

14.

Objektumorientált tervezés

Kérdések

- Hogyan lehet a szoftvert egymással kapcsolatban lévő, önálló működésű, saját állapottal rendelkező objektumok halmazaként leírni?
- Mik az objektum-orientált tervezés lépései?
- Milyen típusú modellek használhatók egy objektum-orientált terv leírására?
- Hogyan használható az UML ezen modellek reprezentálására?

Tartalom

1. Bevezetés
2. Objektumok és objektum-osztályok
3. Az objektumorientált tervezés folyamata
4. A terv evolúciója

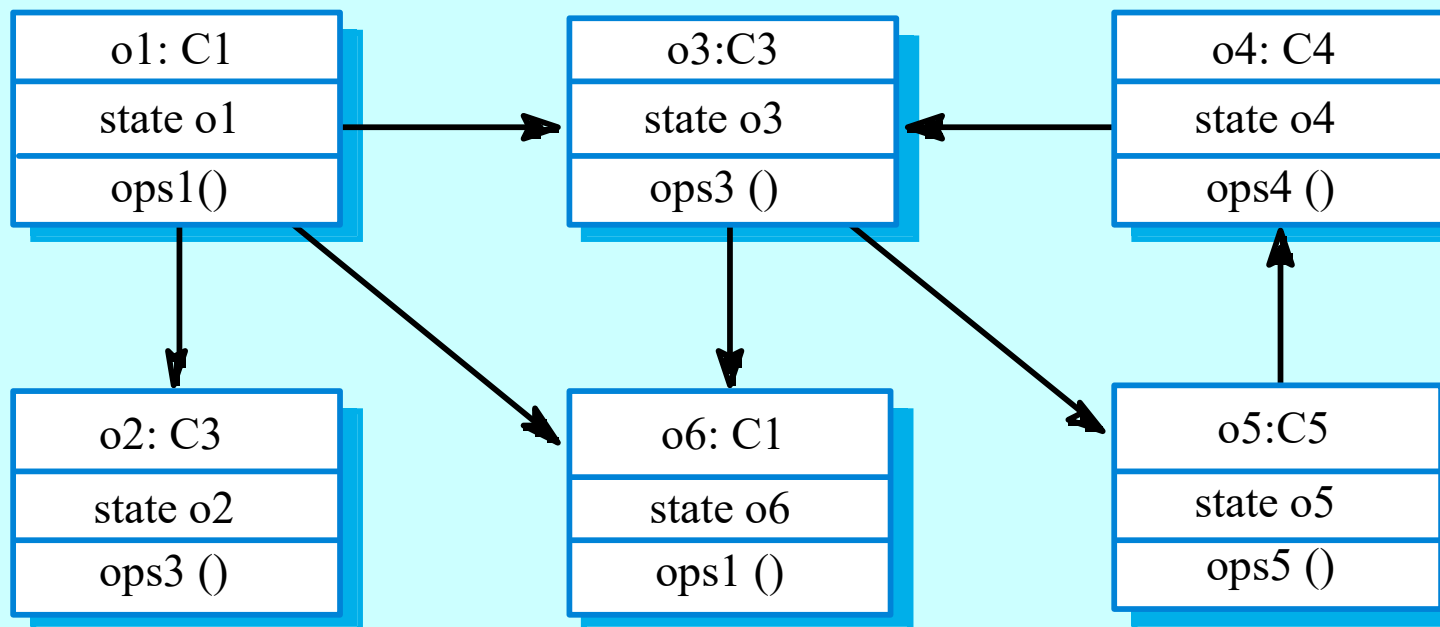
1. Bevezetés

- Az objektumorientált Analízis (OOA), Tervezés (OOT) és Programozás (OOP) egymással kapcsolatban állnak, de különböző fogalmak
- Az **OOA** a felhasználói környezet modelljének kidolgozásával foglalkozik.
- Az **OOT** a követelményeket kielégítő rendszer modelljének kidolgozásával foglalkozik.
- Az **OOP** az OOT realizálásával foglalkozik egy OO nyelv (pl. Java, C++) segítségével.

Az OOT jellemzői

- Az objektumok a való világ entitásainak reprezentációi, amelyek önmagukat menedzselik.
- Az objektumok önállóak és saját, a külvilág számára közvetlenül nem látható állapottal rendelkeznek.
- A rendszer funkcionalitását objektumok szolgáltatásaiként reprezentáljuk.
- Közös adatterületek nem léteznek. Az objektumok üzenetekkel kommunikálnak.
- Az objektumok lehetnek elosztottak, végrehajtásuk lehet szekvenciális vagy párhuzamos.

Objektumok interakciója



Az OOT előnyei

- Könnyű kezelhetőség. Az objektumok önálló entitásokként foghatók fel.
- Az objektumok potenciálisan újrafelhasználható komponensek.
- Sok rendszerben a való világ entitásai könnyen és értelemszerűen képezhetők le a rendszer objektumaira.

2. Objektumok és objektum-osztályok

- A objektumok a szoftver rendszer entitásai, amelyek a való világ és a rendszer entitásait reprezentálják.
- Az objektum-osztályok objektumok sablonjai. Belőlük objektumok hozhatók létre.
- Az objektum-osztályok más objektum-osztályoktól attribútumokat és szolgáltatásokat örökölhetnek.

Objektumok és objektum-osztályok

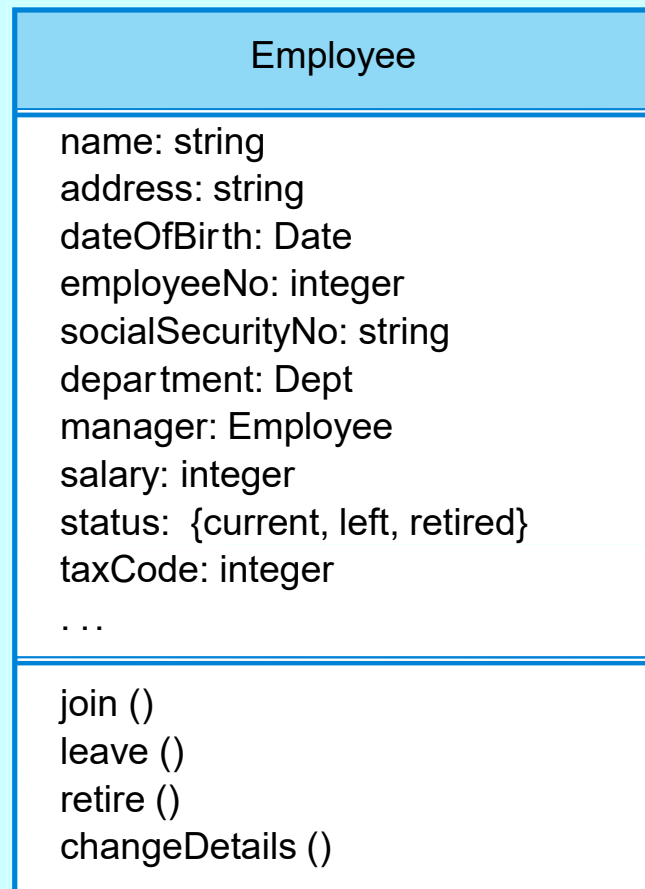
Egy objektum olyan entitás, amelynek van állapota és egy meghatározott operáció-készlete ezen állapot felett. Az állapotot az objektum attribútum-halmaza reprezentálja. Az objektumhoz tartozó operációk más objektumok (kliensek) számára nyújtanak szolgáltatásokat, melyeket azok valamilyen számítási igény felmerülése esetén kérhetnek.

