DEBRECENI EGYETEM
INFORMATIKAI KAR

GRAFIKUS SZERKESZTŐ FELÜLET KÉSZÍTÉSE AZ IND STS 2005 PROGRAMOZÁSÁHOZ

SZAKDOLGOZAT

Témavezető:
Dr. Juhász István
egyetemi adjunktus

Készítette:
Fogas János
programozó matematikus

Debrecen

2008

Tartalomjegyzék

Ábralista

Táblázat lista

1. Bevezetés

Napjainkban a technika fejlődése révén az informatika egyre nagyobb teret hódít meg. Egyre szélesebb körben használnak számítógépes alkalmazásokat, hogy ezáltal egyszerűbbé, kényelmesebbé tegyék mindennapi életünket. Ezek az alkalmazások is együtt fejlődnek az informatikával. Egyre megbízhatóbbak és használhatóbbak, de mindezek mellett egyre monumentálisabbak is. Egy komolyabb szoftver kifejlesztése akár több tucat (vagy akár több száz) ember éves, több éves munkája is lehet. Ezáltal az összetett szoftverek előállítási költsége igen nagy. A fejlesztési idő lecsökkentésével a költségek is jelentős mértékben csökkenthetőek. Szerencsére ma már egyre modernebb integrált fejlesztői eszközök (IDE) állnak rendelkezésünkre, amelyek megkönnyítik a szoftver fejlesztését.

Szakdolgozatomban egy olyan IDE kiterjesztés (plugin) megírását tűztem ki célomul, amelynek segítségével vizualizálható és szerkeszthető egy XML alapú programozási nyelvben megírt program. A vizualizációnak köszönhetően gyorsabb és hibamentesebb program írható.

A szakdolgozat írás időszaka alatt szoftver fejlesztői státuszban dolgoztam az IND Kft.-nél, így alkalmam nyílt bekapcsolódni egy banki Internetes szolgáltatást fejlesztő projektjébe, valamint rálátást nyertem a fejlesztési folyamatokra. A dolgozatban bemutatásra kerül egy XML alapú banki tranzakciót leíró nyelv, és a hozzá készített szerkesztő.

2. A szoftverfejlesztési trendek

A szoftverfejlesztő eszközök és folyamatok együtt fejlődtek a programozási nyelvekkel és az informatikával. Igen nagy utat tettek meg, amíg eljutottak az integrált fejlesztői környezetig. Eme fejlődés lépcsőfokai tömör dióhéjban a teljesség igénye nélkül:

- Ősidő: a gépi kód közvetlenül került bevitelre. Eleinte konzolkapcsolókkal, majd később lyukkártya segítségével.

- Kezdetek (1980-as évek elejéig-közepéig):
    - parancssori eszközök:
        - Minden eszköz egy-egy kis feladat végrehajtására képes (pl.: fordítás).
        - A fejlesztéshez sok ilyen eszköz kell, ezáltal a fejlesztési folyamat felszabdalódik. (Kódszerkesztés -> fordítás -> debug -> ... más és más programmal.)
        - A fejlesztő csak „kezdetleges integrációt” tud végezni. (make)

- Az 1980-as évek vége:
    - Hagyományos tool-ok
        - Már van IDE kezdemény (Turbo Pascal), de ez csak egy gyártóhoz kapcsolódik.
        - Csak egy fejlesztési részfolyamathoz lehetett használni. (Pl.: kódolás)
    - „4GL” (negyedik generációs eszközök):
        - Speciális alkalmazások (pl. ügyviteli rendszerek) gyors fejlesztése (pl. Magic)
        - Ezek csak egy adott futtató környezetben működő alkalmazások.

- 1990-es évek:
    - Több különböző keretrendszer egy grafikus platformra, és emiatt nehézkes az átjárás a különböző tool-ok között.
        - Win32:
            - Borland – Visual Control Library (Delphi, C builder)
            - MS – MS Foundation Classes (Visual Studio)
        - Linux:
            - QT – QT Designer
            - KDE – Kdevelop
            - Kylix – Borland
    - Integrált fejlesztőeszközök nélkül reménytelen az ipari méretű alkalmazások fejlesztése:
        - Bonyolult és összetett platform (több száz osztályt is tartalmazhat)
        - XML telepítési leírók, amelyek többsége nem emberi szerkesztésre van optimalizálva.

- 2000 után
    - Megjelennek a sok komponensekből álló keretrendszerek
    - Napjainkban az Integrált fejlesztőrendszerek (IDE)
        - Közös keretet adnak az eszközöknek.
        - Általában egy gyártó tooljaikat foglalják össze (pl. Borland Delphi, MS Visual Studio)
        - Kényelmes, hatékony
        - Külső eszközökkel nem, vagy nagyon nehezen bővíthetőek.
    - Teret hódítanak a nyílt fejlesztő rendszerek.

A tendenciák a nyílt fejlesztőrendszerek előretörését mutatják, amelyeknek számos előnyös tulajdonsága van:

- jól definiált keretrendszer
- nyílt specifikáció
- könnyen bővíthető
- platform-független

- programozási nyelv független
- optimális a gyorsan változó igényekhez

A felsoroltak mindegyikével rendelkezik az Eclipse, ezért esett a választásom erre a fejlesztői környezetre. Mindezek mellett még erős ipari (pl.: IBM, BEA, Intel, SAP, stb.) támogatottsága is van.
Az Eclipserről bővebb leírás a *6.3. Eclipse* fejezetben található.

3. Mi is az a plugin?

Jogosan merülhet fel a kérdés: mi is az a plugin? Szó szerinti fordításban: *becsatlakoztat*, de úgy gondolom jobb szó rá a *bővítmény* vagy a *modul*.

A definíció szerint a plugin: egy adott szoftverbe, vagy hardverbe opcionálisan beépíthető, annak képességeit bővítő vagy módosító kiegészítő modul.

A bővítmények a komponens alapú szoftverfejlesztési elvet követik. Az így készült programok funkcionalitása fel van darabolva modulokra. Egy funkciót egy (vagy több) bővítmény valósít meg. A bővítmények egymásra épülnek, használhatják egymás szolgáltatásait, illetve a felhasználó felé nyújthatnak további szolgáltatásokat is. Előnye, hogy az egyes bővítmények könnyen cserélhetőek. Új igények esetén nem szükséges a teljes programot újra írni, elegendő csak néhány modult.

Egy bővítmény lehet:

- egyedül álló, azaz egymaga, más bővítmények nélkül képes működni, nem támaszkodik azok szolgáltatásaira.
- nem egyedül álló, azaz támaszkodik más bővítmények szolgáltatásaira, és csak azokkal együtt működőképes.

A szakdolgozatban a plugin szó alatt az Eclipse plugint értem. Erről bővebb ismertető az *5.3.3. Eclipse Pluginok* fejezetben olvasható.

4. Célkitűzések

A plugin célja egy egyszerű, áttekinthető, könnyen használható grafikus szerkesztői felület, amely megkönnyíti a tranzakciós XML fájlok szerkesztését.
A plugin támaszkodik már megírt pluginekre, felhasználva, illetve kiterjesztve azokat. Ezek alapján az alábbi részekre bontható fel:

- *Szöveges szerkesztő* (text editor): az XML fájlok szöveges formában jelennek meg és közvetlenül szerkeszthetőek.
- *Attribútum szerkesztő* (property editor): az egyes XML elemek attribútumait lehet táblázat formájában megtekinteni (név, érték) valamint szerkeszteni.
- *Grafikus szerkesztő* (design editor): az XML dokumentum által leírt program folyamatábráját jeleníti meg, amely módosítható.
- *Vázlat megjelenítő* (outline view): látható az aktuálisan szerkesztett XML dokumentum elemeinek a listája, valamint a grafikus folyamat ábra kicsinyített vázlata. Ezáltal nagyítás, kicsinyítés, vagy több képernyős ábra esetén is könnyen és gyorsan lehet a megfelelő részlethez navigálni.

4.1. Formai követelmények

Áttekinthető felhasználói felület

- A program segítse, ne pedig hátráltassa a felhasználót.
- Az eszköztár legyen egyszerű és egyértelmű.

Megjelenés:

- Az elemek vizuálisan is jól elkülönüljenek egymástól. Színben és formában is különbözzenek az áttekinthetőség végett.
- A szöveges és grafikus szerkesztő egyetlen többlapos (Multipage) ablakban jelenjen meg, ahol fülek segítségével lehet közöttük választani

4.2. Elvárt funkciók

A szöveges szerkesztőtől elvárt funkciók:

- Tagoltan jelenítse meg a fájlokat, és a mentés is úgy történjen.
- Emelje ki színekkel a szintaktikai elemeket.
- Ha változik a fájl tartalma, küldjön értesítést a grafikus szerkesztőnek a változásról, és frissítse azt.

