14 OBJEKTUMORIENTÁLT TERVEZÉS

TÉMAKÖRÖK

A fejezet célja az együttműködő objektumokon alapuló szoftvertervezés bemutatása. Mire a fejezet végére érünk:

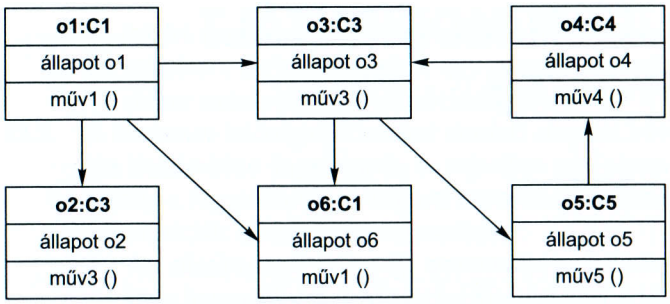
- meg fogjuk érteni, hogyan lehet a szoftvertervet saját állapotukat és műveleteiket kezelő, egymással kölcsönhatásban lévő objektumok segítségével megadni;
- tudni fogjuk, melyek az objektumorientált tervezés általános folyamatának legfontosabb tevékenységei;
- megismerkedünk különböző, objektumorientált tervek leírására szolgáló modellekkel;
- meg fogjuk ismerni ezen modellek UML (Unified Modeling Language) ábrázolásmódját.

TARTALOM

Egy objektumorientált rendszer saját állapotukat karbantartó és erről az állapotról információs műveleteket biztosító, egymással együttműködő objektumokból áll (14.1. ábra). Az állapot reprezentációja privát, az objektumon kívülről közvetlenül nem hozzáférhető. Egy objektumorientált tervezési folyamat az objektumosztályoknak és az azok közötti kapcsolatoknak a megtervezéséből áll. Ezek az osztályok definiálják a rendszer objektumait és a közöttük lévő interakciókat. Ha egy futó program megvalósít egy tervet, akkor a szükséges objektumok az osztálydefiníciók alapján dinamikusan jönnek létre.

Az objektumorientált tervezés az objektumorientált fejlesztés része, amelyben a fejlesztési folyamat során objektumorientált stratégiát használunk:

- Az *objektumorientált elemzés* a szakterületi alkalmazás objektumorientált modelljének kialakításával foglalkozik. A modell objektumai a megoldandó problémával kapcsolatos egyedeket és műveleteket tükrözik.



14.1. ábra. Együttműködő objektumokból felépülő rendszer

- Az *objektumorientált tervezés* a meghatározott követelményeknek megfelelő szoftverrendszer objektumorientált modelljének kialakítása. Az objektumorientált tervezés objektumai a probléma megoldásával kapcsolatosak. Bizonyos problémaobjektumok és megoldásobjektumok igen közeli kapcsolatban állhatnak egymással, de a megoldás megvalósítása érdekében elkerülhetetlen, hogy a tervezők új objektumokat adjanak a rendszerhez, vagy átalakítsák a problémaobjektumokat.
- Az *objektumorientált programozás* a szoftverterv objektumorientált programozási nyelven történő megvalósítása. Egy objektumorientált programozási nyelv – amilyen például a Java is – biztosítja az objektumosztályok definiálásához szükséges konstrukciókat, és egy futtatórendszert az objektumok ezen osztályokból történő létrehozásához.

A fenti lépések közötti átmenetnek ideális esetben észrevehetetlennek kell lennie, és mindegyik lépésben kompatibilis jelölésrendszert kell használni. A következő szakaszba lépés maga után vonja az előző szint finomítását a már létező objektumosztályok részletezésével és a további funkciókat biztosító új osztályok hozzáadásával. Minthogy az információ az objektumokon belül el van rejtve, az adatreprezentációról szóló részletes tervezési döntések elhalaszthatók a rendszer implementálásáig. Bizonyos esetekben, az objektumok elosztására vonatkozó döntés, illetve annak eldöntése, hogy az objektumok szekvenciálisan vagy konkurensen működjenek-e, szintén elhalasztható.

Ez azt jelenti, hogy a szoftvertervezők különböző futási környezetekhez igazítható terveket eszelhetnek ki. Erre példa a modellvezérelt architektúra (Model Driven Architecture, MDA), amely szerint a rendszerek tervezését két szinten kellene végezni (Kleppe *és mások*, 2003): egy implementációfüggetlen és egy implementációfüggő szinten. A rendszer absztrakt modelljét az implementációfüggetlen szinten tervezzük meg, majd ezt leképezzük egy részletgazdagabb, platformfüggő modellre, amely a kódgenerálás alapja lesz. E sorok írásakor az MDA megközelítés még mindig kísérleti fázisban van, és egyelőre megjósolhatatlan, hogy milyen széles körben fog elterjedni.

Az objektumorientált rendszereket a más elven fejlesztett rendszerekkel szemben könnyebb megváltoztatni, mert az objektumok egymástól függetlenek.

Különálló entitásokként foghatók fel és módosíthatók. Egy objektum implementációjának megváltozása vagy új szolgáltatásokkal történő bővülése nem befolyásolhatja a rendszer többi objektumát. Mivel az objektumok dolgokkal kapcsolatosak, a valós világ egyedei (például hardverkomponensek) és a rendszeren belüli, azokat irányító objektumok közötti leképezés gyakran nyilvánvaló. Ez növeli az érthetőséget és így a terv karbantarthatóságát is.

Az objektumok potenciálisan újrafelhasználható komponensek, mivel az állapotnak és a műveleteknek független egységbe zárásai. A tervek az előző tervekben létrehozott objektumok felhasználásával fejleszthetők. Ez csökkenti a tervezés, a programozás és az ellenőrzés költségeit, valamint elvezethet a szabványos objektumok használatához (így növelve a terv érthetőségét) és csökkenti a szoftverfejlesztésben meglévő kockázatot. Azonban, mint ahogy azt a 18. és 19. fejezetekben tárgyalom, az újrafelhasználhatóság megvalósítására sokszor az objektumkollekciók (komponensek és keretrendszerek) használata jobb, mint a különálló objektumoké.

Számos objektumorientált tervezési módszer létezik (Coad és Yourdon, 1990; Robinson, 1992; Jacobsen *és mások*, 1993; Booch, 1994; Graham, 1994). Az ezekben a módszerekben használt jelölésrendszereket az UML-ben egyesítették. A 4. fejezetben tárgyalt Rational Unified Processt (RUP) az UML-ben kifejezhető modellek kihasználására tervezték (Rumbaugh *és mások*, 1999). A fejezetben az UML jelölésrendszerét alkalmazom.

Ahogyan azt a 17. fejezetben tárgyalom, az átfogó frontális tervezésen alapuló rendszerfejlesztés bírálható, mert az átfogó elemzési és tervezési erőfeszítések nem illenek az inkrementális fejlesztéshez és átadáshoz. Az úgynevezett agilis módszerek ennek a problémának a megoldására jöttek létre, és drasztikusan csökkentik, vagy akár teljesen ki is hagyják az objektumorientált tervezési tevékenységet. Én erről azt gondolom, hogy az átfogó, „nehézsúlyú" tervezésre kis- és közepes méretű rendszerek esetén nincs szükség. Nagy rendszerek esetén – különösen kritikus rendszereknél – fontos biztosítani a rendszer különböző részein dolgozó csapatok megfelelő koordinálását. Éppen ezért ebben a fejezetben nem a korábban már használt könyvtári vagy inzulinadagolós példát fogom használni, mert ezek viszonylag kis rendszerek. Ehelyett példaként egy sokkal nagyobb rendszer részét alkalmazom, amelyben a frontális objektumorientált tervezés hasznosabbnak bizonyul.

Egy bizonyos fokig ezt a nézetet tükrözi a Rational Unified Process is, amely a nagy szoftverrendszerek iteratív fejlesztéséhez és inkrementális átadásához köthető. Ez a folyamat egy, a használati esetek és az objektumorientált tervezés köré épült iteratív fejlesztési folyamat, különös figyelemmel az architektúra-központú tervezésre.

A 14.2. alfejezetben tárgyalt tervezési folyamatnak vannak a RUP-pal közös részei, de kisebb hangsúlyt fektet a használati esetek vezérelte fejlesztésre. A használati esetek alkalmazása azt az üzenetet hordozza, hogy a terv felhasználó-központú, és a felhasználók rendszerrel lefolytatott interakcióin alapszik. Az olyan kulcsfigurák követelményeit azonban, akik nem a rendszer közvetlen használói, bonyolult használati esetek segítségével leírni. A használati eseteknek

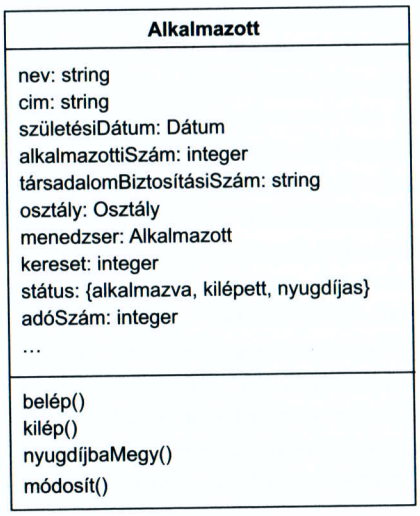
megvan a saját szerepük az objektumorientált elemzés és tervezés folyamatában, de ki kell őket egészíteni más technikákkal a közvetett és nemfunkcionális rendszerkövetelmények felderítése érdekében.

