális formája, amikor az ágensek kölcsönösen egyeztetik terveiket valamilyen globális tervezo algoritmus szerint,
?? *blackboard*: az ágensek egy osztott információ tároló helyen (blackboard) cserélnek információt a blackboard rendszerekben megismertek szerint,
?? *szólás aktusok*: az ágensek közötti kommunikáció a fizikai akciókhoz hasonló akciókból állhat, melyek magukban hordozzák az adataik kölcsönösen egyértelmu értelmezéséhez szükséges kiegészítéseket is.

A KQML ágens kommunikációs rendszer

A KQML valójában több, mint egy ágens kommunikációs nyelv – az ajánlás a kommunikáció, üzenet tartalom, és kontextus szinteken kívül tartalmaz architekturális elemeket is. A következokben a rendszert tervezésének bemutatásának keresztül ismertetjük.

Tervezési szempontok és feltételezések

?? *Architektúra*: az ágensek processzorokon futó folyamatok, melyek között a kommunikációt szabványos hálózati protokollok (TCP/IP) segítségével kell megvalósítani. Az alap protokollok felett egyszeru és jól skálázható alkalmazás szintu protokollokra van szükség.

?? *Kommunikációs módok*: legyen többféle: pont-pont, többes címzésu (multicast). Legyen többféle címzési mód: direkt, név szerinti, ágens leírás szerinti. Legyen többféle szinkronitású: blokkoló és nem blokkoló.

?? *Szintaktikai feltételezések*: a kommunikáció legalsó szintje közömbös (pl. ASCII bájtfolyam), a felsobb szinteken már nyelvfüggo. Ajánlott, hogy ember által is olvasható és értelmezheto legyen.

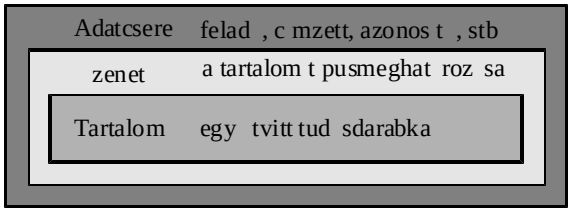
?? *Biztonsági szempontok*: a rendszer támogassa az egyértelmu partnerazonosítást, az üzenetek integritásának és titkosságának védelmét.

?? *Tranzakció megközelítés*: az adatbázisoknál megismert tranzakció típusú üzenetküldés MI rendszerek esetén nem alkalmazható, mivel az állítások visszavonása ezekben késobbi tranzakciók során is bekövetkezhet. A MI területén speciális módszerek léteznek e kérdéskör kezeléséhez, melyeket az üzenetküldés mechanizmusának kidolgozása során figyelembe kell venni.

A kommunikáció szintjei

A számítógép-hálózatoknál megismert beágyazás (encapsulation) elve szerint az ágens-ágens kommunikációs protokollnak is több rétege van. Az állomások között közlekedo üzenet csomagok legkülso rétege, azaz a kommunikáció legalsó protokoll szintje az un. **adatcsere szint**, mely az átküldött üzenet feladóját, címzettjét, azonosítóját, illetve egyéb kommunikációs paraméterét tartalmazza. A következo réteg, az **üzenet szint** az üzenet tartalmának azonosítását, típusmeghatározását segíti, míg a legbelso réteg, azaz a **tartalom szint** közvetíti az átadott információt.

A tartalmi szintre megkötés nincs, bármit tartalmazhat, amiben a kommunikáló ágensek megegyeznek. Bár a formára sincs megkötés, a KQML implementációk általában a LISP nyelvben megismert listás adatszerkezetet használják.



6.1. ábra: a KQML kommunikáció logikai szintjei

Az üzenet szint szerepe kétféle lehet: adminisztratív vagy tartalom jellegu. Az adminisztrativ üzenetek tartalma kötött, a rendszer muködéséhez szükséges adatokat közvetíti. Ilyenek például az új ágensek bemutatkozása, ágens feladatmegoldó képességek közzététele, illetve egy ágens által igényelt adatok

leírását. A tartalom típusú üzenetek egy tudásdarabka átvitelére szolgálnak, meghatározva annak ontológiáját, témáját, nyelvét és tartalmát. A következo két példa egy tartalom és egy adminisztratív jellegu üzeneti szintet mutat.

```
(QUERY
        MODIFIERS             (number-of-answers 1)
        CONTENT-LANGUAGE           KIF
        CONTENT-ONTOLOGY           block-world
        CONTENT-DESCRIPTION   physical-property
        CONTENT               (color block1 ?color)
        )
```

6.1. példa: KQML tartalom jellegu üzenet

Az elso példa egy kérés leírását mutatja be, melyben a kocka világban a KIF nyelv használatával fogalmazunk meg egy fizikai tulajdonságra vonatkozó kérdést: a *block1* nevu kocka szinére vagyunk kiváncsiak. A következo példa egy ágens “hirdetését” mutatja be, melyben az ágens azt a képességét fogalmazza meg, hogy képes kockák szinére utaló állításokat tenni (azaz ilyen kérdésekre válaszolni).

```
(ADVERTISE
        DIRECTION  EXPORT
        (TELL
             CONTENT-LANGUAGE      KIF
             CONTENT-ONTOLOGY      block-world
             CONTENT-DESCRIPTION   physical-property
             CONTENT               (color ?block ?color)
             )
        )
```

6.2. példa: KQML adminisztratív jellegu üzenet

Az utóbbi példából jól látható, hogy ebben az esetben az üzenet tartalma egy újabb üzenet.

Az adatcsere szintjén az üzenetek olyan adatokkal egészülnek ki, melyek a kommunikációt, azaz a csomagok célba érését segítik. Tipikus adatok a címzett, feladó, azonosító, a kommunikáció típusa, stb. Az alábbi példa bemutat egy teljes KQML csomagot.

```
(KQML-MESSAGE
        SOURCE           CAMERA-AGENT
        DESTINATION           AGENT-FACILITATOR
        ID               CA-1293312Y
        COMMUNICATION    NON-BLOCKING
        (ADVERTISE
             DIRECTION       EXPORT
             (TELL
                  CONTENT-LANGUAGE      KIF
                  CONTENT-ONTOLOGY      block-world
                  CONTENT-DESCRIPTION   physical-property
                  CONTENT               (color ?block ?color)
                  )
        )
```

6.3. pédla: egy teljes KQML csomag

Üzenettípusok a KQML nyelvben

Az alábbiakban egy rövid áttekintést szeretnénk nyújtani a nyelv alapveto üzenettípusairól, az azokban használt performatívákról. A nyelv mintegy kéttucat kulcsszót használ az üzenetek meghatározására, melyek alapvetoen hét kategóriába rendezhetok. A performatívák és jelentésük teljes felsorolása a mellékletben található.

Alapveto lekérdezés *- evaluate, ask-if, ask-oke, ask-in, ask-all*

A kérdezo egy kérdés kiértékelését kéri egy másik ágenstol, melyre egy választ vár

Többválaszos lekérdezés *- stream-in, stream-all*

A forrás ágens a céltol azt kéri, hogy a kérdésre minden esetben válaszoljon, amikor az ahhoz kapcsolódó tudása megváltozik. Ez az üzenet tehát a jövoben több választ is eredményezhet, attól függoen, hogy a válaszadó milyen információkat szerez a kérdessel kapcsolatban.

Válaszok *- reply, sorry*

Az egy vagy többválaszos kérdésekre érkezo válasz, illetve annak jelzése, hogy a válaszadónak nincs adat a birtokában.

Általános közlés *- tell, achieve, cancel, untell, unachieve*

A tell egy állítást közöl, az achieve egy kívánságot, mely állítás teljesülését a küldo szeretné elérni. Mindkettonek létezik a visszavonása is, illetve van egy külön kulcsszó egy azonosítóval meghatározott korábbi üzenet törlésére.

