UNIVERSITATEA „ALEXANDRU IOAN CUZA” IAŞI

**FACULTATEA DE INFORMATICĂ**



LUCRARE DE LICENŢĂ

**propusă de**

***Anca-Luiza Adăscăliței***

**Sesiunea:** *Iulie, 2017*

**Coordonator ştiinţific**

**Lector, Dr. Cristian Frăsinaru**

**UNIVERSITATEA ALEXANDRU IOAN CUZA IAŞI**

**FACULTATEA DE INFORMATICĂ**

***Anca-Luiza Adăscăliței***

**Sesiunea:** *Iulie, 2017*

**Coordonator ştiinţific**

**Lector, Dr. Cristian Frăsinaru**

Declarație privind originalitatea și respectarea drepturilor de autor

Prin prezenta declar că Lucrarea de licenţă cu titlul “Aplicație pentru crearea orarului (CodeHour)” este scrisă de mine şi nu a mai fost prezentată niciodată la o altă facultate sau instituţie de învăţământ superior din ţară sau străinătate. De asemenea, declar că toate sursele utilizate, inclusiv cele preluate de pe Internet, sunt indicate în lucrare, cu respectarea regulilor de evitare a plagiatului:

* toate fragmentele de text reproduse exact, chiar şi în traducere proprie din altă limbă, sunt scrise între ghilimele şi deţin referinţa precisă a sursei;
* reformularea în cuvinte proprii a textelor scrise de către alţi autori deţine referinţa precisă;
* codul sursă, imaginile etc. preluate din proiecte *open*-*source* sau alte surse sunt utilizate cu respectarea drepturilor de autor şi deţin referinţe precise;
* rezumarea ideilor altor autori precizează referinţa precisă la textul original.

Iaşi, *data*

Absolvent *Anca-Luiza Adăscăliței*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(semnătura în original)

Declarație de consimțământ

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul “Aplicație pentru crearea orarului (CodeHour)”, codul sursă al programelor şi celelalte conţinuturi (grafice, multimedia, date de test etc.) care însoţesc această lucrare să fie utilizate în cadrul Facultăţii de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iași să utilizeze, modifice, reproducă şi să distribuie în scopuri necomerciale programele-calculator, format executabil şi sursă, realizate de mine în cadrul prezentei lucrări de licenţă.

Iaşi, data

Absolvent Anca-Luiza Adăscăliței

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(semnătura în original)

Acord privind proprietatea dreptului de autor

Facultatea de Informatică este de acord ca drepturile de autor asupra programelor-calculator, format executabil şi sursă, să aparţină autorului prezentei lucrări, *Anca-Luiza Adăscăliței.*

Încheierea acestui acord este necesară din următoarele motive:

Doresc continuarea dezvoltării aplicației si posibilitatea oferirii codului sursă al acesteia altor instituții școlare spre a fi folosită pentru generarea orarului. Codul sursă al programelor este realizat de către mine și doresc să am posibilitatea de a dezvolta variante îmbunătățite ale aplicației pentru ca aceasta să corespundă cât mai precis nevoilor instituțiilor școlare.

Iaşi, *data*

|  |  |  |
| --- | --- | --- |
| Decan *Adrian Iftene* | | Absolvent *Anca-Luiza Adăscăliței* |
|  |  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| (semnătura în original) | | (semnătura în original) |

Cuprins

[Introducere 7](#_Toc486256191)

[1. Motivație 7](#_Toc486256192)

[2. Tehnologii folosite 7](#_Toc486256193)

[3. Aplicații similare 9](#_Toc486256194)

[4. Funcționalități 10](#_Toc486256195)

[5. Structura lucrării 12](#_Toc486256196)

[Contribuții 14](#_Toc486256197)

[Cap.1. Structura aplicației 15](#_Toc486256198)

[1.1 Arhitectura aplicației 15](#_Toc486256199)

[1.2 Proiectarea XML-urilor 16](#_Toc486256200)

[1.3 Interfața 25](#_Toc486256201)

[1.4 Raportul 28](#_Toc486256202)

[Cap.2. Implementarea 30](#_Toc486256203)

[2.1 Citirea din și scrierea în XML 30](#_Toc486256204)

[2.2 Organizarea claselor 31](#_Toc486256205)

[2.3 Algoritmul de rezolvare a problemei orarului 36](#_Toc486256206)

[2.4 Constrângerile implementate 39](#_Toc486256207)

[Cap.3. Direcții de dezvoltare 44](#_Toc486256208)

[3.1 Programare bazată pe constrângeri 44](#_Toc486256209)

[3.2 Algoritmi genetici 45](#_Toc486256210)

[3.3 Java Web Start 46](#_Toc486256211)

[Cap.4. Cazuri de utilizare 48](#_Toc486256212)

[Concluziile Lucrării 51](#_Toc486256213)

[Bibliografie 52](#_Toc486256214)

Introducere

1. Motivație

Aplicația CodeHour are ca scop asistarea utilizatorilor în crearea orarului unei instituții școlare. Nevoia care a ghidat dezvoltarea acestei aplicații a fost lipsa unei aplicații desktop capabilă să rezolve eficient problema orarului, care sa aibă o interfață intuitivă și ușor de utilizat și care să nu necesite multe resurse hardware pentru a rula. Scopul dezvoltării CodeHour este acela de a oferi un instrument eficient pentru reducerea timpului care era necesar până acum pentru crearea unui orar.

Funcțiile aplicației minimizează efortul depus în procesul de creare a orarului, atât din perspectiva introducerii datelor necesare, cât și din cea a modificării ulterioare a orarului rezultat, oferind totodată un algoritm eficient de rezolvare a problemei programării evenimentelor.

Pentru a putea folosi generatorul de orar este necesar ca datele referitoare la elevi, profesori și săli de clasă să fie introduse. Interfața este structurată astfel încât acest proces să fie cât mai ușor și accesibil. De asemenea, după generarea orarului este necesar să fie pus la dispoziția utilizatorului o formă care să facliteze distribuirea acestuia printre elevii și profesorii instituției de învățământ. În CodeHour, după stabilirea unui format final al programării evenimentelor acesta poate fi transferat într-un pachet de pagini HTML care la rândul său poate fi publicat pe site-ul instituției, facilitând astfel acest proces.

2. Tehnologii folosite

Pentru obținerea orarului unei școli, problema care se dorește a fi rezolvată presupune programarea unor evenimente respectând cât mai îndeaproape un set de condiții.

*“[Aceste] condiții - constrângeri - reprezintă o modalitate specific umană de a privi o problemă și astfel ajută la abstractizarea acesteia într-o structură ușor de înțeles și de folosit.* ***Programarea bazată pe constrângeri*** *este chiar din acest motiv folosită pentru rezolvarea problemelor de combinatorică, în special a acelora de planificare și organizare.”* [1]

Formatul ales pentru structurarea inputului este cel **XHSTT (XML High School Timetabling)**, acesta fiind dezvoltat întocmai pentru acest tip de problemă – stocarea datelor pentru orarului unei școli - permițând totodată structurarea constrângerilor. Formatul XHSTT a fost creat tocmai din nevoia de a interschimba date ce țin de programarea calendaristică, fiind necesar ca aceste date să aibă un format comun. Astfel, un grup de cercetători a căzut deacord asupra unul standard XML. Cele 4 tipuri de elemente ale acestui format se referă la timp, resurse, evenimente si constrângeri. Primele trei categorii structurează datele necesare problemei, în timp ce logica este redată prin constrângerile impuse.

CodeHour rezolvă problema orarului folosind specificații pentru datele de intrare și de ieșire în formatul XHSTT adaptat indicațiilor. Peste această structură este aplicat un algoritm euristic ce își are rădăcinile în programarea bazată pe constrângeri. Rezultatul este programul activităților împărțit pe zile și ore, ce se poate repeta săptămânal. Structura rezultată este apoi la dispoziția utilizatorului care o poate folosi pentru a extinde orarul sau pentru a-l modifica într-o formă mai convenabilă.

Implementarea aplicației este realizată folosind platforma de dezvoltare **Java 8**. De asemenea, este folosit **JavaFX** - un set de pachete ce permite dezvoltarea un aplicații client complexe ce operează apoi consistent pe multiple tipuri de platforme. Dezvoltat ca un API pentru Java, codul scris în JavaFX poate referenșia API-uri din orice librărie Java. Fiind o aplicație de tip desktop implementată utilizând JavaFX, CodeHour oferă o rulare consistentă pe diverse platforme software.

Pentru stilizarea interfeței este folosit limbajul **JavaFX** **CSS**. Acest limbaj este bazat pe versiunea W3C CSS 2.1 cu câteva adăugiri din versiunea 3. Sunt incluse și câteva extensii pentru CSS pliate pe feature-uri specifice JavaFX. Scopul limbajului este să permită programatorilor familiarizați cu CSS pentru HTML să folosească CSS pentru a structura și a customiza obiecte specifice interfeței JavaFX într-un mod cât mai facil. Fișierele JavaFX CSS pot fi parcurse de orice parser CSS, chiar dacă acesta nu suportă extensiile JavaFX. O caracteristică a structurii codului este faptul că proprietățile specifice acestui limbaj sunt prefixate cu extensia “-fx-”.

Pentru ca orarul generat să poată fi distribuit cu ușurință este implementată funcția de generare raport. Astfel, după ce forma finală a orarului este stabilită, utilizatorul poate crea un pachet de pagini **HTML** ce pot fi apoi ușor distribuite în mediul online. Aceste pagini sunt create cu ajutorul librăriei **Apache FreeMarker** ce a fost folosită pentru a genera fișiere HTML după un template și datele corespunzătoare. Apache FreeMarker este o librărie Java cu care se poate genera output text (pagini web HTML, e-mail-uri, cid sursă, etc.) pe baza unor structuri stabilite anterior și a unor date ce pot fi diferite în funcție de rulare. Template-urile sunt scrise în limbajul FreeMarker Template Language (FTL). Datele trebuiesc formatate într-un limbaj de programare a priori. Template-ul în limbaj FTL se concentrează pe felul în care sunt prezentate datele, conținutul și structurarea acestora realizându-se în afara template-ului.

3. Aplicații similare

Căteva aplicații similare ce se adresează aceluiași tip de problemă – construirea orarului unei școli sau chiar a unei facultăți sau instituții în general, sunt deja dezvoltate și sunt prezentate în cele ce urmează.

aSc Timetables

Aceasta este o aplicație ce se adresează rezolvării aceleiași probleme precum CodeHour. Aspectele comune sunt faptul că aplicația are nevoie de introducerea datelor specifice pentru generarea orarului, permite generarea automată a acestuia și oferă o modelitate de a distribui orarul în mediul online. De asemenea, odată ce orarul a fost generat, acesta poate fi modificat manual în ambele aplicații.

Aplicația aSc Timetables este o aplicație creată pentru a fi customizată în funcție de țara în care va fi folosit, nefiind pliat pe o anumită structură și necesitând configurații atât înainte cât și în timpul folosirii aplicației. Structura acestei aplicații se bazează pe folosirea de conturi de către utilizatori. și este oferită, de asemenea, o versiune web a aplicației. Funcțiile oferite însă de aplicația desktop necesită multe resurse hardware pentru o rulare eficientă din punctul de vedere al timpului.

eOra

eOra este o aplicație ce nu se adresează doar școlilor, ci poate rezolva problema orarului pentru o multitudine de tipuri de instituții în care este necesară planificarea unor activități. Abordarea flexibilă a problematicii orarului permite aplicației să poată fi ușor adaptată oricărei situații.

