Debreceni Egyetem Informatikai Kar

Információ Technológia Tanszék

Beépülő modul tervezése Eclipse-hez

Témavezető:

Espák Miklós

egyetemi tanársegéd

Készítette:

Tóth Gábor,
Zsenyuk Tamás

programtervező informatikus hallgatók

Debrecen
2009

Tartalomjegyzék

1. Bevezető

1.1. A célkitűzések megfogalmazása

Feladatunk az Eclipse fejlesztői környezet Visual Editor nevű grafikus felhasználói interfész létrehozására szolgáló bővítő moduljának (bővítményének) kiterjesztése. Az Eclipse egy platform-független szoftverfejlesztő eszköz, amely az ECL (Eclipse Public License) értelmében szabad, és nyílt forráskódú szoftver. A fejlesztői környezetet az Eclipse Foundation a komponens alapú tervezési minta alapján alkotta meg.

A kiterjesztő osztályok a HiberGUI projekt bizonyos komponensei lesznek, amelyek szerkezetét, és működését a következő fejezetben mutatjuk be. A Visual Editor modul elsődleges példányosító eszköze az úgynevezett paletta. Célunk, hogy a palettában megjelenjen egy HiberGUI nevű palettakategória, és benne az egyes osztályoknak megfelelő palettabejegyzés, amely által példányosíthatóvá válik az adott osztály. Ezt a funkcionalitást egy bővítmény implementálásával és telepítésével érhetjük el.

További feladatunk annak megvalósítása, hogy a létrehozott bővítmény – a legtöbb Eclipse modulhoz hasonlóan, – tároló szerveren elérhető legyen az Eclipse számára. Ehhez egy frissítéseket tartalmazó oldal, úgynevezett *Eclipse Update Site* létrehozása, és beállítása szükséges.

1.2. A témaválasztás okai

Egy programozónak alaposan kell ismernie azokat a hardver és szoftver eszközöket, melyekkel munkája során dolgozik. Hiányos ismeretek esetén a programozói munka, a szoftvertervezési és -fejlesztési folyamat jelentősen lassulhat, vagy el is akadhat. Ezt szem előtt tartva, témaválasztásunk elsődleges célja az volt, hogy új ismeretekre tegyünk szert,

illetve elmélyítsük tudásunkat az Eclipse fejlesztői környezet szerkezetére, működésére és lehetőségeire vonatkozóan. Második célkitűzésünk pedig az, hogy - tudásunkhoz mérten - az Eclipse IDE kiterjesztésével hozzájáruljunk az eszköz funkcionalitásának és használhatóságának növekedéséhez. Természetesen a témaválasztásnak volt egy harmadik, és egyben igen fontos oka, méghozzá az, hogy a Java nyelvet tartjuk a legmodernebb, és a legdinamikusabban fejlődő magas szintű programozási nyelvnek, amely a szoftverfejlesztés bármely területén alkalmazható, az adatbázismanipulációtól, a mobiltelefonos alkalmazásokig. A Javat megkedveltük, szívesen alkalmazzuk, és nagy kedvvel választottunk olyan szakdolgozati témát, amelynek alapja a Java technológia.

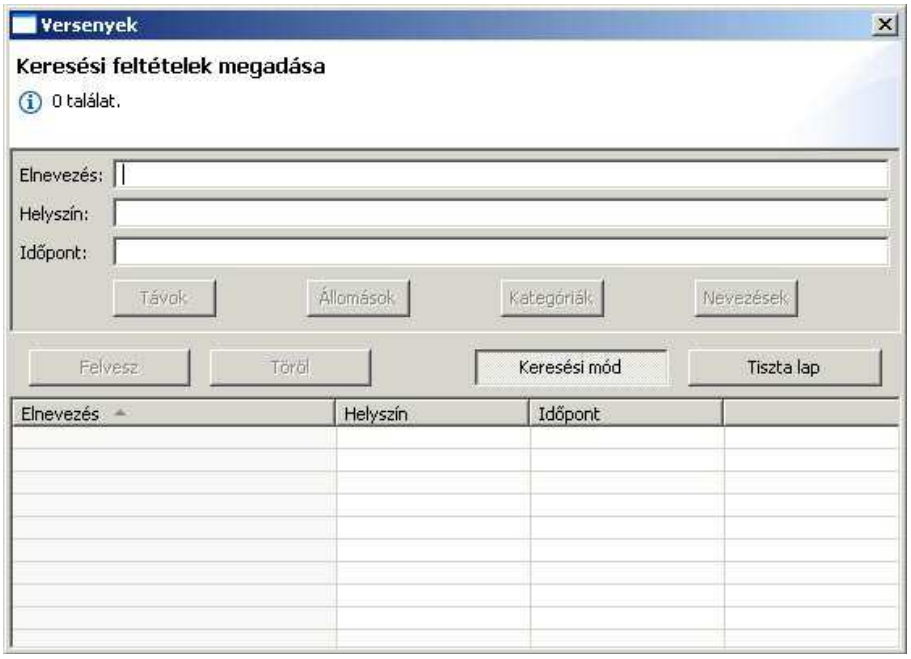
1.3. HiberGUI

A HiberGUI egy SWT alapokon nyugvó adatbázis-manipulációs eszköz, amely ötvözi az SWT használhatóságát és hordozhatóságát a nagyteljesítményű objektumrelációs perzisztencia szolgáltatással. Segítségével a fejlesztő felhasználói felületet készíthet egy adatbázis alapú rendszerhez.

A HiberGUI legfontosabb osztálya a HibernateForm osztály, amely az absztrakt Form osztály leszármazottja. A Form a rendszer egyik alappillére. Ez felel a felhasználói interakciók figyeléséért és a grafikus nézeten megjelenő HibernateForm szerkezetéért. A HibernateForm örökölt tulajdonsága ugyanis, hogy három részből épül fel. Az alábbi ábrán is látható módon, legfelül a `createFields` metódus által definiált elemek vannak. Alattuk a `create`, `delete`, `clear` és `search` gombok helyezkednek el, a harmadik részben pedig egy tábla látható, amely arra szolgál, hogy a felhasználó a számára szükséges elemeket kiválaszthassa.

A készülő form rugalmasságának megőrzése érdekében a `createFields` metódust a HibernateForm osztály valósítja meg. Használatával testre szabhatóvá válik a form, hiszen az itt létrehozott elemek meg fognak jelenni a felső harmadban. További felüldefiniálandó metódus a `createColumnsForTable`, amellyel a táblázat

szerkezetét adhatjuk meg. A HibernateForm felel a munkamenet-, és tranzakció-kezelésért, ennek megfelelően felüldefiniálja a Form osztály ezért felelő metódusait is.



1. ábra: egy HiberGUI-val létrehozott form

A rendszer másik pillérét az absztrakt Edit osztály jelenti. Ezen osztály leszármazottai teszik lehetővé az adatbázisban tárolt entitások tulajdonságainak megváltoztathatóságát. Az Edit osztály leszármazottainak konstruktorai négy paramétert várnak: az első az, az SWT kompozit, amelynek része, a második, a megjelenést befolyásoló konstans, a következő egy tulajdonságnév, amelyet kezelnie kell, és az pedig utolsó az elem neve. A leszármazottakat két csoportra oszthatjuk. Amíg a `property` csomagban lévő osztályok, az entitások primitív típusú tulajdonságainak módosítására valók, addig az `assoc` csomagban lévők az entitások kapcsolatainak kezelését szolgálják.

2. Grafikus komponensgyűjtemények

2.1. AWT

Az első grafikus interfész komponenskészlet az AWT (Abstract Widgeting Toolkit) volt. Natív, vagyis az operációs rendszer grafikus adottságait használja fel. Ha AWT-t használunk alkalmazásunk grafikus felületének megépítésekor, akkor annak megjelenése operációs rendszerenként más és más lehet. Egyszerű eseményrendszert használ, amely nem felelt meg a Java platform független filozófiájának, ezért továbbfejlesztették, amelynek eredménye a Swing lett.

Komponensgyűjteménye igen szegényes, csak azok a komponensek használhatók fel, amelyek minden operációs rendszeren elérhetők. Minden grafikus komponensének ősosztálya a `java.awt.Component`, míg minden konténerének a `java.awt.Container`. A konténerekben helyezhetők el egyszerűbb komponensek, amelyek elhelyezkedéséről elhelyezési stratégiák döntenek. Például:

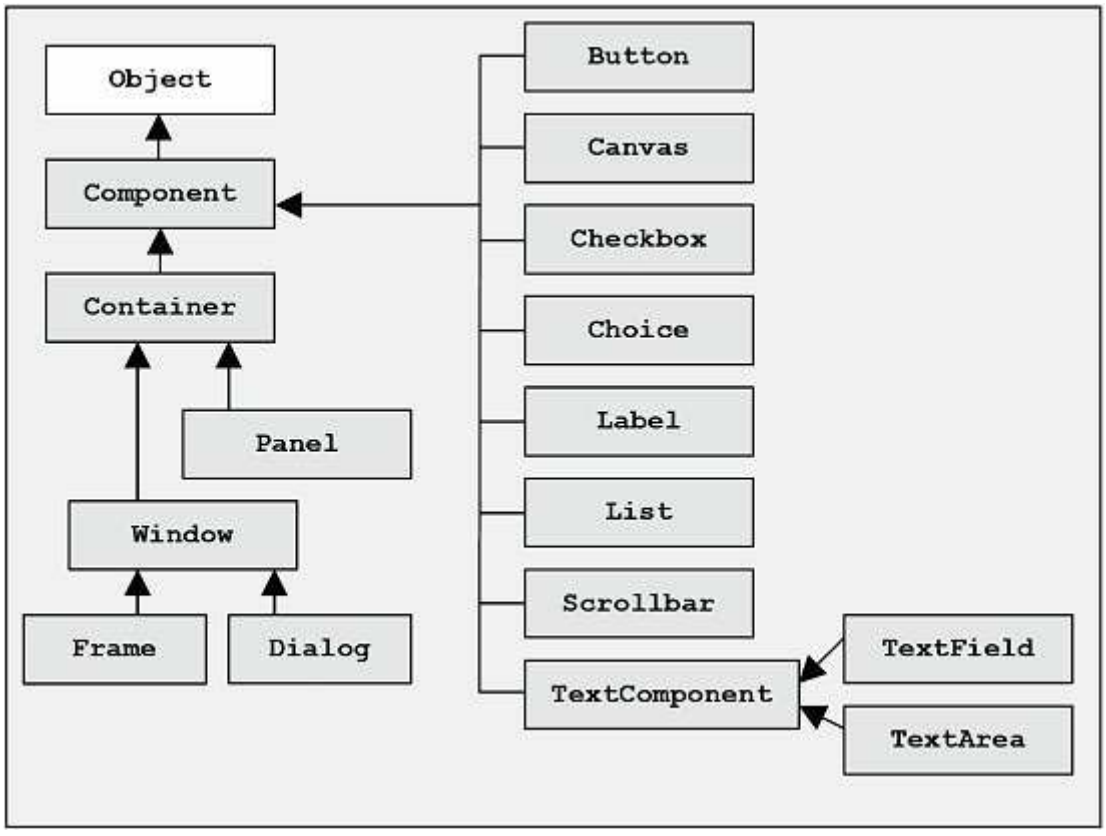
- FlowLayout
- BorderLayout
- CardLayout
- GridLayout

A legegyszerűbb konténer a `java.awt.Panel`, a rajta lévő komponensek tárolásáért és megjelenítéséért felelős. Az AWT-ben lehetőségünk van legördíthető panel létrehozására is: `java.awt.ScrollPane`. Ha a ráhelyezett elemek nem jeleníthetők meg egyszerre, akkor automatikusan előkerülnek a szükséges gördítő sávok. Ha egy ablakot szeretnénk létrehozni, akkor a `java.awt.Window` segítségével ez könnyen megtehető. Ez egy önmagában működni képes ablakot hoz létre. A `java.awt.Frame` a `Window` leszármazottja. Fejléccel és kerettel ellátott ablakok példányosíthatók vele. A `Window` egy másik leszármazottja a `java.awt.Dialog`, ami egyszerűbb adatbevitelre alkalmazható ablakot hoz létre. Lehetőségünk van arra is, hogy weboldalba

ágyazzunk alkalmazást, ekkor a `java.awt.Applet` osztályt kell felhasználnunk. A konténerelemek is a `Component` osztály leszármazottai, így ugyanazokkal a tulajdonsággal rendelkeznek.