Generátor típusú üzenetek *- standby, ready, next, rest, discard, generator*

Ezen üzenetek mindegyike további üzenetek küldését kezdeményezi.

Képesség definíciók *- advertise, subscribe, monitor*

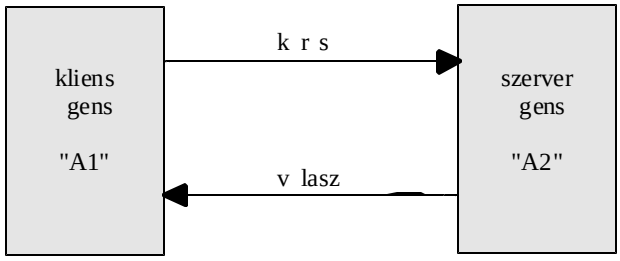
A képességeket definiáló performatívák adminisztratív jellegu üzenetekben az ágensek felajánlhatják szolgáltatásaikat, illetve igényelhetik mások szolgáltatását.

Hálózattal kapcsolatosak *- register, unregister, forward, broadcast, route*

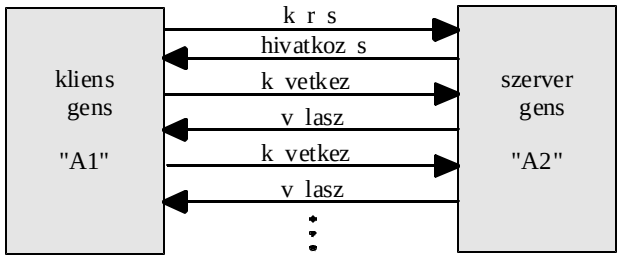
Ezen kulcsszavak írják le egyrészt új ágensek bejelentkezését, régiek kilépését a rendszerbol, másrészt az ágensek közötti csomagok továbbküldését, szétküldését, illetve útjának meghatározását.

Üzenetprotokollok a KQML nyelvben

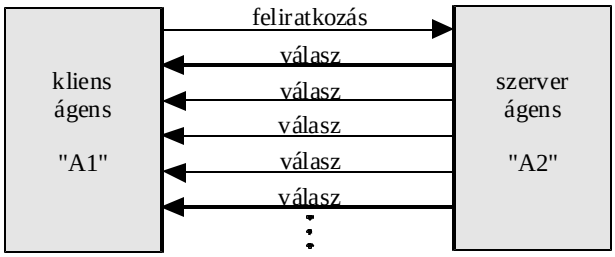
A performatívák típusától függoen többféle párbeszéd is kialakulhat az ágensek között. Egy egyszeru kérdés-válasz pár egy-egy üzenet elküldését jelenti mindkét irányban. Léteznek szinkronizált párbeszédek, amikor egy kérdésre sorozatban érkezo válaszokat a kérdezo egymás utáni üzeneteivel kér, illetve aszinkron jellegu kommunikációk, amikor egy kérdésre érkezo válaszok a válaszadó szándéka szerint bármikor elérhetik a kérdezot. A következo ábrák (7.2, 7.3, 7.4) e három párbeszéd módot mutatják be.



6.2 ábra: KQML egyszerú kérés -válasz párbeszéd két ágens között



6.3. ábra: kérés és ismételt kérésekre adott válaszok típusú párbeszéd



6.4. ábra: feliratkozás és sorozatosan adott válaszok típusú párbeszéd két ágens között

```
(ADVERTISE
        LANGUAGE        KQML
        ONTOLOGY        K10
        (SUBSCRIBE
             LANGUAGE   KQML
             ONTOLOGY   K10
             (STREAM-ABOUT
                  CONTENT-LANGUAGE      KIF
                  CONTENT-ONTOLOGY      block-world
                  CONTENT               block1
                  )
             )
        )
```

6.4. példa: egy ágens (A) egy szolgáltatásának hírdetménye

```
(SUBSCRIBE
        REPLY-WITH      subs1
        LANGUAGE        KQML
        ONTOLOGY        K10
        (STREAM-ABOUT
             CONTENT-LANGUAGE      KIF
             CONTENT-ONTOLOGY      block-world
             CONTENT               block1
```

6.5. példa: egy másik ágens (B) feliratkozik a szolgáltatásra A-nál

```
(TELL
        CONTENT-LANGUAGE      KIF
        CONTENT-ONTOLOGY      block-world
        IN-REPLY-TO           subs1
        CONTENT         (= (color block1) blue)

(TELL
        CONTENT-LANGUAGE      KIF
        CONTENT-ONTOLOGY      block-world
        IN-REPLY-TO           subs1
        CONTENT         (on-top block1 block2)
```

6.6. példa: A üzeneteket küld B-nek a block1 kockával kapcsolatban

A KQML architektúrája

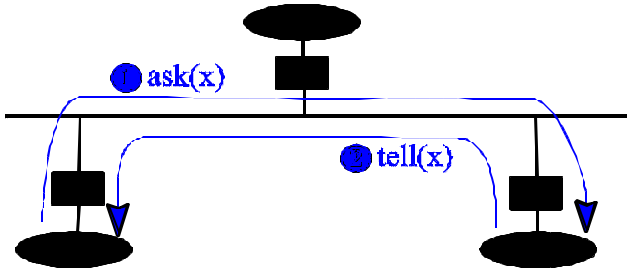
A rendszer a kommunikáció megvalósítására két új komponenst vezet be: az adatcserét közvetlenül megvalósító **KQML útvonal választót (KQML router)** és egy magasabb funkciókat ellátó **kommunikációt segíto ágenst (facilitator)**. Az útvonal választó minden egyes ágens mellett megtalálható, az üzenetek tartalmuktól független továbbítását végzi, illetve adminisztrálja az ágens rendszerbe lépését, illetve távozását. Minden magasabb szintu feladatot a kommunikációt segíto ágens lát el. Az alábbi ábra szemlélteti a rendszer általános architektúráját.

A kommunikációt segíto ágens feladatai

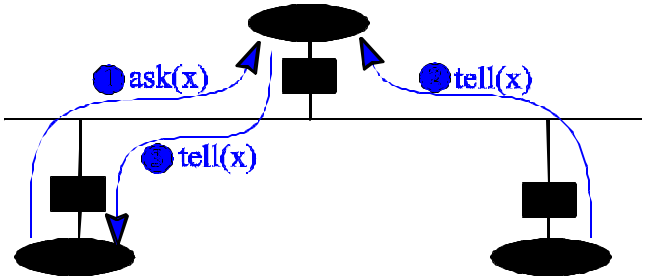
?? A routerek hozzájuk fordulnak a nem közvetítheto, azaz pl. nem teljesen címzett csomagokkal.
?? Az üzenetek tartalma alapján képesek a pontos címzés eloállítására.

?? Nyilvántartják a rendszerben levo ágensek neveit és címét, ez alapján elvégzik a név-cím transzformációt.
?? Ágensek közötti tartalom fordítást végeznek.
?? Az ágensek által közzé tett hírdetményeket tárolják, illetve felhasználják a kérdezo-válaszadó párok kialakításában.

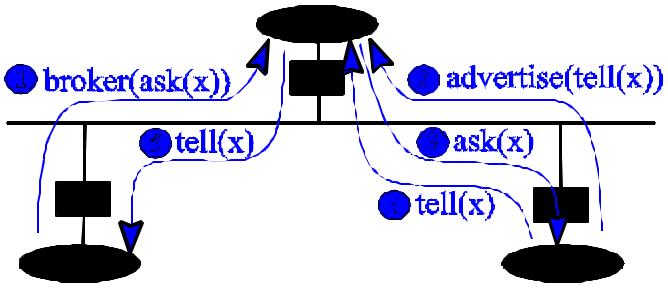
A következo ábrák tipikus kommunikációs utakat mutatnak be az ágensek, útvonal választók és segíto ágemsek között.



