UNIVERSITATEA "POLITEHNICA" TIMISOARA
Facultatea de Electronica si Telecomunicatii
în colaborare cu
École Nationale Supérieure de Télécommunication Bretagne

Proiect de diploma

COMPONENTE DE INDEXARE A RESURSELOR BIOINFORMATICE, ÎN CADRUL PROIECTULUI BIOSIDE

Coordonatori :
Profesor Miranda NAFORNITA
Conferentiar Philippe PICOUET

Autor :
Cristina Viorica TETCU

Sesiunea iunie
- 2007 -

Cuprins:

Tabele:

Figuri:

Capitolul 1. Introducere

Proiectul de diploma a fost realizat sub îndrumarea doamnei profesor Miranda Nafornita, în interiorul departamentului de Comunicatii al Facultatii de Electronica si Telecomunicatii din cadrul UPT, în colaborare cu departamentul LUSSI (Logique des Usages, Sciences Sociales et sciences de l'Information) al Universitatii ENST Bretagne (École Nationale Supérieure de Télécommunication Bretagne). Proiectul se înscrie în domeniul bioinformaticii, domeniu care a cunoscut în ultimele decenii o puternica dezvoltare, existând în momentul de fata diferite unelte informatice capabile sa manipuleze date biologice cu scopul de a extrage informatia digitala continuta de acestea. EMBOSS (the European Molecular Open Software Suite) si Pise (Pasteur Institute Software Environment) sunt câteva exemple.

În cadrul departamentului LUSSI, o echipa de cercetatori si programatori colaboreaza pentru dezvoltarea unei platforme Java, numita BioSide. Destinata gestionarii resurselor bioinformatice, BioSide este o platforma complexa care cuprinde mai multe unelte si interfete, printre care si BioDesc, unealta cu ajutorul careia se realizeaza indexarea programelor.

Lucrarea de fata contine o parte de dezvoltare, concretizata în implementarea facilitatii de import – export dintre interfata si baza de date BioDesc, precum si o parte de analiza care cuprinde: analiza Biodesc, analiza ACD (AJAX Command Definition) si analiza comparativa ACD - BioDesc. Pentru finalizarea partii de dezvoltare sunt necesare cunostinte în domeniile: baze de date, XML (eXtensible Markup Language), Java si UML (Unified Modeling Language).

Tinând cont de faptul ca activitatea care sta la baza acestei lucrari face parte din stagiul desfasurat în cadrul departamentului LUSSI, având durata de sase luni, proiectul de fata prezinta concluziile partiale. În urmatoarele trei luni preocuparea se va axa spre analiza pachetului de programe Pise si pe determinarea posibilitatii de integrare a programelor din ansamblele EMBOSS si Pise în platforma BioSide. În vederea realizarii acestor obiective si tinând cont de necesitatile de ameliorare a BioDesc se ia în considerare posibilitatea restructurarii bazei de date.

În cadrul lucrarii, informatia este organizata în doua sectiuni: o sectiune în care este descris contextul activitatii din punct de vedere bioinformatic si tehnic si o sectiune destinata descrierii implementarii si analizelor.

Capitolul 2. Context

2.1. Context bioinformatic

2.1.1. Introducere în domeniul bioinformatic

Biologia se afla, la momentul actual, în toiul unei paradigme majore, condusa de tehnologia computerelor. Informatica a patruns în toate domeniile, iar biologia nu face exceptie devenind o stiinta informationala din multe puncte de vedere. Astfel cercetarile biologice se orienteaza din ce în ce mai mult spre calcul si analiza. Progresul rapid al cercetarilor în genetica si bio-chimie, combinat cu uneltele oferite de bio-tehnologia moderna, a permis obtinerea unui volum mare de secvente de date genetice.

Aflata la intersectia dintre biologia moleculara si stiinta calculatoarelor, bioinformatica poate fi definita ca analiza informatiilor biologice folosind instrumente proprii statisticii si tehnicii de calcul, ca stiinta dezvoltarii si utilizarii bazelor de date computationale si a algoritmilor în vederea accelerarii si îmbunatatirii cercetarii din domeniul biologiei.

Din momentul în care cercetatorii biologiei moleculare au început sa genereze secvente de ADN, în urma cu 27 de ani, era intuita implicarea matematicienilor si a informaticienilor, deoarece în experimentele biologice exista o mare cantitate de informatie digitala.

Acest domeniu de studiu a câstigat o identitate reala si un nume, bioinformatica, la mijlocul anilor '80. Înca de la începuturi, în bioinformatica s-au formulat trei concepte centrale, valabile pâna astazi:

- **Modul de reprezentare a datelor** – Reprezentarea datelor este realizata în sensul abstractizarii proceselor genetice, prin simplificarea grosolana a realitatii. Acesta se potriveste perfect reprezentarilor din analizele de calcul;

- **Problematica similaritatii** – Datorita evolutiei, secventele genomice similare au functii asemanatoare (genomul = un grup de cromozomi diferiti genetic care formeaza o unitate). De aceea algoritmii de compararea a secventelor si de gasire a regiunilor similare pot fi considerati "inima" bioinformaticii. La nivele

diferite, acesti algoritmi sunt utilizati pentru gasirea genelor, determinarea functiilor lor, etc;

- **Bioinformatica nu este o stiinta teoretica** – Nu exista algoritmi si teorii, în sensul academic traditional, care sa defineasca bioinformatica. Se poate spune ca manipularea datelor biologice reprezinta raspunsuri la nevoile biologilor, iar majoritatea bioinformaticienilor sunt procupati sa gestioneze si sa analizeze datele.

În prezent bioinformatica este utilizata în mai multe domenii precum: medicina moleculara, medicina personalizata, medicina preventiva, terapie genetica, dezvoltarea medicamentelor, în studiile asupra schimbarilor climatice, în descoperirea surselor alternative de energie, în biotehnologie, în cercetarea rezistentei la anitibiotice, etc.

Prin confluenta dintre biologie si informatica, aplicatiile software ale biologiei moleculare atrag tot mai mult atentia cercetatorilor si oamenilor de stiinta activi în domeniul stiintelor vietii. Exista tot mai multe unelte bioinformatice, care pun la dispozitia utilizatorilor o paleta larga de aplicatii ce difera prin modul de utilizare: acces direct la bazele de date, interfete simple în linia de comanda, GUI (Graphical User Interface), mobilitate, etc. încercând astfel sa duca la îndeplinire scopurile pentru care a fost creat domeniul bioinformaticii.

2.1.2. Introducere în BioSide

2.1.2.1. Descrierea BioSide

BioSide reprezinta o platforma de accesare la distanta a resurselor bioinformatice, permitând cooperarea între utilizatori, adica biologi si bioinformaticieni în munca lor de cercetare. Principiile platformei BioSide, au fost definite în urma unui studiu condus de departamentul LUSSI al École Nationale Supérieure de Télécommunication Bretagne. Acest studiu a permis constatarea bogatiei bioinformaticii, dar si a restrictiilor posibile în utilizarea resurselor bioinformatice.

Bioinformaticienii furnizeaza biologilor unelte pentru accesarea resurselor bioinformatice, care pot fi banci de secvente genetice, structuri de proteine, programe cu rol de aliniere a secventelor genetice sau programe filogenetice (filogenie = procesul evolutiei formelor organice). Dar "revolutia genomica", declansata de aparitia bioinformaticii, a generat o productie masiva si permanenta de noi resurse, atât programe, cât si banci de date, fara sa furnizeze unelte care sa permita biologilor descoperirea rapida a modului de utilizare, a limitelor si a utilitatii.

- **Obiectivele functionale**

Platforma BioSide are ca obiectiv sprijinirea biologilor în utilizarea resurselor bioinformatice. Pentru îndeplinirea acestui obiectiv proiectul propune, în acelasi timp, o viziune bioinformatica si o viziune biologica a acestor resurse. Partea de dezvoltare bioinformatica reprezinta un mediu de cooperare, în care sunt stocate bancile de date si programele bioinformatice, simplificându-le astfel utilizarea. Din punct de vedere functional (biologic) este permisa salvarea si publicarea diferitelor scenarii (etape de cercetare), dar si parametrizarea acestora astfel încât sa poata fi puse la dispozitia comunitatilor de biologi, sub forma de înlantuiri de programe, semnificative din punct de vedere biologic.

BioSide ofera un mediu pentru executarea si conservarea traseelor de cercetare, dar si pentru individualizarea mediului grafic. Astfel, BioSide reprezinta o aplicatie complexa care contine trei unelte distincte:

- ***BioSide*** – interfata utilizator - sistem, usor de utilizat, corespunzatoare viziunii biologice,
- ***BioDesc*** – dedicata descrierii resurselor bioinformatice,
- ***BioCommand*** – dedicata publicarii rezultatelor executiilor scenariilor.

Biologii se pot confrunta cu diferite probleme în timpul utilizarii resurselor bioinformatice. Câteva dintre ele sunt de natura informatica: localizarea resurselor, sintaxa de apel, formatul fisierelor, etc. Altele sunt legate de semantica, deseori complexa, a programelor sau de parametrizarea acestora. Exista si probleme legate de conservarea si gestionarea operatiilor si a rezultatelor.

Platforma BioSide, propune raspunsuri adaptate acestui ansamblu de probleme.

Pentru a raspunde la cea mai clasica problema, cea a localizarii resurselor, BioSide detine o arhitectura destinata cooperarii dintre utilizatori, care asigura transparenta în accesarea resurselor. Reuniunea resurselor este gestionata de frontal (fig 2.1.).

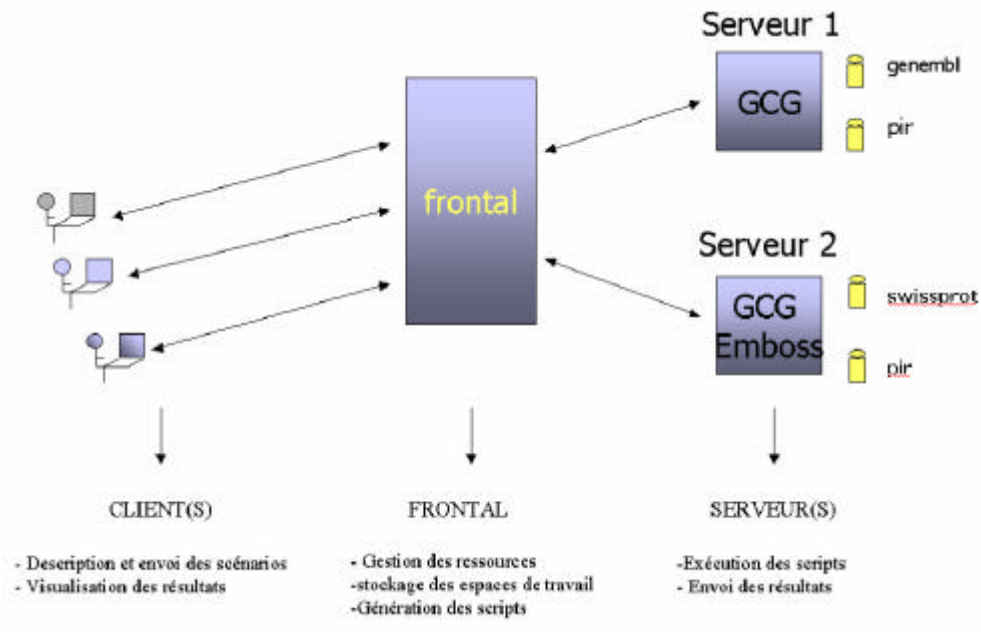

Fig. 2.1. Arhitectura BioSide

Pentru a se adapta cât mai bine la nevoile biologilor, BioSide introduce notiunea de scenariu. Un scenariu consta într-o înlantuire de programe în care rezultatele unui program reprezinta datele de intrare ale altor programe. Acest concept, de scenariu, permite biologilor sa pastreze vizibil întregul proces al lucrarii de cercetare. Fiecarui scenariu îi sunt asociate rezultatele, fapt care permite biologilor sa revina asupra rezultatelor intermediare în orice moment al procesului de cercetare.

Ansamblul de servicii, este înfatisat utilizatorilor, prin intermediul unei interfete grafice usor de utilizat, care invita biologii sa compuna, sa execute si sa gestioneze scenariile, prin reutilizare si redefinire, si sa vizualizeze rezultatele.

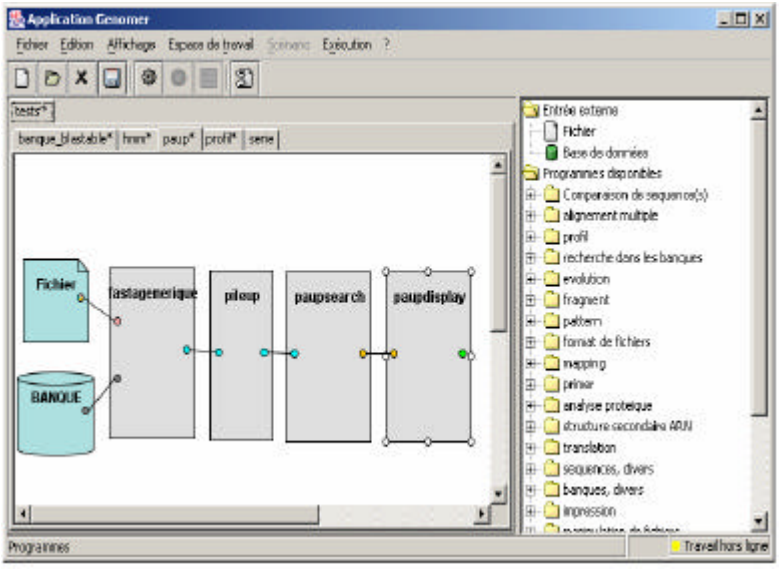

Fig. 2.2. Interfata BioSide

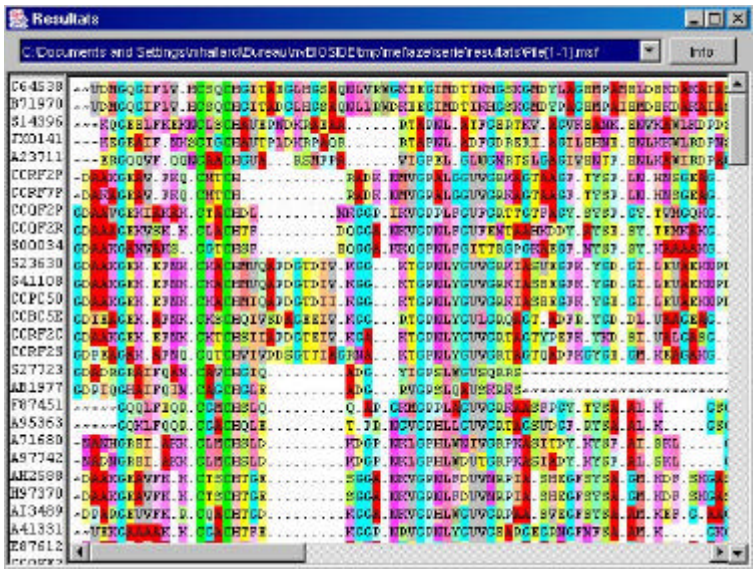

Fig. 2.3. Rezultatele obtinute în urma executiei unui scenariu BioSide

Anumite probleme, precum calitatea parametrizarii unui program sau coerenta semantica a unui scenariu, nu pot fi întotdeauna rezolvate în mod automat. Acesta situatie, rezultata din complexitatea algoritmilor sau din cauza metodelor interne ale programelor,

reprezinta o reala dificultate în utilizarea resurselor bioinformatice; biologii se rezuma la alegerea unei parametrizari implicite, care nu este adaptata nevoilor lor. În cadrul BioSide, urmeaza sa fie instalat un mecanism de publicare ce permite biologilor sa propuna scenarii sau parametrizari personalizate, pe care sa le poata justifica pe baza criteriilor biologice.

BioSide raspunde exigentelor, în termeni de transparenta a accesului la resurse, dar mai important este faptul ca furnizeaza biologilor un mediu care sustine si structureaza munca lor de investigare.

- **Studiu de caz – modalitati de utilizare a BioSide**

În momentul lansarii programului software, utilizatorul are acces la informatiile privind scenariile pe care le-a creat. Pentru fiecare scenariu, programul indica numarul de executii care au avut loc si pune la dispozitie noile rezultate. Asfel, utilizatorul poate sa examineze scenariile existente sau sa creeze noi scenarii.

În cazul crearii sau modificarii unui scenariu, utilizatorul poate vizualiza toate resursele la care are acces. Aceste resurse sunt de trei tipuri: bancile de date, programele si scenariile deja create (publicate de el sau de catre alti utilizatori). Resursele sunt organizate sub forma arborescenta, pornind de la categorii generale, cum sunt bancile proteice, bancile nucleotidice, programele de aliniere etc. În plus, utilizatorul dispune de o interfata grafica ce îi permite sa deseneze scenariul, sa conecteze programele între ele prin intermediul unei redirectari a iesirilor spre intrarile altor programe. El poate sa fixeze parametrii programelor invocând interfata prevazuta pentru acest efect, poate sa lanseze programul în executie sau sa îl salveze pentru ulterioare modificari, urmate de executie.

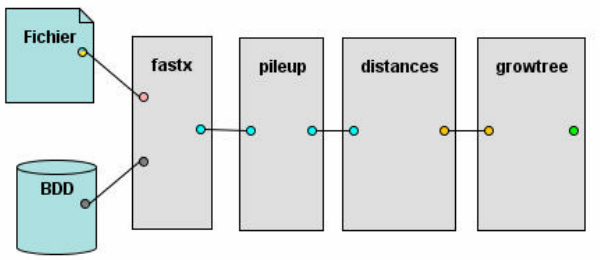

Fig. 2.4. Reprezentarea unui scenariu BioSide

Când examineaza un scenariu, utilizatorul are acces la informatii privind diferitele executii ale acestui scenariu: informatii despre rezultatele intermediare, rezultatele finale, dar si la informatii precise ale executiei scenariului cum sunt versiunea bancii de date, versiunea programelor, etc.

Un utilizator poate sa publice un scenariu, adica sa îl difuzeze împreuna cu parametrizarea corespunzatoare si cu o indexare; toate acestea sunt necesare pentru a-i permite unui alt biolog sa identifice functia scenariului si sa stie cum sa îl utilizeze.

2.1.2.2. Prezentarea BioDesc

BioDesc este o unealta a platformei BioSide, dedicata descrierii resurselor bioinformatice. Este compus dintr-o baza de date si o interfata usor de utilizat, permitând actualizarea si realizarea operatiile de import – export din/în format XML. Descrierile furnizate de BioDesc sunt utilizate de BioSide pentru a genera dinamic atât interfetele caracteristice parametrilor programelor cât si a scenariilor de executie a programelor utilizate în fluxul de date (workflows) BioSide.

- **Interfata BioDesc**

În partea stânga a ecranului este prezentata lista parametrilor programului, cu tipurile specifice fiecaruia.

În dreapta ecranului, pot fi distinse trei sectiuni, diferentiate prin culorile gri, albastru si verde. Sectiunea gri corespunde însusirilor generale ale programului (identificatorii intern si respectiv extern), în timp ce sectiunea albastra corespunde însusirilor parametrilor. Sectiunea verde prezinta valorile caracteristice tipului selectat în sectiunea din stânga ecranului.

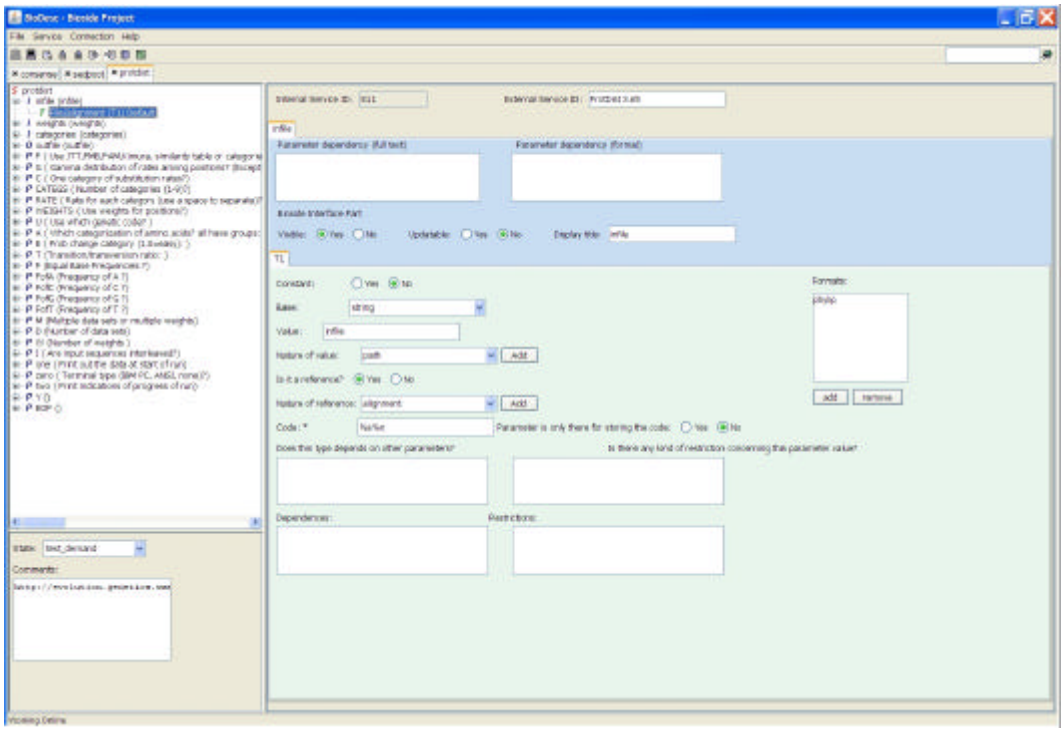

Fig. 2.5. Interfata BioDesc

2.1.3. Introducere în EMBOSS

EMBOSS (the European Molecular Open Software Suite) este un ansamblu de analize software, de tip Open Source, dezvoltat special pentru nevoile comunitatii de utilizatori din domeniul biologiei moleculare. Analizele software, incluse în ansamblul EMBOSS, sunt întâlnite sub denumirea de aplicatii sau programe. Programele EMBOSS au fiecare o functie bine definita si sunt scrise în limbajul de programare C. De obicei, un program EMBOSS primeste drept date de intrare unul sau mai multe fisiere, având un anumit tip de parametrizare de importanta majora pentru functia programului, spre a genera rezultate de iesire, organizate în fisiere, scenarii, pagini web sau doar sub forma de text simplu.