Egyszerű komponensek:

- `Label`: nem módosítható szöveg megjelenítésére szolgál
- `Button`: egyszerű gombot reprezentál
- `Choice`: legördülő lista hozható létre vele
- `List`: hasonló a `Choice`-hoz de több elem kiválasztására is van lehetőség
- `TextField`: egysoros, egyszerű szövegbeviteli mező
- `TextArea`: többsoros szövegbevitelre ad lehetőséget



2. ábra: az AWT osztályhierarchiája

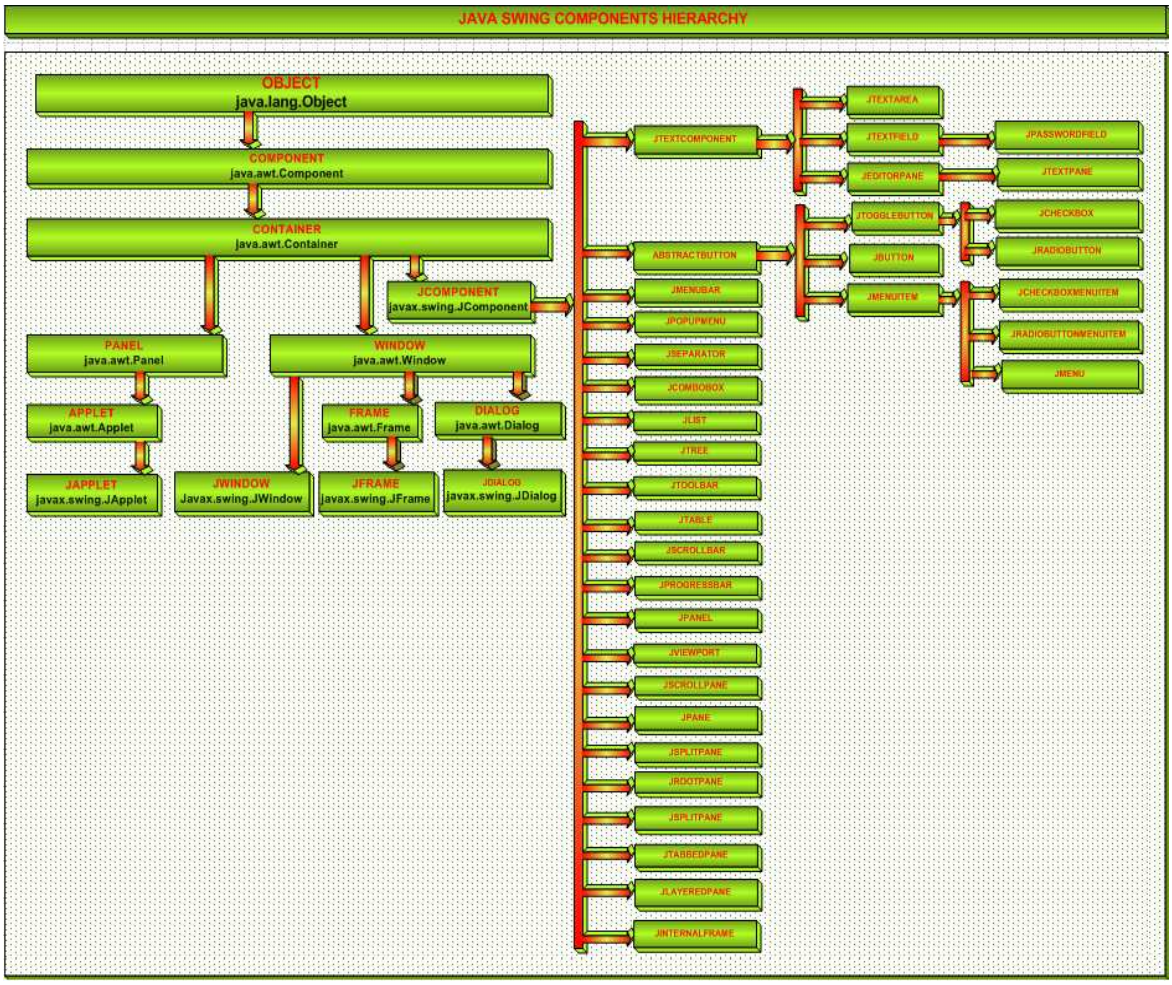
2.2. Swing

Grafikus felhasználói felületek készítésére való. Az AWT-re épül, de nem használja az operációs rendszerek nyújtotta elemeket. Az elemei teljesen Javában megírt elemek, emiatt egy Swing alapú felhasználói felülettel rendelkező alkalmazás minden platformon ugyanolyan kinézetű. A kinézetek bizonyos stílusok révén változtathatók, ez az úgynevezett „*Look & Feel*”. Ez egy olyan mechanizmust valósít meg, hogy a stílus átállításával, az elemek megjelenésén könnyedén végbe menjenek a változások, ne kelljen a kódot a fejlesztőnek jelentősen átalakítania. A Java Standard Edition-ben az 1.2 -es verzió óta alapelem, elérhetősége a csomaghierarchiában: `javax.swing`. Megvalósítja az *MVC*-t (*Model-View-Controler*) mintát.

A Swing

- bővíthető: saját elemekkel felül lehet definiálni a már meglévőket, illetve a beépített interfészeket is lehet implementálni elemek fejlesztésénél,
- testre szabható: az egyes elemek tulajdonságai (például: hátterek, szegélyek színe) könnyen módosíthatóak,
- konfigurálható: a futás közbeni változásokra is könnyen tud reagálni, például megváltoztatható a stílusa.

A Swing hátterében az AWT áll. Ez megfigyelhető egyes elemeknél, ahol minden AWT elem neve elé a Swingben kerül egy „J”, továbbá az eseménykezelés is az AWT csomagból való.



3. ábra: A Swing osztályhierarchiája

2.2.1. Swing felépítése

Minden Swing GUI komponens őse a `javax.swing.JComponent`. Ebben vannak implementálva azok a metódusok, amelyek az eseménykezelésért, előtér és háttérszín és a saját pozíció lekérdezéséért, vagy területük felett az egér alakjának változtatásáért felelősek.

Négy Swing elem, aznoban a `JComponent` osztályból származik:

- a **JApplet**, amely egy Java alkalmazást hoz létre egy Java-t futtatni képes böngészőben.

- a **JDialog**, amellyel dialógusablakot tudunk létrehozni.

- a **JFrame**, amely kerettel és fejléccel ellátott ablakot hoz létre. A `java.awt.Frame` kiterjesztése.

- a **JWindow**, amellyel keretek nélküli ablak készíthető. A `java.awt.Window` kiterjesztése.

A komponensek csoportosíthatók funkció, és feladataik alapján is.
A komponensek funkció szerint lehetnek

- **konténerek**, amelyek különböző módon csoportba foglalnak egyszerű elemeket, vagy újabb konténereket, így egymásba lehet skatulyázni több konténert is. Ilyen konténer például az ablak .
- **egyszerű elemek**, amelyek a primitív komponensek, például a nyomógomb és címke.
- **összetett elemek**, amelyek több primitív elemből építkeznek, például a táblázatok és a fa-struktúrák.
- **származtatott elemek**, amelyeket magunk készítünk egy már meglévő komponensből, kiterjesztéssel.
- **segédelemek**, amelyek ugyan nem láthatóak, de mégis ott vannak a grafikus felületen. Ilyenek például a nyomógombokat csoportba foglaló komponensek, az elrendezés kezelők, illetve a betűtípusok és a színek is.

Az egyszerű elemeket feladatuk alapján is csoportosíthatjuk:

- **beviteli** komponensek, amelyek adatok bevitelét teszik lehetővé.
- **vezérlő** komponensek, amelyekkel a program futását tudjuk irányítani.
- **kiviteli** komponensek, amelyekkel adatokat lehet megjeleníteni a felhasználók számára.

2.3. SWT

Az Eclipse grafikus felülete SWT (Standard Widget Toolkit) alapú. Az SWT IBM fejlesztette ki az Eclipse megalkotásakor, amely bővebb komponenskészlettel rendelkezik az AWT-nél. Az Eclipse felülete használja továbbá a JFace köztes GUI réteget is, az egyszerűbb SWT felületű alkalmazás-készítés miatt.

Az SWT a Java ablakkezelésére és felhasználói felület létrehozására szolgáló komponensgyűjtemény. Natív, ez okozza a gyorsaságát. Támogatja a *drag-and-drop* módszert, segítséget nyújt, (pl.: a nyomtatásban). Az SWT jellemzői:

- átlátható API
- hierarchikus felépítés
- egyszerű komponensek
- jól elkészített eseménykezelés
- saját eszközökkel bővíthető

Az SWT elemei két nagy csoportba oszthatók:

- egyszerű alapelemek
- összefogó elemek

Egyszerű alapelemek például a gombok és a szövegcímkék. Ezek egy csoportját tartalmazó egységek az összefogó, vagy `Composite` elemek. Ilyen például: a `Shell`. A `Composit`-on belül elhelyezhető `Layout` is, amely az elemek belső elhelyezkedésének beállításához nyújt segítséget. Például: `RowLayout`, `FormLayout`, `GridLayout`.

Néhány fontosabb elem:

- A `Shell` egy ablakot reprezentál, melyre `Composite`-ok és `Control`-ok helyezhetők.
- A `Control` egy operációsrendszer szintű vezérlőt jelent.
- A `Button` a különféle gombok készítésére való.

- A `Label` szöveg vagy kép megjelenítésére használható.
- A `Canvas` rajzoláshoz használt osztály.
- A `Menu` menük kialakítására.
- A `Browser` az operációs rendszer böngészőmotorját használja.

Minden elem a `Widget` ősosztály leszármazottja. A grafikus elemek általános létrehozásához szükséges konstruktor is itt van implementálva, amelyben az elem ősét és stílusát szükséges megadni (`new Widget(parent, style)`). Ez a konstruktor azonban (a `Widget` osztály absztrakt mivolta miatt) soha sem hívódik meg közvetlenül.

Az SWT kommunikáció esemény-modellen alapul. A felhasználó vagy az operációs rendszer által generált történések az események, amelyeket a `Listener`-ek dolgoznak fel. Az események feldolgozása típusos és típus nélküli `Listener`-ekkel is történhet. Minden eseményhez tartozik egy osztály, ami az eseményről tárol információkat, egy Listener interfész és egy metódus a regisztrálásához. Ha a Listener több metódust is definiál, akkor egy adapter osztály is hozzá rendelődik. Az eseményekhez kapcsolódó interfészek és osztályok org.eclipse.swt.events csomagban találhatók meg.

A SWT háromféle kivételt tartalmaz a hibák jelzésére:

- `IllegalArgumentException` a hibás bemeneti paraméterek jelzésére
- `SWTException` a hibakódot, a kivételt, amely a hibát okozta és a hibaüzenet a paraméterei. Nem kritikus hiba.
- `SWTError:` Kritikus hiba.

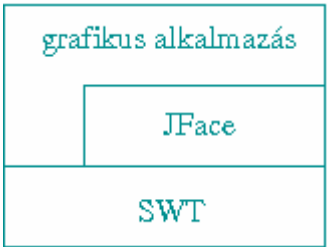
Ha egy SWT elemet törölnünk kell, figyelnünk kell annak leszármazottaira, mert azok is törlődni fognak. Például, ha a `Shell dispose` metódusát meghívjuk, akkor a `Shell-`en lévő elemek is törlődnek.

A billentyűzetről is lehet eseményeket fogadni. Ezek ilyenkor ahhoz a `Control` elemhez kerülnek, amelyek ki vannak jelölve. A gombok lenyomását, felengedését, és mutatóeszközök eseményeit is meg lehet figyelni. Az aktuális pozíciót a kurzor jelzi, amelynek a koordinátái a `Control`-okhoz képest relatívak.

Az SWT, a *JNI(Java Native Interface)* segítségével hívja meg az operációs rendszerhez tartozó SWT API-ját.

2.4. JFace

A JFace az SWT-re épül, azonban jóval strukturáltabb, magasabb szintű komponenseket szolgáltat. Jobban automatizált és könnyebben újrafelhasználható. Az ábrán is látható módon a JFace szolgáltatásait használó alkalmazások azonban az SWT réteget közvetlenül is elérhetik:



4. ábra: a JFace beágyazódása a grafikus alkalmazásba

2.4.1. Akciók

A JFace egy absztraktabb eseménykezelő modellt használ, ennek neve, az *Action-Contribution model*. Az SWT eseményrendszerét (event) a JFace Action-ok foglalják magukba. Ebben a rendszerben egy eseményhez, csak egy eseménykezelő tartozhat, és a komponensekhez csak egy esemény kapcsolódhat. Célja, hogy rendszerezze, ezáltal

használhatóbbá tegye az SWT eszközeit. Saját *Action* osztályokat hozhatunk létre `org.eclipse.jface.action.Action` absztrakt osztály leszármazottjaként. A legfontosabb felüldefiniálandó metódus a `run()`, amelyben a felhasználótól érkező interakcióra válaszként, a szükséges tevékenységeket definiálhatjuk.