6.5. ábra: A és B ágens kérdés-válasz párbeszéde



6.6. ábra: A szolgáltatót keres F-nél egy tény monitorozására, melyre B alkalmas



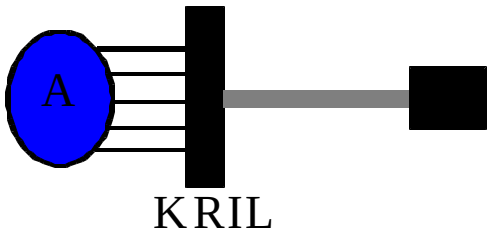
6.7. ábra: A válaszadót keres F-nél egy kérdésre, F ágens B megkérdezése után B-t ajánlja

KRIL: ágensek integrálása a KQML rendszerbe

Az ágensek az útvonal választók segítségével integrálhatóak a KQML kommunikációs rendszerébe, melyek az ágenssel azonos gépen futó folyamatok. Az ágens és router folyamatok között a **KRIL interfész (KQML Router Interface Library)** teremt kapcsolatot. Az ágens programok a KRIL API-t használva egyszeru függvényhívások segítségével érik el a rendszer szolgáltatásait. A következo példa a két legfontosabb API hívást illusztrálja, illetve az ábrák a rendszer felépítését mutatják be.

```
send-kqml-message(char *msg, int msglen);
declare-kqml-message-handler(void (*hndlr)(char *msg, int len));
```

6.7. példa: üzenet küldése, illetve az üzeneteket fogadó függvény deklarálása a KRIL API-ban



6.8. ábra: az ágens, a KRIL interfész és a KQML router kapcsolódása

Biztonsági kérdések a KQML rendszerben

Egy számítógép hálózatban muködo rendszernek fontos biztonsági követelményeknek kell megfelelnie, különösen ha az a Internet protokoll családját használja, Ezen követelmények egyrészt a kapcsolatban részt vevo felek egyértelmu azonosítását, másrészt a közöttük bonyolódó forgalom integritását és titkosságát biztosítják. A következokben röviden összefoglaljuk az ilyen rendszerekkel szemben támasztott követelményeket.

Autentikáció

A rendszernek megoldást kell adnia a kommunikációban részt vevo felek egyértelmu azonosítására (autentikációjára). Ennek megfeleloen a feleket egyedi azonosítóval kell ellátnia, melyet védenie kell az illetéktelen hozzáférésektol, illetve ki kell dolgoznia egy módszert az azonosítók bemutatására.

Integritás

A kommunikáló felek közötti üzenetek tartalmi egységének (integritásának) megorzése szintén alapveto feladat. Meg kell akadályozni, hogy egy harmadik fél az üzenetek tartalmát bármilyen módon befolyásolhassa a kommunikáció során. Ez olyan módszer kidolgozását igényli, mely az üzenetekhez olyan kontrolszumma jellegu adatot csatol, amely egyértelmuen jelzi az eredeti tartalom változását.

Adatvédelem

Lehetoséget kell teremteni a bizalmas adatok védelmére is, azaz arra, hogy a kommunikáló felek között küldött üzenetek tartalmához külso megfigyelok ne jussanak hozzá. Az üzeneteket megfeleloen eros titkosítási algoritmussal kódólni kell, kódolt formában kell a célállomáshoz eljuttatni, majd ott dekódolással vissza kell állítani az eredeti tartalmat.

Üzenetduplázás és –késleltetés

A biztonsági rendszerrel szembeni követelmény az is, hogy detektálja korábbi üzenetek ismételt megérkezését, illetve elküldött üzenetek harmadik fél általi késleltetését. Ez az üzenetekhez csatol egyedi azonosítóval küldési és lejárati dátummal oldható meg, melyeket az üzenethez hasonlóan védeni kell a módosításokkal szemben.

Aláírás

A rendszernek lehetoséget kell arra is biztosítania, hogy egy elküldött üzenetet a küldo ne tagadhasson le. Ehhez olyan aláírással kell ellátnia az üzenetet, mely egyértelmuen azonosítja ot, illetve garantálja, hogy nem harmadik fél helyezte el azt az üzeneten.

Mindezen feladatok megoldásával a **kriptográfia** foglalkozik. A fenti feladatok megoldására leginkább alkalmazott módszerek egyik alap összetevoje az un. **nyilvános kulcsú titkosítás** (public key encryption, Diffie & Hellman), mely két (egy titkos és egy publikus) kódolási kulcs használatával oldja meg a feladatokat. A két kulcs egy felcserélheto kódoló/dekódoló párt alkot, mellyel egyrészt megoldható az üzenetek titkosítása, másrészt egyértelmu azonosítása is.

Erre épülo teljes biztonsági rendszer például a **Kerberos** (MIT Project Athena, 1988), mely fizikailag nem védett számítógépes hálózatok azonosítási és titkosítási rendszere. A rendszerben egy központi komponens, az ún. kulcsosztó gondoskodik a publikus kulcsok tárolásáról. Egy fél publikus kulcsa használható fel egy neki küldött üzenet titkosítására, míg a titkos kulcs a tole érkezo üzeneteket azonosítja. Két fél között titkos és azonosított kommunikáció tehát kettos kódolással, a küldo titkos és a fogadó publikus kulcsával kódolva oldható meg, melyet csak a fogadó tud dekódolni a titkos kulcsával, illetve a küldo nyilvános kulcsával ellenorizheti a feladó személyét. A titkos kulcsok védelmében azonban ezt a módszert általában csak egy üzenetváltás, párbeszéd elején alkalmazzák, amikor egy autentikált vonalat építenek ki egy új kulcs, a **kapcsolati kulcs (session key)** kicserélésvel, melyet azután a késobbi párbeszéd során használnak fel az üzenetek kódolására és dekódolására.

A rendszer emellett megoldást nyújt a fentebb említett egyéb feladatokra is, mint az üzenetduplázás és –késleltetés detektálása, egyértelmu aláírásra, illetve egyéb támadási formák kivédésére is.

A KQML környezetben alkalmazott biztonsági rendszernek a fenti követelmények mellett speciális igényeknek is meg kell felelnie. Ezek inkább implementációs megkötések, melyek azonban kihatással vannak az alkalmazható kriptográfiai megoldásokra is. A titkosítási módszerek nem zavarhatják az üzenetek tartalmát, legyenek függetlenek az alkalmazott szállítási rétegtol, ne használjanak globális órát (mivel a KQML rendszernek ez nem része, és elosztott rendszerekben az implementálása komoly problémákat vethet fel), az ágenseknek ne kelljen kriptográfiai programkönyvtárakat használniuk (azaz a meglévo ágensek minél egyszerubben integrálhatóak legyenek a rendszerbe), valamint támogasson többféle kriptográfiai rendszert.

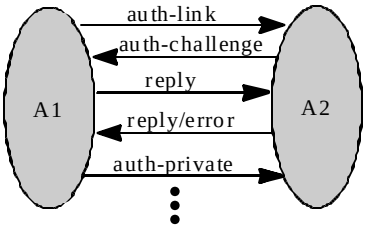
A KQML biztonsági architektúrája

A rendszer kialakítását alapvetoen az ágens megközelítésmód használatával oldották meg. A korábban bevezetett segíto ágensen kívül újabb rendszerkomponens jelent meg: az autentikátor ágens. Ennek feladata lényegében megegyezik a nyilvános kulcsú titkosítási rendszerekben alkalmazott kulcsosztóval: az ágensek publikus kulcsait tárolja. Emellett új nyelvi elemeket, performatívákat is bevezettek, melyek a titkosított üzenetek küldését jelzik.