Az objektumokat valamely objektum-osztály definíciója alapján hozzuk létre. Az objektum osztály definíciója az objektumok számára „sablon”-ként szolgál. Tartalmazza az adott objektum-osztályhoz tartozó összes attribútum és szolgáltatás deklarációját.

Az UML (Unified Modeling Language)

- Az 1980-as és '90-es években számos jelölés-rendszert javasoltak az OO tervek leírására.
- Az UML ezek egyesítéseként született.
- Az OO analízis és tervezés során használt számos modell leírási módját tartalmazza.
- Manapság ez az OO modellezés *de facto* szabványa.

Az *alkalmazott* objektum-osztálya (UML)



Az objektumok kommunikációja

- Elvileg az objektumok üzeneteken keresztül kommunikálnak.
- Üzenetek
 - A hívó objektum által kért szolgáltatás neve;
 - A szolgáltatás végrehajtásához szükséges információ másolata, valamint az eredmény tárolójának neve.
- A gyakorlatban az üzeneteket gyakran eljárás-hívással implementáljuk:
 - Név = eljárás neve;
 - Információ = paraméter-lista.

Példák üzenetekre

// Egy puffer-objektumhoz tartozó
// hívás, amely a puffer következő
// elemét adja vissza

`v = circularBuffer.Get () ;`

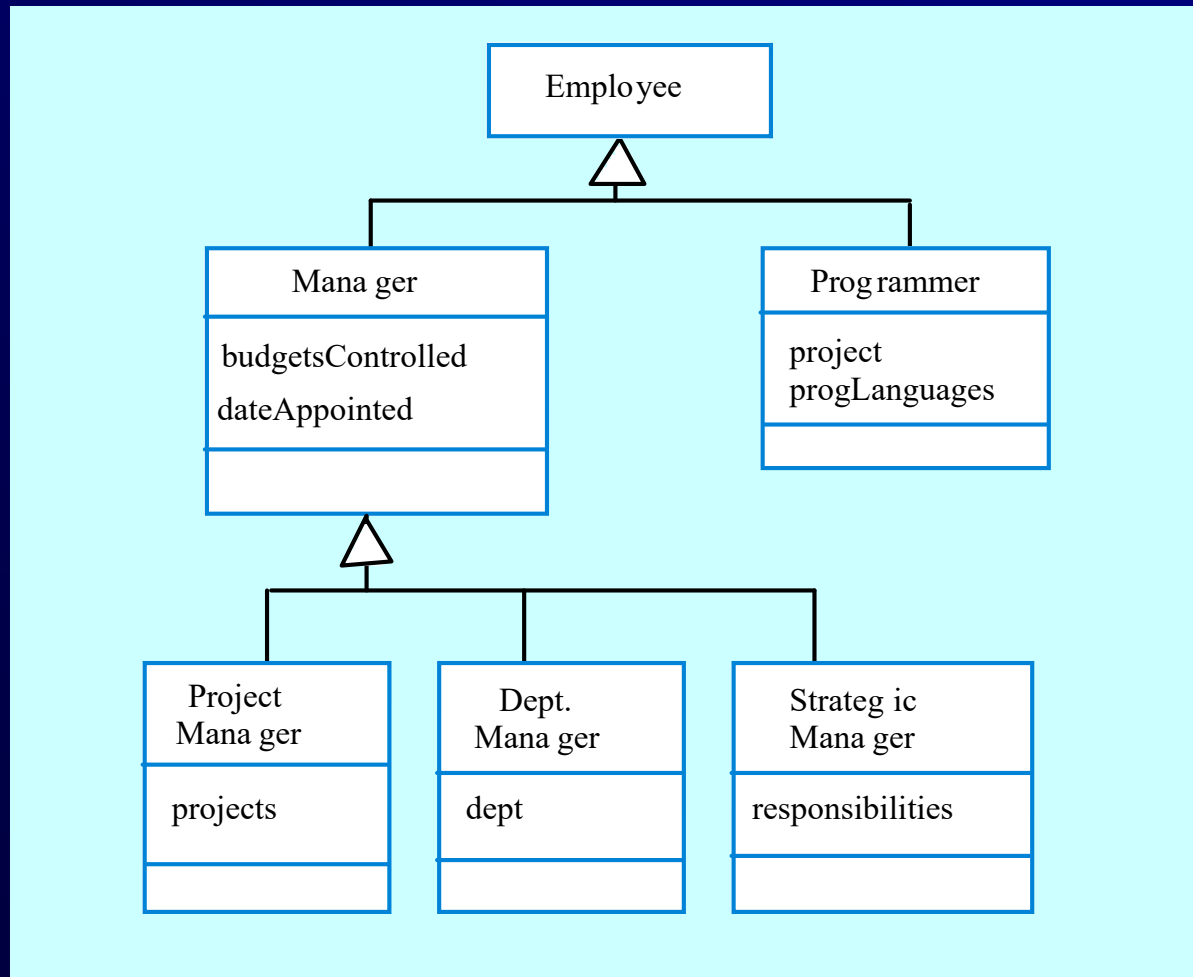
// Egy termosztát-objektumhoz tartozó
// hívás, amely beállítja a tartani kívánt
// hőmérséklet értékét

`thermostat.setTemp (20) ;`

Általánosítás és öröklés

- Az objektumok azon osztályok tagjai, amelyek az attribútumait és az operációit definiálják.
- Az osztályok egy osztály-hierarchiába szervezhetők, ahol egy osztály (szuper-osztály) egy vagy több osztály (al-osztály) általánosítása lehet.
- Az al-osztály örökli a szuper-osztály attribútumait és operációit, valamint saját metódusokat és attribútumokat adhat ezekhez.
- Az UML-beli általánosítást az OO nyelvek öröklésként implementálják.

Az általánosítás-hierarchia



Az öröklés előnyei

- Egy absztrakciós mechanizmus, amely entitások osztályozására használható.
- Egy újrafelhasználási mechanizmus, amely a tervezés és programozás szintjén is használható.
- Az öröklési gráf alkalmazási környezetek és rendszerek szerveződéséről szolgáltat információt.