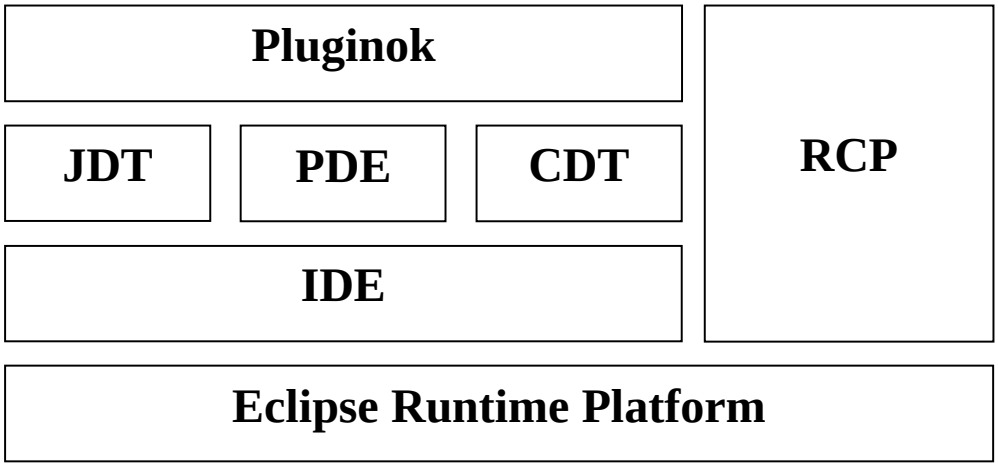
2.4.2. A Viewer keretrendszer

A JFace *viewer* objektumai az SWT táblázat, lista, fa és egyszerű szöveg megjelenítésére szolgáló komponensek tulajdonságait szervezik rendszerbe. A keretrendszer megalkotásának célja az, hogy az említett eszközök újrafelhasználhatóságát javítandó, MVC architektúrába foglalja őket. Ennek eredményeként az eszközök megjelenítése, tartalmuk és a vezérlésük teljes mértékben önálló egységként kezelhető. Lehetőségünk van a megjelenített információk szűrésére, rendezésére.

3. Az Eclipse

Az Eclipse egy platform-független szoftverfejlesztő eszköz, amely az ECL (Eclipse Public License) értelmében szabad, és nyílt forráskódú szoftver. A fejlesztői környezetet az Eclipse Foundation, a komponens alapú tervezési minta alapján alkotta meg.

Ebben a fejezetben az Eclipse modul fejlesztési folyamata szempontjából lényeges architekturális egységeket foglaljuk össze.



5. ábra: az Eclipse architektúrája

3.1. Eclipse Runtime Platform

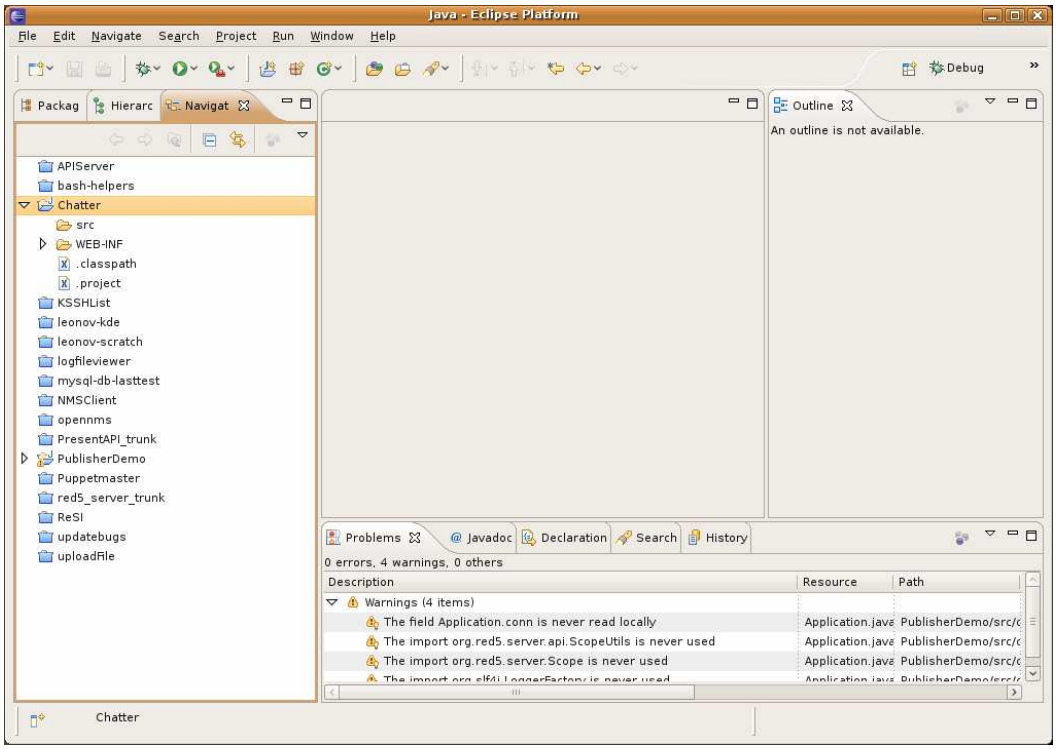
Feladata a rendszer indítása, és futási időben annak menedzselése. Ez az eszköz felel az installált modulok nyilvántartásáért, és azok betöltéséért. Induláskor csak a szükséges bővítmények töltődnek be, ezt *lusta betöltés*nek nevezzük, célja a rendszer indulási idejének csökkentése. Miután a bővítmények detektálása és betöltése megtörtént, elindul a munkaasztal. Ez nem más, mint SWT elemek összessége. Bővítmény fejlesztésekor a tesztelő lépésekben két darab munkaasztallal, és két munkaterülettel dolgozik. Ez egy

tesztelési környezetet alkot, így félkész modulunk nincs hatással az éles rendszerre, hiszen megfelelő módon el van különítve attól.

Az Eclipse Runtime Platform feladata még platform-független módú erőforrás-kezelés, különös tekintettel a munkaterületre, amely egy egyszerű könyvtár az Eclipse projektjeink számára. A rendszer legelső futtatásakor meg kell adnunk ennek útvonalát.

3.2. Integrált fejlesztői környezet

Az integrált fejlesztői környezet (Integrated Development Environment - IDE) kezeli a rendszerben a munkaasztalt felépítő nézeteket (view), perspektívákat (perspective). Egy nézet nem más, mint egy ablak, benne bizonyos információkkal. Lehetőségünk van a nézetek összetételének és helyének megváltoztatására, így a fejlesztés során csak a számunkra fontos információkat jeleníthetjük meg. A nézetek összetételét perspektívákba menthetjük, illetve rendelkezésünkre állnak a rendszer alapértelmezett perspektívái is. Ilyen beállított perspektívája a modulépítés folyamatának is van. Célszerű tehát a kiegészítés-fejlesztés megkezdése előtt ezt beállítani. Az alábbi képen az Eclipse keretrendszer munkafelülete látható Ubuntu operációs rendszerben.

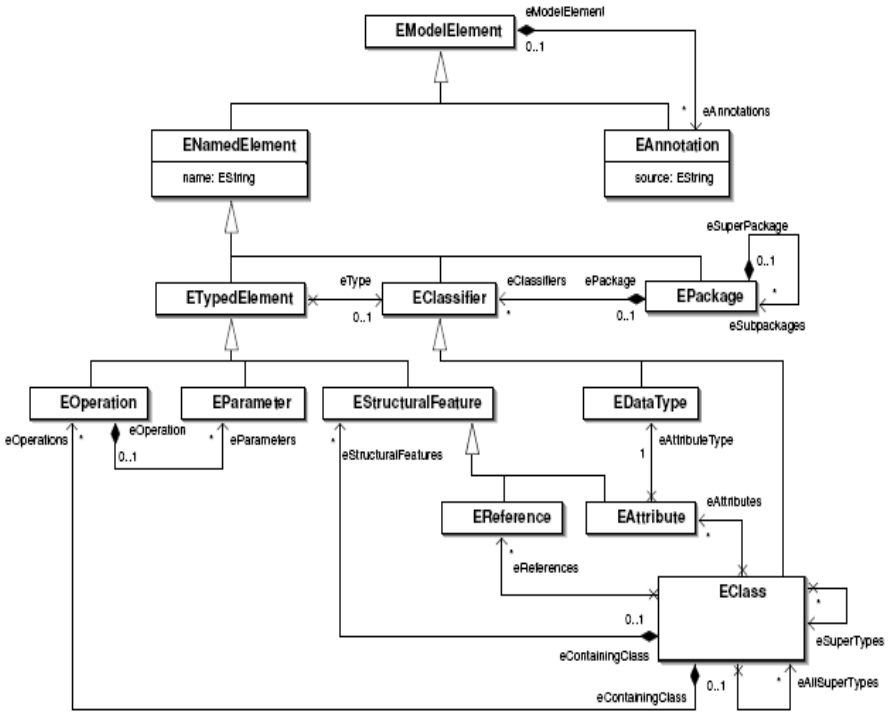


6. ábra: az Eclipse munkaasztala

3.3. Az Eclipse Modelling Framework (EMF)

Az Eclipse modulárisan felépített keretrendszer. Az Eclipse Modelling Framework (EMF) egy modellezési komponens az Eclipse-ben, amellyel modellt lehet fejleszteni, és ebből kódot generálni. Az UML és XML modellkészítési szabvány mellé kínál eszközt, valamint Javában implementációt.

A modell az EMF-ben az Ecore-Meta-modellen alapszik, amelyet az alábbi ábra szemléltet.



7. ábra: EMF ecore-meta-modell

A meta-modellek maguk is XML fájlok. Minden *ecore* fájl egy csomag definíciójával kezdődik, amely a többi modellelemet tartalmazza. Létrehozásához az Eclipse *ecore* editorokkal rendelkezik.

Egy osztály attribútumai és referenciái `EAttributte` illetve `EReference` tagként ábrázoljuk. Mind a kettő egy absztrakt típustól, az `EStructuralFeature` osztályból öröklődnek. Egy osztály metódusai és ezek paraméterei az Ecore-Metamodellben az `EOperation,` illetve `EParameter` lesznek. A primitív adattípusok `EDataType`-ként vannak deklarálva. Ezeknek és az `EStructuralFeature` osztályoknak a közös ősosztálya az `ETypedElement`. A modellelemnek van egy típusa, amelyet referenciaként az `EClassifier` el tud kérni az `eType()` metódus segítségével (`EAttribute` esetén ez mindig egy `EDataType`, `EReference` esetén pedig `EClass`, egyébként meg tetszés szerinti). Az eddig említett modellelemeknek van nevük, és az absztrakt `ENamedElement` leszármazottai, amelynek az ősosztálya az `EModelElement`. Az `EModelElement`-hez megjegyzés, úgynevezett

`EAnnotations` is tartozhat, amelynek ősosztálya az `EModelElement`. *Az Ecore-Model* minden objektumának ősosztálya az `EObject`.

3.4. EMF eszközkészlete

A metadmodellt megadása után le kell képeznünk a platformunkra, ez lesz a platformspecifikus modell. A modellről lehet készíteni egy fájlt (genmodel), amelyben részletes beállítási lehetőségek vannak a generálásra vonatkozóan. Ez alapján és egy sablon alapján történik a kódgenerálás. A *Generator* elkészíti a *Generator Model*-t (genmodel), a *Platform Independent Model*-t(platform független ecore modell) és a *Java Emitter Templates* (felhasználható Java sablonok) segítségével a kódot.

EMF.editor:

Automatikusan generált grafikus editor. Feladata SWT/JFace kód automatikus generálása, munkaasztali elemek beállítása, menük, varázslók létrehozása, események és akciók összekötése.

EMF.edit:

Modell manipuláció. Szerepe GUI és a modell szétválasztása. A model-elemekhez egy adapter jön létre.

EMF.model:

Modell perzisztencia megvalósítása XML fájlok támogatásával. Minden interfésze az `EObject` interfészt terjeszti ki. Az `EClass` implementációjával minden osztály tartalmaz plattformspecifikus elemeket.

3.5. EMF használatának folyamata

Először is a meta-modell definiálása szükséges *ecore*-ként, vagy egy olyan formátumban amely *ecore*-rá konvertálható. Ezután a generálás következik a *genmodel* alapján. Az eredményt ellenőrizni kell, és ha nem megfelelő akkor pedig módosítani.