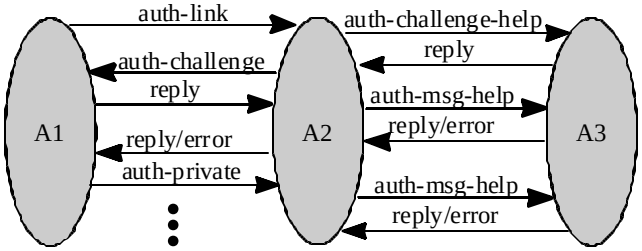
Titkos üzenettípusok

?? *auth-link* azonosítás, és egy kapcsolati kulcs eloállítása
?? *auth-challenge* válasz az elobbi üzenetre, a másik fel autentikációjának ellenorzése
?? *auth-private* a kapcsolati kulccsal kódolt üzenet
?? *auth-*-help* többféle üzenet, amelyben egy titkosításra képtelen ágens kér segítséget mástól

A következokben két példát mutatunk két titkosításra képes ágens titkos párbeszédére, illetve egy titkosításra képtelen ágens közvetíto titkosítón keresztül folytatott beszélgetésére.



6.9. ábra: autentikált kapcsolat kiépítése két ágens között



6.10. ábra: autentikált kapcsolat kiépítése titkosításra nem képes ágens esetén

Problémák a KQML rendszerrel

A rendszer alapfeladata, hogy biztosítsa a kommunikációt a koordináció és a koherencia megteremtésére. Mindezt a szólás aktusok elmélete alapján, ún. performatívák bevezetésével oldották meg. Ezen kulcsszavak használhatóak az ágensek közötti párbeszéd (kérdés, felelet) során, illetve rendszer szintu információk közlésére (belépés, kilépés, képesség definíciók, hirdetmények, problémamegoldó keresése, stb). A performatívák egymásba ágyazhatóak, így komplex üzenetek is kialakíthatóak.

A rendszerrel szemben felmerülo legfontosabb kritika a szemantika aluldefiniáltsága, azaz az, hogy a performatívák definíciói kétértelmuek, hiányosak, rosszak. Alapvetoen a következo három problémakört említik a rendszer hiányosságai között:

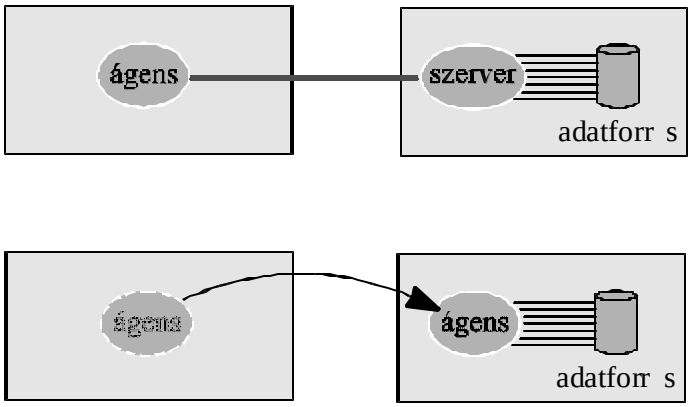
?? *Kétértelmuség.* Több kulcsszó értelmezése nem pontosan definiált, nem egyértelmu. A szintaktikailag pontosan definiált struktúrák szemantikája nem mindig egyértelmu, implementációtól függhet.

?? *Félreértelmezett performatívák.* Néhány performatívaként megadott kulcsszó valójában nem performatíva, mivel nem eredményez közvetlen akciót, csak egy felkérést jelent, amit azonban a fogadó figyelmen kívül hagyhat.

?? *Hiányzó performatívák.* Az alap KQML nyelvbol teljesen hiányoznak a megbízás típusú performatívák, amelyekre a fogadónak egy rövid válasszal mindenképpen reagálnia kell (elfogadon, nem fogadom el a megbízást). Az ilyen jellegu azonnali visszajelzések hiánya nagyban ronthatja egy kooperáló rendszer hatékonyságát.

7. Mobil ágensek

Az eddig ismertetett ágensek helyhez (processzorhoz) kötött szoftverek voltak, azaz azon a számítógépen futottak végig, amelyiken elindultak. Ha más helyen lévo adatokra volt szükségük, akkor a kommunikáció segítségével más ágensektol, programoktól szerezték be azokat. A **mobil ágens** futása közben nincs helyhez kötve, magát – saját akaratából – áthelyezheti egy másik gépre, és ott futhat tovább. Íly módon az adatok forrásával nem hálózaton keresztül kommunikál, hanem közvetlenül azon a gépen, ahol az megtalálható.



7.1. ábra: helyhez kötött és mobil ágens kommunikációja egy adatforrással

A mobil szoftverek ötlete természetesen nem korlátozódik az ágensek területére. Egy egyre erosödo terület, a **mobil számítástechnika (mobile computing)** foglalkozik általánosságban a kérdéskörrel, egyelore azonban meglehetosen szerteágazó módon, kevés letisztult fogalmat és módszert definiálva. Mindez magának az alkalmazási környezetnek köszönheto: az Internet technológiák rohamos fejlodése, a szerteágazó kutatási területeken párhuzamosan dolgozó csoportok, cégek különbözo technológiai kezdeményei (több esetben csak igéretei) meglehetosen kusza helyzetet alakítottak ki. A széles köru érdeklodés ellenére meglehetosen kevés mobil kódot használó rendszer épült ki, s ez meghatározza a terület elfogadottságát is. A mobil számítástechnika talán legnagyobb olyan részterülete, ahol már több sikeres alkalmazás is készült, a mobil ágenseken alapuló rendszerek.

A mobil programok elonyei és problémái

A mobil programok koncepciója több területen is igéretesnek bizonyulhat. A mozgó kód kiküszöbölheti a hálózat okozta hátrányokat (megbizhatatlan, lassú, stb) a feldolgozó algoritmusok adatforráshoz telepítésével. Sokkal jobban támogathatja elosztott rendszerek kialakítását, mivel csak egy általános fogadó állomást kell telepíteni a rendszerben részt vevo számítógépekre, ahol azután a szoftver komponensek az adott alkalmazásnak megfeleloen, telepítés nélkül jelennek meg, ráadásul a fogadóállomások terheltségétol függoen alakíthatják ki a feladat megoldásában közremuködok körét. Mindez a rendszert sokkal jobban skálázhatóvá teszi, hiszen újabb és újabb állomások kapcsolhatók be egy probléma megoldásába annak eroforrás igényétol és fontosságától függoen.

A mobil számítástechnika a kliens-szerver modell helyett egy új filozófiát alakíthat ki, a csomópontokból álló eroforrás hálózatot, ahol nincs kitüntetett szerver és kliens alkalmazás, a feladatokat bármely csomópontban meg lehet oldani. Ez bizonyos alkalmazási körökben sokkal könnyebben megértheto, mint az eddigi elosztott számítási paradigma, így tisztább, egyszerubb megoldások alakíthatóak ki.

A mobilitás legnagyobb problémája a biztonság kérdésköre. Számítógépes vírusok számára egy ilyen infrastruktúra megjelenése nyilvánvalóan rengeteg új támadási lehetoséget kínál. Általános hálózati környezetben jelenleg szinte kizárólag a felhasználó kezdeményezésére, tudtával és engedélyével érkeznek és indulnak el programok a felhasználó saját gépén. A legjelentosebb kivétel a Web-es környezet Java és Javascript programjai, melyek azonban szigorú biztonsági rendszerekkel körbebástyázott fogadóállomásokra érkeznek. (Ennek ellenére a Javascript esetében több, nagyon súlyos biztonsági lyukat

találtak már.) Az automatikus (felhasználótól független) kódmobilitást lehetové tevo infrastruktúrának, különösen ha az általános célú rendszer kíván lenni, komoly biztonsági kihívásokkal kell szembenéznie.

A mobil kódok másik legnagyobb problémája a kódvesztés, amikor egy mozg ó program az egyik géprol egy másikra költözik, és a hálózati átvitel során megsérül, elveszik. Olyan mechanizmusokat kell kidolgozni, melyek garantálják egy átköltözo program megérkezését és elindulását a fogadóállomáson.

