OpenCad.org

Proiectarea unei aplicaţii CAD pentru arhitectură

Absolvent	Silviu-Georgian Aprozeanu
Profesor îndrumător	prof. dr. ing. Florica Moldoveanu

Bucureşti, 10 iunie 2007

Cuprins

Listă de tabele

Listă de figuri

Capitolul 1

Introducere

Capitolul 2

Obiective

Capitolul 3

Tehnologii

3.1 Eclipse Rich Client Platform

Proiectul Eclipse îşi are începuturile alături de compania IBM de care s-a separat în 2003; astăzi este printre cele mai cunoscute proiecte opensource, fiind recunoscut ca un foarte popular mediu de dezvoltare de aplicaţii Java şi nu numai.

Binecunoscutul mediu de dezvoltare Eclipse este dezvoltat pe baza Eclipse Rich Client Platform. Aceasta reprezintă suita minimă de plugin-uri necesare pentru dezvoltarea unei aplicaţii "rich client" [1]. Deşi platforma este destinată dezvoltării de aplicaţii tip IDE, prin înlăturarea anumitor componente ale platformei se pot dezvolta orice tip de aplicaţii desktop.

Structura de bază a acestei platforme poate fi rezumată după cum am prezentat în Tabela 3.1.

3.1.1 Eclipse Runtime

Această componentă a Eclipse RCP este cea mai abstractă şi în acelaşi timp cea mai puternică componentă a sa. Structura sa de bază este descrisă în Tabela 3.2.

[1] eng. *client bogat* – O aplicaţie desktop care beneficiază de un set de elemente de interfaţă evoluate ("bogate"), care oferă o putere mare de abstractizare şi în acelaşi timp control al funcţionalităţii programatorului, păstrînd o experienţă de utilizare plăcută pentru utilizator. În general orice aplicaţie desktop care foloseşte elemente de interfaţă predefinite pînă la un anumit punct poate fi considerată ca un rich client.

Tabela 3.1: Structura Eclipse RCP [7]

Eclipse Runtime	Suport pentru plugin-uri, puncte de extensie şi extensii. Construită peste framework-ul OSGi
SWT	Proiectat să ofere acces eficient şi portabil la facilităţile de interfaţă utilizator ale sistemului de operare pe care este implementat
JFace	Un framework de interfaţă utilizator, construit pest SWT, pentru tratarea multor sarcini de programare de interfeţe
Workbench	Construit peste toate componentele anterioare, aduce un mediu de lucru ultra scalabil, cu interfaţă publică ce suportă mai multe ferestre pentru administrarea vizualizărilor, editoarelor, perspectivelor, acţiunilor, preferinţelor şi multe altele.

Tabela 3.2: Structura Eclipse Runtime [1]

org.eclipse.core.contenttype	Suport pentru definirea şi administrarea tipurilor de fişiere
org.eclipse.core.jobs	Infrastructură pentru programare paralelă în Eclipse
org.eclipse.equinox.registry	Sistemul prin care un plugin publică alte plugin-uri de care depinde şi defineşte puncte de extensie pentru ca alte plugin-uri să-i îmbogăţească funcţionalitatea
org.eclipse.equinox.preferences	O infrastructură prin care un plugin îşi păstrează preferinţele în seturi de perechi cheie/valoare modificabile de către utilizator
org.eclipse.equinox.common	Administrarea proiectelor, resurselor şi fişierelor

Equinox este o implementare a specificaţiilor OSGi R4 [2] o serie de plugin-uri care implementează diverse servicii opţionale OSGi şi alte infrastructuri pentru rularea sistemelor bazate pe OSGi. [3]

Equinox produce, printre alte lucruri, implementarea OSGi folosită de Eclipse RCP (şi celelalte proiecte bazate pe Eclipse) ca un model de componente. [2]

[2] `http://osgi.org/osgi_technology/download_specs.asp?section=2`

3.1.2 SWT

SWT[3] este una din cele mai populare tehnologii dezvoltate de Fundaţia Eclipse. Această tehnologie foloseşte la crearea de interfeţe utilizator în mediul de dezvoltare Java, oferind programatorului capacitatea de a construi interfeţe utilizator portabile între diverse sisteme de operare. În prezent suportă toate sistemele de operare uzuale şi diverse arhitecturi maşină (e.g. x86, amd64).

Structura sa este similară cu cea a altor toolkit-uri similare, ca AWT [4] şi SWING[5]. Elementele fundamentale sunt Shell-ul şi Controlul. Un shell este abstracţia conceptului de fereastră şi reprezintă în general un container pentru controale. Controalele sunt elementele interfeţei utilizator. Butoanele, etichetele, arborii şi tabelele sunt toate controale şi utilizatorii sunt obişnuite cu ele din alte programe ale desktopului. [9] În principiu orice widget poate fi simplu sau compus (i.e. descendent al clasei Composite) şi întreaga lor ierharhie este controlată de relaţiile dintre clasele SWT.

Interacţiunile dintre controale şi utilizator cît şi controlul tranziţiei stărilor controalelor se face cu ajutorul Evenimentelor. Fiecare modificare de stare a unui control (cum ar fi apăsarea pe un buton) este semnalată de SWT către aplicaţie prin declanşarea unui eveniment. Orice Listener (ascultător înregistrat pentru un anumit eveniment) va primi o notificare (i.e. o metodă declarată în interfaţa de Listener va fi apelată) ce va conţine evenimentul ce a avut loc. Orice clasă poate deveni listener prin implementarea unor interfeţe specifice, SWT oferind astfel o mare flexibilitate în cadrul aplicaţiilor care îl folosesc.

