PORTÁLFEJLESZTÉS ÚJ ALAPOKON

Pasztuhov Dániel, dani@iit.bme.hu
Dr. Szeberényi Imre, szebi@iit.bme.hu
BME IIT

Megváltozott a mezőkód

Megváltozott a mezőkód

1 Bevezetés

A 2006-os Networkshop konferencián már bemutattuk a Conflet rendszert [6, 7, 8], mely kifejlesztésének kettős célja volt. Egyrészt célunk volt vele, hogy a felhasználókat olyan barátságos, portálos eszközzel lássuk el, amely segítségével egyszerűen, a feladathoz illeszkedő felületen tudnak párhuzamos és grid feladatokat indítani. Másrészt ezt úgy kívántuk elérni, hogy eközben a fejlesztőnek ne kelljen újra és újra azonos feladatokat megoldani, és folyton hasonló portleteket kifejlesztenie.

Így a Conflet alapfilozófiája, hogy egy keretrendszert biztosít a fejlesztőnek, mely keretrendszer elemeit JSP- és XML- és egyéb fájlok elkészítésével tud elérni, anélkül, hogy a leggyakrabban használt funkciókhoz Java kódot kellene írnia, meggyorsítva ezzel a munkáját.

Felismertük azonban, hogy a munka további gyorsításának érdekében szükséges, hogy a felhasználót még jobban támogassuk, hiszen az esetlegesen nagyszámú konfigurációs fájl karbantartása nem egyszerű feladat. Ez a feladat megfelelő fejlesztőkörnyezettel támogatható. Az általunk kidolgozott megoldás keretét az Eclipse platform [9] adja, mely nyílt és integrált tulajdonságai miatt különösen megfelel ilyen feladatokra.

A cikk röviden ismerteti az Eclipse fontosabb tulajdonságait (2. fejezet), majd bemutatja a fejlesztőrendszerrel szemben támasztott követelményeket (3. fejezet). A 4. fejezet ismerteti az elkészült rendszer architektúráját, a felhasznált Amateras Project-beli EclipseHTMLEditor bővítményt [5], végül néhány érdekesebb sajátosságra hívja fel a figyelmet, melyből képet kaphatunk az Eclipse-programozás szépségeiről.

2 Az Eclipse platform

Az Eclipse nyílt és integrált fejlesztőrendszer.

A nyíltság azt jelenti, hogy interfészek jól definiáltak, bárki számára elérhetők, melynek következménye, hogy a rendszer könnyen bővíthető, és nem kötődik kizárólag egy szállító termékeihez. Bár nem követelmény, általában platform- és ideális esetben programozási nyelv-független egy ilyen rendszer.

Az integráltság jelentése pedig, hogy az eszközöket (általában egy szállító eszközeit) közös keretben fogja össze, ami kényelmessé és hatékonnyá teszi a fejlesztést.

A két tulajdonság egyesítését a bővítmények (plug-in) technológiája teszi lehetővé. A bővítmények kapcsolódási pontjait – melyek lehetővé teszik a bővítmények platformhoz és egymáshoz való kapcsolódását – pontosan rögzítik, elérve ezáltal a nyíltságot, továbbá a bővítmények egymás funkcionalitását bővítik ki, lehetővé téve az integráltságot is.

Integrált, de nem nyílt fejlesztőrendszer pl. a Microsoft Visual Studio .NET eszköze, a Borland eszközei, az Oracle Application Builder. Nyílt és integrált pl. a NetBeans és az Eclipse. [1]

2.1 Eclipse-történelem

Az Eclipse-et az IBM egyik 1996-ban felvásárolt részlege, a fejlesztőkörnyezetek fejlesztésében nagy tapasztalattal rendelkező Object Technology International (OTI) kezdte el fejleszteni 1998-ban. A cél az volt, hogy az IBM különböző fejlesztőeszközeinek közös platformot hozzanak létre, mivel ezek az eszközök úgy néztek ki, mintha különböző gyártóktól származtak volna – nem voltak képesek együttműködni. Az Eclipse fejlesztésének elkezdésével egy időben az IBM egy másik részlege Eclipse-en alapuló eszközt kezdett el készíteni.