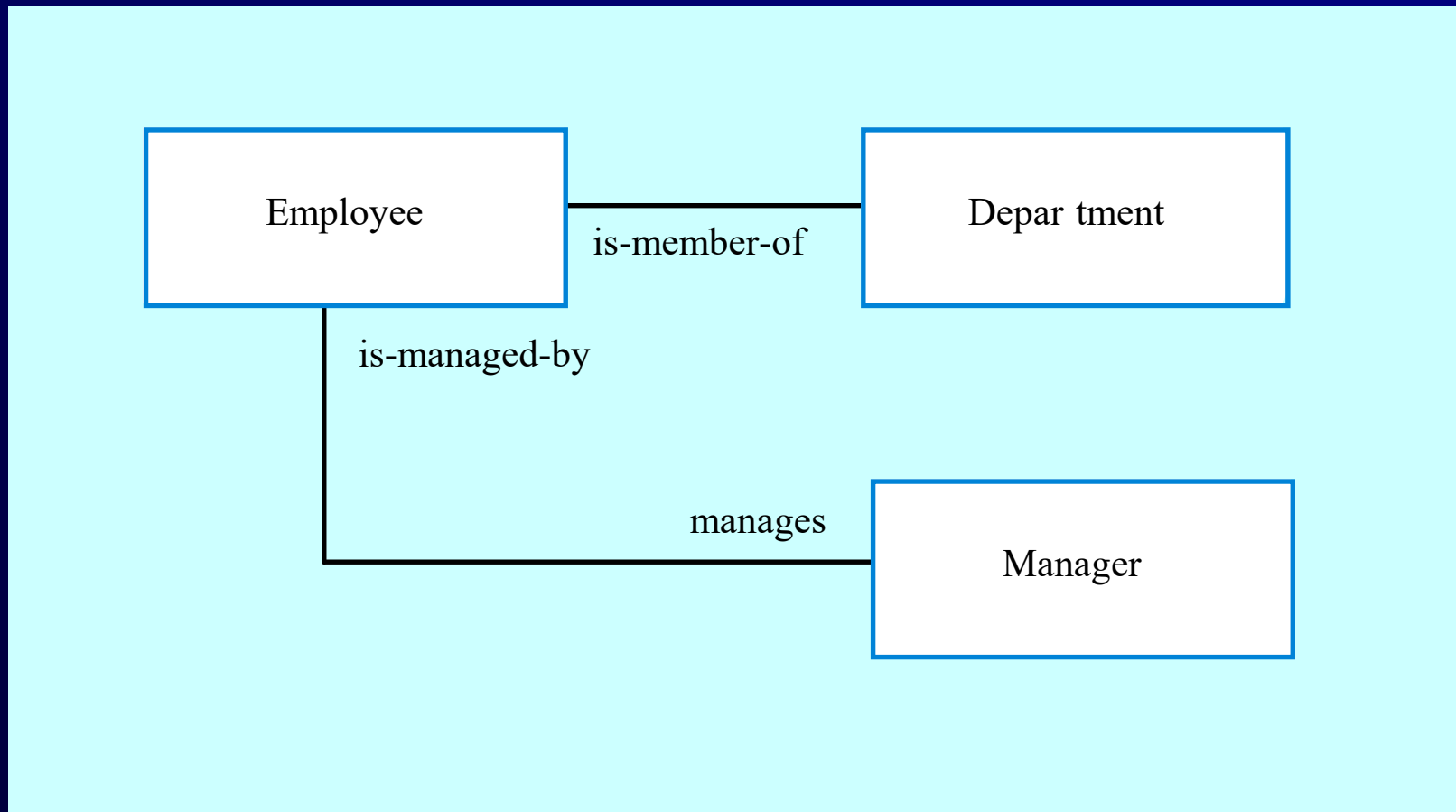
Az öröklés problémái

- Az objektum-osztályok nem „önjáróak”. A megértésükhöz szükséges a szuper-osztályuk ismerete is.
- A tervezők gyakran újrahasznosítják az analízis során készített öröklési gráfot. Ez a hatékonyság kárára válhat.
- Az analízis, tervezés és implementáció során használt öröklési gráfok célja más és más, ezeket egymástól függetlenül kell kezelni.

Az UML asszociációi

- Az objektumok és objektum-osztályok más objektumokkal és objektum-osztályokkal lehetnek kapcsolatban.
- A UML-ben az általánosított kapcsolatot az asszociációval jelezzük.
- Az asszociációkon külön szöveges információ írhatja le az asszociáció jellegét.

Példa: egy asszociáció-modell



Konkurens objektumok

- Az objektumok önálló entitások, így alkalmasok párhuzamos implementációra.
- Az objektum-kommunikáció üzenet-modelljét közvetlenül lehet implementálni, ha az objektumok egy elosztott rendszerben, különböző processzorokon futnak.

Szerverek és aktív objektumok

- Szerverek
 - Az objektumot párhuzamos folyamatként (szerver) implementáljuk, ahol az objektum operációi belépési pontok lesznek. Ha nincs hívás az objektumra, akkor az felfüggeszti magát és várakozik további szolgáltatás-hívásokra.
- Aktív objektumok
 - Az objektumot párhuzamos folyamatként implementáljuk. A belső állapotokat az objektum maga is megváltoztathatja, nem kell hozzá külső hívás.

Példa: aktív *transponder* objektum

- Az aktív objektumok attribútumait operációkkal is megváltoztathatjuk, de autonóm módon, belső műveletekkel ezt maguk is megtehetik.
- Egy **transponder** objektum egy repülőgép pozíció adatait szolgáltatja. A pozíciót egy műholdas rendszer segítségével határozza meg: az objektum periodikusan változtatja a *pozíció* attribútumát a műholdas háromszögelés segítségével.

Példa: aktív *transponder* objektum

```
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
```

Java szálak

- A Jáva szálai (thread) egyszerű módszert adnak konkurens objektumok implementálására.
- A szálnak tartalmazni kell egy `run()` metódust, amit a Java futtató rendszere indít el.
- Az aktív objektumok tipikusan egy végtelen ciklust tartalmaznak.

3. Az objektumorientált tervezés folyamata

- Strukturált tervezési módszer, melynek során több, különböző rendszermodell kerül kifejlesztésre.
- Ezen modellek fejlesztése és karbantartása nagy erőfeszítést igényel, ami kis rendszerek esetén nem kifizetődő.
- Nagy rendszerek esetén, amelyeket több csoport fejleszt, a tervezési modellek alapvető kommunikációs mechanizmust biztosítanak.

A folyamat elemei

- A legfontosabb tevékenységek:
 - A rendszer kontextusának és felhasználási módozatainak definiálása;
 - A rendszerarchitektúra tervezése;
 - Az alapvető rendszerobjektumok meghatározása;
 - A tervezési modellek kidolgozása;
 - Az objektum interfészek specifikálása.

Példa: Meteorológiai rendszer

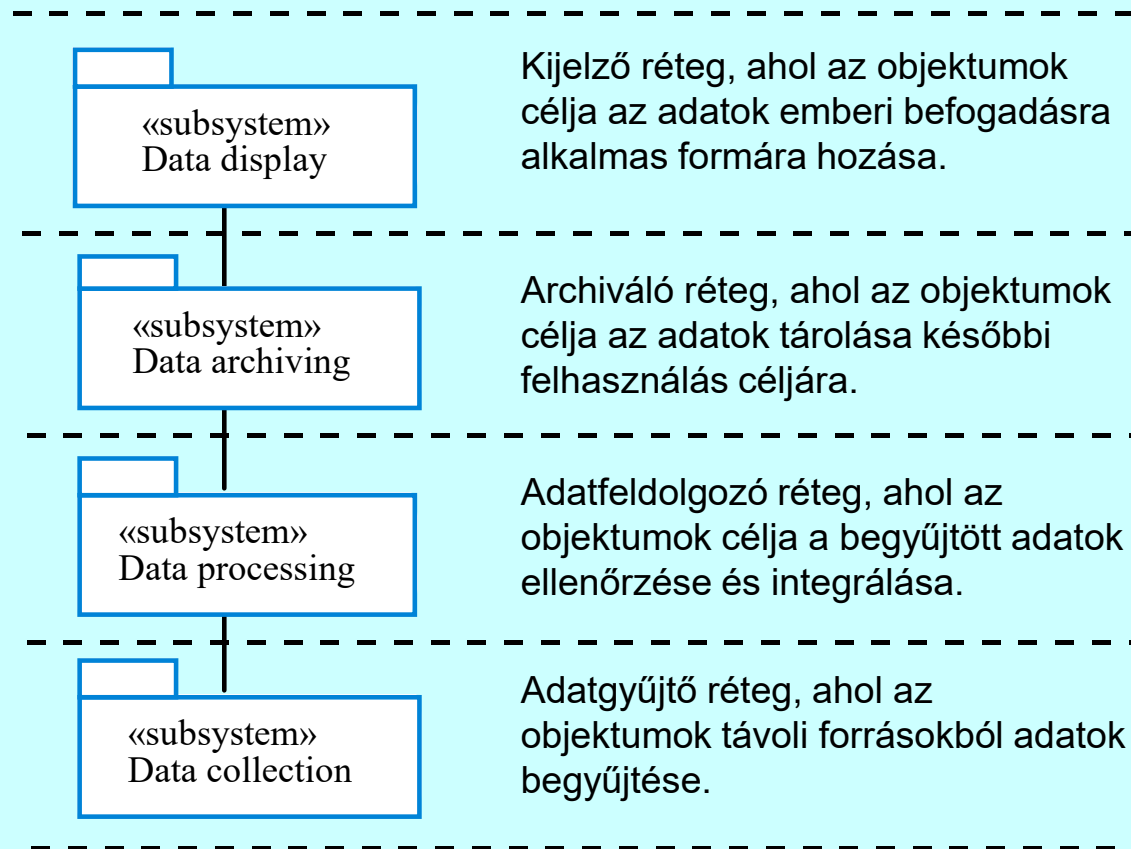
A **meteorológiai térképgeneráló rendszer** meteorológiai térképeket generál távoli meteorológiai állomások és más források (pl. megfigyelők, léggömbök, műholdak) adatainak felhasználásával. A meteorológiai állomások adataikat a körzeti számítógépekhez továbbítják azok kérésére.