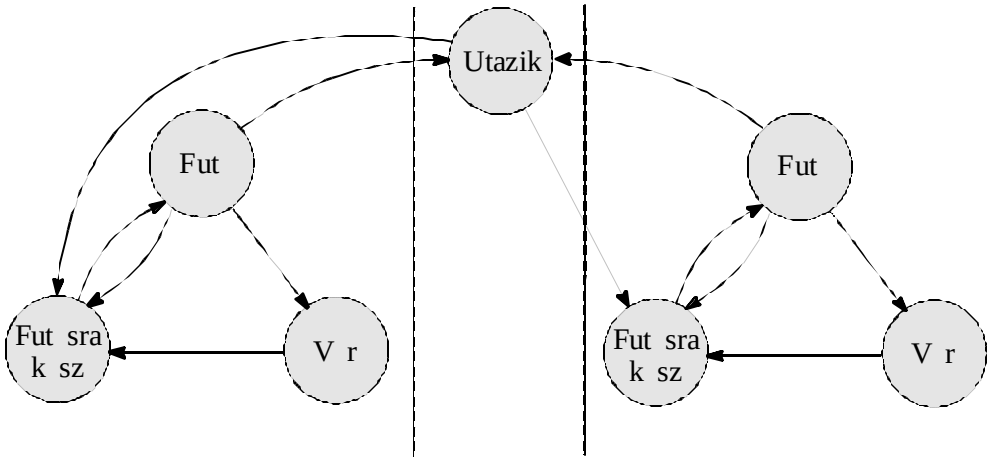
A mobil kódfejlesztés komoly problémája a programok tesztelése, belövése. Kü lönösen a programok futási környezet változtatása során bekövetkezo lépések nehezek követhetok. Nehéz a feladatra valós futtató környezetet (nyomkövetot) készíteni - a rendszert elsosorban szimulációs úton lehet e llenorizni.

A mobil ágensek alapfogalmai

A fejezet bevezetésében említettek miatt nincs kialakult fogalomrendszer a területen. Az alábbiakban azokat a legfontosabb területeket tekintjük át, melyek közösek a jelenlegi rendszerekben.

A program állapota

A hagyományos folyamatok ismert alapállapotai (fut, vár, futásra kész) mellett egy új állapot, az **utazás (travel)** jelenik meg, mely azt fejezi ki, amikor egy program A és B gép között átmozog. Nem kizárólag a programkód átvitelét írja le ez az állapot, hanem minden, az átvitellel kapcsolatos futtatórendszer feladatot. A program szempontjából az új állapot egy olyan speciális várakozó állapotként jelenik meg, amikor az alatta elhelyezkedo futtató rendszer a program átvitelét valósítja meg.



7.2. ábra: új folyamat állapot: utazás

Helyszín

A mobilitás tárgyalásához szükséges a **helyszín (place)** fogalmának bevezetése, mely képes mobil kódok fogadására. A helyszín az alapveto mechanizmusok (kód fogadása, tovább futtatása, felfüggesztése és továbbküldése) mellett elsosorban biztonsági feladatokat lát el: meghatározza és korlátozza a futó kód hozzáférését a helyszínt körbevevo számítógépes környezethez. Egy ágens **helye** annak a helyszínnek a címe, melyen az ágens éppen tartózkodik (fut).

Fennhatóság

A biztonsággal kapcsolatos egyik legfontosabb fogalom a **fennhatóság (authority)**. A mobil programok egyik legfontosabb jellemzoje, hogy személyi azonosítóval rendelkeznek, azaz egyértelmuen és biztonságosan meg tudják adni azt a személyt vagy szervezetet, melynek a nevében eljámak. Ha rendelkeznek a megfelelo meghatalmazással egy adott helyszín használatához, akkor beléphetnek a helyszín fennhatósága alá, és használhatján annak eroforrásait. A különbözo fennhatósági területeket valamilyen logikai egységbe foglaló fogalom a **régió (region)**. A helyek, helyszínek, fennhatóságok és régiók adminisztrálására a mobil kódok navigációját segíto eszközöket is be kell vezetni (címtár, szolgáltatások és eroforrások leírása, stb).

Programátvitel

A kódmobilitás kulcsa a futó program bájtsorrá alakítása, amely már átviheto a számítógépes hálózaton. Ez az ún. **sorosítás (serialization)**. Ennek során nyilvánvalóan nem csak a program kódját, hanem a futás közbeni teljes adat és kódszegmenst, valamint a futás aktuális állapotát (pogramszámlálót, vermet) is át kell akalíani bájtsorrá. A soros átvitel után a programot vissza kell akalítani, mely feladat magában foglalja az eltéro architektúrára ültetést is (eltéro bájtsorrend a számábrázolásban, adatszerkezetek változása, stb).

Programozási nyelv és környezet

Az egyik legfontosabb követelmény mobil programok nyelvével, fejlesztoi környezetével kapcsolatban a platformfüggetlenség, mely a kódok könnyu áthelyezhetoségét támogatja. Ilyen szempontból a értelmezett (interpreted) nyelvek elonyt élveznek a fordított (compiled) nyelvekkel szemben. Ilyenek pl. a perl, a Tcl, a LISP (Scheme, CLIPS) vagy a python, melyeknek különbözo operációs rendszer és hardver platformon is létezik futtató környezete.

A programozási nyelvekkel és környezetekkel kapcsolatos másik fontos követelmény a megfelelo biztonsági elemek. Ezek egyrészt jelentenek nyelvi korlátozásokat, védelmeket, másrészt a futtató környezetben megvalósítható elemeket. Értelmezett nyelvek esetén a legfontosabb kérdés az, hogy a futtató rendszer (értelmezo) mennyire tudja korlátozni a futó program muködését. Fordított programok esetén a fordítóprogram és a futtató környezet is ellát biztonsági feladatokat.

A harmadik követelmény a sorosítás egyszeru megoldhatósága, azaz az, hogy a futó kód (saját kezdeményezésére) lehetoleg könnyen átviheto legyen a hálózaton, majd a másik gépen az átvitelt közvetlenül megelozo állapotból futhasson tovább.

A Sun által fejlesztett Java objektum-orientált nyelv mindhárom területen sok pozitivummal rendelkezik. A nyelv részben értelmezett és fordított (a fordítóprogram egy közbülso bájtkódot állít elo, melyet a futtató környezet értelmez). Másrészt a nyelven készített programokat Web környezetbe integráló **programka (applet)** koncepció a hálózati programátvitelt és a biztonsági feladatokat megoldó komponenseket is bevezet. A nyelv kidolgozása során fontos szempont volt biztonsági elemek beépítése (pl. teljesen hiányzik a mutató típus), valamint a programkákat futtató Java Virtuális Gép (JVM: Java Virtual Machine) a programok elindítása elott biztonsági ellenorzést végez, a futás közben pedig korlátozza a program muködését. A nyelvben található eszköz a futó programkód átvitelére is, az ún. **objektum sorosítás (object serialization)**, mely objektumokat készít fel (alakít át) hálózati átvitelre.

Mobil ágens rendszerek

Az alábbiakban néhány jellemzo példát mutatunk be mobil ágens rendszerekre. A példák részletes ismertetése túlno e jegyzet keretein, itt csak a lényegi tulajdonságaikat foglaljuk össze. A téma iránt bovebben érdeklodok a felsorolt internet címeken találhatnak bovebb információkat. Az ismertetett rendszerek általában szabadon hozzáférhetoek és használhatóak.