Kezdetben a kereskedelmi partnerek vonakodtak az ismeretlen Eclipse-től, így – az elfogadás elősegítésére – az IBM nyolc más szervezettel együtt létrehozta az Eclipse konzorciumot és az eclipse.org-ot.

Az Eclipse első komolyabb kiadásai gyorsan népszerűvé váltak a fejlesztők körében, azonban a piac kétkedéssel fogadta, hiszen úgy tűnt számukra, hogy az Eclipse egy az IBM befolyása alá tartozó termék. E nyomás hatására az Eclipse konzorcium átalakult, létrejött az Eclipse Alapítvány, egy non-profit szervezet, mely a támogató szervezetek által fizetett, de önálló fejlesztői gárdája van. [2, 3]

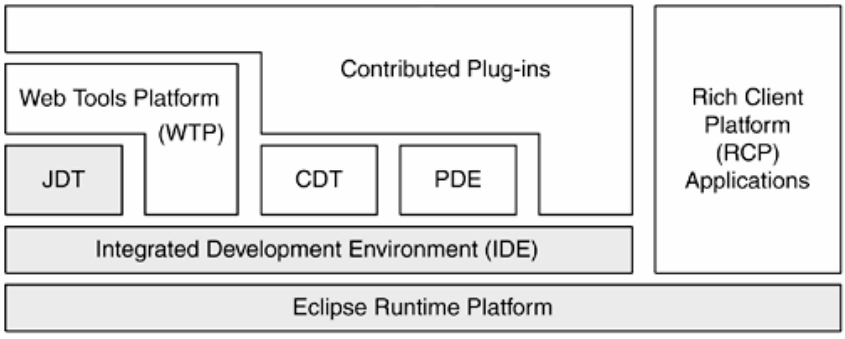
2.2 Bővítmények

Az Eclipse esetében – egy minimális kernelt leszámítva – minden bővítményekkel van megvalósítva. Minden bővítmény két részből áll: egyrészt egy szöveges definíciós részből (manifest.mf és plugin.xml), másrészt az implementációt tartalmazó osztályokból. A definíciós rész néhány adminisztrációs adaton kívül azon információkat tartalmazza, hogy a bővítmény mely más bővítményeket igényel, azokat mely pontokon (kiterjesztési pontok) egészíti ki, és maga milyen kiterjesztési pontokat definiál.

A kiterjesztési pontok névvel, azonosítóval és egy XML Schema fájl segítségével definiálhatók. Az XML Schema fájl írja le, hogy az adott kiterjesztési ponthoz tartozó bővítésnek milyen adatokat kell megadnia magáról. Így a csatlakozás tetszőlegesen alakítható a programozó igényeihez.

2.3 Architektúra

Az 1. ábrán – mely a [4] irodalomból származik – az Eclipse IDE architektúráját láthatjuk. Az feljebb található dobozokkal reprezentált funkciók építenek az alattuk lévőkre.



1. ábra

Az *Eclipse Runtime Platform* biztosítja az Eclipse alapvető szolgáltatásait, melyek a következők:

- **Bővítménykezelő** (Plug-in registry): Betölti és kezeli a bővítményeket. Feloldja és kezeli a függőségeket. A teljesítmény növelése érdekében a bővítmények kezelésekor

„lusta” betöltést alkalmaz, azaz a plug-in osztályait csak akkor tölti be, ha azokra feltétlenül szükség van.

- **Erőforrás-kezelés**: Platformfüggetlen elérést biztosít a fájlok és könyvtárak számára. Kezeli az Eclipse munkaterületét (workspace), amely lehetővé teszi a benne lévő elemek relatív úton való elérését, valamint képes a fájlok változását nyomon követni. A munkaterületet az Eclipse a projektek tárolására használja. A projekt összetartozó fájlok és könyvtárak halmaza, melyből egy (vagy több) végterméket lehet előállítani. Projekt lehet pl. egy Java alkalmazás. Kezeli a fájlkódolásokat, tartalomtypusokat (content type), valamint a hivatkozott erőforrásokat (linked resources), melyek nem a munkaterületen találhatók, de onnan hivatkozottak.
- **Felhasználói felület komponensek** állnak rendelkezésre. Az Eclipse kétféle (de egymásra épülő) lehetőséget biztosít grafikus felület létrehozására. A Standard Widgeting Toolkit (SWT) platformfüggő (de az Eclipse által támogatott platformokon elérhető), natív grafikus elemeket használó könyvtár, míg a JFace magasabb szintű, SWT-re épülő, platformfüggetlen megoldást kínál.
- **Frissítési támogatás**. Mivel bővítmények egymásra épülnek, frissítésük nem kézenfekvő feladat. Ebben nyújt segítséget a Runtime Platform frissítési támogatása, mely URL-lel elérhető helyekről (így távoli, internetes forrásokból) való telepítést és frissítést támogat.
- **Súgó támogatás**. Az Eclipse súgója moduláris szerkezetű, minden bővítmény delegálhat bele tartalmakat. HTML/XML-alapú, az oldalak a helyi számítógépen, vagy akár az interneten is elhelyezkedhetnek, sőt, az oldalak dinamikusan is előállíthatók.