14.1. OBJEKTUMOK ÉS OBJEKTUMOSZTÁLYOK

Az *objektum* és az *objektumorientált* fogalmakat alkalmazzák az egyedek, tervezési módszerek, rendszerek és programozási nyelvek különböző fajtáira is. Általánosan elfogadott azonban, hogy az objektum az információt elrejti. Ez tükröződik az objektumra és az objektumosztályra általam adott definícióban is:

> *Egy objektum egy állapottal és az ezen az állapoton ható, meghatározott műveletekkel rendelkező entitás. Az állapotot objektumattribútumok halmazaként adjuk meg. Az objektum műveletei szolgáltatásokat biztosítanak a többi objektum (a kliensek) számára, amelyek ezekre a szolgáltatásokra bizonyos tevékenységek elvégzése érdekében igényt tarthatnak.*
> *Az objektumok egy objektumosztály-definíció alapján jönnek létre. Egy objektumosztály definíciója egyszerre típusspecifikáció és egy objektumok létrehozására szolgáló sablon. Az adott osztályba tartozó objektummal kapcsolatos összes attribútum és művelet deklarációját tartalmazza.*

Az UML-ben az objektumosztályokat két részre osztott, névvel ellátott téglalap reprezentálja. A felső részben az objektum attribútumai vannak felsorolva. Az objektum műveletei az alsó részben láthatók. Ezt a jelölést a 14.2. ábra mutatja be egy olyan objektumosztályon keresztül, amely egy szervezet egy alkalmazottját modellezi. Az UML-ben a *művelet* kifejezés alatt egy tevékenység specifikációját értjük, míg a *módszer* kifejezés a művelet implementációjára utal.



14.2. ábra. Az Alkalmazott objektumosztály

Az Alkalmazott osztály számos, az alkalmazottakról szóló információt tároló attribútumot definiál, például név, cím, taj-szám, adószám stb. A ... azt jelöli, hogy az adott osztály rendelkezik olyan további attribútumokkal, amelyek itt nincsenek feltüntetve. Az objektumon végezhető műveletek a belép (akkor hívódik meg, ha az alkalmazott csatlakozik az adott szervezethez); a kilép (akkor hívódik meg, ha az alkalmazott elhagyja a szervezetet); a nyugdíjbaMegy (akkor hívódik meg, ha az alkalmazott nyugdíjba megy); illetve a módosít (akkor hívódik meg, ha az alkalmazott adataiban változtatni kell).

Az objektumok úgy kommunikálnak, hogy szolgáltatásokat kérnek más objektumoktól (meghívják azok módszereit), valamint szükség esetén kicserélik egymás között a szolgáltatás ellátásához szükséges információkat. A szolgáltatás végrehajtásához szükséges információ és a szolgáltatás végrehajtásának eredményei paraméterként adódnak át. Erre néhány példa:

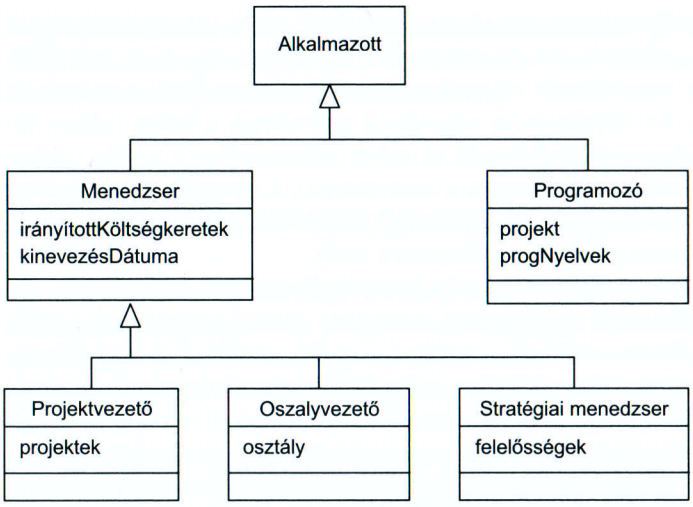
```
// Egy puffer objektum módszerének hívása, amely
// visszaadja a pufferben található következő értéket
v = circularBuffer.Get ();
// A karbantartott hőmérsékletet beállító termosztát
// objektum módszerének hívása
thermostat.setTemp(20);
```

Szolgáltatásalapú rendszerekben az objektumok kommunikációja közvetlenül XML-formátumú szöveges üzenetekkel valósul meg, amelyeket az objektumok kicserélnek egymás között. A fogadóobjektum elemzi az üzenetet, azonosítja a szolgáltatást és az ahhoz kapcsolódó adatokat, majd végrehajtja a kívánt szolgáltatást. Amikor azonban az objektumok ugyanabban a programban egyidejűleg léteznek, a módszerhívások ugyanúgy vannak implementálva, mint az eljárás- és függvényhívások a C-ben.

Ha a szolgáltatáskérelmek ily módon vannak implementálva, az objektumok közötti kommunikáció szinkron, azaz a hívóobjektum megvárja a szolgáltatás befejeződését. Ha azonban az objektumok konkurens folyamatokként vagy szálakként implementáltak, aszinkron kommunikáció lehetséges közöttük, azaz a hívóobjektum folytatja működését az általa igényelt szolgáltatás futása alatt is. Az objektumok konkurens folyamatokként történő megvalósításának mikéntjét az alfejezet későbbi részében magyarázom el.

Ahogyan arról már a 8. fejezetben szóltam – többféle lehetséges objektummodell bemutatása során –, az objektumosztályok egy generalizációs vagy öröklődési hierarchiába szervezhetők, amely az általános és a specifikus objektumosztályok közötti kapcsolatot jeleníti meg. A specifikusabb objektumosztályok teljesen konzisztensek az általános objektumosztályokkal, de további információt tartalmaznak. Az UML-ben a generalizációt a szülőosztályra mutató nyíllal jelöljük. Az objektumorientált programozási nyelvekben a generalizáció az öröklődés segítségével implementálható. A leszármazott osztály örökli szülőosztályának attribútumait és műveleteit.

A 14.3. ábrán látható példa egy ilyen hierarchiát mutat be, az alkalmazott különböző osztályai láthatók. A hierarchiában lejjebb található osztályok rendelkez-



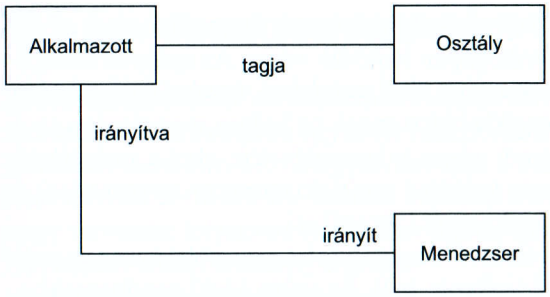
14.3. ábra. Generalizációs hierarchia

nek ugyanazokkal az attribútumokkal és műveletekkel, mint szülőosztályaik, de új attribútumokkal és műveletekkel egészítheti ki azokat, valamint meg is változtathatják szülőosztályaik attribútumainak és műveleteinek némelyikét. Ez egyirányú felcserélhetőséget jelent. Ha a modellben egy szülőosztály neve szerepel, az azt jelenti, hogy az objektum a rendszerben ezen osztálynak vagy annak valamely leszármazottjának lehet példánya.

A 14.3. ábrán látható Menedzser osztály rendelkezik minden olyan attribútummal és művelettel, amivel az Alkalmazott, és ezenkívül két új attribútuma is van: az egyik a menedzser által felügyelt költségkereteket jelzi, a másik pedig a menedzser kinevezésének dátumát tartalmazza. Hasonlóan a Programozó osztály is két új attribútumot vezet be: az egyik azt a projektet adja meg, amin a programozó dolgozik, a másik pedig az általa ismert programozási nyelveket írja le. A Menedzser és Programozó osztályok objektumai így bárhol használhatók, ahol Alkalmazott osztályba tartozó objektumra van szükség.

Az objektumosztályokba tartozó objektumok kapcsolatban vannak más objektumokkal. Ezek a kapcsolatok objektumosztályok közötti asszociációk leírásával modellezhetők. UML-ben az asszociációkat az objektumosztályok közötti vonallal és esetlegesen az asszociációról szóló megjegyzésekkel jelzik. Erre a 14.4. ábrán láthatunk példát, amely az Alkalmazott és Osztály, illetve az Alkalmazott és Menedzser osztályok objektumai közötti asszociációt írja le.

Az asszociáció nagyon általános kapcsolat és az UML-ben gyakran annak jelzésére használják, hogy az objektum egyik attribútuma egy vele kapcsolatban álló objektum, vagy pedig hogy az objektum egyik módszerének implementációja a vele kapcsolatban álló objektumtól függ. Azonban – legalábbis elviekben – tetszőleges asszociációfajta lehetséges. Az egyik leggyakoribb asszociációs kapcsolat az aggregáció, amely azt mutatja be, hogy az objektumok hogyan épülnek fel más objektumokból. Az asszociációnak ezt a fajtáját a 8. fejezetben tárgyaltam.



14.4. ábra. Asszociációs modell

14.1.1. Konkurens objektumok

Fogalmi szinten azt mondhatjuk, hogy egy objektum „szolgáltatáskérés" üzenetet küld annak az objektumnak, amelytől szolgáltatást kér. A soros végrehajtás, amikor az objektum megvárja a kért szolgáltatás befejezését, nem követelmény. Ebből következően az objektumok közötti kölcsönhatás általános modellje lehetővé teszi az objektumok számára, hogy azok konkurens módon, párhuzamos folyamatokként legyenek végrehajtva. Ezek az objektumok akár egyazon számítógépen, akár különböző gépeken osztott objektumokként is végrehajtódhatnak.