A körzeti számítógépes rendszer validálja az összegyűjtött adatokat és összesíti más forrásokból származó adatokkal. Az integrált adatokat archiválják. Az archívum adatainak és egy digitalizált térkép adatbázisnak felhasználásával helyi időjárási térképeket generálnak. A térképek speciális térkép-nyomtatók felhasználásával kinyomtathatók, vagy különféle formátumokban kijelezhetők.

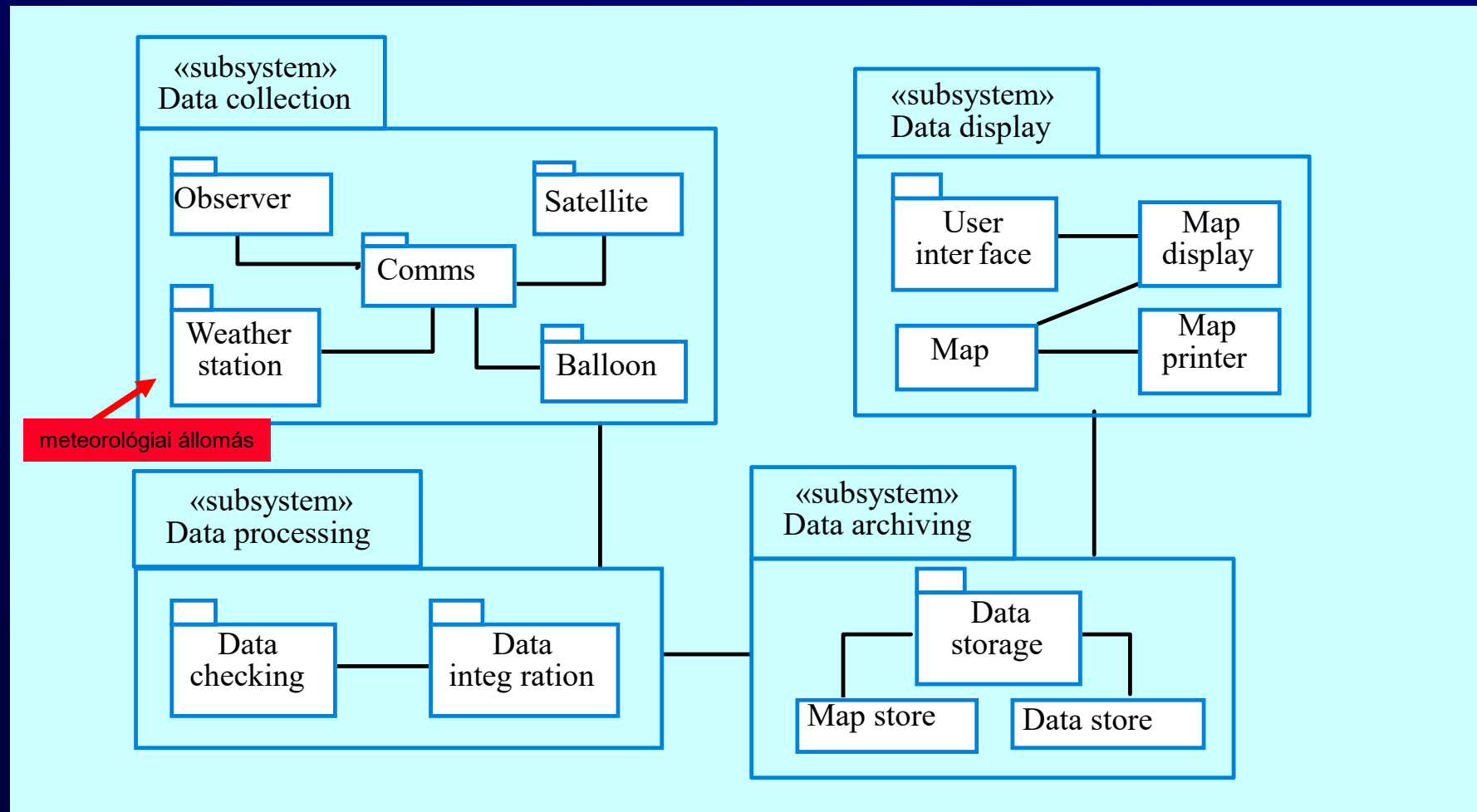
3.1 A rendszer kontextusa és felhasználási módozatai

- A fejlesztendő szoftver és külső környezete közti kapcsolatok feltérképezése
- A rendszer kontextusa
 - Statikus modell, amely leírja a környezetben levő más rendszereket. Alrendszer (*subsystem*) modellek használhatók más rendszerek jelzésére. A következő ábra a meteorológiai állomás (*weather station*) körüli rendszereket mutatja.
- A rendszerhasználat modellje
 - Dinamikus modell, amely a rendszer és környezetének interakcióját mutatja be. *Use-case* modellek használhatók az interakciók leírására.