3.5.1. Az EMF felüldefiniáló mechanizmusa

Strukturális változtatások adhatók meg a Visual Editor Modell-ben `EStructural-Feature` objektumok egy osztályhoz való csatolásával, vagy egy új *ecore* modell beemelésével.

A Visual Editor-ban egy EMF modell van, amely leírja, hogyan módosítható egy osztály. Ez a modell definiálja a példányokat, azok kapcsolatát, hatásköröket és tulajdonságaik beállítását. Az objektumok példányai ebben a modellben az `org.eclipse.jem.internal.instantiation.base.IJavaInstance` interfészt implementálja. Például, ha egy vizuális osztályban van egy `Shell`, amelynek mérete 200×200 pixel, ezeket értékeket két `IJavaInstance` objektum tartalmazza. Ezek közül az első a `Shell`-hez, a második pedig egy `Point`-hoz tartozik, amely a méretet reprezentálja. Ekkor egy EMF kapcsolat létesül a `Shell` és a `Point` példányai között, amely strukturális jellemzője a méret tulajdonságnak.

Az EMF egy önleíró struktúra, azaz a példány modell mögött egy meta-modell van, amely leírja, magukat az osztályokat - az ő metódusaikat, tulajdonságaikat, hozzáadott eseményeiket és hierarchiájukat. A meta-modellt a Visual Editor hozza létre, *reflection*, és *JavaBeans Introspection* kombinációjának használatával. A strukturális jellemzők az `EReference` osztály példányai lesznek, amelyek a Java osztály tulajdonságait reprezentálják.

A Visual Editor a Java osztály EMF modelljét nem csak annak tulajdonságai definiálására használja, hanem bizonyos számú segédosztály definiálására is. Ezek

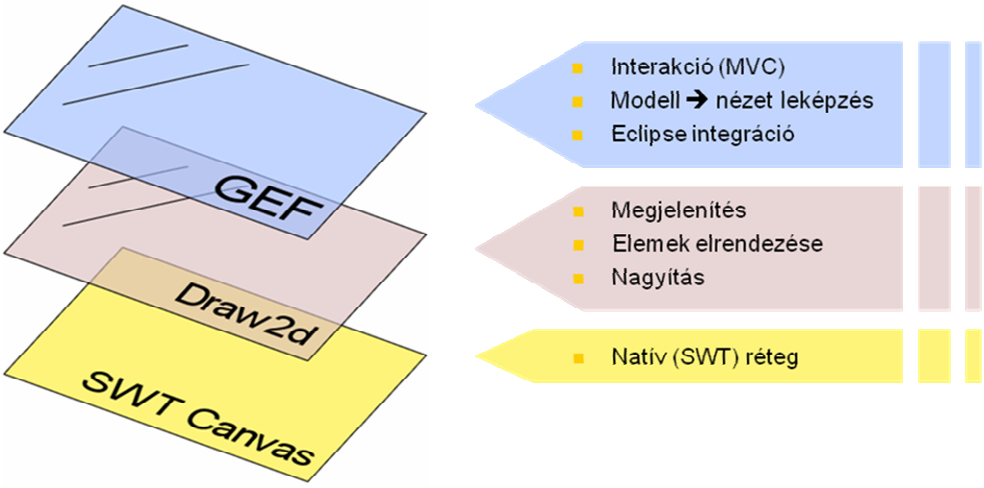
tipikusan összekötő osztályok, melyeket a Visual Editor alrendszerei használnak. Ezen osztályok nevei úgy állnak elő, hogy az alaposztály nevéhez a Helper szót fűzzük.

Összefoglalva, a felülbíráló osztályok, ahogy jelentésük is az, arra használhatók, hogy módosítsunk, hozzáadjunk, vagy töröljünk egy EMF Java osztályból.

Ahhoz, hogy a plugin érzékelje az felüldefiniáló fájlokat, a modulleíró XML-ben egy új kiterjesztési pontot kell felvenni: `org.eclipse.jem.beaninfo.registrations`. Ez specifikálja *build path*-t, amelynek eredményeként, az `ún. override` állományok elérhetővé válnak, valamint megadja ezek elérési útját a projekt könyvtárstruktúrájában.

3.6. A Graphical Editing Framework (GEF)

A Graphical Editing Framework segítségével könnyebben implementálható egy grafikus eszköz. A GEF-ben a Modell-View-Controler tervezési minta (Design Pattern) kerül megvalósításra. A Modell-View-Controler-t (röviden MVC), egy csoport, a „*négyek bandája*” (Gang of Four – GoF) írt le legelőször. A csoport tagjai: Erich Gamma, Richard Helm, Ralph Johnson és John Vlissides. 1995-ben készült el [13.] könyvük. A tervezési minta nem más, mint egy gyakori probléma és megoldásának formája leírva, a későbbi újrafelhasználás céljából. A modell például egy EMF-Modell szabályai, a kinézet pedig egy megszokott grafikus felület a Java-ban. Minden változás a modellen egy változáshoz vezet a kinézetben és fordítva. Mivel ez a kommunikáció a modell és a kinézet között történik, létezik egy úgynevezett vezérlő, ezt nevezik a GEF-ben *EditParts*-nak.



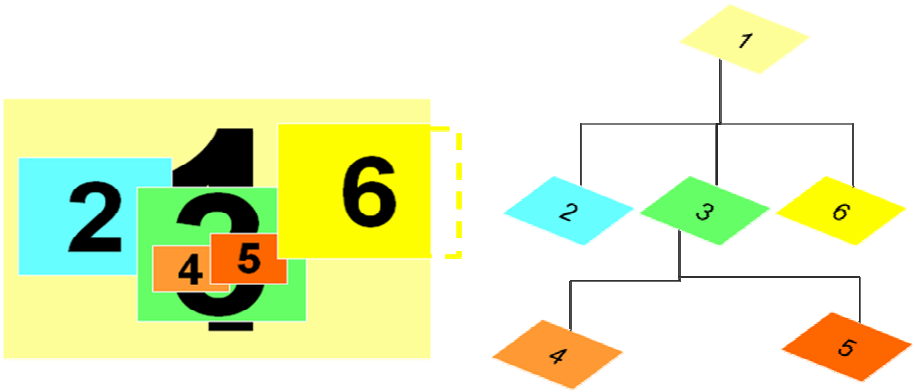
8. ábra: a GEF és az SWT kapcsolata

Modell:

EMF-ből, Java osztályokból, adatbázisból állhat. Felépítésének hierarchikusnak kell lennie, és támogatnia az értesítéseket.

View:

A Draw2D osztályai valósítják meg, amely az SWT-re épülő grafikus könyvtár. Hierarchikus megjelenés, alapeleme a Figure. Bármely SWT felülettel rendelkező alkalmazásban használható.



9. ábra: grafikus elemek tartalmazó-tartalmazott kapcsolatai

Minden gyerek csak a szülőjén belül létezik. Csak egyetlen gyökérelem van. A gyerek elemek elhelyezkedéséről a *LayoutManager* gondoskodik. Draw2D-ben geometriai alakzatok rajzolására is nyílik lehetőség a vásznon. Ilyen elemek például:

- Ellipse
- Triangle

Definiál konténer elemeket is pl.:

- Panel,

amelyre egyszerű elemeket helyezhetünk pl.:

- Label
- Button
- Image.

Vezérlő:

A modell minden eleme egy *EditPart*. Az *EditPart*-ok feladata többek között az alakzathoz tartozó elemek létrehozása, és ezen elemek frissítése a kinézetben. A modell változásai lesznek az *Action* osztályok.

3.6.1. A GEF szerkesztő lépései

- Kezdeti nézet felépítése: a modell alapján létrejönnek az *EditPart*-ok, amelyek segítségével a *Figure* példányok jönnek létre. Meghatározásra kerülnek a szabályok.
- A felhasználói akciók és az üzenetek feldolgozása a szabályok alapján, majd a modell módosítása.
- Nézetek frissítése.

3.7. Modulok

Az Eclipse IDE már az első futtatáskor rendelkezik jó pár beépített bővítménnyel, amelyeket összefoglalóan ún. *core plugin*-oknak, nevezünk. Ezek számát és összetételét a fejlesztő a céljaihoz szabhatja, olyan modulokkal, amelyek az alaprendszer funkcionalitását növelik, de alapvetően nem részei annak. Ezeket pedig *non-core plug-in*-oknak nevezzük. Ezek a modulok jellemzően tároló szervereken vannak, telepítéskor kapcsolat létesül az IDE és az adott szerver között. A modulok rendelkeznek egy leíró fájllal, ennek neve kötelezően `plugin.xml`. Ez a fájl mondja meg az Eclipse Runtime Platform-nak, hogy mire van szüksége a kiegészítés aktiválásához (ezek a modulfüggőségek – Plugin Dependencies). Az elemzett modul specifikációk ekkor *plugin registry* nevű nyilvántartóba kerülnek, és ennek alapján az Eclipse példányosíthatja őket. A specifikáció plusz információkat is tartalmazhat a modulról, úgy mint a fejlesztő neve, és a verzió.

Egy minimális plugin.xmi fájl tartalma a következő:

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin
  name="JUnit Testing Framework"
  id="org.junit"
  version="3.7"
  provider-name="Eclipse.org">
  <runtime>
    <library name="junit.jar">
      <export name="*"/>
    </library>
  </runtime>
</plugin>
```

Ez a példa XML fájl információkat szolgáltat a JUnit tesztelő infrastruktúra szolgáltatásairól a munkaasztalon. A kódban XML elemeket, és attribútumokat láthatunk. A legelső elem, minden valid XML fájlhoz hasonlóan, a XML dokumentum verziójára vonatkozó tag. Ezt követi a `plugin` elem. Ebben vannak az Eclipse számára szükséges

információk. Így, a `name` tulajdonság, amely a modul nevét adja meg. Erre a bejegyzésre nincs semmilyen megszorítás. Nem úgy az `id` attribútumra, amelynek egyedinek kell lennie, hiszen a modulra való hivatkozásra szolgál. Ez különbözteti meg a modult a többitől, valamint arra is használható, hogy segítségével a forráskódból a következő utasítással elérjük a kiegészítés futó példányát:

```
Plugin p = Platform.getPlugin(pluginID);
```

A példányokat azonban sohasem explicit módon a kódból érjük el, ezt mindenkor az Eclipse Runtime Platform menedzseli.

4. Eszközök grafikus interfész építéséhez

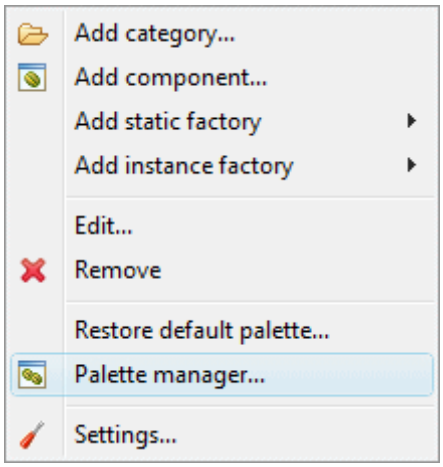
Szakdolgozatunkhoz való információgyűjtés során meglehetősen sokféle GUI Builder eszközzel megismerkedtünk. Néhány ilyen szoftvert áttanulmányozva, előnyeiket és hátrányaikat mérlegelve esett választásunk az Eclipse Visual Editor beépülő moduljára, amely nem a legmodernebb ezek közül, és dokumentációja sem a legrészletesebb, viszont vitathatatlan előnye, hogy nyílt forrású szoftver, nagyon rugalmas, és az SWT alapú grafikus interfészépítést is támogatja, és ez a három szempont bírt a legnagyobb súllyal választásunkkor.

A téma átfogó vizsgálata azonban csorbul, ha nem teszünk említést a további lehetőségekről. Tekintettel arra, hogy ezek bizonyos tulajdonságaiban messze felülmúlják a Visual Editort. Az alábbiakban rövid kitekintést teszünk, és nagyvonalakban bemutatjuk ezeket az eszközöket.

4.1. A Window Builder

A Window Builder egy kiforrott, könnyen használható Java GUI tervező eszköz. Alkalmas SWT, Swing és GWT alapú grafikus felhasználói felületek gyors létrehozására. A szoftver, hasonlóan a Visual Editorhoz, egy bővítmény, amely az Eclipse-hez, és más Eclipse alapú fejlesztői környezethez (RAD, RSA, MyEclipse, JBuilder) csatolható. Magas minőségű támogatást nyújt a kétirányú kódgeneráláshoz, eseménykezeléshez, a nemzetközisítéshez stb.

A szerkesztő a későbbiekben tárgyalt VisualEditor-hoz hasonló felhasználói interfész komponensekből áll: tervező nézet, forráskód nézet, komponensek fa szerkezetének megjelenítése, tulajdonságlap és a paletta.



A paletta kiterjeszthetősége a számunkra legfontosabb tulajdonság. A Window Builderben a paletta pár kattintással megváltoztatható. Ehhez a palettán, jobb gombbal való kattintáskor a képen látható pop-up menü áll segítségünkre. A menüben az *Add category…* és *Add component…* bejegyzésekkel villámgyorsan új paletta-kategóriát, illetve bejegyzést lehet a palettához fűzni. Ezzel ellentétben a Visual Editorban a palettakibővítés meglehetősen nehézkes, hiszen kényelmesebb módszer híján, kézzel kell az ezért felelős XML alapú állományokat manipulálni. Ezen túlmenően a *Palette manager…* menüpontban lehetőségünk van a paletta eszköz teljes konfigurációjára. A megjelenő dialógus ablakban átrendezhetjük és törölhetjük a palettabejegyzéseket. Az egyes kategóriáknál pedig megadható, hogy alapértelmezett módon melyik legyen nyitott, vagy zárt állapotban a paletta betöltődésekor.

Hátránya viszont, hogy kereskedelmi forgalomba szánt szoftver. Így nem felel meg alapelvárásainknak.

4.2. A Jigloo

A Jigloo egy szintén kereskedelmi forgalomba szánt, GUI építésre alkalmas Eclipse bővítmény. Elsősorban Swing és SWT fölött használható. Támogatja a kódgenerálást a vizuális megjelenés alapján, valamint a kód módosításával is manipulálhatjuk a kinézetet, (azaz a kétirányú kódgenerálást), és tartalmaz sok, a Visual Editor-nál megismert lehetőséget.

Mivel gyakorlatilag funkcionálisan egyáltalán nem nyújt többet a Visual Editortól, és még kereskedelmi szoftver is, ezért hamar elvetettük, mint lehetséges munkaeszközt.

4.3. Matisse

A NetBeans GUI építő modulja, az alapkonfigurációnak a része. A szoftver ingyenes, használata egyszerű, az alaprendszerbe való integrációja jó minőségű. Az ok, amely miatt mégsem ezt az eszközt választottunk, az volt, hogy nem támogatja az SWT alapú grafikus interfészfejlesztést, valamint nem alkalmas kétirányú kódgenerálásra.

5. Visual Editor

Fontosnak tartjuk a Visual Editor részletes, elemző leírását, hiszen a „Hibernate Plugin bemutatása” című témakörben, folyamatosan hivatkozni fogunk az itt ismertetett eszközökre, funkciókra, kifejezésekre.

5.1. Funkcionális áttekintés

A Visual Editor arra használható, hogy megnyissunk és manipuláljunk vele bármilyen `java` kiterjesztésű fájlt. Elemzi az megnyitott fájl forráskódját, megkeresi a vizuális osztályokat a kódban. Amennyiben talál ilyet, Visual Editor eszközöket indít el, amelyekkel több módon megtekinthetjük és/vagy manipulálhatjuk a forráskódot. Továbbá varázslók állnak rendelkezésünkre új vizuális osztály létrehozásához és szabad alakításához.

A Visual Editor által biztosított eszközöket az alábbi táblázat foglalja össze:

Funkcionális komponens	**Szolgáltatások**	**Kinézet**
Grafikus megjelenítő (Graphical Viewer)	A grafikus felhasználói interfész előnézeti képének megjelenítése. Beanek kijelölése, és a vászonra, vagy másik beanre dobása, vagy ezek méretének, helyének manipulálása. Felugró ablakban lévő szolgáltatások kezelése (pl.: másolás, törlés, események hozzáadása, stb.). Egy bean kijelölésekor, annak tulajdonságainak	A beanteket egy különálló virtuális gép példányosítja (ez a Target/Proxy VM). Ez után a forrásban megjelenik a bean, és a bean képe is a vásznon.

	megjelenítése a tulajdonságok ablakban (Properties), és a forráskód releváns részéhez való automatikus görgetés.	
JavaBeanek fa szerkezete (JavaBeans Tree)	Beanek kijelölése, a palettáról és a fa szerkezetre, vagy másik beanre való ejtése. Előugró menü kezelése. A tulajdonságok megjelenítése, és a forráskód szerkesztőben (Source Editor) való automatikus görgetés az előbb leírtak szerint.	A beanek fa szerkezetben jelennek meg, így mutatva a tartalmazó-tartalmazott kapcsolatokat. Ezen túl megjeleníti még az egyes beanekhez rendelt eseményeket is.
Tulajdonság ablak (Properties View)	A kiválasztott bean tulajdonságainak módosítása, a megfelelő CellEditor használatával.	Megjeleníti a kiválasztott bean tulajdonságait.
Forráskód szerkesztő (Source Editor)	Egy teljes értékű Java Editor.	Ha kijelölünk egy beant, akkor automatikusan a releváns kódrészletre görget.
Paletta (Palette)	Beanlétrehozó eszköz, ennek részletes bemutatását később részletezzük.	Az elérhető beanek listája.
Varázslók (Wizards)	A „New Visual Class” varázslóval lehetőségünk van előre definiálni a GUI sablonokat	Egy form, amellyel könnyebben írhatunk GUI osztályt.

5.2. A Visual Editor komponensei

5.2.1. A forráskód szerkesztő és kódgeneráló komponens

A Visual Editor forráskód szerkesztőjének és kódgeneráló komponensének kulcsfontosságú elemei a következők:

JavaVisualEditorPart

Az `org.eclipse.jdt.internal.ui.javaeditor.CompilationUnitEditor` osztály a *JavaVisualEditorPart* osztálynak egy leszármazottja, amely az alap-értelmezett *Java Editor* az Eclipse-ben. Ez az osztály azonban nem csak egy beágyazott forráskód-szerkesztőként funkcionál, hanem a Visual Editor belépési pontját jeleni. Ezért ez sokkal több, mint egy szerkesztési lehetőség.

Azon túl, hogy vizuális komponenseket hoz létre, amikor a felhasználó a grafikus szerkesztőt használja, vagy közvetlenül a forráskódot manipulálja, a rendszer létrehoz egy *IModelBuilder* objektumot, amely elemzi a Java kódot és ebből felépíti a *VE Model*-t és ez pedig, azokat az osztályokat, amelyek a grafikus megjelenés, és a forráskód közötti konverziót végzik.

Visual Editor-ban több forrásból frissülhet a forráskód, ilyen a grafikus megjelenítő, a forráskód szerkesztő, stb. Néhány frissítés a kijelző-vezérlő szálon helyezkedik el, mások lassabb, háttérben futó szálakon. Ezért a forrás fájl frissítéseit a JDT *CompilationUnit*-ja munkamásolatok használatával koordinálja.

JavaSourceTranslator

Feladata felépíteni egy `IBeanDeclModel`(BDM) példányt. *JavaSourceTranslator* az `IDiagramModelBuilder` és az `IDiagramSourceDecoder` interfészeket imlpementálja és egy `JavaBeanModelBuilder` példányt és egy `CodeSnippetModelBuilder` példányt használ, attól függően, hogy először elemzi a

kódot, vagy csak reagál a szerkesztőtől származó bevitelre. A BDM a forráskód egy közbenső állapota, mely lehetővé teszi a forráskód frissítését, így felhasználói beavatkozások a grafikus megjelenítőben, a JavaBean-ek fa szerkezetében és a tulajdonságlapon, a *VE Model* frissítéséhez vezetnek. Ezek a változtatások aztán eljutnak a *BDM Model*-hez és innen a forráskódhoz. A felhasználók közvetlenül is szerkeszthetik a forráskódot a forráskód szerkesztőben. Ezeket a változásokat a *CodeSnippetModelBuilder* dolgozza fel, amely frissíti a *BDM Model*-t.

Amint a BDM felépült, a *JavaSourceTranslator* egy `IVEModelInstance` példány létrehozására használja a *VE Model*-t.

5.2.2. A grafikus megjelenítő és a JavaBeanek fa szerkezete

Az Eclipse Visual Editor moduljában nem minden elem jelenik meg a grafikus megjelenítőben és a JavaBean-ek fa szerkezetében. A grafikus megjelenítőn csak az *IJavaObjectInstance* példányokat reprezentáló látható komponensek jelennek meg, mint például a *JFrame, Jpanel, JButton*. A *JavaBeansTree*-ben pedig ezeken, a komponenseken kívül minden hozzájuk tartozó eseménykezelő is megjelenik. A felhasználói tevékenységeket a GEF eszköz kezeli. Ez az eszköz egy parancsot kér egy megfelelő *EditPart*-tól. Az *EditPart* delegál egy a kapott parancsból épített új parancsot az *EditPolicy* objektumoknak. Ezek képesek eldönteni, hogy a kapott parancs valid, vagy sem, és abban az esetben, ha helyes, végrehajtják a megfelelő parancsot. Helytelen parancs esetén pedig a felhasználót értesítik az anomáliáról. Ha a parancs értelmezhető, akkor a GEF *CommandStack* használatával futtatja.

5.2.3. Visual Editor kiterjesztési pontjai

Mint a legtöbb Eclipse modul, a VE is definiálja saját kiterjesztési pontjait, ezek publikus interfészek, amelyeken keresztül a modulfejlesztők kiterjeszthetik, az adott modul alapértelmezett viselkedését.

A VE az alábbi kiterjesztési lehetőségeket biztosítja:

org.eclipse.java.core.style

Lehetővé teszi a kódgenerálási stílus felüldefiniálását. Ezek a stílusok szabályokat foglalnak magukban például: *IChildRule, IPropertyRule, IInstanceVariableRule* stb. Ezeket a szabályokat Eclipse arra használja, hogy meghatározzon olyan elemeket, mint a nevek, a stílus, az elhelyezkedés, a hatáskör stb., így a részletes kódolási stílust a felhasználó választhatja meg.

org.eclipse.ve.internal.java.core.vce.lookandfeel

Használatával új, a kinézetre vonatkozó beállítások adhatók meg. Egy Eclipse modulhoz konfigurációs lehetőségként teszi lehetővé színek kiválasztását. Egy saját *Property page* létrehozása szükséges, amelyen be lehet állítani a megfelelő színeket, fontokat, elrendezést. Ehhez az `org.eclipse.ui.PreferencePages` kiterjesztési ponthoz kell kapcsolódni. A kiterjesztési pont igényli, hogy megadjuk, milyen névvel, illetve milyen másik oldal alá szeretnénk betenni az új lapunkat, amely lehetővé teszi a kényelmes hierarchikus beállítás-szervezés összeállítását.

org.eclipse.ve.java.core.newStyleComponent

Arra használható, hogy új vizuális osztály stílusokat, és komponens-kiterjesztéseket építsünk a Visual Editor-ba. Ezek az új elemek meg fognak jelenni a *New Visual Class* menüpont alatti varázslóban.

org.eclipse.ve.java.core.choosebean

A paletta *Choose Bean* dialógusablakának testre szabását teszi lehetővé.

org.eclipse.ve.java.core.contributors

A Visual Editor funkcionalitásához hozzájáruló állományok csatlakozási pontja. A paletta kiterjesztéséért felelős XMI fájl is ezen a ponton csatolható.

org.eclipse.ve.java.core.registrations

A felhasználó projektjéhez adható, könyvtárak listájának bővíthetőségét szolgálja.

org.eclipse.ve.cde.adapter

Egy „belső” kiterjesztési pont, amelyhez csak Eclipse Visual Editor példányok csatlakozhatnak.

org.eclipse.ve.cde.editPartContributor

TreeEditPart és *GraphicalEditPart* kiterjesztésére, felüldefiniálására vonatkozó információk megadhatóságát teszi lehetővé.

org.eclipse.ui.popupMenus

Új bejegyzés adható vele a helyi, felugró menühöz.

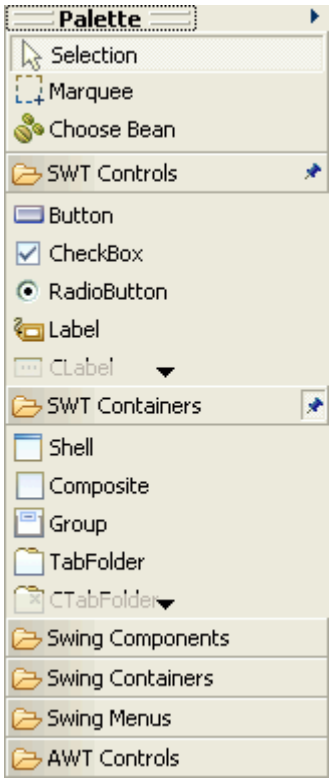
org.eclipse.jem.beaninfo.registrations

A felülbíráló (*.override*), és *BeanInfo* fáljok csatolását kezeli.

5.3. A paletta eszköz

A paletta nem más, mint a *Graphical Editing Framework (GEF) Viewer* egy speciális formja. Segítségével kényelmesen tudunk programunkra grafikus felhasználói interfész réteget építeni. Palettabejegyzésekből épül fel, amelyek egy ikonból, és egy alkalmas névből állnak. Minden egyes bejegyzés egy *JavaBean*-t reprezentál. Ezek példányosítása a vizuális osztályra történő ejtéssel valósul meg. Ekkor a forráskódban

megjelenik az adott osztálynak egy példányosítása, valamint a beállított `Layout`-tól függő további információk. A palettabejegyzések ún. palettakategóriákban helyezkednek el, amelyek szintén rendelkeznek valamilyen, a bejegyzések közös tulajdonságára utaló, névvel. A kategóriák is tovább bonthatók bejegyzéscsoportokra, ezen csoportok határát vízszintes vonal jelöli. A kategóriákon túl néhány további eszköz is rendelkezésünkre áll, ilyen a *Selection* eszköz, amely egyfajta kurzorként funkcionál a vásznon. Segítségével létező komponenst tudunk kijelölni. Ehhez hasonló a *Marquee* eszköz, amely téglalapos kijelölést valósít meg. Minden olyan komponens, amely teljes egészében benne van a kijelölő téglalapban, jelölést kap. A *Choose Bean* sor pedig egy szelekciós párbeszédablakot nyit, amelyen az elérhető összes bean közül kiválaszthatjuk a palettán kívülieket.



A paletta eszköz működése gyakorlatilag minden mai programozó számára ismert. Számunkra azonban a háttérben lévő mechanizmusok és kódok lesznek lényegesek. A paletta lelke nem egyéb, mint XMI fájlok összessége. Minden olyan Eclipse plugin, amelynek funkcióit az általa manipulált palettán keresztül érhetjük el, rendelkezik ilyen XMI fájllal. Amikor az Eclipse Core példányosítódik, minden csatolt modult elemez. A *plugin.xml* alapján megvizsgálja, hogy az adott kiegészítő rendelkezik-e saját palettabejegyzésekkel, ha igen, akkor ezeket hozzáfűzi a kezdetben még teljesen üres palettához.

.

6. A HiberGUI plugin bemutatása

A modul készítés első lépéseként be kell szereznünk az Eclipse-ünk mellé a szükséges kiegészítéseket. A moduljainkat a VisualEditor plugin kiterjesztéseként szeretnénk létrehozni, emiatt a legfontosabb függőség a Visual Editorral való kapcsolat. Mivel az Eclipse alapkonfigurációja nem rendelkezik, a Visual Editor kiegészítővel le kell töltenünk a megadott tárolóról: (http://update.soyatec.org/Ganymede/ve/1.4), és telepítenünk az alábbiakban megadott módon: http://wiki.eclipse.org/VE/Update.

Ezt követően a HiberGUI projektet SVN technológiával letöltöttük. Miután mindkettőnk rendelkezésére állt a projekt teljes kódja nekiláttunk a projekt plugin-projektté való alakításához. Ez abból áll, hogy létrehozunk egy `plugin.xml` fájt a projekt gyökerében, és a plugin-projekt működéséhez szükséges kódokkal feltöltjük.

Ezt az állományt Eclipse IDE-ben kétféle módon is manipulálhatjuk. Egyik lehetőség a beépített *Source Editorral* történő szerkesztés, a másik, kényelmesebb, modernebb mód, egy *Eclipse Forms* alapú összetett editor eszköz használata, az egyes mezők kitöltése után automatikusan generált `plugin.xml`, és `Manifest.mf` fájlt kapunk. Az Eclipse modul-fejlesztő eszközrendszerében, erre a grafikus eszközre való fokozatos átállás figyelhető meg. Az átállást azért mondhatjuk fokozatosnak, mert szakdolgozatunk megírásának idejében, minden beállítást meg lehet adni mindkét eszköz segítségével, viszont a korábbi leírásoktól eltérően nem, minden információ a `plugin.xml`-ben foglal helyet, ugyanis az *Eclipse Form*s alapú technológia, a kiterjesztési pontokon kívül minden információt a `Manifest.mf` állományba generál. Ez, a fejlesztés korai szakaszában, sok problémát gördített elénk. Az Eclipse Forms alapú megoldásnál az első fülön (*Overview*) tudjuk megadni a modulunk adatait: azonosító, verziószám, plugin név, szerzők, platformszűrő, aktivátor.

6.1. Aktiváció

Az aktivizáció nem egyéb, mint processzkezelés. Az egyszerűbb kiegészítéseknél nem szükséges ennek specifikus megvalósítása, ebben az esetben, kihasználjuk az Eclipse pluginkezelő általános viselkedésmódját. Ha azonban arra van szükségünk, hogy a modulunk valamilyen speciálisat tegyen aktivációjakor, vagy deaktivációjakor, akkor a saját pluginkezelőnkben, amely `org.eclipse.core.runtime.Plugin` leszármazottja, felül kell definiálnunk a `start`, és a `stop` metódusokat. Majd ennek a saját osztálynak a nevét, `class` attribútum értékeként fel kell tüntetni a `plugin.xml` állományban.

Mivel szakdolgozatunk alapprojektjében nem volt szükség, speciális aktivációra, ezért megfelelőek, a következő generált metódusok.