Crearea orarului poate fi facută asistat – situație în care utilizatorul uman controlează manual modul de plasare a activităților iar sistemul verifică să nu existe suprapuneri și să fie îndeplinite constrângerile impuse – sau automat – pe baza constrângerilor introduse fie ca urmare a formalizării preferințelor profesorilor, fie legate de utilizarea sălilor sau chiar impuse orarului. În cazul aplicației CodeHour crearea orarului poate fi făcută doar automat, urmând apoi ca utilizatorul să poată modifica asistat soluția rezultată, modificările ce nu respectă constrângerile neputând fi efectuate.

Aplicația permite vizualizarea și tipărirea de rapoarte care surprind toate aspectele specifice. De asemenea este posibilă generarea și publicarea pe Web a orarului. Ca soluție pentru ușurarea distribuirii orarului, CodeHour poate genera orarul ales în format HTML ce poate fi publicat pe Web.

Funcțiile oferite de eOra fac această aplicație să fie mai mult decât un program de creare a orarului, eOra este un sistem integrat pentru planificarea activităților și gestiunea resurselor unei instituții. Acest grad ridicat de dezvoltare are însă dezavantajul că necesită resurse hardware destul de puternice iar utiliztorul poate avea nevoie de îndrumare pentru a folosi aplicația la adevarata capacitate.

4. Funcționalități

CodeHour pune la dispoziția utilizatorului suport pentru diferite acțiuni în procesul generării orarului. Astfel, principalele funcții ce sunt oferite prin intermediul aplicației sunt descrise mai jos.

Introducerea datelor de bază

Informațiile despre profesori, elevi, săli de clasă, intervale disponibile, legătura dintre acestea în cadrul orelor, precum și diferitele constrângeri pot fi adăugate folosind interfața ce are un ecran dedicat acestei acțiuni (Ecranul Creare). Prin intermediul acestuia, utilizatorul este asistat în completarea datelor, unele câmpuri necesare algoritmului și structurării input-ului fiind completate automat.

Posibilitatea de a modifica direct fișierele XML de input este disponibilă, însă procesul are un grad de dificultate mai ridicat din cauza faptului că trebuie respectat formatul specific ales pentru această problemă. Interfața vine aici în ajutorul utilizatorului, simplificând desfășurarea acestei activități necesare.

Modificarea datelor unui orar deja existent

Această funcție este adăugată pentru a oferi suport în situația în care se crează orarul pentru aproximativ același set de date, cum ar fi în cazul orarului aceleiași școli pentru semestrul I și pentru semestrul al II-lea. O mare parte din datele de input vor fi asemănătoare (clasele, profesorii, materiile, sălile).

Pentru ca utilizatorul să nu fie obligat să introducă din nou datele deja existente în orarul semestrului I, acesta poate importa datele din acest orar și să le modifice pentru a îndeplini cerințele necesare semestrului al doilea.

Generarea automată a programării evenimentelor

Aplicația folosește un algoritm ce se bazează pe contrângerile impuse asupra datelor pentru a programa în mod cât mai eficient orele. Algoritmul are două posibile utilizări: poate fi folosit pentru a crea un orar nou pentru datele introduse sau pentru a adăuga ore unui orar deja existent.

Ambele variante folosesc setul de date introdus. În cazul în care se dorește doar adăugarea unor evenimente la un orar deja generat, evenimentelor care au fost programate anterior li se păstrează intervalele stabilite, fiind doar adăugate după posibilitățile rămase ultimele ore introduse.

Vizualizarea orarului în funcție de clasă/ profesor/ sală

În ecranul de vizualizare al aplicației, pentru a oferi acces la orar în forma în care va fi acesta vizualizat de către elevi/profesori informația este structurată în formatul specific, utilizatorul putând accesa atât programarea fiecărei resurse (ex.: orarul clasei 10A, orarul Prof. Ana Stan), cât și orarul general, în care sunt reprezentate programările tutror evenimentelor.

Există trei orare generale – unul pentru clase, unul pentru profesori și unul pentru săli, acestea oferind o idee completă despre structurarea orelor în ansamblul lor. De asemenea, aceste trei orare sunt editabile, motiv pentru care ecranul de vizualizare are, în fapt, și funcția de modificare a datelor.

Modificarea manuală a soluției rezultate

După generarea automată a soluției orarului aceasta poate fi modificată de către utilizator. Evenimentele pot fi interschimbate sau li se poate modifica intervalul în care sunt programate, însă numai dacă modificările dorite nu contravin constrângerilor impuse asupra datelor. Utilizatorului nu ii este permisă schimbarea orarului decât în cazul în care constrângerile sunt respectate.

Generarea raportului final

După stabilirea formei finale a orarului, acesta trebuie distribuit celor pe care îi privește. CodeHour poate crea un set de pagini HTML ce pot fi ușor publicate online. Formatul acestor pagini poate fi schimbat după preferințe, fiind necesar respectarea unor reguli de bază care sunt însă foarte customizabile.

5. Structura lucrării

În Cap.1 este prezentată aplicația CodeHour din perspectiva interacțiunii utilizatorului cu aceasta. Este prezentat întregul proces prin care trece utilizatorul atunci cînd dorește realizarea unui orar. În acest capitol sunt detaliate cele trei componente principale ale acestei interacțiuni: fișierele XML, interfața și raportul.

Completarea fișierelor XML în formatul specific ales reprezintă un pas necesar, acestea conținând datele de input ale problemei. Interfața, structurată în cele trei ecrane, este intuitivă și separă funcționalitățile aplicației în cele două mari componente: introducerea datelor și modificarea orarului efectiv. Raportul în format HTML oferă posibilitatea de a publica pe Web orarul, distribuind-ul astfel tutror celor interesați.

Al doilea capitol prezintă aplicația din punctul de vedere al developer-ului. Este prezentată arhitectura programului, cu legăturille dntr pachete și clase. Apoi sunt detaliate elementele aplicației ce au necesitat un grad de atenție mai ridicat în timpul dezvoltării aplicației, având un mare impact asupra comportamentului acesteia. Structura necesară citirii și scrierii din fisierele XML, precum și algoritmul implementat, a cărui funcționalitate depinde în mare măsură de implementarea constrângerilor problemei fac parte din capitolul al doilea.

Posibilele direcții de dezvoltare ale aplicației sunt descrise pe scurt apoi în al treilea capitol, incluzând descrieri ale paradigmelor ce pot fi implementate pentru rezolvarea problemei orarului folosind structura de date deja adăugată, precum și modlitatea cea mai potrivită de deployment a aplicației.

O schemă cuprinzătoare a utilizării aplicației este prezentată în Cap.4, scenariile de utilizare fiind cuprinse în aceasta. Sunt descrise opțiunile oferite utilizatorului, precum și comportamentul aplicației în urma selectării acestora, fiind oferită o explicație schematică a declanșării proceselor pe parcursul utilizării aplicației.

Contribuții

Funcționalitatea în jurul căreia a fost creată întreaga structură a aplicației este oferirea pentru utilizator a unui algoritm eficient ce rezolvă problema orarului, aceasta fiind una dintre contribuțiile principale aduse lucrării, împreună cu modalitatea de implementare a logicii constrângerilor, pe care se bazează rata de succes a algoritmului.

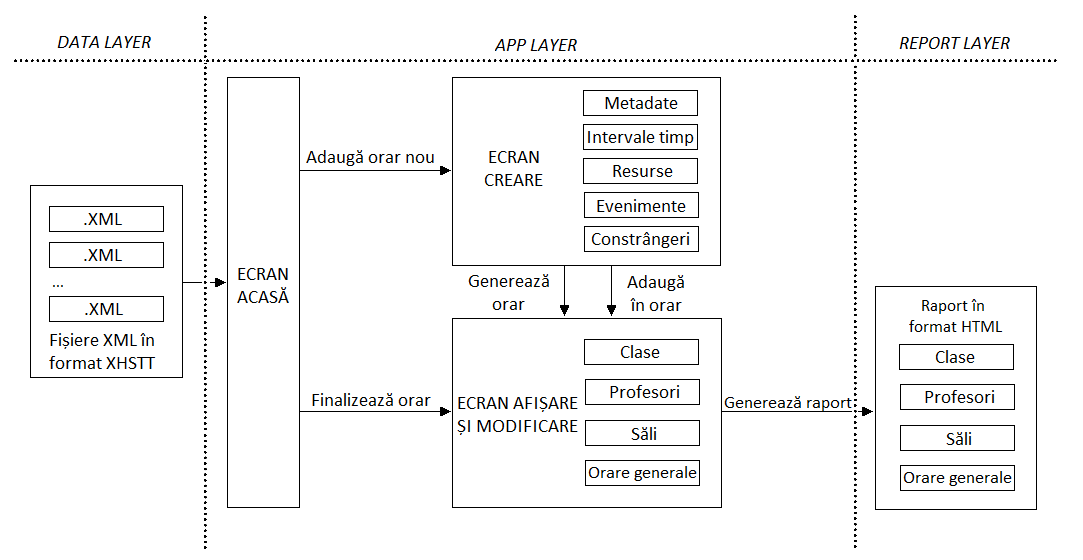
Ușurința cu care decurge procesul generării unui orar poate fi mult influențată de structura interfeței folosite, de cât de intuitivă este aceasta. În acest sens, una dintre contribuțiile aduse aplicației CodeHour este structurarea și implementarea unei interfețe ușor de folosit ce ghidează utilizatorul pe parcusul întregului proces de utilizare, automatizând anumite funcții acolo unde este posibil.

În capitolele următoare este prezentată aplicația din pucntul de vedere al structurii, precum și din cel al implementării componentelor principale ce au un impact puternic asupra formei codului și a interfeței.

Cap.1. Structura aplicației

* 1. Arhitectura aplicației

Interacțiunea utilizatorului cu aplicația poate fi împărțită în trei nivele. Aceastea sunt descrise în *Fig.1*. Primul – *Data Layer* – este cel în care este definit inputul problemei și este reprezentat de fișierul/fișierele XML. Următorul nivel este *App Layer* reprezentând interacțiunea cu interfața aplicației și ultimul este *Report Layer* – varianta finală a orarului într-un format ce permite ușor distribuirea informației.



*Fig.1. Schemă a arhitecturii aplicației*

Inputul problemei este definit prin intermediul fișierului/fișierelor XML. Utilizatorul are la dispoziție două metode de a crea și modifica un fișier: fie prin intermediul interfeței accesând secțiunea pentru crearea orarului, fie direct folosind un editor (ex.: Visual Studio Code, Atom, Notepad++). Este necesar ca aceste fișiere să respecte formatul XHSTT specificat în detaliu în secțiunea *1.2 Proiectarea XML-urilor*. În cazul utilizării interfeței însă, problema respectării formatului dispare, utilizatorul fiind asistat în completarea datelor, acestea fiind apoi scrise automat în XML după formatul necesar.

Interfața este formată din trei ecrane principale:

* Ecranul acasă – de aici utilizatorul poate alege să vadă și să modifice un orar ce are deja soluție, intrând în ecranul de afișare, sau poate alege să adauge date unui orar nefinalizat ori să creeze un orar complet nou, intrând în ecranul de creare.
* Ecranul de creare – unde utilizatorul este asistat în adăugarea datelor de input ale problemei. Se poate crea un orar nou sau se poate porni de la datele unui orar deja existent care urmează a fi modificate. După completarea datelor butoanele Generează orar și Adaugă la orar permit rularea algoritmului, iar la finalizarea acestuia soluția este afișată deschizându-se ecranul de afișare și modificare.
* Ecranul de afișare și modificare – unde pot fi vizualizate si modificate soluțiile deja existente pentru fișierele XML cu datele problemei. Când orarul este în forma finală butonul Generează raport creează un pachet de pagini HTML de prezentare a orarului ce permit distribuirea acestuia online.

Funcționalitatea fiecărui ecran este descrisă în detaliu în secțiunea *1.3 Interfața*.