Fiecare aplicatie EMBOSS depinde de o definitie de tip interfata catre utilizator descrisa într-un fisier AJAX Command Definition (ACD). Fisierele ACD definesc toate caracteristicile de

care are nevoie aplicatia. Prin intermediul apelului unei biblioteci în timpul executiei codului sursa al aplicatiei, definitiile din fisierul ACD sunt transformate în cereri catre utitizator în linia de comanda.

Limbajul ACD acopera marea majoritate a tipurilor de date utilizate de aplicatiile EMBOSS. Acest limbaj este flexibil, astfel încât noi tipuri de date pot fi usor adaugate.

Fiecare program EMBOSS este însotit de un fisier ACD, care descrie parametrii de care acesta are nevoie. Astfel fisierul ACD contine informatii despre intrarile, iesirile si alti parametri de care are nevoie programul pentru ca executia sa se realizeze în conditii optime. De asemenea, fisierul ACD contine si alte caracteristici si constrângeri ale parametrilor: daca sunt obligatorii, care sunt valorile limita pe care le pot avea, daca depind de existenta altor parametri.

Ansamblul de programe EMBOSS a fost special construit pentru sistemul de operare UNIX. Apelarea, executarea simpla sau complexa, prin specificarea diferitelor optiuni, se realizeaza în linia de comanda.

2.1.3.1. Structura ansamblului de programe EMBOSS

Ca structura, exista cinci nivele principale în ierarhia EMBOSS:

- nivelul **AJAX:** Este un director care contine biblioteci având functii de nivel scazut responsabile pentru: citirea si scrierea de secvente, gestionarea fisierelor, a sirurilor de caractere, a memoriei, a operatiilor matematice, etc.

- nivelul **NUCLEUS:** Acest director contine biblioteci având functii de nivel superior precum si algoritmi specifici biologiei moleculare: aliniere, restrictii, corespondenta între tipare, etc.

- nivelul **PLPLOT**: În directorul PLPLOT sunt înregistrate rutine pentru generarea fisierelor de iesire de tip grafic.

- nivelul **EMBOSS**: Acest nivel contine ansamblul de programe sau aplicatii. Aici gasim codurile sursa ale programelor, un subdirector în care sunt salvate datele si un subdirector în care sunt fisierele acd corespunzatoare aplicatiilor.

- nivelul **DOC**: Acest nivel contine documente cu format HTML care coincid cu documentatia accesibila pe Internet.

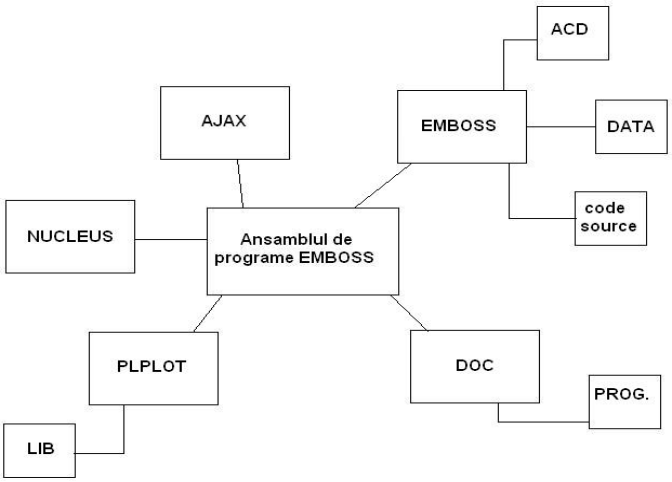

Fig. 2.6. Structura ansamblului de programe EMBOSS: directoare si subdirectoare

2.1.3.2. Descrierea procesului de lansare în executie: comanda si procesele implicate

Pentru a lansa în executie un program bioinformatic din ansamblul EMBOSS este suficienta introducerea în linia de comanda a numelui aplicatiei.

Exemplu: **% gcf**

Obs.: Apelul programului poate fi însotit si de alti calificatori care specifica formatul si numele fisierului de intrare si/ sau iesire sau de calificatori globali care determina comportamentul programului.

Orice program bioinformatic EMBOSS este descris cu ajutorul a doua fisiere: un fisier aplicatie care contine codul sursa scris în limbajul de programare C (nume_program.c) si un fisier ACD în care sunt definite caracteristicile si parametrii programului (nume_program.acd).

a. <u>Pasii necesari pentru lansarea în executie a unei aplicatii bioinformatice</u>

Pentru a lansa în executie o aplicatie bioinformatica EMBOSS trebuie urmate etapele descrise mai jos:

1. Scrierea continutului fisierului gcf.acd:

```
appl: gcf [
  doc: "Work out GC fraction"
  groups: "DNA: sequence composition"
]

sequence: green [
  param:Y
  type: DNA
]

outfile: boggo [
  param: Y
]
```

2. Scrierea continutului fisierului sursa gcf.c:

```
#include "emboss.h"

int main(int argc, char **argv)
{
   AjPSeq seq=NULL;
   AjPStr str=NULL;
   AjPFile outf=NULL;
   char *p;
```

```
    ajint  len;
    ajint  count=0;
    ajint  begin;
    ajint  end;
    ajint  c=0;

    embInit("gcf",argc,argv);

    seq  = ajAcdGetSeq("green");
    outf = ajAcdGetOutfile("boggo");

    str = ajStrNew();

    begin = ajSeqBegin(seq);
    end   = ajSeqEnd(seq);

    p = ajSeqChar(seq);

    ajStrAssSubC(&str,p,--begin,--end);
    ajStrToUpper(&str);
    p = ajStrStr(str);  /* see section 10.2.6 for the "proper" data hiding method */
    len = ajStrLen(str);

    while(*p)
    {
        c = *p++;
        if(c=='G' || c=='C')
          ++count;
    }

    ajFmtPrintF(outf,"GCFraction = %f\n", (float)((float)count/(float)len));

    ajStrDel(&str);
    ajExit();
    return 0;
}
```

3. Editarea fisierului Makefile.am (din directorul EMBOSS)

În fisierul MakeFile.am trebuie adaugat:

- o linie de program prin care **gcf** este definit în **bin_PROGRAMS**
- **gcf_SOURCES = gcf.c**.

Dupa efectuarea modificarilor în fisierul Makefile.am, trebuie introdus în linia de comanda apelul ***make*** si procesul de lansare în executie este încheiat.

b. <u>Explicarea procesului executie a aplicatiei EMBOSS</u>

- **directiva include**
  - **#include "emboss.h"** - realizeaza importul întregii interfete EMBOSS. Aceasta înseamna ca toate bibliotecile EMBOSS sunt disponibile pentru a fi apelate.
- **importul liniei de comanda**
  - Este foarte important ca linia de comanda sa fie disponibila pentru toate aplicatiile, de accea este necesar ca functia **main** sa includa importul acesteia: **int main (int argc, char **argv)**
- **declaratii**
  - **un obiect secventa – AjPSeq** – necesar pentru reprezentarea secventei required (cu caracter obligatoriu) de la intrarea programului bioinformatic;
  - **un obiect fisier – AjPFile** – necesar pentru reprezentarea fisierului required (cu caracter obligatoriu) de la iesirea programului bioinformatic;
  - **un obiect sir de caractere – AjPStr** – pentru diferite prelucrari.

Obs.: 1. Datele de intrare si iesire, secventa respectiv fisierul, sunt reprezentate sub forma de obiecte pentru a putea fi folosite diferite formate, folosind acelasi sistem.

2. Restul declaratiilor sunt declaratii obisnuite pentru limbajul C.

- **procesarea fisierului ACD**
  - **embInit("gcf",argc,argv)** – citeste din baza de date locala definitiile aplicatiei si gaseste fisierul ACD corespunzator pe baza argumentelor argc si argv. În momentul imediat urmator citeste secventa de intrare, pe care o salveaza în memorie si deschide fisierul de iesire pentru salvarea rezultatelor executiei.

Obs.: Daca iesirea programului este de tip grafic, atunci se foloseste linia **ajGraphInit("prog", argc, argv)** în locul embInit.

- **cautarea valorilor ACD**
  - ACD a pozitionat în memorie toate valorile de care are nevoie aplicatia. Aceste valori pot fi gasite cu ajutorul familiei de functii **ajAcdGet**. Astfel functiile embInit si ajAcdGet aloca memoria necesara pentru toti parametrii aplicatiei.
- **construirea unui nou obiect**
  - **Exemplu: str = ajStrNew()** – pentru cazul unui sir de caractere (cazul programului gcf)**.**
- **distrugerea unui obiect**
  - **ajExit()** – în finalul programului, prin aceasta comanda distrugem obiectele create la începutul aplicatiei.

Obs.: În cazul programului gcf doar obiectul de tip sir de caractere trebuie sters.

- **iesirea din program**
  - **return 0** – returneaza valoarea zero deoarece functia main a fost declarata ca **int** (integer).

2.1.3.3. Introducere în ACD

EMBOSS descrie programele bioinformatice utilizând un strat software intermediar între programul executabil (aplicatia scrisa în limbajul de programare C) si interfata pe care o acceseaza utilizatorul. Stratul software suplimentar îl constitue limbajul de programare ACD – Ajax Command Definition, cu rolul de a descrie toate tipurile de programe din cadrul ansamblului în asa fel încât acestea sa poata fi reprezentate într-o interfata unica.

În interiorului unui fisier ACD corespunzator unei aplicatii EMBOSS se gasesc definitia aplicatiei si definitiile parametrilor si calificatorilor acesteia.

- **Privire de ansamblu asupra ACD**

Fisierele ACD sunt simple fisiere text care contin definitii. În general, au aceeasi denumire cu programul pe care îl însotesc; extensia lor este în mod obligatoriu **.acd**.

Exista câteva reguli gramaticale ale limbajului ACD, descrise mai jos:

- comentariile încep cu „#";
- fiecare linie este cuprinsa între etichete, numite „**token**", delimitate de spatii;
- toate valorile sunt privite drept siruri de caractere, pâna în momentul executiei lor;
- toate variabilele dependente de alte valori trebuie sa fie definite dupa definirea valorilor de care depind;
- parametrii si calificatorii sunt definiti prin intermediul unui singur token, urmat de „**:**" si un spatiu, care este urmat de un al doilea token; definirea parametrilor si a calificatorilor trebuie realizata între paranteze drepte"[ ]".

```
token: token [
  definitie;
]
```

Primul token semnifica tipul datei AJAX, care urmeaza sa fie definita, iar al doilea token reprezinta numele parametrului. Între paranteze drepte este definit, în mod obligatoriu, parametrul.

Exista câteva notiuni în limbajul ACD: **aplicatie** sau **program**, **atribute**, **parametri**, **calificatori**. Tipurile de date sunt diferite de datele întâlnite, uzual, în diferitele limbaje de progamare.

2.2. Context tehnic – concepte informatice utilizate

Îndeplinirea obiectivelor proiectului BioSide presupune cunostinte în domeniul bazelor de date, precum si cunoasterea diferitelor limbaje de programare si aplicatii software generale sau destinate bioinformaticii, cum ar fi XML, Java, UML, Eclipse with UML, PostgreSQL, XMLSpy, BioSide, BioDesc.

Eclipse with UML este un editor de programe Java. Întreg proiectul BioSide este conceput în Java, folosind arhitectura MVC (Model-View-Controller). Astfel orice interventie asupra interfetelor sau bazelor de date este posibila prin scrierea si integrarea codului Java corespunzator. Diagramele UML (Unified Modeling Language) sunt de o reala importanta, mai ales pentru vizualizarea unei structuri clare a claselor si a relatiilor dintre acestea.

La fel ca si Eclipse, ***PostgreSQL,*** este o unealta informatica. Aceasta pune la dispozitia utilizatorului o serie de facilitati în lucrul cu bazele de date: o interfata ergonomica, accesarea simpla a bazelor de date, interogarea structurata si rapida a acestora, etc.

XMLSpy reprezinta o aplicatie software complexa. Printre diversele functionalitati exista posibilitatea de a verifica corectitudinea formei unui fisier XML sau validitatea fisierului în raport cu reprezentarea descriptiva (în fisierul DTD sau XML Schema) la care se face referinta în antetul sau.

2.2.1. Introducere în baze de date relationale

Principiile modelului relational si bazele de date relationale au fost pentru prima data prezentate de matematicianul Dr. E. F.Codd, de la centrul de cercetari al IBM, în iunie 1970, când a publicat un articol numit "Un model relational pentru marile banci de date" (în original *A Relational Model of Data for Large Shared Databanks*). În respectivul articol, se propune modelul *relational* pentru sistemele de baze de date. Date economice, cataloage ale bibliotecilor, fisiere de personal au fost manipulate si prelucrate chiar si înaintea propunerii matematicianului, însa într-un mod mai putin formalizat, neunitar; este vorba de perioada când portabilitatea

aplicatiilor de pe un sistem de calcul pe altul era înca un proiect, când majoritatea aplicatiilor care lucrau cu seturi mari de date se programau în limbaje precum COBOL, Fortran, PL/1, Algol.

Un exemplu de utilizare a Bazelor de Date Relationale îl constitue Internetul. În cazul Internetului vorbim despre Baze de Date Distribuite, plasate pe mai multe servere si accesibile de pe diferite statii de la distanta. Este vorba de un nivel mai ridicat de complexitate al gestiunii informatiilor si al aplicatiilor specifice, dar, în majoritatea cazurilor este vorba de Baze de Date Relationale. În plus, cautarile în World Wide Web, indiferent care este motorul de cautare folosit, nu sunt decât cautari în bazele de date construite de aceste motoare de cautare cu datele colectate în procesul de investigare al paginilor Web accesibile. Diferentele pe care, eventual, le sesizam în raspunsurile diferitelor motoare de cautare depind tocmai de modul în care sunt construite si întretinute respectivele baze de date.

Baza de date relationala este perceputa de utilizatorii sai ca o colectie de tabele, tablouri bidimensionale de date, numite si relatii.

Conceptele bazelor de date relationale:

1. **tabelele**
2. **coloanele tabelului**
3. **rândurile tabelului**
4. **câmpurile din tabel, de pe o anumita coloana si linie.**

Exemplu:

Tabel Clienti

NrS.	Nume	Stare	Oras
S1	Ionescu	20	Londra
S2	Popescu	10	Paris
S3	Teodorescu	30	Paris

rând→

↑ coloana

Tabel Parti

NrP.	NumP	Culoare	Greutate	Oras
P1	Nit	Rosu	12	Londra
P2	Piron	Verde	17	Paris
P3	Surub	Albastru	17	Roma
P4	Surub	Rosu	14	Londra

Tabel Ordini

NrS.	NrP.	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S2	P1	300
S2	P2	400
S3	P3	200

***liniile si coloanele sunt completate de câmpuri

Modelul relational al bazei de date imita procesele unei ramuri a algebrei, cunoscute sub numele de "Algebra relationala". Aceste procese implica:

- o colectie de obiecte cunoscute sub numele de *RELATII*
- o multime de operatori care actioneaza asupra relatiilor pentru a produce noi relatii.

O relatie, în cazul discret, poate fi privita si înteleasa ca o Tabela. Modificarea datelor se realizeaza prin operatii relationale aplicate asupra tabelelor. Spre exemplu, **restrictia** unei relatii, a unui tabel, va fi un nou tabel în care se pastreaza doar anumite linii. Iar **proiectia** unei relatii - unui tabel - va fi un nou tabel în care se pastreaza doar anumite coloane.

Operatorii relationali

Operatorii relationali, în cadrul bazelor de date relationale, sunt definiti în tabelul urmator:

Operatorul Relational	Descrierea
Restrictia	Este o operatie care preia si afiseaza datele din relatie.Este posibil sa se afiseze toate rândurile sau doar rândurile care îndeplinesc o anume conditie (sau mai multe conditii). Aceasta este de multe ori numita "submultime orizontala"
Proiectia	Este operatia care afiseaza anumite coloane din relatie, fiind numita de aceea si "submultime verticala".
Produsul	Este rezultatul obtinut când rândurile a doua multimi de date sunt concatenate conform conditiilor specificate.
"Join"	Este rezultatul obtinut când rândurile a doua multimi de date sunt concatenate conform conditiilor specificate.
Reuniunea	Afiseaza toate rândurile care apar în una, în cealalta, sau în ambele relatii.
Intersectia	Afiseaza toate rândurile care apar în ambele relatii.
Diferenta	Afiseaza rândurile care apar numai în prima relatie fara sa apara în cea de a doua (în SQL se utilizeaza chiar semnul - operatorul *minus*).

Tabel 2.1. Operatorii relationali în SQL

Proprietati ale bazelor de date relationale:

- O baza de date relationala apare ca o **colectie de relatii** (tabele).
- Exista o multime de **operatori** pentru transformarea si combinarea relatiilor:
  - selectia,
  - proiectia
  - produsul
  - join-ul
  - reuniunea
  - intersectia
  - diferenta

- Nu apar pointeri; **conexiunile sunt facute numai pe baza datelor**.
- Exista o **independenta totala a datelor**.
- Limbajul utilizat pentru interogarea bazei de date este **non-procedural** si similar limbii engleze.
- Utilizatorul nu specifica accesul si nu are nevoie sa stie cum este **structurata fizic informatia**.
- Comenzile pentru selectia sau refacerea datelor, cât si acelea pentru realizarea schimbarilor în baza de date sunt incluse într-un singur limbaj, standardizat acum ca **SQL**.

Proprietatile relatiilor tabelare:

Fiecare tabela, individual, are urmatoarele proprietati:

- Nu exista rânduri duplicate
- Nu exista nume de coloane identice (duplicate)
- Ordinea rândurilor este neimportanta
- Ordinea coloanelor este neimportanta
- Valorile (câmpurile) sunt atomice (nedecompozabile).

SQL

- este limbajul utilizat pentru a accesa o baza de date relationala;
- poate fi utilizat atunci când accesul la baza de date este necesar, de fiecare instrument de gestiune a bazelor de date (Access, Oracle, SQL-Server, MySQL,...);
- este un limbaj simplu, ne-procedural, cu comenzi intuitive în limba engleza;
- este un limbaj standard pentru bazele de date.

Vorbind despre SQL, în multe carti se spune SQL = Structured Querry Language. Dar SQL nu este un limbaj structurat în sensul structurarii limbajelor de programare. De fapt, curând dupa aparitia lucrarii lui Codd, la IBM s-a realizat un sistem prototip pentru baze de date relationale. S-a numit *System R* si se baza pe un set de comenzi catre baza de date, set de comenzi

numit SEQUEL (*Structured English QUEry Language*). Cu alte cuvinte un subset de *cereri* catre baza de date formulate strcturat în engleza. A fost punctul de plecare pentru SQL, care a fost standardizat ca limbaj pentru bazele de date, independent de sistemul de gestiune si de platforma de calcul. Ca si consecinta, numele limbajului a fost prescurtat, pentru a evita confuzii cu System R si cu vechiul limbaj SEQUEL.

O prima standardizare a SQL s-a petrecut înca din 1986, când au aparut specificatiile ANSI (American National Standards Institute), extinse în standardizarea ISO din 1989, apoi în 1992 si înca mai recent, în 1999.

Simplitatea limbajului se reflecta în numarul mic de comenzi (zece) si în faptul ca nici dimensiunea bazei de date, numarul de înregistrari, sau numarul de coloane, nici tipul datelor nu influenteaza forma comenzilor SQL. Ca urmare, toate sistemele actuale de gestiune de baze de date accepta comenzi în limbajul SQL.