A gyakorlatban a legtöbb objektumorientált programozási nyelvben a soros végrehajtási modell az alapértelmezés, amelyben az objektumok szolgáltatásai iránti kérelem ugyanúgy van implementálva, mint a függvényhívások. Éppen ezért, amikor egy theList nevű objektumot létrehozunk egy szokványos objektumosztályból, a következőt írjuk Javában:

```
theList.append(17);
```

Ez meghívja a theList objektum append módszerét, amely hozzáadja a theList-hez a 17-es elemet, és a hívó objektum végrehajtása mindaddig felfüggesztődik, amíg a hozzáfűzési művelet be nem fejeződik. A Java azonban tartalmaz egy nagyon egyszerű mechanizmust (a szálakat), amely megengedi a konkurens módon végrehajtódó objektumok létrehozását. Éppen ezért objektumorientált terv alapján könnyű olyan implementációt készíteni, ahol az objektumok konkurens folyamatok.

A konkurens objektumok kétféleképpen implementálhatók:

1. *Szerverek,* ahol az objektum a megadott objektummműveleteknek megfelelő módszerekkel rendelkező párhuzamos folyamat. A módszerek egy külső üzenetre válaszolva indulnak el és más objektumok módszereivel párhuzamosan futhatnak. Mikor befejezték tevékenységüket, az objektum várakozó állapotba kerül, és további kérésekre vár.
2. *Aktív objektumok,* ahol az objektum állapotát az objektumon belül végrehajtódó belső műveletek megváltoztathatják. Az objektumot reprezentáló folyamat ezeket a műveleteket folyamatosan végrehajtja, így soha nem függesztődik fel.

A szervereket leginkább osztott környezetben érdemes használni, ahol a hívó és a hívott objektum különböző számítógépen hajtódik végre. Az igényelt szolgáltatásra adott válaszidő megjósolhatatlan, így ahol csak lehet, úgy kell megtervezni a rendszert, hogy a szolgáltatást igénylő objektumnak ne kelljen megvárnia a szolgáltatás befejezését. A szerverek egyedi gépen is használhatók, ahol a szolgáltatás befejezéséhez némi időre van szükség (például egy dokumentum nyomtatása) és a szolgáltatást számos különböző objektum is igényelheti.

Az aktív objektumokat akkor célszerű használni, ha egy objektumnak saját állapotát megadott időközönként frissítenie kell. Ez valós idejű rendszerekben gyakori, ahol az objektumok a rendszer környezetéről információt gyűjtő hardvereszközökkel állnak kapcsolatban. Az objektum módszerei a többi objektum számára is hozzáférést biztosítanak az állapotinformációhoz.

A 14.5. ábra azt mutatja meg, hogy hogyan lehet egy aktív objektumot definiálni és implementálni Javában. Ez az objektumosztály egy repülőgép helyzetérzékelőjét reprezentálja. A helyzetérzékelő a repülőgép pozícióját egy műholdas navigációs rendszer segítségével követi nyomon, és képes reagálni a légi irányítást vezérlő számítógépektől kapott üzenetekre. A givePosition kérésre adott válasza megadja a repülőgép jelenlegi pozícióját. Ez az objektum szálként van implementálva, ahol a run módszerben található végtelen ciklus magjában olvasható kód a műholdakról kapott jelek alapján kiszámítja a repülőgép pozícióját.

```
class Transponder extends Thread {

        Position currentPosition;
        Coords c1, c2;
        Satellite sat1, sat2;
        Navigator theNavigator;

        public Position givePosition()
        {
            return currentPosition;
        }
        public void run()
        {
            while (true)
            {
                c1 = sat1.position();
                c2 = sat2.position();
                currentPosition = theNavigator.compute(c1, c2);
            }
        }
} // Transponder
```

14.5. ábra. Aktív objektum implementációja Java-szálak segítségével

14.2. EGY OBJEKTUMORIENTÁLT TERVEZÉSI FOLYAMAT

Ebben a szakaszban az objektumorientált tervezés folyamatát egy automatizált meteorológiai állomásba ágyazott vezérlőszoftver terve kifejlesztésének példáján keresztül mutatom be. Ahogyan azt már a bevezetőben is említettem, az objektumorientált tervezésnek számos módszere létezik, de nincs „legjobb" módszer vagy tervezési folyamat. Az itt leírt eljárás általános, egyesíti a legtöbb OOD-folyamatban közös tevékenységeket.

Az objektumorientált tervezéshez itt használt általános folyamat több lépésből áll:

1. a rendszer összefüggéseinek és használati módjainak megértése és meghatározása;
2. a rendszer architektúrájának megtervezése;
3. a rendszer legfontosabb objektumainak azonosítása;
4. tervezési modellek fejlesztése;
5. objektuminterfészek meghatározása.

Ezt szándékosan nem egyszerű folyamatdiagramként ábrázoltam, mert abból az következne, hogy a folyamat tevékenységei között van valamilyen világos sorrend. Valójában a fentiekre mint egymást átfedő, összefésült tevékenységekre kell gondolnunk. A rendszer architektúrájának definiálásával azonosítjuk az objektumokat és teljesen vagy részben meghatározzuk az interfészeket. Az objektummodellek létrejöttével ezek az egyedi objektumdefiníciók finomíthatók, ami a rendszer-architektúra változását jelentheti.

A szakasz hátralévő részében ezeket a lépéseket a tervezési folyamat különálló lépéseiként tárgyalom. Ebből azonban nem szabad azt a következtetést levonni, hogy a tervezés egyszerű, jól strukturált folyamat. A valóságban a terv úgy készül, hogy megoldásokat vázolunk fel és azokat finomítjuk, mihelyt az információ elérhetővé válik. Ha problémák merülnek fel, elkerülhetetlenül vissza kell lépnünk és újra kell próbálkoznunk. Néha részletesen meg kell vizsgálni a választási lehetőségeket, hogy működőképesek-e; máskor a részletek egészen a folyamat kései szakaszáig figyelmen kívül hagyhatók.

Kövessük végig a fenti tevékenységeket egy példa kidolgozásán keresztül. Az objektumorientált tervezés bemutatására használt példa része egy meteorológiai térképek – automatikusan összegyűjtött meteorológiai adatok segítségével történő – létrehozására szolgáló rendszernek. Egy ilyen meteorológiai térképrendszer részletes követelményei több oldalt is kitennének, azonban az átfogó rendszer-architektúra viszonylag rövid rendszerleírásból is kialakítható:

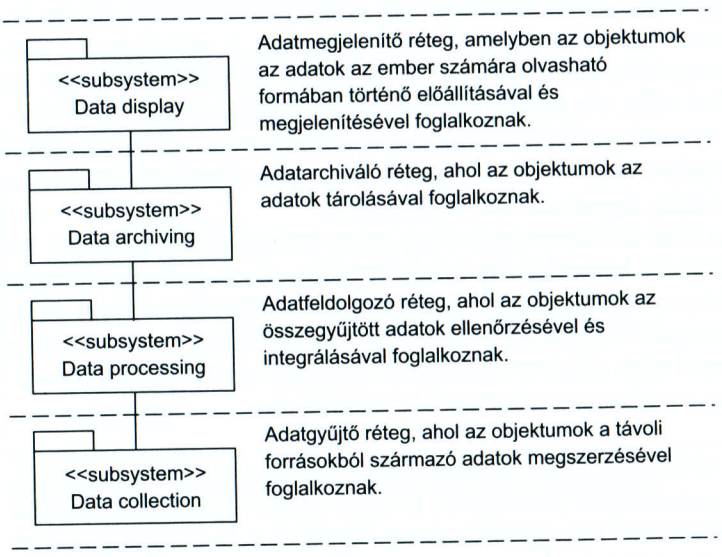
> *Egy meteorológiai térképrendszernek meteorológiai térképeket kell előállítania, távoli, felügyelet nélküli meteorológiai állomásoktól és egyéb adatforrásoktól, mint például léggömböktől és műholdaktól összegyűjtött adatok alapján. A meteorológiai állomások adataikat egy körzeti számítógép kérésére megküldik a kérelmező gépnek.*

A körzeti számítógépes rendszer ellenőrzi a begyűjtött adatokat és integrálja a különböző adatforrásokból érkező adatokat. Az integrált adatokat archiválja, és az archívum, valamint egy digitalizált térképadatbázis alapján létrehozza a helyi meteorológiai térképeket. A térképek egy speciális célú térképnyomtatón kinyomtathatók, majd szétoszthatók, vagy számos más módon megjeleníthetők.