Pentru a ușura distribuirea orarului generat, utilizatorul are opțiunea de a genera un raport format dintr-un pachet de pagini HTML ce poate fi ușor publicat pe site-ul instituției. Formatul raportului poate fi modificat în funcție de preferințe, acesta fiind generat cu ajutorul librăriei Apache FreeMarker pe baza unor template-uri. Utilizatorul poate adăuga date sau schimba formatul orarului prin modificarea fișierelor .ftl din folderul templates. Mai multe detalii despre formatul raportului se pot găsi în secțiunea *1.4 Raportul.*

* 1. Proiectarea XML-urilor

Fiecărui orar generat îi corespunde un fișier în care sunt stocate și datele de la care s-a pornit în generarea orarului. Aceasta este posibilă datorită faptului că structura XHSTT aleasă permite stocarea în același fișier XML atât a datelor de input ale problemei, cât și a celor de output.

Fisierele XML pot fi create atât prin intermediul interfeței, cât și prin editare directă, utilizatorul având acces la ele în folderul CodeHourXMLs. În cazul în care acestea sunt editate direct, fișierul rezultat trebuie să respecte întru totul structura redată mai jos. Structura XHSTT corespunzătoare, pe care este necesar să o respecte fișierul peste care va fi rulat algoritmul, este descris în *Fig.2*.

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" standalone="yes"?>  <timetable id="Id orar">  <metadata>  ...  </metadata>  <basicTimeUnitInMinutes>  ...  </basicTimeUnitInMinutes>  <times>  ...  </times>  <resources>  ....  </resources>  <events>  ....  </events>  <eventConstraints>  ...  </eventConstraints>  <resourceConstraints>  ...  </resourceConstraints>  <solutions>  ...  </solutions>  </timetable> |

*Fig.2. Formatul de bază XHSTT al fișierelor*

Inputul problemei este reprezentat de:

* metadate (ex.: data creării orarului, durata unui curs, autor orar)
* intervalele de timp în care se pot programa evenimentele (ex.: luni în intervalul 8 - 8:50),
* resursele, reprezentând participanții la evenimente și se împart în 3 categorii:
* clase (ex.: clasa a IX-a A),
* profesori (ex.: Popescu Mariana),
* săli de clasă (ex.: sala 202),
* evenimentele neprogramate, constând în orele de activitate (ex.: o ora de geografie),
* constrângeri pentru evenimente (ex.: să nu fie programate mai mult de două ore de matematică consecutiv),
* constrângeri pentru resurse (ex.: sala 202 este ocupată marți în intervalul 10:00 - 14:00).

Outputul algoritmului de soluționare a problemei orarului sunt:

* soluțiile – o listă ce conține toate variantele de programări rezultate din rularea algoritmului.

În cele ce urmează este prezentată fiecare etichetă de mai sus în detaliu.

Metadatele

Acestea sunt datele despre crearea orarului curent, cuprinzând:

* id – atributul etichetei <timetable> reprezentând id-ul unic al orarului curent
* <contributor> – autorul,
* <date> – data creării orarului,
* <description> – câteva detalii despre orarul curent,
* <name> – numele orarului.

Am inclus în categoria metadatelor și eticheta <basicTimeUnitInMinutes> care specifică durata unei unități de timp de bază. Altfel spus, această etichetă definește durata minimă a unui curs în minute. Această durată va fi de folos atunci când un eveniment (de ex.: curs, prezentare) are alocat un interval mai mare de timp decât cel normal. De exemplu, dacă o oră durează 60 de minute vom seta <basicTimeUnitInMinutes> ca fiind 60. Dacă mai apoi avem de programat un eveniment ce necesită două ore vom seta numărul de intervale de timp necesare evenimentului ca fiind 2 (2 \* 60 min).

Un exemplu de completare a metadatelor și a câmpului <basicTimeUnitInMinutes> este descris în *Fig.3*.

|  |
| --- |
| <metadata>  <contributor>Autor orar</contributor>  <date>2017-03-26T17:31:50.798+03:00</date>  <description>Descriere orar</description>  <name>Nume orar</name>  </metadata>  <basicTimeUnitInMinutes>50</basicTimeUnitInMinutes> |

*Fig.3. Exemplu metadate în format XHSTT*

Intervalele de timp

Eticheta <times> conține toate intervalele în care se poate face programarea evenimentelor. Pentru un singur interval de timp (<time>) sunt necesare următoarele câmpuri:

* <day> – ziua intervalului,
* <hourInteval> – intervalul orar formatat după cum se dorește să apară în orar,
* <name> – un id care în cazul completării datelor cu ajutorul interfeței este generat automat. Id-ul are forma *number\_day*, unde *day* este ziua de care aparține intervalul, iar *number* este numărul de ordine al intervalului curent în acea zi.
* <numberOfTimeUnits> – durata intervalului în unități de timp de bază definite în categoria metadatelor,
* id – un atribut al etichetei <time> care specifică numărul de ordine al intervalului curent în întreaga listă de intervale.

În *Fig.4*  este prezentat un exemplu pentru completarea câmpurilor descrise mai sus.

|  |
| --- |
| <times>  <time id="1">  <day>monday</day>  <hourInterval>8:00-8:50</hourInterval>  <name>1\_mon</name>  <numberOfTimeUnits>1</numberOfTimeUnits>  </time>  <time id="2">  <day>monday</day>  <hourInterval>9:00-9:50</hourInterval>  <name>2\_mon</name>  <numberOfTimeUnits>1</numberOfTimeUnits>  </time>  <time id="3">  <day>tuesday</day>  <hourInterval>8:00-8:50</hourInterval>  <name>1\_tue</name>  <numberOfTimeUnits>1</numberOfTimeUnits>  </time>  <time id="4">  <day>tuesday</day>  <hourInterval>9:00-9:50</hourInterval>  <name>2\_tue</name>  <numberOfTimeUnits>1</numberOfTimeUnits>  </time>  </times> |

*Fig.4. Exemplu intervale de timp XHSTT*

Resursele

Tag-ul <resources> reprezintă lista de resurse disponibile ale școlii. Resursele sunt reprezentate de profesori, clase și săli. Fiecare resursă (<resource>) are următoarele detalii:

* id – id-ul unic al resursei respective. În cazul claselor, acesta poate fi chiar numele clasei (ex.: 9A, 9B, 10A), în cazul săților, acesta poate fi numele prin care este identificată sala (ex.: 203), iar în cazul profesorilor este preferabil ca acest id să fie identificatorul lor unic ales în funcție de preferințe (ex.: dpop pentru Prof. Dan Popovici).
* <name> – numele resursei. În cazul profesorilor se menționează numele întreg, acesta fiind numele ce va apărea în orar. În cazul sălilor și a grupurilor de studiu acest atribut poate fi lasat necompletat, nefiind absolut necesar. Pentru toate resursele ce nu au numele completat, la generarea orarului va fi afișat id-ul resursei respective.
* <resourceType> – atribut ce trebuie sa aibă una din următoarele trei posibile valori: studyGroup, teacher, classroom.

Un exemplu de completare pentru fiecare tip de resursă este prezentat în *Fig.5* de mai jos. În cazul resursei 9A de tip studyGroup nu a fost completat câmpul <name>, acesta nefiind unul obligatoriu. Pentru o mai ușoară înțelegere a datelor, în câmpul <name> al sălilor poate fi trecut tipul acestora (ex.: laborator de biologie, laboraor de fizică, cabinet franceză, etc.) așa cum este și în exemplul de mai jos.

|  |
| --- |
| <resources>  <resource id="9A">  <resourceType>studyGroup</resourceType>  </resource>  <resource id="id\_prof">  <name>Nume Profesor</name>  <resourceType>teacher</resourceType>  </resource>  <resource id="101">  <name>Laborator biologie</name>  <resourceType>classroom</resourceType>  </resource>  </resources> |

*Fig.5. Exemplu resurse în format XHSTT*

Evenimente

Fiecare eveniment reprezentat prin tag-ul <event> are asociate propriile date în felul următor:

* id – identificatorul unic al evenimentului. Acest id este generat automat dacă datele sunt introduse din interfață. Formatul său este *ev\_number*, unde *number* este numărul evenimentului curent în cadrul listei de evenimente.
* <description> – numele evenimentului, ce va apărea în orarul final, fiind de preferat să reprezinte numele materiei respective (ex.: Biologie).
* <resources> – listă de resurse, fiecare având structura descrisă în tabelul Exemplu resurse în format XHSTT de la pagina 13.

*Fig.6* prezintă un model de completare a câmpurilor unui eveniment în formatul XHSTT.

|  |
| --- |
| <events>  <event id="ev\_1">  <description>Biologie</description>  <resources>  <resource id="9A">  <resourceType>studyGroup</resourceType>  </resource>  <resource id="id\_prof">  <name>Nume Profesor</name>  <resourceType>teacher</resourceType>  </resource>  <resource id="101">  <name>Laborator biologie</name>  <resourceType>classroom</resourceType>  </resource>  </resources>  </event>  </events> |

*Fig.6. Exemplu evenimente în format XHSTT*

Constrângeri impuse evenimentelor

Aceste constrângeri se aplică unuia sau mai multor evenimente. Formatul tag-ului <eventConstraints> este următorul:

* <assignResourceConstraint id="assignResourceConstraint1"> – acest tag este reprezintat de numele constrângerii și poate avea una dintre următoarele valori:
  + <assignResourceConstraint>,
  + <assignTimeConstraint>,
  + <limitRepeatActivityConstraint>,
  + <limitRepeatDailyConstraint>,

iar id-ul este de forma *numeConstrângereNumăr*, unde *Număr* este numărul de ordine al constrângerii curente,

* <appliesToEvents> – lista de evenimente pentru care este valabilă constrângerea curentă,
* <required> – care poate avea doua valori: true sau false, prima însemnând că este necesar ca această constrângere să fie respectată, iar a doua însemnând că algoritmul poate sugera soluții care încalcă această constrângere, deși este de preferat ca toate constrângerile să fie respectate,
* <weight> - cu cât este mai mare acest parametru cu atât mai mult se va încerca respectarea constrângerii în găsirea unei soluții pentru orar.

Constrângerile de tip limitRepeatActivityConstraint au în plus un câmp numit maximumNumberOfRepetitions, care reprezintă numărul de ore ce pot apărea consecutiv, în plus față de prima programată. Același câmp apare în plus și la constrângerile din categoria limitRepeatDailyConstraint, însă aici el reprezintă numărul de ore ce pot apărea în plus în aceeași zi față de prima programată.

Modelul XHSTT din *Fig.7* este un exemplu de completare a unei constrângeri de tip assignResourceConstraint și a uneia de tip assignTimeConstraint.

|  |
| --- |
| <eventConstraints>  <assignResourceConstraint id="assignResourceConstraint1">  <appliesToEvents>  <event id="ev\_1">  ...  </event>  </appliesToEvents>  <required>true</required>  <weight>1</weight>  </assignResourceConstraint>  <assignTimeConstraint id="assignTimeConstraint1">  <appliesToEvents>  <event id="ev\_1">  ...  </event>  </appliesToEvents>  <required>true</required>  <weight>1</weight>  </assignTimeConstraint>  </eventConstraints> |

*Fig.7. Exemplu contrângeri evenimente XHSTT*

Constrângeri impuse resurselor

Acest tip de constrângeri se aplică resurselor. Tag-urile <required> și <weight> au aceleași roluri ca și în cazul contrângerilor pentru evenimente. Celelalte etichete sunt:

* <limitIdleTimeConstraint id="limitIdleTimeConstraint\_1"> – atât tag-ul cât și id-ul sunt definite la fel ca în cazul tagului cu numele pentru constrângeri de evenimente. Aici însă eticheta poate avea forma
  + <limitIdleTimeConstraint>,
  + <limitResourceAvailableTimesConstraint>.
* <appliesToResources> – lista de resurse pentru care este valabilă constrângerea,

În cazul contrângerii de tip <limitIdleTimeConstraint> vom avea un parametru în plus: <maximumIdleHours>. Acesta este numărul de intervale de bază (definit în categoria metadatelor – *basicTimeUnit*) ce pot fi lăsate neprogarmate pentru o resursă, între intervalele programate, în cadrul unei zile. De exemplu, dacă dorim să poată fi lăsate ferestre de maxim două ore între orele unui profesor în cadrul aceleiași zile, iar <basicTimeUnitInMinutes> are valoarea unei ore, atunci <maximumIdleHours> va avea valoarea 2.