În sistemele Oracle de baze de date, pentru scrierea aplicatiilor si pentru manipularea datelor în afara bazei de date, se foloseste un limbaj procedural numit *PL/SQL*, o extensie procedurala a limbajului SQL, de fapt principalul limbaj (procedural) pentru realizarea aplicatiilor în sistemele Oracle . In particular, *SQL-Plus* este un produs Oracle în care pot fi utilizate limbajele SQL si PL/SQL.

Vom vedea ca, în sistemul *MySQL* (de fapt în pachetul PHP, MySQL, Apache Web server) exista functii si rutine speciale destinate scrierii unor programe cu comenzi SQL, adica înglobarii SQL într-o secventa procedurala structurata.

2.2.2. Introducere în XML

Limbajul XML a fost pus la punct de XML Working Group sub egida World Wide Web Consortium (W3C), începând cu anul 1996. A fost recunoscut ca si limbaj, în momentul în care W3C a lansat specificatiile pentru XML 1.0, la data 10 februarie 1998.

XML este un acronim si provine de la eXtensible Markup Language, adica un limbaj având tag-uri extensibile. Un astfel de limbaj se caracterizeaza prin combinarea sectiunilor de

text si a informatiilor extra privind sectiunile de text. Cel mai cunoscut exemplu de limbaj „markup” este HTML (Hyper Text Markup Language).

XML reprezinta un subset al SGML (Standard Generalized Markup Language), fiind definit astfel încât sa fie cât mai lizibil. XML poate fi considerat o versiune îmbunatatita a limbajului HTML, care permite definirea de noi tag-uri pentru a descrie documentele. Spre deosebire de HTML, care este caracterizat ca un limbaj definit si solidificat, având un numar limitat de tag-uri, XML poate fi considerat un metalimbaj ce permite definirea altor limbaje prin introducerea de noi tag-uri, specifice pentru un orice tip de document. O alta deosebire semnificativa dintre XML si HTML este faptul ca XML reprezinta un format de descriere a datelor si nu modalitatea de reprezentare a acestora, cum este cazul HTML.

Forta pe care o are XML rezida în capacitatea de a putea descrie date provenind din orice domeniu datorita extensibilitatii sale; acest limbaj permite structurarea, inventarea vocabularului si sitaxei datelor pe care le va contine documentul.

Asadar XML este un limbaj ce utilizeaza tag-uri care nu sunt predefinite, ci utilizatorul este cel care le scrie în acord cu continutul documentului pe care doreste sa îl conceapa. Ïn plus, XML utilizeaza Document Type Definition (DTD) sau XML Schema pentru a defini tag-urile proprii.

Sintaxa XML:

Structura fisierelor XML comporta trei sectiuni:

- **elementele** (Ex: note, date, day, etc.)
  - sunt etichete (tag-uri) XML care structureaza documentul de o maniera ierarhica;
  - structura trebuie întocmai respecatata.
- **atributele** (Ex: <gangster name=’George „Shotgun” Ziegler>)
  - reprezinta informatiile complementare asociate unui tip de element;
  - permit specificarea unui anumit numar de caracteristici proprii elementului curent, pe care îl descriu (adica este vorba de o eticheta care nu contine doar numele elementului ci mai contine alte informatii caracteristice ale acestuia).

- **textul**
  - textele sunt noduri terminale în structura XML;
  - aceasta sectiune este optionala.
- **componente optionale** (<!--We can have more than one person on the „to” section -->)
  - pot fi: comentarii, un prolog care face referire de fisierul descriptiv DTD, instructiuni de tratare a fisierului XML – transformarile pe care le poate suferi.