A grafikus szerkesztőtől elvárt funkciók:

- Az egyes elemek tulajdonságai szerkeszthetőek legyenek a property nézetben.
- Mentéskor az ábrát képezze le egy XML fájllá. Az állomány és az ábra között kölcsönös megfeleltetésnek kell lennie, azaz a fájlból egyértelműen felépíthető az ábra és viszont.
- A menü ikonok segítségével a felhasználónak lehetősége legyen:
    - Nagyítani, kicsinyíteni a vektorgrafikus ábrát.
    - Több elem kijelölése esetén a felhasználó választhat különböző igazítási szempontokból.
    - Egy eseményt visszavonni vagy megismételni. (Undo – Redo). Ezáltal, ha a felhasználó meggondolja magát könnyedén vissza tudja állítani az ábrát egy előző állapotba.
- Amint változik az ábra, küldjön értesítést a szöveges szerkesztőnek, hogy változás történt, és frissüljön az ablak.
- Szövegek (címkék) szerkesztése közvetlenül a rajzon (direct editing)

4.3. Felhasználóbarát működés

Az alkalmazás elsődlegesen informatikusok számára készül. Nem szükséges hozzá a tranzakciós XMLek felépítésének az ismerete.

Egyéb felhasználót segítő elemek:

- A felület rácsozott, és az elemek rácshoz igazítása automatikus, így a felhasználónak nem kell bajlódnia az elemek egy vonalba rendezésével.
- Az elemek méretezhetőek, így egyedibbé tehető az ábra.

5. Tervezés

Ebben a fejezetben bemutatom, hogyan készült a program. Milyen ötletek merültek fel, és vetődtek el. Milyen problémákba ütköztem, és hogyan sikerült megoldani őket.

5.1. Szöveges szerkesztő (Text Editor)

Mivel a szakdolgozat elsődleges célja egy grafikus szerkesztő létrehozása, így úgy döntöttem nem írok saját szöveges szerkesztőt. Két lehetőség közül lehetett választanom:

- felhasználok egy már megírt plugint, vagy
- létrehozok egy sajátot.

Az utóbbi mellett döntöttem, mert az Eclipse Plugin Project tartalmaz sablonokat, amelyek közül az egyik egy alap XML szerkesztő. Így kézhez kaptam egy editort az alábbi tulajdonságokkal:

- kivágás, másolás, beillesztés
- szövegben való keresés
- dupla kattintás támogatása (szavakat tudunk így kijelölni)
- szintaktikai kiemelés (az elemeket kiszínezi)

A továbbfejlesztési lehetőségek között részletezem az ebben rejlő lehetőségeket.

5.2. A grafikus szerkesztő

A szerkesztő a *GraphicalEditorWithPalette* leszármazottja. Ez az osztály egy editort szolgáltat, amelyik két részre van osztva. Az egyik oldalt egy paletta található, a másikon egy grafikus szerkesztő. A grafikus szerkesztőn megjelenő elemek mindegyike az alábbi osztályokból épülnek fel:

- *EditPart*: a *createFigure* metódusával létrehozza a grafikus felületen megjelenő elemet, ami az *IFigure* osztály egyik leszármazottja.

- *Model*: a grafikus ábra mögötti adat modell. Feladata, hogy tárolja és kezelje az adatokat.

5.2.1.Elemek

Mivel a tranzakciós XML túl sok elemet tartalmaz, ezért szükségessé vált az elemek számának a csökkentése, és csak a legszükségesebbek megtartása. Boehm és Jacopini 1966-ban publikáltak egy cikket, amelyben leírják, hogy minden algoritmus felépíthető a következő három vezérlési szerkezet segítségével:

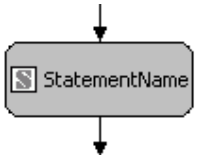
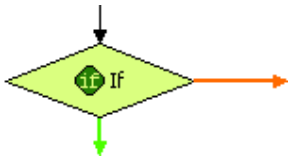
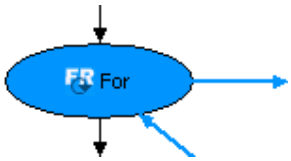
- Szekvencia
- Szelekció
- Iteráció

Ezt még nem tudták bizonyítani egészen 1968-ig, amikor is Mills bebizonyította.

Mindezek mellett még egy plusz elemet szükséges volt hozzáadnom; egy gyökér elemet, hogy egyértelmű legyen, hol kezdődik az ábra. A Description (leírás) elem pont megfelelt erre a célra. Kötelező elem a tranzakciós XML-be, és a fájl legelején szerepel.

Ezek alapján az alábbi elemeket tartalmazza a szerkesztő:

Elem	Ikon	Név	Leírás	Ismertető
		Description	Világos narancssárga téglalap, vastag fekete kerettel.	Rövid ismertetőt tartalmaz a tranzakciós XMLről. Kötelező elem. Az ábra gyökéreleme. Csak egyetlen (fekete) nyíl mutathat belőle.

Elem	Ikon	Név	Leírás	Ismertető
		Statement	Szürke lekerekített sarkú téglalap, vékony fekete kerettel	Egy utasításnak felel meg. Opcionális elem. Több nyíl is mutathat rá, de csak egy nyíl indulhat belőle. Csak olyan színű nyíl indulhat belőle, amilyen rá is mutat.
		If	Zöld rombusz, fekete kerettel. Alul zöld nyíl indul belőle. Jobb oldalt egy piros	Elágazás. (Szelekció.) Opcionális elem. Több nyíl is mutathat rá. A zöld színű (True) nyílnak kötelezően kell belőle indulnia. A piros színű (False) nyíl opcionális.
		For	Kék ellipszis fekete körvonallal. Egy kék nyíl mutat rá, és egy kék és egy fekete nyíl mutat belőle.	Iteráció. Opcionális elem. Több nyíl is mutathat rá. A kék színű (For block) nyílnak kötelezően kell belőle indulnia, és kötelezően mutatnia is kell rá egynek. (A ciklus magját jelképezi a kék nyilak.) A ciklus vége utáni következő elem fekete nyíllal következik.

1. Táblázat Elem típusok

5.2.2.Nyilak

Az egyes elemeket nyilak kötik össze egymással, így egyértelmű sorrendiséget állítanak fel. Eleinte három nyíl típus volt tervezve, a fekete nyíl, a zöld (true) és a piros (false). Úgy gondoltam, a For-hoz elegendő lesz a zöld és a piros nyíl. A ciklus magját a zöld nyilak jelölték volna. A ciklusmag utolsó eleme zöld nyíllal visszamutatott volna a For elemre. A For-ból pedig egy piros nyíl mutatott volna a cikluson kívüli első elemre. Ez az ötlet azért lett elvetve, mert ha egymásba ágyazunk elemeket (pl.: If Else ágán belül van egy For), akkor nem egyértelmű, hogy hol ér véget az egyik blokk és hol kezdődik a másik. Így a jobb áttekinthetőség végett lett bevezetve egy új nyíl típus (kék) a For blokkjának. A For blokkja kékkel van jelölve, és a blokk utáni elemre fekete nyíl mutat a For-ból.
A különböző nyíl típusok tulajdonságai az alábbi táblázatban látható:

Nyíl	Ikon	Név	Leírás	Ismertető
→		Arrow	Fekete nyíl, normál vonal vastagsággal.	Az elemek egymásutániságát hivatott mutatni.
→		True Arrow	Zöld nyíl, dupla vonal vastagsággal.	Az IF true ágát mutatja. Azok az elemek, amelyek ezzel vannak összekötve egy blokkot alkotnak.
→		False Arrow	Piros nyíl, dupla vonal vastagsággal.	Az IF false ágát mutatja. Azok az elemek, amelyek ezzel vannak összekötve egy blokkot alkotnak.
→		For Block Arrow	Kék nyíl, dupla vonal vastagsággal.	A FOR ciklusmagját jelöli. Azok az elemek, amelyek ezzel vannak összekötve egy blokkot alkotnak.

2. Táblázat Nyíl típusok

5.2.3. Betöltés

Az XML fájl betöltésére két elterjedt módszer van:

- **D**ocument **O**bject **M**odel (*DOM*): A DOM értelmező az XML dokumentumot egy fa szerkezetű objektum modellé fordítja le. Minden eleme maga is objektum.
- **S**imple **A**PI for **X**ML (*SAX*). Az SAX értelmező szekvenciálisan dolgozza fel a XML dokumentumot, és minden eseményről értesíti a kezelő programot callback mechanizmussal.

A SAX mellett döntöttem, mert azzal egyszerűbben fel tudom építeni az ábrát a grafikus szerkesztőben.