În *Fig.8* este exemplificată completarea câmpurilor pentru o constrângere de tipul limitIdleTimeConstrain.

|  |
| --- |
| <resourceConstraints>  <limitIdleTimeConstraint id="limitIdleTimeConstraint1">  <appliesToResources>  <resource id="9A">  ...  </resource>  </appliesToResources>  <maximumIdleHours>1</maximumIdleHours>  <required>true</required>  <weight>1</weight>  </limitIdleTimeConstraint>  </resourceConstraints> |

*Fig.8. Exemplu contrângeri resurse XHSTT*

Soluții

Algoritmul implementat în varianta curentă a aplicației are ca output o singură soluție. Arhitectura permite însă adăugarea mai multor tipuri de algoritmi. Așadar, formatul XHSTT ales permite ca pentru un set de date de input să poată fi stocate în același fișier mai multe variante pentru orar, fiecare fiind o soluție a problemei. Eticheta <solutions> conține fiecare dintre aceste soluții individuale stocate prin tag-ul <solution> care are componentele:

* <description> – nume al soluției curente,
* <events> – listă de evenimente programate, adică fiecare eveniment are asociat un element <time> reprezentând unul din intervalele introduse la început sub eticheta <times>.

Un model de soluție se regăsețte în *Fig.9.*

|  |
| --- |
| <solutions>  <solution id="sol1">  <description>First Solution</description>  <events>  <event id="ev\_1">  <description>Biologie</description>  <resources>  ...  </resources>  <time id="1">  ...  </time>  </event>  </events>  </solution>  </solutions> |

*Fig.9. Exemplu soluții în format XHSTT*

* 1. Interfața

Funcționalitatea aplicației implementată în interfață poate fi împărțită în două componente: adăugarea de date (input) și modificarea soluției (output). Utilizatorul interacționează cu trei ecrane principale, fiecare dintre acestea fiind descris în cele ce urmează.

Ecranul acasă

Ecranul principal oferă posibilitatea vizualizării tuturor fișierelor create cu ajutorul aplicației într-o listă, precum și crearea unei structuri noi. Astfel, utilizatorul poate vedea soluțiile deja finalizate, poate continua lucrul la fișierele care nu au încă o soluție sau poate crea orare noi.

Componentele acestui ecran sunt:

* Lista de cu orarele disponibile – la selectarea celor care au deja soluție aceasta va fi afișată în ecranul de display, iar pentru cele care au eticheta *nefinalizat* se va deschide ecranul de creare completat cu datele deja existente pentru a putea fi completate.
* Butonul pentru crearea unui orar nou – acesta deschide ecranul de creare fără date completate.

Ecranul creare

Aici se poate crea un orar complet nou sau se poate porni de la un orar deja existent ce urmează a fi modificat. Componentele acestui ecran sunt bara de meniu și opțiunile de adăugare din stânga ecranului ce deschid fiecare o pagina diferită în partea dreaptă, precum și butoanele pentru generarea orarului pe baza datelor introduse.

Bara de meniu conține:

* Buton *CodeHour* (Acasă)
* Buton *Alege* – permite alegerea unui orar dintre cele deja existente.
* Buton *Orar nou* – șterge datele completate până la momentul curent pentru a putea crea un orar nou.

Opțiunile de adăugare au ca element comun butonul *Salvează* care înregistrează datele. Dacă acestea sunt incorecte sau incomplete, câmpurile care nu corespund sunt evidențiate. În cazul în care cerințele minime sunt respectate, butonul *Salvează* ori va deschide o fereastră de creare a fișierului în care vor fi salvate datele dacă un astfel de fișier nu este deja setat sau va salva datele în filierul deja ales.

Un alt elemnt comun al opțiunilor este tabelul cu datele ce sunt deja înregistrate. Pentru a șterge o înregistrare se folosește butonul roșu *Delete* prezent la finalul fiecărui rând. Pentru a modifica o înregistrare se selectează rândul corespunzător, se modifică doar câmpurile dorite și se salvează. Modificările apar apoi în tabelul de date.

Opțiunile pentru adăugarea datelor sunt:

* *Detalii orar* *(Metadate)* - se adaugă în ecranul din dreapta *Id*-ul orarului, *Numele*, *Autorul*, *Durata unei unități de timp* (durata unui curs), acestea fiind elementele necesare pentru crearea orarului; în plus se pot adăuga câteva cuvinte în plus despre orar în categoria *Descriere*.
* *Intervale orare* - pentru un interval se alege *Ziua*, se adaugă *Intervalul orar* în formatul dorit la afișarea orarului final, precum și durata intervalului curent în *Unități de timp* de bază a căror lungime în minute este setată la *Metadate*. Pe măsură ce se adaugă intervale de timp acestea vor fi afișate în tabelul din partea de jos a paginii din care pot fi apoi modificate sau șterse. Tabelul arată și id-ul și numele generat automat pentru fiecare interval de timp astfel încât să fie respectate cerințele formatului XHSTT dorit.
* *Resurse (Clase, Profesori, Săli de clasă)* - câmpurile ce se completează sunt *Id* și *Nume*; pentru toate cele trei categorii de resurse este necesar *Id*-ul, însă *Numele* trebuie completat doar în cazul profesorilor. În partea de jos a fiecărei dintre cele trei pagini va apărea tabelul cu datele compleatte până la momentul curent.
* *Evenimente* - pentru câmpul *Materie* opțiunile disponibile sunt afișate într-o listă editabilă și sunt preluate din fișierul Subjects.txt din folderul UserFiles. Acest fișier conține materiile de bază din programă însă poate fi editat după nevoile utilizatorului. *Număr evenimente* este numărul de evenimente cu aceleași specificații care trebuie adăugate (ex.: câmpul va avea valoarea 5 pentru 5 ore de matematică pentru clasa 9A cu profesorul Dan Pop în sala 202). *Lista de resurse* este o listă din care se pot alege clasele, profesorii și sălile necesare pentru un curs (eveniment). Pe măsură ce sunt adăugate resurse acestea apar ca fiind selectate, putând fi deselectate prin butonul roșu de delete atașat fiecăreia. În tabelul cu evenimentele exitente este afișat și id-ul generat automat pentru fiecare eveniment.
* *Constrângeri* – câmpul *Greutate* setează gradul în care contrângerea trebuie respectată comparativ cu celelalte constrîngeri în cazul unor incompatibilități. *Constrângere necesară* solocită algoritmul să gasescă exclusiv soluții care să respecte constrângerea marcată astfel. *Lista de evenimente* funcționează asemănător cu *Lista de resurse* din panoul *Evenimente*,permițand alegerea evenimentelor pentru care etse valabilă contrângerea dintr-o listă cu cele deja existente*.* Constrângerile sunt apoi afișate în tabelul din partea de jos a paginii.

Butoanele din stanga jos permit rularea algoritmului peste datele adăugate. La apăsarea oricăruia dintrce cele două butoane, dacă datele nu sunt comlete sau sunt incorecte acest lucru va fi semnalat cu un mesaj de atenționare afișat în partea de jos a ecranului din stânga. Cele doua butoane au următoarele funcționalități:

* *Generează orar* – rulează algoritmul de creare a soluției peste datele introduse.
* *Adaugă în orar* – pornește de la soluția selectată deja creată și lasă evenimentele deja programate neschimbate, doar adăugând orarului programarea evenimentelor adăugate nou de la intrarea în ecranul *Creare*.

Ecranul afișare și modificare

Aici poate fi vizualizat orarul în forma în care va fi distribuit ulterior. Structurat asemănător cu *Ecranul de creare*, având aceleași structuri de componente: bara de meniu, opțiunile din stânga ecranului ce deschid fiecare o pagina diferită în partea dreaptă, precum și butoanul pentru generarea raportului.

Componentele barei de meniu:

* Buton *CodeHour* (Acasă),
* Buton *Alege orar* – permite alegerea unui orar dintre cele deja existente care au soluția generată.

În stânga ecranului se găsesc câteva opțiuni, dintre care prima permite alegerea soluției dorite pentru orarul curent. Restul opțiunilor au rolul de a afișa un tabel cu orarul în funcție de ce este selectat. Astfel, sunt disponibile:

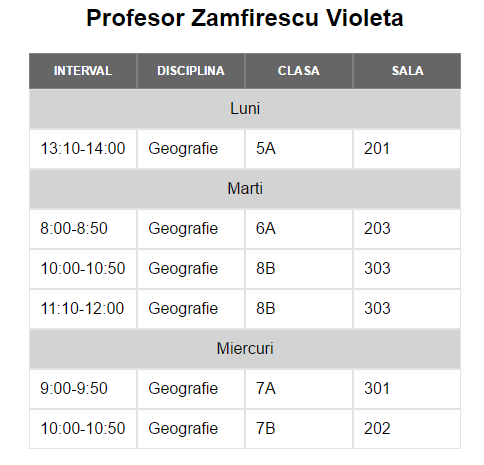
* *Soluție* – permite selectarea unei soluții pentru orarul curent, fiind posibil ca un orar să aibă mai multe soluții diferite.
* *Clase, Profesori, Săli de clasă* – prin selectarea uneia dintre inregistrările disponibile, fie ea clasă, profesor sau sală, se afișează în partea din dreapta a ecranului un tabel needitabil cu orarul resursei selectate.
* *Orare generale* – trei orare, unul ce conține orele tuturor claselor, unul cu orarul din punctul de vedere al profesorilor și unul pentru săli de clasă. Aceste trei tabele sunt editabile. Astfel, locul a două evenimente în tabel poate fi interschimbat. De asemenea, un eveniment poate fi programat intr-o locație care este liberă. Două evenimente nu pot fi programate în același timp. Dacă sunt contradicții pentru modificările dorite acest lucru va fi semnalat printr-un mesaj de atenționare afișat deasupra tabelei.

Butonul *Generează raport* din colțul din stânga jos al ecranului permite crearea unui folder cu pagini HTML aflat la path-ul UserFiles/Reports. Acestea pot fi ușor publicate apoi pe site-ul școlii. Formatul este editabil, iar în cazul în care utilizatorul dorește, poate interveni în fișierele .ftl din folderul templates.

* 1. Raportul

Raportul este reprezentat de un set de pagini HTML ce poate fi găsit la path-ul UserFiles/Reports într-un folder cu id-ul orarului curent și al soluției dorite.

Formatul de bază al raportului este constituit dintr-o pagină principală mainPage.html, având o listă cu toate clasele, toți profesorii și toate sălile, împreună cu link-uri spre paginile care conțin orarul individual pentru fiecare resursă. Un exemplu de tabel pentru orarul unui profesor este prezent în *Fig.10. Exemplu raport profesor*. De asemenea sunt disponibile link-uri spre paginile cu orarele generale pentru clase, profesori și săli de clasă.



*Fig.10. Exemplu raport profesor*

Raportul este generat folosind Apache FreeMarker și template-urile din folderul templates. Modificarea formatului raportului presupune modificarea fișierelor din acest folder, urmărind a se păstra numele etichetelor datelor transmise.