Exemplu:

```
<?xml version="1.0"?>
<note>
<date>
   <day>12</day>
   <month>11</month>
   <year>99</year>
</date>
<to>Tove</to>
<!--We can have more than one person on the „to” section -->
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

Utilitatile XML:

- ***XML este capabil sa pastreze separate datele de codul HTML.***

Atunci când utilizam HTML pentru a afisa date, acestea sunt salvate în interiorul fisierului HTML. Utilizând XML, datele pot fi salvate în mod separat în fisiere de tip

XML. Astfel, chiar daca intervin modificari în ceea ce priveste continutul datelor, fisierul HTML (modalitatea de afisare a acestora) nu va fi afectat.

- ***Stocând date în format XML, acestea pot fi utilizate în acelasi timp de catre aplicatii diferite.***

  Acest fapt usureaza extinderea si trecerea la noi versiuni ale sistemelor de operare, serverelor, aplicatiilor si navigatoarelor web.

- ***Un rol important îl reprezinta posibilitatea de a schimba formatul datelor cu ajutorul limbajului XML***

  Sistemele de calcul si bazele de date contin date având formate incompatibile. Un mare efort îl constitue schimbarea formatului datelor provenind de la sistemele de calcul sau din bazele de date, într-un format specific pentru Internet. Solutia este convertirea acestora în forma XML, acesta fiind capabil sa genereze date într-un format ce poate fi usor interpretat de o multime de aplicatii diferite.

Validarea unui fisier XML:

Validitatea unui fisier XML este determinata de: corectitudinea formei acestuia (well formed) si validitatea sa în raport cu un fisier DTD (Document Type Definition) sau cu un fisier XML Schema, asociat acestuia.

Corectitudinea formei unui fisier XML este verificata în raport cu regulile sintaxei de scriere a unui astfel de fisier, stabilite de echipa W3C.

Rolul unui fisier DTD este de a defini gramatica unei clase de fisiere XML: structura cu elemente si atribute. În antetul documentului XML este specificat numele fisierului DTD, în raport cu care se verifica validitatea sa. În cazul în care documentul XML specifica un fisier DTD, fara sa aiba o structura corespunzatoare lui, utilizatorului îi va fi semnalat un mesaj de eroare.

XML Schema are un rol asemanator fisierului DTD, doar ca într-un fisier XML Schema putem descrie cu mai mare precizie elementele, atributele, valorile posibile, constrângerile de sintaxa, structurale si ale tipurilor, pentru o clasa de fisiere XML.

Operatiile de verificare a corectitudinii formei si a validitatii pentru fisierele XML, pot fi realizate folosind diferite unelte informatice (Ex.: XMLSpy).

2.2.3. Prezentarea arhitecturii MVC

Model-View-Controller (MVC) este un model de design (pattern) care leaga eficient interfata cu utilizatorul de modelul de date, în programarea orientata pe obiecte. Acesta arhitectura este larg utilizata în programarea în limbajele Java, C++ sau Smalltalk, permitând reutilizarea codului sursa si reducând astfel durata de dezvoltare a aplicatiilor cu interfete utilizator.

Arhitectura model-view-controller este alcatuita din trei componente principale:

- componenta ***Model***, reprezentata de structura logica de date a aplicatiei si clasele de nivel înalt asociate cu ea. Componenta Model nu contine informatii despre interfata utilizator.
- componenta ***View***, care este o colectie de clase reprezentând interfata cu utilizatorul - toate obiectele pe care utilizatorul le poate vedea pe ecran si cu care poate interactiona, cum ar fi butoane, casete de text, etc.
- componenta ***Controller***, este constituita din clasele ce realizeaza comunicarea între clasele din Model si cele din View.

Capitolul 3. Analiza si dezvoltare în BioDesc

3.1. Analiza BioDesc

BioDesc este o unealta dedicata descrierii resurselor bioinformatice. Este compus dintr-o baza de date si o interfata prietenoasa permitând actualizarea si importul / exportul în format XML. Descrierile furnizate de BioDesc sunt utilizate de BioSide pentru gestionarea dinamica a interfetelor de descriere a parametrilor si pentru generarea codurilor de executie ale programelor utilizate în BioSide.

Tinând cont de aceasta legatura speciala cu BioSide, câteva caracteristici ale BioDesc reflecta, în mod natural, o parte din preocuparile conceptuale ale BioSide. În acest sens, în continuare pot fi observate doua dintre principalele aspecte:

- **notiunea de identificator al programului**

  În bioinformatica si nu numai este încetatenita mentionarea unui program, în functie de numele sau, fara a tine cont de versiunea acestuia, care a evoluat în mod regulat de-a lungul timpului, sau de diferitele aplicatii executabile asociate unui cod unic al programului (de exemplu o versiune MacOS X si Windows pot genera rezultate usor diferite). Din aceste cauze, obiectivul BioSide în materie de acces cooperativ la resursele informatice cuprinde posibilitatea de duplicare a acestor resurse pe calculatoare diferite. În ceea ce priveste urmarirea executiilor resurselor duplicate este necesar sa existe un identificator care sa defineasca programul si descrierea sa în cadrul BioDesc.

- **descrierea programelor bioinformatice**

  În BioDesc descrierea programelor este structurata pe baza unei interfete în program si parametrii. Aceasta structurare este comuna diferitelor gramatici de descriere a programelor bioinformatice, acestea fiind utilizate pentru invocarea programului final.

a. Prezentare generala a interfetei BioDesc

Interfata Biodesc este structurata dupa cum indica figura 3.1.

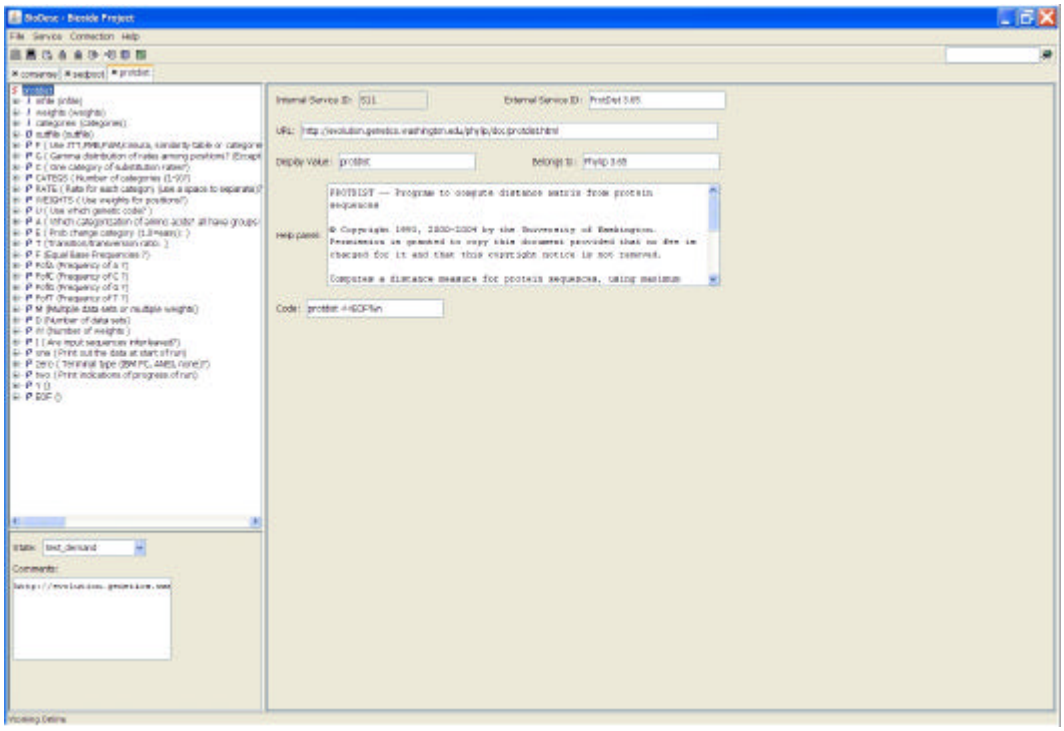

Fig. 3.1. Interfata BioDesc

În partea superioara exista o bara de meniu si icoane care permit conectarea / deconectarea la, respectiv de la, baza de date, precum si încarcarea, salvarea, importarea sau exportarea unei aplicatii (sau program bioinformatic).

Partea principala a interfetei o constitue sectiunea plasata sub bara de meniu si icoane, care este împartita în ferestre. Fiecare fereastra, corespunzatoare unui program bioinformatic, este împartita în trei zone:

- Prima zona, numita în continuare ***zona A***, este situata în stânga sus. Ea contine lista parametrilor si permite expandarea acestora într-un mod similar explorarii fisierelor. În urma acestei actiuni, devine vizibila descrierea fiecarui parametru si se pot accesa tipurile sale.

- A doua zona, numita în continuare ***zona B***, este pozitionata în stânga jos si indica nivelul de descriere a unui parametru: în curs, testare, produs. Tot aici sunt furnizate comentariile privind acest nivel de descriere.

- A treia zona se afla în partea dreapta a interfetei. Aceasta descrie programul, parametrul sau tipul selectat în zona A. În figura 3.1. este încarcat programul **protdist** si este selectat numele programului. Ca urmare, în partea dreapta a interfetei este afisata descrierea programului.

b. Analiza BioDesc

- **Aplicatia sau programul**
  - ***Semantica***

Programul bioinformatic este identificat prin numele si versiunea sa (Ex.: ProtDist 3.65) si este independent de sistemul de operare. Descrierea programelor bioinformatice se îmbogateste pe masura trecerii timpului conducând la aparitia noilor versiuni. Datorita acestui fenomen, reprezentarea BioDesc a unui program depinde de versiunea acestuia.

Un program primeste, în BioDesc, un identificator intern care specifica serviciul pe care programul îl efectueaza (Ex: S11).

În BioDesc este disponibila o scurta descriere a fiecarui program dar si calea spre documentatia completa, oferita de producatori.

Programul este asociat ansamblului din care face parte si poate fi apelat prin generarea codului executabil de apelare, pornind de la secventa de cod afisata în câmpul **code**.

  - ***Sintaxa***

În figura 3.1. este prezentata descrierea programului bioinformatic protdist. În continuare, zona din dreapta interfetei, care descrie caracteristicile programului, având culoarea gri, va fi numita ***zona C***.

În tabelul 3.1. este prezentata descrierea BioDesc a programului **protdist**, sub apectul interfetei si al bazei de date.

Obs.: În tabel sunt prezentate caracteristicile din zona C, iar pentru atributele care nu se afla în aceasta zona, se precizeaza pozitia în interfata BioDesc.

Denumirea în BioDesc	**Denumirea în baza de date**	**Exemplu (protdist)**
Display Value	name (varchar)	protdist
Internal Service ID	id (varchar)	S11
External Service ID	id_ext (varchar)	ProtDist 3.65
Belongs to	belongs_to (varchar)	Phylip 3.65
Help panel	description (varchar)	PROTDIST---program to compute distances matrix from protein sequences...
Code	code (varchar)	protdist<<EOF%n
URL	url (varchar)	http://evolution.genetics.washington.edu/phylip/doc/protdist.html
State (zona B)	state (varchar)	test_demand
Comments (zona B)	comments (varchar)	http://evolution.genetics.was

Tabel 3.1. Sintaxa aplicatiei (programului)

- **Parametrii**
  - ***Semantica***

Obs.: Descrierea parametrilor, din punctul de vedere al interfetei, urmeaza o abordare similara. În continuare, zonele A si B sunt zonele definite mai sus, iar zona de descriere a parametrilor, suprapusa partial peste zona C, va fi numita **zona D** (culoare albastra).

În figura 3.2. este prezentata descrierea programului bioinformatic **protdist** si detaliile unuia dintre parametrii acestuia:

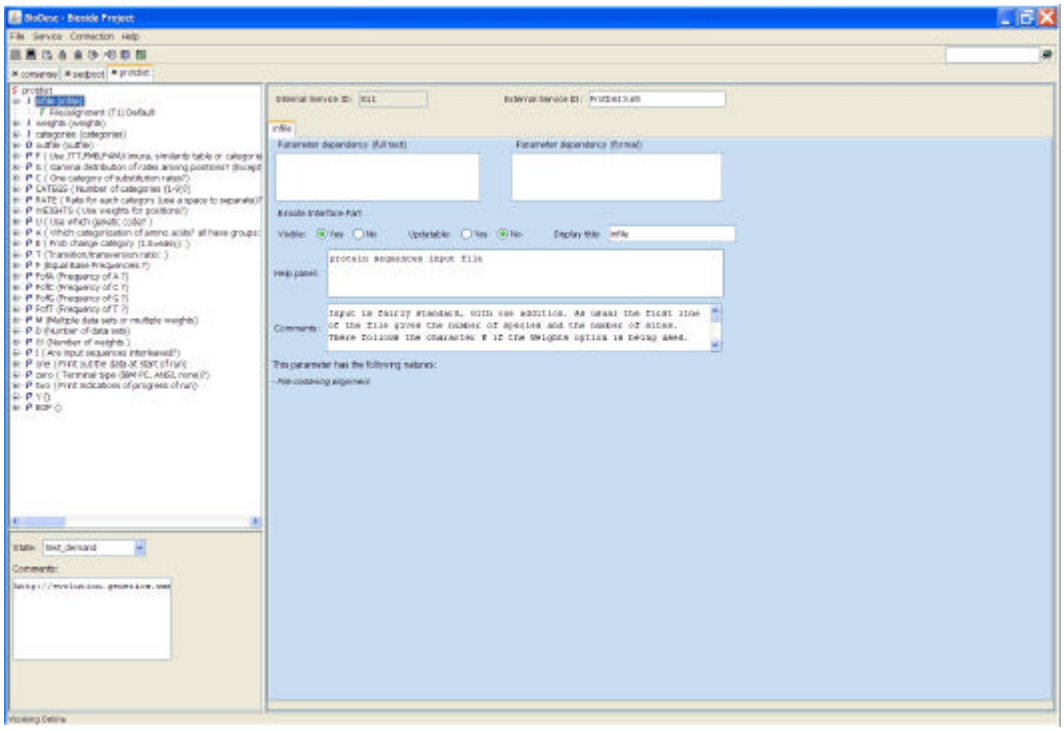

Fig. 3.2. Interfata parametrilor

În cadrul BioDesc, fiecare parametru este definit prin intermediul unui nume, care îl va reprezenta în interfata BioSide si al unui identificator. Explorând, în zona A, descrierea parametrilor, în cadrul interfetei vor fi afisate tipurile corespunzatoare fiecarui parametru. În figura 3.2. este exemplificat parametrul având identificatorul **I**, numele **infile (infile)** si tipul **File/alignment**.

Exista trei tipuri de parametri:

- parametrii **input** – descriu intrarile programului,
- parametrii **output** – descriu iesirile programului,
- parametrii **regular** – sunt cei care exprima comportamentul progamului.

Selectând, în zona A a interfetei, unul dintre parametri, în zona D se poate observa descrierea caracteristicilor sale: help, comentarii, etc. Câmpurile care exprima **dependentele textuala si formala** permit exprimarea legaturilor de dependenta dintre

parametri. De exemplu, parametrul Rate, nu are sens decât daca parametrul care are identificatorul C este inactiv (figura 3.3.).

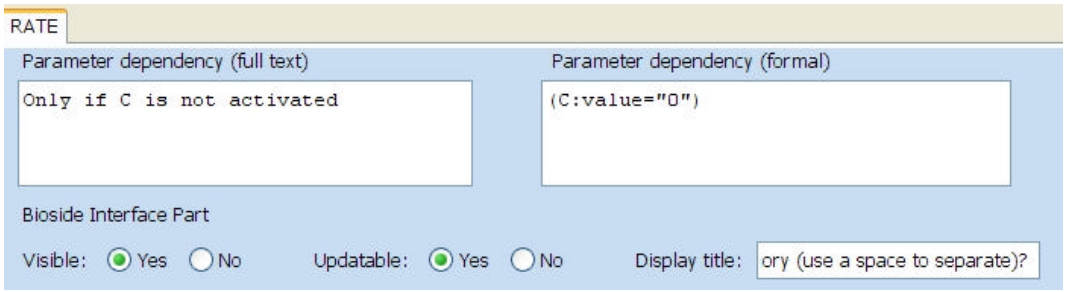

Fig. 3.3. Ilustrarea caracteristicilor unui parametru

Atributul **visible** indica daca parametrul este sau nu accesibil în interfata BioSide de descriere a parametrilor. Exista parametri pentru care afisarea în interfata BioSide nu prezinta interes, de exemplu, parametrul care descrie tipul terminalului.

Atributul **updatable** indica daca valoarea parametrului poate sau nu sa fie modificata prin intermediul interfetei BioSide de descriere a parametrilor si se aplica parametrilor „regular”; pentru parametrii „input” si „output”, redirectarea si numirea sunt gestionate intern de catre BioSide.

- ***Sintaxa***

În tabelul 3.2. este descrisa sintaxa de definire a parametrilor, în cadrul BioDesc.

Obs.: În tabel sunt prezentate caracteristicile din zona D, iar pentru atributele care nu se afla în aceasta zona, se precizeaza pozitia în interfata BioDesc.

Denumirea în BioDesc	Denumirea în baza de date	Exemplu (parametrul infile al prog. protdist)
În zona A - parametri	id (varchar)	I
În zona C - program	service_id (varchar)	S11
Parameter dependency (full text)	bio_dependences (varchar)	---
Parameter dependency (formal)	dependences (varchar)	---
Visible	visible (boolean)	„Yes”(true)
Updatable	updatable (boolean)	„No”(false)
Display title	name (varchar)	infile
Help panel	description (varchar)	Protein sequence input file
Comments	comments (varchar)	Input is fairly standard...

Tabel 3.2. Sintaxa parametrilor

Obs.: În baza de date mai exista câmpurile **parameter_status, position** si **order** care nu au corespondent în interfata BioDesc. *Parameter_status* este dedus prin procedee informatice, iar câmpurile *position* si *order* au rol în gestiunea afisarii, în cadrul interfetei BioSide.

- **Tipurile**
  - ***Semantica***

Obs.: Caracteristicile tipurilor sunt afisate în partea dreapta a interfetei, în **zona E** (culoare verde), partial suprapusa peste zona D (fig 3.4.).

Un parametru este caracterizat de mai multe tipuri diferite, dar poate fi asociat cu un singur tip la un moment dat. În cazul parametrilor multipli, caracterizati de o lista de tipuri, unul dintre tipuri este setat ca valoare implicita a parametrului.

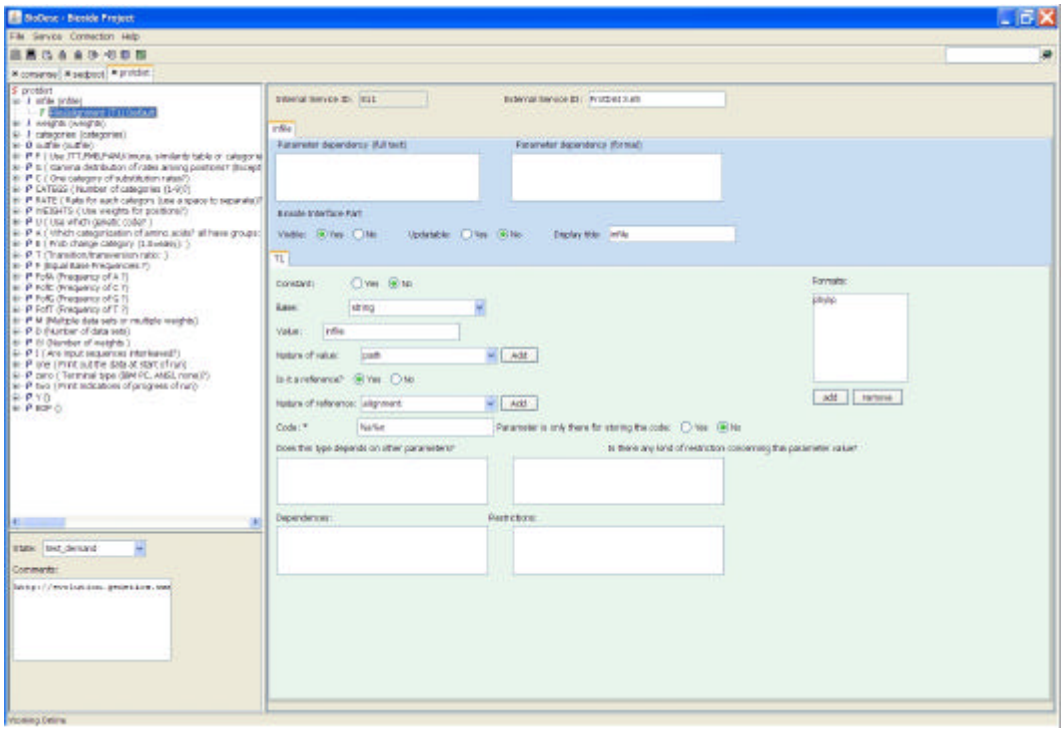

Fig. 3.4. Interfata tipurilor

Zona E, de descriere a tipurilor, este, fara îndoiala, cea mai dificil de manipulat. În figura 3.4. pot fi observate caracteristicile tipului **File/ alignment** corespunzator parametrului **infile** din cadrul programului bioinformatic **protdist**:

➔ **constant** este o caracteristica utilizata pentru a indica daca un parametru are sau nu o valoare variabila. Un tip constant nu prezinta interes decât în cazul parametrilor multipli. Este vorba, în majoritatea cazurilor, de o lista de tipuri constante – o lista de valori din care utilizatorul alege valoarea dorita.

➔ **base** este câmpul care descrie tipul de baza al valorii. Putem alege dintre tipurile clasice precum: sir de caractere (string), întreg (integer), real (float) si boolean.

➔ **value** contine valoarea implicita a parametrului.

➔ **nature of value** este câmpul care descrie semantica valorii indicate. În exemplul din figura 3.4. este vorba de „path” care ne indica faptul ca tipul pe care îl descrie este un fisier.

➔ **is it a reference?** indica daca valoarea tipului este o valoare terminala sau de redirectare.

➔ în cazul în care raspunsul la întrebarea „is it a reference” este afirmativ, câmpul **nature of value** devine activ si permite alegerea naturii datelor referite dintr-o lista de valori posibile.

➔ în mod similar, acelasi raspuns ca si mai sus, determina activarea câmpului **format**, permitând alegerea unui format dintr-o lista de formate posibile.

➔ **parameter is only for storing the code?** În BioDesc exista câtiva parametri care nu prezinta interes din punct de vedere bioinformatic, ci sunt definiti doar ca instrumente informatice cu rol de gestiune (Ex.: Y, EOF care sunt utilizate pentru sesizarea asocierii parametri – tipuri aleasa de catre utilizator, respectiv pentru sesizarea încheierii parcurgerii fisierului de intrare al programului).

➔ **dependences** si **restrictions**, în reprezentare textuala si formala, exprima conditii de dependenta si de restrictie a tipurilor parametrilor.

- ***Sintaxa***

  În tabelul 3.3. este descrisa sintaxa de definire a tipurilor, în cadrul BioDesc.

Obs.: În tabel sunt prezentate caracteristicile din zona E, iar pentru atributele care nu se afla în aceasta zona, se precizeaza pozitia în interfata BioDesc.

Denumirea în BioDesc	Denumirea în baza de date	Exemplu (tipul File/alignment al parametrului infile din prog. protdist)
În zona A - tipuri	id (varchar)	T
În zona C - program	parameter_service_id (varchar)	S11
În zona D - parametri	parameter_id (varchar)	I
Constant	constant (boolean)	„No” (false)
Base	base (varchar)	string
Value	value (varchar)	infile
Nature of value	nature_of_value (varchar)	path
Nature of reference	nature_of_reference (varchar)	alignment
Code	code (varchar)	%s%n
Parameter is there only for storing the code	isCode (boolean)	„No” (false)
Does this depends on other parameters	bio_dependences (varchar)	---
Dependences	dependences (varchar)	---
Is there any kind of restriction concerning this parameter value?	bio_restrictions (varchar)	---
Restrictions	restrictions (varchar)	---
Format	format (varchar)	Types_has_formats
Default	default_type (boolean)	true

Tabel 3.3. Sintaxa de definire a tipurilor

c. <u>Baza de date BioDesc</u>

În cadrul BioDesc, datele sunt salvate într-o baza de date relationala, formata din doisprezece tablouri. În figura 3.5. poate fi observata diagrama UML a bazei de date.

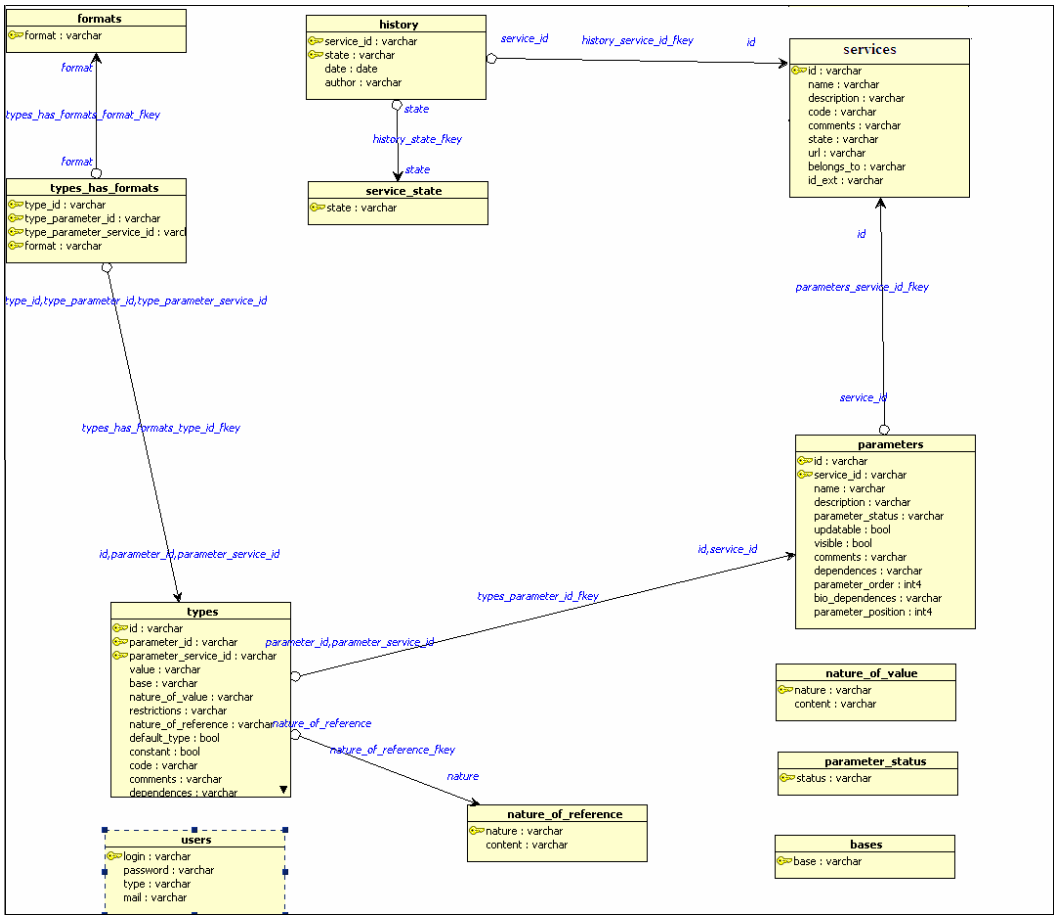

Fig. 3.5. Diagrama UML a bazei de date BioDesc

Fiecare tablou este reprezentat sub forma unei clase de variabile, corespunzatoare câmpurilor din baza de date. Sagetile pun în evidenta relatiile care se stabilesc între tablourile bazei de date precizând cheile de indexare si numele variabilelor implicate în relatie (Ex: **nature_of_reference** din tabloul *types* este preluat din câmpul **nature** al tabloului *nature_ of_reference* având cheia de indexare **nature_of_reference_fkey**).

3.2. Implementarea versiunii V2bis de import – export

În ceea ce priveste lucrul cu aplicatiile bioinformatice, BioDesc pune la dispozitia utilizatorilor, prin intermediul interfetei, facilitatile de încarcare, salvare, import si export.

În ultimii ani, datorita cresterii complexitatii Internetului, a aparut nevoia de a separa descrierile web de datele propriu-zise. O solutie flexibila si simpla o reprezinta XML (eXtensible Markup Language). Ceea ce face ca XML sa fie din ce în ce mai mult utilizat în bioinformatica si nu numai este faptul ca documentele XML pot fi convertite în diferite formate: HTML, postscript, PDF, Scalable Vector Graphics (SVG) si altele.

Importul presupune extragerea datelor cuprinse într-un fisier XML, urmata de adaugarea lor corecta în baza de date BioDesc. Pentru a asigura importul riguros al acestora, în conformitate cu structura bazei de date, fisierul XML trebuie sa fie bine format si sa fie valid în raport cu un fisier de tip Document Type Definition, care impune o structura identica cu cea din BioDesc.

Operatia de **export** reprezinta facilitatea de transfer complet al datelor unei aplicatii bioinformatice din baza de date BioDesc într-un fisier XML, respectând aceeasi structura: program, parametri, tipuri. Structura fisierului XML rezultat în urma exportului este generata pornind de la un fisier de tip Document Type Definition, care, la rândul sau, trebuie definit în concordanta cu structura bazei de date.

În BioDesc mai exista trei variante de import si doua variante de export, dar care nu sunt adaptate nivelului actual de dezvoltare al bazei de date. În urma unei analize am observat ca exista pierderi de informatii atât la efectuarea operatiei de import cât si în urma exportului. Acesta este motivul care a stat la baza implementarii unei noi versiuni de import – export între formatul XML si baza de date BioDesc.

3.2.1. Generarea fisierului Document Type Definition (DTD)

Un fisier de tip Document Type Definition (DTD) contine definitiile tag-urilor (etichetelor specifice limbajelor „markup”) si modul de structurare pe care trebuie sa îl respecte o

anumita categorie de fisiere XML. Cu alte cuvinte, rolul unui astfel de fisier este de a specifica gramatica asociata unei categorii de fisiere XML.

Fisierele XML, care sunt generate pornind de la un anumit DTD, trebuie sa respecte regulile de definire si structurare specificate de acesta, caz în care spunem ca fisierele XML sunt valide (în raport cu DTD-ul corespunzator).

Atunci când utilizam formatul XML pentru transferul de informatii este absolut necesar sa asiguram concordanta dintre datele primite si formatul asteptat. Este tocmai cazul importului din BioDesc. Importul este corect realizat doar daca fisierul XML si baza de date, în care urmeaza sa fie importat, au aceeasi structura. Pentru a realiza un export complet, trebuie sa utilizam fisierul DTD care defineste o structura identica cu cea a bazei de date.

În concluzie, pentru implementarea facilitatilor de import – export între fisiere XML si baza de date BioDesc, este necesara definirea unui DTD care impune fisierelor XML o structura identica cu cea din BioDesc.

- **Nivelul program (aplicatie)**

  În cadrul BioDesc, descrierea programului coincide cu descrierea serviciului oferit prin utilizarea sa.

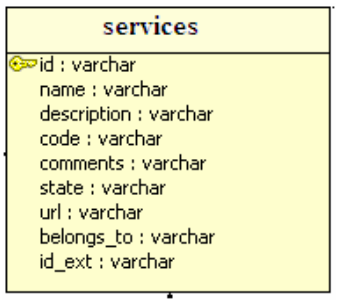

Fig. 3.6. Diagrama descriptiva a serviciilor (programelor)

În figura 3.5., în care este reprezentata diagrama UML a bazei de date BioDesc, se poate observa ca între tablourile **services** si **parameters** exista o relatie directa. Aceasta exprima faptul ca un serviciu sau program este caracterizat de o serie de parametri cu ajutorul carora îi este descris comportamentul.

Astfel, putem observa ca exista doua nivele de descriere a unui program bioinformatic: nivelul de definire si identificare si nivelul de descriere a comportamentului.

Definirea tag-urilor aplicatiei în fisierul DTD:

*<!ELEMENT **program** (name_s?, ext_id, belongs_to, description_s, code_s, comments_s?, state, url?, parameters)>*
<!ATTLIST program
id %integer; #REQUIRED
>
<!-- id este un atribut care va fi inclus în tag-ul program -->

<!ELEMENT name_s (#PCDATA)>
<!ELEMENT ext_id (#PCDATA)>
<!ELEMENT belongs_to (#PCDATA)>
<!ELEMENT description_s (#PCDATA)>
<!ELEMENT code_s (#PCDATA)>
<!ELEMENT comments_s (#PCDATA)>
<!ELEMENT state (#PCDATA)>
<!ELEMENT url (#PCDATA)>

<!-- ..... -->
<!-- Fiecare program contine cel putin un parametru -->
<!-- ..... -->
<!ELEMENT parameters (parameter)+>

Exemplu – descrierea aplicatiei seqboot în fisierul XML:

*<**program id**="seqboot">*
*<**name_s**>seqboot</**name_s**>*
*<**ext_id**>seqboot 3.65</**ext_id**>*
*<**belongs_to**>Phylip 3.65</**belongs_to**>*
*<**description_s**>Reads in a data set, and produces multiple data sets from it by bootstrap resampling. Since most programs in the current version of the package allow processing of multiple data sets, this can be used together with the consensus tree program CONSENSE to do bootstrap (or delete-half-jackknife) analyses with most of the methods in this package. This program also allows the Archie/Faith technique of permutation of species within characters. It can also rewrite a data set to convert it from between the PHYLIP Interleaved and Sequential forms.</**description_s**>*
*<**code_s**>seqboot <<EOF%n</**code_s**>*
*<**comments_s**>Big:</**comments_s**>*
*<**state**>building_biology</**state**>*
*<**url** />*
*<**parameters**>*

<*parameter*>
...
</*parameter*>
.....
<*parameter*>
...
</*parameter*>
</*parameters*>

Concluzii si observatii

1) Se observa ca toate atributele, din baza de date, corespunzatoare programului, sunt exprimate în cadrul fisierului XML sub forma de tag-uri.
2) Parametrii si programul se afla în relatia de cardinalitate 1..n : 1.
3) Structura fisierului XML este în concordanta cu cea impusa de fisierul DTD si cu structura BioDesc.

- **Nivelul parametri**

Pentru descrierea de la nivelul parametrilor, BioDesc foloseste urmatoarele caracteristici:

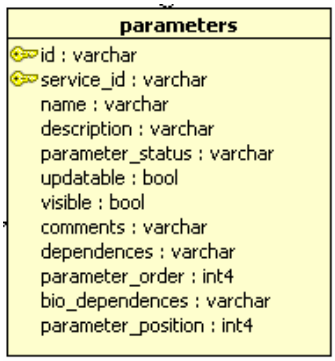

Fig. 3.7. Diagrama descriptiva a parametrilor

Nu este necesara definirea unui identificator al serviciului, deoarece parametri sunt descrisi în calitate de caracteristici ale unui serviciu sau program.

Pentru un import/export riguros este importanta identificarea tipurilor parametrilor: „input” sau „output”, daca este multiplu sau nu (a se vedea mai jos definirea atributului **attlist type**).

Parametrii sunt caracterizati pe de o parte de atribute care au rol de identificare, definire si reprezentare, în cadrul interfetei BioSide, iar pe de alta parte de tipuri, care sunt descrise la rândul lor de o serie de caracteristici.

Definirea tag-urilor parametrilor în fisierul DTD:

```
<!ELEMENT parameter (name?, description?, position, order, comments?, status, types?, dep?, bio_dep?)>

<!ATTLIST parameter
  id %integer; #REQUIRED
  ismandatory %boolean; #IMPLIED
  ishidden %boolean; #IMPLIED
  isupdatable %boolean; #IMPLIED
  type (command | code | input | output | int | float | string | switch | Excl | List ) #REQUIRED
>
<!ELEMENT name (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT position (#PCDATA)>
<!ELEMENT order (#PCDATA)>
<!ELEMENT comments (#PCDATA)>
<!ELEMENT status (#PCDATA)>
<!ELEMENT dep (#PCDATA)>
<!ELEMENT bio_dep (#PCDATA)>

<!ELEMENT types (type)+>
```

Exemplu – descrierea unui parametru al programului seqboot, în fisierul XML:

```
<parameter id="weights" isupdatable="1" type="command" ismandatory="1">
    <name>weights</name>
    <description>input weight file</description>
    <position>27</position>
    <order>2</order>
    <comments />
    <status>input</status>
    <types>
        <type>
            ...
        </type>
        <type>
            ...
        </type>
            ...
```

</types>
<dep>(W:value="1")</dep>
<bio_dep>de W</bio_dep>
</parameter>

Concluzii si observatii

1) Caracteristicile din baza de date, corespunzatoare parametrilor, sunt exprimate în cadrul fisierului XML sub forma de tag-uri sau în calitate de atribute în interiorul acestora.
2) Tipurile corespunzatoare unui parametru se afla în relatia de cardinalitate 1..n : 1.
3) Structura din fisierul XML, la nivel de parametri, este în concordanta cu cea impusa de fisierul DTD si cu structura BioDesc.
4) Caracteristicile care sunt de tip boolean, sunt reprezentate în XML sub forma unor atribute incluse în tag-ul de definire a parametrului.

- **Nivelul tipuri**

În mod similar, tipurile sunt definite cu ajutorul unor atribute specifice. Nu este obligatoriu ca aceste atribute sa aiba valori nenule.

În figura 3.8. este reprezentata diagrama corespunzatoare tabloului **types** al bazei de date BioDesc.

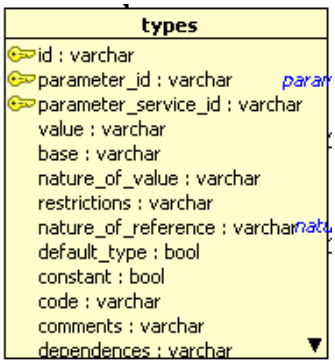

Fig. 3.8. Diagrama descriptiva a tipurilor

Nu este necesara definirea identificatorului parametrului si nici a identificatorului serviciului, deoarece tipurile reprezinta caracteristici ale unui anumit parametru, care, la rândul sau, este o caracteristica a unui program sau serviciu.

Pe lânga toate atributele din figura 3.8., tipurile mai sunt caracterizate si de atributul **format**. Acesta nu este reprezentat în diagrama, deoarece este preluat prin relatiile dintre tablourile **types – types_has_format – formats** ale bazei de date BioDesc**.**

Definirea tag-urilor tipurilor în fisierul DTD:

<!ELEMENT type (base, value?, nvalue?, nreference?, code_t?, bio_depend?, depend?, bio_res?, res?, formats?, comments_t?)>