Réteges szerkezet



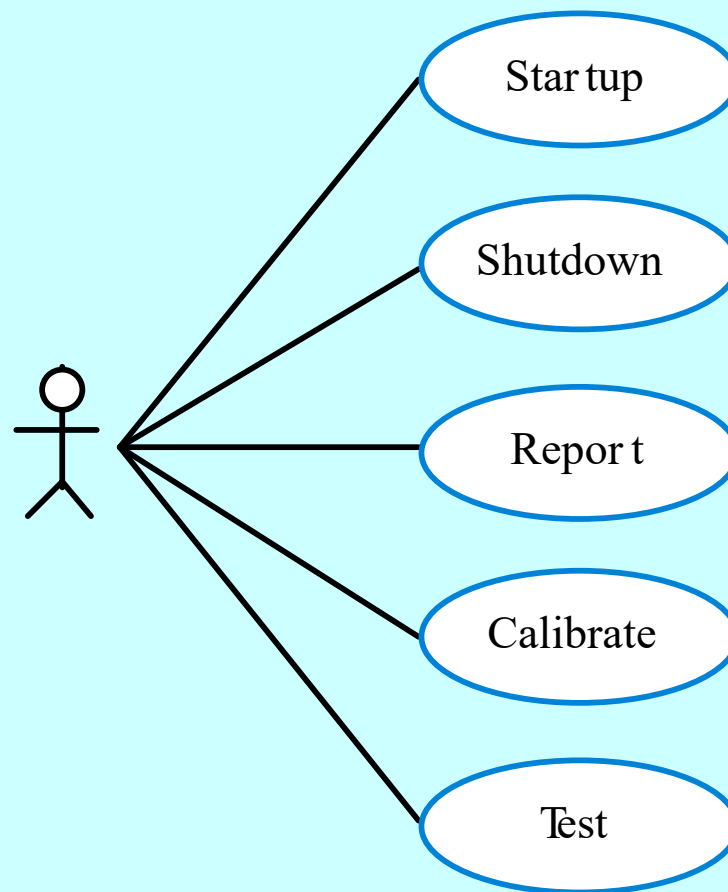
A meteorológiai térképgeneráló rendszer alrendszerei



Use-case modellek

- *Use-case* modellek használatával a rendszer valamennyi interakciója leírandó.
- A *use-case* modellek a rendszer szolgáltatásait ellipszisek segítségével jelölik. Az interakcióban résztvevő entitást pálcikaember jelzi.

A meteorológiai állomás *use-case*-ei



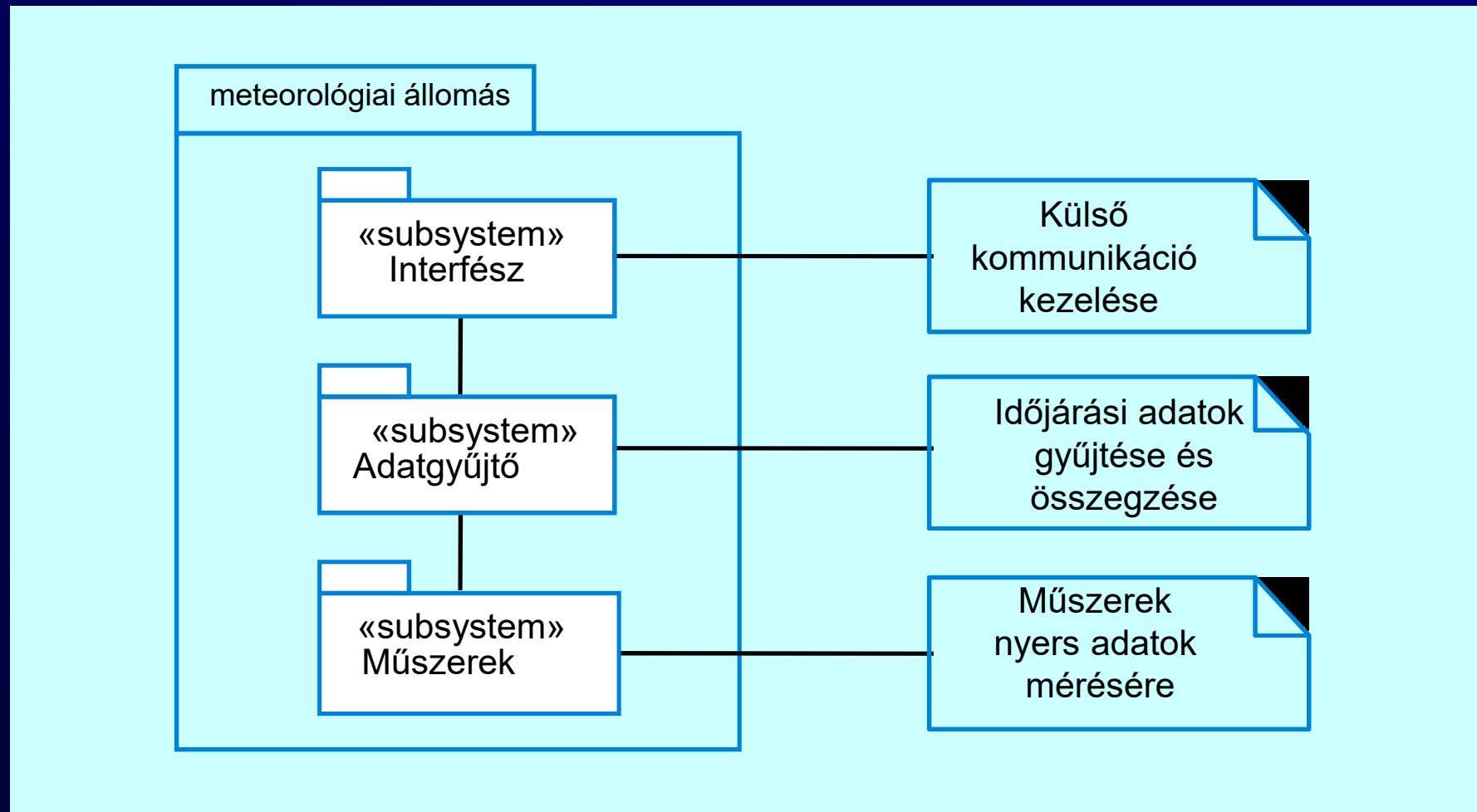
Egy *use-case* leírása

System	Meteorológiai állomás
Use-case	Adatküldés (Report)
Actors	Időjárás adatgyűjtő rendszer (<i>data collection</i>), Meteorológiai állomás (<i>weather station</i>)
Data	A meteorológiai állomás elküldi a műszerek által az előző adatgyűjtési periódus alatt szerzett adatok összesítését az időjárás adatgyűjtő rendszernek. Az elküldött adatok a következők: maximum, minimum és átlagos hőmérséklet, maximum, minimum és átlagos légnyomás, maximum, minimum és átlagos szélsébség, teljes csapadékmennyiség, valamint a szélirány 5 perces intervallumokban mintavételezve.
Stimulus	Az időjárás adatgyűjtő rendszer modemes kapcsolatot hoz létre a meteorológiai állomással és kéri az adatok elküldését.
Response	Az összegzett adatok elküldése az időjárás adatgyűjtő rendszerbe.
Comments	A meteorológiai állomásokat általában óránként egyszer kérdezik le, de ez a gyakoriság állomásról állomásra más és más érték lehet és a jövőben változhat.

3.2 Az architektúra tervezése

- A rendszer és környezete közötti interakciók megértése után ez az információ felhasználható a rendszer-architektúra tervezésére.
- A meteorológiai állomás céljára egy réteges szerkezet alkalmas :
 - *Interfész réteg*: kommunikáció kezelése;
 - *Adatgyűjtő réteg*: a műszerek kezelése;
 - *Műszerek réteg*: adatgyűjtés.
- Egy architektúra-modell ne tartalmazzon több, mint 7 entitást.

A meteorológiai állomás architektúrája



3.3 Objektumok azonosítása

- Az objektumok (vagy inkább objektum-osztályok) identifikációja, azonosítása az OO tervezés legnehezebb lépése.
- Az objektumok identifikálására nincs „mágikus módszer”. A rendszertervező tapasztalatától, gyakorlatától, valamint az alkalmazási környezetről szerzett tudásától függ.
- Az objektum-identifikáció iteratív eljárás. Nagyon valószínűtlen, hogy elsőre sikerül.