A *MySAXParser* osztály végzi a dokumentumból az ábra leképezését.

5.2.4. Mentés

A grafikus ábra leképezése XML dokumentummá **D**ocument **O**bject **M**odel (*DOM*) segítségével történik. A tervezés során az alábbi ötleteim merültek fel:

- A memóriában megtalálható egy DOM, ami az ábrát reprezentálja, és bármilyen esemény történik, arról értesül és frissül
    - Előnye: azonnal tudok menteni, a modell a rendelkezésemre áll
    - Hátránya: minden egyes változásról értesülnie kell, sokat kell frissíteni.
- Amikor a felhasználó menteni szeretne, akkor készítem el a reprezentációt. Elkérem az ábrát, megkeresem a gyökérelemet (Description, ami épp ezért került bevezetésre), majd bejárom az ábrát.
    - Előnye: nem igényel nagy módosítást, egyszerűen kivitelezhető
    - Hátránya: nem azonnal történik meg a mentés, előtte még a teljes ábrát be kell járnom.

Az utóbbi megoldást választottam egyszerűsége miatt.

A *ModelToDom* osztály végzi az ábra leképezését DOM-má és a dokumentum fájlba írását. Eleinte a fájl írása a standard API-ban lévő *Transformer* osztály segítségével történt, de a kimenet nem volt tagolt és áttekinthető, ezért most már az *XMLSerializer* osztály végzi ezt a feladatot.

5.3. Több lapos szerkesztő (Multipage Editor)

A szerkesztő akkor lehet igazán hatékony és kényelmes használatú, ha egyszerre képes kezelni a szöveget és az ábrát. Erre nyújt megoldást a *MultiPageEditorPart* osztály. Egy osztályt kell származtatni belőle, amelynek legalább az alábbi metódusokat felül kell definiálnia:

- *createPages*: létrehozza a tabokat és az azokon elhelyezkedő editorok egy példányát. Az első tab-on helyezkedik el a grafikus szerkesztő. A második tabra pedig az XML szerkesztő került.
- *doSave*: a tartalom mentését végzi. A legegyszerűbb megoldás, ha az éppen aktív tabon lévő editorra bízzuk a mentést, meghívva az ő *doSave* metódusát.

5.4. Vázlat megjelenítő (Outline view)

A vázlat megjelenítőhöz szükséges egy osztályt létrehozni, amely kiterjeszti a *ContentOutlinePage* osztályt. Ez lesz a *MyConentOutlinePage* osztály. Az alábbi metódusokat szükséges felüldefiniálni:

- a konstruktort, ahol egy fanézetet (*TreeViewer*) adunk hozzá
- *init*, ahol a parancsokat definiáljuk. (Pl.: Undo, Redo, Delete)
- *createControl*, hozzáadunk egy vásznat (*Canvas*), amin a vázlat fog megjelenni, és egy eseménykezelőt, ami akkor fut le, ha bezáródik a nézet.
- *dispose*, ahol az eseménykezelőt távolítjuk el a bezáródás előtt.

5.5. Tulajdonság szerkesztő (Property Editor)

Ahhoz, hogy egy elemet kijelölve megjelenjenek az elem tulajdonságai a Property Editorban szükséges a modellünknek implementálja az *IpropertySource* interfészt.

6. Felhasznált technológiák

6.1. XML

Az XML az *e***X***tensible* **M***arkup* **L***anguage* (bővíthető jelölő nyelv) szavak rövidítését jelenti. Az XML egy jelölő nyelv, amely nem határozza meg:

- a nyelv jelölő elem készletét és
- nyelvtanát sem,

ezért teljes mértékben kiterjeszthető. Az XML technológia szabványosításával a W3C, vagyis a World Wide Web Consortium foglalkozik. Az ajánlása, egy olyan szintaxist ad meg, amelyet betartva különböző jelölő nyelvek (mint például az XHTML) hozhatóak létre. Éppen ezért szokták metanyelvnek, nyelvleíró nyelvnek is nevezni.

Egy XML dokumentum elemekből áll, amelyek neve (*szókincs*), egymáshoz való kapcsolata és tartalma szabályokkal rögzíthető (*nyelvtan*). Az XML specifikáció megad egy szintaxist mind az XML dokumentumokra - vagyis az elemek jelölésére -, mind a szabályok leírására (DTD). A megadott szintaktikai szabályok betartásával bárki saját nyelvet (*dokumentum-típust*) készíthet, s azt a megfelelő XML-konform eszközzel ellenőrizheti, feldolgozhatja. A nyelvtan és a szókincs megadása nem kötelező.

A fentiek alapján:

- *bővíthető* (eXtensible), mert saját elemeket lehet deklarálni;
- *jelölő* (Markup), mert az elemek - egy megadott módon - jelöléssel különböztethetőek meg egymástól;
- *nyelv* (Language), mert rögzíthető a szókincs és a szabály.

6.1.1. Az XML eredete (SGML)

Az XML az SGML (*Standard Generalized Markup Language*, ISO8879) részhalmaza. Az SGML-t dokumentumok számítógépes tárolására használják, kihasználva a tartalmi jelölés nyújtotta előnyöket (pl. repülőgép-dokumentációk, szótárak tárolásánál).

A legismertebb SGML alkalmazás minden bizonnyal a HTML, amely nagy karriert futott be az Internet terjedésének köszönhetően. A HTML elemkészlete az idő haladtával elégtelennek

bizonyult, és a különböző böngészőgyártó cégek saját elemekkel bővítették, amelyek ráadásul formai leírásra szolgáltak. Ilyen elem volt például a Netscape `<CENTER>` eleme. Ezek ellensúlyozására vezették be a CSS stíluslapot, hogy egy külső leírással lehessen a megjelenésen változtatni, ám ez csak ideig-óráig jelentett haladékot. Ekkor a W3C SGML munkacsoportja egy teljesen új megoldáson kezdett dolgozni 1996-ban, az XML-en. Tekintettel 20 éves - sikeres - múltjára és képességeire, az SGML-t választották alapnak, s megfogalmaztak néhány olyan megszorítást vele szemben, amely egyszerűbbé teszi a szükséges eszközök fejlesztését, de követelmény maradt, hogy minden XML dokumentum SGML-konform legyen. Ez az egyszerűsítés nagy lökést adott az XML-nek, és rengeteg eszköz került ki egy-két éven belül a különböző nagy (és nem olyan nagy) szoftver-gyártóktól. Az XML és SGML alkalmazások közti különbséget úgy szokták megfogalmazni, hogy az XML-t alkalmazás-orientált, míg az SGML-t dokumentum-orientált megközelítés jellemzi.

6.1.2. XML verziók

Az XML 1.0-t 1998-ban definiálták, és azóta többször is módosították anélkül, hogy új verziószámot kapott volna. Jelenleg a 2004. február 4-én publikált harmadik kiadás az aktuális. Széles körben elterjedt, és még mindig ajánlják általános felhasználásra. Az XML 1.1-et egy napon adták ki az XML 1.0 harmadik kiadásával.

6.1.3. Alkalmazási lehetőségek

Az XML elsősorban az internetre készült, így alkalmazható különböző dokumentumok „webes” publikálására, feldolgozására. A formátuma lehet egyfajta „ősdokumentumnak”, amelyből automatikusan generálható különböző formátumokba, mint például HTML, PDF. Nagyjából a következő típusú alkalmazások ismertek (a teljesség igénye nélkül):

- *Szövegfeldolgozás*: Az SGML „szakterülete”, de természetesen az XML is felhasználható különböző szövegek, mint például lexikonok, újságok, dokumentációk elektronikus leírására. Megfelelő tartalmi jelölésekkel a megjelenítésen túlmenő,

mélyebb gépi feldolgozásra is alkalmas dokumentumokat lehet előállítani. Például: TEI Lite XML version, DocBook.

- *Programozási nyelv*: számos programozási nyelv született, amelyek szintaktikája XML alapú. Pl.: o:XML, MetaL
- *„Közös nyelv”*: különböző rendszerek közt áramló adatok formátuma lehet. A meglévő XML-feldolgozó programok révén nem kell minden konverzióra nulláról írni egy célszoftvert. Pl. a hírek leírására alkalmas NewsML.
- *Protokoll* szintaxisa is lehet XML. Ilyen például a web-dav protokoll, amely a HTTP feletti protokoll, és HTTP-n keresztüli fájl-transzfert tesz lehetővé (és még egyebeket is). Ugyancsak XML-alapú protokoll a SOAP, amely Web-service alkalmazások közti kommunikációra használatos.
- *Alkalmazások objektumainak szerializálási formája*.
- *E-business*: Például EDI XML változata.
- *Telefónia*: WAP telefonok WML nyelve. Weblapok, dokumentumok telefonon való megjelenítése.
- Végül, de nem utolsósorban: XSL vagy CSS stylesheet segítségével a HTML helyett vagy mellett is használható. A HTML-t átdolgozták XML-be: az XHTML már az XML előírásainak is megfelel.