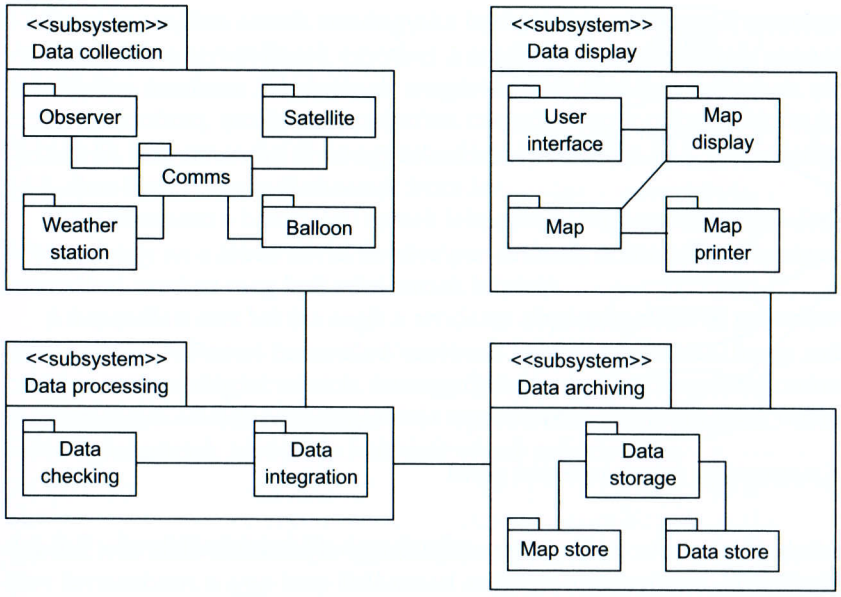
Ebből a leírásból kitűnik, hogy a teljes rendszer egy adatgyűjtéssel, egy különböző forrásból jött adatok integrálásával, egy adatok archiválásával és egy meteorológiai térképek készítésével foglalkozó résszel rendelkezik. A 14.6. ábra egy ebből a leírásból származtatható lehetséges rendszer-architektúrát mutat be. Ez egy rétegezett architektúra (amit a 11. fejezet tárgyal), amely a rendszer feldolgozási folyamatának különböző lépéseit, nevezetesen az adatgyűjtést, az adatintegrációt, az adatarchiválást és a térképkészítést tükrözi. A rétegezett architektúra megfelelő ebben az esetben, mert minden szint működéséhez csak az előző szint eredményére van szükség.

A 14.6. ábra a különböző rétegeket mutatja be. Az UML csomagszimbólumában helyeztük el az egyes rétegek neveit, amelyeket alrendszerekként jelöltem. Egy UML-csomag objektumok és egyéb csomagok gyűjteményét ábrázolja. Itt annak bemutatására használtam, hogy minden réteg számos egyéb komponenst tartalmaz.

A 14.7. ábrán ezt az absztrakt architekturális modellt kibővítettem az alrendszerek komponenseinek bemutatásával. Ezek a komponensek még mindig nagyon absztraktak és a rendszer leírásában olvasható információkból származtathatók. A tervezési példát a meteorológiai állomás alrendszerre összpontosítva folytatjuk, amely az adatgyűjtési réteg része.



14.6. ábra. Meteorológiai térképrendszer rétegezett architektúrája



14.7. ábra. A meteorológiai térképrendszer alrendszerei

14.2.1. Rendszerkörnyezet és a használat modelljei

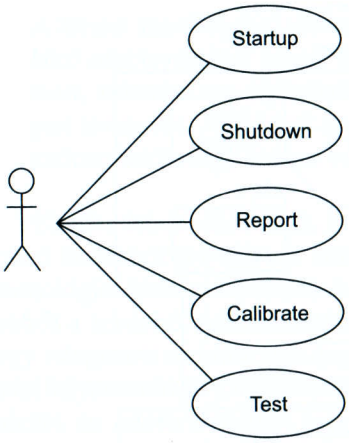
Minden szoftvertervezési folyamat első lépéseként meg kell érteni a tervezés alatt álló szoftver és annak külső környezete közötti kapcsolatot. Ez a megértés segít eldönteni, hogyan biztosítsuk a rendszer szükséges funkcióit, és hogyan struktúráljuk a rendszert annak érdekében, hogy az tudjon a környezetével kommunikálni.

A rendszer környezete és a rendszerhasználat modellje a rendszer és környezete közötti kapcsolatok két, egymást kiegészítő modelljét ábrázolják:

1. A rendszerkörnyezet statikus modell, amely leírja a környezetben található többi rendszert.
2. A rendszerhasználat modellje dinamikus modell, amely azt írja le, hogy a rendszer hogyan működik együtt valójában a környezetével.

Egy rendszer környezeti modelljét asszociációk segítségével (14.4. ábra) írhatjuk le, amikor is egy egyszerű blokkdiagram készül a teljes rendszer-architektúráról. Ebből UML-csomagok segítségével egy alrendszermodellt származtatunk, mint ahogy az a 14.7. ábrán látható. Ez a modell bemutatja a meteorológiai állomás rendszerkörnyezetét az adatgyűjtéssel foglalkozó alrendszeren belül, valamint a meteorológiai térképrendszert felépítő egyéb alrendszereket is.

Ha a rendszer környezetével való együttműködését modellezzük, egy olyan absztrakt megközelítést kell alkalmaznunk, amely nem részletezi túlságosan eze-



14.8. ábra. A meteorológiai állomás használati esetei

ket a kölcsönhatásokat. Az RUP azt javasolja, hogy olyan használati eset modelleket fejlesszünk ki, amelyekben minden használati eset egy, a rendszerrel való együttműködést reprezentál. A felhasználói eset modellekben (7. fejezet) minden lehetséges interakció egy ellipszisbe írt névvel van ellátva, az együttműködő külső entitást pedig egy pálcikaember ábrázolja. A meteorológiai állomás esetében ez a külső entitás nem ember, hanem a meteorológiai adatok feldolgozó rendszere.

A meteorológiai állomás felhasználói eset modellje a 14.8. ábrán látható. A meteorológiai állomás külső entitásokkal működik együtt az indítás és leállítás, az összegyűjtött meteorológiai adatokból való jelentéskészítés, valamint a műszerek tesztelése és kalibrálása céljából.

Rendszer	Meteorológiai állomás
Használati eset	Report
Aktorok	Meteorológiai adatgyűjtő rendszer, meteorológiai állomás
Adatok	A meteorológiai állomás a gyűjtési időszakban a műszerektől begyűjtött meteorológiai adatokról küld összefoglalást a meteorológiai adatgyűjtő rendszernek. Az elküldött adatok: a maximális, minimális és átlagos talaj- és levegő-hőmérséklet, a maximális, minimális és átlagos légnyomás, a maximális, minimális és átlagos szélsebesség, a teljes esőmennyiség és a szélirány ötpercenként vett mintái.
Inger	A meteorológiai adatgyűjtő rendszer modemkapcsolatot létesít a meteorológiai állomással és adatátvitelt kérelmez.
Válasz	Az összegzett adatok elküldésre kerülnek a meteorológiai adatgyűjtő rendszernek.
Megjegyzések	A meteorológiai állomásoktól általában óránként kérnek jelentéseket, de ez a gyakoriság állomásról állomásra változhat, és a jövőben akár módosulhat is.

14.9. ábra. A Report használati esetének leírása

Ezen használati esetek mindegyike leírható egy strukturált természetes nyelven. Ez segít a tervezőknek egyrészt a rendszer objektumainak azonosításában, másrészt a rendszer feladatának megértésében. Ennek a leírásnak stilizált formáját használom, amely egyértelműen megadja, hogy milyen információk cserélődnek ki, hogyan indul el az együttműködés stb. Ez a 14.9. ábrán látható, ahol a 14.8. ábra Report használati esetét írom le.

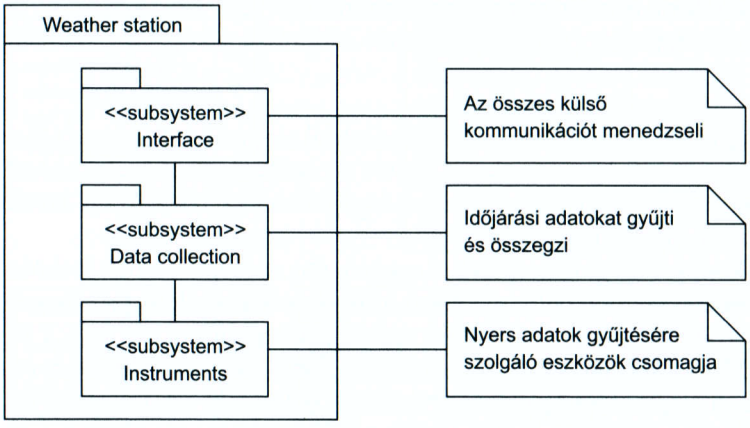
Természetesen a használati esetek leírására bármilyen módszer alkalmazható, feltéve, hogy az a leírás rövid és könnyen érthető. A modellben szereplő minden használati esethez meg kell adni annak leírását.

A használati eset leírása segít a rendszer objektumainak és műveleteinek azonosításában. A Report használati esetének leírásából kitűnik, hogy szükség van mind a meteorológiai adatok összegyűjtését végző műszereket, mind pedig a meteorológiai adatok összefoglalását reprezentáló objektumokra, valamint a meteorológiai adatok kérését és küldését végző műveletekre.

14.2.2. Architekturális tervezés

Miután megadtuk a tervezés alatt álló szoftverrendszer és annak környezete közötti együttműködést, ez az információ alapjául szolgálhat a rendszer architektúrájának tervezéséhez. Természetesen ezt kombinálni kell az architekturális tervezés alapelveinek általános ismeretével és részletesebb szakterületi tudással.

Az automatizált meteorológiai állomás egy viszonylag egyszerű rendszer, architektúrája újfent ábrázolható rétegezett modellként. Ezt a 14.10. ábrán láthatjuk, ahol az általánosabb Weather station csomagon belül három másik UML-csomag található. További információ leírására az UML-megjegyzés jelölését (lehajtott sarkú dobozba írt szöveg) használjuk.