Az identifikáció lehetséges módszerei

- A rendszer természetes nyelvi leírásán végzett nyelvi elemzés (A HOOD OOT módszerben használt)
- Az alkalmazási környezetbeli kézzelfogható dolgok alapján.
- Viselkedési megközelítés: mely objektumok mely viselkedésben vesznek részt.
- Szenárió-alapú analízis: minden szenárióban azonosítjuk az objektumokat, attribútumokat és a metódusokat.

A meteorológiai állomás leírása

A **meteorológiai állomás** szoftver-vezérelt műszerek együttese, amely adatokat gyűjt, adatfeldolgozást végez, valamint továbbítja ezen adatokat további feldolgozás céljából. A műszerek között megtalálhatók hőmérők, szélirány- és szélesebségmérő, barométer és csapadékmérő. Az adatok gyűjtése periodikusan történik.

Amikor adatok továbbítására kérés érkezik, a meteorológiai állomás feldolgozza és összegzi az összegyűjtött adatokat. Az összegzett időjárási adatok a térképező számítógépbe továbbítódnak, amikor erre kérés érkezik.

A meteorológiai állomás objektum osztályai (nem teljes lista)

- Hőmérő, szélmérő, barométer
 - Az alkalmazási környezet objektumai, kézzel fogható dolgok. „Hardver” objektumok, amelyek a rendszerben található műszerekhez kapcsolódnak.
- Meteorológiai állomás
 - A meteorológiai állomás alapvető interfésze a környezete felé. A use-case-ekben identifikált interakciókat tükrözi.
- Időjárási adatok
 - A műszerekből gyűjtött összesített adatok.

A meteorológiai állomás objektum osztályai

WeatherStation
identifier
reportWeather () calibrate (instruments) test () startup (instruments) shutdown (instruments)

WeatherData
airTemperatures groundTemperatures windSpeeds windDirections pressures rainfall
collect () summarise ()

Ground thermometer
temperature
test () calibrate ()

Anemometer
windSpeed windDirection
test ()

Barometer
pressure height
test () calibrate ()

További objektumok és objektumok finomítása

- Az alkalmazási környezetről gyűjtött információk alapján további objektumok és operációk azonosítása
 - A meteorológiai állomásoknak egyéni azonosító kell;
 - A meteorológiai állomások távoli helyeken üzemelhetnek, így a műszerek meghibásodását automatikusan jelenteni kell. Az önellenőrzéshez további attribútumok és operációk kellenek.
- Aktív vagy passzív objektumok
 - Ebben az esetben az objektumok passzívak, az adatgyűjtést kérésre, és nem autonóm módon végzik. Előny: rugalmas rendszer; hátrány: időzítések azonosak.

3.4 Tervezési modellek

- A tervezési modellek az objektumokat, objektum-osztályokat, valamint ezen entitások közötti kapcsolatokat írják le.
- A statikus modellek a rendszer statikus struktúráját írják le objektum-osztályok és relációik segítségével.
- A dinamikus modellek az objektumok közötti dinamikus interakciókat írják le.

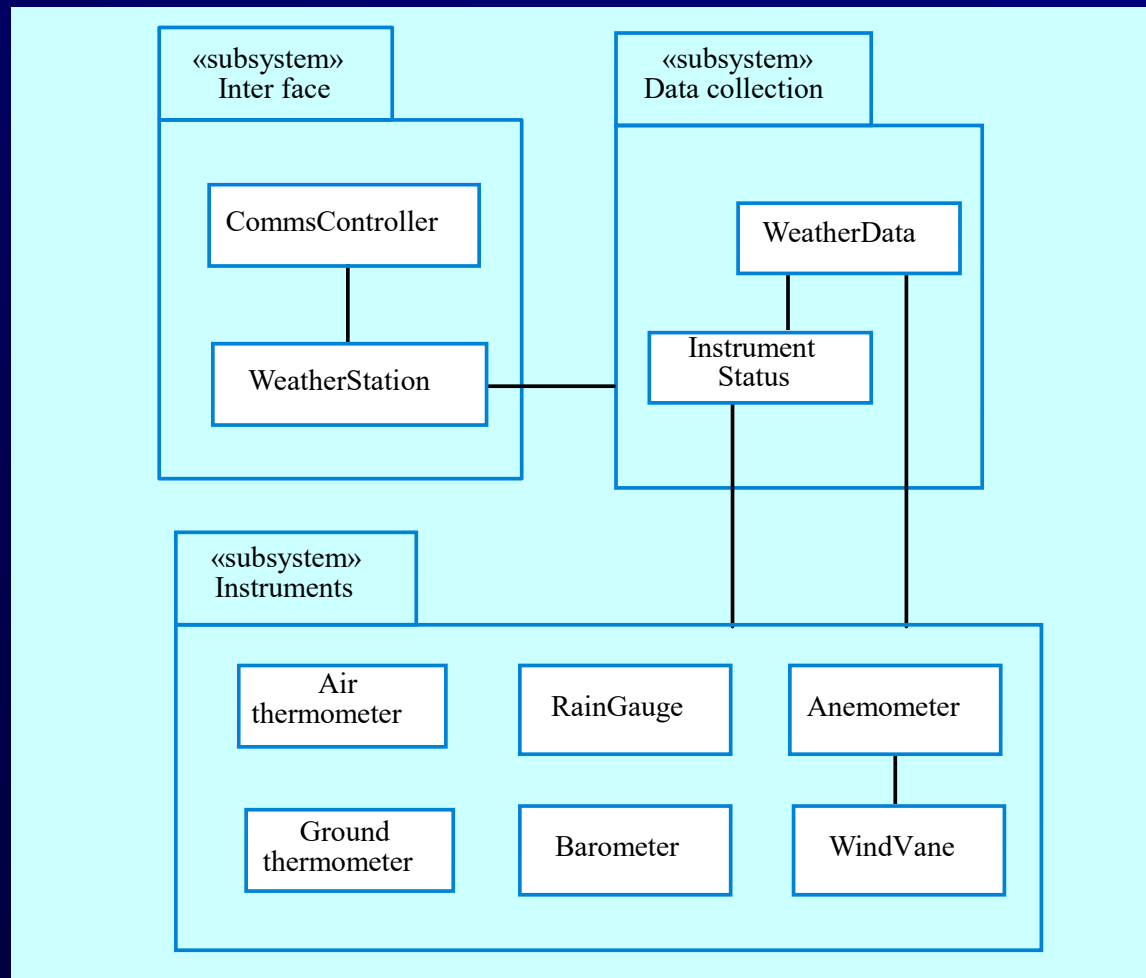
Példák tervezési modellekre

- Alrendszer (*sub-system*) modellek, amelyek az objektumok koherens alrendszerekre való logikus csoportosítását adják.
- A szekvencia-diagramok az objektumok közötti interakciók sorozatát írják le.
- Az állapotgép-modellek azt írják le, hogy az egyes objektumok hogyan változtatják állapotukat eseményekre reagálva.
- Egyéb modellek: use-case modellek, aggregációs modellek, általánosítási modellek, stb...

Alrendszer modellek

- A terv logikusan kapcsolódó csoportokba való szervezését mutatja be.
- Az UML-ben ezeket csomagok (package) formájában használjuk. Ez egy logikai modell, a rendszerben az objektumok valódi csoportosítása ettől különbözhet.