Cap.2. Implementarea

2.1 Citirea din și scrierea în XML

Tehnologia folosită pentru citirea din și scrierea în XML-uri este API-ul JAXB (Java Architecture for XML Binding). Acesta simplifică accesul la fișierele XML prezentându-le conținutul în formatul Java. Structura XML-ului va avea corespondent un set de clase Java care să o reprezinte. Obiectele pot fi apoi adăugate sau încărcate din fișier folosind comenzile marshal și unmarshal.

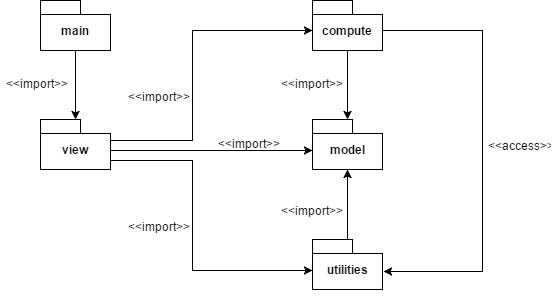
Structura XML-urilor este dată de formatul XHSTT descris în subcapitolul *1.2 Proiectarea XML-urilor*. Pentru a particulariza clasele astfel încât aceastea să poată citi documentele sunt folosite adnotări specifice JAXB. Exemple de astfel de adnotări sunt:

* @XmlRootElement(name = "timetable") – elementul de bază al fișierului XML,
* @XmlElement(name = "events") – exemplu de adnotare folosită pentru toate elementele ce apar în structura XML-ului; atrubutul name ia valori diferite pentru fiecare element,
* @XmlAttribute – ce este adăugat proprietăților care trebuie să apară ca atibut pentru elementele XML; de exemplu, câmpul *id* penru clasa *Resource*,
* @XmlTransient – folosit pentru clasele sau câmpurile ce nu trebuiesc incluse în fișiere; un exemplu este clasa abstractă *Constraint* al cărei scop este doar simplificarea creării claselor pentru constrângeri, conținând toate metodele comune ale acestora, precum și metoda abstractă validate ce trebuie implementată de fiecare constrângere în parte.

Structura claselor este influențată de folosirea tehnologiei JAXB, pachetul *model* fiind creat special pentru aceasta și reprezentând clasele de bază pentru datele cu care lucrează întreaga aplicație. În subcapitolul următor este descrisă intocmai această organizare a claselor aplicației, conținând detalii și despre pachetul *model* precizat anterior.

2.2 Organizarea claselor

Clasele sunt structurate, după funcționalitate, în cinci pachete principale: *main*, *view*, *compute*, *model* și *utilities*. Schema relațiilor dintre pachete este indicată în *Fig.11. Diagrama de pachete*.



*Fig.11. Diagrama de pachete*

Pachetul main

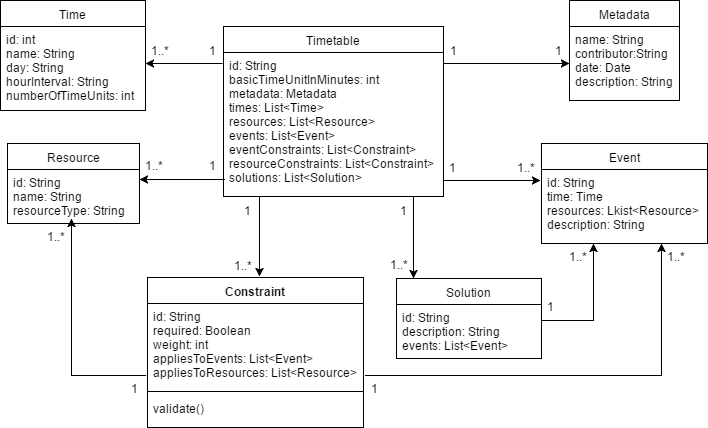
Conține clasa *Main* – prima care este apelată la rularea programului. Această clasă extinde *Application*, implementează funcția *start()* și apelează *lauch()* pentru rularea aplicației JavafX.

Pachetul model

Clasele din acest pachet structurează datele de bază cu care lucrează aplicația. Atât proprietățile fiecărei clase, cât și relațiile dintre ele sunt reprezentate în *Fig.11. Diagrama de clase – pachetul model*.

Acest pachet se concentrează pe crearea structurii clasei principale *Timetable*, fiecare clasă structurând elementele principale necesare creării unui orar astfel:

* *Timetable* – clasa ce înglobează întreaga structură a pachetului, conținând elementele necesare creării orarului, precum și structura în care vor fi stocate soluțiile acestuia;

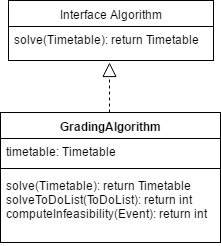
*Fig.12. Diagrama de clase - pachetul model*

* *Metadata* – datele precum *date* (data creări orarului), *contributor* (autorul) ajută la identificarea orarului curent, oferind pobilitatea de a înregistra date esențiale despre orar;
* *Time* ­– reprezentând structura unui interval de timp (ex: marți, 8:00 - 8:50); id-urile intervalelor vor fi numere întregi începând de la 1 și semnificând ordinea intervalelor pe parcursul unei săptămâni;
* *Resource* – structurată pentru a stoca resurse precum clase, profesori sau săli de clasă; numele este cel ce va apărea în orarul final pentru pagina de orar a fiecărei resurse dacă aceasta este de tip *teacher*, iar altfel va fi afișat *id*-ulș
* *Event* – reprezintă o oră sau orice alt tip de eveniment ce trebuie programat; inițial proprietatea *time* nu este setată, fiind completată la rulatea algoritmului cu intervalul de timp în care este programat evenimentul; lista conține toate resursele necesare desfașurării evenimentului, cum ar fi profesor, clasă de elevi, sală de clasă; în *description* este stocat numele evenimentului asa cum va fi trecut în orarul final (ex: Istorie);
* *Constraint* – este o clasă abstractă ce este implementată apoi de fiecare clasă ce reprezintă un tip specific de constrângere; metoda *validate(Event)* este abstractă și trebuie implementată de toate clasele care o extind pe aceasta, reprezentând metoda care returnează costul unui eveniment dacă acesta nu respectă constrângerea; fiecare constrângere are o modalitate proprie de a calcula acest cost, unele având nevoie de câteva elemente suplimentare pentru aceasta;
* *Solution*  – reprezintă obiectul în care este stocată o soluție și conține, în esență, lista de evenimente programate (cu proprietatea *time* setată); *id*  este de ajutor pentru diferențierea soluțiilor dacă există mai multe, în câmpul *description* putând fi reținute mai multe detalii despre soluția curentă.

Pachetul *model* este singurul referențiat de majoritatea celorlalte pachete, însă care nu depinde de nici un altul, scopul său fiind acela de a structura datele problemei de rezolvat.

Pachetul compute

Pachetul *compute* conține interfața *Algorithm* ce are metoda *solve(Timetable)*. Această interfață permite adăugarea mai multor implementări de algoritmi ce rezolvă problema programării evenimentelor. Condiția este ca orice clasă adăugată în acest scop să implementeze *Algorithm* și metoda *solve(Timetable)*. Mai jos este diagrama de clase pentru pachetul *compute*.



*Fig.13. Diagrama de clase - pachetul compute*

Clasa *GradingAlgorithm* reprezintă algoritmul recursiv de rezolvare descris în secțiunea *2.3.Algoritmul*. Cele trei metode au următoarele roluri:

* *solve(Timetable)* – realizează inițializările necesare rezolvării problemei și apelează metoda *solveToDoList(ToDoList)*;
* *solveToDOList(ToDoList)* – metodă recursivă ce primește ca parametru lista tuturor evenimentelor de programat și se oprește atunci când nu mai există nici un astfel de eveniment; metoda salvează soluția pe măsură ce aceasta este generată în câmpul *solution* din obiectul *timetable* al clasei;

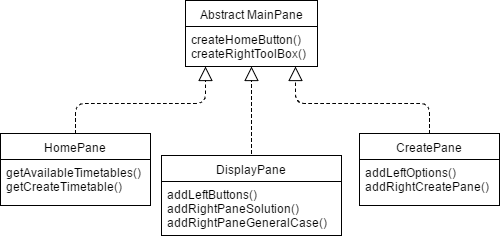
Structura de date din pachetul *model* este referențiată din pachetul *compute*. De asemenea, pentru rezolvarea anumitor sarcini, pachetul *compute* apelează la funcții implementate în pachetul *utilities*.

Pachetul view

Acest pachet este responsabil cu afișarea interfeței grafice. *Fig.14. Diagrama de clase – pachetul view*  reprezintă clasele pentru fiecare ecran ce sunt conținute în pachetul *view.panes*. Clasa abstractă *MainPane* pune la dispoziția claselor ce o extind metodele pentru crearea butonului Acasă și a meniului ce apar în partea de sus a ecranului. Ea este extinsă de clasele:

* *HomePane* – cu cele două metode principale – *getAvailableTimetables()* pentru afișarea listei de orare disponibile și *getCreateTimetable()* pentru afișarea butonului de creare a unui orar nou;
* *DisplayPane* – pentru ecranul de afișare a orarului, având metodele principale pentru adăugarea opțiunilor prezente în stânga ecranului, și metodele ce sunt pelate la selectarea acetor opțiuni, pentru afișarea tabelelor corespunzătoare în dreapta ecranului;
* *CreatePane* – ce reprezintă ecranul de creare a orarului, funcțiile principale ale acesteia fiind aceea de adăugate a opțiunilor din stânga ecranului și de afișare a câmpurilor corespunzătoare fiecărei opțiuni atunci când aceasta este selectată. Pentru aceasta, logica fiecărui ecran de completare din dreapta este structurată în clase diferite regăsite în pachetul *view.create*.

În pachetul *view.styles* se regăsesc și fișierele CSS folosite pentru stilizarea interfeței grafice. Acest pachet folosește atât clase din *model*, cât și din *compute* – atunci când este apăsat butonul de generare a orarului– și din *utilities*.

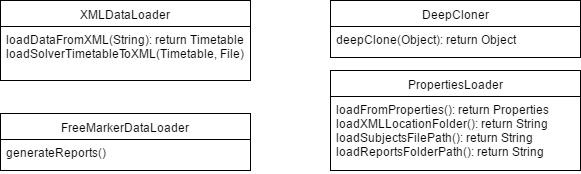


*Fig.14. Diagrama de clase - pachet view*

Pachetul utilities

Clasele conținute aici au rolul de a oferi utilități de bază, fără a relaționa între ele și folosind doar elemente din *model* pentru a-și structura funcționalitatea. Fiecare clasă oferă servicii în modul următor:

* *XMLDataLoader* – folosește funcțiile specifice JAXB (marshal și unmarshal) pentru a scrie date în XML și pentru a le citi;
* *PropertiesLoader* – aplicația are înregistrate câteva date - precum path-urile folderului cu fișierele XML sau cel al fișierului ce conține lista materiilor sau cel al folderului unde sunt salvate rapoartele - într-un fișier de proprietăți; pentru accesarea acestora este implementată clasa *PropertiesLoader* ce are funcții specifice ce pot fi apelate pentru citirea elementelor din fisierul de proprietăți;
* *DeepCloner* – atât în algoritm cât și în interfață este necesară o copie a unor elemente fără ca vreo referință să fie păstrată întrecele două. Aceasta problemă poate fi rezolvată folosind serializarea. Astfel, obiectele pot fi salvate într-un stream (flux de date) și pot fi restaurate din acesta. Dacă un obiect este serializar, atunci orice alt obiect referit de acesta este serializat. Clasa *DeepCloneer* oferă o funcție ce are întocmai acest scop, creând o copie completă a unui obiect primit ca parametru prin serializarea și deserializarea acestuia.
* *FreeMarkerDataLoader* – această clasă este responsabilă cu generarea rapoartelor sub forma unui set de pagini HTML folosind template-urile din pachetul *utilities.freemarker.templates*.



