Modell – Háttér információk

A modell definíció szerint, a valóság vagy hipotetikus világ egy részének egyszerűsített képe, amely a rendszert helyettesíti bizonyos megfontolásokkal. Amikor egy modellt képezünk le egy rendszerről, a nem releváns részek elhagyhatóak. Ezzel a rendszer egy kisebb, véges változatát kapjuk, amivel egyszerűbbé és gyorsabbá válik a munka. Például fizikai kísérletben elhanyagolhatjuk a légellenállást. Azonban ezek a részletek torzítják is a modellt a valósághoz képest, és a valós helyzetben nem biztos, hogy helyes működést fog produkálni. Ráadásul, egyes tényezők annyira komplexek, hogy modellezni sem lehet őket, például emberi döntések.

Szoftverfejlesztésnél is van lehetőség modelleket készítésére, ezt modell alapú fejlesztésnek (model-driven engineering) nevezzük. Modell segítségével könnyedén leírható, hogy a szoftvernek milyen komponensei vannak, azok hogyan viselkednek, és milyen kapcsolatban vannak más komponensekkel. Egy modell legfőbb előnye, hogy lehet hozzá nézeteket készíteni, ezeket diagramoknak hívjuk. A diagramok ugyanis megkönnyítik a kommunikációt, amely a fejlesztés egyik legalapvetőbb aspektusa. Diagramok segítségével gyorsan lehet bonyolult tervezési döntéseket feljegyezni, tárolni és átadni két fél között, legyen az ember vagy gép.

A modellek nemcsak informatikában jártas személyek között segítik az együttműködést. Más tudományágak szakemberei értelemszerűen csak a saját szakterületükhöz értenek, és nem jártasak az informatikában vagy a szoftverfejlesztésben. Azonban készíthetőek olyan diagramok, amelyek specifikusan az adott szakterület jellemzőit használják vagy jelrendszerét hordozzák. Ezáltal más specialistákat, akár stakeholdereket vagy felhasználókat is be lehet vonni a fejlesztésbe.

Egy modell elkészítése nemcsak a tervezés fázisában segíthet. A tervezést megkönnyíti, és átláthatóvá teszi. A modellek egyik tulajdonsága, hogy a fejlesztők szabják meg a modell részletességét. Ha több részlettel gazdagítják a modellt, azt finomításnak nevezzük, ha pedig csökkentik a modell komplexitását, akkor absztrakciót végeznek. A modell mindig igazodni fog ahhoz, hogy a fejlesztők adott pillanatban hol tartanak a fejlesztésben, illetve, hogy mi a tervezés végeztével a kívánt eredmény.

A szoftver implementálása alatt segíthet, ha a diagramokat átolvasva rövid időn belül betekintést kaphatunk a programjaink elvárt működésébe. Futtatható kód generálása is lehetséges, megfelelő feldolgozó és generátor készítésével, és ez a generált programkód meggyorsítja az implementálást.

A modell a tesztelésben, validálásban és verifikálásban is segíthet. Modell készítésével még el nem készült objektumok és komponensek viselkedését lehet szimulálni és méréseket végezni. A modell tesztelhető, hogy minden specifikált követelménynek megfele-e, vagy megvizsgálhatjuk, hogy teszteseteink a modell minden részére   
kiterjednek-e. Továbbá, ha a kódgenerátort is helyesen teszteljük, a modell által generált kód is kevesebb hibát tartalmaz.

Végül a dokumentálásnál is előnyt jelent, ha rendelkezünk egy modellel. A modellből készített diagramok önmagukban is rengeteg információt hordoznak, de akár generálhatunk is leírásokat, paramétereket is a szoftverünk számára.

Modellek és diagramok készítésénél meg van szabva, hogy az informatikai fejlesztők vagy más szakemberek milyen elemeket adhatnak hozzá a modellhez. Ezeket az előírások szabványok (pl. UML) korlátozzák. Azt, hogy a modellben lévő egyes komponensek fajtája mi lehet, és a komponensek milyen kapcsolatban állhatnak egymással, a metamodell írja le. A metamodellre tekinthetünk úgy is, mint egy modellnek a modellje.

Metamodellre példa az UML osztálydiagram és objektumdiagram. Az objektumdiagramban lévő objektumok példányosítják az osztálydiagram osztályait és ugyanahhoz az osztálydiagramhoz több különböző objektumdiagramot készíthetünk. Másik szemszögből vizsgálva, az osztálydiagram leírja, milyen *fajta* elemeket tárolhat egy objektumdiagram, tehát az osztálydiagram az objektumdiagram metamodellje. Hasonló példa egy XML dokumentum, melynek metamodellje az XML séma.

Ezt a gondolatmenetet folytatva, mivel a metamodellek is teljes értékű modellek, ezért azoknak is létezik elkészíthető metamodellje. A metamodellek hierarchiája egy végtelen rekurziót hoz létre, amit egy olyan metamodell oldja fel, amely saját magát írja le, saját maga metamodellje.