A fejlesztők a Runtime Platform szolgáltatásai egy nagyon kis „kernelben” implementálták, így minden egyéb bővítményként van megvalósítva.

Az *Integrated Development Environment* (IDE) komponens általános célú fejlesztői eszközöket biztosít – anélkül, hogy konkrétan megkötné a felhasznált nyelvet vagy környezetet. Az IDE a következő szolgáltatásokat nyújtja:

- **Megosztott nézetek**et biztosít, melyeket több bővítmény is használhat egyszerre. Többnyire statikus információk megjelenítésére használtak, mint amilyen az erőforrásfa vagy a tulajdonságok nézet.
- **Keresőmotor**, amely kiterjeszthető a környezetre jellemző különlegességekkel.
- **Több szöveges tartalom** megjelenítése, összehasonlítása, egyesítése.
- **Nyomkövetési és hibakeresési támogatás**, mely magában foglalja az erőforrások (stack frame, memória, regiszterek…) kezelését, kiértékelendő kifejezések megadását, változók értékeinek figyelését. Parancsok definiálhatók léptetésre, indításra stb. Természetesen mindezt nyelvfüggetlenül.
- **Ant-támogatás**. Az egyre népszerűbb, javás fordítást segítő eszköz, az Ant – mely a UNIX rendszerekben használt make utódja kíván lenni – használatát teszi lehetővé.
- **Csoportmunka támogatása**. A CVS támogatása beépített, de más verziókezelő eszközök integrációjára is lehetőség van.
- **Perspektívák támogatása**. Az Eclipse-ben több perspektíva is élhet egymás mellett, melyek közül mindig csak egy aktív. A perspektíva meghatározza, hogy milyen nézetablakok legyenek nyitva és azok hogyan helyezkedjenek el a szerkesztőablakhoz viszonyítva.
- **Beállítások ablak**. A beállítások közé minden plug-in delegálhat egy több oldalból álló, hierarchikusan elrendezett struktúrát, melyen a bővítmény különböző beállításai adhatók meg.

A következő tulajdonságokat nem az IDE-ben valósították meg, de a felhasználók számítanak arra, hogy minden környezetben elérhetők:

- **A szerkesztőablak és az áttekintőablak (outline) összekapcsolása**: ha a szerkesztőablak megváltozik, az áttekintőablakban az azonnal észrevehető.
- **Content assist**: A szerkesztő az adott kurzorpozícióban értelmes kiegészítéseket kínál fel Ctrl+Space megnyomására.
- **Sablonok** (template): Olyan kódrészletek, melyek a Content Assist segítségével beilleszthetők a szövegbe.
- **Formázó**. Szintaktikai formázási szabályok állíthatók be és alkalmazhatók a kijelölt vagy a teljes szövegre.
- **Valós időben azonosított problémák**. A hibák és figyelmeztetések nem csak a fordítást követően, hanem gépelés közben láthatóvá válnak.

A *Java Development Tools* (JDT) az IDE komponenst egészíti ki Java nyelvfüggő elemekkel. Tartalmazza a Java nyelv objektummodelljét, mely metódus szintig képes kezelni a Java forráskódot. Képes a lekérdezésen kívül módosítani is azt, sőt, képes kódot is generálni. Továbbá tartalmaz absztrakt szintaxisfát, valamint az IDE-nél leírt általános eszközök javás megfelelőjét (szerkesztő, áttekintőablak, Content Assist, sablonok, formázás, Java nézetablakok, nyomkövető rendszer).