*Fig.15. Diagrama de clase - pachetul utilities*

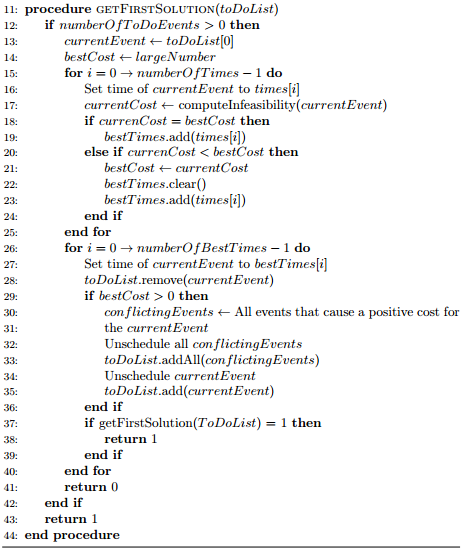
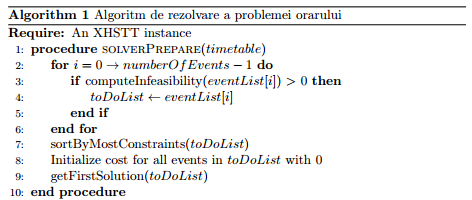
Clasele din pachetul *utilities* sunt reprezentate în *Fig.15. Diagrama de clase – pachetul utilities.* Ele sunt folosite de cele din pachetul compute, pentru rularea algoritmului, precum și de cele din *view*, anumite funcții fiind necesare în implementarea interfeței.

2.3 Algoritmul de rezolvare a problemei orarului

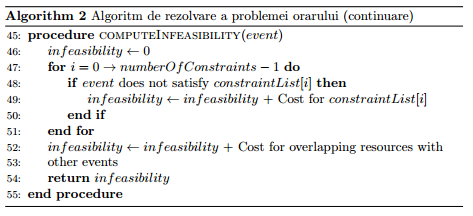
Algoritmul ales pentru rezolvarea problemei programării evenimentelor este unul euristic ce se bazează pe constrângerile impuse de utilizator asupra datelor. Datorită faptului că pentru rezolvarea acestui tip de problemă există o multitudine de posibili algoritmi, arhitectura aleasă pentru aplicație permite adăugarea facilă a mai multor implementări diferite. În continuare este descrisă implementarea unui algoritm recursiv ce construiește orarul respectând constrângerile impuse.

Algoritmul este descris în pseudocod în *Fig. 16. Algoritm - rezolvare problema orarului* și *Fig. 17. Algoritm - rezolvare problema orarului (continuare)*, cu cele trei proceduri principale. Procedura *solverPrepare(*timetable*)* este cea care trebuie apelată inițial cu obiectul *timetable* ce conține o instanță a problemei în formatul XHSTT. Aici este inițializată lista de evenimente ce trebuiesc programate *(toDoList)* cu toate evenimentele ce au scorul de infezabilitate strict pozitiv.

Un exemplu sunt evenimentele ce încă nu au asignate un interval de timp sau evenimentele programate dar care utilizează resurse ce sunt deja angajate în alte evenimente. Lista este apoi sortată descrescător în funcție de numărul de constrângeri impuse pentru fiecare eveniment.



*Fig. 16. Algoritm - rezolvare problema orarului*

**

*Fig. 17. Algoritm - rezolvare problema orarului (continuare)*

Costul de programare al fiecărui eveniment este inițializat cu 0. Acest cost este o notă a gradului de dificultate al programării unui eveniment. Cu cât este mai mare, cu atât mai dificil este un eveniment de programat. Acest cost crește pentru pentru un eveniment de fiecare dată când programarea respectivului eveniment este ștearsă.

Funcția *getFirstSolution(toDoList)* realizează parcurgerea în adâncime a spațiului soluțiilor. Metoda este recursivă, returnând valoarea 1 – ce reprezintă faptul că toate evenimentele au fost programate – atunci când *toDoList* nu mai are nici un element. În caz contrar, este ales primul element din listă și se încearcă programarea acestuia în toate intervalele de timp existente, calculându-se totodată scorul de infezabilitate pentru respectiva programare.

Sunt reținute în *bestTimes* intervalele ce au cel mai mic scor de infezabilitate, adică cele care respectă cel mai bine constrângerile impuse și care sunt cel mai ușor de programat în orarul generat până la momentul curent. Pentru un scor egal cu 0 evenimentul curent este programat cu primul interval din lista de best costs, este scos din lista de evenimente *toDoList* și este reapelată funcția *getFirstSolution(toDoList)*.

Dacă însă scorul de infezabilitate este sctrict pozitiv, adică evenimentul fie folosește resurse ce sunt deja programate fie nu respectă constrângerile, atunci toate evenimentele ce sunt incompatibile cu evenimentul curent sunt deprogramate și sunt introduse din nou în *toDoList* împreună cu evenimentul curent și funcția *getFirstSolution(toDoList)* este reapelată. Dacă această procedură returnează 0 înseamnă că nu a fost posibilă programarea unui eveniment în orarul curent.

Procedura *computeInfeasibility(event)* primește ca parametru un eveniment cu un interval alocat. Se adaugă la scorul unui eveniment costul asociat fiecărei contrângeri nerespectate. Acest cost poate fi calculat în mod diferit pentru fiecare tip de constrângere. De asemenea, la *infeasibility* se adaugă și costul de programare pentru fiecare eveniment care are resurse a căror programare se suprapune cu prugramarea eveniementului curent. Numărul returnat simbolizează cât de fezabilă este programarea curentă a evenimentului primit ca parametru.

2.4 Constrângerile implementate

Constrângerile reprezintă relații peste variabilele ce definesc spațiul soluțiilor, specificând restricții pentru valorile pe care aceste variabile le pot lua. Fiind structurate într-un mod concis și natural, pentru a fi înțelese și aplicate cât mai ușor de către utilizator, constrângerile permit modelarea problemei orarului astfel încât să descrie cât mai bine nevoile unei școli. În programarea orarului este necesar ca repartizarea rezultată a evenimentelor să corespundă cerințelor elevilor și profesorilor implicați. Însă fiecare școală poate avea cerințe diferite, a căror modelare este realizată prin constrângerile impuse datelor.

În implementarea actuală, adăugarea unui tip de constrângere necesită adăugatea unei clase ce implementează clasa abstractă *Constraint*, obținând astfel acces la toate proprietățile acesteia și implementând funcția abstractă *validate(Event)*. Acestă funcție este esențială în algoritmul implementat pentru calculatea scorului de infezabilitate. Primind ca parametru fie un *Event* fie un *Resource*, acestea având asociate un element de tip *Time*,valoarea returnată reprezintă costul programării elementului primit în intervalul de timp specificat din punctul de vedere al constrângerii curente.

Tipurile de constrângeri implementate și detaliile despre funcția validate pentru fiecare tip sunt descrise în următoarele rânduri:

* *AssignResourceConstraint* și *AssignTimeConstraint* – constrângeri de bază impuse automat pentru orice orar, pentru toate evenimentele; funcția *validate(Event)* verifică dacă evenimentul are asignate resursele necesare și dacă este programat. In caz contrar returnează valoarea *weight* asignată constrângerii.
* *LimitRepeatActivityConstraint* – constrângere pentru limitarea numărului de activități de același tip ce pot fi programate consecutiv pentru o clasă. Rezultatul funcției *validate(Event)* este valoarea *weight* a contrângerii în caz că aceasta nu este respectată (evenimentul dat se repetă de mai mult de *maximumNumberOfRepetitions*) și 0 altfel. În *Fig. 18* este redată implementarea funcției *validate(Event)*, unde se parcurge repetitiv lista de evenimente programate și se verifică dacă la ultima repetiție a crescut intervalul în care se repetă evenimentele. Dacă după o repetiție acest interval nu mai crește înseamnă că s-au găsit toate elementele consecutive și, dacă funcția nu a returnat deja rezultatul *weight* din cauza faptului că s-a depășit numărul maxim de repetiții permis, se returnează costul 0, adică, din punctul de vedere al constrângerii curente, evenimentul nu implică niciun cost de programare la intervalul setat.

|  |
| --- |
| @Override  public int validate(Event event) {  //The interval in which the events with the same description are found  int upperInterval = event.getTime().getId();  int lowerInterval = event.getTime().getId();  if(eventDescriptions.contains(event.getDescription())) {  conflictingEvents = new ArrayList<>();  Resource studyGroup = getStudyGroupForEvent(event);  boolean intervalGotBigger = true;  while(intervalGotBigger) {  intervalGotBigger = false;  for (Event e : programmedEvents.getEvents()) {  if (!e.getId().equals(event.getId()) && e.getDescription().  equals(event.getDescription())) {  Resource eStudyGroup = getStudyGroupForEvent(e);  if(studyGroup.getId().equals(eStudyGroup.getId())) {  if (upperInterval - e.getTime().getId() == 1) {  upperInterval--;  intervalGotBigger = true;  conflictingEvents.add(e);  }    if (e.getTime().getId() - lowerInterval == 1) {  lowerInterval++;  intervalGotBigger = true;  conflictingEvents.add(e);  }  if(lowerInterval - upperInterval > maximumNumberOfRepetitions){  return weight;  }  }  }  }  }  }    return 0;  } |

*Fig.18. Implementare funcția validate pentru clasa LimitRepeatActivityConstraint*

* *LimitRepeatDailyConstraint* – se adresează tot limitării repetării evenimentelor, însă nu a celor consecutive precum constrângerea anterioară, ci a celor zilnice. Funcția *validate(Event)* verifică dacă resursele de tip *studyGroup* implicate în eveniment nu au deja programate în ziua evenimentului curent activități de același tip. Implementarea sa este descrisă în *Fig. 19*. *Implementare funcția validate pentru clasa LimitRepeatDailyConstraint*. De exemplu, evenimentul de verificat este o oră de Matematică, cu clasa 7B, miercuri în intervalul 10:00-10:50. Dacă grupul 7B are deja programată o oră de matematică miercuri și constrângerea specifică faptul că poate fi maxim o oră de matematică zilnic pentru această clasă, atunci funcția *validate(Event)* va returna costul *weight* setat pentru constrângere.

|  |
| --- |
| @Override  public int validate(Event event) {  Resource studyGroup = getStudyGroupForEvent(event);  conflictingEvents = new ArrayList<>();  if(eventDescriptions.contains(event.getDescription())) {  int repeated = 0;  for (Event e : programmedEvents.getEvents()) {  if (!e.getId().equals(event.getId()) &&  e.getDescription().equals(event.getDescription()) &&  e.getTime().getDay().equals(event.getTime().getDay())) {  Resource eStudyGroup = getStudyGroupForEvent(e);  if(studyGroup.getId().equals(eStudyGroup.getId())) {  repeated++;  conflictingEvents.add(e);  }    if (repeated > maximumNumberOfRepetitions) {  return weight;  }  }  }  return 0;  } |