Ca şi în AWT sau SWING, poziţia controalelor este stabilită cu ajutorul Layout-urilor. Un layout reprezintă un set de reguli de poziţionare pentru un control compus ce va decide pentru toate controalele incluse cum vor fi ele poziţionate pe ecran. Unele layout-uri asociază anumite date fiecărui control pentru a reţine poziţia în care va fi desenat. Poziţionarea manuală a tuturor controalelor este o problemă complexă şi dacă acel algoritm nu este scris eficient, el poate afecta performaneţele întregii aplicaţii. De aceea, SWT oferă nişte seturi predefinite de layout-uri ce serversc cele mai comune cazuri întîlnite de programatori.

[3] abrev. *Standard Widget Toolkit*

[4] abrev. *Abstract Window Toolkit* – toolkit-ul IU standard Java `http://en.wikipedia.org/wiki/Abstract_Windowing_Toolkit`

[5] Un toolkit IU independent de platformă `http://en.wikipedia.org/wiki/Swing_(Java)`

3.1.3 JFace

JFace este un toolkit pentru interfeţe utilizator ce conţine clase pentru tratarea multora din sarcinile uzuale de programarea interfeţelor utilizator. JFace este independent de sistemul de ferestre atît in interfaţa sa pentru programatori cît şi în implementare, şi este proiectat să funcţioneze cu SWT fără a-i ascunde funcţionalitatea.

JFace include componentele uzuale de toolkit IU, regiştri de imagine şi seturi de caractere, text, dialog-uri, framework-uri pentru preferinţe şi wizard-uri [6] şi raportarea progresului pentru sarcini cu durată mare de execuţie. [4]

Pachete principale din JFace, care oferă funcţionalitatea de bază a tehnologiei sunt prezentate în Tabela 3.3

Tabela 3.3: Structura JFace [8]

Ferestre	org.eclipse.jface.window	Facilităţi de creerea şi administrarea ferestrelor. Interesantă este clasa ApplicationWindow, care aduce un nou nivel de abstracţie peste conceptul de fereastră şi înglobează o buclă program SWT.
Vizualizări	org.eclipse.jface.viewers	Vizualizări ca şi TreeViewer sau TableViewer, care sunt componente ghidate de model ce folosesc widgeturi SWT şi adaptează conţinutul modelului la conţinutul widgetului.
Dialog-uri	org.eclipse.jface.dialogs	Diverse dialog-uri des folosite.
Acţiuni	org.eclipse.jface.actions	Un framework de acţiuni IU similar cu cel găsit în SWING pentru a implementa comportament partajat între două sau mai multe componente, cum ar fi o intrare în meniu şi un buton de pe bara de instrumente.
Wizard-uri	org.eclipse.jface.wizard	Un framework avansat de construcţie a wizard-urilor.
Resurse	org.eclipse.jface.resource	Suport pentru administrarea resurselor, cum ar fi imaginile şi fonturile SWT.
Text	org.eclipse.jface.text	Un framework de crearea, manipularea, afişarea şi editarea documentelor text.

[6] Dialogurile des întîlnite ce automatizează diverse sarcini repetitive

3.1.4 Workbench

Această parte a Eclipse RCP reprezintă o colecţie largă de clase şi interfeţe pentru construirea de interfeţe utilizator complexe.

Iată principalele elemente ce constituie un Workbench.

Workbench

Ca termen strict, se referă la fereastra care conţine întreaga aplicaţie. Defineşte un container abstract pentru toate elementele de interfaţă din aplicaţia dezvoltată cu Eclipse RCP.

Pagina

Reprezintă, în mod intuitiv, toată partea ferestrei care nu conţine barele de unelte şi meniul.

Perspective

Folosesc pentru organizarea conţinutului într-o Pagină. O perspectivă defineşte o colecţie de Vizualizări, aşezarea lor şi acţiunile ce pot fi efectuate pentru o sarcină a utilizatorului. Perspectivele pot fi schimbate de către utilizatorul. Modificarea Perspectivei afectează doar Vizualizările nu şi Editoarele. [1]

Vizualizările şi Editoarele

Reprezintă extensiile aduse de programator într-o aplicaţie Eclipse RCP. Un editor urmăreşte modelul unei maşini de stări în sensul că el are o etapă de deschidere, editare şi o etapă de salvare.

În schimb orice modificare a stării unei Vizualizări este salvată imediat. Vizualizările sunt folosite la a arăta structura modelului deschis într-un Editor, la a facilita accesul la fişiere şi la alte resurse ale unui proiect.

Editoarele urmăresc în general modelul unui editor de text clasic şi oferă aceeaşi funcţionalitate însă la un nivel mai abstract, nefiind dependent de o intrare ca fişier text, nici măcar ca intrarea să fie orice fel de fişier.