A *C Development Tools* (CDE) a JDT-hez hasonló nyelvi kiterjesztés, azonban nem Javahoz, hanem C-hez és C++-hoz.

A *Plugin Development Environment* (PDE) támogatást nyújt saját bővítmények fejlesztéséhez, a futtatásukhoz, a definíció elkészítéséhez, és a publikáláshoz is.

A *Web Tools Platform* [10] segítségével webalkalmazásokat (azok minden komponensével együtt a statikus HTML-ektől kezdve egészen a servletekig és JSP-kig) tudunk készíteni és az alkalmazásszerverre telepíteni.

A *Contributed Plug-ins* rész a saját bővítményeinknek van fenntartva.

Rich Client Platform: Lehetőségünk van az Eclipse Runtime Platformra közvetlenül építő alkalmazások írására is, kihasználva az Eclipse alapvető lehetőségeit anélkül, hogy fölösleges, a fejlesztőkörnyezetekben használt bővítményekkel terhelnénk a programunkat.

2.4 Miért az Eclipse?

Célunk az volt, hogy fejlesztőeszközt hozzunk létre a Conflet rendszer nagyszámú konfigurációs állományának előállításához.

A lehetőségek számbavételekor felmerült többek között az, hogy egy teljesen új eszközt hozzunk létre a semmiből. Azonban olyan funkciókat is implementálni kellett volna, amik más fejlesztőeszközökben már elérhetők (pl. szintaxisvezérelt színezés, súgórendszer, szerkesztők, különféle megjelenítő-panelek stb), ezek az eredeti feladatot többszörösére hizlalták volna fel. Sokkal jobb ötletnek tűnt egy már meglévő eszköz kiterjesztése.

A szóba jöhető eszközök számbavételekor kiestek a nem nyílt fejlesztőeszközök, valamint – Java-centrikussága folytán – a NetBeans is. Így az Eclipse rendszerre esett a választásunk, ami megfelelő rugalmasságot és platformfüggetlenséget biztosít a további fejlesztésekhez, bővítésekhez.

3 Követelmények

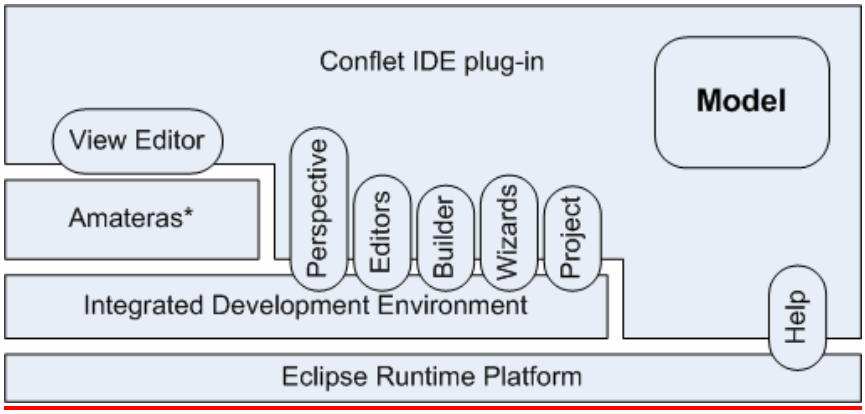
A következőkben azokat a követelményeket ismertetjük, melyek vezérfonalul szolgáltak a fejlesztés későbbi szakaszaiban.

- A plug-in legyen képes új, Conflet konfigurációt tartalmazó projekt létrehozására. A projekthez tartozzon saját perspektíva, melyben a nézetablakok a kényelmes fejlesztés céljából optimálisan vannak elhelyezve.
- A bővítmény legyen képes különféle típusú új fájlokat létrehozni.