6.2. Java

A Java egy objektum orientált programozási nyelv, amelyet a Sun Microsystems fejleszt a 90-es évek elejétől kezdve napjainkig. Bár a nyelv neve kezdetben *Oak* (tölgyfa) volt (James Gosling, a nyelv atyja nevezte így az irodája előtt növő tölgyfáról), végül is *Java* néven vált ismertté. Négy fontos szempontot tartottak szem előtt, amikor a Javát kifejlesztették:

- objektumorientáltság;
- függetlenség az operációs rendszertől, amelyen fut (többé-kevésbé)
- olyan kódokat és könyvtárakat tartalmazzon, amelyek elősegítik a hálózati programozást;
- távoli gépeken is képes legyen biztonságosan futni.

A második tulajdonság, a platformfüggetlenség azt jelenti, hogy a Javában íródott programok hasonlóan fognak futni különböző hardvereken. Ezt úgy lehet megvalósítani, hogy a Java fordítóprogram csak egy úgynevezett *Java gépi kódra* fordítja le a forráskódot, ami aztán futtatva lesz a virtuális gépben, amely lefordítja az illető hardver gépi kódjára. Továbbá, szabványos könyvtárcsomagok léteznek, amelyek lehetővé teszik az illető hardver sajátságosságait (grafika, szálak és hálózat) egységes módon.
Vannak olyan Java fordítóprogramok, amelyek natív gépi kódra fordítják le a forráskódot, ilyen például a GCJ, így valamelyest felgyorsítják a futtatást, de ugyanakkor a lefordított program elveszti hordozhatóságát.

6.3. Eclipse

Az Eclipse egy nyílt forráskódú közösség, akiknek a projektjei egy nyitott fejlesztő platform készítésére összpontosítanak magába foglalva a bővíthető keretrendszert és különböző eszközöket a build, deploy és manage folyamatokhoz.
Vezető technológia cégek, egyetemek és kutató intézetek bővítik és támogatják az Eclipse platformot. Az Eclipse nyílt forráskódú közösség több mint 60 nyílt forrású projektet tartalmaz.

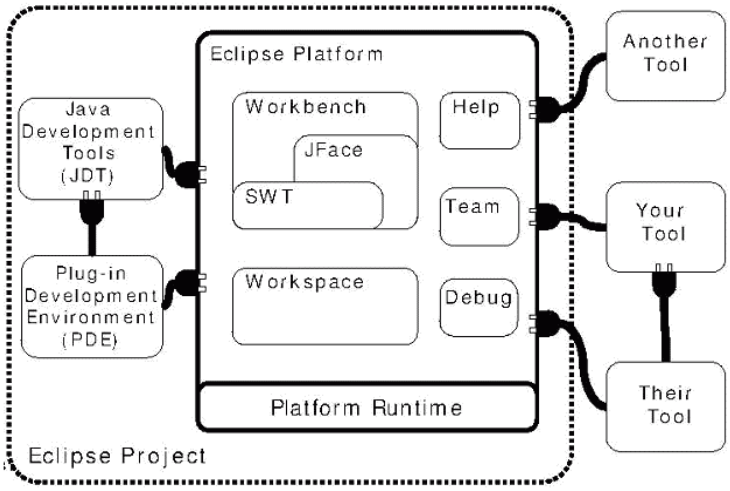
6.3.1. Eclipse Foundation

Az Eclipse Foundation egy nem profit orientált, tagok által támogatott vállalat, amely az Eclipse projekteket hosztolja. IT infrastruktúrával támogatja az Eclipse közösséget, mentora a nyílt forráskódú projekteknek az Eclipse fejlesztési folyamatban, továbbá marketing és üzleti fejlesztésekkel támogatja azt.
Az Eclipse Foundation nem fejleszt nyílt forráskódot. Minden nyílt forrású szoftvert önkéntesek, szervezetek, vagy vállalatok készítenek.

6.3.2.Eclipse Frame Work

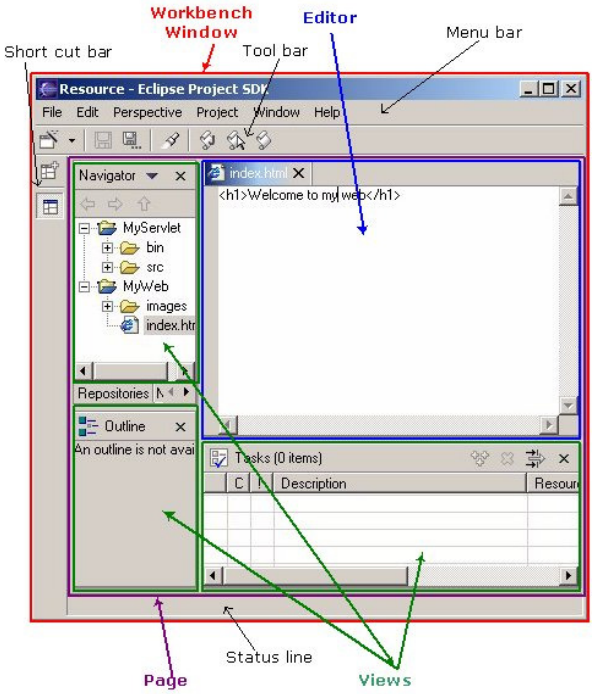
Maga az Eclipse Frame Work nem egy konkrét, monolit program, hanem egy kis kernel *plugin loader* néven. Szinte semmilyen látható szolgáltatása sincs a pluginek betöltésén és kezelésén kívül.

A szolgáltatásokat gyakorlatilag a pluginek adják, amik egymásra épülnek, használják egymás szolgáltatásait, illetve a felhasználó felé is nyújthatnak további szolgáltatásokat.



1. ábra Az Eclipse Frame Work

A Workspace gyakorlatilag egy könyvtár, ahol a felhasználó munkája tárolódik. Fontos megjegyezni, hogy nem csak a szokásos munka fájlokat (forráskód, XML, kép, stb.) tartalmazza, hanem az összes beállítást is.



2. ábra A Workbench felület

A Workbench a felület, amin keresztül a felhasználó munkáját végzi. A szokásos elemeken kívül perspektívákat, editorokat és nézeteket tartalmazhat.

A nézetek az editorokat támogatják, továbbá alternatív prezentációt vagy navigációt biztosítanak a Workbench számára.

A perspektívák a különböző nézetek és egy editor területet tartalmaznak. Míg a nézetek halmaza eltérhet a különböző perspektívákban, addig az editor terület megosztott közöttük.

6.3.3.Eclipse Pluginok

Az Eclipse keretrendszer minden eleme egy plugin. Ez a hozzáállás ellentétes a legtöbb IDE tervezési elvével, ahol is minden funkcionalitás központosítva a forráskódba van beépítve. Az Eclipse plugin rendszere a komponens alapú szoftverfejlesztési elvet követi. Megfelelő pluginek telepítésével az Eclipse kiterjeszthető úgy, hogy a JAVAn kívül más programnyelveket, például C, Perl, Ruby, Python, PHP és COBOL nyelveket is támogasson. Az Eclipse pluginek világa azonban nem ér véget a programnyelveknél, létezik LaTeX plugin is. Vannak telnet és adatbázis-kezelő pluginek is az Eclipse keretrendszerhez. Az Eclipse plugin architektúra nyitott, így a specifikáció alapján tetszőleges új kiterjesztések is írhatóak.

Az Eclipse indulásakor a *plugin loader* betölti az *eclipse/plugins* könyvtárban lévő összes plugin leíróját (*plugin.xml*). Ezután:

- megállapítja a plugin szolgáltatásait
- függőségeket vizsgál
- nem példányosít (*lazy loading*)

A plugin az aktiválásakor példányosodik, és csakis egy példányban minden más class előtt. A pluginek a bővítési pontokon (*extension points*) fejti ki a „hatását". Mindegyik saját azonosítóval (*ID*) rendelkezik.

6.4. Standard Widget Toolkit (SWT)

A Standard Widget Toolkit vagy SWT a Java programozási nyelv ablakkezelésre és grafikus felhasználói felületek létrehozására szolgáló komponensgyűjteménye. Hasonló az AWT, Swing megoldásokhoz, de ellentétben velük:

- nem része a szabvány Java API-nak,
- gyorsabb
- gazdagabb
- átgondoltabb
  - egyszerű struktúra
  - hierarchikus szerkezetek
  - átlátható eseménykezelés
  - átlátható API
- az implementációja OS függő, bár ez a programozás szempontjából transzparenst.