3.2 OpenGL şi extensia OpenGL pentru SWT

OpenGL[7] este o specificaţie de standard independent de limbaj ce defineşte un API pentru scrierea de aplicaţii ce produc grafice tridimensionale sau bidimensionale pe calculator. Interfaţa este compusă din peste 250 de apeluri de funcţii ce pot fi folosite pentru a desena scene tridimensionale complexe din primitive simple. OpenGL a fost dezvoltat de SGI[8] în 1992. Este folosit pe scară largă în aplicaţii CAD, realitate virtuală, vizualizare ştiinţifică, vizualizarea informaţiei, simularea zborului, dezvoltare de jocuri pe calculator, ş.a. [5]

3.2.1 OpenGL în Java

Java reprezintă una dintre cele mai puternice platforme de dezvoltare de aplicaţii cunoscute în ziua de azi. Dezvoltată iniţial pentru a separa sarcina de dezvoltare de aplicaţii de platforma pe care va rula, astăzi limbajul Java este folosit în numeroase aplicaţii software, începînd de la clienţi destop pînă la jocuri pentru telefoane mobile şi aplicaţii web. Un astfel de sistem nu putea să rămînă fără o implementare a OpenGL. În prezent, există două extensii OpenGL pentru Java, JOGL şi LWJGL.

JOGL[9] este varianta agreată de Sun şi urmează să facă parte din distribuţia oficială a JDK[10]. Principala sa caracteristică este că foloseşte AWT ca şi suport de interfaţă grafică.

LWJGL[11] este o soluţie mai amplă dedicată dezvoltării de jocuri în OpenGL. Librăria oferă şi suport pentru tratarea sunetului şi a evenimentelor de interacţiune cu utilizatorul, specializate cum am spus după nevoile programării jocurilor pe calculator.

3.2.2 OpenGL pentru SWT

Din păcate, nici una din aceste tehnologii nu se împacă foarte bine cu SWT. Pentru că atît OpenGL cît şi SWT şi AWT implică apeluri native către sistemul de operare, de multe ori aceste apeluri sunt ignorante unele de celelalte şi multe interferenţe apar între interfaţa grafică desenată cu SWT şi fereastra de desenare OpenGL.

[7] abrev. *Open Graphics Library*
[8] abrev. *Silicon Graphics Inc.*
[9] Java OpenGL Library `http://jogl.dev.java.net`
[10] După cum precizează JSR231 `http://jcp.org/en/jsr/detail?id=231`
[11] Lightweight Java Game Library `http://lwjgl.org/`

Din aceste raţiuni, o a treia variantă a fost aleasă de noi, anume extensia OpenGL pentru Java făcută de Eclipse pentru SWT. Este de menţionat că rezultate similare s-au obţinut şi LWJGL, care la rîndul său poate fi integrat cu puţin efort cu SWT. JOGL din păcate, datorită legăturii sale puternice cu AWT, nu poate fi adus în scopul aplicaţiei noastre într-un mod care să poate fi uşor reproductibil de autor.

Limitările bibliotecii OpenGL pentru SWT făcute de Eclipse se referă la versiunea de OpenGL suportată (i.e. OpenGL 1.1) şi la faptul că interesul pentru această tehnologie este oarecum scăzut, ultima versiune existentă datează din septembrie 2005[12] şi sunt şanse ca versiuni ulterioare să nu mai existe, pe cînd celelalte două librării sunt în continuă dezvoltare.

La momentul determinării acestei alegeri, după o testare a tuturor tehnologiilor disponibile, am ajuns la concluzia că funcţionalitatea oferită de varianta Eclipse a legăturii cu OpenGL este atît suficientă ca şi capaciţăţi necesitate de acest proiect cît şi destul de stabilă pentru a oferi o calitate înaltă de producţie.

3.3 Raţiunea alegerii tehnologiilor

Încă de la începutul dezvoltării acestui proiect, s-a pus problema tehnologiei care să fie folosită pentru îndeplinirea cît mai facilă a obiectivelor proiectului. Munca de cercetare în privinţa alegerii tehnologiei potrivite a implicat o documentare amplă asupra tehnologiilor luate în considerarea cît şi testarea practică a multora dintre ele.

Vom încerca mai jos să trecem în revistă o serie de alegeri care le-am făcut şi raţiunea ce stă în spatele lor, prin prisma avantajelor şi dezavantajelor aduse de alegerile făcute.

3.3.1 PHP şi Python

Consideraţia asupra alegerii limbajului de programare în care urma să fie dezvoltat acest proiect a început desigur de la aria cunoştinţelor autorului. Alegerea s-a făcut astfel între C, C++ şi Java. Alte alternative mai ,,exotice" au fost luate în considerare marginal, cum ar fi Python[13] sau PHP[14].

[12] http://www.eclipse.org/swt/opengl/
[13] http://www.python.org/
[14] http://www.php.net/

Capacităţile limbajelor scripturale de a intercaţiona cu libării native şi de a crea interfeţe grafice a cunoscut o dezvoltare masivă în ultima perioadă. Există pentru Python PyGTK[15], o librărie ce permite programarea facilă a intefeţelor grafice cu GTK[16] sub Python. O extensie similară există pentru PHP, numită PHP-GTK[17].

Din punctul de vedere al graficii tridimensionale, extensii similare există pentru ambele limbaje, oferind funcţionalitate similară cu extensia prezenză în limbajul Java.