Telescript, Odyssey

Az egyik elso mobil ágens rendszer a General Magic (http://www.genmagic.com/) Telescript rendszere. A Telescript egy objektum-orientált nyelv, nagy méretu elosztott alkalmazások készítésére hozták létre. A tervezés során a két legfontosabb szempont az eros mobilitás támogatás és a biztonsági követelmények figyelembe vétele volt. Ebben a környezetben definiálták az elsok között a korábban ismertetett fogalmakat (helyszín, fennhatóság, régió, stb). Az ágensek a `go` utasítás segítségével változtathatják futási környezetüket a helyszínek között, illetve a `send` utasítás segítségével hozhatják létre önmaguk mását egy másik helyszínen. A mobil ágens rendszerek közül a Telescript nyújtja a legátfogóbb biztonsági rendszert. Minden objektumhoz rendeltek olyan attributumokat, melyek meghatározzák annak biztonsággal kapcsolatos tulajdonságait. Minden ágens egy valós világbeli ember vagy szervezet névjegyét hordozza magával (authority), rendelkezik engedélyekkel (permits), melyek meghatározzák, hogy milyen akciókat hajthat végre (ilyen kell a `go` utasításhoz is), illetve milyen eroforrásokhoz milyen mértékben fér hozzá az egyes helyszíneken. Az engedélyek átlépése a programtól függetlenül megszakítást eredményez, megszakítva annak további muködését. Az ágensek átvitele a hálózaton keresztül kódolt formában, biztonságosan történik.

Az Odyssey (http://www.genmagic.com/technology/odyssey.html) a Telescript átdolgozása a Java programozási környezetre. Ezzel a Telescript elterjedésének egyik legnagyobb gátját szüntette meg a General Magic: a speciális programozási nyelvet és környezetet.

Agent Tcl, D'Agents

A Dartmouth College (USA, http://www.cs.dartmouth.edu/~agent) által fejlesztett eszköz a Tcl értelmezo kiterjesztése a kód mobilitást támogató funkciókkal. Az ágensek UNIX folyamatokként implementáltak, melyek csak az operációs rendszer által nyújtott szolgáltatások segítségével használhatnak más ágensekkel közös eroforrásokat. Ezen eroforrások jellegüknél fogva géphez kötöttek. Az ágensek háromféle módon léphetnek kapcsolatba más gépen levo eroforrásokkal: átugorhatnak egy másik gépre (`jump`), indíthatnak egy új ágenst egy másik gépek (`fork`), vagy átküldhetnek egy kódot (kódrészletet) egy másik gépre (`submit`). A mobilitás szempontjából legérdekesebb az elso módszer, mely megköveteli az aktuális futási állapot átvitelét is. Ezt a rendszer a teljes tcl interpreter átvitelével oldja meg, beleértve annak kódját és teljes futási állapotát. Mivel a rendszer kizárólag a Tcl nyelvet támogatja, így az ágensek forráskódú állapotban mozognak a gépek között, nincs szükség átfordítási mechanizmusokra. A fejlesztok tervei között szerepel a Scheme és a Java nyelv támogatása is, illetve a jelenleg még eléggé hiányos biztonsági elemek továbbfejlesztése. Ezen irányokat fogja össze a jelenlegi is folyó D'Agents szoftver fejlesztés, melynek publikus verziója egyelore még Tcl-alapú, belso fejlesztési változata azonban már emellett támogatja a Java, Python, és Scheme nyelveket is.

Java Aglets

Az IBM Tokyo Research Laboratory (Japán, http://www.trl.ibm.co.jp/aglets) által fejlesztett API gyujtemény a Java nyelvet egészíti ki kódmobilitást támogató elemekkel. Az agletek (nevüket az appletek mintájára ágens mivoltukra utalva kapták) a Java Virtuális Gépen futó szálak, melyek köré az Aglets API egy kontextust rendel az alapveto mobilitási szolgáltatások biztosítására. A rendszer két migrációs primitívet definiál: a `dispatch` egy azonnali kódátvitelt kezdeményez egy másik kontextusba, míg a `retract` primitív ágens visszahívására szolgál abba a környezetbe, ahol az utasítást kiadták. Az átmozgatott ágens mindkét esetben újból kezdi kódja végrehajtását, azonban az objektum változói megorzik értéküket, mely változó kezdoállapotokat szolgáltat a futásához.

Függelékek

A. Függelék: A hálózati programozás alapjai

A hálózati számítási modell

A számítási módszerekben a gépeke összekapcsoló hálózat megjelenése egy új fejezetet nyitott. A soros vonali terminál illesztés (dialup) továbbfejlodése gépek közötti pont-pont adatkapcsolat kialakítását tette lehetové. Megjelent a kliens-szerver modell, mely szolgáltatás-alapon különítette el a felhasználót (kliens) és a szolgáltatót (szerver). A fizikai hálózati közeg felett kommunikációs protokollok (standardok) publikusak voltak, így a különbözo gyártók adott rendszerkörnyezetben kompatibilis implementációkat készítettek (Ethernet, TCP/IP, stb).

A hálózati protokollok általános szabványa az ISO/OSI hét rétegu hálózati modell.

Az Internet

Az Internet a TCP/IP-re alapuló hálózatok összekapcsolt világhálózata. Alapvetoen a UNIX operációs rendszer mentén alakult ki, de ma már minden, hálózati szolgáltatást nyújtó operációs rendszer támogatja.

?? Címzés: Minden hálózatba kötött gépnek egyedi címe van (IP cím), mely egy 4*8=32 bites szám. A cím elso része kijelöl egy lokális hálózatot, a maradék azon belül egy gépet. A két tartomány méretétol (a határ helyétol) függoen vannak A, B és C osztályú címek. Az alkalmazások gépen belüli címzésére a portszámok szolgálnak.

?? Nevek: A könnyebb kezelhetoség végett a gépekhez név is tartozik, mely egy gépnévbol, és egy domain névbol tevodik össze. A szám és név szerinti címzés között az un. domain név szervíz (DNS: domain name service) tart kapcsolatot.

?? Réteges felépítés: Az Internetes protokollok réteges felépítésuek (layered network modell), bár nem követik az ISO/OSI modellt.

?? Csomagkapcsolt: Az adatok csomagok formájában közlekednek. Réteg - csomag leképezés: Az egyes rétegek a leképezhetoek a csomagok belsejében található belso csomagokra (data encapsulation).

Internet programozás

A gépeken futó alkalmazások közötti adatátvitel programozói interfésze nagyon hasonlatos a fájlkezeléshez (UNIX alatt "minden adatmozgatás fájlkezelésre vezetheto vissza"). A fájlmuveletek hivatkozási eszköze a fájl leíró (file descriptor). Az alapmuveletek a megnyitás, írás, olvasás, bezárás, létrehozás, törlés.

A hálózati csatorna ("fájl") leírója a socket (megjegyzés: nem ez az egyetlen módszer hálózati programozásra, de ez a legelterjedtebb). A socket (és az adatforgalom) alapvetoen két típusú lehet (zárójelben az IP fölötti réteg protokolljának rövidítése olvasható):

?? stream (TCP): megbízható, sorrendhelyes, kétirányú kapcsolat

?? datagram (UDP): kapcsolatmentes csomagküldés, ahol a sorrend és a megérkezés nem garantált.

A socket létrehozása

A hálózati programozás legkényesebb (legnehezebben megértheto) része a socket létrehozásával, illetve konfigurálásával kapcsolatos. Ellentétben a fájlmuveletek egyszeru open() rendszerhívásával, itt több rendszerhívást is használni kell, valamint a kitöltendo adatstruktúrák is bonyolultabbak. A socket a következo rendszerhívással hozható létre:

```
int socket(int domain, int type, int protokol);
```

ahol a domain esetünkben az AF_INET (más domain-ek is léteznek, mivel a socket nem csak hálózati kommunikációra használható), a típus stream vagy datagram (SOCK_STREAM, vagy SOCK_DGRAM), a protokoll praktikusan 0 (akkor más, ha a típuson és domain-en belül többféle protokoll is létezik). A visszatérési érték a socket leíró, vagy hibajelzés. Ez azonban nem elegendo adatok küldéséhez.

