Modellezés és metamodellezés

Hibatűrő Rendszerek Kutatócsoport

2019

Tartalomjegyzék

Bevezetés

Ebben a fejezetben a modellezés alapfogalmaival fogunk megismerkedni. Az itt bevezetett fogalmak újra és újra megjelennek majd a későbbi fejezetekben, ahol részletesen ki fogunk térni az adott területen történő értelmezésükre.

1. Modellezés

Mi értelme van modellezni? Fejben szinte mindig modellezünk, bármilyen probléma kerül elénk. Nincs ez másképp egy szoftver fejlesztésekor sem. Lássuk hát, miről beszélünk, amikor a modell szót használjuk.

Definíció. *Modell*: egy valós vagy hipotetikus világ (a „*rendszer*”) egy részének egyszerűsített képe, amely a rendszert helyettesíti bizonyos megfontolásokban. Egy modell elkészítésének mindig egy kérdés megválaszolása a célja.

A modell tehát egy többnyire bonyolult rendszer egyszerűsített reprezentációja, amiben csak az aktuális probléma szempontjából lényeges vonásokat szerepeltetjük. Egy adott rendszer modellé történő leképezésének általában két fontos előnye van:

- A modell az eredeti rendszernél *kisebb*, hiszen az aktuális problémához nem (vagy lazábban) kapcsolódó, elhanyagolható információk nem jelennek meg benne.
- A modell az eredeti rendszernél *áttekinthetőbb*, hiszen csak az adott probléma szempontjából érdekes, releváns információkat és kapcsolatokat kell vizsgálni.

Példa. Modellekre sok példát láthatunk a hétköznapokban is. Nem csak gyerekek körében népszerű játék a modellvasút. Itt valóban modellről beszélhetünk, hiszen a játékvonatok számos tekintetben hűen reprezentálják a valódi vonatokat, azonban például „szemet hunyunk" a méretükkel, tömegükkel, a bennük lévő villanymotor paramétereivel és még sok egyéb hasonló tulajdonságukkal kapcsolatban.
Az informatika egyik leggazdagabb modellforrása a matematika. Amikor gráfelméletet tanulunk, valójában rengeteg, bizonyos szempontból hasonló probléma közös modelljét vizsgáljuk. Valóban, a matematika egyik célja az ilyen modellek azonosítása és minél hatékonyabb eszköztárak kidolgozása a rajtuk megfogalmazott feladatok megoldására. A gráfoknál maradva (gráfokról bővebben a *Struktúra alapú modellezés* c. segédanyagban és a *Bevezetés a számításelméletbe 2.* c. tárgyban lehet tájékozódni) egy város úthálózata jól reprezentálható egy élsúlyokkal ellátott gráffal, ahol a csomópontok a kereszteződések, az élek az útszakaszok, az élsúlyok pedig a szakaszok hosszait jelölik. Ez a modell kiválóan alkalmas legrövidebb útvonalak tervezésére (mi kellene még a *leggyorsabb* útvonal tervezéséhez?), de figyelmen kívül hagy számos egyéb paramétert, például az utakon lévő kanyarulatokat, a legnagyobb megengedett sebességet stb.

Megjegyzés. Az, hogy a modell kisebb, nem mindig „kényelmi" szempont. Gyakran lehet olyan problémával találkozni, aminek a mérete bizonyos szempontból *végtelen* (például végtelen sok állapota van, folytonos változók vannak benne, esetleg részét képezi az *idő*), viszont egy alkalmas *véges* modellen a számunkra releváns tulajdonságok továbbra is jól vizsgálhatók maradnak. Egy autót fizikai szempontból modellezhet a pillanatnyi sebességvektora, ami három valós szám, tehát a modellünk így végtelen. Ha viszont feltételezzük, hogy a sebesség nagysága nem lehet 200 km/h-nál több, és két tizedesjegy pontossággal adjuk meg a vektor koordinátáit (tehát *diszkrét értékekkel* jellemezzük a rendszert), akkor máris véges modellt kapunk. Természetesen ez a modell kicsit torzítani fog a valósághoz képest, de számos problémánál ez a hiba elhanyagolhatóan kicsi lesz (ráadásul végtelen modellen lehet, hogy nem is tudnánk megoldást adni).

Fontos kérdés, hogy hogyan lehet ábrázolni egy modellt. Maga a modell ugyanis többnyire egy hipotetikus struktúra, nem kézzel fogható és nem is mindig célszerű

teljes részletességgel ábrázolni.