La nivel de dezvoltarea de aplicaţii desktop complexe, pentru Python există framework-ul DABO[18] bazat pe wxPython[19]. Apropierea acestor sisteme de cerinţele noastre nu a fost evidentă, cît şi popularitatea relativă a acestor proiecte a făcut considerarea lor doar trecătoare.

3.3.2 C şi C++

Revenind la principalii candidaţi, limbajele tradiţionale pentru dezvoltarea de aplicaţii software nu puteau să treacă neobservate autorului. Există o suită impresionantă de aplicaţii desktop dezvoltate exclusiv doar pe baza acestor două limbaje de operare.

Ca şi set de widgeturi interesante pentru noi, menţionăm librăriile GTK, QT[20] şi bineînţeles .NET Framework[21].

Programarea la nivel de bază, cum este cea în limbajul C nu a fost niciodată agreată de autor pentru capacităţile limitate de scalare (sau viteza redusă de scalare şi efortul depus pentru scalare), lipsa unei integrări independente de platformă la un nivel superior şi cu o structură coerentă, orientată obiect.

Aceste considerente au cîntărit greu împotriva bibiliotecii GTK, cea mai populară arhitectură open-source de dezvoltare de aplicaţii grafice în C, ce stă la baza sistemului desktop GNOME[22].

În contrabalans a stat librăria gtkmm[23], o extensie a lui gtk pentru C++. Pînă în ziua de

[15] `http://www.pygtk.org/`

[16] abrev. *The GIMP Toolkit* `http://www.gtk.org/`

[17] `http://gtk.php.net/`

[18] `http://dabodev.com/` Un framework orientat pe o funcţionalitate client similară aplicaţiei Visual FoxPro

[19] `http://www.wxpython.org/` un framework similar cu wxWidgets (despre care vom discuta ulterior), ce oferă portabilitate pentru codul intefeţei grafice între diverse sisteme de operare

[20] `http://en.wikipedia.org/wiki/Qt_(toolkit)`

[21] `http://msdn.microsoft.com/netframework/`

[22] `http://www.gnome.org/`, un sistem destkop complet, întîlnit în majoritatea distribuţiilor Linux cît şi a majorităţii altor platforme UNIX

[23] `http://www.gtkmm.org/`

astăzi, o despărţire clară de această tehnologie din partea autorului nu există. Unicele considerente împotriva acestei alegeri rămîn dependinţa de limbajul C++ şi lipsa unei portabilităţi între sisteme de operare de o stabilitate apreciabilă.

Răspunsul acestor lipsuri a părut să îl aibă wxWidgets[24]. O platformă cu o tradiţie considerabilă în dezvoltarea de aplicaţii client, wxWidgets reprezintă poate cea mai bună alegere pentru un dezvoltator de aplicaţii client portabile sub C++, în opinia noastră.

Însă o analiză mai profundă a acestei platforme a scos la iveală faptul că wxWidgets există ca şi sistem de dezvoltare de aplicaţii client dinaintea extensiei STL[25] pentru C++. De departe – în opinia autorului – cea mai atractivă componentă a limbajului C++ suferă de un suport consistent pentru wxWidgets.[6] Această caracteristică a bibliotecii a reprezentat un criteriu de nedepăşit în alegerea noastră.

Biblioteca QT a trebuit să fie dată la o parte pe baza principiului portabilităţii. Din păcate, există puţine implementări de succes de software implementat cu QT pentru platforma Windows. De asemenea, QT nu este distribuit sub o licenţă compatibilă cu intenţiile de distribuţie ale autorului.

3.3.3 .net Framework

Dacă pînă acum ne-am uitat la tehnologii care pornesc dinspre domeniul open-source şi Linux, .net Framework reprezintă răspunsul gigantului Microsoft la evoluţia sistemelor software moderne.

Cu o structură de clase avansată, .net Framework este printre puţinele soluţii software atît de impresionante care nu au reuşit să-şi găsească un răspuns rezonabil în lumea opensource.

Deşi reprezintă ţelul unui întreg proiect, mono[26], la data la care am făcut cercetările noastre, portabilitatea sistemului .net pentru platforme UNIX nu a ajunsese încă, în opinia autorului, la o maturitate demnă de luat în seamă. Aflat într-o continuă dezvoltare, proiectul mono promite să realizeze o revoluţie în ceea ce priveşte portabilitatea programelor software pe diverse sisteme de operare.

[24] `http://www.wxwidgets.org/`
[25] abrev. *Standard Template Library* `http://en.wikipedia.org/wiki/Standard_Template_Library`
[26] `http://www.mono-project.com`

3.3.4 Java

Dintre toate sistemele de dezvoltare software existente astăzi în lumea dezvoltatorilor de aplicaţii desktop, noi am decis că Java reprezintă cel mai convenabil, uşor de dezvoltat, uşor de menţinut şi uşor de distribuit mod de a dezvolta proiectul de faţă.

Cu o istorie legendară de realizări, Java reprezintă poate cel mai dezvoltat sistem software existent, cu biblioteci dezvoltate de un număr impresionant de programatori din întreaga lume, cu o serie de tehnologii pentru toate aplicaţiile software cunscute.