A socket kötése

A socket létrehozása csak a névtérben történik meg a socket() rendszerhívással. A socket a következo rendszerhívással kötheto a lokális gépen egy porthoz.

```
int bind(int sockfd, const struct sockaddr *addr, int addrlen);
```

Ezt tipikusan a szerver alkalmazásokban használjuk. Az így kötött portra kapcsolódhatnak a kliens alkalmazások a következo rendszerhívással:

```
int connect(int sockfd, const struct sockaddr *addr, int addrlen);
```

A connect() segítségével a socket egy tényleges adatúthoz kötheto, azaz egy másik gépen futó alkalmazásig (a másik gép adott portjáig) vezeto hálózati kapcsolatot hozunk létre. Ez a hívás egyrészt felépíti az adatutat, másrészt hozzárendel egy helyi portot, amihez köti a socketet. Elso ránézésre nincs különbség az elozo bind()-hez képest, azonban a cím (addr) struktúra kitöltése különbözo a két esetben. A cím kitöltéséhez egy struktúrát használunk:

```
struct sockaddr {
   short int     sin_family;    /* Address family, AF_INET */
   char           sa_data[14];   /* 14 bájtos protokoll cím */
}
```

Ez túl általános, illetve nehezen kezelheto, ezért az AF_INET domain-re készült egy jobb struktúra is:

```
struct sockaddr_in {
   short int          sin_family;    /* AF_INET */
   unsigned short int sin_port;      /* portszám */
   struct in_addr     sin_addr;      /* IP cím */
   unsigned char      sin_zero[8];   /* feltöltés sockaddr
                                        struktúra méreture */
};
```

Ez a struktúra méretében megegyezik az elozovel (fontos a sin_zero feltöltése nullával), ezért használható helyette, csak a függvények hívásakor kell típuskonverziót végezni. FONTOS tudni, hogy a sin_port és a sin_addr hálózati bájt sorrendet követ (network byte order), tehát erre a formára kell hozni. Erre (illetve a visszaalakításra) a következö függvények valók: htons(), htonl(), ntohs(), és ntohl(), ahol az elso betu a forrás (n: network, h: host), a negyedik betu a cél, míg az utolsó a típus (l: long, s: short). A kommunikáció során az adatokat is célszeru ilyen bájtsorrendben küldeni, mivel így kikerülhetok a különbözo architektúrák eltéro ábrázolási sorrendjébol fakadó problémák.

A fenti struktúrában szereplo in_addr kifejtése a következo

```
struct in_addr {
   unsigned long      s_addr;        /* 4 bájtos cím */
}
/* példa */
struct in_addr server_addr = inet_addr("152.66.82.1");
               /* hálózati bájt sorrenddel tér vissza */
```

A név szerinti címek leírására a következo struktúra, illetve kezelésükre a következo függvény használhatóak.

```
/* FONTOS! Minden adat hálózati bájtsorrendben adott */
struct hostent {
   char  *h_name;      /* elsodleges név */
   char **aliases;     /* alternatív nevek */
   int    h_addrtype;  /* esetünkben AF_INET */
   int    h_length;    /* a cím hossza bájtokban */
   char **h_addr_list; /* a gép címei, nullával végzodik */
}
#define h_addr h_addr_list[0]
                       /* a gép elsodleges címe */

/* A címek lekérdezése név alapján: */
struct hostent *gethostbyname(const char *hostname);
```

A socket használata

A szerverek a bind() hívás után bejövo adatokra várakoznak. Mielott ezt megtennék egy rendszerhívással konfigurálják a várakozási sor hosszát.

```
int listen(int sockfd, int backlog);
                       /* backlog: a bejövo sor hossza */

int accept(int sockfd, void *addr, int *addrlen);
```

Az operációs rendszer a portra beérkezo kapcsolatokat egy sorban helyezi el, aminek hosszát határozza meg a backlog paraméter. A szerver az accept() rendszerhívással a sor elején álló kérést fogadja, az addr struktúrában megkapva annak paramétereit. A következo accept() a sorban következo kéréssel fog foglalkozni. Az accept visszatérési értéke egy új socket leíró, mely a klienshez létrejött kapcsolatban használható adatküldésre és fogadásra.

```
int send(int sockfd, const void *msg, int len, int flags);
int recv(int sockfd, void *buf, int len, unsigned int flags);
/* Mint minden fájlleíróra, itt is használható a read() és a
write() */
```

Mindkét rendszerhívás az elküldött, illetve fogadott bájtok számával tér vissza, amely küldésnél kevesebb is lehet, mint az eloírt.

A socket lezárása

A socket a shutdown() és a hagyományos close() rendszerhívásokkal zárható le. Az elobbinek paraméterként megadható, hogy a kétirányú adatforgalomból melyik irányt zárja le.

Tipikus szerver és kliens alkalmazás

Egy tipikus szerver alkalmazás szerkezete UNIX alatt:

```
servsock = socket();
bind(servsock, ...);
listen(servsock, ...);
while (1) {
   newsock=accept(servsock, ...);
   if (fork()==0) { /* a gyerek kiszolgálja a kérést */
      if (send(new_fd, "Hello, world!\n", 14, 0) == -1)
         perror("send");
      close(new_fd);
      exit(0);
   } /* a gyerek vege */
   close(new_fd); /* a szülonek nem kell */
} /* while() */
```

A kliens programjának szerkezete:

```
#define SERVERPORT  1201
srvent = gethostbyname(server_hostname);
sockfd = socket();
srv_addr.sin_family = AF_INET;
srv_addr.sin_port = htons(SERVERPORT);
srv_addr.sin_addr = *((struct in_addr *)srvent->h_addr);
bzero(&(srv_addr.sin_zero), 8);
connect(sockfd, (struct sockaddr *)&srv_addr, \
        sizeof(struct sockaddr))
numbytes=recv(sockfd, buf, MAXDATASIZE, 0)
buf[numbytes] = '\0';
printf("Received: %s",buf);
...
close(sockfd);
```

Megjegyzések

Az itt ismeretett rendszerhívások gyakorlatilag minden operációs rendszer alatt így néznek ki (könnyen írható olyan forráskód, mely UNIX, DOS, Windows, OS/2 alatt is lefordul). Legnagyobb különbség az egyéb rendszerhívásokban van, pl. új folyamatot másképp kell Windows és UNIX alatt índítani (de még a UNIX verziók között is lehet választási lehetoség). Windows, OS/2, illetve újabb UNIX verziók (pl. Solaris 2) esetén ajánlott a szálak (thread) használata, míg régebbi UNIX-ok (pl. SunOS) esetén a processz.

B. Függelék: Web programozás

A Web, mint kliens-szerver architektúrájú dokumentum szolgáltató és lekérdezo rendszer sokféle programozási területtel rendelkezik. Ide kapcsolhatóak egyrészt a hálózati jellegbol fakadó területek (hálózati kommunikáció, elosztott rendszerek, stb), másrészt a dokumentumok eloállításával, manipulálásával kapcsolatos feladatok is (HTML szerkesztés, dinamikus dokumentum generálás, stb).

A Web programozás fobb területei

Web programozásnak sok területet neveznek. A következokben megpróbálok egy teljes képet adni mindarról, ami e témába tartozhat.

Dokumentumok készítése

Ide elsosorban HTML formátumú dokumentumok készítése tartozik, ami ugyan nem programozás, de gyakran a web programozás részeként említik. Részleteire nem térek ki.

Szerver-közeli programozás

A szerver közeli programozás fo feladata Web dokumentumok dinamikus eloállítására alkalmas programok készítése. Ezek egyrészt nem Web dokumentumokat alakíthatnak át, másrészt más dokumentációs rendszerekhez biztosíthatnak interfészt.