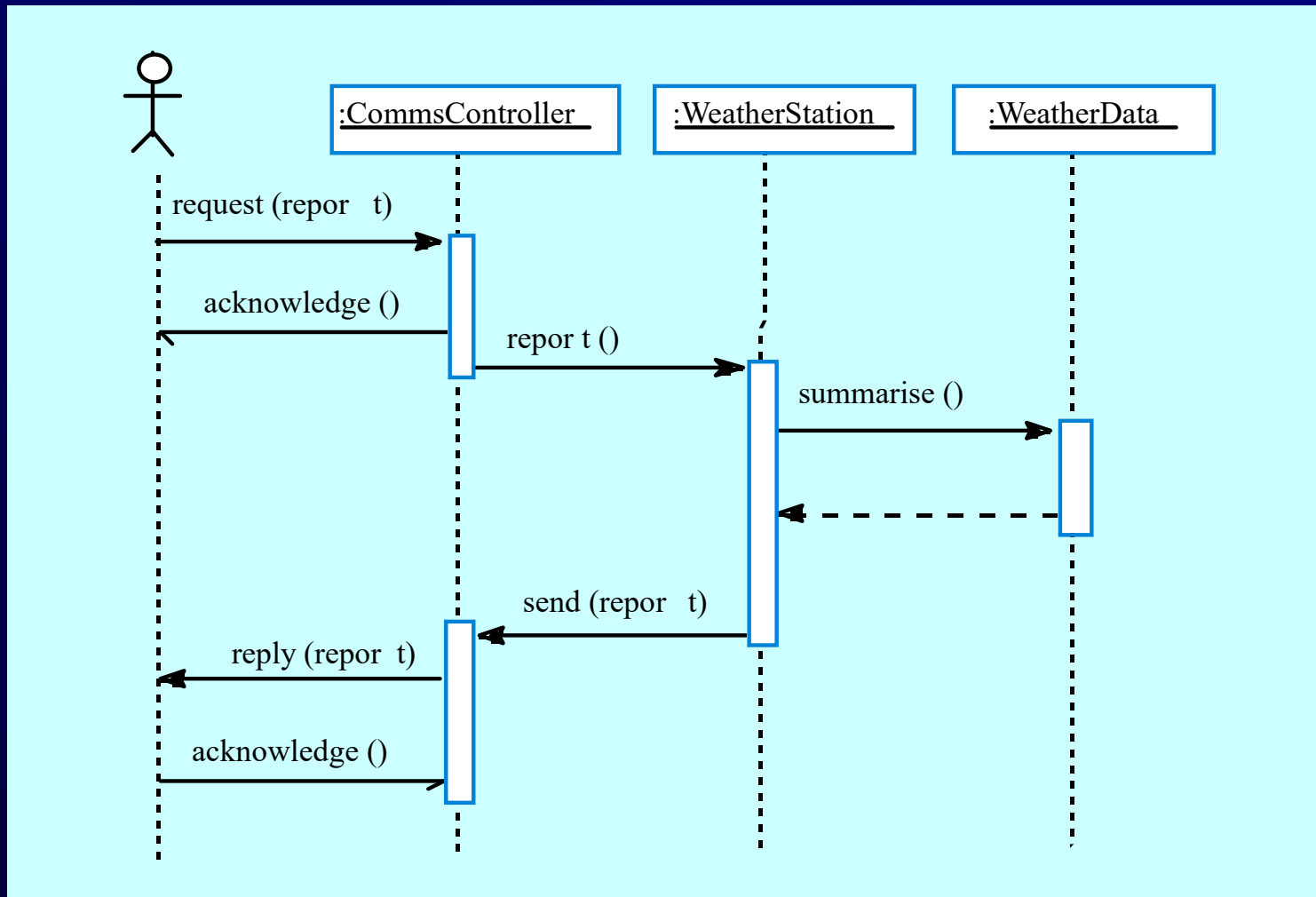
A meteorológiai állomás alrendszerei



Szekvencia-modellek

- Az objektum-interakciók sorozatát mutatják
 - Az objektumok az ábra tetején, vízszintesen egymás mellett vannak elhelyezve;
 - Az idő függőlegesen, fentről lefelé telik;
 - Az interakciókat (szöveges) nyilak jelzik, különböző formájú nyilak különböző típusú interakciókat reprezentálnak;
 - Az objektum vonalán egy vékony téglalap jelzi azt az időtartamot, amíg az objektum a rendszert vezérlő objektum.

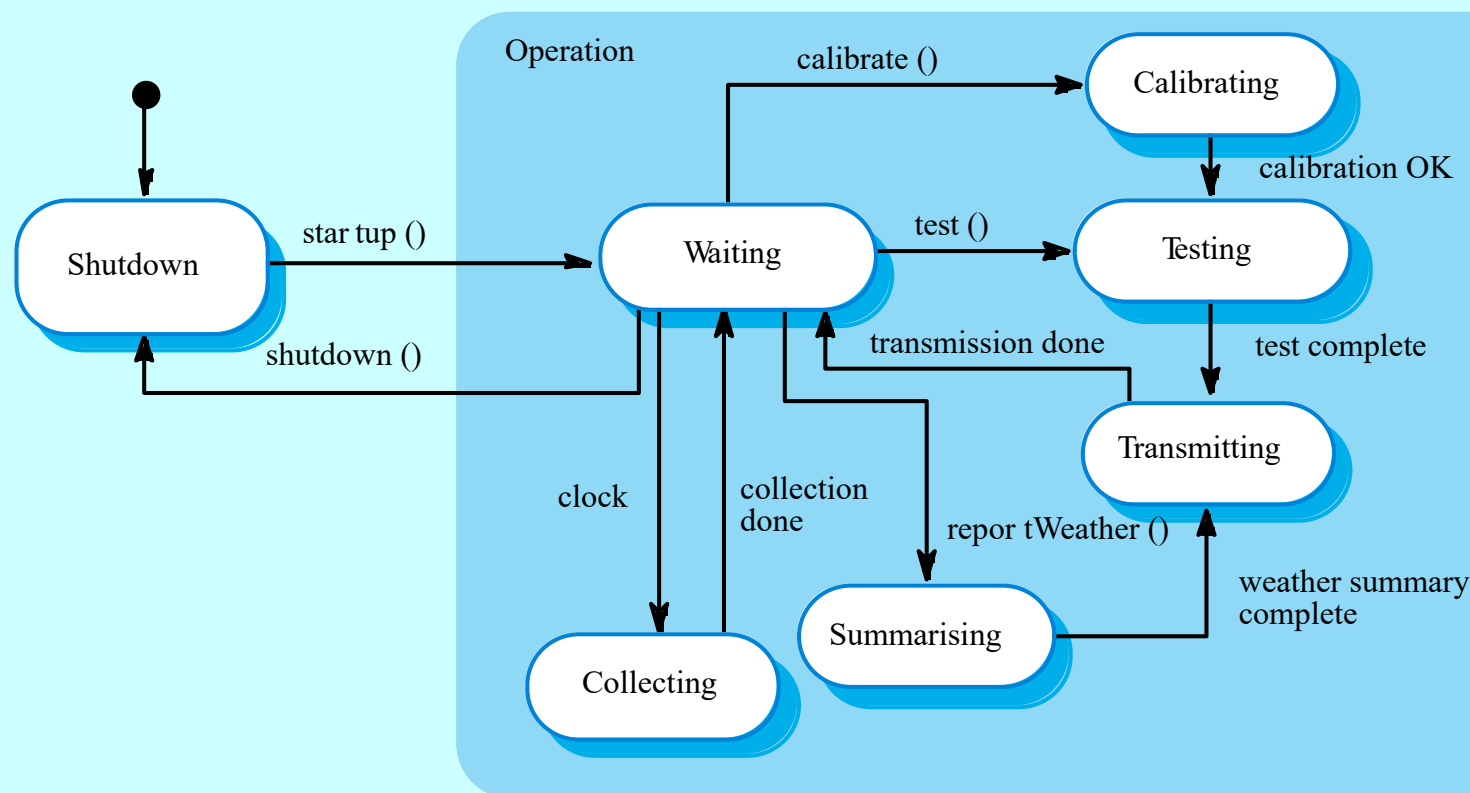
Az adatgyűjtés-szekvencia



Állapotdiagramok (*statechart*)

