

A dokumentáció felépítése

Készítette: Keszthelyi Zsolt, 2010. szeptember

A szoftver dokumentációját az itt megadott szakaszok szerint kell elkészíteni. A szoftvert az Egységesített Eljárás (Unified Process) ajánlásai alapján kell készíteni. A modellezéshez UML diagramokat kell használni.

1. Követelmény feltárás

A követelmény feltárás során felméri és összegyűjti a megrendelt szoftverrel szemben támasztott felhasználói követelményeket, elemzi az alkalmazási szakterületet.

1.1. Célkitűzés, projektindító dokumentum

A projektindító dokumentum egy viszonylag rövid, szöveges leírás, mely a megrendelő legfontosabb elvárásait tartalmazza.

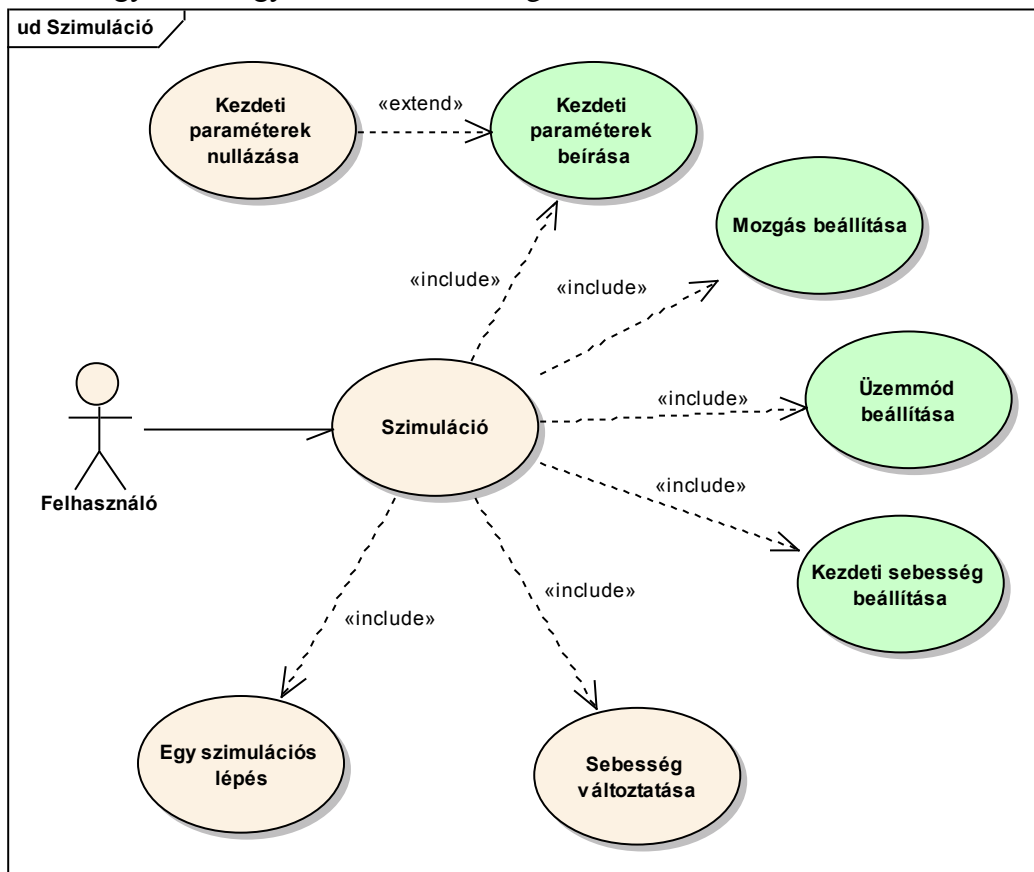
1.2. Szakterületi fogalomjegyzék

A fejlesztett szoftver által támogatott szakterület fogalmainak gyűjteménye, magyarázata. Szükséges a szoftver fejlesztői-, felhasználói dokumentációjának megértéséhez. A fogalomjegyzék a fejlesztés előrehaladása során folyamatosan bővül.

1.3. Használatieset-modell, funkcionális követelmények

A programmal szemben támasztott funkcionális- és szakterületi követelményeknek használati esetek felelnek meg. A használatieset-modell használati eset diagramokat és ezek szöveges leírását tartalmazza. A megrendelő számára is érthető termék, mely segíti a megrendelő és a fejlesztő közti kommunikációt. A használati eseteket a teljes fejlesztési folyamat során figyelembe kell venni.