*Fig.19. Implementare funcția validate pentru clasa LimitRepeatDailyConstraint*

* *LimitIdleTimesConstraint* – are rolul de a sigura constrolul asupra numărului de intervale libere atât în orarul profesorilor, cât și în cel al elevilor; funcția *validate(Event)* returnează costul constrângerii dacă pentru programarea evenimentului curent se creează într-o zi din programul resurselor ce țin de eveniment o fereastră mai mare, ce nu poate fi acoperită respectând numărul maxim de ore ce pot fi lăsate libere nici dacă toate evenimentele rămase care folosesc resursa curentă ar fi programate în fereastra respectivă; Implementarea este descrisă în figura *Fig.20.*

|  |
| --- |
| @Override  public int validate(Event event) {  int cost = 0;  Map<Integer, Event> timeIdEventMap = new HashMap<>();  conflictingEvents = new ArrayList<>();  List<Resource> resources = event.getResources().getResources();  for(Resource resource: resources) {  if (appliesToResources.getResources().contains(resource)) {  //The times when the resource is scheduled  List<Time> resourceTimes =  getScheduledTimesForResource(resource);    //Sorting the times when the resource is scheduled  sortTimesById(resourceTimes);  if (resourceTimes.size() > 1) {  int eventTimeIndex = resourceTimes.indexOf(t);  int maximumPosibleEvents =  unprogrammedEventsWithResource(resource);  if(eventTimeIndex>0){  Time previousTime = resourceTimes.get(eventTimeIndex-1);  if(previousTime.getDay().equals(event.getTime().getDay())){  int hoursBetween = event.getTime().getId() –  previousTime.getId();  if (hoursBetween-maximumPosibleEvents> maximumIdleHours {  cost += this.getWeight() \* (hoursBetween –  maximumIdleHours);  conflictingEvents.add(timeIdEventMap.  get(previousTime.getId()));  }  }  }    if(eventTimeIndex<resourceTimes.size() - 1){  Time nextTime = resourceTimes.get(eventTimeIndex+1);    if(nextTime.getDay().equals(event.getTime().getDay())) {  int hoursBetween = nextTime.getId() –  event.getTime().getId();    if (hoursBetween-maximumPosibleEvents> maximumIdleHours){  cost += this.getWeight() \* (hoursBetween –  maximumIdleHours);  conflictingEvents.add(timeIdEventMap.  get(nextTime.getId()));  }  }  }  }  }  }  return cost;  } |

*Fig.20. Implementare funcția validate pentru clasa LimitIdleTimesConstraint*

* *LimitResourceAvailableTimesConstraint* – care are rolul de a limita intervalele de timp în care o resursă poate fi programată, implementarea ei fiind prezentată mai jos în figura *Fig.21. Implementare funcția validate pentru clasa LimitResourceAvailableTimesConstraint*; valoarea returnată de funcția *validate(Event)* este costul *weight* pentru fiecare resursă implicată în evenimentul transmis ca parametru dacă aceasta nu poate fi programată ăn intervalul dorit.

|  |
| --- |
| @Override  public int validate(Event event) {  int cost = 0;  for(Resource r: event.getResources().getResources()){  for(Resource r1: appliesToResources.getResources()){  if(r1.getId().equals(r.getId())){  boolean ok = true;  for(Time t: notAvailableTimes.getTimes()){  if(t.getId() == event.getTime().getId()){  ok = false;  break;  }  }    if(!ok){  cost += weight;  }    break;  }  }  }    return cost;  } |

*Fig.21. Implementare funcția validate pentru*

*clasa LimitResourceAvailableTimesConstraint*

Constrângerile implementate ajută la crearea unui orar viabil și ușor de folosit. Structura de clase implementată permite adăugarea cu ușurință a unor noi clase ce structurează diferite tipuri de constrângeri pentru o modelare și mai precisă a inputului problemei orarului.

Cap.3. Direcții de dezvoltare

În acest capitol sunt decrise alte abordări în rezolvarea problemei orarului însă la nivel teoretic, aceasta fiind baza pentru o dezvoltare ulterioară a aplicației din punct de vedere algoritmic. De asemenea este cuprisă și o secțiune despre tehnologii ce pot fi folosite pentru deployment-ul aplicației.

3.1 Programare bazată pe constrângeri

Pentru rezolvarea unei probleme specificate prin constrângeri o soluție este folosirea unui **backtracking cronologic**. Această strategie parcurge spațiul soluțiilor în adâncime. Nodurile de căutare reprezintă asignări parțiale pentru variabilele de decizie ale problemei. Dacă o asignare parțială nu poate fi extinsă fără a încălca nici o constrângere, backtracking-ul cronologic se întoarce la cel mai recent nod de căutare ce prezintă posibile soluții și continuă explorarea alternativelor. Eficiența acestei metode depinde într-o măsură mare de calitatea alegerii arborelui de soluții.

Pentru a reduce efortul de căutare backtracking-ul cronologic este îmbinat cu **strategii de propagare a constrângerilor**, acestea fiind apelate după fiecare extindere a soluției. În mod normal, propagarea constrângerilor este realizată de **constraint solvere**. Acestea există independent unele de altele și comunică prin variabilele domeniului. Când un contraint solver este aplicat unei constrângeri, informațiile pe care acesta le are la dispoziție sunt constrângerea în sine și starea variabilelor la momentul respectiv. Această structură permite adaptarea cu ușurință a solverelor la cerințe noi.

În practică, o librărie ce este dedicată constraint programming este **Choco solver**. Utilizatorul modelează problema într-un model declarativ exprimând constrângerile ce trebuie satisfăcute în fiecare soluție. Apoi problema este rezolvată prin alternarea algoritmilor de filtrare cu un mecanism de căutare. Elementele de bază ale implementării unui algoritm de rezolvare ce folosește Choco solver sunt următoarele:

* *Model* – ce permite descrierea problemei într-un mod simplu, declarativ; pur și simplu sunt înregistrate variabilele problemei și constrângerile; scopul problemei este acela de a asigna o valoare fiecărei variabile ținând cont de domeniul ei de definiție și respectând constrângerile impuse;
* *Solver* – care citește variabilele și constrângerile din model și structurează arborele de soluții, folosind o strategie de căutare; acesta poate fi configurat în diferite moduri pentru a optimiza căutarea soluțiilor; căutarea poate fi oprită după găsirea primei soluții sau poate fi continuată până la găsirea tuturor soluțiilor sau până la parcurgerea întregului arbore și concluzionarea cu faptul că problema nu poate fi soluționată.

Programarea bazată pe constrângeri este o direcție de dezvoltare viabilă pentru stadiul actual al aplicației, structura implementată fiind inspirată întocmai din acest concept. Atât în implementarea actuală cât și într-o implementare folosind un algoritm de constraint programming, ideea principală este aceea ca utilizatorului să îi revină doar îndatorirea de a specifica cât mai clar problema, modalitatea în care este aceasta rezolvată ținând în întregime de algoritm.

3.2 Algoritmi genetici

Comportamentul algoritmilor genetici poate fi explicat într-o formă extrem de simplistă prin descoperirea, accentuarea și recombinarea părților bune din multiple soluții. Ideea principală este aceea că soluțiile bune tind să fie compuse din componente potrivite – combinații de valori ce conferă un rezultat mai bun de fitness pentru soluția în care se regăsesc.

Majoritatea metodelor ce se încadrează în categoria algoritmilor genetici au următoarele elemente în comun: populația de **soluții candidat** (cromozomi), care evoluează de-a lungul unor **generații** sub controlul unei **funcții de fitness** ce măsoară meritul individual. Selecția în funcție de scorul de fitness, crossover și mutații pentru a produce noi elemente. Cea mai simplă formă a acestui tip de algoritmi implică următoarele trei tipuri de operatori:

* *Selecție* – acesta selectează cromozomi din populație pentru reproducere. Cu cât este mai potrivit cromozomul pentru a soluționa problema, cu atât este mai probabil ca acesta să fie selectat;
* *Crossover* – acest operator alege aleator o poziție și interschimbă secvențele ce o încadrează între doi cromozomi pentru a crea doi noi cromozomi;
* *Mutație* – care schimbă în mod aleator anumite părți din cromozom.

Structura unui algoritm de acest tip începe cu o populație generată random de *n* instanțe (posibile soluții pentru problemă), calculându-se pentru fiecare valoarea de fitness. Apoi pentru a crea următoarea populație de *n* instanțe se aplică cei trei operatori menționați mai sus. Astfel, se poate selecta o pereche de cromozomi din populația curentă. Selecția este realizată cu înlocuire, acesta însemnând faptul că același cromozom poate fi selectat de mai multe ori pentru a deveni părinte.

Cu o anumită probabilitate se poate realiza crossover-ul unei perechi la un punct al acestora ales random, cu probabilitate uniformă pentru a forma doi noi cromozomi. Dacă nu va avea loc crossover-ul, urmașii vor fi copii exacte ale părinților. O altă posibilitate este mutația a doi cromozomi, cei doi noi rezultați fiind parte din noua populație. Acești pași se repetă până se atinge numărul *n* de cromozomi în noua populație, moment în care aceasta ia locul vechii generații. Importante sunt aici probabilitățile cu pentru aplicarea operatorilor, precum și alegerea funcției de fitness.

Implementarea unui algoritm genetic pentru rezolvarea problemei orarului se bazează pe construirea funcției de fitness în jurul constrângerilor, indivizii din fiecare generație fiind reprezentați de soluții complete sau parțiale. O soluție este cu atât mai bună cu cât respectă mai bine constrângerile impuse sistemului. Pentru operatorii mutație și crossover se va avea în vedere ca soluțiile din noua generație rezultate să respecte numărul de ore pentru fiecare disciplină, profesor și clasă așa cum este el setat la început.

Un individ fiind reperezentat de o soluție a problemei, adică de o listă de evenimente ordonate după id, crossover-ul se poate realiza alegând cu probabilitatea specifică operației unul dintre id-urile din listă și interschimbând programările evenimentelor din secvențele alese. Mutația se poate realiza în mod asemănător, alegând cu probabilitatea de mutație evenimentele din listă pe care să le reprogrameze.

Acest tip de algoritm reprezintă o opțiune viabilă pentru structura de date aleasă datorită faptului că, în termeni generali, constrângerile pun la dispoziție, prin funcția *validate(Event)*, chiar un scor al capacității unui eveniment de a îndeplini cerințele necesare. Prin combinarea acestor scoruri pentru toate evenimentele unei soluții și pentru toate constrângerie existente se poate crea funcția de fitness necesară.

3.3 Java Web Start

Software-ul Java Web Start permite descărcarea și rularea aplicațiilor Java de pe web, garantând ca la fiecare rulare aplicația să fie actualizată la cea mai recentă versiune și anulând procesele complicate de instalare și de upgrade. Java Web Start este inclus în JRE (Java Runtime Environment) începând cu versiunea Java 5.0. Software-ul este lansat automat atunci când o aplicație ce îl folosește este descărcată penru prima oară. Întreaga aplicație este ctocată local. La fiecare rulare a aplicației componenta Java Web Start verifică site-ul aplicației pentru versiune mai recentă și în caz că o astfel de versiune există ea este descărcată și lansată.

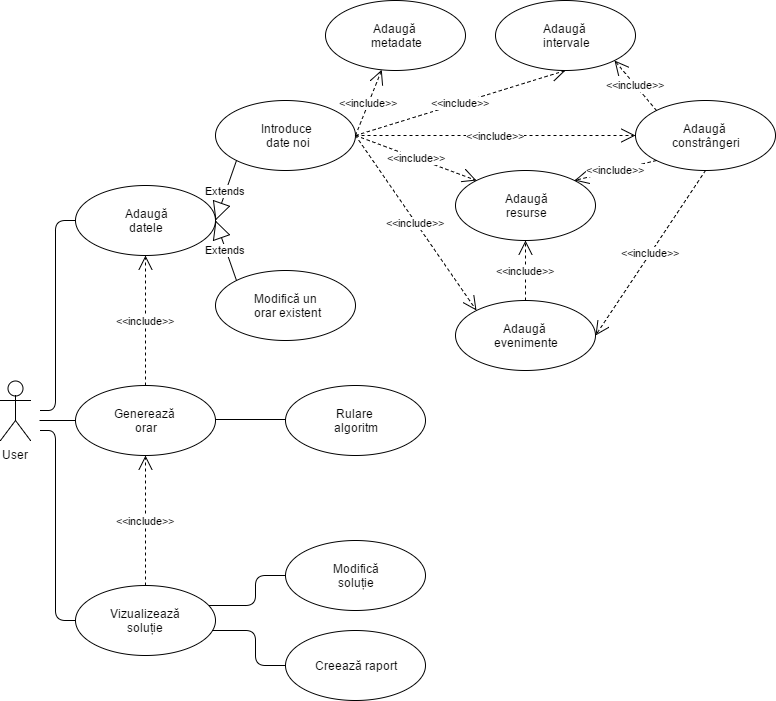
Pentru a realiza corect deploy-ul, toate fișierele ce fac parte din aplicație trebuie să fie accesibile printr-un server Web. Aceasta poate fi realizată prin acordarea accesului serverului Web la unul sau mai multe fișiere JAR împreună cu un fișier JNLP (Java Network Launching Protocol). Acest tip de fișier definește modalitatea de lansare a aplicației. Când un utilizator selectează un link care refereniază un fișier JNLP, browser-ul lansează JavaWeb Start care descarcă și afișează utilizatorului opțiunea de a rula aplicația Java.