- Hogyan reagál az objektum különböző szolgáltatás-kérésekre, valamint milyen állapot-átmeneteket triggerelnek ezek a kérések
 - Ha az objektum *Shutdown* állapotban van, akkor csak a *Startup()* üzenetre reagál;
 - A *Waiting* állapotban az objektum további üzenetekre vár:
 - A *reportWeather()* üzenetre a rendszer a *Summarising* állapotba megy át;
 - A *calibrate()* üzenetre a rendszer a *Calibrating* állapotba megy át;
 - A *Collecting* állapotba óraütésre (*clock* üzenet) megy át.

A meteorológiai állomás állapot-diagramja



3.5 Objektum interfészek specifikációja

- Az objektumok interfészeit definiálni kell, hogy az objektumok és más komponensek párhuzamosan fejleszthetők legyenek.
- A tervezés során el kell kerülni az interfész *reprezentáció* tervezését. Ezt az objektumba kell rejteni.
- Az objektumoknak számos interfésze lehet, amelyek a szolgáltatott különféle metódusokra jellemzőek.
- Az UML-ben osztály-diagramot használunk az interfészek specifikálására, de pl. a Java nyelv is használható e célra.

A meteorológiai állomás interfésze

```
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
```

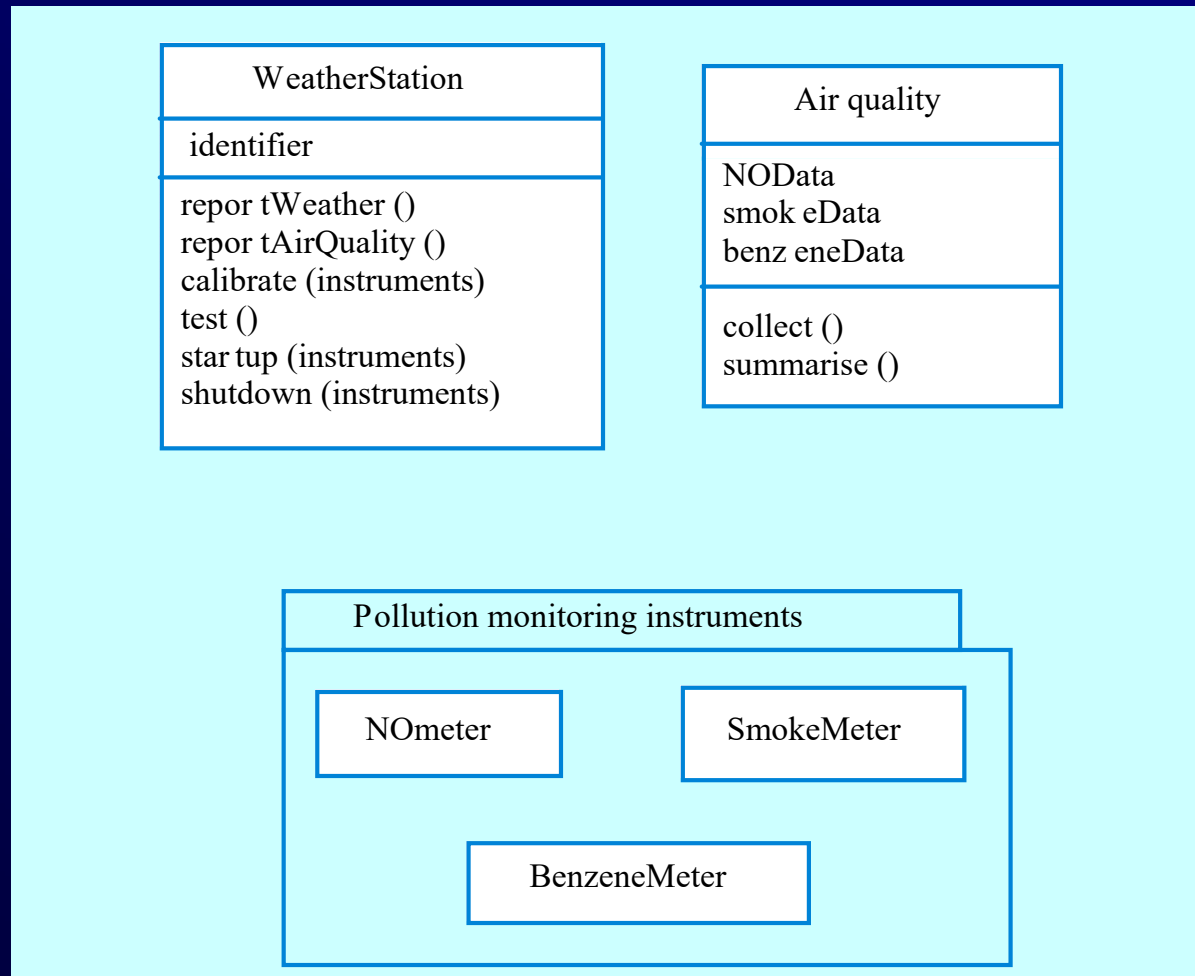
4. A terv evolúciója

- Az objektumokba való információ-rejtés következménye, hogy az objektumon végrehajtott változások nem befolyásolnak más objektumokat előre nem látható módon.
- Példa:
 - légszennyezettség-mérést kell a meteorológiai állomás szolgáltatásaihoz adni (a levegő mintavételezéséből kiszámítja az atmoszférában jelenlevő különféle szennyező anyagok mennyiségét).
 - A légszennyezettség-adatokat az időjárási adatokkal együtt továbbítjuk.

A szükséges változtatások

- Hozzáadjuk az **Air quality** objektum-osztályt a WeatherData mellé.
- A **WeatherStation** objektum-osztályt kiegészítjük a **reportAirQuality** operációval. A vezérlő szoftvert kiegészítjük a szennyezettség-értékek gyűjtésével.
- A légszennyezést mérő műszereket jelképező objektumokat adunk a rendszerhez.

Légszennyezés-monitorozás



Összefoglalás

- Az OOT olyan tervezési megoldás, ahol a komponenseknek saját állapotaik és operációik vannak.
- A objektumoknak rendelkezniük kell létrehozó (constructor) és megfigyelő (inspection) operációkkal. Ezek más objektumok számára biztosítanak szolgáltatásokat.
- Az objektumok soros vagy párhuzamos módon is implementálhatók.
- Az UML (Unified Modeling Language) számos jelölési módot biztosít különféle objektum-modellek definiálásához.

Összefoglalás

- Az objektum-orientált tervezési folyamat során számos különböző objektum-modellt hozunk létre. Ezek statikus vagy dinamikus rendszermodellek lehetnek.
- Az objektumok interfészeit precízen definiálni kell pl. egy programozási nyelv (pl. Java) segítségével.
- Az objektum-orientált tervezés nagy valószínűséggel leegyszerűsíti a rendszer evolúcióját.