Mai mult, deschiderea perspectivelor software către alte platforme decît cele tradiţionale (i.e. Mac, PC), cu dezvoltarea din ce în ce mai mare a tehnologiilor web şi a nevoilor crescînde în domeniul server, Java tinde să devină cel mai variat sistem software existent vreodată. În ceea ce priveşte dezvoltarea aplicaţiilor client, Java a cunoscut o serie de etape remarcabile pe parcursul evoluţiei sale. Primul sistem de interfeţe pentru utilizatori a fost AWT, tehnologie care permitea legarea apelurilor native de construcţie a widget-urile grafice de o structură de clase abstractă şi independentă de platforma de execuţie. Dezvoltarea tehnologiei a fost însă prematur întreruptă, fiind folosită astăzi doar sporadic în dezvoltarea de aplicaţii client, acolo unde performanţa este considerată prioritară uşurinţei de utilizare a aplicaţiei.

A urmat SWING, care reprezintă răspunsul dezvoltatorilor Java la problemele utilizatorilor cu platforma AWT. Rămînînd complet în spectrul Java, codul bibliotecii SWING nu interacţionează direct cu sistemul de operare pe care rulează aplicaţia ci alege să-şi deseneze întreaga interfaţă folosind libăria Java 2D[27]. Pierzînd contactul cu interfaţa nativă, SWING a avut dintotdeauna problema performanţei. Evoluţia tehnologiei a adus progrese semnificative în acest sens. SWING aducea de asemenea o nouă formă a interfeţei grafice, diferită de orice ce altă interfaţă a sistemelor de operare.

Acest aspect este considerat de mulţi un atu al tehnologiei, permiţînd aplicaţiei să arate la fel pe sisteme de operare diferite. Neajunsul este însă faptul că utilizatorii noi trebuie să se obişnuiască cu o nouă interfaţă grafică cînd deschid aplicaţia, lucru ce poate afecta prima impresie şi curba de învăţare a utilizatorilor.

Eclipse a început proiectul SWT pentru dezvoltarea aplicaţiei Eclipse. Proiectul dorea să ofere tot ce AWT nu a reuşit să ofere. Printre caractersticile principale ale SWT este că foloseşte un număr mai mare de widget-uri comune decît AWT şi că pentru platformele UNIX foloseşte

[27] `http://java.sun.com/products/java-media/2D/`

biblioteca gtk de widget-uri, faţă de biblioteca motif[28] folosită de AWT, o tehnologie care astăzi a fost lăsată în urmă pentru dezvoltarea de interfeţe grafice în sisteme UNIX/Linux.

3.3.5 Eclipse RCP

În final, vom încerca să motivăm şi alegerea folosirii RCP pentru dezvoltarea acestui proiect. Considerentele de bază au pornit în primul rînd de la succesul aplicaţiei Eclipse ca şi un IDE pentru documente Java, aplicaţii web, ş.a. Cunoaşterea şi familiarizarea cu modul de lucru al acestui IDE au dus inevitabil la recunoaşterea calităţilor de interfaţă utilizator ale Eclipse RCP, care stă la bază Eclipse IDE.

Deşi RCP a fost proiectat în principal pentru editarea documentelor de tip text, am considerat că experimentarea cu un nou tip de model de editare, cea de modelare tridimensională poate deschide o nouă perspectivă în aria de aplicaţii a Eclipse RCP.

După iniţierea dezvoltării proiectului, mare parte a premizelor asupra uşurinţei de extindere şi dezvoltare a platformei s-au confirmat. Bineînţeles, volumul cel mai mare de muncă l-a cerut „învăţarea" lui Eclipse să editeze un model tridimensional, sarcină care la rîndul ei a fost facilă datorită abstracţiei puternice de care se bucură structura aplicaţiilor dezvoltate în RCP.

[28] `http://www.opengroup.org/motif/`

Capitolul 4

Arhitectura aplicaţiei

Proiectată privind către viitor, aplicaţia noastră oferă posibilitatea extinderii pe viitor în numeroase direcţii de dezvoltare. Am construit o platformă solidă pe baza căreia se poate dezvolta o aplicaţie CAD de un nivel comparabil cu a altor aplicaţii de acest tip.

Vom orienta prezentarea noastră în două direcţii. Întîi ne vom concentra asupra modelului de lucru. El reprezintă inima proiectului, un concept care uneşte în el toate capacităţile acestui proiect. Apoi vom trece la descrierea interfeţei cu utilizatorul, a cărei concepţie în sine prezintă o considerăm remarcabilă şi reprezintă o bază pentru dezvoltări ulterioare.

4.1 Modelul de lucru

Definiţia 4.1 *Numim **Model de lucru** reprezentarea într-o colecţie de obiecte Java a unei structuri arhitecturale ce poate fi modelată cu ajutorul acestui proiect.*

Modelul de lucru este o reprezentare a unei structuri pe care proiectantul doreşte să o reprezinte cu această unealtă într-o formă pe care programul o poate recunoaşte şi o poate reface cît mai fidel cu modelul real imaginat de proiectant.

Definiţia 4.2 *Numim **Primitivă** unitatea structurală şi funcţională a Modelului de lucru. O primitivă poate fi o colecţie de alte primitive. De asemenea, o primitivă este un element ce poate fi materializat atît la nivel logic cît şi într-o reprezentare grafică.*

Primitivele sunt analogiile entităţilor logice în care un model real poate fi divizat pentru a-l reprezenta structurat. În seria de primitive ce pot apare în modelul logic pot exista primitive care nu au un analog imediat în lumea reală.