Tipikus programozási eszközök: Perl, C/C++, Java (Servlet), JavaScript

CGI programozás

A CGI (Common Gateway Interface) egy általános módszer szerver oldali külso programok futtatására, az elso elterjedt módszer volt interaktív web lapok megalkotására. Alapveto módszere a web lapon elhelyezett speciális HTML hivatkozás egy szerver oldali önálló programra (általában Perl script), melyet a web szerver indít el, bemetet kaphat a klienstol, majd a szabványos formátumú válaszát a szerver értelmezi, és tartalmát visszaküldi a kliensnek. Tipikusan adatbázis-elérésre, formanyomtatványok feldolgozására, web-es levél, üzenet küldésére használják. Alapveto problémája, hogy a HTTP állapotmentes protokoll, így nehéz igazi párbeszédes programot írni, valamint kevéssé biztonságos, mivel a kliens közvetlenül paraméterezheti a szerveren futó programot. Ezen kívül a program indítása minden kérésre külön történik, így minden esetben lassú.

Szerver modulok

A legtöbb web szerver biztosít egy programozói felületet a szerverbe töltheto modulok írása számára, melyek pl. dinamikusan tölthetok be kliens oldali kérések hatására a szerver futó folyamatába. Tipikus fejleszto eszköz a C/C++ nyelv. Tipikusan új tipusú kliens kérések kiszolgálására használható.

Ellentétben a CGI-vel itt csak az elso kérés (melynek hatására a szerver betölti az igényelt modult) kiszolgálása lassú, a modul ezek után a memóriában marad.

Szervlet

A szerver oldali Java applet (servlet) olyan speciális modul, mely a szerver Java futtató gépe (JVM: Java Virtual Machine) segítségével fut. Elérik az elozo pontban említett szerver API-t, illetve ezen kívül a Java nyújtotta API lehetoségeit is.

Kliens-közeli programozás

A kliens oldali programozás egyrészt olyan speciális dokumentumok készítését jelenti, melyek programok, és a kliens böngészoje fogja oket végrehajtani, másrészt - hasonlóan a szerver oldali modulokhoz - a böngészo kiegészítését jelentik újabb típusú dokumentumok megjelenítéséhez.

HTML dokumentumok kiegészítése "scripting" funkciókkal

A terület célja dokumentumok, vagy részletek dinamikus eloállítotása, valamint a böngészo kezelése a dokumentumokból. A script egy speciális HTML kulcs (<SCRIPT type=... language=... src=...>...</SCRIPT>) segítségével illesztheto a dokumentumokhoz. Tipikus programozói eszközök: Javascript, VBScript. A Javascript a Netscape által kifejlesztett C++ alapokra építkezo nyelve, míg a

VBScript a Microsoft Visual Basic alapú eszköze. Mindketto felhasználható mind kliens, mind szerver oldali programok készítésére is.

Java applet

A Java applet egy olyan eszköz, mellyel a web dokumentumokban a hagyományos ablakozó rendszerekben megismert programokhoz hasonló interaktivitást érhetünk el. A Java a Sun Microsystems C++ alapú általános célú programnyelve, mely nemcsak ebben a környezetben használható. Alapveto cél egy platformfüggetlen fejlesztési környezet létrehozása volt, mely jól illeszkedik a web (illetve a hálózat) filozófiájába, sok olyan szolgáltatást nyújt, mellyel ezeken a területeken leegyszerusödik a programkészítés. A nyelv fobb jellemzoi: C++ jellegu szintaktika, platformfüggetlen aritmetikai típusok, szigorú kivételkezelés, egyszeres öröklodés, szálak (thread), biztonságos (pl. nincs mutató típus), automatikus szemétgyujtés (garbage collection), stb. A forráskódú programokat a Java fordító bájtkódra fordítja, melyet a Java virtuális gép (JVM: Java Virtual Machine) értelmez és hajt végre. Az applet olyan speciális Java bájtkód, melyet a böngészon belül futó JVM hajt végre, és további megkötések érvényesek rá (gyakorlatilag nem férhet hozzá a futtató gép egyetlen eroforrásához sem, csak amit a böngészo biztosít, illetve hálózati kapcsolatot csak ahhoz a géphez kezdeményezhet, ahonnan a böngészo letöltötte). Hasonlóan a <SCRIPT> kulcshoz, ebben az esetben is egy speciális HTML kulcs szolgál az applet beillesztésére: az <APPLET>.

Böngészo modulok (plug-in)

A böngészo moduláris kiterjesztésének elsodleges célja üj típusü dokumentumok megjenítése (pl. VRML, MPEG, stb). Alapveto fejleszto eszközök a C/C++, és a különbözo böngészokhöz kiadott "plugin SDK" (software development kit).

Web integáció

E területbe tartoznak azon eszközök, melyek kliens vagy szerver oldalon elosegítik nem web-es alkalmazások beillesztését a web dokumentációs rendszerébe. Elsodleges feladat az adatcsere megvalósítása, valamint a szoftver komponensek funkcionalitásának elérése (vezérlés). Az eddigi eszközök is nyújtanak erre megoldást, ide inkább az egységes, szabványosított (vagy az alatt álló) megoldásokat soroltam.

Adatbázis kapcsolat (ODBC)

Az ODBC (Open Data Base Connectivity) interfész célja egy szabványos adatbázis API nyújtása, mely mind web szerverbol, mind kliens oldali programokból (appletbol) elérheto. Az interfész a mai adatbázis kezelok része.

Elosztott Java

RMI..., JDBC..., ...

ActiveX technológia (Microsoft)

Az OLE/COM technológia alkalmazható Web-es területeken is, a VBScript és Javascript nyelvekbe integrált rendszerhívásokon keresztül. A DCOM elosztott komponens modell mindezen eszközök kiterjesztése egy elosztott, objektum-orientált rendszerré.

Corba (OMG)

A Corba (Common Object Request Broker Architecture) azt specifikálja, hogy egy hálózati környezetben elosztott objektumok hogyan működhetnek együtt, függetlenül az aktuális kliens/szerver felépítéstol, operációs rendszerektol és programozási nyelvektol. Az OMG (Object Management Group) egy konzorcium, mely e technológiát felügyeli. A Corba gyakorlatilag a Microsoft DCOM vetélytársa. Az alapfeladat objektumok összekapcsolása oly módon, hogy az objektum számára az összekapcsolás csak egy módszerhívásként (függvényhívásként) jelenjen meg, mindenfajta adminisztrációs, hálózati, stb. feladatot a Corba rendszer oldjon meg. Erre a célra vezették be az Object Request Brokert (ORB), mely az objektumok között adatút kiépítéséért felelos. A rendszer egy programozási nyelvtol független módszert biztosít az objektumok interfészeinek leírására (mely alapján a kliens kiépítheti a kapcsolatot a szerverrel), ez az interfész leíró nyelv, avagy IDL (interface definition language). Egy objektum IDL leírásából az IDL fordító készít egyrészt az objektumot, mint szervert használni kívánó kliensekhez egy un. csonkot, mely a kliens számára elérhetové teszi a szervert; valamint a szerver számára egy csontvázat, amely a szervert

beilleszti a Corba rendszerbe. A Corba architektúra részei még a szervizek (CORBAservices) és a szolgáltatások (CORBAfacilities). A szervizek segítséget nyújtanak az objektumok életútjához, és az objektumok közötti kommunikációhoz. Tipikus szervizek: név szervíz, esemény szerviíz, tranzakció szervíz, security, óra, licensz, stb. A szolgáltatások alapvetoen kétfélék lehetnek: horizontálisak (horizontal common facilities), és vertikálisak (vertical common facilities). Az elobbiek az alkalmazások többsége által igényelt, nem alacsony szintu szolgáltatások, mint pl. az elosztott dokumentum kezelés, információ menedzsment, rendszer menedzsment, taszk menedzsment, stb. Az utóbbiak egy-egy alkalmazási területen nyújtanak további szolgáltatásokat, mint pl. egészségügy, stb.