3. ábra Egy SWT-vel készült program különböző OS-ek alatt

Az SWT-t eredetileg az IBM fejlesztette ki, de jelenleg az Eclipse alapítvány végzi a fejlesztést és a karbantartást. Az első SWT-t használó alkalmazás az Eclipse volt.

6.5. JFace

A JFace egy java segédosztály gyűjtemény, amely a grafikus felhasználói felületek (GUI) programozását hivatott megkönnyíteni. A JFace réteg az SWT réteg felett helyezkedik el, és olyan feladatokat lát el, amelyek minden felületi elemre azonosak. A JFace tulajdonképpen az MVC (Model-View-Controller) elvet valósítja meg a Standard Widget Toolkitben (SWT). A JFace célja, hogy megkönnyítse az SWT használatát, és nem az, hogy elrejtse az SWT réteget a programozó elől. A JFace függ az SWT-től, de az SWT nem függ a JFace-től. Maga az Eclipse Workbench is csak részben támaszkodik JFace-re, vannak helyek, ahol mélyebbre nyúltak a fejlesztők, és közvetlenül az SWT szolgáltatásait használták ki.

A JFace főbb komponensei a következők:

- Viewer osztályok: Lista-, táblázat- és faszerkezetű adatok megjelenítésére, rendezésére és szűrésére lehet használni a `ListViewer`, a `TableViewer` és a `TreeViewer` osztályokat. Szöveges információ megjelenítésére használható a `TextViewer` osztály.
- Betűtípusok, színek és képek kezelésére használható az `org.eclipse.jface.resource` csomag.
- Dialógusablakok és varázslók létrehozását segítik az `org.eclipse.jface.dialogs` és az `org.eclipse.jface.wizard` csomagok.
- Az akciók, melyek több részből tevődnek össze. Minden akcióhoz tartozik egy XML deklaráció, egy `IAction` objektum, melyet az Eclipse UI példányosít, és egy `IActionDelegate` objektum, mely az akció tényleges implementációját tartalmazza.
- Ablakok létrehozására és kezelésére használható az `org.eclipse.jface.window` csomag.
- Időigényes műveleteknél használható jól az `org.eclipse.jface.operation` csomag, hogy a felhasználó nézhesse a kék csíkot a progress barban, amíg a művelet tart.

6.6. Graphical Editing Framework (GEF)

A GEF az Eclipse eszközök egyike. Segítségével grafikus szerkesztő programokat integrálhatunk az Eclipse környezetbe. Magas absztrakciós szinttel rendelkezik, és tetszőleges modell megjelenítésére alkalmas.

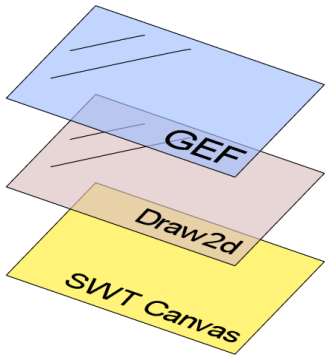
6.6.1. A GEF felépítése

A legalsó szinten található a Natív (SWT) réteg.
E fölött helyezkedik el a Draw2d réget, amely felelős:

- Megjelenítésért
- Elemek elrendezéséért
- Nagyításért

E fölött található a GEF, amely az alábbiakért felelős:

- Interakció (MVC)
- Modellből nézet leképezés
- Eclipse integráció



6.6.2. A GEF szerkesztő lépései

1. Kezdeti nézet felépítése
   - Modell bejárása, nézet elkészítése
   - Szerkesztési „szabályok” meghatározása
2. Felhasználói akciók
   - Üzenetek értelmezése a szerkesztési „szabályok” alapján
   - Modell módosítása
3. Nézetek frissítése

6.6.3.A GEF további lehetőségei

A GEF az alábbi további lehetőségekkel rendelkezik:

- Modell tulajdonságok szerkesztése az Eclipse *Properties* nézetében
- Szövegek (címkék) szerkesztése közvetlenül a rajzon (direct editing)
- Nagyítási lehetőség
- Vonalzó, automatikus igazítás
- Különálló fa modell nézet

6.7. Az IND Secure Transaction Server 2005 (STS 2005)

Az IND Secure Transaction Server 2005 (IND STS), egy bankok, pénzügyi szervezetek számára kínált, J2EE alkalmazásszerveren működő tranzakciós szerver. Az IND STS segítségével valósítható meg az igazi **„**E-Front Office**”** koncepció, azaz, hogy az elektronikus értékesítési csatornák (internet, call center, ivr, wap, sms, fax) és a fióki felületek is egy egységes platformon legyenek kiszolgálva. Mivel az üzleti logika és a megjelenítési réteg teljes mértékben különválik, így az egyszer megírt tranzakciók (utalás, egyenleg, stb.) újra felhasználhatóak más csatornákon. Az IND STS a banki számlavezető, kártya-, bróker- és CRM rendszerek legtipikusabb elérési módjaihoz nyújt támogatást, valamint egyedi elérések is programozhatók benne.

6.7.1.Az IND STS technikai felépítése

Az IND STS egy tranzakciós futásidejű környezet és egyben egy fejlesztési keretrendszer is. Az üzleti logika, a tranzakciók, az adattípusok és a háttérrendszer elérések is XML-ben definiálhatóak, így ezeket felhasználva egy fejlesztési projektben a fejlesztési idő akár 50%-a is megspórolható.

6.7.2. Az IND STS főbb funkcióinak listája

J2EE fejlesztési keretrendszer:

- XML-ben definiálható tranzakciók (üzleti logika programozás)
- XML-ben definiálható háttérrendszer-elérések (EAI programozás)
- TCP/IP, JDBC, JMS, MQ, fájl támogatás
- XML-ben definiálható adattípusok és adatellenőrzések
- Minden XML definíció újra felhasználható
- Web framework HTML alapú felületek építésére
- További gateway-ek (TCP/IP, MQ, fájl) és SOAP / web services felület

J2EE tranzakciós szerver:

- Perzisztens session-kezelés (megszakadás esetén a session folytatása)
- Szinkron- és időzített tranzakciók futtatása
- Store & forward tranzakciók kezelése (háttérrendszerek offline idejére)
- Klaszterezés és terhelésmegosztás J2EE alkalmazásszervereken
- Magas tranzakciószámra optimalizált működés (1,4 millió tranzakció / óra)
- Valós 7x24 órás rendelkezésre állás

7. Továbbfejlesztési lehetőségek

A véleményem szerint nincs olyan program, amit ne lehetne tovább fejleszteni. A fejlesztés során sorban jöttek az ötletek, amelyekkel tovább lehet fejleszteni a programot. Ezek nagy része kényelmesebbé, egyszerűbbé tenné a programot, de ezek megvalósítása nem tárgya ennek a dolgozatnak. Néhány továbbfejlesztési lehetőség:

- *Kommentek támogatása:* egy ábra legyen bármennyire is szemléletes magyarázó szövegekkel még szemléletesebbé lehet tenni. A jelenlegi szerkesztő grafikus része nem támogatja ezek megjelenítését. Akár felcímkézett nyilakkal, vagy külön megjelenő elemként a folyamatábra szerves része lehetne.

- *Bővebb „nyelvi" támogatás:* a szerkesztő a tranzakciós XML-ek nyelvi készletének csupán töredékét ismeri, használja. Hogy igazán használható legyen, szükséges lenne a felhasználható elemek bővítése. (Pl.: try catch blokkok, set, backend function, internal function, switch – case, stb.)

- *Nyilak automatizálása:* a felhasználó ideje nagy részét ne azzal töltse, hogy nyilakkal köti össze az egyik elemet a másikkal. Ezt lehetne automatizálni:
    - Ha egy új elemet „dob rá" egy már felvitt elemre, akkor az automatikusan utána fűződne.
    - Ha egy nyíl közepére rakunk egy új elemet, akkor automatikusan ékelődjön be két elem közé.
    - Ha egy elem alá helyezünk el egy másik elemet, akkor automatikusan létrejöhetne közöttük a kapcsolat, vagy ha egy speciális elem mellé. (Pl.: IF ábra mellé rakunk egy Statement ábrát, akkor az ELSEághoz fog tartozni, és össze lehet kötni nyíllal.)

- Kódkiegészítés:
    - Szöveges szerkesztőben: egy billentyűkombinációra egy legördülő listában megjelenhetne az adott helyre beszúrható összes lehetséges elem, amelyből a felhasználó tetszése szerint választhatna.
    - Grafikus felületen: új elem felvitele esetén a property editorban megjelenne az összes lehetséges attribútum. A kötelező attribútumok egy alapértelmezett értékkel kitöltve, az opcionálisak pedig üresen hagyva jelennének meg. Azok, amelyeknek az értéke egy előre meghatározott csoportból kerülhetnek ki, egy legördülő menü segítségével kapnának értéket. Mentéskor csak azok az attribútumok mentődnének el, amelyek kaptak értéket.