14.10. ábra. A meteorológiai állomás architektúrája

A meteorológiai állomás szoftverének három rétege a következő:

1. az *interfészréteg*, amely a rendszer többi részével való kommunikációval foglalkozik és biztosítja a rendszer külső interfészeit;
2. az *adatgyűjtő réteg*, amely az adatok műszerektől való begyűjtésének kezelésével és a begyűjtött meteorológiai adatok térképrendszerhez történő elküldése előtti összegzésével foglalkozik;
3. a *műszerek rétege*, amely magában foglalja az időjárási viszonyokról nyers adatokat gyűjtő összes műszert.

Egy rendszert általában úgy kell felbontani, hogy az architektúrák a lehető legegyszerűbbek legyenek. Ez megközelítőleg azt jelenti, hogy egy architekturális modellben ne legyen hétnél több alapvető entitás. Ezek az entitások különkülön is leírhatók, de természetesen az entitások szerkezete a 14.7. ábrán látható módon is feltárható.

14.2.3. Objektumok azonosítása

Mire a tervezési folyamatnak erre a szintjére érünk, meg kell fogalmaznunk néhány gondolatot a tervezett rendszer lényeges objektumairól. A meteorológiai állomás rendszerében nyilvánvaló, hogy a műszerek objektumok, és hogy minden architekturális szinten szükség van legalább egy objektumra. Ez fejezi ki azt az általános elvet, hogy az objektumok a tervezési folyamat során bukkannak fel.

Habár ennek az alfejezetnek a címe az „objektumok azonosítása", a gyakorlatban ez a folyamat az objektumosztályok azonosításával foglalkozik. A tervet ezeknek az osztályoknak a fogalmaival írjuk le. A kezdetben azonosított objektumosztályokat elkerülhetetlenül finomítani kell, és a terv egyre mélyebb megértésével vissza kell térni a folyamatnak erre a pontjára.

Az objektumosztályok azonosítására számos különféle elképzelés született már, ezek:

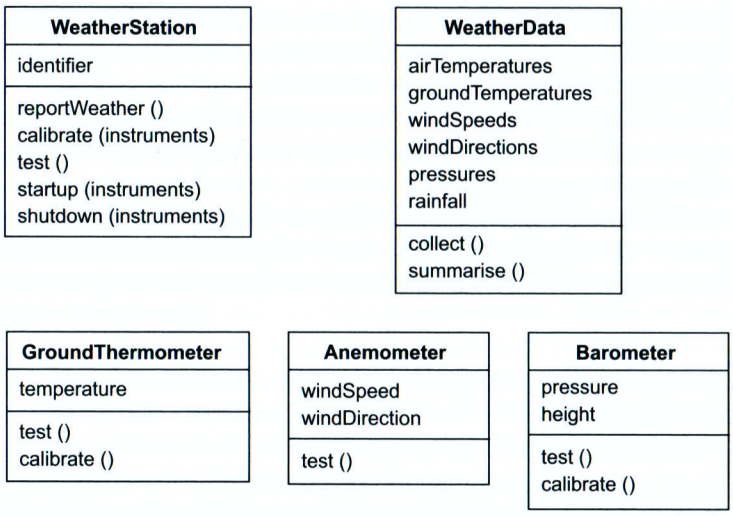
1. A rendszer természetes nyelvi leírásának nyelvtani elemzése. Az objektumok és attribútumok a főnevek, a műveletek, illetve szolgáltatások az igék (Abbott, 1983). Ez a módszer került be az európai légiiparban széles körben ismert HOOD-módszerbe (Robinson, 1992).
2. A szakterület konkrét dolgainak (például repülőgép), szerepköreinek (például menedzser), eseményeinek (például kérés), interakcióinak (például találkozó), helyszíneinek (például iroda), szervezeti egységeinek (például cég) stb. a felhasználása (Shlaer és Mellor, 1988; Coad és Yourdon, 1990; Wirfs-Brock *és mások*, 1990). Ez elősegíthető a tárolási szerkezetek (absztrakt adatszerkezetek) meghatározásával.
3. Viselkedési megközelítés használata, amelyben a tervező először megérti a rendszer teljes viselkedését. A különböző viselkedéseket a rendszer külön-

böző részeihez kell rendelni, és a viselkedés megértése az alapján történhet, hogy ki kezdeményezte és ki vett részt benne. A jelentős szerepkörrel rendelkező résztvevők lesznek az objektumok (Rubin és Goldberg, 1992).

4. Forgatókönyv-alapú elemzés alkalmazása, ahol a rendszerhasználat különböző forgatókönyveit felváltva azonosítják és elemzik. Az egyes forgatókönyvek elemzésekor az elemzést végző csapatnak azonosítania kell a szükséges objektumokat, attribútumokat és műveleteket. A CRC-kártyákon alapuló elemzés – ahol az elemzők és tervezők magukra veszik az objektumok szerepét – hatékonyan támogatja a forgatókönyv-alapú megközelítést (Beck és Cunningham, 1989).

Ezek a módszerek abban segítenek, hogy hogyan kezdjünk neki az objektumazonosításnak. A gyakorlatban a különböző objektumok és objektumosztályok felfedezésére több, különböző forrásból származó ismeretet használunk. A tervezés kiindulópontjaként az objektumosztályok, attribútumok és műveletek kezdetben az informális rendszerleírás alapján azonosíthatók, majd később ezek a kezdeti objektumok a szakterület ismereteiből, illetve a forgatókönyv-elemzésből nyert további információk alapján finomíthatók és bővíthetők. Ezek az információk a követelményt leíró dokumentumokból, a felhasználókkal történő megbeszélésekből és létező rendszerek elemzéséből nyerhetők ki.

A meteorológiai állomás objektumainak azonosítására egy hibrid módszert használok. Terjedelmi okokból nem írom le az összes objektumot, csak a 14.11. ábrán látható ötöt. A GroundThermometer, Anemometer és Barometer objektumosztályok a szakterület objektumait jelölik, a WeatherStation és WeatherData objektumokat pedig a rendszer és a forgatókönyv (használati esetek) leírása alapján azonosítottuk.



14.11. ábra. Példák a meteorológiai állomás rendszerének objektumosztályaira

Ezek az objektumok a rendszer architektúrájának különböző szintjeihez kapcsolhatók:

1. A WeatherStation objektumosztály a meteorológiai állomás és környezete közötti alapinterfészt adja meg. Műveletei ezért a 14.8. ábrán látható interakciókat tükrözik. Ebben az esetben egyetlen objektumosztályt használtunk ezen tevékenységek egységbe zárására, de más tervekben elképzelhető, hogy a rendszer interfészének biztosítására több osztályt is kell használni.
2. A WeatherData objektumosztály a meteorológiai állomás különféle műszereitől kapott összesített adatokat kezeli. Műveletei a szükséges adatok összegyűjtésével és összegzésével foglalkoznak.
3. A GroundThermometer, az Anemometer és a Barometer objektumosztályok közvetlenül a rendszer műszereivel állnak kapcsolatban. A rendszer kézzelfogható hardverentitásait fejezik ki, a műveleteik pedig ezen hardverek vezérlésével foglalkoznak.

A tervezési folyamatnak ezen a szintjén további objektumok és szolgáltatások azonosítására a szakterület ismerete szolgálhat. Példánkban tudjuk, hogy a meteorológiai állomások gyakran egymástól távol helyezkednek el. Minden meteorológiai állomáson különféle műszerek vannak, amelyek el is romolhatnak. A műszerek meghibásodásait automatikusan jelenteni kell. Ez magával vonja olyan attribútumok és műveletek szükségességét, amelyek a műszerek helyes működését ellenőrzik. Nyilvánvalóan több távoli meteorológiai állomás létezik, ezért valamilyen módon azonosítani kell az egyes állomásoktól begyűjtött adatokat, így minden meteorológiai állomás saját azonosítóval rendelkezik majd.

Úgy döntöttem, hogy az egyes műszerekkel kapcsolatban álló objektumok ne legyenek aktív objektumok. A WeatherData objektum collect művelete felszólítja a műszert reprezentáló objektumot, hogy szükség esetén olvassa le a műszer állását. Az aktív objektumokba beleértjük saját vezérlésüket is, ami ebben az esetben azt jelentené, hogy minden műszer saját maga döntené el, hogy mikor olvassa le az állást. Ennek hátránya azonban, hogy új objektumosztályok bevezetésére lehet szükség, ha az adatgyűjtés időzítése megváltozik, vagy ha a különböző meteorológiai állomások különbözőképpen gyűjtik az adatokat. Azáltal, hogy a műszerobjektumok csak erre vonatkozó igény esetén olvassák le saját állapotukat, az adatok összegyűjtésének stratégiájában bekövetkező változások könnyen implementálhatók anélkül, hogy a műszerekhez tartozó objektumokat meg kellene változtatni.

14.2.4. Tervezési modellek

A tervezési modellek a rendszer objektumait és objektumosztályait, valamint alkalmas esetben az ezen entitások közötti különféle kapcsolatokat mutatják meg. Lényegében a terv tervezési modellek összessége. Ezek alkotják a rendszer követelményei és implementálása közötti hidat. Ez azt jelenti, hogy ezeknek a mo-

delleknek egymásnak ellentmondó követelményeik vannak. Absztraktnak kell lenniük ahhoz, hogy a szükségtelen részletek ne rejtsék el a köztük és a rendszerkövetelmények között meglévő kapcsolatokat. Ugyanakkor elég részletesnek kell lenniük ahhoz, hogy a programozók implementációs döntéseket hozhassanak.