- Legyen képes a konfigurációs fájlokat az alkalmazásszerverre való feltöltéshez előkészíteni (összetömöríteni).
- Legyen benne minden fájltípushoz külön szerkesztő. A view és controller fájlokhoz szintaxis-kiemelést tartalmazó szöveges alapú, míg az adatforráshoz, tabs.xml-hez és többnyelvűséget támogató fájlokhoz egyéni – táblázatos jellegű – szerkesztő tartozzon.
- Legyen egy olyan nézetablak, mely – view és controller fájlok esetén – tartalmazza azon parancsokat, melyek a megadott szerkesztőbe beilleszthetők. Ebben a nézetablakban a felhasználó kezdeményezhesse a parancs szerkesztőablakba való beszúrását.
- Legyen olyan nézetablak, mely nyomon követi a megadott kurzorpozíciónál érvényes változókat és azok értékeit. View és controller szerkesztő esetén a felhasználó kezdeményezheti a megadott változóra való hivatkozásnak a szerkesztőbe illesztését.
- Legyen olyan nézetablak, mely megjeleníti a nyelvi kulcsokat, valamint kiválaszthatóan hozzá egy tetszőleges nyelven a megfelelő értékeket. A felhasználó kezdeményezheti a megadott kulcsra való hivatkozás view vagy controller szerkesztőbe való illesztését.

4 A Conflet IDE

Az elkészült Conflet IDE egy olyan Eclipse plug-in, amely az Eclipse különböző részeinek valamint az Amateras projekt EclipseHTMLEditor nevű HTML- és JSP-szerkesztő plug-in [5] módosított változatának kiterjesztési pontjaira épül. Ezt a ráépülést szemlélteti a 2. ábra.



2. ábra

Az ábrán megfigyelhető, hogy a View Editor nevű komponens – amely a Confletben a view fájlok szerkesztésére alkalmas – az Amateras bővítményére épül, míg a Conflet perspektívája, a View Editoron kívüli többi szerkesztő, a Builder komponens – mely a forrásfájlokból a feltöltendő archívum előállítására szolgál –, a különféle varázslók és a Conflet projekthez kapcsolódó funkciók az Eclipse IDE komponensét terjesztik ki. Az Eclipse Runtime Platformra épül a súgó komponens. Emellett a Conflet plug-in tartalmaz még egy viszonylag kezdetleges modellt a Conflet konfiguráció fájljairól.

4.1 Az Amateras plug-in

A Conflet IDE-vel szemben támasztott követelmények kielégítéséhez szükséges volt egy JSP-szerkesztő megírása. Mielőtt azonban saját JSP-szerkesztő írásába kezdtünk volna, megvizsgáltuk az elérhető egyszerű JSP-szerkesztőket, ami a saját igényeknek megfelelően

kiterjeszthető. Megvizsgáltuk a WTP [10] és Lomboz [11] projektekben található szerkesztőket, de úgy találtuk, hogy a szükséges funkcionalitás izolálása túl nagy energia befektetését igényelné, így az nem éri meg. Ezután találtunk rá az Amateras projekt bővítményére, melyben könnyedén azonosítani tudtuk a szükséges részeket.

Az Amateras projekt EclipseHTMLEditor nevű bővítménye képes HTML, JSP, XML, CSS, DTD és JavaScript fájlok szerkesztésére. Funkciói között megtalálható többek között a HTML/JSP/XML/CSS/DTD/JavaScript szintaxiskiemelés, HTML/JSP előnézet, HTML/JSP/XML validáció, Content Assist, HTML/JSP/XML fájlok létrehozásában segítséget nyújtó varázslók, és még sok más.

Ezen funkcionalitások közül csak a JSP-szerkesztő szintaxiskiemelése, a content assist, valamint a JSP-szerkesztő konfigurálható toolbarjára volt leginkább szükségünk. A többit a definíció módosításával üzemen kívül helyeztük.

A bővítményben azonban kódszintű módosítást is kénytelenek voltunk tenni. A változtatások főleg a láthatóságot érintették (néhány tag privát volt védett helyett, így a kiterjesztés során nem alkalmazhattunk az ezek által megvalósított funkciókat), és az általánosságot erősítették.

4.2 A súgó

Mint korábban írtuk, az Eclipse súgójához bármely bővítmény hozzáteheti a saját súgóját. Ebben a fejezetben a súgót vesszük egy kicsit jobban górcső alá.

Az Eclipse súgója könyvekből épül fel. Minden bővítmény létrehozhatja a saját könyvét. Ehhez az „org.eclipse.help.toc” kiterjesztési ponthoz kell egy kiterjesztést definiálnia, melyben megadja, hogy hol helyezkednek el a tartalomjegyzékeket leíró fájlok. A kiterjesztésekben definiálható tartalomjegyzékek lehetnek elsődlegesek (új könyv létrehozására), vagy nem elsődlegesek. Ez utóbbi esetben a könyv egy részletének tartalomjegyzékét írják le.