Például így néz ki egy használati eset diagram:



Egy használati eset szöveges leírásának a sablonja:

<p>A használati eset neve: Szimuláció Leírás: szöveges leírás Előfeltétel: szöveges leírás Utófeltétel: szöveges leírás</p>

1.4. Szakterületi követelmények

[A fejlesztett szoftver által megcélzott alkalmazási terület szabályszerűségei.](#)

1.5. Nem funkcionális követelmények

[Használati esetekkel nem leírható követelmények listája.](#)

Például:

<p>Fejlesztési módszertan:</p>

- Egységesített Eljárás

<p>A fejlesztéshez szükséges hardver:</p>
--

- CPU: Pentium 4, RAM: 1 GB, videó: 1024x768

<p>A fejlesztéshez használt szoftverek:</p>
--

- Operációs rendszer: Windows XP SP3
- Követelmény elemzés: Word szövegszerkesztővel, dokumentum-sablonok használatával
- CASE eszköz: Enterprise Architect 8
- Java fejlesztőeszköz: NetBeans 6.9.1

<p>A futtatáshoz szükséges operációs rendszer:</p>

- Tetszőleges operációs rendszer, melyhez létezik JRE 6 implementáció

<p>A futtatáshoz szükséges hardver:</p>
--

- Operációs rendszerek szerint megadva

<p>Egyéb követelmények:</p>

- Intuitív felhasználói felület, könnyű kezelhetőség

2. Tervezés

2.1. A program architektúrája

[A program legfontosabb elemeinek és ezek kapcsolatainak meghatározása. A megoldáshoz felhasznált tervezési minták megadása.](#)

2.2. Osztálymodell

[Az osztálymodell a program statikus modellje, mely megmutatja a program szerkezeti felépítését. Osztálydiagramokból és osztályleírásokból áll.](#)

[Ebben a részben kell meghatározni a probléma megoldáshoz szükséges osztályokat, az osztályok feladatait és kapcsolatait. Az osztályokon belül először csak a legfontosabb adatokat és metódusokat kell megadni.](#)

[Először célszerű szakterületi osztálydiagramot készíteni. Ezen csak olyan osztályok szerepelnek, melyeket a programozói tudással nem rendelkező megrendelő is megért. A szakterületi osztálydiagram segíti a megrendelő és a fejlesztő közti kommunikációt. A szakterületi osztálydiagramon nem szerepelnek a fejlesztőeszköz osztálykönyvtárának osztályai.](#)

[Az osztálymodell része lehet az adatbázisterv is.](#)

Az osztályleírás sablonja:

Osztálynév

Sztereotípiák: vezérlő, határ, információhordozó (más néven egyed), konténer, vegyes, egyéb

Ős:

Példány: a példány(ok) neve(i)

Feladat: szöveges leírás

2.3. Adatbázis terv

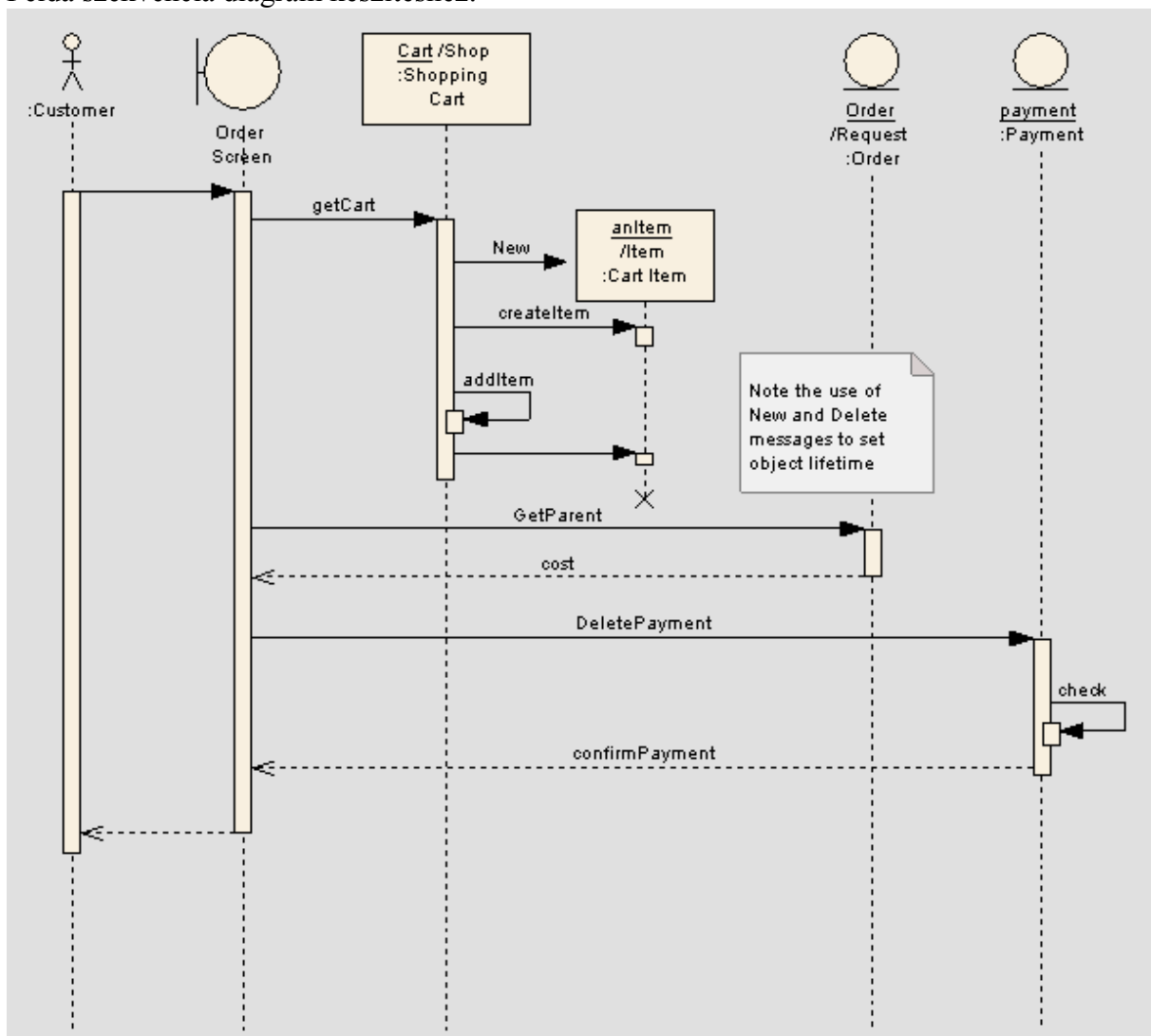
Megadható osztálydiagrammal.