```
@Override
public void start(BundleContext context) throws Exception        {
        super.start(context);
        plugin = this;
}

@Override
public void stop(BundleContext context) throws Exception {
        plugin = null;
        super.stop(context);
}
```

Az aktivátor osztálynak természetesen tartalmaznia kell annak a modulnak az azonosítóját, amelyet a `plugin.xml`-ben megadtunk.

6.2. Modulok közötti kapcsolatok

Az Eclipse modellben, a modul kapcsolatban állhat más modulokkal, a kapcsolat módja kétféle is lehet:

- Függőség (Dependency): Függőségi szempontból egy plugin lehet függő, vagy előfeltétel modul. Az előfeltétel modul hozzájárul a függő funkcionalitásához.
- Kiterjesztés (Extension): Kiterjesztés szerint pedig lehet gazda és kiterjesztő modul. A kiterjesztő modul kiterjeszti a gazda funkcionalitását.

6.3. Függőségek

Amikor a kiegszítés függ más moduloktól, akkor a függőséget a következő módon, egy `requires` tag-gel jelölnünk kell a `plugin.xml`-ben.

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin
    id="com.bolour.sample.eclipse.demo"
    name="Extension Processing Demo"
    version="1.0.0">
    <runtime>
        <library name="demo.jar"/>
    </runtime>
    <requires>
        <import plugin="org.eclipse.ui"/>
    </requires>
</plugin>
```

Ha azonban az `Eclipse Forms` alapú editort használjuk, akkor az előfeltétel modulokra való hivatkozások a Manifest.mf állományba generálódnak. A két példát összehasonlítva látható, hogy ugyanarra szolgálnak, a különbség az XML struktúra hiánya.

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: HiberGUI Plugin
Bundle-SymbolicName: HiberGUI;singleton:=true
Bundle-Version: 1.0.0
Bundle-Activator: hibergui.Activator
Require-Bundle: org.eclipse.ui,
 org.eclipse.core.runtime,
 org.eclipse.ve.Java.core;bundle-version="1.4.0",
 org.eclipse.jdt.core;bundle-version="3.4.2",
 org.eclipse.jem;bundle-version="2.0.201",
 org.eclipse.jem.proxy;bundle-version="2.0.100",
 org.eclipse.ve.cde;bundle-version="1.4.0",
```

```
 org.eclipse.ve.swt;bundle-version="1.4.0",
 org.eclipse.ve.propertysheet;bundle-version="1.4.0",
 org.eclipse.gef;bundle-version="3.4.1",
 org.eclipse.emf.ecore;bundle-version="2.4.1"
Eclipse-AutoStart: true
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
Bundle-ClassPath: hiberguiplugin.jar
Bundle-Vendor: Zsenyuk-Tóth
```

Bármelyik módon is adjuk meg, a definíció mindig, fordítási és futásidejű direktíva is, hiszen az *Eclipse Runtime Platform*-nak biztosan tudnia kell, hogy az előfeltétel osztály elérhetővé tehető-e a függő osztály számára, amikor a függő plugin aktiválódik. És fordítási időben, az Eclipse utasítást kaphat ezen bejegyzés alapján arra, hogy bővítse a C*lasspath*-t minden, előfeltétel modul `jar` kiterjesztésű fájljával.

A `Manifest.mf` fájlban látható függőségek áttekintése:

org.eclipse.ui

Ez a csomag felelős a grafikus interfész betöltéséért az Eclipse indulásakor.

org.eclipse.core.runtime

Feladata a rendszer indítása, és futási időben annak menedzselése. Ez az eszköz felel a csatolt modulok nyilvántartásáért, és azok betöltéséért

org.eclipse.ve.Java.core

Ez a felelős a Visual Editor magjának betöltéséért.

org.eclipse.jdt.core

Tartalmazza a Java objektummodellt és a hozzá szorosan kapcsolódó eszközöket.

org.eclipse.jem, org.eclipse.jem.proxy

Az EMF felüldefiniáló mechanizmusához hozzájáruló csomagrendszer.

org.eclipse.ve.cde

Ennek példányai a tulajdonságlap számára szolgáltatnak információkat a hozzátartozó beanről.

org.eclipse.ve.swt

A grafikus szerkesztő(Visual Editor) megjelentésére szolgál.

org.eclipse.ve.propertysheet

A beanek tulajdonságlapjának betöltéséért felelős.

org.eclipse.gef

A grafikus eszközök megjelenéséért és vezérléséért felelős.

org.eclipse.emf.ecore

A két irányú kódgenerálás menedzselését végzi.

6.4. Kiterjesztés

Ha a modulunk egy funkcióját direkt módon elérhetővé kell tennünk a felhasználó számára, akkor egy, vagy több felhasználói interfész elemet is hozzá kell adnunk, vagy a meglévőket megváltoztatnunk az Eclipse munkaasztalban. Például, ha a plugin-unkhoz megváltoztatott palettát akarunk társítani, akkor, ki kell bővítenünk a Visual Editor palettabetöltéséért felelős osztályait, modulunkra vonatkozó bejegyzéssel. Ezt a problémát kiterjesztéssel oldhatjuk meg. A kiterjesztés, célja, hogy a gazda modul viselkedését módosítsuk a sajátunkkal. Tipikusan arra használhatjuk, hogy az Eclipse munkaasztalát megváltoztassuk.

Egyszerű esetben a kiterjesztés létrehoz egy `Callback` (,a továbbiakban kommunikátor) objektumot, ezen keresztül kommunikál a gazda, és a kiterjesztő plugin. A kommunikátor objektumot, eltérően a *non-core plugin*-októl, az Eclipse automatikusan példányosítja, és menedzseli életciklusa alatt. Egy kiterjesztés több ilyen objektumot is

létrehozhat, a kiterjesztési kapcsolatban álló modulok tulajdonságaitól függően. A kiterjesztési modellben lehetőségünk van egyedi kommunikátor objektum létrehozására is, erre azonban szakdolgozati projektünkben nem volt szükség.

Egy rugalmas modul több kapcsolódási pontot is felajánlhat, amelyek mentén a kiterjesztő modulok kapcsolódhatnak hozzá. Ezeket nevezzük kiterjesztési pontoknak (*Extension Point*). Egy kiterjesztési ponton több modul is kapcsolódhat a gazda modulhoz.

Az eddigiekhez hasonlóan a kiterjesztési pontokat is fel kell jegyeznünk mind a gazda, mind pedig a kiterjesztő modul `plugin.xml` állományában. A kiterjesztési pont deklarációja `extension-point` XML tag-gel adható meg, mindkét esetben. A különbség csak az, hogy amíg, gazda az elérhető kiterjesztési lehetőségeit, addig a kiterjesztő a kiterjeszteni kívánt jelzi ezekben a deklarációkban.

Mivel saját modulunk nem biztosít kapcsolódási lehetőséget más bővítmények számára, ezért az alábbi példán az Eclipse Help menüjéért felelős moduljából származó kiterjesztési pont deklarációja látható:

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin
    id="org.eclipse.ui"
    name="Eclipse UI"
    version="2.1.0"
    provider-name="Eclipse.org"
    class="org.eclipse.ui.internal.UIPlugin">
    <extension-point
           id="org.eclipse.ve.Java.core.contributors"
     <!-- Other specifications omitted. -->
</plugin>
```

A specifikáció definiál egy egyedi azonosítót. Ez az azonosító nem feltétlenül egyedi globálisan, elegendő, ha csak a tartalmazó modulban, lokálisan egyedi, hiszen a korábban tárgyalt modul azonosító ismeretében, globálisan is azonosíthatóvá válik a kiterjesztési pont.