Definíció. *Diagram*: a modell egy nézete, amely a modell bizonyos aspektusait grafikusan ábrázolja.

Megjegyzés. Fontos megjegyezni, hogy nem minden modell, ami modellnek látszik.

- *A modell nem a valóság*: az általunk definiált modellen bizonyos állítások igazak lehetnek, amik a valóságban nem állják meg a helyüket.
- *A modell nem a diagram*: a diagram csak egy ábrázolás módja a modellnek, amivel olvashatóvá tesszük.

2. Modellezési nyelvek

A modellek leírásához nem elég, ha diagramokat rajzolunk. Egy modell precíz megadásához szükség van egy *modellezési nyelvre*. A modellezési nyelvek legfőbb célja a kommunikáció gép-gép, ember-gép, vagy akár ember-ember között is. Egy modellezési nyelv lehet *szöveges* (pl. Verilog, VHDL, Java stb.) vagy *grafikus* (pl. webes konyhatervező, UML diagramok stb.). Gyakori, hogy egy modellezési nyelv szöveges és grafikus jelölésrendszert is definiál, ugyanis mindkettőnek megvannak a saját előnyei és hátrányai: jellemzően modellt építeni könnyebb szöveges nyelv használatával, míg a modell olvasása grafikus nyelvek (diagramok) segítségével egyszerűbb.

Definíció. Egy *modellezési nyelv* négy részből áll:

- *Absztrakt szintaxis* (más néven *metamodell*): meghatározza, hogy a nyelvnek milyen típusú elemei vannak, az elemek milyen kapcsolatban állhatnak egymással, és a típusoknak mi a viszonya egymáshoz. Ezt a reprezentációt használják gépek a modell belső tárolására.
- *Konkrét szintaxis*: az absztrakt szintaxis által definiált elemtípusokhoz és kapcsolatokhoz szöveges vagy grafikus jelölésrendszert definiál. Ezt a reprezentációt használják az emberek a modell olvasására és szerkesztésére.
- *Jólformáltsági kényszer*: Megadja, hogy egy érvényes modellnek milyen egyéb követelményeknek kell megfelelnie.
- *Szemantika*: az absztrakt szintaxis által megadott nyelvi elemek jelentését definiáló szabályrendszer.

Egy modellezési nyelvhez több konkrét szintaxis is adható.

Az absztrakt szintaxis tekinthető a modellezési nyelv modelljének, ezért hívjuk *metamodellnek* is. A konkrét szintaxis a metamodellhez képest annyival több, hogy a definiált elemekhez megjeleníthető reprezentációkat rendel, ezáltal válik olvashatóvá és szerkeszthetővé egy modell leírása. Ha nem okoz félreértést, akkor a szintaxis szót leggyakrabban az absztrakt és konkrét szintaxis együttes jelölésére használjuk (tehát az elemkészletre és a jelölésekre együtt). A jólformáltsági kényszerek tovább

szűkítik a lehetséges modellek körét olyan szabályokkal, mint például az azonos nevű elemek tiltása. Végül a szemantika megadja, hogy az így leírt modell pontosan mit is jelent, vagy hogyan „működik”. A szemantika fogalmát strukturális és viselkedési modellek esetén kissé eltérően értelmezzük majd, erre a későbbi fejezetekben térünk vissza.

Példa. A C nyelv egy szöveges nyelv, melyben elemeknek tekinthetjük az `if`, `for`, `switch`, `változónevek`, `metódusnevek`, `típusok` stb. részeket. Ezek azonban nem tetszőleges sorrendben állhatnak egymás mellett, hanem egy jól meghatározott szabályrendszer szerint. Ez a szintaxis. Azt, hogy az `if` kulcsszóval elágazást, a `for` kulcsszóval pedig egy ciklust definiálunk és nem pedig valami teljesen más dolgot, a szemantika határozza meg. Vagyis a szemantika jelentéssel tölti meg a nyelvi elemeket.

Példa. A Yakindu Statechart Tools[a] egy állapotgépek specifikálását és fejlesztését lehetővé tévő nyílt forráskódú eszköz, mely az Eclipse IDE alapjaira épül. Kényelmi funkcióihoz tartozik az elkészített modell validációja, szimulálásának lehetősége, ill. a modellalapú kódgenerálás is. A szöveges (XML) nyelv mellett egy könnyen használható grafikus felületet is biztosít, ahol lehetőségünk van állapotokat és átmeneteket definiálni. Ezek nyelvi elemként, mint dobozok és nyilak jelennek meg. A grafikai szintaxis meghatározza, hogy a dobozokból csak nyilakkal lehet összekötni más dobozokat. Ekkor igazából azt adjuk meg, hogy egy adott állapotból alkalmazásunk milyen másik állapotba kerülhet. Ez a szemantikai jelentés. Az állapot alapú modellek leírásával részletesen az *Állapot alapú modellezés* fejezetben fogunk foglalkozni.