```
<!ATTLIST type
   id %integer; #REQUIRED
   isconstant %boolean; #IMPLIED
   default %boolean; #IMPLIED
   iscode %boolean; #IMPLIED
   isreference %boolean; #IMPLIED
>

<!ELEMENT base (#PCDATA)>
<!ELEMENT value (#PCDATA)>
<!ELEMENT nvalue (#PCDATA)>
<!ELEMENT nreference (#PCDATA)>
<!ELEMENT code_t (#PCDATA)>
<!ELEMENT bio_depend (#PCDATA)>
<!ELEMENT depend (#PCDATA)>
<!ELEMENT bio_res (#PCDATA)>
<!ELEMENT res (#PCDATA)>
<!ELEMENT formats (#PCDATA)>
<!ELEMENT comments_t (#PCDATA)>
```

Exemplu – descrierea unui tip din cadrul programului seqboot, în fisierul XML:

```
<type isconstant="0" id="T1" iscode="0">
        <base>string</base>
        <value>factors</value>
        <nvalue>path</nvalue>
        <nreference>factors</nreference>
        <code_t>%s%n</code_t>
        <formats>phylip</formats>
        <comments_t />
</type>
```

Concluzii si observatii

1) Caracteristicile din baza de date, corespunzatoare tipurilor, sunt exprimate în cadrul fisierului XML sub forma de tag-uri sau în calitate de atribute în interiorul acestora.
2) Structura din fisierul XML, la nivel de tipuri, este în concordanta cu cea impusa de fisierul DTD si cu structura BioDesc.
3) Caracteristicile care sunt de tip boolean, sunt reprezentate în XML sub forma unor atribute incluse în tag-ul de definire a parametrului.
4) Tag-urile care au valori nule, nu sunt afisate decât daca sunt marcate ca fiind obligatorii în fisierul DTD; tag-urile optionale sunt însotite de semnul întrebarii „?”.

3.2.2. Crearea claselor de import - export

Proiectul bioinformatic BioSide este conceput în întregime utilizând limbajul de programare orientat pe obiecte, Java, arhitectura MVC (Model – View – Controller).

Clasele de import, respectiv de export, fac parte din pachetul **control** si poarta numele XMLv2bisToModel.java, ModelToXMLv2bis.java (a se vedea Anexa 4).

Clasa responsabila de import extrage informatiile corespunzatoare tag-urilor XML ale unui fisier, pe care îl deschide în prealabil cu ajutorul unei variabile de tip Element; interfata Element reprezinta un element din cadrul unui document HTML sau XML. Fiecare valoare extrasa este adaugata imediat în baza de date.

Ex.: ***deschiderea unui fisier XML:***

```
private Element root;                  // Element descrie elementele fisierului
                                          XML
static Document document;              // Document defineste comportamentul
                                          unui document XML, modelat în Java

SAXBuilder sxb = new SAXBuilder();
document = sxb.build(new File(file));  // Construieste un document pornind de
```

la fisierul furnizat

root = document.getRootElement();

extragerea unei valori din fisierul XML

String serviceURL = root.getChildText("url"); // „url” este tag-ul XML

adaugarea unei valori în baza de date

Service service; // Service este clasa din pachetul **model** care descrie serviciile din baza de date

service.setURL(serviceURL);

Clasa responsabila de export, extrage valorile din baza de date si le plaseaza între tag-urile unui fisier XML. Mai întai este creat fisierul XML, folosind o metoda care tine cont de fisierul DTD asociat iar apoi se creeaza câte un tag XML pentru fiecare valoare citita din baza de date. Pentru pastrarea ierarhiei serviciu (program) -> parametri -> tipuri, adaugarea valorilor în fisierul XML se face corespunzatr nuvelului ierarhic.

Ex: ***crearea unui fisier XML***

createXMLFile(file) // metoda prin care este creat fisierul XML, unde se mentioneaza calea catre fisierul DTD asociat

crearea elementului corespunzator programului (serviciului)

private Element root;
root = new Element("program");

citirea valorilor din baza de date si crearea elementelor XML corespunzatoare

Element serviceName, serviceParameter;

serviceName = new Element("name_s"); // nivelul serviciu
serviceName.setText(service.getName());

parameterElement = new Element("parameter"); // nivelul parametri
parameterName = new Element("name");
parameterName.setText(parameter.getName());

adaugarea valorilor în fisierul XML

root.addContent(serviceName); // nivelul serviciu

parameterElement.addContent(parameterName); // nivelul parametru
root.addContent(serviceParameters); // nivelul serviciu

Obs.: Acestea sunt principiile de import si export între XML si baza de date BioDesc. Codul complet poate fi observat în Anexa 4.

3.3. Perspective

Obiectivele de dezvoltare BioDesc pot fi structurate astfel:

- **Conceperea unei noi baze de date**
  - Exista necesitatea de a diferentia programul bioinformatic de serviciul pe care acesta îl efectueaza în cadrul scenariilor de cercetare biologica. Datorita evolutiei programelor bioinformatice, de-a lungul timpului au aparut diferite versiuni ale aceluiasi program. În concluzie, exista mai multe programe diferite ca si descriere, dar care efectueaza acelasi serviciu. Drept urmare, baza de date va contine un tablou de relatii corespunzator descrierii serviciilor si un altul pentru descrierile programelor.

  - În cadrul bazei de date actuale, câmpurile caracteristice datelor bioinformatice si, respectiv celor biologice sunt descrise în coloane diferite ale acelorasi tablouri. Pentru a avea o evidenta mult mai clara si pentru a facilita munca bioinformaticienilor, datele cu caracter biologic vor fi plasate în tablouri distincte.

  - În urma analizei BioDesc s-a constat ca nu exista o diferenta clara în definirea atributelor care descriu comportamentul unui program si a celor care au rol în cadrul interfetei BioSide; acestea sunt descrise în aceleasi tablouri, fara a face distinctia între rolurile lor.

- **Îmbogatirea resurselor bioinformatice în cadrul BioDesc**
  - La momentul de fata, BioDesc contine un numar modest de programe bioinformatice. Analiza ansamblelor de programe EMBOSS si Pise are în vedere traducerea caracteristicilor programelor din reprezentarea ACD si, respectiv mobyle, în reprezentarea BioDesc.

  - Exista necesitatea de integrare în baza de date BioDesc a bancilor de date, continând proteine, nucleotide si acizi.

- **Adaptarea interfetei BioDesc**
  - Structura interfetei va contine înca o sectiune dedicata caracterizarii programului, iar sectiunea care descrie serviciul va cuprinde doar informatiile specifice acestuia. Momentan descrierea serviciilor si cea a programelor este combinata.

  - Se doreste, la fiecare nivel structural (serviciu, programe, parametri, tipuri) o diferentiere clara a câmpurilor cu caracter biologic si a celor care au rol în caracterizarea interfetei BioSide.

Capitolul 4. Analiza ACD

4.1. Sinteza ACD

4.1.1. Structura fisierului ACD

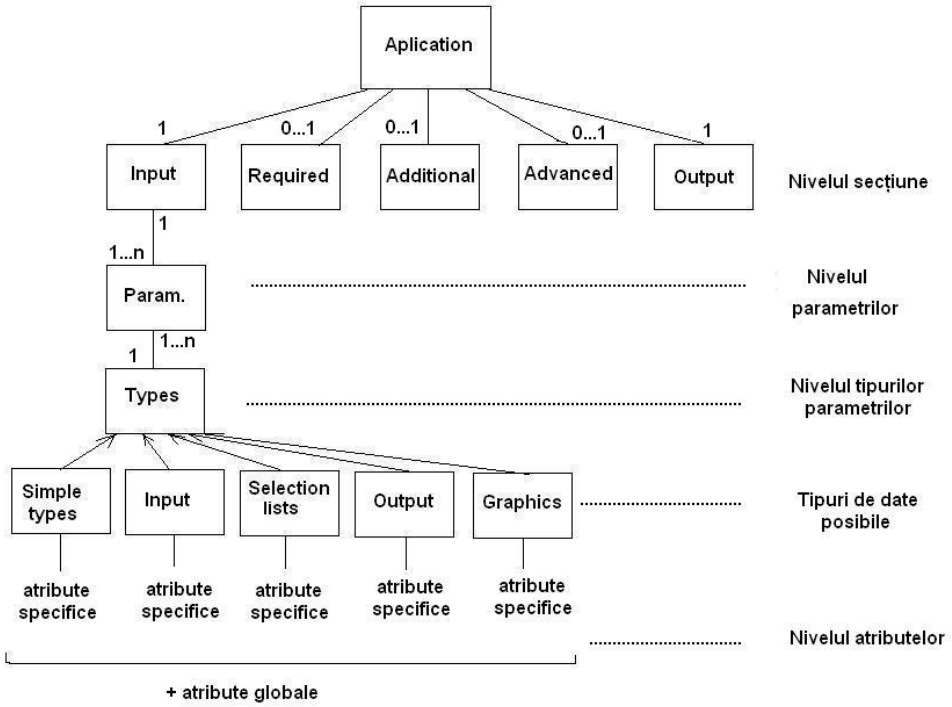

Fig. 4.1. Structura unui fisier ACD

Obs.: Exemplele utilizate în descrierea sectiunilor sunt din cadrul fisierului fseqbootall.acd, disponibil în varianta completa în Anexa 2.

Notatii utilizate:

1) Cuvintele având format italic din codul specific fisierului ACD, sunt cuvinte rezervate.

2) Valorile "Y", "Yes", "1" sunt echivalente între ele si corespund valorii de tip boolean "adevarat", iar valorile "N", "No", "0" sunt echivalente si corespund valorii de tip boolean "fals".

- **Aplicatia**
  - ***Semantica***

    Aplicatia reprezinta programul executabil care contine codul sursa. Ea este caracterizata de un nume, o descriere si o pozitie în clasamentul programelor.

    Exista programe apartinând altor ansamble de programe, dar care sunt integrate în EMBOSS. Pentru acestea, în caracterizarea aplicatiei trebuie specificat si grupul de programe din care acesta face parte (la sectiunea embassy).

  - ***Sintaxa***

    application: nume_program [
    documentation: "un sir de caractere care descrie programul"
    groups: "numele grupului de programe din care face parte aplicatia"
    embassy: "numele ansamblului de programe din care face parte"
    ]

    Obs.: Grupurile de programe predefinite în ACD sunt cuprinse în primul tabel din prima anexa.

    Exemplu:

    application: fseqbootall [
    documentation: "Bootstrapped sequences algorithm"
    groups: "Phylogeny:Molecular sequence"
    embassy: "phylipnew"
    ]

- **Sectiunea**
  - ***Semantica***

Fisierul ACD este structurat în cinci sectiuni:

1) Sectiunea **Input** – în aceasta sectiune este definit în mod obligatoriu ansamblul de parametri si calificatori care sunt de tip Input. Tot aici pot fi definiti si alti calificatori care au legatura cu datele de intrare si care nu sunt de tip Input sau Output.

2) Sectiunea **Required** – contine toti parametrii si calificatorii care sunt caracterizati de atributul *standard*: "Y" (absenta acestui atribut este echivalenta situatiei în care *standard* are valoarea "N"). Tot în aceasta sectiune pot fi definiti si calificatorii marcati ca si *additional* pentru care valoarea "Y" a acestui atribut depinde de o conditie si care în acelasi timp sunt definiti ca si *standard*. Parametrii si calificatorii de intrare si iesire (care sunt de tip Input si Output), chiar daca sunt marcati ca fiind *standard* trebuie definiti în sectiunile Input, respectiv Output.

3) Sectiunea **Additional** – reuneste definitiile calificatorilor care sunt marcati ca fiind *additional* (atributul *additional*: "Y"), chiar daca valoarea "Y" a acestui atribut depinde de o operatie conditionala. Nu este posibil ca parametrii si calificatorii de tip Input si Output sa fie plasati în aceasta sectiune chiar daca sunt definiti ca fiind *additional*. În schimb, calificatorii aflati în legatura cu datele de intrare, dar care nu sunt de tip Input sau Output, pot fi definiti în sectiunea Additional.

4) Sectiunea **Advanced** – contine toti calificatorii care nu sunt de tip Input sau Output si care nu sunt definiti ca fiind *standard* sau *additional*. Aici pot fi plasati calificatorii care sunt legati de datele de intrare sau iesire ale aplicatiilor, dar care nu sunt de tip Input sau Output.

5) Sectiunea **Output** – în aceasta sectiune sunt plasati în mod obligatoriu toti parametrii si calificatorii de tip Output sau Graphics. Tot aici pot fi plasati si alti calificatori care au legatura cu datele de iesire si care nu sunt de tip Input sau Output.

Obs.: a) *additional* si *standard* sunt atribute globale; sunt prezentate într-un paragraf ulterior din acest capitol.

b) tipurile Input, Output si celelalte tipuri de date ale parametrilor sau calificatorilor sunt definite în paragraful urmator.

- ***Sintaxa***

section: numele sectiunii [
information: "descrierea sectiunii"
comments: "comentarii generale atunci când sunt necesare"
type: "tipul interfetei"
side: specificatii ale partilor când tipul este "frame"
border: dimensiunea bordurii
folder: numele directorului în care sunt salvate informatiile cand tipul e "page"
]

Obs.: a) Atributele *information* si *type* au caracter obligatoriu.

b) *type* are semnificatie pentru interfata grafica sau www.

Exemplu:

section: input [
information: "Input section"
type: "page"
]

- **Parametrii si tipuri de date**
  - ***Semantica***

    În definirea parametrilor, trebuie neaparat sa se tina cont de tipurile de date ale acestora. Sunt numite parametrii sau calificatori, entitatiile definite în fisierul ACD, în cadrul sectiunilor. În special, parametrii sunt definiti ca si calificatori descrisi de atributul global *parameter:* "Y".

    În ACD sunt definite cinci categorii de date, care au un nume unic. Un tip nu poate sa apara în doua categorii diferite. Fiecare tip de date este caracterizat de o serie de atribute specifice. Pe lânga atributele specifice, parametrii pot fi definiti cu ajutorul atributelor globale.

    Atributele cu caracter global pot fi utilizate independent de tipul parametrilor si contribuie la pozitionarea definitiilor parametrilor în cadrul sectiunilor fisierului ACD. O parte din atributele globale sunt în strânsa legatura cu calificatorii globali. Putem observa ca cele doua roluri ale acestor atribute sunt strict legate de interfata cu utilizatorul (mai multe detalii se pot gasi în paragraful dedicat calificatorilor si atributelor globale, în acest subcapitol).

    Atributele specifice sunt atribute proprii fiecarui tip de date. În Anexa 1 exista tabele în care sunt grupate atributele specifice prntru fiecare tip de date din cadrul celor cinci categorii posibile (Tabelele: 2, 4, 5, 6, 7, 8).

    Pentru tipul de date ***sequence*** din cadrul categoriei Input sunt definite atribute calculate, care sunt generate automat în momentul procesarii fisierului ACD. Acestea pot fi gasite în Anexa 1, tabelul 9.

    Cele cinci categorii de tipuri de date sunt:

    - **tipurile de intrare (Input types)** – în Anexa 1, tabelul 5

      Exemplu:

      properties: categories [
      additional: "Y"
      characters: ""
      information: "File of input categories"
      nullok: "Y"
      size: "1"
      length: "$(sequence.length)"
      ]

- **tipurile de iesire (Output types)** – în Anexa 1, tabelul 7

  Exemplu:

```
outfile: outfile [
        parameter: "Y"
        knowntype: "seqbootseq output"
        information: "Phylip seqboot_seq program output file"
 ]
```

- **tipurile simple (Simple types)** – corespund tipurilor de date utilizate în limbajele de programare clasice (int, float, string, boolean, etc.) – în Anexa 1, tabelul 4

  Exemplu:

```
integer: blocksize [
        information: "Block size for bootstraping"
        additional: "@($(test) == b)"
        default: "1"
        minimum: "1"
    ]
```

- **tipurile meniu de selectie** – acestea sunt liste de selectie si difera între ele doar prin modul în care este dispusa informatia în interfata cu utilizatorul.

  Exemplu:

```
list: matrix [
        def: "blosum"                   # valoarea implicita
        min: 1                          # este posibila o singura alegere
        max: 1
        header: "Comparison matrices"   # descrierea parametrului
        values: "B:blosum, P:pam, I:id" #3 valori valide
        delim: ","                      # delimitatorul implicit ";"
        codedelim: ":"                  # delimitatorul implicit al
                                            etichetei ":"
        prompt: "Select one"            # cererea catre utilizator
        button: Y                       # tipul butoanelor
 ]
```

***în interfata apar:

```
        Comparison matrices
        B : blosum
```

P : pam
I : id
Select one [blosum] : P

Elementele din meniul ***list*** sunt identificate prin litere B, P, I.

select: matrix [
def: "blosum" # valoarea implicita
min: 1 # este posibila o singura
max: 1 alegere
header: "Comparison matrices" # descrierea param.
values: "blosum, pam, id" # valori valide
delim: "," # delimitatorul implicit
";"
prompt: "Select one" # cererea catre utilizator
button: Y # tipul butoanelor
]

***în interfata apar:
Comparison matrices
1 : blosum
2 : pam
3 : id
Select one [blosum] : 2

Elementele meniului ***select*** sunt identificate prin numerele 1, 2, 3.

- **tipurile grafice (Graphic types)** – date de iesire sub diverse forme grafice – ilustrarea lor se gaseste în tabelul 8 din Anexa 1.

o ***Sintaxa***

tipul de date: numele parametrului [
atribut1: valoare
atribut2: valoare
atribut3: valoare
...
]

Obs.: **a)** Atributele pot fi globale sau specifice pentru tipul de date. Valorile atributelor sunt din categoria tipurilor simple.

b) Atributul specific pentru tipul de date ***type*** poate avea doar anumite valori. În Anexa 1, în tabelul 10, pot fi gasite valorile posibile ale acestui atribut.

- **Atribute specifice pentru tipurile de date**
  - ***Semantica***

    Atributele specifice pentru tipurile de date au fost ilustrate în definirea parametrilor. Exista un set de atribute predefinite pentru fiecare tip de date din fiecare din cele cinci categorii. Cu ajutorul lor sunt descrise atât caracteristici ale parametrilor, ale modului de afisare cât si constrângeri.

  - ***Sintaxa***

    nume atribut: valoare

    Exemplu:

    list: seqtype [
    minimum: "1"
    maximum: "1"
    header: "test"
    values: "d:dna; p:protein; r:rna"
    ]

    Obs.: Toate atributele (specifice si globale) sunt cuprinse între paranteze drepte [...].

- **Atributele globale si calificatorii**

  - **<u>Atribute globale</u>**
    - ***Semantica***

      Atributele globale au o actiune generala asupra parametrilor si calificatorilor pe care îi descriu, indiferent de tipul acestora.

      Mai jos pot fi observate câteva exemple semnificative de atribute globale:

- **default** – marcheaza valoarea implicita a parametrilor;
- **information** – are importanta pentru modul în care sunt afisate informatiile în interfata cu utilizatorul;
- **parameter** – are valoare de tip boolean si arata faptul ca un calificator este definit ca si parametru
- **standard** – are valoare de tip boolean si exprima caracterul obligatoriu al unui parametru – determina sectiunea interfetei în care va fi afisat parametrul.
- **additional** – are valoare de tip boolean si exprima caracterul aditional al unui parametru – determina sectiunea interfetei în care va fi afisat parametrul.

Tot atribute globale sunt si: help, expect, valid, knowntype, prompt, missing, needed, outputmodifier, code, comment, style.

- ***Sintaxa***

nume atribut: valoare

Exemplu:

```
boolean: enzymes [
    additional: "@($(datatype) == r)"
    information: "Is the number of enzymes present in input
                  file"
    default: "N"
  ]
```

Obs.: Atributele globale sunt prezentate în tabelul 2 din Anexa 1.

❖ Calificatorii

Obs.: În acest paragraf sunt tratati calificatorii care sunt mentionati în linia de comanda împreuna cu secventa de apel a aplicatiei. Acest termen, ***calificator***, este folosit si pentru a desemna entitatile descrise în cadrul fisierului ACD. Pentru acestia din urma se utilizeaza si apelativul de parametri.

- ***Semantica***

Acesti calificatori nu sunt definiti în cadrul fisierului ACD. Sunt cuvinte consacrate având roluri bine definite, utilizate în linia de comanda.

Calificatorii ACD sunt: globali sau specifici tipurilor de date.

Calificatorii globali

Calificatorii globali sunt cuvinte rezervate care, scrise în linia de apel a unei aplicatii EMBOSS, determina modificarea comportamentului aplicatiei. De exemplu, calificatorul global ***–auto*** determina executia programului bioinformatic atribuind parametrilor valorile implicite. În mod normal dupa apelul unui program, în interfata de comada apar cereri catre utilizator, iar utilizând calificatorul –auto aceste cereri lipsesc; apar doar mesaje de eroare în cazul în care un anumit parametru nu are definita o valoare implicita desi ar trebui (adica atributul *expected* are valoarea "Y").

Un alt exemplu este calificatorul ***–option*** care permite afisarea parametrilor aditionali în interfata cu utilizatorul. Parametrii marcati ca si *standard* sau care au valoarea "Y" a atributului *parameter* sunt în mod obligatoriu afisati.

Lista calificatorilor globali este urmatoarea: acdlog, acdpretty, auto, debug, filter, help, options, stdout. (Anexa 1, tabelul 11).

Acelasi rol îl au si variabilele de mediu specifice acestor calificatori. Aceste variabile pot fi gasite în Anexa 1, tabelul 12.

Calificatorii specifici tipurilor de date

Asa cum exista atribute specifice tipurilor de date, exista si calificatori care pot fi folositi în linia de comanda pentru a caracteriza tipurile de date Input, Output sau Graphics (Anexa 1, tabelele 13, 14, 15).

Unul dintre cei mai utilizati calificatori de acest tip, este calificatorul ***format*** cu ajutorul caruia se specifica formatul fisierelor de intrare / iesire.

- ***Sintaxa***

➔ calificator global:

% **nume_program** *–calificator_global*

➔ calificator specific:

% **nume_program** fisier intrare/ iesire *–calificator_specific*

Exemplu:

% **seqret** *sequence.seq* –debug (calificator global)

- porneste modul debug pentru depistarea eventualelor erori ale programului bioinformatic seqret

% **seqret** *sequence.seq* -sformat fasta (calificator specific)

- atribuie secventei de intrare, sequence.set, a programului bioinformatic seqret formatul fasta

4.1.2. Sectiunile interfetei versus sectiunile fisierului ACD

Dupa cum s-a putut observa în subcapitolul referitor la structura fisierului ACD, în cadrul unui fisier descriptiv AJAX Command Definition exista cinci sectiuni distincte: Input, Required, Additional, Advanced si Output. Modul în care pozitionam fiecare parametru în respectivele sectiuni este explicat în detaliu în subcapitolul amintit.

Spre deosebire de structura fisierului ACD, interfata cu utilizatorul este structurata în trei sectiuni distincte:

- **Standard (Mandatory) qualifiers** – aici sunt afisati parametrii si calificatorii pentru care atributul global *standard* are valoarea "Y" si cei pentru care atributul global *parameter* are valoarea "Y".

- **Additional (Optional) qualifiers** – în acesta sectiune sunt prezentati toti parametrii care sunt descrisi cu ajutorul atributului global *additional:* "Y".

- **Advanced (Unprompted) qualifiers** – parametrii care nu sunt de tip Input sau Output si care nu sunt nici *standard* nici *additional.*

Sectiunile înterfetei generate la apelul unei aplicatii EMBOSS	Sectiunile fisielului ACD	Tipuri de date posibile
Standard - ***orice tip de date***;	**Input** - tipurile ***Input, Simple, Selection lists;***	
	Required - tipurile ***Simple, Selection lists;***	
Additional - ***orice tip de date***;	**Output** - tipurile ***Output, Graphics, Simples, Selection lists;***	
	Additional - tipurile ***Simples, Selection lists;***	
Advanced - tipurile de date ***Simple, Selection lists, Graphics***.	**Advanced** - tipurile ***Simples, Selection lists.***	

Tabel 4.1. Sectiunile interfetei versus sectiunile fisierului ACD

4.1.3. Alte caracteristici ale limbajului ACD

În cadrul datelor de tip **integer** sau **boolean**, parametrii de acest tip pot fi definiti ca rezultate ale unor operatii aritmetice, respectiv conditionale.

Forma generala de definire a unei operatii este: *@(operatie)*

Operatiile aritmetice posibile în cadrul limbajului ACD sunt:

@(*a*+*b*) (Adunare)

@(*a*-*b*) (Scadere)

@(*a***b*) (Înmultire)

@(*a*/*b*) (Împartire)

Exemplu: *variable*: protlen "@($(sequence.length)/ 3)"
integer: window [
maximum: "@($(protlen) - 50)"
default: 50
]

Operatiile conditionale posibile în cadrul limbajului ACD sunt:

1) *operatii de tip boolean*

   @(*token1==token2*) (Egalitate)

   @(*token1!=token2*) (Neegalitate)

   @(*token1<token2*) (Mai mic decât)

   @(*token1>token2*) (Mai mare decât)

   @(!*token1*) (Negare)

   @(*token1*|*token2*) (Sau)

   @(*token1&token2*) (Si)

Exemplu: *sequence*: seq [
standard: Y
]
infile: data [
standard: @(seq.type==DNA)
]

***Parametrul standard va avea valoarea „Y” numai daca tipul secventei este ‚DNA’.

2) *operatie simpla conditionala*

   @(boolval ? iftrue : iffalse)

   Exemplu:

   string: matrix [
   default: "@($(asequence.protein) ? BLOSUM62 : DNAMAT)"
   ]

   ***Matrix va avea valoarea implicita „BLOSUM 62” daca secventa este o proteina.

3) *operatie de tip **case***

   @(testval = poss_valA : ass_valA poss_valB : ass_valB else : default_val)

Exemplu:

string: matrix [
default: "@($(sequence.type) =
protein : BLOSUM62
dna : dnamat
rna : rnamat
else : unknown)"
]

***Valoarea implicita a calificatorului matrix va fi „BLOSUM62” daca tipul secventei este proteic, „dnamat” daca este ADN (acid dezoxiribonucleic), „rna” daca este ARN (acid ribonucleic) sau „unknown”.

Observatii:

1. Utilizarea variabilelor în ACD reprezinta o metoda de simplificare a continutului fisierului .acd.
2. Parametrii si calificatorii definiti în ACD, sunt procesati în ordinea în care apar.

4.2. Concluziile analizei ACD

Analiza realizata în lucrarea de fata reprezinta o caracterizare a modului în care EMBOSS gestioneaza si afiseaza în interfata cu utilizatorul caracteristicile programelor bioinformatice. Aceasta caracterizare este realizata în raport cu functionalitatile programelor bioinformatice originale.

Pentru analiza, am considerat programul bioinformatic ***seqboot*** din ansamblul de programe Phylip; acesta este programul original, propus de Joe Felsenstein (Universitatea Washington) în cadrul programelor bioinformatice de tip filogenetic.

```
Bootstrapping algorithm, version 3.6

Settings for this run:
  D      Sequence, Morph, Rest., Gene Freqs?  Molecular sequences
  J  Bootstrap, Jackknife, Permute, Rewrite?  Bootstrap
  %    Regular or altered sampling fraction?  regular
  B      Block size for block-bootstrapping?  1 (regular bootstrap)
  R                    How many replicates?  100
  W              Read weights of characters?  No
  C                Read categories of sites?  No
  S     Write out data sets or just weights?  Data sets
  I             Input sequences interleaved?  Yes
  0      Terminal type (IBM PC, ANSI, none)?  ANSI
  1       Print out the data at start of run  No
  2     Print indications of progress of run  Yes

  Y to accept these or type the letter for one to change
```

Fig. 4.2. Interfata seqboot

Interfata propusa de programul ***seqboot*** este cea din figura 4.2. Este vorba de un meniu prin care putem naviga uitilizând identificatorii parametrilor: literele si simbolurile din stânga meniului.

Abordarea EMBOSS a programului bioinformatic seqboot este ***fseqbootall*** (fisierul aplicatie si fisierul ACD). Interfata generata este diferita de cea a seqboot; este vorba tot de o interfata în linia de comanda, dar care cere utilizatorului sa introduca valori pentru parametrii programului.

Standard (Mandatory) qualifiers		**Allowed values**	**Default**
[-infile] (Parameter 1)	(Aligned) sequence set filename and optional format, or reference (input USA)	Readable set of sequences	**Required**
[-outfile] (Parameter 2)	Phylip seqboot program output file	Output file	
Additional (Optional) qualifiers		**Allowed values**	**Default**
-categories	File of input categories	Property value(s)	
-mixfile	File of mixtures	Property value(s)	

-ancfile	File of ancestors	Property value(s)	
-weights	Weights file	Property value(s)	
-factorfile	Factors file	Property value(s)	
-datatype	Choose the datatype	s *(Molecular sequences)* m *(Discrete Morphology)* r *(Restriction Sites)* g *(Gene Frequencies)*	s
-test	Choose test	b *(Bootstrap)* j *(Jackknife)* c *(Permute species for each character)* o *(Permute character order)* s *(Permute within species)* r *(Rewrite data)*	b
-regular	Altered sampling fraction	Toggle value Yes/No	No
-fracsample	Samples as percentage of sites	Number from 0.100 to 100.000	100.0
-rewriteformat	Output format	p *(PHYLIP)* n *(NEXUS)* x *(XML)*	p
-seqtype	Output format	d *(dna)* p *(protein)* r *(rna)*	d

-morphseqtype	Output format	p *(PHYLIP)* n *(NEXUS)*	p
-blocksize	Block size for bootstraping	Integer 1 or more	1
-reps	How many replicates	Integer 1 or more	100
-justweights	Write out datasets or just weights	d *(Datasets)* w *(Weights)*	d
-enzymes	Is the number of enzymes present in input file	Boolean value Yes/No	No
-all	All alleles present at each locus	Boolean value Yes/No	No
-seed	Random number seed between 1 and 32767 (must be odd)	Integer from 1 to 32767	1
-printdata	Print out the data at start of run	Boolean value Yes/No	No
-[no]dotdiff	Use dot-differencing	Boolean value Yes/No	Yes
-[no]progress	Print indications of progress of run	Boolean value Yes/No	Yes
Advanced (Unprompted) qualifiers		**Allowed values**	**Default**
(none)			

Fig 4.3. Interfata fseqbootall

La prima vedere cele doua aplicatii par destul de diferite, dar navigând prin meniul programului seqboot, gasim aceiasi parametri ca si în fseqbootall. În cazul seqboot este vizibil (în fig. 4.2.) un caz particular, în care parametrii au valorile pe care le putem observa în dreapta figurii, dupa semnul de întrebare. În cazul fseqbootall, în interfata vor fi afisati aceiasi parametri pentru cazul particular ilustrat de seqboot; în ACD exista posibilitatea de a afisa toti parametrii unui program utilizând calificatorul global *acdpretty*.

În programul original caracteristicile ***categories, weights, mixfile, ancfile, factors*** sunt implementate sub forma de parametri de tip boolean, în timp ce în feseqbootall acesti parametri sunt valori de tip sir de caractere.

O alta diferenta între cele doua reprezentari este modul de afisare a parametrilor care pot avea o singura valoare dintr-o lista de valori. În seqboot, valorile posibile sunt afisate între paranteze, iar meniul poate fi parcurs prin introducerea simbolului identificator al parametrului respectiv. În fseqbootall, meniurile sunt afisate riguros, fiecare valoare posibila având un identificator propriu.

Exista parametri a caror activare depinde de valoarea unui parametru anterior. În fseqbootall, toti parametrii sunt definiti în mod separat având identificatori proprii, în timp ce în seqboot exista cazuri în care parametrul conditionat nu are un identificator în meniul principal (simbolurile din stânga vezi figura 4.2-) ci apare spontan în interfata ca o cerere catre utilizator.

Facând analiza seqboot în raport cu fseqbootall putem observa ca exista doua caracteristici care sunt prezente doar în seqboot:

- *Input sequence interleaved* -> în fseqbootall nu se pune problema tipului secventei;
- *Terminal type(tipul sistemului de operare)* -> EMBOSS acest detaliu nu este problematic – caracteristicile nu se schimba în functie de tipul terminalului.

În concluzie, seqboot si fseqbootall reprezinta doua abordari diferite ale acelorasi functionalitati bioinformatice. Diferentele existente se datoreaza tipurilor de afisaj, implementarilor diferite ale unor parametri dar si evolutiei tehnologice, EMBOSS îmbunatatind variantele anterioare ale programelor bioinformatice.

Capitolul 5. ACD versus BioDesc

5.1. Paralela semantica

Din punct de vedere semantic s-au identificat câteva problematici de importanta majora pentru o ulterioara integrare a programelor EMBOSS în baza de date BioDesc. În continuare sunt explicate si exemplificate toate aceste problematici.

- **Modalitati de identificare a programelor bioinformatice**
  - **BioDesc**

    În BioDesc, fiecare program bioinformatic este identificat prin intermediul ***numelui*** si al ***versiunii*** (Ex: seqboot 3.65). În identificarea programului se specifica de asemenea ***ansamblul de programe*** din care face parte (Phylip 3.65) si, în calitate de identificator intern, i se aloca un ***numar unic al serviciului efectuat***. Pentru urmatoarea versiune a BioDesc, între serviciu si program nu va mai exista o cardinalitate 1:1 ci 1:n deoarece vor exista mai multe versiuni ale aceluiasi program care, bineînteles, efectueaza acelasi serviciu.

  - **ACD**

    În cazul programelor din pachetul EMBOSS, identificarea unui program se face prin ***nume***. Aici, pentru fiecare versiune se schimba numele programului si simultan numele fisierului ACD care îl însoteste. De exemplu, variantele EMBOSS ale programului seqboot (pentru un caz particular) poarta denumirile: eseqbootall – în pachetul phylip si fseqbootall – în pachetul phylipnew.

    Tot pentru identificarea programului este precizat ansamblul de programe din care acesta face parte (Ex: phylip, phylipnew) si grupul de programe în care este încadrat. În ACD, programele sunt grupate în categorii, având nume sugestive, în functie de scopul lor (Ex: "Phylogeny: Molecular sequence").

- **Format**
  - **BioDesc**

Fisierele si secventele care constitue intrari sau iesiri ale programelor bioinformatice sunt caracterizate de un anumit format al datelor. În BioDesc fisierele si secventele mentionate sunt reprezentate sub forma de tipuri ale parametrilor de intrare / iesire, iar formatele pot fi alese dintr-o lista vizuala, dupa cum se poate observa în exemplul urmator.

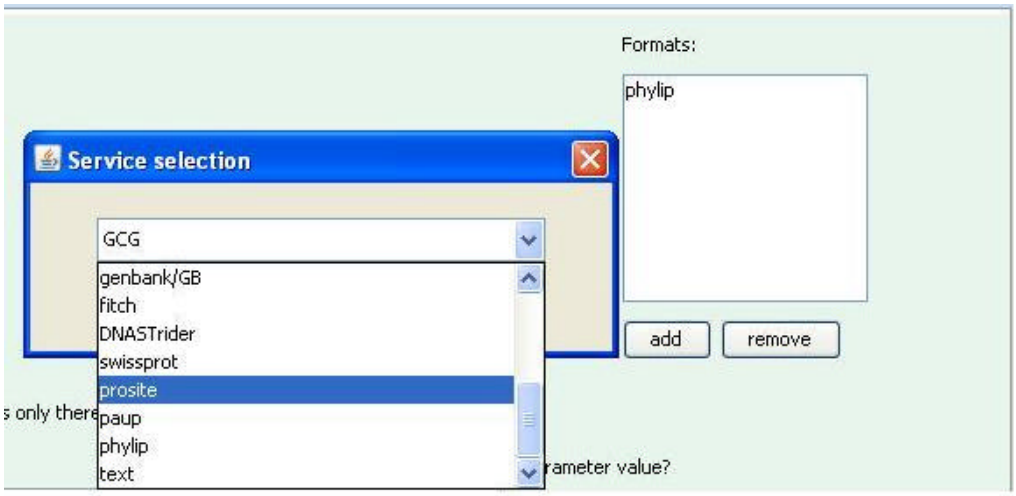

Fig. 5.1. Lista formatelor în BioDesc

(afisate ca urmare a selectarii butonului ***add*** din figura)

  - **ACD**

În cazul EMBOSS, formatul nu este definit în fisierul descriptiv ACD ci este atribuit secventelor de intrare / iesire (Input, Output) în linia de comanda, cu ajutorul calificatorilor specifici. Fiecare tip de secventa din categoriile Input si Output este caracterizat de o serie de calificatori. Printre acesti calificatori sunt definiti si cei responsabili cu formatul datelor. Este bine de observat ca nu toate tipurile de date din categoriile Input si Output sunt caracterizate de un astfel de calificator si ca pâna în acest moment calificatorii specifici sunt definiti doar pentru datele de tip secventa.

Atribuirea formatelor se realizeaza în timpul apelului de executie al programului bioinformatic. În linia de comanda, pe lânga denumirea programului, sunt scrise numele secventelor de intrare si iesire si apoi fiecare din calificatorii caracteristici responsabili de format urmat de numele formatului.

Exemplu:

Descrierea din fisierul ACD:

```
application: seqtest
sequence: asequence1 [
   parameter: Y
  ]
outfile:  outfile     [
   parameter: Y
  ]
sequence: asequence2 [
   parameter: Y
  ]
```

Linia de comanda:

% **seqtest** filename1.seq seqtest.out filename2.seq \
-sformat1 gcg -sformat2 fasta

Explicatie:

Ca urmare a executiei programului seqset, secventei de intrare **filename1.seq** i se atribuie formatul **gcg,** iar secventei de intrare **filename.seq** i se atribuie formatul **fasta**. Pentru cazul calificatorilor multipli, acestia trebuie numerotati în linia de comanda, pentru a specifica secventa pe care o caracterizeaza.

Tipurile de formate definite pentru secventele de intrare si pentru cele de iesire pot fi gasite în Anexa 3, tabelele 1 si 2.

- **<u>Problematica parametrilor multipli</u>**

În caracterizarea programelor bioinformatice se utilizeaza diferiti parametri. Exista parametri care sunt definiti cu ajutorul unuia sau mai multor tipuri. Parametrii multipli, adica cei definiti pe baza unei liste de tipuri, pot fi clasificati în doua categorii:

- parametrii pentru care tipurile sunt date de valoarea lor,
- parametrii pentru care fiecare dintre tipuri este un fisier (o referinta).

1) Parametrii liste de valori

o **BioDesc**

În BioDesc pot fi definiti ca si „liste de valori" parametrii care au valoarea constanta. Este si firesc sa fie asa deoarece, în acest caz, tipul parametrului este dat de valoare. Fiecare tip în parte este definit, la rândul sau, de o serie de caracteristici pe care utilizatorul le poate modifica actionând în interfata BioDesc. Caracterul constant sau variabil al valorilor tipurilor este descris în interfata cu ajutorul a doua butoane radio având valorile „Yes " si „No" drept raspuns la întrebarea „Is constant?" si corespund valorilor „true", respectiv „false" ale câmpului boolean „constant" al bazei de date.

Un exemplu de astfel de parametru este ***D Input data type*** întâlnit în programul **seqboot**, care poate avea unul din tipurile: Molecular Sequences, Discrete Morphology, Restriction Sites, Gene Frequencies. Aceste valori sunt siruri de caractere – au tipul de baza String.

o **ACD**

În ACD parametri „liste de valori" sunt reprezentati sub forma de calificatori (parametri), având tipul de date **Selection lists**. Valorile posibile sunt de tip sir de caractere ca si în reprezentarea BioDesc.

Obs: Calificatorii de tip **Selection lists** pot avea drept valori doar date de tip String. (Anexa 1 – Tabelul 6).

Ex: - parametrul datatype, în reprezentarea ACD:

list: datatype [
additional: "y"
minimum: "1"
maximum: "1"
header: "Choose the datatype"

values: "s:Molecular sequences; m:Discrete Morphology;
r:Restriction Sites; g:Gene Frequencies"
information: "Choose the datatype"
default: "s"
]

****în linia de comanda vor fi afisate*:
-datatype menu [s] Choose the datatype (Values:
s (Molecular sequences); m (Discrete Morphology);
r (Restriction Sites); g (Gene Frequencies))

2) Parametrii liste de referinte

- **BioDesc**

Parametrii de intrare si iesire ai unui program bioinformatic sunt definiti drept „liste de referinte". Acesti parametri sunt caracterizati de tipuri, care preiau drept denumire propria valoare de referinta (**nature of reference**) si nu sunt constante. Pentru fiecare tip din lista, caracteristica tip de baza este String, dar variabila „***Is reference***" setata pe „Yes" si valoarea „path" a variabilei „***nature of value***" ne indica faptul ca acest sir de caractere reprezinta denumirea unui fisier.

Exemplu: parametrul **Infile** al programului **seqboot**

- tipurile posibile pentru acest parametru sunt:
  - ***File/alignment***,
  - ***File/disc_morph***,
  - ***File/rest_sites***,
  - ***File/frequencies***.

Cele patru tipuri posibile sunt siruri de caractere care indica numele unor fisiere si sunt identice cu valorile variabilei „nature of reference".

- **ACD**

În ACD nu este posibil sa definim calificatori sau parametri sub forma unor „liste de referinte". Singura posibilitate de a defini parametri multipli este declararea tipului lor de data ca ***Selection*** sau ***List***. Dar valorile lor vor fi considerate întotdeauna siruri de caractere.

Atât în BioDesc cât si în ACD datele de intrare si iesire sunt referinte spre alte fisiere. Pe lânga ca nu este posibil sa definim în mod direct calificatori si parametri sub forma unor liste de referinte, în ACD mai exista si alte constrângeri:

- orice aplicatie poate avea un singur fisier de intrare;
- pentru fiecare fisier de intrare se declara fisiere de iesire corespunzatoare în care vor fi salvate rezultatele;

Astfel, programul bioinformatic seqboot, este descris în EMBOSS cu ajutorul a patru aplicatii diferite si a fisierelor ACD corespunzatoare:

- **fseqbootall** – are ca intrare o secventa ***alignment (seqset :*** **infile*)***,
- **fdiscboot** – are ca intrare un fisier ***disc_morph (infile :*** **disc_morph*)***,
- **frestboot** – are ca intrare un fisier ***rest_sites (infile :*** **rest_sites*),***
- **ffreqboot** – are ca intrare un fisier ***frequencies (infile :*** **frequencies*).***

➢ **Conceptul "nature of value"**

o **BioDesc**

Variabila „nature of value” exprima categoria din care face parte valoarea fiecarui tip ce caracterizeaza parametrii programelor bioinformatice. De exemplu, pentru cazul tipurilor de intrare/ iesire „nature of value” are valoarea „path”. Pâna în prezent, în BioDesc, au fost gestionate din punctul de vedere al naturii valorii doar fisierele. Este prevazut ca unealta software BioDesc sa poata deosebi tipurile de secvente, nu doar sa le eticheteze drept referinte catre fisiere (de exemplu sa poata sa faca distinctie între secvente de proteine si secvente ADN).

o **ACD**

În cazul ACD, conceptul de „nature of value” este gestionat cu ajutorul atributului specific **type.** Tipurile de date sunt caracterizate de diferite atribute specifice, iar atributul „type” nu trebuie sa fie prezent în mod obligatoriu, ci acesta este caracteristic anumitor tipuri de date, asa cum se poate observa în tabelele din Anexa 1.

Exemplu: în fisierul ACD fseqbootall:

```
seqset: infile [
        parameter: "Y"
```

***type*: "gapany"**
aligned: "Y"
]

Valoarea „gapany” a atributului type ne indica faptul ca este vorba de o secventa de nucleotide (Anexa 1, tabelul 10).

Obs.: Este important de remarcat ca pâna în momentul de fata acest concept este gestionat, în ACD, doar pentru tipul de data secventa.

- **Interfata si descrierea tipurilor**

Descrierile programelor bioinformatice realizate cu ajutorul BioDesc si ACD trateaza atât definitiile caracteristicilor de comportament ale programelor cât si modul lor de afisare în interfata cu utilizatorul.

Acest criteriu de analiza comparativa pune în evidenta în ce proportie sunt combinate descrierile caracteristicilor unui program bioinformatic si cele ale interfetei acestuia si daca modul de afisare este determinat de valorile caracteristicilor.

- **BioDesc**

Elementele care sunt definite expres ca având rol în cadrul interfetei BioSide sunt:

- **is visible** – valoarea „Yes” corespunde afisarii respectivului parametru, în cadrul interfetei Bioside;
- **descrierile**
- **comentariile**

În BioDesc exista variabile de tip boolean care determina aparitia sau disparitia unor caracteristici ale programului în cadrul interfetei.

În tablourile bazei de date nu exista o diferentiere clara între cele doua tipuri de caracteristici; sunt definite în coloane diferite ale acelorasi tablouri si nu exista relatii care sa fie special compuse din elementele responsabile de afisaj.

În concluzie exista un grad de amestec în definirea caracteristicilor comportamentale si a caracteristicilor de afisare.

- **ACD**

Spre deosebire de BioDesc, în ACD nu exista calificatori sau parametri special definiti pentru descrierea interfetei ci toti calificatorii, respectiv parametrii sunt caracterizati cu ajutorul atributelor globale si a celor specifice; atributele globale determina modul de afisare al interfetei, în linia de comanda, iar atributele specifice descriu caracteristicile comportamentale ale programelor.

În fisierul ACD pot fi definiti calificatori având tipul de data din categoria **Selection lists**, care prin definitie reprezinta modul de afisare a unui meniu cu multiple posibilitati de alegere (a se vedea capitolul 4).

Calificatorii globali sunt entitati ce caracterizeaza interfata, dar ei nu sunt definiti în fisierul ACD, ci sunt intodusi în linia de comanda, de catre utilizator, în functie de necesitatile de afisare.

În concluzie, în ACD descrierile caracteristicilor de afisaj si, respectiv, ale comportamentului unui program bioinformatic sunt întotdeauna combinate.

5.2. Paralela din punctul de vedere al sintaxei

Pentru a pune în evidenta diferentele de sintaxa, acest capitol propune o traducere, acolo unde este posibil, a componentelor programului bioinformatic **fseqbootall** din reprezentarea ACD, în reprezentarea BioDesc.

În BioDesc, elementele programelor bioinformatice sunt salvate în tablourile unei baze de date relationale. Traducerea componentelor din reprezentarea ACD înseamna importarea lor în baza de date BioDesc.

În continuare se poate urmari operatia de import a caracteristicilor programului bioinformatic fseqbootall din ansamblul EMBOSS în baza de date BioDesc.

- **Aplicatia**

application: fseqbootall [
documentation: "Bootstrapped sequences algorithm"
groups: "Phylogeny:Molecular sequence"

embassy: "phylipnew"
]

Aplicatia din EMBOSS este analoaga serviciului din BioDesc.

id varchar	**name** varchar	**description** varchar	**code** var char	**comments** varchar	**state** var char	**url** var char	**belongs_to** varchar	**id_ext** varchar
Creat prin procedee informa-tice	fseqbootall	Bootstrapped sequences algorithm	---	---	---	---	phylipnew	fseqbootall

Tabel 5.1. Tabloul services

Se observa ca descrierea ACD a aplicatiei este mult mai saraca în detalii fata de reprezentarea specifica BioDesc. Pe de alta parte, în ACD programele sunt grupate în categorii, în functie de scopul lor, ceea ce reprezinta un avantaj pentru utilizatori. Aceasta din urma caracteristica nu va fi importata deoarece nu este prevazuta, momentan, în BioDesc.

- **Intrarile, iesirile si alti parametri**

Tinând cont de faptul ca diferitele tipuri de date sunt caracterizate de atribute specifice, în continuare este exemplificat importul parametrilor si calificatorilor de tipuri diferite din fseqbootall.

1) seqset: sequence [
parameter: "Y"
type: "gapany"
aligned: "Y"
]

2) properties: categories [
additional: "Y"
characters: ""
information: "File of input categories"
nullok: "Y"
size: "1"

```
        length: "$(sequence.length)"
  ]

3) list: datatype [
        additional: "y"
        minimum: "1"
        maximum: "1"
        header: "Datatype"
        values: "s:Molecular sequences; m:Discrete Morphology;
                r:Restriction Sites; g:Gene Frequencies"
        information: "Choose the datatype"
        default: "s"
  ]

  4) toggle: regular [
        additional: "@( $(test) == { b | j } )"
        information: "Altered sampling fraction"
        default: "N"
    ]

   5) boolean: enzymes [
        additional: "@($(datatype) == r)"
        information: "Is the number of enzymes present in input file"
        default: "N"
    ]

   6) float: fracsample [
        additional: "@(!$(regular))"
        information: "Samples as percentage of sites"
        default: "100.0"
        minimum: "0.1"
        maximum: "100.0"
    ]

   7) integer: blocksize [
        information: "Block size for bootstraping"
        additional: "@($(test) == b)"
        default: "1"
        minimum: "1"
    ]