A kiterjesztő oldalon lévő `plugin.xml` egy része pedig a következő:

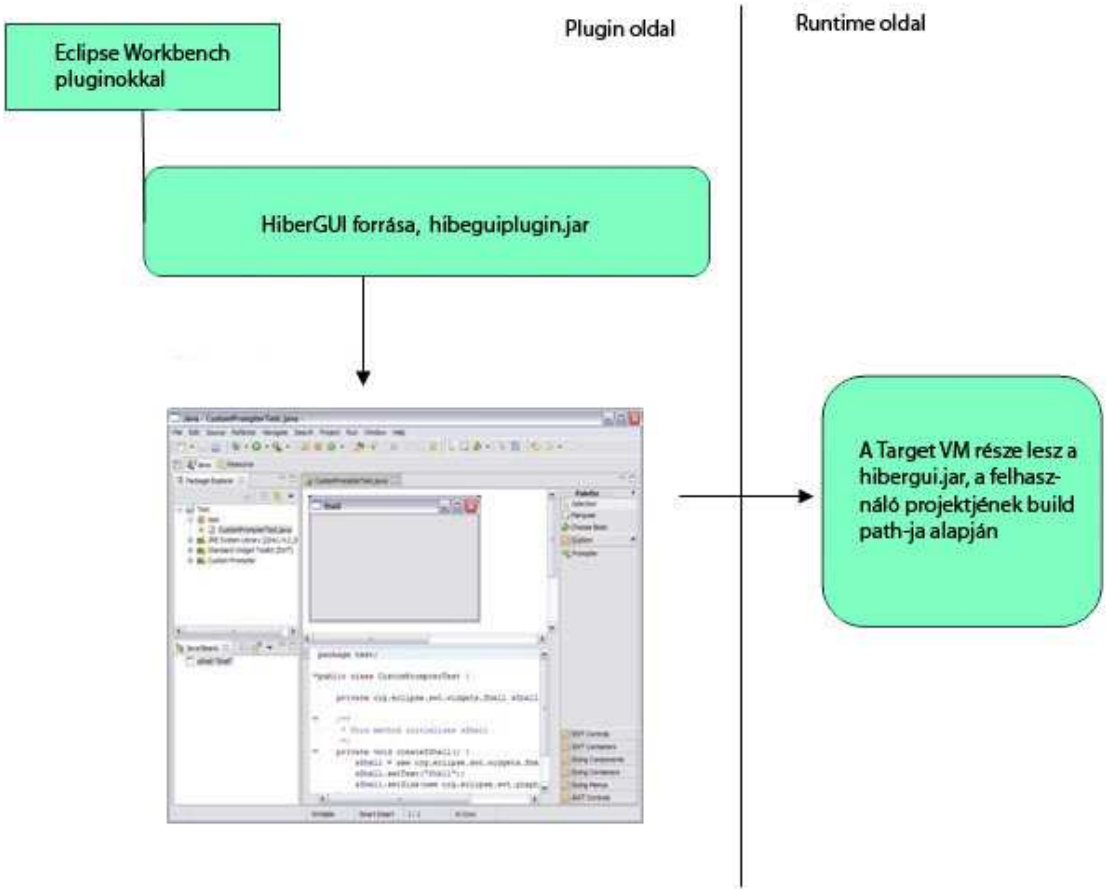
```
<plugin>
<!-- ... -->
<extension
         point="org.eclipse.ve.Java.core.contributors">
      <palette
            container="HiberGUIContainerID"
            categories="paletta.xmi"/>
   </extension>
<!-- ... -->
</plugin>
```

A kiterjesztő `plugin.xml` fájlában is hasonló `extension-point` bejegyzés szükséges a korrekt kiterjeszthetőséghez. Az `id` annak a gazdamodulbeli kiterjesztési pontnak a globális azonosítója, amelyhez kapcsolódik. Ennek mindig létező azonosítóra kell mutatnia.

A kiragadott példa azt a kiterjesztési pontot mutatja, amely mentén a paletta kiterjesztődik, a `hibergui.xmi` palettabejegyzései alapján.

6.5. Különböző `.jar` kiterjesztésű állományok szerepe a modulban

Amikor bővítményt fejlesztünk végül két `jar` kiterjesztésű állományt kell létrehoznunk, ezek közül az első, ún. *plugin side jar*. Ez a HiberGUI projekt teljes forrását tartalmazza. A másik a *runtime side jar*. Ez a *plugin side jar* állományt és a modulleíró `plugin.xml`, és `Manifest.mf` állományokat csomagolja, ez sohasem fut IDE-n belül, ehelyett a felhasználó projektjének *build path*-jába kerül és minden esetben *Target VM* futtatja.



10. ábra: a jar állományok

6.6. A Classpath konténer

A könnyű kezelhetőség érdekében a `plugin.xml` fájlban, kiterjesztési pontként megadható egy *Classpath Container* amely tartalmazza a HiberGUI kódját és kényelmes importálási lehetőséget nyújt. A kiterjesztési pontban kötelezően meg kell adnunk a név tulajdonságot, a konténer egyedi azonosítóját illetve a felhasználandó varázsló hivatkozását. Ezt követően inicializáló bejegyzések szükségesek további kiterjesztési pontokban. Ennek eredményeként, amikor a felhasználó csatolja Eclpise példányához modulunkat, akkor rendelkezésére áll az *Add Lbrary* menüpont alatt megjelenő varázsló oldal. Amelynek sorai a HiberGUI bejegyzéssel bővülnek ki. Ennek kiválasztásával,

hozzáadódik a HiberGUI projekt teljes kódja, a felhasználó projektjének *build path*-jához, és a `plugin.xml`-ben megadott `paletta.xmi` által definiált palettabejegyzések ekkor terjesztik ki a palettát.

Modulunk leírófájljának erre vonatkozó részei az alábbiak:

```
    <extension
             point="org.eclipse.jdt.ui.classpathContainerPage">
          <classpathContainerPage
                name="HiberGUI"

class="org.eclipse.ve.internal.Java.wizard.RegisteredClasspathContainerW
izardPage"
                id="HiberGUIContainerID">
          </classpathContainerPage>
       </extension>

    <extension
             point="org.eclipse.jdt.core.classpathContainerInitializer">
          <classpathContainerInitializer

class="org.eclipse.ve.internal.Java.core.RegisteredClasspathContainerIni
tializer"
                id="HiberGUIContainerID">
          </classpathContainerInitializer>
    </extension>

    <extension
         point="org.eclipse.ve.Java.core.registrations">
         <registration
             container="HiberGUIContainerID"
             description="HiberGUI">
             <library runtime="hibergui.jar"/>
         </registration>
    </extension>
```

6.7. A paletta.xmi

A projektünk következő lépése a korábban többször is hivatkozott, `paletta.xmi` létrehozása, és a megfelelő XML struktúrával való feltöltése.

Az XML verzió meghatározás, és az `xmlns` kezdetű sorokban lévő szabvány szerinti importálások után, a `palette:CategoryCmp` tag-ben kerül definiálásra az a kategória, amellyel a plugin kiterjeszti a palettát. A `categoryLabel` szolgál a kategória nevének beállítására, ez esetünkben HiberGUI lesz. Ezt követi a csoport deklarációja, amelybe az egyes bejegyzéseket reprezentáló tag kerül. Ezek a `cmpEntries` tag-ek típussal, egyedi azonosítóval, és egy, a bejegyzés ikonjára, és a példányosítandó osztályra történő hivatkozással, valamint a palettán megjelenő címkével rendelkeznek.

```
    <?xml version="1.0" encoding="UTF-8"?>
    <xmi:XMI xmi:version="2.0"
      xmlns:xmi=
       "http://www.omg.org/XMI"
      xmlns:xsi=
       "http://www.w3.org/2001/XMLSchema-instance"
      xmlns:ecore=
       "http://www.eclipse.org/emf/2002/Ecore"
      xmlns:palette=
       "http:///org/eclipse/ve/internal/cde/palette.ecore"
      xmlns:utility=
       "http:///org/eclipse/ve/internal/cde/utility.ecore">
      <palette:CategoryCmp xmi:id="swtCat0">
        <categoryLabel xsi:type=
               "utility:ConstantString" string="HiberGUI"/>
        <cmpGroups xsi:type="palette:GroupCmp">

          <cmpEntries xsi:type=
               "palette:AnnotatedCreationEntry"
               xmi:id="entry2"
               icon16Name=
               "platform:/plugin/HiberGUI/icons/picture.gif">
            <objectCreationEntry
               xsi:type="palette:EMFCreationToolEntry"
               creationClassURI=
               "Java:/hibergui.hibernate#HibernateForm"/>
            <values xmi:type=
               "ecore:EStringToStringMapEntry"
               key=
               "org.eclipse.ve.internal.cde.core.nameincomposition"
value="hibernateForm"/>
             <entryLabel xsi:type=
               "utility:ConstantString" string="HibernateForm"/>
          </cmpEntries>
          <!—további bejegyzések…-->
        </cmpGroups>
      </palette:CategoryCmp>
    </xmi:XMI>
```

A fent látható kódban lévő csoportbejegyzést az alábbi osztályok mindegyikére végrehajtottuk

- hibergui.property.CheckBoxEdit.Java
- hibergui.property.DateCombo.Java
- hibergui.property.DateCombo2.Java
- hibergui.property.DateEdit.Java
- hibergui.property.IntegerEdit.Java
- hibergui.property.RadioEdit.Java
- hibergui.property.StringEdit.Java
- hibergui.property.TextEdit.Java
- hibergui.property.TimeEdit.Java
- hibergui.property.TimeEdit1.Java

Ennek eredményeként a HiberGUI paletta kategóriájában megjelennek ezen osztályoknak megfelelő bejegyzések, amelyek a paletta tulajdonságainak megfelelően ráhúzhatóak a vizuális osztályunk előnézeti képére.

6.8. A megvalósítás akadályai

A palettát sikeresen kiterjesztettük, a *classpath* konténert, és a `HibernateForm` osztályunkat megfelelően beállítottuk ahhoz, hogy a palettán megjelenjen és innen a vizuális osztályunkra tudjuk húzni. Amikor azonban megpróbáltuk ráhúzni akkor a célosztály forráskódjában helyesen generálódott le a példányosítás, de nem jelent meg grafikus nézetben. Hosszas kutatómunka és témavezetőnkkel való többszöri konzultáció után sikerült elérni, hogy az előbb említett osztály grafikusan is megjelenjen.

A szakdolgozatunk eredeti célkitűzésének megfelelően a következő lépés, hogy a `Composite`-ot származtató osztályban lévő `Composite`-ra kell ráhúzhatóvá tennünk az

adott `Control-`t, és nem pedig magára az osztályra, amely a Visual Editorral, a határidő közelsége, és a projekt szegényes dokumentációja miatt, eddigi ismereteink alapján nem megoldható.

7. Eclipse update site

Az Eclipse hivatalos oldaláról letöltött kiadását további elemekkel lehet bővíteni. Ilyen volt a szakdolgozati munkánkhoz szükséges *VisualEditor* is. Ezeket, az elemeket előre elkészített *Update Site*-okról lehet elérni. Ezeknek, az oldalaknak a segítségével a már letöltött és használt elemeket lehet frissíteni. A szakdolgozatunk egyik célja volt, hogy az elkészített modult egy ilyen oldalra felrakjuk, ahonnan elérhetővé válik bárki számára. Néhány szó az oldalak használatáról Eclipse-ben.

Eclipse-ben a *Help* menüpont alatt ki kell választanunk a *Software Updates* menüpontot. Az első fülön *Installed Software* tekintők meg a már feltelepített elemek, amelyek innen egyszerűen frissíthetők az *Update* gombbal.

A másik fül *Available Software*. Itt találhatók az Eclipse-be alap állapotban beállított oldalak. A felső mezőt használhatjuk konkrét elem keresésére. Lehetőség van új oldal felvételére: *Add Site*, amelyre szükségünk lesz, hogyha a saját kész oldalunkat is használni szeretnénk az Eclipse-ben.