[a] `http://statecharts.org`

Megjegyzés. A metamodell ugyanúgy modell, mint a segítségével leírható modellek. Ebből kifolyólag ugyanúgy adható hozzá modellezési nyelv, aminek szintén lesz egy metamodellje. A gyakorlatban ez addig folytatódik, amíg az egyik metamodellt már egy szabványos modellezési nyelv segítségével írjuk fel. Érdekesség, hogy az UML metamodell-hierarchiájának gyökerében egy *önleíró modell* található: önmaga a saját metamodellje is egyben.

Példa. Modellekre és metamodellekre számos példa kerül majd elő a későbbi tanulmányok során:

- Az UML [1] objektum diagramján ábrázolt objektummodelleket az osztálydiagramokon ábrázolt metamodelljeik szerint adhatjuk meg (*Szoftvertechnológia* tárgy).
- Az XML dokumentumok metamodelljét az XML sémák adják (*Szoftvertechnológia* tárgy).
- Egy adatbázis tábla metamodellje a relációs adatbázisséma (*Adatbázisok* tárgy).

A konkrét szintaxist (jelölésrendszert) mindegyik esetben külön határozzák meg, az UML esetében többféle grafikus és szöveges szintaxis is definiált.

Tipp. Minden fejezetben, ahol modellezési formalizmusokkal ismerkedünk meg, szerepelni fog az adott modelltípus metamodellje is.

3. Nyílt és zárt világ feltételezés

Egy modell szemantikájának definiálása során több feltételezésből is kiindulhatunk. Ezeket rögzíteni kell, és a modell értelmezésekor döntő szerepük van a különböző állítások igazságtartalmának eldöntésekor.

Definíció. *Zárt világ feltételezés*: Minden állítás, amiről nem ismert, hogy igaz, hamis.

Definíció. *Nyílt világ feltételezés*: Egy állítás annak ellenére is lehet igaz, hogy ez a tény nem ismert.

A két feltételezés között az egyik legfontosabb különbség, hogy a nyílt világ feltételezés elismeri és használja az *ismeretlen* fogalmát, míg a zárt világban minden tudás ismertnek tekintett. Mindkét feltételezésnek megvan a maga szerepe és helye, ahogy azt a következő két példa is illusztrálja.

Példa. A zárt világ feltételezést olyankor alkalmazzuk, amikor egy rendszernek a teljes szükséges információ a rendelkezésére áll. Erre példa egy tömegközlekedési adatbázis: ha megkérdezzük, hogy közlekedik-e közvetlen járat a Magyar Tudósok körútja és a Gellért tér között, és ilyen járat nincs az adatbázisban, akkor a válasz egyértelmű „nem”. Ebben az esetben valóban ez az elvárt és helyes válasz.
A nyílt világ szemantika olyankor alkalmazandó, amikor nem feltételezhetjük, hogy minden információ a rendelkezésünkre áll. Például egy orvosi nyilvántartó rendszerben előfordulhat (sőt, számítani kell rá), hogy egy beteg annak ellenére allergiás valamire, hogy ez a nyilvántartásban nem szerepel.

4. Absztrakció és finomítás

A modell definíciójának szerves része az *egyszerűsítés*, az adott probléma szempontjából irreleváns információk elhanyagolása. Ugyanannak a rendszernek több modelljét is elkészíthetjük más-más részletek kihagyásával. Ráadásul egy elkészített modellből további részleteket hagyhatunk el, vagy éppen újabb részleteket vehetünk bele, így a modellek a részletgazdagság szerint egyfajta hierarchiába rendezhetők.

Különbséget kell tennünk azonban egy modell részletekkel gazdagítása és megváltoztatása között. Ehhez segít, ha megkülönböztetjük a modellt (illetve a modellezett rendszert) és a környezetét.