Această tehnologie are un set de avantaje, precum faptul că aplicația devine disponibilă pentru o varietate de platforme (Windows, Linux, Solaris), mai multe versiuni de Java SE sunt beneficiază de suport, o aplicație poate rula independent de browserul Web, pentru operațiile off-line și aplicația poate fi lansată și din shortcut-uri. Din punctul de vedere al securității, aplicațiile ce folosesc JavaWebStart sunt rulate într-un environement restricționat, sau *sandbox.* Astfel, utilizatorii sunt protejați de tentativele de a transmite cod ce poate afecta fișiere locale sau ce poate încerca să acceseze sau să deterioreze date sau rețele. Aplicațiile din sandbox rămân în sandbox, neputând accesa fișiere locale sau rețele.

Pentru a ajuta la deployment-ul mai facil al aplicației de generare a orarului, Java Web Start este o opțiune dedicată acestui tip de software, cu avantaje atât din punctul de vedere al utilizatorului, ușurând procesul de instalare și de update și permițand rularea într-un mediu ce poate fi configurat după o gamă largă de opțiuni, cât și din punctul de vedere al securității, restricționând accesul pe care îl are aplicația în sistemul utilizatorului.

Cap.4. Cazuri de utilizare

Funcționalitățile sistemului sunt descrise în diagrama *Fig.22. Diagrama generală de utilizare a aplicației* ce structurează scenariile de utilizare ale aplicației CodeHour. Această figură descrie cum interacționează utilizatorul cu sistemul și cum reacționează acesta.



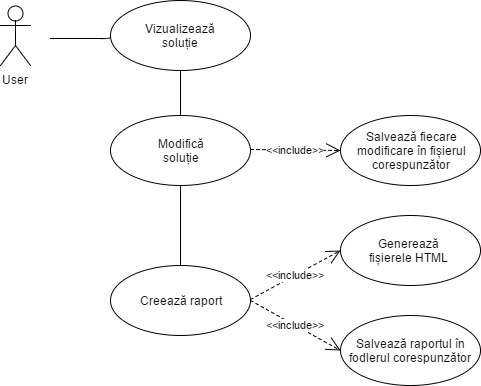
*Fig.22. Diagrama generală de utilizare a aplicației*

Cele trei opțiuni principale pe care le are userul la dispoziție sunt:

* Adaugă datele – acțiune ce se poate realiza fie pornind de la un fișier deja existent și modificându-l, fie prin introducerea unor date noi. Pentru acesta din urmă este necesară adăugarea de metadate, intervale și resurse, pentru ca apoi să urmeze evenimentele (ce depind de resurse), respectiv constrângerile (ce depind de evenimente, resurse și intervalele de timp).
* Generează orar – opțiune ce declanșează rularea algoritmului pentru fișierul selectat. Pentru ca această opțiune să fie accesibilă, este necesar ca un set valid de date să fie introdus. La finalizarea rulării, soluția este automat afișată în ecranul de afișare.
* Vizualizează soluție – această opțiune este posibilă doar pentru fișierele ce au deja generată o soluție. Ecranul e vizualizare nu permite doar structurarea orelor în funcție de resurse (elevi, profesori și săli), ci și modificarea soluției alese pentru a se potrivi mai bine cerințelor. Când utilizatorule este mulțumit de forma la care a ajuns orarul el poate crea un raport pe care să îl distribuie apoi în mediul online.

Diagrama adăugată mai sus este una cuprinzătoare ce modelează toate posibilitățile de acțiune pe care le are utilizatorul, descriind acțiunile necesare pentru crearea unui orar de la introducerea datelor până la crearea raportului final.

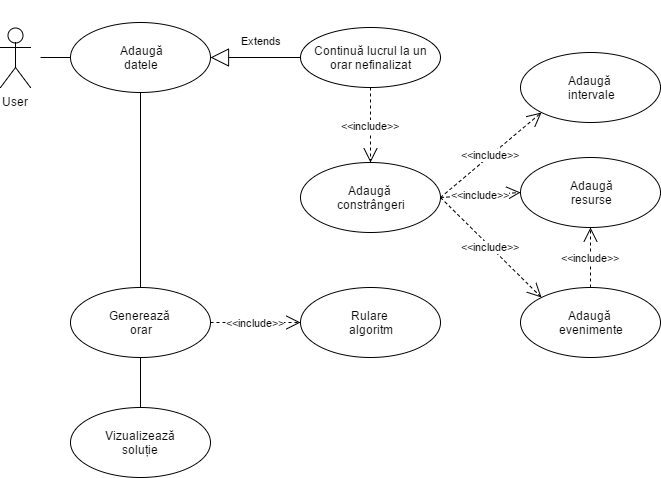
Există, desigur și cazuri mai mici de scenarii de utilizare, reprezentând doar o parte din flow-ul complet*.* Un exemplu ar fi vizualizarea unui orar soluționat și modificare acestuia direct din ecranul de vizualizare, urmată de crearea unui raport pentru noul orar. Acest scenariu este reprezentat în *Fig.23. Diagrama Use Case 1.*



*Fig.23. Diagrama Use Case 1*

Un alt scenariu este acela în care este completat un orar nefinalizat, după care se generează soluția pentru date introduse. Acesta este prezentat în *Fig.23. Diagrama Use Case 2*. Utilizatorul contiuă adăugarea datelor prin completarea constrângerilor, pentru care este necesar ca intervalele de timp, resursele și evenimentele să fie adăugate în prealabil.

La apăsarea butonului *Generează orar* este rulat algoritmul, iar cand o soluție este găsită aceasta este afișată în ecranul de vizualizare. Utilizatorul are acum generată și salvată soluția pentru date, putând oricând să o încarce pentru modificări și pentru generarea raportului.



*Fig.24. Diagrama Use Case 2*

Acestea sunt doar o parte dintre posibilele scenarii de utilizare a aplicației, prima diagramă, din *Fig.22. Diagrama generală de utilizare a aplicației*, este una reprezentativă pentru opțiunile generale ce pot fi accesate, precum și comportamentul aplicației pentru acestea.

Concluziile Lucrării

Aplicația CodeHour oferă asistență pentru rezolvarea problemei orarului oferind o interfață intuitivă, structurată astfel încât să ușureze procesul de introducere a datelor și oferind posibilitatea de a vizualiza și modifica apoi soluția rezultată pentru a se conforma cât mai îndeaproape cerințelor elevilor și profesorilor. Procesul este apoi finalizat prin crearea rapoartelor pentru distribuirea orarului generat.

Tehnologiile folosite precum și arhitectura implementată nu necesită resurse puternice pentru o rulare eficientă, CodeHour fiind cu atât mai potrivită pentru școli sau licee.

Structura folosită pentru datele problemei este una ce se bazează pe constrângeri, acestea înglobând cerințele logice, iar algoritmul oferă soluții viabile, bazându-se pe structura aleasă. Arhitectura aplicației este una ușor de extins, permițând dezvoltarea ulterioară a acesteia prin adăugarea mai multor tipuri de constrângeri, precum și a unor algoritmi bazați pe diferite paradigme. De asemenea, celelalte funcționalități ale aplicației sunt rapide și eficiente.

Viitoarele extensii pot fi legate nu doar de dezvoltarea unui set de algoritmi pentru rezolvarea problemei orarului, ci și de procesul de deployment al aplicației.

Bibliografie

[1] Constraint Programming: In Pursuit of the Holy Grail, Roman Barták:

[*http://kti.ms.mff.cuni.cz/~bartak/downloads/WDS99.pdf*](http://kti.ms.mff.cuni.cz/~bartak/downloads/WDS99.pdf)

[2] JavaFX Overview:

[*http://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm*](http://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm)

[3] JavaFX CSS:

[*https://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html*](https://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html)

[4] A Hyperheuristic for generating Timetables in the XHSTT Format, Mathijs ter Braak:

[*http://essay.utwente.nl/62055/1/MSc\_M\_ter\_Braak.pdf*](http://essay.utwente.nl/62055/1/MSc_M_ter_Braak.pdf)

[5] Structura XHSTT:

[*https://www.utwente.nl/ctit/hstt/tutorial/*](https://www.utwente.nl/ctit/hstt/tutorial/)

[6] Apache FreeMarker:

[*http://freemarker.org/*](http://freemarker.org/)

[7] aScOrare:

[*https://ascorare.ro/#!/home/features*](https://ascorare.ro/#!/home/features)

[8] eOra:

[*https://profs.info.uaic.ro/~orar/app/doc/manual.html*](https://profs.info.uaic.ro/~orar/app/doc/manual.html)

[9] JAXB:

[*http://www.oracle.com/technetwork/articles/javase/index-140168.html*](http://www.oracle.com/technetwork/articles/javase/index-140168.html)

[10] JAXB Annotations:

[*http://www.techferry.com/articles/jaxb-annotations.html*](http://www.techferry.com/articles/jaxb-annotations.html)

[11] JAXB Annotations:

[*https://www.ms.sapientia.ro/~manyi/teaching/oop/oop\_romanian/curs15/curs15.html*](https://www.ms.sapientia.ro/~manyi/teaching/oop/oop_romanian/curs15/curs15.html)

[12] Models and Algorithms for School Timetabling – A Constraint-Programming Approach, Michael Marte:

[*http://www.en.pms.ifi.lmu.de/publications/dissertationen/PMS-DISS-2003-1/PMS-DISS-2003-1.pdf*](http://www.en.pms.ifi.lmu.de/publications/dissertationen/PMS-DISS-2003-1/PMS-DISS-2003-1.pdf)

[13] Choco solver:

[*http://www.choco-solver.org/*](http://www.choco-solver.org/)

[14] Documentație Choco solver:

[*http://www.dcs.gla.ac.uk/~pat/cpM/jchoco/jchoco2.1.0/manuals/choco.pdf*](http://www.dcs.gla.ac.uk/~pat/cpM/jchoco/jchoco2.1.0/manuals/choco.pdf)

[16] Algoritmi genetici, Breabăn Mihaela:

[*https://profs.info.uaic.ro/~pmihaela/GA*](https://profs.info.uaic.ro/~pmihaela/GA/laborator3.html)

[15] An Introduction to Genetic Algorithms, Mitchell Melanie:

[*https://svn-d1.mpi-inf.mpg.de/AG1/MultiCoreLab/papers/ebook-fuzzy-mitchell-99.pdf*](https://svn-d1.mpi-inf.mpg.de/AG1/MultiCoreLab/papers/ebook-fuzzy-mitchell-99.pdf)

[16] Java Web Start:

[*https://www.java.com/en/download/faq/java\_webstart.xml*](https://www.java.com/en/download/faq/java_webstart.xml)