- *Hiba ellenőrzés:*
    - Szöveges hiba ellenőrzés: a szöveges szerkesztőben a szerkesztő jelezné (pl.: aláhúzással), hogy hibás vagy hiányos, amit a felhasználó írt. (Pl.: hiányzik egy kötelező attribútum, vagy nincs lezárva egy tag.)
    - Grafikus hiba ellenőrzés: a grafikus szerkesztő jelenleg is tartalmaz megszorításokat a lehetséges nyilakra vonatkozóan (pl. IF elemből nem húzható csak egy TRUE és egy FALSE nyíl), de a hurkok figyelése nincs implementálva. További megszorítások bevezetésével a hibák minimalizálhatóak lennének.

- *Sablonok:* ha a felhasználó egy új fájlt szeretne létrehozni, akkor a szerkesztő felajánl néhány lehetséges előre elkészített sablont. Ezáltal a felesleges „robotmunkát” lehetne csökkenteni, hiszen minden fájl tartalmaz kötelező elemeket.

- *Beállítások:* A jelenlegi program nem tartalmaz személyre szabható beállítási lehetőségeket. Az Eclipse keretrendszer támogatást nyújt ahhoz, hogy az egyes pluginok egyedi beállításokkal rendelkezhessenek felhasználónként. (Pl.: rácsvonal ki és bekapcsolása, mérete, stb.)

- *Nyomtatás és exportálás:* nagy segítség lehet, ha a felhasználónak lehetősége van kinyomtatni vagy exportálni valamilyen képi formátumba az elkészített ábrát.

- *Közvetlen szerkesztés a grafikus felületen:* minden grafikus elem tartalmaz egy ikont, melyre kattintva egy szöveges editor jelenne meg, melyben az elemhez tartozó szöveges kódrészlet lenne látható. A szöveg közvetlenül szerkeszthető, így elkerülhető a váltás a grafikus és a szöveges editor között. Természetesen itt is értelmezettek lennének a kódkiegészítésnél és a hiba ellenőrzésnél leírtak.

- *Kijelölés átvitele a lapok között:* ha a felhasználó kijelöl egy elemet az egyik szerkesztőben és átvált a másikra, akkor az legyen az éppen kijelölt elem a másik szerkesztőben is. (Pl.: kijelölt egy If elemet a grafikus felületen, és átvált szöveges szerkesztőre, akkor a kurzor a szöveges szerkesztőben az If elemhez tartozó kódnál legyen.

- *Nyilak tördelése:* a jelenlegi grafikus szerkesztőben a nyilak a legrövidebb útvonalat keresik meg két pont között, ügyelve, hogy ne keresztezzék se egymást, se az elemeket. Ez kiegészíthető lenne azzal, hogy amikor a felhasználó összeköt két elemet, akkor a nyíl nem a legrövidebb utat mutatná, hanem automatikus töréspontok segítségével még szemléletesebbé tenné az ábrát. (Pl.: nem átlósan kötődne össze két elem, hanem két szakasszal, amelyek egymásra merőlegesek.)

8. Felhasználói kézikönyv

8.1. A program általános ismertetése

Az alkalmazás célja, hogy STS tranzakciós XML fájlokat szerkeszthessünk vele könnyen és egyszerűen. Mindezek mellett demonstrálja a GEF keretrendszer grafikus szerkesztői felületének a lehetőségeit. A felhasználó kétféle nézet közül választhat:

- grafikus nézet (egy folyamatábra)
- szöveges nézet (egy szövegszerkesztő ablak)

Mind a két nézettel ugyanazt az XML dokumentumot szerkeszti a felhasználó.

8.2. A program üzembe helyezése

A plugin futtatásához szükségesek az alábbi programok:

- Java 1.5 vagy újabb verzió
- Eclipse IDE 3.3 vagy újabb verzió

Telepítés:

- A letöltött jar fájlt másoljuk be az eclipse/plugins mappába, majd indítsuk el az Eclipse-t.

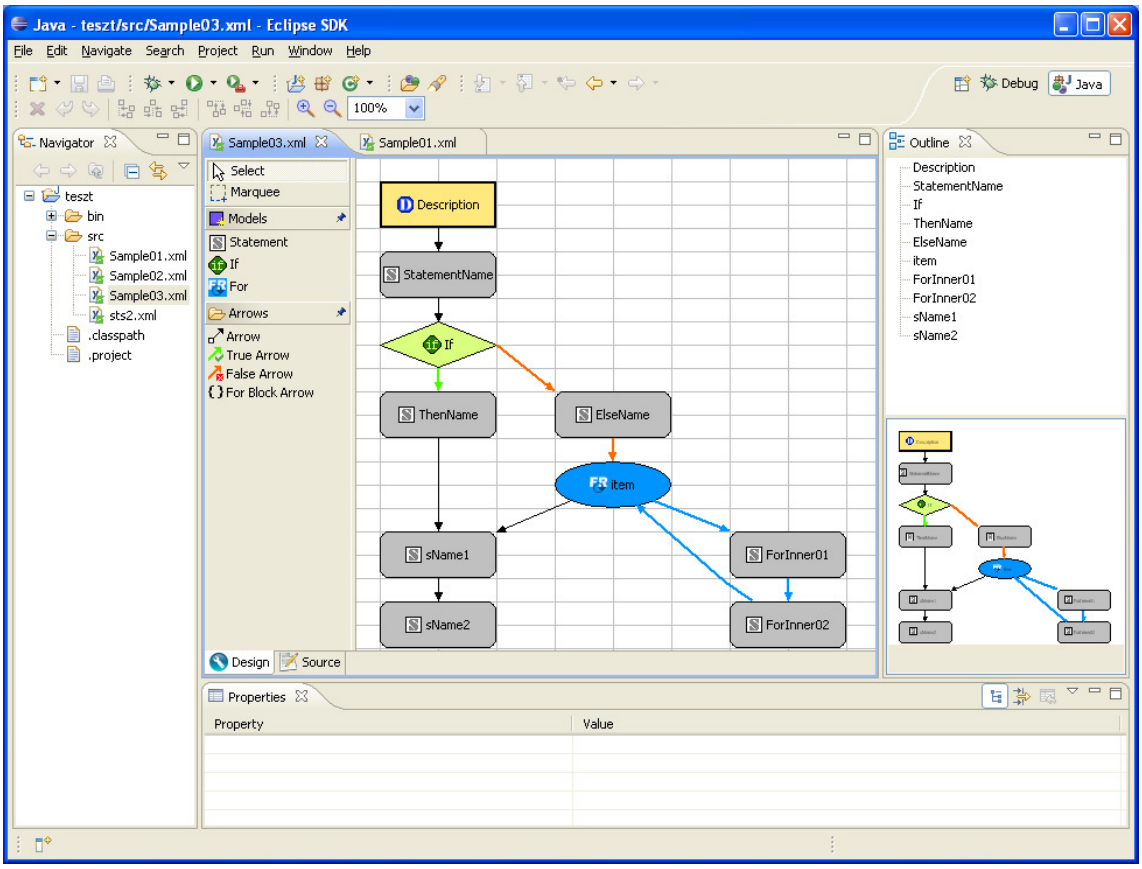
A plugin indítása:

- A telepítést követően, ha az Eclipse-ben megnyitunk egy .xml kiterjesztésű fájlt, akkor az alapértelmezetten az STS XML Editorral fog történni.

8.3. Ismerkedés a programmal

Az itt leírtak nem terjednek ki az Eclipse IDE bemutatására és használatára. Feltételezi, hogy a felhasználó legalább alapfokú ismeretekkel rendelkezik a használatát illetően.

Miután megnyitottunk egy XML dokumentumot, az alábbi látvány fogad minket:



4. ábra Grafikus szerkesztői felület

8.3.1. Eszköztár (Toolbar)

A képernyő felső részén található az eszköztár. A rajta elhelyezkedő elemek kétféle állapotban lehetnek:

- *Aktív* (az ikonok színesek, és ha rákattintunk, végrehajtódik az ikonhoz tartozó parancs.)
- *Inaktív* (az ikonok szürkék és halványak. Kattintásra nem történik semmi.)

Az állapotok esemény vezéreltek. (Pl.: ha nincs kijelölve egyetlen egy elem sem, akkor a törlés ikon inaktív lesz, hiszen nincs mit törölni.)