A tartalomjegyzékekben témákat (topic) adhatunk meg, amelyekhez cím és egy HTML-oldal tartozik. A HTML-oldal lehet lokális gépen tárolt, vagy interneten elérhető is. A témák tartalmazhatnak további témákat (az adott fájlban vagy külön fájlban tárolva), amivel a súgó elemei hierarchikus rendszerbe foglalhatók.

A súgóban – bár nem használtam a funkciót – megadhatók dinamikus tartalmak is, tovább bővítve az Eclipse súgójának lehetőségeit.

4.3 A perspektíva

Mint korábban említettük a perspektívák elsősorban a nézetablakok szerkesztőablak körüli elhelyezésének beállítására, valamint az adott környezetre érvényes beállítások érvénybe léptetésére való.

A perspektíva létrehozásához az „org.eclipse.ui.perspectives” kiterjesztési ponthoz kell kiterjesztést írnunk a plugin.xml-ben. Meg kell adnunk egy azonosítón kívül a perspektíva nevét (ami a perspektívák listáiban fog megjelenni), egy ikont, azt, hogy a felhasználó módosíthatja-e a beállításainkat, valamint a perspektívát megvalósító osztályt.

A perspektívát egy IPerspectiveFactory interfészű osztály valósítja meg. Ennek egyetlen metódusában megadhatjuk, hogy a szerkesztő körül merre (balra, jobbra, fent, lent) milyen nézetet vagy nézeteket szeretnénk elhelyezni, és azok mekkorák legyenek. (Úgy találtuk, hogy a nézetek méretének megadása elég nehezen kezelhető volt). Beállíthatjuk továbbá például, hogy mely varázslókat és nézeteket szeretnénk gyorsabban elérni, azaz a megfelelő listában kiemelni.

4.4 A builder

A builderek olyan osztályok, melyek arra használhatók, hogy forrásfájlokból – legyen az bármilyen fájl – származtatott fájlokat állítson elő. A művelet során megkülönböztethetünk inkrementális és teljes előállítást.

Az inkrementális előállítás során az osztály – belső állapotát felhasználva – csak azon származtatott állományokat állítja újra elő, amelyeket szükséges, megtakarítva egy több száz vagy ezer forrásállományból álló projekt esetén a szükségtelen részek újragenerálását. Cserébe viszont az implementáció sokkal körülményesebb.

Teljes előállítás során a meglévő származtatott állományokat figyelmen kívül hagyjuk, és a forrásfájlokból újra előállítjuk őket.

Ezek alapján a buildert megvalósító osztályban a két módszernek megfelelően két metódust implementálhatunk. A build() metódus az inkrementális előállítást végzi el, míg a fullBuild() metódus a teljes előállításért felelős.

A builder aktiválásához egyrészt ki kell terjesztenünk a „org.eclipse.core.resources.builders” kiterjesztési pontot, ahol megadhatjuk – többek között – a megfelelő osztályt is. Másrészt implementálnunk kell az IncrementalProjectBuilder osztályt kiterjesztő osztályunkat. Harmadrészt a megadott előállító osztályt hozzá kell kötnünk egy projektfajtához (project nature), így a keretrendszer akár az automatikus előállítást is tudja kezelni.

4.5 Szintaxis-kiemelés

A szintaxis-kiemeléshez (syntax highlighting) az Eclipse egy szabályalapú rendszert használ. Felső szinten megadhatjuk, hogy egy adott szöveget milyen partíciókra osszon fel (nyilvánvalóan egy Java kommentre más kiemelési szabályok vonatkoznak, mint egy forrás-szöveg-részletre), majd ezen partíciókhoz hozzárendelhetünk különféle szabályokat megadott sorrend szerint. Amennyiben egy szabálynak megfelel az éppen vizsgált részlet, akkor az megadja, hogy a részlet milyen attribútumokkal (szín, vastagság, háttér, …) rendelkezzen. Ha a szabálynak nem felel meg a részlet, akkor továbblép a sorrendben következő szabályra.