Általában ezt a problémát különböző – eltérő részletezettségű – modellek kifejlesztésével kerüljük meg. Ahol a követelménytervezők, a rendszertervezők és a programozók között igen szoros kapcsolat van, csak absztrakt modellekre van szükség. A rendszer implementálásakor speciális tervezési döntések történhetnek. Ha a rendszert specifikálók, a tervezők és a programozók közötti kapcsolat csak közvetett (például ha a rendszer tervezése és implementálása különböző szervezetekben, illetve egy szervezet különböző részeiben történik), részletesebb modellekre van szükség.

A tervezési folyamat egyik fontos lépése ezért az, hogy eldöntsük, milyen tervezési modellekre van szükségünk és azok milyen részletesek legyenek. Ez is a kifejlesztendő rendszer fajtájától függ. Egy szekvenciális adatfeldolgozó rendszert másképpen kell megtervezni, mint egy beágyazott valós idejű rendszert, és ezért ezek tervezésekor különböző tervezési modelleket kell használni. Nagyon kevés olyan rendszer van, ahol minden modellre szükség van, és a létrehozott modellek számának minimalizálása egyaránt csökkenti a tervezés költségeit, valamint a tervezési folyamat befejezéséhez szükséges időt.

Egy objektumorientált terv leírására normális esetben kétféle tervezési modellt kell megalkotni:

1. *Statikus modelleket,* amelyek a rendszer objektumosztályainak és a közöttük meglévő kapcsolatoknak a segítségével írják le a rendszer statikus szerkezetét. Ezen a szinten írhatók le olyan fontos kapcsolatok, mint a generalizációs, a használja/használva van és a kompozíciós kapcsolatok.
2. *Dinamikus modelleket,* amelyek a rendszer dinamikus szerkezetét írják le, és megmutatják az objektumok (nem objektumosztályok!) közötti kölcsönhatásokat. A leírható kölcsönhatások között meg kell említeni az objektumok szolgáltatáskéréseinek sorozatát és azt a módot, ahogyan a rendszer állapota ezekhez az objektum-interakciókhoz kapcsolódik.

Az UML a terv dokumentálására 12 különböző statikus és dinamikus modellt biztosít. Terjedelmi okokból nem részletezem mindet, és nem is mindegyik felel meg a meteorológiai állomás példájának. Az ebben a szakaszban tárgyalt modellek a következők:

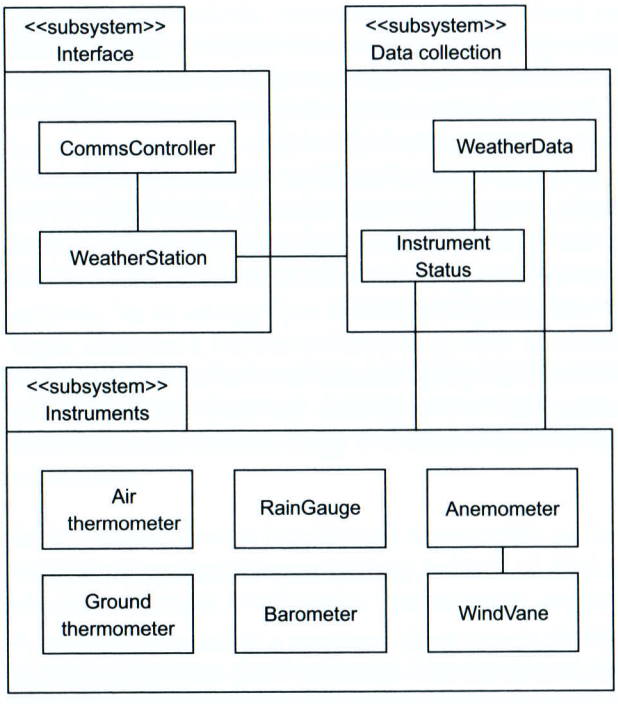
1. *Alrendszermodellek,* amelyek az objektumok összefüggő alrendszerekbe történő logikai csoportosítását írják le. Ezeket egy olyan osztálydiagram segítségével írhatjuk le, ahol az egyes alrendszerek csomagként jelennek meg. Az alrendszermodellek statikusak.
2. *Szekvenciamodellek,* amelyek az objektumok interakcióinak sorrendjét írják le. Ezeket egy UML szekvencia- vagy együttműködési diagram segítségével írhatjuk le. A szekvenciamodellek dinamikusak.

3. *Állapotgépmodellek,* amelyek azt mutatják meg, hogy az egyes objektumok, bizonyos események bekövetkezése esetén, hogyan változtatják meg állapotukat. Ezeket UML-ben állapotdiagramok segítségével ábrázolhatjuk. Az állapotgépmodellek dinamikusak.

Az objektumorientált elemzés és tervezés során használható egyéb modelleket már korábban tárgyaltam. A használati eset modellek a rendszerrel való kölcsönhatást mutatják be (14.8. ábra; 7.6. és 7.7. ábra, 7. fejezet), az objektummodellek az objektumosztályokat írják le (14.2. ábra), a generalizációs vagy öröklődési modellek (8.10., 8.11. és 8.12. ábra, 8. fejezet) azt mutatják be, hogy hogyan lehetnek egyes osztályok más osztályok generalizációi, míg az aggregációs modellek (8.13. ábra) azt mutatják meg, hogy az objektumok kollekciói hogyan állnak egymással kapcsolatban.

A 14.12. ábra a meteorológiai állomás alrendszereinek objektumait mutatja be, a modell néhány kapcsolatával együtt. A CommsController objektum például kapcsolatban van a WeatherStation-nel, ami pedig kapcsolatban áll a Data collection csomaggal. Ez utóbbi azt jelenti, hogy a WeatherStation objektum a csomag egy vagy több objektumával van kapcsolatban. A csomagmodell és az objektum-osztály-modell együttesen leírják a rendszer logikai felépítését.

Az alrendszermodellek igen hasznos statikus modellek, mivel azt mutatják meg, hogy hogyan lehet a tervet objektumok logikailag összefüggő csoportjaiba



14.12. ábra. A meteorológiai állomás csomagjai

szervezni. Ilyen modellre a 14.7. ábrán már láttunk példát, ahol egy meteorológiai térképrendszer alrendszerei látszanak. Az UML-ben a csomagok a bezárás eszközei, és nincsenek közvetlen hatással a fejlesztés alatt álló rendszer entitásaira, azonban közvetlenül befolyásolhatják az olyan szerkezeti konstrukciókat, mint amilyenek a Java-könyvtárak.

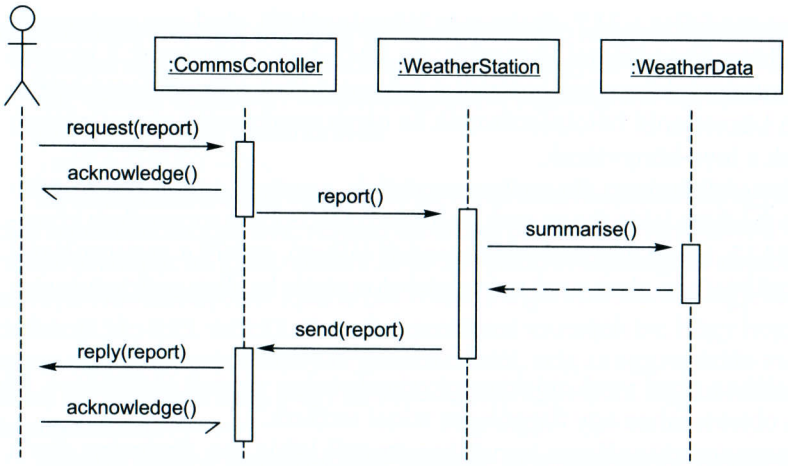
A *szekvenciamodellek* olyan dinamikus modellek, amelyek az interakció minden lehetséges módjára leírja, hogy ezek az interakciók milyen sorrendben következnek be. A 14.13. ábrán egy szekvenciamodell látható, amely a meteorológiai állomás adatgyűjtésében részt vevő műveleteket mutatja be. Egy szekvenciamodellben:

1. Az interakcióban részt vevő objektumok vízszintesen vannak elrendezve, és mindegyik objektumhoz egy függőleges vonal tartozik.
2. Az idő a szaggatott függőleges vonalakon fentről lefelé van ábrázolva, így a modellből könnyen kiolvasható a műveletek sorrendje.
3. Az objektumok közötti interakciókat a függőleges vonalakat összekötő feliratozott nyilak ábrázolják. Ezek *nem* adatáramlást, hanem az interakció számára alapvető üzeneteket és eseményeket jelölnek.
4. Az objektum életvonalán látható vékony téglalap azt az időszakot ábrázolja, amikor az adott objektum a rendszer vezérlőobjektuma. Az objektum a téglalap tetejéhez érve kapja meg a vezérlést, és a téglalap aljához érve átadja a vezérlést egy másik objektumnak.

A terv dokumentálása során minden lényeges interakcióra el kell készíteni a szekvenciamodellt. Ha készítettünk használati eset modellt, akkor minden egyes használati esethez is tartoznia kell egy szekvenciamodellnek.