A grafikus szerkesztőhöz az alábbi eszköztár tartozik:

5. ábra Eszköztár

Az eszköztáron az alábbi elemek találhatóak:

Ikon	Név	Esemény	Mikor aktív?
	Delete (Törlés)	Törli a kijelölt elem(eke)t.	Ha legalább egy elem vagy nyíl ki van jelölve.
	Undo (Visszavon)	Visszavonja a legutóbbi eseményt.	Ha már módosított valamit a felhasználó.
	Redo (Ismét)	Megismétli a legutóbbi eseményt.	Ha már visszavont valamit a felhasználó.
	Align Left (Balra igazít)	Balra igazítja a kijelölt elemeket.	Ha legalább két elemet kijelölt a felhasználó.
	Align Center (Középre igazít)	Függőlegesen középre igazítja a kijelölt elemeket.	Ha legalább két elemet kijelölt a felhasználó.
	Align Right (Jobbra igazít)	Jobbra igazítja a kijelölt elemeket.	Ha legalább két elemet kijelölt a felhasználó.
	Align Top (Fentre igazít)	Fentre igazítja a kijelölt elemeket.	Ha legalább két elemet kijelölt a felhasználó.
	Align Middle (Középre igazít)	Vízszintesen középre igazítja a kijelölt elemeket.	Ha legalább két elemet kijelölt a felhasználó.
	Align Bottom (Lentre igazít)	Lentre igazítja a kijelölt elemeket.	Ha legalább két elemet kijelölt a felhasználó.
	Zoom In (Nagyítás)	Nagyítja az ábrát.	Ha az ábra 2000%nál kisebb nagyítással van megjelenítve.
	Zoom Out (Kicsinyítés)	Kicsinyíti az ábrát.	Ha az ábra 25%nál nagyobb nagyítással van megjelenítve.
100%	Custom Zoom (Egyedi zoom)	A menüből kiválasztott %-nyira nagyítja az ábrát.	Mindig aktív.

3. Táblázat Eszköztár elemei

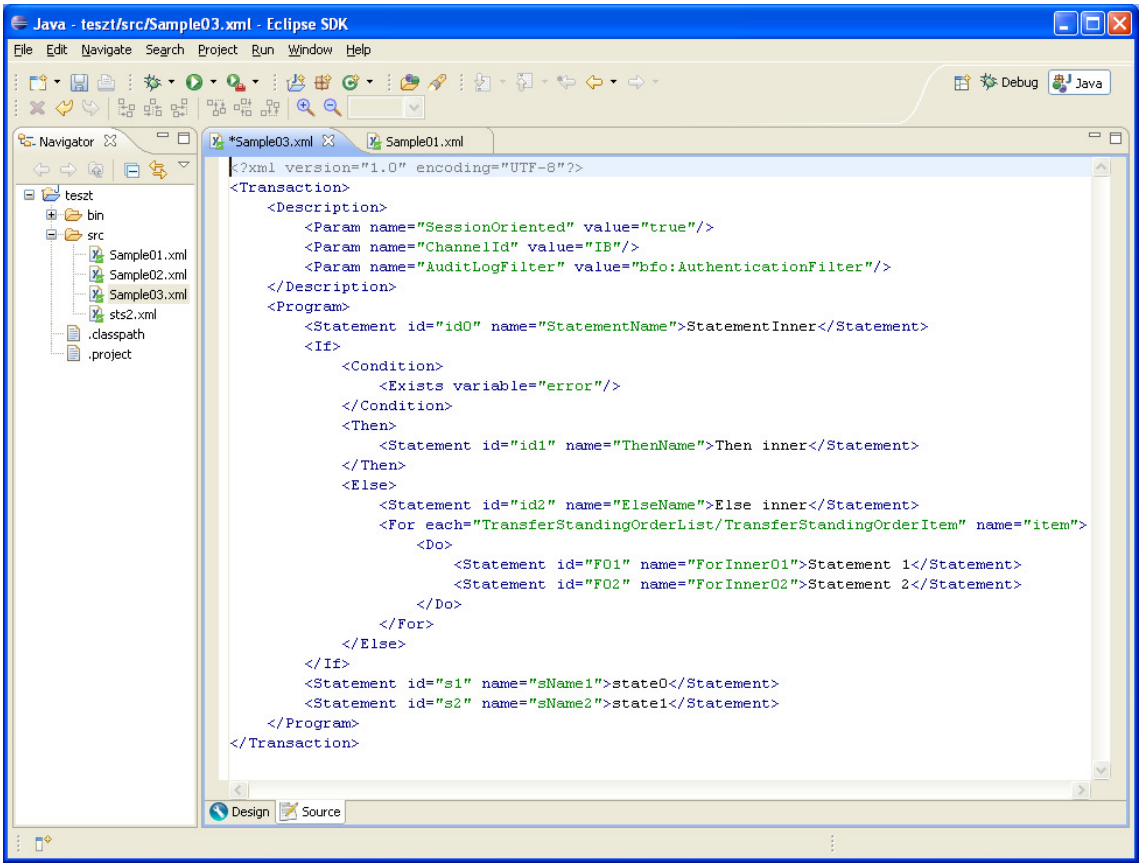
8.3.2.Fülek (Tabok)

A fülek segítségével tudunk váltani a grafikus és a szöveges szerkesztő között. A szerkesztő bal alsó sarkában található.

6. ábra Fülek (Tabok)

8.3.3.Szöveges szerkesztő

Egyszerű szövegszerkesztő műveleteket végezhetünk el. (Pl.: módosítás, kivágás, másolás, beillesztés, stb.) A szerkesztő szintaktikai kiemelést végez. (Kiszínezi a szöveget a jobb áttekinthetőség végett.)



7. ábra Szöveges szerkesztő

8.3.4. Grafikus szerkesztő (Design Editor)

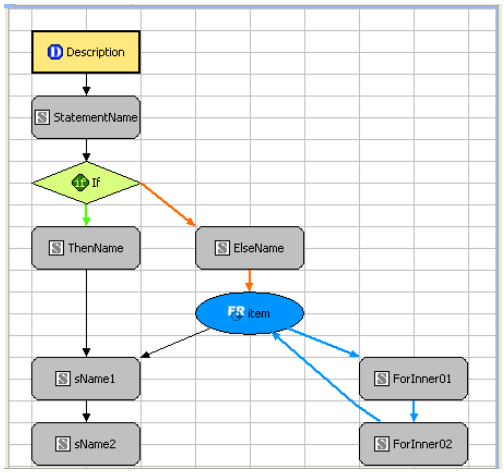
Két fő részre van osztva:

- *Paletta*: bal oldalt található. Hasonló szerepet tölt be, mint az eszköztár. Ha egy ikont kiválasztunk, akkor azzal tudunk a szerkesztő felületre (vászonra) új dolgokat felvinni, vagy csak módosítani az eddigieket. Az ikonok sohasem inaktívak (ellentétben az eszköztárral) és egy elem mindig ki van választva.

Ikon	Név	Esemény
Select	Kiválasztás	Kiválaszthatunk, átméretezhetünk, mozgathatunk egy vagy több elemet.
Marquee	Kijelölés	Kijelölhetünk egy vagy több elemet, nyilat a vásznon
Models	Modellek főcsoport	Megjelenik a modellek palettája.
Statement	Statement	Ha a vászonra kattintunk, hozzáad egy új Statement elemet.
If	If	Ha a vászonra kattintunk, hozzáad egy új If elemet.
For	For	Ha a vászonra kattintunk, hozzáad egy új For elemet.
Arrows	Nyilak főcsoport	Megjelenik a nyilak palettája.
Arrow	Nyíl	Egyszerű nyilat rajzolhatunk két elem közé.
True Arrow	Igaz nyíl	Igaz nyilat rajzolhatunk két elem közé.
False Arrow	Hamis nyíl	Hamis nyilat rajzolhatunk két elem közé.
For Block Arrow	For blokk nyíl	Blokk nyilat rajzolhatunk két elem közé.

4. Táblázat A grafikus szerkesztő palettája

- Szerkesztő felület (vászon): jobb oldalt helyezkedik el. Rácsozott. Az elemek a rácsokhoz lesznek igazítva. A paletta segítségével tudunk rá rajzolni, vagy módosítani a tartalmát.

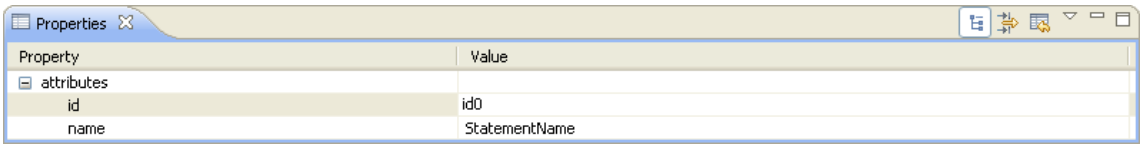


8. ábra A grafikus szerkesztő vászna

8.3.5. Tulajdonság szerkesztő (Properties)

Ha a tulajdonság szerkesztő nem látható, akkor a menüben navigáljunk el a következő helyre: *Window -> Show View -> Other...* menüponthoz, majd a felugró ablakban keressük meg a *Properties*-t és kattintsunk az *OK* gombra.