2.4. Dinamikus működés

A program dinamikus működése modellezhető szekvencia diagrammal, együttműködési diagrammal, állapot-átmeneti diagrammal, aktivitás diagrammal.

A szekvencia diagram a rendszerben lévő objektumok közötti üzenetváltásokat mutatja az idő függvényében. Segít meghatározni a megoldáshoz szükséges metódusokat, és segít feltárni az objektumok közti kapcsolatokat. Egy használati eset végrehajtásának egy lehetséges forgatókönyvét mutatja be.

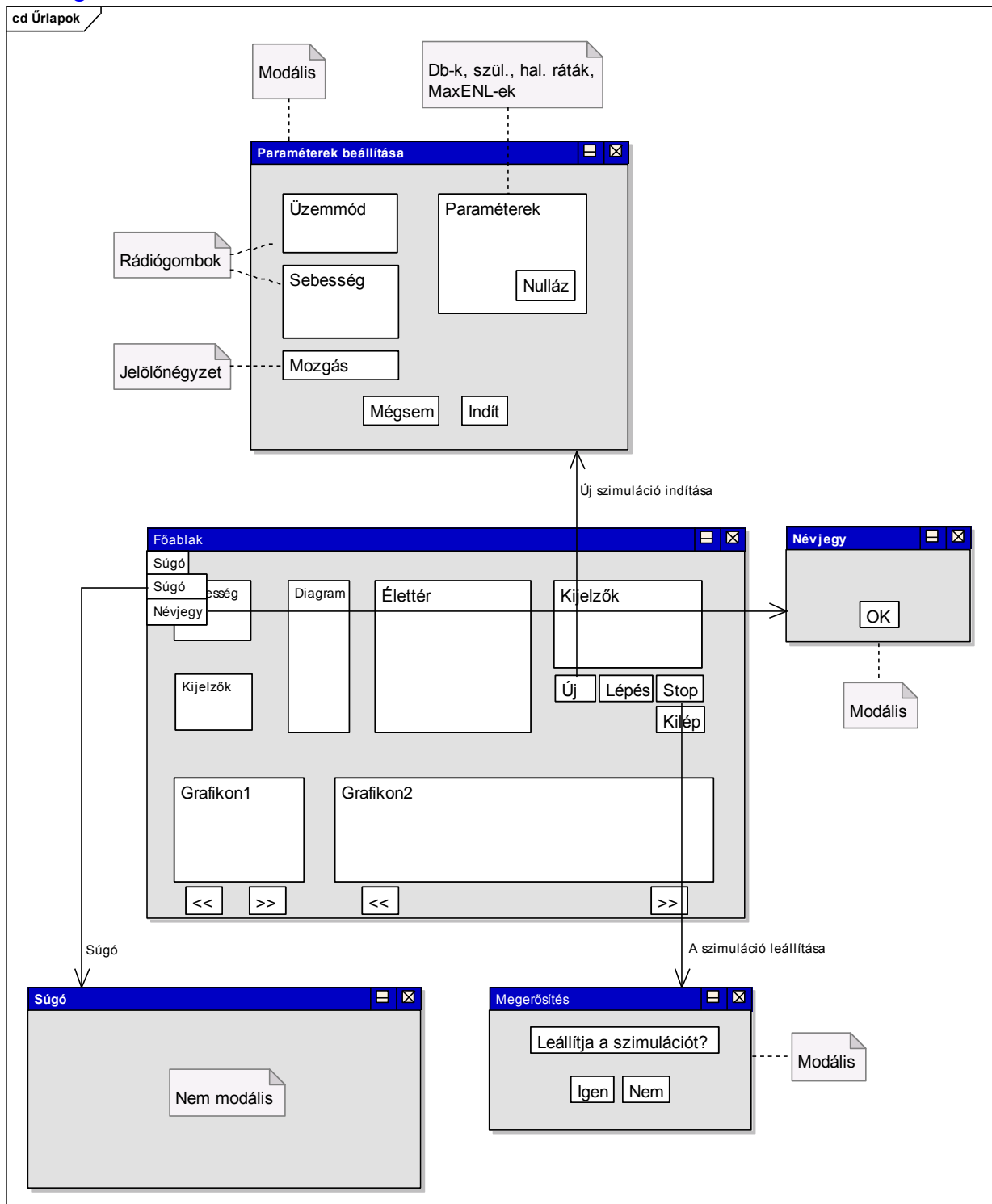
Példa szekvencia diagram készítéshez:



Egy CASE eszköz segítségével a szekvencia diagramokat az osztálydiagramokkal párhuzamosan, egymással szinkronban készíthetjük. Egy objektumnak küldött üzenet meghatározásakor választhatunk az osztálydiagramon szereplő üzenetek közül, vagy kitalálhatunk egy új üzenetet.

2.5. Felhasználói-felület modell

Képernyőtervek. A felhasználói-felületet modellezhetjük CASE eszközzel. A modell megmutatja, hogy a program milyen ablakkal rendelkezik, és ezek hogyan kapcsolódnak egymáshoz. Itt még nem szükséges teljes részletességgel megtervezni az egyes ablakok belső felépítését, hiszen a felhasználói felület részletes kidolgozása 4GL fejlesztőeszközzel is lehetséges. Például:



2.6. Részletes programterv

A részletes programterv osztálydiagramokból és szöveges osztályleírásokból áll. Bemutatja a programban definiált összes osztályt, az osztályok összes adatát, metódusát.

Az itt olvasható osztályleírások kiegészítik az „Osztálymodell” című részben található osztályleírásokat. Szövegesen definiálni kell az összes általunk készített osztály összes metódusának a feladatát.

Az osztályleírásokban ne szerepeltessük a `get/set/is` metódusokat, mivel a feladatuk egyértelmű!

3. Implementáció

3.1. Fejlesztőeszközök

Milyen fejlesztőeszközöket, technológiákat, munkamódszereket használunk.

3.2. Forráskód, futtatható kód

A forráskódot, futtatható kódot, erőforrásokat (képek, hangok stb.) tartalmazó fájlok nevei. A fájlok csomagolása.

3.3. Alkalmazott kódolási szabványok

<http://www.oracle.com/technetwork/java/codeconv-138413.html>

A különböző jellegű azonosítók elnevezésére érdemes szabványokat bevezetni. Ebben a szakaszban le kell írni a bevezetett szabványokat.

A forráskódban „beszédes” azonosító neveket kell használni.

Be kell tartani a Java-ra vonatkozó kódolási konvenciókat.

<http://www.oracle.com/technetwork/java/codeconventions-135099.html#367>

Például az osztályok neve nagybetűvel kezdődik, az objektumok neve kisbetűvel kezdődik, a beállító metódusok neve **set** előtaggal kezdődik, a lekérdező metódusok neve **get/is** előtaggal kezdődik. A Swing komponensek elnevezése (Hungarian prefix notation):

```
JButton btIgen, btNem;  
JCheckBox cbMozgas;  
JMenuItem miMentes;  
JLabel lbKerdes;
```

4. Tesztelés

Milyen módszertannal tesztelünk?

A teszteléshez választhatjuk a feketedoboz módszert. A feketedoboz módszer dinamikus tesztelési eljárás. Adatvezérelt, nem veszi figyelembe a program belső szerkezetét. A teszteseteket a használati esetek lehetséges forgatókönyvei alapján kell összeállítani.

4.1. „A” használati eset tesztelése

Bemenő, kimenő adatok megadása.
Ellenőrző lista

4.2. „B” használati eset tesztelése

4.3. „C” használati eset tesztelése

stb.

Választhatjuk a fehérdoboz módszert.

Írhatunk tesztelő osztályokat is (test driver).

[Modul teszt, integrációs teszt.](#)

5. Felhasználói dokumentáció

5.1. A futtatáshoz ajánlott hardver-, szoftver konfiguráció

5.2. Telepítés

5.3. A program használata

A felhasznált irodalom jegyzéke (nyomtatott és internetes anyagok)