A 14.13. ábrán a külső térképrendszer által a meteorológiai állomástól történő adatkérés esetén bekövetkező interakciók sorrendje látható. A szekvenciadiagramokat fentről lefelé olvassuk:

1. A CommsController osztály egyik példánya (:CommsController) környezetéből kap egy időjárás-jelentés elküldésére vonatkozó kérelmet és nyugtázza a kérést. A fél nyílhegy azt jelöli, hogy az üzenet küldője nem vár választ.
2. Ez az objektum egy üzenet küldésével felszólít egy WeatherStation objektumpéldányt az időjárás-jelentés elkészítésére. A CommsController példány ezek után felfüggeszti önmagát (vezérlési téglalapja véget ér). Az itt használt nyíl stílusa azt jelzi, hogy a CommsController és a WeatherStation objektumpéldányok konkurens módon is végrehajthatók.
3. A WeatherStation objektumpéldány küld egy üzenetet egy WeatherData objektumnak, amelyben kéri, hogy az összegezze a meteorológiai adatokat. Ez esetben az itt használt nyílhegyfajta azt jelzi, hogy a WeatherStation objektumpéldány válaszra vár.
4. A WeatherData objektum kiszámolja az összesítést, és a vezérlés visszakerül a WeatherStation objektumhoz. A szaggatott nyíl a vezérlés visszatértét jelöli.



14.13. ábra. Az adatgyűjtés műveleteinek sorrendje

5. Ez a WeatherStation objektum küld egy üzenetet a CommsController-nek, amelyben kéri az adatok távoli rendszerre történő átvitelét, majd az objektum felfüggeszti saját magát.
6. A CommsController objektum az összesített adatokat elküldi a távoli rendszernek, erről kap egy nyugtát, majd önmagát felfüggesztve várja a következő kérést.

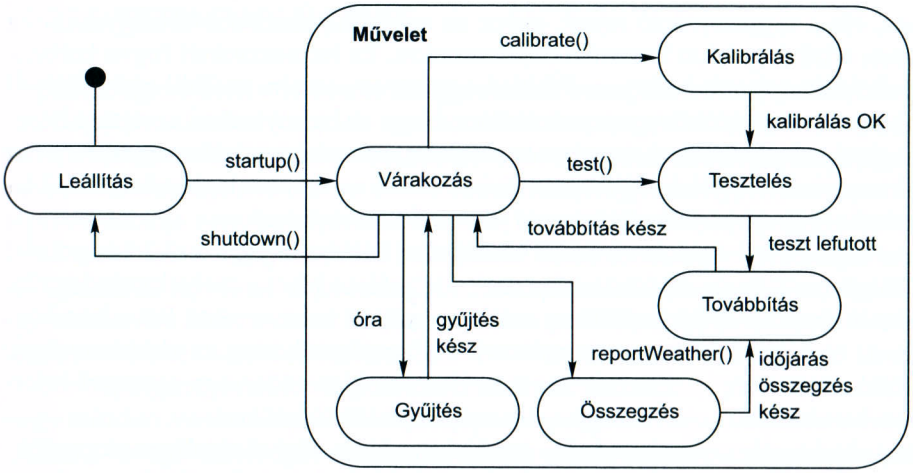
A szekvenciadiagramon az látható, hogy a CommsController és a WeatherStation objektumok valójában konkurens folyamatok, ahol a végrehajtás felfüggeszthető, majd újra folytatható. Lényegében a CommsController objektumpéldány figyeli és dekódolja a kívülről érkező üzeneteket, és kezdeményezi a meteorológiai állomás műveleteit.

A szekvenciadiagramokat objektumok egy csoportja együttes viselkedésének modellezésére használják, de szükségünk lehet arra, hogy összefoglaljuk az önálló objektumok általuk feldolgozható különböző üzenetekre válaszul adott viselkedését is. Ehhez egy állapotgépmodellt használhatunk, amely azt mutatja meg, hogy az objektumpéldány hogyan változik meg az általa kapott üzenetektől függően. Az UML az állapotgépmodellek leírására a Harel (Harel, 1987) által bevezetett állapotdiagramokat használja.

A 14.14. ábra a WeatherStation objektum állapotdiagramja, amely azt ábrázolja, hogy ez az objektum hogyan válaszol a különféle szolgáltatásokra vonatkozó kérésekre.

Ez a diagram a következőképpen olvasható:

1. Ha az objektum Leállítás állapotban van, akkor csak a startup() üzenetre reagál, ami egy olyan állapotba viszi át az objektumot, amelyben az további üzenetekre vár. A fekete pontból kiinduló címkézetlen nyíl azt jelöli, hogy ez a Leállítás állapot a kezdőállapot.



14.14. ábra. A WeatherStation állapotdiagramja

2. A Várakozás állapotban a rendszer további üzenetekre vár. Ha shutdown() üzenetet kap, akkor visszatér a Leállítás állapotba.
3. Ha egy reportWeather() üzenet érkezik, akkor a rendszer az Összegzés állapotba kerül, majd amikor az összegzés elkészül, átmegy a Továbbítás állapotba, amelyben az információ a CommsController segítségével adódik tovább. Ezt követően visszatér a Várakozás állapotba.
4. Egy calibrate() üzenet érkezése esetén a rendszer sorrendben Kalibrálás, Tesztelés, majd Továbbítás állapotba kerül, mielőtt visszakerülne a Várakozás állapotba. A test() üzenet a rendszert közvetlenül a Tesztelés állapotba viszi át.
5. Az órától érkező jel esetén a rendszer a Gyűjtés állapotba kerül, amelyben begyűjti az adatokat a műszerektől. A műszereket sorban utasítják az adatgyűjtésre.

Általában nem szükséges minden objektumhoz állapotdiagramot készíteni. Sok objektum annyira egyszerű, hogy az állapotgépmodell nem segítené elő az implementációt végző programozók számára az objektumok jobb megértését.

14.2.5. Objektuminterfész-specifikáció

Minden tervezési folyamatban lényeges a terv különböző komponensei közötti interfészek specifikációja. Azért kell interfészeket megadni, hogy az objektumok és egyéb komponensek tervezése párhuzamosan folyhasson. Ha egy interfész adott, akkor a többi objektum fejlesztői feltételezhetik, hogy az az interfész implementálásra is kerül.

Az interfész tervezése során nem szabad az interfész reprezentációjára vonatkozó információkkal foglalkoznunk, a reprezentációnak rejtettnek kell lennie, és műveleteket kell biztosítani az adatokhoz való hozzáférés és módosítás érde-

kében. Ha a reprezentáció rejtett, akkor az anélkül változtatható, hogy kihatna az ezen attribútumokat használó objektumokra. Ez természeténél fogva karbantarthatóbb tervet eredményez. Például egy verem esetén anélkül térhetünk át tömbreprezentációról listareprezentációra, hogy az befolyásolná a vermet használó objektumokat. Ezzel szemben a statikus tervezési modellben gyakran van értelme annak, hogy felfedjük az attribútumokat, mivel ez a legtömörebb módja annak, hogy az objektumok lényeges jellemzőit bemutassuk.

Az objektumok és az interfészek között nem feltétlenül egyszerű 1:1 kapcsolat van. Ugyanannak az objektumnak számos interfésze lehet, amelyek mindegyike más-más szempontból közelíti meg az általa nyújtott módszereket. Ezt a Java közvetlenül támogatja, hiszen az interfészek külön adhatók meg az objektumoktól, amelyek aztán „implementálják" az interfészeket. Hasonlóan egy egyszerű interfészen keresztül objektumok egy egész csoportját is el lehet érni.

Az objektuminterfész tervezése az objektumokra, illetve objektumcsoportokra adott interfészek részletezettségének megadásával foglalkozik. Ez az objektum, illetve objektumcsoport által biztosított szolgáltatások szignatúrájának és szemantikájának meghatározását jelenti. Az interfészeket UML-ben az osztálydiagramok jelölésrendszerével adhatjuk meg, azzal a különbséggel, hogy itt nincs attribútum rész, a név részben pedig használni kell az <interface> sztereotípust.

Egy általam jobban szeretett másik megközelítés szerint az interfészeket egy programozási nyelv segítségével adhatjuk meg. Ez a 14.15. ábrán látható, amely a meteorológiai állomás Javában írt interfész-specifikációját mutatja be. Az interfészek egyre bonyolultabbá válásával ez a megközelítés hatékonyabb, mivel a fordítóprogram szintaktikai ellenőrzője az interfész leírásában megbújó hibák és belső ellentmondások felfedezésére is használható. A Java-leírásból látható, hogy néhány módszer rendelkezhet különböző paraméterszámú változatokkal is. Épp ezért, a shutdown módszer paraméter nélküli hívás esetén leállítja az egész állomást, míg a paraméteres formát használva egyetlen műszer leállítására nyílik lehetőség.

```
interface WeatherStation {

    public void WeatherStation () ;

    public void startup () ;
    public void startup (Instrument i) ;

    public void shutdown () ;
    public void shutdown ( Instrument i ) ;

    public void reportWeather () ;

    public void test () ;
    public void test ( Instrument i ) ;

    public void calibrate ( Instrument i ) ;

    public int getID () ;
} //WeatherStation
```

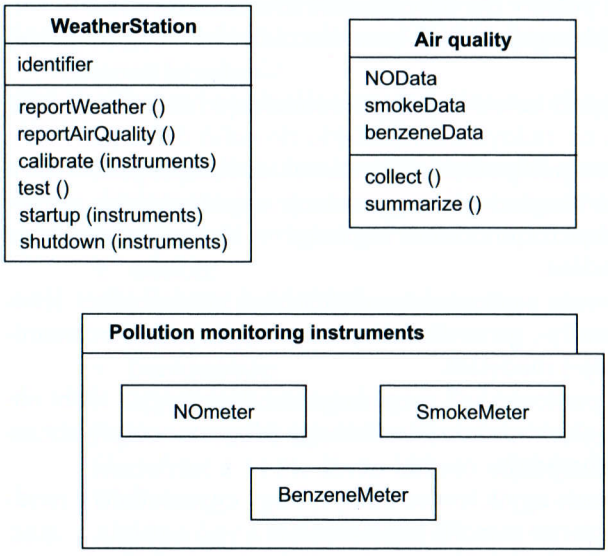
14.15. ábra. A meteorológiai állomás interfészleírása Javában

14.3. A TERV EVOLÚCIÓJA

Miután megszületett a rendszer kifejlesztésére vonatkozó döntés, elkerülhetetlen, hogy javaslatokat kapjunk arra vonatkozóan, hogy mit kellene a rendszerben változtatni. Az objektumorientált tervezés egyik fontos előnye, hogy egyszerűbben változtathatóvá teszi a tervet. Ennek oka, hogy az objektum állapotának reprezentációja nem befolyásolja a tervet. Nem valószínű, hogy egy objektum belső részleteinek megváltozása befolyással lenne a rendszer többi objektumára. Ráadásul, mivel az objektumok lazán kapcsolódnak egymáshoz, általában egyszerűen be lehet vezetni új objektumokat anélkül, hogy az jelentős hatással lenne a rendszer további részeire.