Definíció. *Rendszer* és *környezete*: a környezet (vagy kontextus) a rendszerre ható tényezők összessége. Modellezéskor a modellezett rendszernek mindig egyértelműen definiálni kell a *határait*. A határon belül eső dolgokat a rendszer részének tekintjük, az azon kívül esők adják a környezetet. Szokás még a környezet elemeit két csoportba sorolni: *releváns* környezeti elemek azok a dolgok, amik a rendszerrel közvetve vagy közvetett módon kapcsolatban állnak, míg *irreleváns* környezeti elem, ami a rendszerrel nincs kapcsolatban.

Megjegyzés. Gyakori, hogy egy rendszer környezetéről (is) készítünk modelleket. Ennek haszna a rendszer és a környezet interakcióinak pontosabb megértése, tervezése és analízise. Az elkészült rendszerek tesztelésekor például gyakran nem az éles környezetben tesztelünk, hanem a környezet modellje alapján szimuláljuk az interakciókat.

A környezet szempontjából nézve a rendszert tekinthetjük *fekete doboznak* vagy *fehér doboznak*. Előbbi esetben a rendszer belső felépítését és viselkedését nem ismerjük, míg utóbbi esetben ezek az információk rendelkezésre állnak. Ez a két fogalom főleg tesztek tervezésekor érdekes, erről bővebben a *Modellek ellenőrzése* fejezetben lesz szó.

Példa. Sportautók tervezésekor a rendszer jól körülhatárolható. Aerodinamikai szempontból releváns külvilágnak tekinthető az autó körül áramló levegő. Tervezéskor célszerű a karosszéria (rendszer) és a légáramlás (környezet) modelljét is elkészíteni, hogy minél pontosabb szimulációkkal megfelelő jóslatokat tudjunk tenni a terv minőségét illetően. Később, amikor az elkészült prototípus tesztelése folyik, a szélcsatorna tekinthető a légáramlás egy pontosabb, fizikai modelljének.

A környezet segítségével definiálhatjuk, hogy mit jelent a modell részletekkel gazdagítása.

Definíció. *Finomítás*: a modell olyan részletezése (pontosítása), hogy a környezet szempontjából nézve a finomított modell (valamilyen tekintetben) helyettesíteni tudja az eredeti modellt.

A finomítás mindig többletismeret bevitelét jelenti a modellbe. A finomítás konkrét módja és a helyettesíthetőség pontos definíciója problémafüggő, amikre sok példát láthatunk majd a későbbi fejezetekben. Természetesen a finomításnak ellentétes művelete is van.

Definíció. *Absztrakció*: a finomítás inverz művelete, vagyis a modell részletezettségének csökkentése, a modellezett ismeretek egyszerűsítése.

A szabályos finomítás után az eredeti modell mindig visszakapható egy szabályos absztrakcióval. Érdemes megjegyezni, hogy finomítás során többnyire tervezői döntést hozunk annak tekintetében, hogy egy adott részletet hogyan illesztünk a modellbe, míg az absztrakció során a részlet elhagyása egyértelmű. Következésképpen egy finomítási lépésnek általában sokféle eredménye lehet, de egy adott absztrakciós lépésnek mindig csak egy.

Példa. Egy közlekedési lámpának megadható egy absztrakt és egy részletes modellje is. Az absztrakt modellben a lámpának két állása van: *tilos* és *szabad*. A részletesebb modellben a *szabad* állapotnak megfelelhet a *zöld* állapot, a *tilos* állapotban pedig a *sárga*, *piros* és *piros-sárga*.
Figyeljük meg, hogy az absztrakt modellből a részletesebbe lépés (finomítás) során többlet ismeretet vittünk a modellbe, mégpedig a lámpákkal kapcsolatban. Ráadásul mivel az új állapotok mindegyike egyértelműen megfeleltethető egy réginek, a környezet számára a rendszer továbbra is leírható az absztrakt modellel.
Azt is láthatjuk, hogy a finomítási lépésnek („bontsuk szét a *tilos* állapotot") több megoldása is lehetett volna, míg az ennek megfelelő absztrakciós lépésnek („vonjuk össze a *sárga*, *piros* és *piros-sárga* állapotokat") csak egyetlen megoldása van (az elnevezésektől eltekintve).

Tipp. A későbbi fejezetekben az adott témakör vonatkozásában számos példát adunk majd absztrakcióra és finomításra.

Hivatkozások

[1] Object Management Group: Information technology – Object Management Group Unified Modeling Language (OMG UML) – part 2: Superstructure. ISO/IEC 19505-2:2012. Jelentés, 2012, Object Management Group.
URL `http://www.omg.org/spec/UML/ISO/19505-2/PDF/`.

Tárgymutató