Propoziţia 4.1 *Modelul logic în totalitatea lui este o Primitivă.*

Plecînd de la aceste două noţiuni, vom încerca să prezentăm întreaga structură a Modelului de lucru şi modul în care acesta interacţionează cu aplicaţia în sine.

În esenţă, o primitivă este orice poate fi desenat. De aceea, la rîndul său modelul este o primitivă. Introducem două direcţii în care modelul poate fi reprezentat.

Definiţia 4.3 *Numim **Reprezentare Reală** o materializare strictă în comenzi Java/OpenGL a unei reprezentări schematice tridimensionale a unei Primitive, realizată cu scopul formării unei imaginii asupra rezultatului final al construcţiei fizice a entităţii logice din spatele acelei primitive.*

Modelul tridimensional este deci cea mai apropiată formă de realitate pe care acest proiect o va reda pentru utilizatorii săi.

Definiţia 4.4 *Numim **Reprezentare Editabilă** o materializare strictă în comenzi Java/OpenGL a unei reprezentări schematice bidimensionale a unei Primitive, realizată cu scopul identificării vizuale a primitivelor şi accesării facilă prin intermediul unui editor a proprietăţilor primitivelor.*

Reprezentarea Editabilă este apropiată ca raţiune figurilor din desenul tehnic, şi de aceea identitatea lor vizuală este la rîndul ei asemănătoare acelor figuri. Alegerea acestor noţiuni este în strictă corelaţie cu detaliile de implementare ale aplicaţiei, care le vom detalia în următorul capitol. Pentru a ne face o idee despre cum aceste două reprezentări se potrivesc peste model, vom spune că reprezentarea modelului de lucru este juxtapunerea tuturor reprezentărilor celorlalte primitive existente în model, în ambele reprezentări. Desigur, asta implică că modelul în sine nu este decît un simplu container logic pentru toate celelalte componente ale aplicaţiei.

4.1.1 Tipuri de primitive

Alegerea setului de primitive de care va dispune această aplicaţie a fost o sarcină dificilă. Timpul de implementare este în directă corelaţie cu volumul de primitive care trebuiesc implementate.

Vom lăsa ca un exerciţiu viitor adăugarea de noi primitive care ar fi de un real ajutor utilizatorului acestei aplicaţii. Setul ales este considerat de autor ca fiind suficient pentru a oferi un set de funcţionalităţi de bază utilizabile pentru această aplicaţie şi destul de variate pentru a scoate în evidenţă potenţialul de creştere al acestui proiect.

Ziduri şi Colţuri de Ziduri

Elementele constructive esenţiale ale unei structuri sunt zidurile. Ele delimitează forma şi suprafaţa construcţiei, avînd un impact crucial asupra preţului de construcţie şi facilităţile ce vor putea fi oferite de acea construcţie.

Colţurile sunt un tip de primitivă indirect disponibilă proiectantului. Unirea capetelor a două ziduri se poate face printr-un colţ, care introduce de fapt o legătură permanentă între marginile acelor două ziduri. Constrîngerea este una punctuală, zidurile putînd avea orice orientare în cadrul modelului atîta timp cît sunt conectate la un colţ. Colţurile pot fi privite ca nişte puncte într-un plan.

Orice zid este conectat la două colţuri, **Colţul de Start** şi **Colţul de Stop**. Astfel poziţionat, lungimea şi orientarea unui zid este dictată de poziţia celor două colţuri. Astfel, un zid poate fi privit ca un segment ce leagă oricare două puncte (colţuri) dintr-un plan.

Caracteristici de Ziduri

Definiţia 4.5 *Numim **Caracteristică a unui Zid** orice Primitivă ce este asociată în mod direct cu un zid.*

Toate Caractersticile sunt constrînse poziţional în lungimea zidului. Ele adaugă elemente suplimentare modelului care sunt strict legate în realitate de existenţa unui zid. Exemple tipice de caracterstici ce elucidează şi mai bine ce reprezintă o caracteristică ar fi:

- Fereastră
- Uşă
- Deschidere – orice trecere completă prin volumul unui zid (o intrare fără uşă, sau deschiderea unei ferestre fără o fereastră montată în ea)

- Cavitate – orice intrare parţială în volumul unui zid, ca un raft interior într-un perete fals.
- Ingroşăre – orice adăugare la volumul unui zid, o îngroşare într-o anumită zonă, ce poate servi, prin analogie, ca un raft exterior într-un perete.

Decoraţiuni

Definiţia 4.6 ***Decoraţiunile*** *sunt Primitive care nu au legătură strictă cu structura construită, însă oferă un plus de realism şi poate sugera viitoarea destinaţie a diferitelor spaţii din model.*

- Canapea
- Masă
- Toaletă
- Chiuvetă de baie
- Vană
- Chiuvetă de bucătărie
- Aragaz
- Frigider
- Masă de bucătărie

Dintre aceste Decoraţiuni, unele dintre ele vor fi implementate ca şi Caracterstici, datorită constrîngerii lor reale de a apărea pe un zid. Dintre cele menţionate mai sus, dăm exemplu chiuveta care întotdeauna apare în vecinătatea unui perete.

În Figura 4.1 vom sumariza cele prezentate mai sus printr-o diagramă cu toate componentele modelului de lucru.