Az objektumorientált megközelítés erejének szemléltetésére tételezzük fel, hogy minden meteorológiai állomást képessé teszünk a légszennyezés mérésére. Ehhez a levegő minőségét mérő szerkezetek elhelyezése szükséges, amelyek a légkör különféle szennyeződéseinek nagyságát mérik. A leolvasott szennyezettségi adatok a meteorológiai adatokkal egyszerre továbbíthatók. A terv módosításához a következőket kell elvégezni:

1. A WeatherStation osztály részeként be kell vezetni egy Air quality objektumosztályt ugyanarra a szintre, ahol a WeatherData előfordul.
2. A WeatherStation osztályhoz hozzá kell adni egy reportAirQuality műveletet, amely a szennyezettségi információkat elküldi a központi számítógépnek. A meteorológiai állomást vezérlő szoftvert úgy kell módosítani, hogy az automatikusan összegyűjtse a szennyezettségre vonatkozó leolvasásokat, ha ezt egy csúcsszintű WeatherStation objektum igényli.



14.16. ábra. Új objektumok a szennyezettség figyelésének támogatásához

3. A rendszert ki kell bővíteni a szennyezettséget figyelő műszereket reprezentáló objektumokkal. Példánkban a nitrogénmonoxid, a füst és a benzol szintje mérhető.

A légszennyezettség-mérés objektumait egy Pollution monitoring instruments nevű önálló csomagba helyezzük. Ez kapcsolatban áll az Air quality-vel és a WeatherStation-nel, de nem kapcsolódik a meteorológiai adatokat gyűjtő objektumok egyikéhez sem. A 14.16. ábrán a WeatherStation, valamint a rendszerhez újonnan hozzáadott objektumok láthatók. A rendszer legfelső szintjétől (WeatherStation) eltekintve, a meteorológiai állomás eredeti objektumaiban semmilyen szoftveres változtatásra nincs szükség. A szennyezettségre vonatkozó adatok összegyűjtésének rendszerhez történő hozzáadása a meteorológiai adatok gyűjtését semmilyen módon nem befolyásolja.

Kulcsfogalmak

- Az objektumorientált tervezés a szoftvertervezés olyan eszköze, amelyben a terv alapvető komponensei saját állapottal és műveletekkel (nem pedig funkciókkal) rendelkező objektumokat reprezentálnak.
- Egy objektumnak rendelkeznie kell konstruktorral és megfigyelő műveletekkel, amelyek lehetővé teszik állapotának megfigyelését és megváltoztatását. Az objektum a többi objektum számára szolgáltatásokat (állapotinformációt felhasználó műveleteket) biztosít. Az objektumok az objektumosztály definíciójában megadott specifikáció felhasználásával futási időben jönnek létre.
- Az objektumok szekvenciális vagy konkurens módon implementálhatók. Egy konkurens objektum lehet passzív (ha állapota csak interfészén keresztül változik meg), vagy aktív objektum (ha külső beavatkozás nélkül meg tudja változtatni állapotát).
- Az UML az objektumorientált tervezés dokumentálásához használható jelölésrendszereket biztosít.
- Az objektumorientált tervezés folyamata a következő tevékenységekből áll: a rendszer architektúrájának megtervezése, a rendszer objektumainak azonosítása, a terv különböző objektummodellek segítségével történő leírása és az objektuminterfészek megadása.
- Az objektumorientált tervezés során számos különböző modell jöhet létre. Ezek lehetnek statikus (osztály-, generalizációs, asszociációs), illetve dinamikus (szekvencia-, állapotgép-) modellek.
- Az objektuminterfészeket pontosan kell megadni, mert azokat így a többi objektum is használhatja. Az objektuminterfészeket egy programozási nyelv segítségével (például Java) írhatjuk le.
- Az objektumorientált tervezés egyik fontos előnye, hogy egyszerűsíti a rendszer evolúcióját.

További irodalom

Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process, 2nd ed. Jó bevezetést nyújt az UML objektumorientált tervezési folyamatban történő használatáról. A tervezési mintákról szóló része a 18. fejezet témájába vág. (Larman, C., 2001, Prentice Hall.)

The Unified Modeling Language User Guide. Az UML-ről és annak objektumorientált tervek leírására szolgáló használról szóló alapkönyv. Két másik ezzel kapcsolatos könyv is létezik: az egyik az UML hivatkozási kézikönyv, a másik pedig egy objektumorientált fejlesztési folyamatot ír le. (Booch, G. *és mások,* 1999, Addison-Wesley.)

Az UML új verzióját (UML 2.0) 2003 közepén befejezték, de e sorok írásakor ezek a könyvek még nem frissültek ennek tükrében. Az új szabványról szóló részeket tartalmazó kiadások várhatóan 2004-ben jelennek meg.

A weben nagy mennyiségű bevezető anyag és segédlet lelhető fel az UML-lel kapcsolatban. A könyv weblapjai között megtalálható néhány ilyenre mutató hivatkozás.

Feladatok

14.1. Magyarázza el, hogy miért éppen a lazán kapcsolódó, a reprezentációjukra vonatkozó információt elrejtő objektumokon alapuló tervezési megközelítés alkalmazása eredményezhet könnyen módosítható tervet.

14.2. Példák segítségével magyarázza el, hogy mi a különbség az objektum és az objektumosztály között.

14.3. Mikor célszerű konkurens módon végrehajtódó objektumokból felépülő tervet készíteni?

14.4. Az UML objektumosztályok jelölésére szolgáló eszközeivel tervezze meg az alább felsorolt objektumosztályokat, az attribútumok és a műveletek azonosításával. Annak eldöntéséhez, hogy milyen attribútumokat és műveleteket kapcsoljon az egyes objektumokhoz, támaszkodjon saját tapasztalataira:

- telefon;
- személyi számítógép nyomtatója;
- hifitorony;
- bankszámla;
- könyvtári katalógus.

14.5. Fejlessze tovább (részletezze) a meteorológiai állomás tervét: adjon interfészleírást a 14.11. ábrán látható objektumokhoz Javában, C++-ban vagy UML-ben.

14.6. Alakítsa úgy a meteorológiai állomás tervét, hogy az megmutassa az adatgyűjtő alrendszer és a meteorológiai adatok összegyűjtését végző műszerek közötti interakciókat. Ábrázolja ezeket szekvenciadiagramon.

14.7. Azonosítsa a következő rendszerekben a lehetséges objektumokat, és készítse el ezen rendszerek objektumorientált tervét. A terv elkészítésekor éljen ésszerű feltételezésekkel.

- Egy csoportos határidőnapló-kezelő rendszer munkatársak megbeszéléseinek és találkozóinak időzítésére szolgál. Ha több embert érintő találkozót kell tartani, akkor a rendszer megtalálja az érintettek határidőnaplóiban azt az időpontot, amikor minden érintett ráér, és erre az időpontra szervezi a találkozót. Ha ilyen időpont nincs, akkor jelzi a felhasználóknak, hogy át kell szervezniük időbeosztásukat, hogy helyet csináljanak a találkozónak.
- Egy benzinkút teljesen automatizált módon működik. A sofőrök lehúzzák a hitelkártyájukat a kúthoz csatlakoztatott leolvasón, a kártya a hitelintézet számítógépével történő kommunikáció során ellenőrzésre kerül, majd megállapításra kerül, hogy legfeljebb mennyi üzemanyagot szabad tankolni. A sofőr ezután a szükséges üzemanyag-mennyiséget a tankba tölti, majd visszahelyezi a helyére az üzemanyagpisztolyt. Ekkor a sofőr hitelkártyájára ráterhelődik a tankolt üzemanyag ára, majd pedig a hitelkártya visszakerül tulajdonosához. Ha a kártya érvénytelen, akkor azt a kút még az üzemanyag kiadása előtt visszaadja.

14.8. Adja meg a 14.7. feladatban definiált objektumok pontos interfész-definícióját Javában vagy C++-ban.

14.9. Készítsen szekvenciadiagramot, amely a csoportos határidőnapló-kezelő rendszer objektumai közötti kölcsönhatást ábrázolja, amikor néhány ember megbeszélést szervez.

14.10. Készítsen állapotdiagramot, amely egy vagy több – a 14.7. feladatban ön által megadott – objektum lehetséges állapotváltozásait mutatja be.