Egy *update site*, összefoglalók (*feature*) összessége. A *feature*-ök modulok csoportosítására szolgálnak. A működését a `feature.xml` írja le, amely felépítése hasonló a `plugin.xml`-éhez. A csoportokban található modulok egyszerre aktiválhatók vagy tilthatók le.

7.1. Update site létrehozása

Először is lét kell hozni a modult, modulokat amelyek az *update site*-ra fognak kerülni helyezni. Ezek után létre kell hozni a csoportokat, amelyek majd felkerülnek az oldalra.

New menüpontban a Plug-in Developement menüben a *Feature Projekt*-el lehet csoportot létrehozni. Ha létrejött a projekt, akkor a már a `plugin.xml`-nél látott grafikus XML szerkesztő eszköz jelenik meg. A kezdő *Overview* lapon kell megadni egy egyedi azonosítót, verziószámot és egy nevet, amellyel majd az *update site*-ot kinyitva megjelenik. Meg lehet adni, hogy ez a modul csoport milyen operációs rendszereket, architektúrát támogat illetve hogy milyen nyelvet. A következő fülön a licenszek, leírások, információk elérhetőségét lehet megadni. Ezek után a következő fülön meg kell adni a csoportba tartozó modulokat, ha ez a létrehozáskor a varázslóban nem lett beállítva, vagy bővítésre szorul a modullista. Minden modulhoz beállítható külön a verziószáma, a méret, amelyet le kell tölteni a telepítéshez és az a méret, amely a telepítéséhez szükséges. Itt is megadható a csoportoknál megadható információk az operációsrendszerre, architektúrára és nyelvre vonatkozóan. Létező csoportot a következő fülön lehet hozzáadni. Itt is be lehet állítani a csoportot jellemző információkat. A következő fülön meg kell adni azt, hogy ha a felhasználó telepíti ezt a csoportot, akkor milyen plusz csoportokra, elemekre van szükség a használatához. Ezeket a függőségeket tallózás segítségével lehet kiválasztani, amelyeket majd a csoport telepítése előtt vagy a telepítésekor telepíteni kell. Ezek után a telepítési részleteket lehet megadni, amelyek opcionálisak. Akinek nem felel meg az Eclipse alap telepítője, az sajátot írhat, amelyet itt kell megadni. Megadható a fordításhoz szükséges elemek, de megfelelő általában az alapállapot. Ha mindent be lett állítva, akkor kész az Eclipse által generált `feature.xml` fájlunk.

Példa egy `feature.xml` fájlra.

```
    <?xml version="1.0" encoding="UTF-8"?>
    <feature
          id="HiberGUICsoportID"
          label="HiberGUICsoport"
          version="1.0.0">
       <description url="https://dev.inf.unideb.hu/svn/hibergui">
          Eclipse IDE Visual Editor plugin-jában lévő paletta eszköz
kiterjesztése, a HiberGUI projekt bizonyos grafikus osztályainak
megfelelő bejegyzésekkel úgy, hogy a
```

```
hibergui.hibernate.HibernateForm.java osztály, a paletta
tulajdonságainak megfelelően, drag-and-drop módszerrel átvihető legyen
bármely SWT alapú vizuális osztályra.
       </description>
       <copyright url="https://dev.inf.unideb.hu/svn/hibergui">
          Copyright Notice
       </copyright>
       <license url="https://dev.inf.unideb.hu/svn/hibergui">
          Licens Agreement
       </license>
       <requires>
          <import plugin="org.eclipse.ui"/>
          <import plugin="org.eclipse.core.runtime"/>
          <import plugin="org.eclipse.ve.java.core" version="1.4.0"
match="greaterOrEqual"/>
          <import plugin="org.eclipse.jdt.core" version="3.4.2"
match="greaterOrEqual"/>
          <import plugin="org.eclipse.jem" version="2.0.201"
match="greaterOrEqual"/>
          <import plugin="org.eclipse.jem.proxy" version="2.0.100"
match="greaterOrEqual"/>
          <import plugin="org.eclipse.ve.cde" version="1.4.0"
match="greaterOrEqual"/>
          <import plugin="org.eclipse.ve.swt" version="1.4.0"
match="greaterOrEqual"/>
          <import plugin="org.eclipse.ve.propertysheet" version="1.4.0"
match="greaterOrEqual"/>
          <import plugin="org.eclipse.gef" version="3.4.1"
match="greaterOrEqual"/>
          <import plugin="org.eclipse.emf.ecore" version="2.4.1"
match="greaterOrEqual"/>
       </requires>
       <plugin
             id="HiberGUI"
             os="linux"
             arch="x86"
             download-size="0"
             install-size="0"
```

```
            version="1.0.0"/>
    </feature>
```

Következő lépés az oldal létrehozása. Az oldal egy egyszerű statikus oldal, amelyet egy szerverre felmásolva egyből működő frissítő oldalt jön létre, de használható helyi lemezről is. Létrehozásához ahol a *Feature Project* lett kiválasztva, ott az *Update Site Project*-et kell kiválasztani. Itt is a grafikus xml szerkesztő kerül elő, amellyel a `site.xml` tartalmát lehet beállítani kényelmesen. Az első fülön kell megadni az oldal tartalmát. Az oldal tartalmához hozzá kell adni az oldalon látni kívánt csoportokat, illetve megadhatóak azok a kategóriák, amelyekbe a csoportokat lehet belehelyezni a jobb átláthatóság kedvéért, végül a *BuildAall* gombbal le kell fordítani az egészet.

Példa egy egyszerűbb site.xml-re

```
    <?xml version="1.0" encoding="UTF-8"?>
    <site>
    Hivatkozás a felhasznált feature-ra.
       <feature url="features/HiberGUICsoport_1.0.0.jar"
id="HiberGUICsoportID" version="1.0.0">
    A kategória megadása még a feature tag en belül.
          <category name="HibernatePluginKategoria"/>
       </feature>
    A featureokban megadott kategóriák definíciója.
       <category-def name="HibernatePluginKategoria"
label="HibernatePluginKategoria"/>
    </site>
```

Ezután vagy fel kell másolni egy szerverre, vagy csak a saját merevlemezről is kipróbálható az oldal. Az előbbiekben írt módon fel kell venni frissítési oldalnak a `site.xml` helyét. Ezután el lehet érni, és telepíteni lehet a csoportot vagy az oldalon lévő összes kategóriát, csoportot. Telepítés és újraindítás után készíteni kell egy projektet

amelyben pedig egy vizuális osztályt. Ezután a projekt könyvtárai közé fel kell venni a modulunkat. Sikeres munka esetén, szerepelni fog a könyvtárlistában, ezzel is megkönnyítve a használatát, mert nem kell betallózni a letöltési helyről, hanem az Eclipse automatikusan felismeri és használhatóvá teszi.

8. Felhasznált eszközök

8.1. Gimp

A GIMP (**GNU Image Manipulation Program**) egy nyilvános licenszű, rasztergrafikus képszerkesztő program. A szoftvert a paletta-bejegyzésekben megjelenő ikonok létrehozására használtuk.

8.2. Verziókövetők

A verziókövető rendszerek fő feladata, a bennünk tárolt adatok megőrzése oly módon, hogy a rajtuk végzett módosítások visszakövethetőek legyenek. Tehát egy olyan eszköz, amelyben az adatainkon végzett módosítások visszavonhatók tetszőleges mélységig. Hasznos lehet csak egy ember számára is, aki biztonsági mentéseket kíván készíteni bizonyos időközönként a munkájáról, mert minden egyes verziószámhoz tartozó verzió tárolásra kerül, de több fős csoportmunka esetén használatosabb inkább. Előnye, hogy több ember fér hozzá ugyanahhoz a kódhoz és a módosítások gyorsan frissíthetők a náluk letöltött forrásokon, amely nagyban megkönnyíti a szinkronizációt. Szerkeszthetik akár ketten is ugyanazt a fájlt egyszerre, ha nincs átfedés kettejük által használt részek között, akkor a verziókövető rendszer összeollózza a munkákat. Az esetleges közös részeket a rendszer konfliktusrészként megjelöli.

Az SVN (Subversion) egy olyan verziókövető rendszer, amelyet a CollabNet Inc. indított el 2000-ben. A szerver egy olyan fájlszerver, amelyen fájlműveletek nem végezhetők, tehát a központi tárolót közvetlenül nem módosíthatják a felhasználók. Dokumentumok, weblapok forráskódok történeti kezelésére lehet használni. Először létre kell hozni a tárolandó adatokat a szerveren, és ezekből a kliensünkre egy munkapéldányt, amelyen majd dolgozhatunk és, amelyet majd frissíthetünk az esetleges változások után.

Ha a szerveren lévő adaton módosítás történik, azaz egy újabb verzió kerül fel, akkor azokat a módosításokat egyszerűen el lehet érni. A saját munkapéldányunkon a módosítások elvégzése után frissíthetjük a szerveren tárolt adatokat a nálunk lévő frissebb adatokra, amelyet ezután a többi szerverhez kapcsolódó felhasználó is el tud érni. Az SVN-t jelenleg is több, nagyobb nyílt forráskódú projektben is használják (GNOME, KDE).

9. Összegzés

Szakdolgozatunkban a Java programozási nyelv hatalmas eszközpalettájának egy érdekes, és sok programozó számára hasznos részét dolgoztuk fel legjobb tudásunk szerint. Mivel a modulfejlesztés témaköre meglehetősen összetett és kiterjedt, ezért ketten kaptuk szakdolgozati témaként. Ez lehetőséget nyújtott számunkra, hogy megtapasztaljuk a csapatmunka előnyeit. Gondos tervezés előzte meg a fejlesztés folyamatát, melynek során az egyes részfeladatokat és a szakdolgozat fejezeteinek kifejtését felosztottuk egymás között. Az egyes részfeladatokat az egyéni érdeklődési kör, és az adott témában való jártasság alapján vállaltuk el, szem előtt tartva, hogy mindkettőnknek pontosan ugyanannyi feladata legyen. Fontos szempont volt még az is, hogy a párhuzamosan végezhető folyamatokat lehetőség szerint párhuzamosan is végezzük. Az egyéni részeredményeket pedig egymással, és témavezetőnkkel való folyamatos konzultációk során beépítettük a létrejövő szoftverbe, vagy témakifejtés esetén, a szakdolgozat szövegébe.

10. Köszönetnyilvánítás

Ezúton szeretnénk köszönetet mondani konzulensünknek Espák Miklósnak, aki témaválasztásunktól szakdolgozatunk befejezéséig figyelemmel kísérte és segítette munkánkat.

11. Irodalomjegyzék

1. IBM Corporation :Eclipse Platform Technical Overview (2009.02.13.)
   http://www.eclipse.org/whitepapers/eclipse-overview.pdf

2. IBM Corporation : Extending The Visual Editor: Enabling support for a custom widget (2008.10.15)
   http://www.eclipse.org/articles/Article-VE-Custom-Widget/customwidget.html

3. Enrico Biermann und Günter Kuhns : Konzeption und Implementierung einer regelbasierten Transformationskomponente für das Eclipse Modeling Framework ----- Diplomarbeit
   2006

4. The Eclipse Foundation :Eclipse Modeling Framework Project (EMF) (2009.03.10.)
   http: //www.eclipse.org/emf

5. Sebastian Brick : Strukturierte Definition und Generierung graphischer Editoren basierend auf der Eclipse-Technologie
   Diplomarbeit
   2007

6. The Eclipse Foundation: The Graphical Editing Framework (GEF) (2009.03.15.)
   http://www.eclipse.org/gef/

7. The Eclipse Foundation: Notes on the Eclipse Plug-in Architecture (2009.03.25.)
   http://www.eclipse.org/articles/Article-Plug-in-architecture/plugin_architecture.html

8. Balogh András: Nyílt Fejlesztőrendszerek kurzus (2009.04.01.)
http://home.mit.bme.hu/~abalogh/

9. IBM Corporation : Visual Editor for Java User Guide : The visual editor palette (2009.04.10)
http://help.eclipse.org/help32/index.jsp?topic=/org.eclipse.ve.doc/topics/cve_palette.html

10. Instantiations : WindowBuilder User Guide (2009.04.10.)
http://download.instantiations.com/D2WBDoc/continuous/latest/docs/html/index.html

11. Cloud Garden :Jigloo SWT/Swing GUI Builder for Eclipse and WebSphere (2009.05.12.)
http://www.cloudgarden.com/jigloo/

12. CollabNet : SVN Documentations(2009.05.23.)
http://subversion.tigris.org/

13. Erich Gamma, Richard Helm, Ralph Johnson, John M. Vlissides: Design Patterns, Elements of Reusable Object-Oriented Software (Addison-Wesley, 1995).