4.2 Editorul OpenGL

Înainte de a fi un editor pentru Modelul de Lucru (Definiţia 4.1), editorul OpenGL dezvoltat de noi este un punct de plecare foarte bun pentru orice editor care necesită folosirea facilităţilor OpenGL.

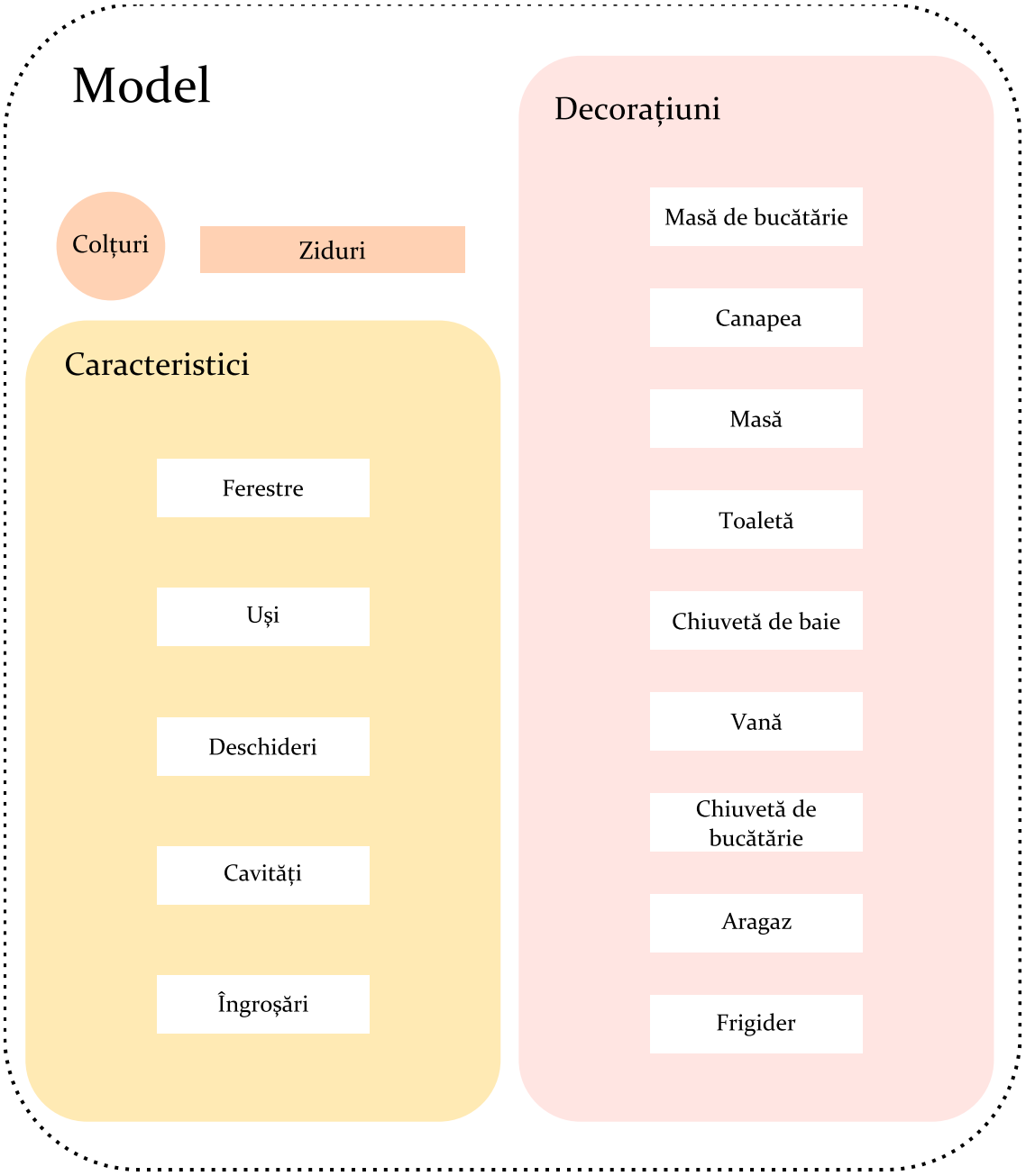

Figura 4.1: Arhitectura Modelului de Lucru

Vom încerca să descriem succint arhitectura editorului, evidenţiind caracteristicile generice cît şi particularizările de suprafaţă care au fost necesare pentru integrarea cu funcţionarea dorită.

Editorul este în principiu o suprafaţă de desenare OpenGL cu aspect bidimensional ce poate interacţiona cu utilizatorul prin intermediul interfeţei oferite de RCP, adică evenimentele ale mausului, ale tastaturii, ale vizualizărilor de structură şi de proprietăţi, despre care vom aminti mai jos.

4.2.1 Aspecte ale modelului

Definiţia 4.7 *Numim **Aspect al Modelului de Lucru** orice formă de interpretare a unei primitive aflate într-un model de lucru care nu are legătură cu modelul şi structura sa, dar ajută alte componente ale aplicaţiei în a interacţiona cu acesta.*

Pentru modelarea funcţionalităţii editorului, orice Primitivă (Definiţia 4.2) este considerată ca un element ce poate fi desenat într-un editor. Fiecare Primitivă trebuie să specifice modul în care ea urmează să fie desenată în spaţiul bidimensional al editorului.