Ha a grafikus szerkesztőben kiválasztunk egy elemet, és az rendelkezik tulajdonságokkal, akkor azok megjelennek a Propertiesben. Az értékek módosíthatóak.



9. ábra Tulajdonság szerkesztő (Properties)

8.3.6. Vázlat megjelenítő (Outline view)

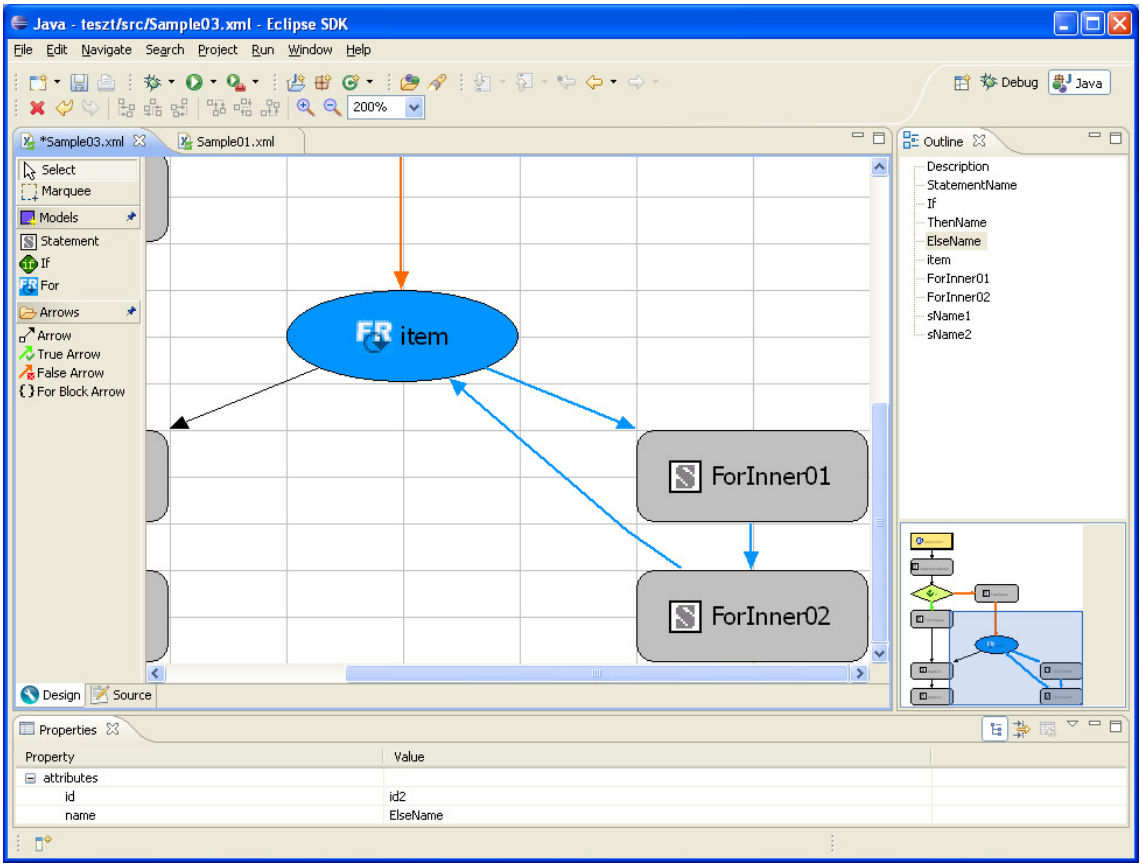
Ha az Outline nem látható, akkor a menüben navigáljunk el a következő helyre: *Window -> Show View -> Outline* menüponthoz, majd kattintsunk rá.

Két részből áll:

- *Fa nézet*: az outline felső részében található. Az összes elem fel van sorolva ebben a nézetben. (Ha egymásba ágyazhatóak lennének az elemek, akkor nem csak egy

egyszerű lista lenne, hanem itt is látszana az egymásba ágyazottság.) Ha itt kijelölünk egy elemet, akkor az a grafikus szerkesztőn is kijelölődik.

- *Vázlat*: Az alsó részben található. Segítségével a grafikus szerkesztőn navigálhatunk gyorsan és könnyedén, ha az fel van nagyítva, és nem fér ki a képernyőre, ahogy a lenti ábrán is látható. Ekkor egy téglalap jelenik meg a vázlaton, ami az éppen látható részt mutatja. Ha az egérrel mozgatjuk a vázlaton, akkor a grafikus szerkesztő nézete is változik ennek megfelelően.



10. ábra Navigálás Outline segítségével

9. Zárógondolatok

A program elkészítése során elsődleges célom az volt, hogy egy alap funkciókkal rendelkező, de használható alkalmazást készítsek, amellyel demonstrálni lehet a grafikus fejlesztői felületben rejlő további lehetőségeket. Úgy érzem, ezt a célt sikerült elérnem.

Visszagondolva a szakdolgozat elkészítésével töltött időre, hasznosnak és tanulságosnak tartom. Megismerkedtem számos új technológiával amelyeknek - reményeim szerint - később még nagy hasznát fogom venni. A GEF egy jól átgondolt és megtervezett keretrendszer, amely kiválóan alkalmas az ilyen jellegű fejlesztésekre. Egyetlen negatívummal találkoztam csupán: a dokumentáltsága nem a legjobb, de remélem ez a közel jövőben változni fog.

Most, a fejlesztés végén úgy érzem, hogy ez nem valaminek a végét jelenti. Biztos vagyok benne, hogy a plugin további fejlesztésére még sok időt fogok fordítani, hisz a szívemhez nőtt.

Az Eclipse pluginok készítésével személyes igényeinkre szabhatjuk a fejlesztői környezetünket, ezáltal is megkönnyítve a munkánkat. Mindenkinek csak javasolni tudom, hogy ismerkedjen meg a plugin készítés rejtelmeivel.

10. Köszönetnyilvánítás

Legelőször is Sallai Krisztián Zsoltnak mondok köszönetet, aki értékes szakmai segítséget nyújtott és mindig kész volt időt szakítani a kérdéseim megválaszolására. Dolgozatom megírásához sok szakmai segítséget kaptam Juhász Istvántól, az egyetemi témavezetőmtől, amit ezúton szeretnék megköszönni. Hálás vagyok a közvetlen munkatársaimnak, hogy türelmükkel és rugalmasságukkal hozzájárultak a szakdolgozatom megírásához. Nagy köszönettel tartozom szüleimnek, a sok esztendőn át tartó lelki, erkölcsi támogatásukért. Végül, de nem utolsó sorban hálás vagyok mindazoknak, akik javaslataikkal, észrevételeikkel vagy bármilyen más módon hozzájárultak e szakdolgozat megírásához.

11. Irodalomjegyzék

[1] Eric Clayberg and Dan Rubel: *Eclipse Building Commercial-Quality Plug-ins,* Kiadó: Addison-Wesley, 2004

[2] IND Secure Transaction Server 2005 (STS)
http://www.indgroup.hu/termekek/ind_sts_2004/index.ind

[3] Wikipedia: *Standard Widget Toolkit (SWT)*
http://hu.wikipedia.org/wiki/Standard_Widget_Toolkit

[4] Balogh András: *Nyílt fejlesztőrendszerek* BME 2006 előadás jegyzet
http://home.mit.bme.hu/~abalogh/

[5] Sallai Krisztián Zsolt: *Eclipse tréning* ME 2007
előadás jegyzet

[6] Nyékyné G. Judit: *Java 2 útikalauz programozóknak 1.3,*
Kiadó: ELTE TTK Hallgatói alapítvány, Budapest

[7] Wikipedia: *Java (programozási nyelv)*
http://hu.wikipedia.org/wiki/Java_programoz%C3%A1si_nyelv

[8] Czinkos Zsolt: *Mi az az XML?*
http://www.xmlinfo.hu/XML/miazazxml.html

[9] Dr. Mester Gyula: *Az XML nyelv alapjai*
http://www.mk.u-szeged.hu/~gmester/XML_Web.pdf

[10] *Observe Eclipse*
http://www13.plala.or.jp/observe/index.html#gef

[11] Wikipedia: *Eclipse*
http://hu.wikipedia.org/wiki/Eclipse

[12] Fábián Zoltán: Programozás jegyzet
http://fabionet.extra.hu/download/download.php?fname=./Programozas.pdf