Az Eclipse rendelkezik néhány beépített szabállyal is: Van olyan szabály, amely egy olyan egysoros részletre illeszkedik, amely valamilyen karakterlánccal kezdődik és valamilyen – esetlegesen más – karakterlánccal fejeződik be. Van, ami többsoros részletre illeszkedhet. Van, amelyik a whitespace karakterekre illeszkedik.

Írhatunk saját szabályt is – ezt tettem én is a program megírása során. A programozás során nagy segítségünkre van egy ICharacterScanner interfészű osztály, melyet a környezet ad át. Ettől az osztálytól kérhetünk karaktert a szövegből, vagy – ha szükséges – vissza is tehetünk a szövegbe már egyszer beolvasott karaktereket.

A szintaxis-kiemeléssel kapcsolatos beállításokat egy SourceViewerConfiguration osztályból származó osztályban adhatjuk meg, melyet az adott szerkesztő konstruktorában kell megadnunk.

5 Továbbfejlesztési lehetőségek

Az elkészült fejlesztőrendszer fejlesztése nem ért még véget.

A további integráció érdekében szükséges a Conflet nyelvi modelljét továbbfejleszteni, kiterjeszteni az igényelt funkcionalitásoknak megfelelően.

A kiterjesztett nyelvi modell segítségével további szerkesztőket tudunk írni a konfigurátor minél teljesebb körű kiszolgálása érdekében (pl. lapok közötti kapcsolatok kezelését megjelenítő és kezelő szerkesztő).

6 Köszönetnyilvánítás

E munka részben a Nemzeti Kutatási és Technológiai Hivatal Pázmány Péter programjának (RET-06/2005) támogatásával jött létre. A szerzők szeretnék kifejezni a köszönetüket az EGEE projektnek is (EU INFSO-RI-031688)

7 Összefoglalás

Cikkünkben bemutattuk az elkészült, Conflet konfigurációs fájlok elkészítéséhez használható fejlesztőkörnyezetet, melyet Eclipse bővítményként valósítottunk meg. A fejlesztőkörnyezettel a konfigurátor még egyszerűbben tud a felhasználó számára párhuzamos és grid feladatok indítására alkalmas, feladat-specifikus portál-felületeket létrehozni.

Bemutattuk továbbá a platformként használt Eclipse főbb tulajdonságait, és betekintést nyújtottunk az Eclipse-szel való fejlesztés lehetőségeibe.

8 Hivatkozások

[1] Balogh András, *Nyílt fejlesztőrendszerek*, BME jegyzet, 2006., http://home.mit.bme.hu/~abalogh/open2006/index.html, 2007. február 8. 15:25

[2] Gery Cernosek, *A brief history of Eclipse*, http://www-128.ibm.com/developerworks/rational/library/nov05/cernosek/index.html, 2007. február 8. 15:35

[3] Venu: *History of Eclipse*, 2006. július 21., http://www.venukb.com/blog/2006/07/21/history-of-eclipse/, 2007. február 8. 15:46

[4] D. Carlson, *Eclipse Destilled*, Boston, MA, Addison-Wesley

[5] Amateras Project, http://amateras.sourceforge.jp/

[6] Pasztuhov Dániel, dr. Szeberényi Imre, „Konfigurálható portlet (Conflet) alkalmazása ClusterGrid környezetben”, *Networkshop 2006. konferencia*, Miskolc, https://nws.niif.hu/ncd2006/docs/ehu/015.pdf

[7] Pasztuhov Dániel, *Paraméterezhető portletek fejlesztése ClusterGRID Portál környezetben*, TDK dolgozat, BME, Budapest, 2004.

[8] Pasztuhov Dániel, *Konfigurálható felületű portletek fejlesztése és alkalmazása ipari feladatok megoldásában*, Diplomaterv, BME, Budapest, 2005.

[9] Eclipse.org Home, http://www.eclipse.org/

[10] Web Tools Project, http://www.eclipse.org/webtools/

[11] Lomboz Project, http://forge.objectweb.org/projects/lomboz/

Megváltozott a mezőkód

Megváltozott a mezőkód

Megváltozott a mezőkód

Megváltozott a mezőkód

Formázott: Behúzás: Első sor: 0.75 cm, Nincs felsorolás vagy számozás

Megváltozott a mezőkód

Megváltozott a mezőkód

Megváltozott a mezőkód

Megváltozott a mezőkód