A modellezés egyik fajtája a Domain Specific Language-ek (DSL). Egy DSL-re programozási nyelvként lehet tekinteni, azonban kevesebb funkcióval rendelkezik, mint egy átlagos nyelv. A DSL-eket – néhány kivételtől eltekintve – nem általános célokra készítik el, hanem egy konkrét feladat elvégzésére, konkrét terület problémáinak megoládására. Ebből kifolyólag kevés műveletet lehet végezni velük, de eleget ahhoz, hogy az adott feladat teljesíthető legyen. Olyan DSL-ek is léteznek, melyek csupán információk rögzítésére, vagy leírás készítésére szolgál. A DSL-ek legtöbbször nem rendelkeznek saját fordítóval, hanem egy általános célú programnyelvre konvertálódnak át. A DSL-ek másik, kevésbé elterjedt megnevezése a mini-language.

Egy meghatározott feladat elvégzésében sok előnnyel jár a feladathoz készült DSL használata, mint egy megszokott, általános célú programozási nyelv alkalmazása. Ezeket az előnyöket a végrehajtható műveletek korlátozott számának köszönheti a DSL. A DSL-ben történő programozást gyorsabban meg lehet tanulni, mint egy általános célú nyelv használatát. Egy jól kidolgozott DSL átláthatóbb kódot és gyorsabb fejlesztést biztosít. Komplex feladatok elvégzésére, például egy teljes szoftverrendszer megírásához nem létezik DSL, és nem is érdemes ilyen DSL-t készíteni. Azonban többfajta DSL rugalmas használatával bonyolultabb célokat is elérhetünk.

Gyakori, hogy az informatikusoknak olyan szakemberekkel kell együttműködnie, akik nem, vagy keveset értenek informatikához. Az informatikusok pedig az adott szakterületben nem jártasak, ami megnehezíti a fejlesztés számára elengedhetetlen kommunikációt. Ilyen helyzetekre kínálnak megoldást a DSL-ek. Az programozók készíthetnek egy DSL-t, amelyben az idegen szaknyelvet és ismeretlen fogalmakat lehetne megosztani. Az informatikában laikus szakemberek könnyedén megtanulhatják a DSL használatát, és az informatikusok számára érthető információkat lesznek képesek nyújtani.

Szoftverfejlesztők rengeteg DSL-t használnak munkájuk során, akár tudtukon kívül. DSL-nek tekinthető a HTML nyelv, bash szkriptek, reguláris kifejezések, SQL parancsok vagy az XML dokumentumok, bár ezek sokkal általánosabb célúak. A CSS remek példa a DSL-ek könnyű elsajátítására, mivel vannak olyan weboldalkészítők, akik jártasak a CSS használatában, mégsem tekintik magukat programozóknak.

Egy DSL nyelv készítésének főbb momentuma, ha a követelmények adottak, a nyelvben használt szintaxis létrehozása. Egy metamodellt kell létrehozni, mely megszabja, milyen modellelemeket írhatunk a DSL nyelvünkben. Mind a metamodell, mind a nyelvben elkészített program elképzelhető grafikus (ábrák, diagramok) vagy szöveges formában. Mindkét esetben meg kell határozni az elemek formátumát, szöveges esetben a használt jelöléseket, grafikus esetben az alakzatokat. A modellelemek formátumára vonatkozó megkötéseket együttesét nevezzük a nyelv nyelvtanának.

A nyelvtan elkészítésével létrejött a nyelvünk és készen áll, hogy programokat írjunk benne. Egy program megírásakor a DSL-nek először elemeznie kell a kódot (parsing), szintaktikai analízist kell végeznie. Megvizsgálja, hogy a megírt kód, pontosabban modell megfelel-e a korábban megadott szintaxisnak. Egy ilyen elemző modult (parser) körülményes megírni, de már több olyan program is létezik, melyek képes működő elemzőt készíteni adott nyelvtanhoz. Ezeket parser generator-nek vagy compiler-compiler-nek neveznek; a Java világában a legismertebb ilyen eszköz az ANTLR.

Mivel a DSL a modellezés egyik eszköze, egy adott DSL nyelv használatakor sok lehetőségük van a fejlesztőknek, hogy a lekódolt információkból valamilyen szöveges formátumot generáljanak. Ez a formátum általában futtatható kódot jelent, de akár ember számára értelmes szöveget vagy a szoftver által felhasználható paramétereket is szokás generálni. Ezt a műveletet nevezzük kódgenerálásnak.

DSL esetében a kód helyességének ellenőrzése után történhet kódgenerálás. A DSL fejlesztését támogató környezetek a DSL nyelvben megírt programot, mint modellt leképzik egy általános célú programozási nyelvre, például C#-ra vagy Java-ra. Ebben a lépésben már tényleges kódgenerálás történik. Azonban az esetek egy részében, a generált kód nem célnak megfelelő, hiszen az csupán a DSL nyelvben készült modell reprezentációja, további átalakítás, további kódgenerálás szükséges ezt a köztes kódot felhasználva. Természetesen, a köztes kódgenerálás lépése elhagyható, ha nem használunk ilyen környezetet, de ez nem ajánlott, ugyanis a fejlesztést jelentősen megkönnyíti.