Unele primitive sunt desenate direct de către model, altele, cum ar fi Caracteristicile (Definiţia 4.5), ale căror desenare intră în sarcina zidului din care fac parte.

Acest aspect al modelului reprezintă materializarea **Reprezentării Editabile** (Definiţia 4.4).

Un alt aspect adăugat modelului este **Selectabilitate**. Orice Primitivă care din punct de vedere al interfeţei editorului poate fi selectat prin diverse metode de utilizator se spune că este **selectabilă**. Primitivele pot reacţiona la schimbarea stării lor de selectabilitate şi la rîndul său editorul poate trata evenimentul schimbării selecţiei în funcţie de implementarea dorită.

În fine, ultimul aspect al modelului necesar implementării editorului este cel de **Navigabilitate**. În momentul în care mausul trece deasupra unei astfel de componente, ea poate reacţiona prin evenimente implementate la nivelul editorului. Multe Primitive folosesc această proprietate pentru a-şi schimba starea de selectare.

4.2.2 Editorul ca maşină de stări

Editorul OpenGL funcţionează ca o maşină de stări. Editorul suportă înregistrarea stărilor noi şi controlează viaţa tuturor stărilor înregistrate în stiva sa de stări.

La un moment dat o singură stare este activă. O stare activă poate controla toate evenimentele pe care le primeşte editorul şi poate să introducă noi stări în stivă sau să modifice modelul în funcţie de evenimentele care au avut loc. Starea decide cînd viaţa ei a luat sfîrşit. Această decizie este comunicată editorului care apoi deînregistrează starea din stivă.

În Tabela 4.1 vom prezenta ciclul de viaţă al unei stări. Acest model de ciclu de viaţă a fost stabilit în funcţie de necesităţile editorului OpenGL, pentru a-i permite acestuia să interacţioneze sincron cu schimbările de stare ce pot fi controlate asincron de către evenimentele aplicaţiei.

Dintr-o anumită privinţă, acest ciclu de stare poate fi privit ca o serie de meta-stări ale aplicaţiei (i.e. stări ale stărilor).

4.2.3 Facilităţile Editorului OpenGL

Bla bla bla

Tabela 4.1: Ciclul de viaţă al stărilor Editorului OpenGL

FRESH	La construcţia unei stări ciclul de viaţă este setat în poziţia FRESH. Editorul va citi orice stare setată pe acest mod şi o va seta ca şi stare activă, înregistrînd toate rutinele de tratare a evenimentelor pentru această stare în cadrul editorului. Starea curentă este setată ca activă şi modul ei este trecut în RUNNING. Starea activă anterioară este trecută pe modul SLEEPING.
RUNNING	După ce editorul iniţializează o stare, ea este în modul RUNNING. În acest mod starea primeşte toate evenimentele editorului. Decizia de a părăsi această stare se poate lua după două criterii. a) La discreţia stării în sine, prin trecerea în modul TERMINATED sau b) la discreţia editorului, în momentul sosirii unei noi stări FRESH, cînd starea curentă trece în SLEEPING.
SLEEPING	Toate stările care au fost oprite de editor la apariţia unei stări noi se adună într-o stivă şi modul lor este trecut în SLEEPING. Ele pot sta în acest mod un timp nedefinit şi pot să nu-l părăsească niciodată. Ele pot reveni la starea RUNNING în momentul în care se află la vîrful stivei şi starea activă trece în modul TERMINATED. Atunci editorul automat va muta starea din capul stivei ca stare activă şi ea va deveni RUNNING. De aceea rutina de tratare a tranziţiei în RUNNING cît şi rutina de tratare în starea TERMINATED trebuie să aibă în vedere posibilitatea rulării de mai multe ori pentru aceeaşi stare (i.e. să nu trateze iniţializări, care se fac în mod normal în starea FRESH)
TERMINATED	Odată ce starea consideră că şi-a încheiat activitatea, ea poate alege să treacă în modul TERMINATED. Odată ajunsă în acest mod, starea va fi deînregistrată din lista de captare a evenimentelor editorului şi va fi scoasă permanent din lista stărilor editorului.

Capitolul 5

Implementare

Capitolul 6

Concluzii

Bibliografie

[1] Eclipse sdk documentation
http://help.eclipse.org/.

[2] Equinox frequently asked questions
http://www.eclipse.org/equinox/faq.php.

[3] Equinox
http://www.eclipse.org/equinox/.

[4] Jface
http://wiki.eclipse.org/index.php/JFace.

[5] Opengl on wikipedia
http://en.wikipedia.org/wiki/OpenGL.

[6] wxwidgets faq: General
http://www.wxwidgets.org/docs/faqgen.htm#stl.

[7] Martin Oberhuber, Nick Boldt, Daniel Spiewak, Thomas Bonk, et al. Rich client platform frequently asked questions
http://wiki.eclipse.org/index.php/RCP_FAQ.

[8] Brian Sam-bodden and Christopher Judd. Rich clients with the swt and jface
http://www.javaworld.com/javaworld/jw-04-2004/jw-0426-swtjface.html.

[9] Joe Winchester and Steve Northover. Swt - a native widget toolkit for java
http://java.sys-con.com/read/37463.htm.