   8) outfile: outfile [
        parameter: "Y"
        knowntype: "seqbootseq output"
        information: "Phylip seqboot_seq program output file"
    ]
```

id varchar	**service_id** varchar	**name** varchar	**description** varchar	**status** var char	**updatable** boolean	**visible** boolean	**comments** varchar	**depend** varchar	**order** int	**bio_dep** var char	**position** int
---	din tabloul services	sequence	---	---	---	---	---	---	---	---	---
---	din tabloul services	categories	Files of input categories	---	---	---	---	---	---	---	---
---	din tabloul services	datatype	Datatype	---	---	---	---	---	---	---	---
---	din tabloul services	regular	Alter sampling fraction	---	---	---	---	((test) = { b| j}	---	---	---
---	din tabloul services	enzymes	Is the number...	---	---	---	---	datatype = r	---	---	---
---	din tabloul services	fracsamples	Samples as...	---	---	---	---	regular = "Y"	---	---	---
---	din tabloul services	blocksize	Blocksize...	---	---	---	---	test = b	---	---	---
---	din tabloul services	outfile	Pylip seqboot...	---	---	---	---	---	---	---	---

Tabel 5.2. Tabloul parameter

Base varchar	**value** varchar	**nature of value** varchar	**nature of reference** varchar	**code** var char	**bio_dep** var char	**dep** varchar	**bio_restrictions** varchar	**restrictions** varchar	**formats** varchar	**comments** varchar
---	---	---	---	---	---	---	---	---	---	---
---	---	---	---	---	---	---	---	---	---	---
---	Molecular sequence, ...	---	---	---	---	---	---	---	---	---
---	---	---	---	---	---	---	---	---	---	---
---	---	---	---	---	---	---	---	---	---	---
---	---	---	---	---	---	---	---	---	---	---
---	---	---	---	---	---	---	---	---	---	---
---	---	---	---	---	---	---	---	---	---	---

Tabel 5.3. Tabelul types

➔ Daca în BioDesc datele sunt structurate în program, parametrii si tipuri, în ACD nu întâlnim o structura de tip: program, parametri. Din punctul de vedere al descrierii parametrilor pot fi importate **denumirea**, **descrierea** si **conditiile de aparitie în interfata cu utilizatorul**.

➔ Din rândul atributelelor globale ce caracterizeaza parametrii si calificatorii ACD, doar ***information*** si ***additional*** pot fi importate în tabloul **parameters** al bazei de date BioDesc, deoarece sunt analoagele variabilelor ***description*** si ***bio_dep*** (depedente biologice).

➔ Atributele globale, exceptând *information, additional* si *default*, sunt elemente caracteristice interfetei proprii ACD, fara corespondent în BioDesc.

➔ Atributele specifice reprezinta, de cele mai multe ori, conditii asupra valorilor pe care le pot avea parametrii si calificatorii: valoare minima, valoare maxima, lungime, numarul de alegeri posibile în cazul meniului list. Astfel de caracteristici nu sunt prevazute în BioDesc si deci nu pot fi importate.

➔ Valorile diferitelor atribute ACD trebuie alese în conformitate cu tipurile declarate în tabelele din Anexa 1.

➔ Variabila **value** din BioDesc este singura variabila, la nivelul **type**, care are corespondent în reprezentarea ACD si aceasta corespondenta este valabila numai pentru parametrii din categoria **Selection lists.**

➔ Tabloul **types** al bazei de date contine, pe lânga elementele din tabelul 5.3., si variabilele de tip boolean: ***constant, default, iscode, isreference***.

➔ În BioDesc problematica valorii implicite a unui parametru este exprimata sub forma unei variabile de tip boolean caracteristica tipului parametrului (variablia „is default" care poate avea valorile „true" sau „false"). În ACD valoarea implicita este exprimata direct cu ajutorul atributului global ***default***. Acest atribut este de tip sir de caractere (a se vedea exemplele de mai sus).

➔ ***constant, iscode, isreference*** sunt strict legate de structura BioDesc si nu au corespondent în ACD.

Capitolul 6. Concluzii

Lucrarea „Componete de indexare a resurselor bioinformatice, în cadrul proiectului BioSide” descrie modul în care pot fi utilizate resursele bioinformatice, cu ajutorul platformei BioSide, precum si îmbunatatirea aplicatiei software BioDesc, responsabila cu descrierea acestor resurse.

Partea de dezvoltare a proiectului presupune implementarea completa si riguroasa a facilitatii de import – export între XML si baza de date BioDesc. Aceasta facilitate este adaptata versiunii actuale a bazei de date.

Partea de analiza are în vedere integrarea programelor din ansamblul EMBOSS în baza de date BioDesc. În urma comparatiei ACD versus BioDesc, s-a observat ca gramaticile corespunzatoare celor doua reprezentari se suprapun într-o proportie suficient de mare încât sa fie posibil importul programelor EMBOSS în BioSide. Exista totusi câteva diferente semnificative între descrierile ACD si BioDesc, dar care pot fi solutionate pe cale informatica.

Pe lânga intergrarea în BioDesc a programelor din ansamblul EMBOSS, în perspectiva se urmareste si înglobarea programelor din ansamblul Pise. Dupa o analiza a gramaticii propuse în cadrul descrierii Pise, etapa urmatoare consta în crearea unei noi baze de date adaptata celor trei reprezentari: BioDesc, EMBOSS si Pise. Noua baza de date trebuie sa tina cont de necesitatile de diferentiere a programelor (aplicatiilor) de serviciile bioinformatice care le corespund, de nevoia de distinctie între descrierile datelor cu caracter biologic si a celor de natura informatica, de diferentierea datelor care au rol în cadrul interfetei de cele care descriu comportamentul programului.

Ca o consecinta a modificarii structurii bazei de date, se impune adaptarea interfetei BioDesc în acord cu criteriile enuntate anterior.

Proiectul BioSide reprezinta un pas în dezvoltarea domeniului bioinformaticii, un exemplu concret al unei simbioze perfecte între doua ramuri diferite ale stiintei, biologia si informatica, fiind un posibil punct de plecare pentru cercetarile ulterioare, în domeniul bioinformaticii cât si în domeniile adiacente, cum ar fi medicina sau inteligenta artificiala.

Capitolul 7. Bibliografie

- Deparatamentul LUSSI „Description de ressources – point du départ”
- Departamentul LUSSI, „BioSide Laussane”
- Departamentul LUSSI, „Doc Anvar”
- R. Elmasri & S. Navathe, „Conception et architecture des bases de données”, 4° édition, Pearson education France, 2004
- http://bioweb.pasteur.fr/seqanal/phylogeny/phylip-fr.html
- http://emboss.sourceforge.net/what/#Overview
- http://emboss.sourceforge.net/developers/acd/
- http://emboss.sourceforge.net/developers/developers_course/schedule.html
- http://emboss.sourceforge.net/docs/themes/SequenceFormats.html#not
- http://www.es.embnet.org/Doc/EMBOSS/Apps/index.html
- http://evolution.genetics.washington.edu/phylip/doc/protdist.html
- http://evolution.genetics.washington.edu/phylip/doc/seqboot.html
- http://www.memoireonline.com/01/06/68/gestionnaire-processus-bioinformatique.html
- http://www.pasteur.fr/~letondal/conf/nesc-pp.pdf
- http://relis.uvvg.ro/Cursuri/BdDR.html
- http://www.roseindia.net/bioinformatics/

Abrevieri

ACD - AJAX Command Definition

ADN (DNA) - Acid DezoxiriboNucleic

ARN (RNA) - Acid RiboNucleic

DTD - Document Type Definition

EMBOSS - the European Molecular Open Software Suite

ENST - École Nationale Supérieure de Télécommunication Bretagne

GUI - Graphical User Interface

HTML - Hyper Text Markup Language

LUSSI - Logique des Usages, Sciences Sociales et sciences de l'Information

MVC - Model-View-Controller

Pise - Pasteur Institute Software Environment

SGML - Standard Generalized Markup Language

SQL - Structured Querry Language

SVG - Scalable Vector Graphics

UML - Unified Modeling Language

W3C - World Wide Web Consortium

XML - eXtensible Markup Language

Ex. - Exemplu

Obs. - Observatie

Anexe

Anexa 1

Tabel 1. Grupuri de programe predefinite în ACD

Top Level	Second Level	Description
Acd		ACD file utilities
Alignment	Consensus	Merging sequences to make a consensus
	Differences	Finding differences between sequences
	Dot_plots	Dot plot sequence comparisons
	Global	Global sequence alignment
	Local	Local sequence alignment
	Multiple	Multiple sequence alignment
Display		Publication-quality display
Edit		Sequence editing
Enzyme_Kinetics		Enzyme kinetics calculations
Feature_tables		Manipulation and display of sequence annotation
HMM		Hidden Markov Model analysis
Information		Information and general help for users
Menus		Menu interface(s)
Nucleic	2D_structure	Nucleic acid secondary structure
	Codon_usage	Codon usage analysis
	Composition	Composition of nucleotide sequences
	CpG_islands	CpG island detection and analysis
	Gene_finding	Predictions of genes and other genomic features
	Motifs	Nucleic acid motif searches
	Mutation	Nucleic acid sequence mutation
	Profiles	Nucleic acid profile generation and searching
	Primers	Primer prediction
	Repeats	Nucleic acid repeat detection
	RNA_folding	RNA folding methods and analysis
	Restriction	Restriction enzyme sites in nucleotide sequences
	Transcription	Transcription factors, promoters and terminator prediction
	Translation	Translation of nucleotide sequence to protein sequence
Phylogeny	Consensus	Phylogenetic consensus methods
	Continuous_characters	Phylogenetic continuous character methods
	Discrete_characters	Phylogenetic discrete character methods
	Distance_matrix	Phylogenetic distance matrix methods
	Gene_frequencies	Phylogenetic gene frequency methods
	Molecular_sequence	Phylogenetic tree drawing methods

	Tree_drawing	Phylogenetic molecular sequence methods
	Misc	Phylogenetic other tools
Protein	2D_structure	Protein secondary structure
	3D_structure	Protein tertiary structure
	Composition	Composition of protein sequences
	Motifs	Protein motif searches
	Mutation	Protein sequence mutation
	Profiles	Protein profile generation and searching
Test		Testing tools, not for general use.
Utils	Database_creation	Database installation
	Database_indexing	Database indexing
	Misc	Utility tools

Tabel 2. Tipuri de date / obiecte în ACD

Data type / Object	**Description**	**Calculated Attributes**	**Specific Attributes**	**Command Line Qualifiers**
All data types				
	All data types		additional: "N" code: "" comment: "" default: "" expected: "" help: "" information: "" knowntype: "" missing: "N" needed: "y" outputmodifier: "N" parameter: "N" prompt: "" qualifier: "" relations: "" standard: "N" style: "" template: "" valid: ""	
Simple types				
array	List of floating point numbers		minimum: *(-FLT_MAX)* maximum: *(FLT_MAX)* increment: 0 precision: 0 warnrange: Y size: 1 sum: 1.0 sumtest: Y	

			tolerance: 0.01	
boolean	Boolean value Yes/No			
float	Floating point number		minimum: *(-FLT_MAX)* maximum: *(FLT_MAX)* increment: 1.0 precision: 3 warnrange: Y	
integer	Integer		minimum: *(INT_MIN)* maximum: *(INT_MAX)* increment: 0 warnrange: Y	
range	Sequence range		minimum: 1 maximum: *(INT_MAX)* size: 0 minsize: 0	
string	String value	length (integer)	minlength: 0 maxlength: *(INT_MAX)* pattern: "" upper: N lower: N word: N	
toggle	Toggle value Yes/No			
Input types				
codon	Codon usage file in EMBOSS data path		name: "Ehum.cut" nullok: N	format: ""
cpdb	Clean PDB file		nullok: N	format: ""
datafile	Data file		name: "" extension: "" directory: "" nullok: N	
directory	Directory		fullpath: N nulldefault: N nullok: N extension: ""	
dirlist	Directory with files		fullpath: N nullok: N extension: ""	
discretestates	Discrete states file		length: 0 size: 1 characters: "01" nullok: N	
distances	Distance matrix	distancesize (integer) replicates (boolean) hasmissing (boolean)	size: 1 nullok: N missval: N	

features	Readable feature table	fbegin (integer) fend (integer) flength (integer) fprotein (boolean) fnucleic (boolean) fname (string) fsize (string)	type: "" nullok: N	fformat: "" fopenfile: "" fask: "N" fbegin: "0" fend: "0" freverse: "N"
filelist	Comma-separated file list		nullok: N	
frequencies	Frequency value(s)	freqlength (integer) freqsize (integer) freqloci (integer) freqgenedata (boolean) freqcontinuous (boolean) freqwithin (boolean)	length: 0 size: 1 continuous: N genedata: N within: N nullok: N	
infile	Input file		nullok: N	
matrix	Comparison matrix file in EMBOSS data path		pname: "EBLOSUM62" nname: "EDNAFULL" protein: Y	
matrixf	Comparison matrix file in EMBOSS data path		pname: "EBLOSUM62" nname: "EDNAFULL" protein: Y	
pattern	Property value(s)		minlength: 1 maxlength: *(INT_MAX)* maxsize: *(INT_MAX)* upper: N lower: N type: "string"	pformat: "" pmismatch: "" pname: ""
properties	Property value(s)	propertylength (integer) propertysize (integer)	length: 0 size: 1 characters: "" nullok: N	
regexp	Regular expression pattern	length (integer)	minlength: 1 maxlength: *(INT_MAX)* maxsize: *(INT_MAX)* upper: N lower: N type: "string"	pformat: "" pname: ""
scop	Clean PDB file		nullok: N	format: ""
sequence	Readable sequence	begin (integer) end (integer) length (integer) protein (boolean) nucleic (boolean)	type: "" features: N entry: N nullok: N	sbegin: "0" send: "0" sreverse: "N" sask: "N" snucleotide: "N"

		name (string) usa (string)		sprotein: "N" slower: "N" supper: "N" sformat: "" sdbname: "" sid: "" ufo: "" fformat: "" fopenfile: ""
seqall	Readable sequence(s)	begin (integer) end (integer) length (integer) protein (boolean) nucleic (boolean) name (string) usa (string)	type: "" features: N entry: N minseqs: 1 maxseqs: *(INT_MAX)* nullok: N	sbegin: "0" send: "0" sreverse: "N" sask: "N" snucleotide: "N" sprotein: "N" slower: "N" supper: "N" sformat: "" sdbname: "" sid: "" ufo: "" fformat: "" fopenfile: ""
seqset	Readable set of sequences	begin (integer) end (integer) length (integer) protein (boolean) nucleic (boolean) name (string) usa (string) totweight (float) count (integer)	type: "" features: N aligned: N minseqs: 1 maxseqs: *(INT_MAX)* nulldefault: N nullok: N	sbegin: "0" send: "0" sreverse: "N" sask: "N" snucleotide: "N" sprotein: "N" slower: "N" supper: "N" sformat: "" sdbname: "" sid: "" ufo: "" fformat: "" fopenfile: ""
seqsetall	Readable sets of sequences	begin (integer) end (integer) length (integer) protein (boolean) nucleic (boolean) name (string) usa (string) totweight (float) count (integer) multicount (integer)	type: "" features: N aligned: N minseqs: 1 maxseqs: *(INT_MAX)* minsets: 1 maxsets: *(INT_MAX)* nulldefault: N nullok: N	sbegin: "0" send: "0" sreverse: "N" sask: "N" snucleotide: "N" sprotein: "N" slower: "N" supper: "N" sformat: "" sdbname: "" sid: "" ufo: "" fformat: "" fopenfile: ""
tree	Phylogenetic tree	treecount (integer) speciescount (integer) haslengths (boolean)	size: 0 nullok: N	

Selection lists types				
list	Choose from menu list of values		minimum: 1 maximum: 1 button: N casesensitive: N header: "" delimiter: ";" codedelimiter: ":" values: ""	
selection	Choose from selection list of values		minimum: 1 maximum: 1 button: N casesensitive: N header: "" delimiter: ":" values: ""	
Output types				
align	Alignment output file		type: "" taglist: "" minseqs: 1 maxseqs: *(INT_MAX)* multiple: N nulldefault: N nullok: N	aformat: "" aextension: "" adirectory: "" aname: "" awidth: "0" aaccshow: "N" adesshow: "N" ausashow: "N" aglobal: "N"
featout	Writeable feature table		name: "" extension: "" type: "" multiple: N nulldefault: N nullok: N	offormat: "" ofopenfile: "" ofextension: "" ofdirectory: "" ofname: "" ofsingle: "N"
outcodon	Codon usage file		name: "" extension: "" nulldefault: N nullok: N	odirectory: "" oformat: ""
outcpdb	Cleaned PDB file		nulldefault: N nullok: N	
outdata	Formatted output file		type: "" nulldefault: N nullok: N	odirectory: "" oformat: ""
outdir	Output directory		fullpath: N nulldefault: N nullok: N extension: ""	
outdiscrete	Discrete states file		nulldefault: N nullok: N	odirectory: "" oformat: ""
outdistance	Distance matrix		nulldefault: N nullok: N	
outfile	Output file		name: "" extension: "" append: N nulldefault: N	odirectory: ""

			nullok: N	
outfileall	Multiple output files		name: "" extension: "" nulldefault: N nullok: N	odirectory: ""
outfreq	Frequency value(s)		nulldefault: N nullok: N	odirectory: "" oformat: ""
outmatrix	Comparison matrix file		nulldefault: N nullok: N	odirectory: "" oformat: ""
outmatrixf	Comparison matrix file		nulldefault: N nullok: N	odirectory: "" oformat: ""
outproperties	Property value(s)		nulldefault: N nullok: N	odirectory: "" oformat: ""
outscop	Scop entry		nulldefault: N nullok: N	odirectory: "" oformat: ""
outtree	Phylogenetic tree		name: "" extension: "" nulldefault: N nullok: N	odirectory: "" oformat: ""
report	Report output file		type: "" taglist: "" multiple: N precision: 3 nulldefault: N nullok: N	rformat: "" rname: "" rextension: "" rdirectory: "" raccshow: "N" rdesshow: "N" rscoreshow: "Y" rusashow: "N" rmaxall: "0" rmaxseq: "0"
seqout	Writeable sequence		name: "" extension: "" features: N type: "" nulldefault: N nullok: N	osformat: "" osextension: "" osname: "" osdirectory: "" osdbname: "" ossingle: "N" oufo: "" offormat: "" ofname: "" ofdirectory: ""
seqoutall	Writeable sequence(s)		name: "" extension: "" features: N type: "" minseqs: 1 maxseqs: *(INT_MAX)* nulldefault: N nullok: N	osformat: "" osextension: "" osname: "" osdirectory: "" osdbname: "" ossingle: "N" oufo: "" offormat: "" ofname: "" ofdirectory: ""
seqoutset	Writeable sequences		name: "" extension: "" features: N type: ""	osformat: "" osextension: "" osname: "" osdirectory: ""

			minseqs: 1 maxseqs: *(INT_MAX)* nulldefault: N nullok: N aligned: N	osdbname: "" ossingle: "N" oufo: "" offormat: "" ofname: "" ofdirectory: ""
Graphics types				
graph	Graph device for a general graph		nulldefault: N nullok: N	gprompt: "N" gdesc: "" gtitle: "" gsubtitle: "" gxtitle: "" gytitle: "" goutfile: "" gdirectory: ""
xygraph	Graph device for a 2D graph		multiple: 1 nulldefault: N nullok: N	gprompt: "N" gdesc: "" gtitle: "" gsubtitle: "" gxtitle: "" gytitle: "" goutfile: "" gdirectory: ""

Tabel 3. Conventii recomandate pentru atribuirea numelor

Name	**Datatype**	**Usage**
sequence	sequence	primary input sequence, generally required
outseq	outseq	primary output sequence, generally required, generally should default to the primary input sequence name, extension defaults to the name of the output sequence format.
outfile	outfile	primary output non-sequence results file, generally required. The file extension should be allowed to default to the application name.
data	infile	primary auxiliary input data file, generally optional
minlen	int	minimal length of sequence feature to be found
maxlen	int	maximum length of sequence feature to be found
wordsize	int	word size for hash tables etc. generally minimum=2 for protein, 4 for DNA
window	int	window length for calculating dotplots/features/etc.
shift	int	amount by which window is shifted in each iteration
consensus	bool	flag for whether consensus sequence should be output
gap	float	gap penalty
gapext	float	gap extension penalty
from	int	position of start of input sequence to specify for an operation (e.g. deletion), defaults to start of sequence, minimum value = 1, maximum value = sequence length
to	int	position of end of input sequence to specify for an operation (e.g.: deletion), defaults to the 'from' value, minimum value = 'from', value, maximum = sequence length.
threshold	float/int	threshold for various operations
left	bool	operation should be done at the start of the sequence

right	bool	operation should be done at the end of the sequence
pattern	string	pattern to search for in sequence
patterns	infile	file of patterns to search for in sequence

Tabel 4. Atribute specifice pentru tipurile de date simple (Simple)

Data type	**Attribute definition**	**Description**
array	minimum: *float*	Minimum value Default: *(-FLT_MAX)*
	maximum: *float*	Maximum value Default: *(FLT_MAX)*
	increment: *float*	(Not used by ACD) Increment for GUIs Default: 0
	precision: *integer*	(Not used by ACD) Floating precision for GUIs Default: 0
	warnrange: *Y/N*	Warning if values are out of range Default: Y
	size: *integer*	Number of values required Default: 1
	sum: *float*	Total for all values Default: 1.0
	sumtest: *Y/N*	Test sum of all values Default: Y
	tolerance: *float*	Tolerance (sum +/- tolerance) of the total Default: 0.01
float	minimum: *float*	Minimum value Default: *(-FLT_MAX)*
	maximum: *float*	Maximum value Default: *(FLT_MAX)*
	increment: *float*	(Not used by ACD) Increment for GUIs Default: 1.0
	precision: *integer*	Precision for printing values Default: 3
	warnrange: *Y/N*	Warning if values are out of range Default: Y
integer	minimum: *integer*	Minimum value Default: *(INT_MIN)*
	maximum: *integer*	Maximum value Default: *(INT_MAX)*
	increment: *integer*	(Not used by ACD) Increment for GUIs Default: 0
	warnrange: *Y/N*	Warning if values are out of range Default: Y
range	minimum: *integer*	Minimum value Default: 1
	maximum: *integer*	Maximum value Default: *(INT_MAX)*

	size: *integer*	Exact number of values required Default: 0
	minsize: *integer*	Minimum number of values required Default: 0
string	minlength: *integer*	Minimum length Default: 0
	maxlength: *integer*	Maximum length Default: *(INT_MAX)*
	pattern: *string*	Regular expression for validation Default: ""
	upper: *Y/N*	Convert to upper case Default: N
	lower: *Y/N*	Convert to lower case Default: N
	word: *Y/N*	Disallow whitespace in strings Default: N

Obs.: boolean si toggle nu detin atribute specifice.

Tabel 5. Atribute specifice pentru tipurile de date de intrare (Input)

Data type	**Attribute definition**	**Description**
codon	name: *string*	Codon table name Default: "Ehum.cut"
	nullok: *Y/N*	Can accept a null filename as 'no file' Default: N
cpdb	nullok: *Y/N*	Can accept a null filename as 'no file' Default: N
datafile	name: *string*	Default file base name Default: ""
	extension: *string*	Default file extension Default: ""
	directory: *string*	Default installed data directory Default: ""
	nullok: *Y/N*	Can accept a null filename as 'no file' Default: N
directory	fullpath: *Y/N*	Require full path in value Default: N
	nulldefault: *Y/N*	Defaults to 'no file' Default: N
	nullok: *Y/N*	Can accept a null filename as 'no file' Default: N
	extension: *string*	Default file extension Default: ""
dirlist	fullpath: *Y/N*	Require full path in value Default: N
	nullok: *Y/N*	Can accept a null filename as 'no file' Default: N
	extension: *string*	Default file extension Default: ""

discretestates	length: *integer*	Number of discrete state values per set Default: 0
	size: *integer*	Number of discrete state set Default: 1
	characters: *string*	Allowed discrete state characters (default is '01' for binary characters) Default: "01"
	nullok: *Y/N*	Can accept a null filename as 'no file' Default: N
distances	size: *integer*	Number of rows Default: 1
	nullok: *Y/N*	Can accept a null filename as 'no file' Default: N
	missval: *integer*	Can have missing values (replicates zero) Default: N
features	type: *string*	Feature type (protein, nucleotide, etc.) Default: ""
	nullok: *Y/N*	Can accept a null filename as 'no file' Default: N
filelist	nullok: *Y/N*	Can accept a null filename as 'no file' Default: N
frequencies	length: *integer*	Number of frequency loci/values per set Default: 0
	size: *integer*	Number of frequency sets Default: 1
	continuous: *Y/N*	Continuous character data only Default: N
	genedata: *Y/N*	Gene frequency data only Default: N
	within: *Y/N*	Continuous data for multiple individuals Default: N
	nullok: *Y/N*	Can accept a null filename as 'no file' Default: N
infile	nullok: *Y/N*	Can accept a null filename as 'no file' Default: N
matrix	pname: *string*	Default name for protein matrix Default: "EBLOSUM62"
	nname: *string*	Default name for nucleotide matrix Default: "EDNAFULL"
	protein: *Y/N*	Protein matrix Default: Y
matrixf	pname: *string*	Default name for protein matrix Default: "EBLOSUM62"
	nname: *string*	Default name for nucleotide matrix Default: "EDNAFULL"
	protein: *Y/N*	Protein matrix Default: Y
pattern	minlength: *integer*	Minimum pattern length Default: 1
	maxlength: *integer*	Maximum pattern length

		Default: *(INT_MAX)*
	maxsize: *integer*	Maximum number of patterns Default: *(INT_MAX)*
	upper: *Y/N*	Convert to upper case Default: N
	lower: *Y/N*	Convert to lower case Default: N
	type: *string*	Type (nucleotide, protein) Default: "string"
properties	length: *integer*	Number of property values per set Default: 0
	size: *integer*	Number of property sets Default: 1
	characters: *string*	Allowed property characters (default is " for all characters) Default: ""
	nullok: *Y/N*	Can accept a null filename as 'no file' Default: N
regexp	minlength: *integer*	Minimum pattern length Default: 1
	maxlength: *integer*	Maximum pattern length Default: *(INT_MAX)*
	maxsize: *integer*	Maximum number of patterns Default: *(INT_MAX)*
	upper: *Y/N*	Convert to upper case Default: N
	lower: *Y/N*	Convert to lower case Default: N
	type: *string*	Type (string, nucleotide, protein) Default: "string"
scop	nullok: *Y/N*	Can accept a null filename as 'no file' Default: N
sequence	type: *string*	Input sequence type (protein, gapprotein, etc.) Default: ""
	features: *Y/N*	Read features if any Default: N
	entry: *Y/N*	Read whole entry text Default: N
	nullok: *Y/N*	Can accept a null filename as 'no file' Default: N
seqall	type: *string*	Input sequence type (protein, gapprotein, etc.) Default: ""
	features: *Y/N*	Read features if any Default: N
	entry: *Y/N*	Read whole entry text Default: N
	minseqs: *integer*	Minimum number of sequences Default: 1
	maxseqs: *integer*	Maximum number of sequences Default: *(INT_MAX)*

	nullok: *Y/N*	Can accept a null filename as 'no file' Default: N
seqset	type: *string*	Input sequence type (protein, gapprotein, etc.) Default: ""
	features: *Y/N*	Read features if any Default: N
	aligned: *Y/N*	Sequences are aligned Default: N
	minseqs: *integer*	Minimum number of sequences Default: 1
	maxseqs: *integer*	Maximum number of sequences Default: *(INT_MAX)*
	nulldefault: *Y/N*	Defaults to 'no file' Default: N
	nullok: *Y/N*	Can accept a null filename as 'no file' Default: N
seqsetall	type: *string*	Input sequence type (protein, gapprotein, etc.) Default: ""
	features: *Y/N*	Read features if any Default: N
	aligned: *Y/N*	Sequences are aligned Default: N
	minseqs: *integer*	Minimum number of sequences Default: 1
	maxseqs: *integer*	Maximum number of sequences Default: *(INT_MAX)*
	minsets: *integer*	Minimum number of sequence sets Default: 1
	maxsets: *integer*	Maximum number of sequence sets Default: *(INT_MAX)*
	nulldefault: *Y/N*	Defaults to 'no file' Default: N
	nullok: *Y/N*	Can accept a null filename as 'no file' Default: N
tree	size: *integer*	Number of trees (0 means any number) Default: 0
	nullok: *Y/N*	Can accept a null filename as 'no file' Default: N

Tabel 6. Atribute specifice pentru tipurile de date de tip meniu (Selection lists)

Data type	Attribute definition	Description
list	minimum: *integer*	Minimum number of selections Default: 1
	maximum: *integer*	Maximum number of selections Default: 1
	button: *Y/N*	(Not used by ACD) Prefer checkboxes in GUI Default: N

	casesensitive: *Y/N*	Case sensitive Default: N
	header: *string*	Header description for list Default: ""
	delimiter: *string*	Delimiter for parsing values Default: ";"
	codedelimiter: *string*	Delimiter for parsing Default: ":"
	values: *string*	Codes and values with delimiters Default: ""
selection	minimum: *integer*	Minimum number of selections Default: 1
	maximum: *integer*	Maximum number of selections Default: 1
	button: *Y/N*	(Not used by ACD) Prefer radiobuttons in GUI Default: N
	casesensitive: *Y/N*	Case sensitive matching Default: N
	header: *string*	Header description for selection list Default: ""
	delimiter: *string*	Delimiter for parsing values Default: ":"
	values: *string*	Values with delimiters Default: ""

Tabel 7. Atribute specifice pentru tipurile de date de iesire (Output)

Data type	**Attribute definition**	**Description**
align	type: *string*	[P]rotein or [N]ucleotide Default: ""
	taglist: *string*	Extra tags to report Default: ""
	minseqs: *integer*	Minimum number of sequences Default: 1
	maxseqs: *integer*	Maximum number of sequences Default: *(INT_MAX)*
	multiple: *Y/N*	More than one alignment in one file Default: N
	nulldefault: *Y/N*	Defaults to 'no file' Default: N
	nullok: *Y/N*	Can accept a null filename as 'no file' Default: N
featout	name: *string*	Default base file name (use of -ofname preferred) Default: ""
	extension: *string*	Default file extension (use of -offormat preferred) Default: ""
	type: *string*	Feature type (protein, nucleotide, etc.) Default: ""

	multiple: *Y/N*	Features for multiple sequences Default: N
	nulldefault: *Y/N*	Defaults to 'no file' Default: N
	nullok: *Y/N*	Can accept a null UFO as 'no output' Default: N
outcodon	name: *string*	Default file name Default: ""
	extension: *string*	Default file extension Default: ""
	nulldefault: *Y/N*	Defaults to 'no file' Default: N
	nullok: *Y/N*	Can accept a null filename as 'no file' Default: N
outcpdb	nulldefault: *Y/N*	Defaults to 'no file' Default: N
	nullok: *Y/N*	Can accept a null filename as 'no file' Default: N
outdata	type: *string*	Data type Default: ""
	nulldefault: *Y/N*	Defaults to 'no file' Default: N
	nullok: *Y/N*	Can accept a null filename as 'no file' Default: N
outdir	fullpath: *Y/N*	Require full path in value Default: N
	nulldefault: *Y/N*	Defaults to 'no file' Default: N
	nullok: *Y/N*	Can accept a null filename as 'no file' Default: N
	extension: *string*	Default file extension Default: ""
outdiscrete	nulldefault: *Y/N*	Defaults to 'no file' Default: N
	nullok: *Y/N*	Can accept a null filename as 'no file' Default: N
outdistance	nulldefault: *Y/N*	Defaults to 'no file' Default: N
	nullok: *Y/N*	Can accept a null filename as 'no file' Default: N
outfile	name: *string*	Default file name Default: ""
	extension: *string*	Default file extension Default: ""
	append: *Y/N*	Append to an existing file Default: N
	nulldefault: *Y/N*	Defaults to 'no file' Default: N
	nullok: *Y/N*	Can accept a null filename as 'no file'

		Default: N
outfileall	name: *string*	Default file name Default: ""
	extension: *string*	Default file extension Default: ""
	nulldefault: *Y/N*	Defaults to 'no file' Default: N
	nullok: *Y/N*	Can accept a null filename as 'no file' Default: N
outfreq	nulldefault: *Y/N*	Defaults to 'no file' Default: N
	nullok: *Y/N*	Can accept a null filename as 'no file' Default: N
outmatrix	nulldefault: *Y/N*	Defaults to 'no file' Default: N
	nullok: *Y/N*	Can accept a null filename as 'no file' Default: N
outmatrixf	nulldefault: *Y/N*	Defaults to 'no file' Default: N
	nullok: *Y/N*	Can accept a null filename as 'no file' Default: N
outproperties	nulldefault: *Y/N*	Defaults to 'no file' Default: N
	nullok: *Y/N*	Can accept a null filename as 'no file' Default: N
outscop	nulldefault: *Y/N*	Defaults to 'no file' Default: N
	nullok: *Y/N*	Can accept a null filename as 'no file' Default: N
outtree	name: *string*	Default file name Default: ""
	extension: *string*	Default file extension Default: ""
	nulldefault: *Y/N*	Defaults to 'no file' Default: N
	nullok: *Y/N*	Can accept a null filename as 'no file' Default: N
report	type: *string*	[P]rotein or [N]ucleotide Default: ""
	taglist: *string*	Extra tag names to report Default: ""
	multiple: *Y/N*	Multiple sequences in one report Default: N
	precision: *integer*	Score precision Default: 3
	nulldefault: *Y/N*	Defaults to 'no file' Default: N
	nullok: *Y/N*	Can accept a null filename as 'no file' Default: N

seqout	name: *string*	Output base name (use of -osname preferred) Default: ""
	extension: *string*	Output extension (use of -osextension preferred) Default: ""
	features: *Y/N*	Write features if any Default: N
	type: *string*	Output sequence type (protein, gapprotein, etc.) Default: ""
	nulldefault: *Y/N*	Defaults to 'no file' Default: N
	nullok: *Y/N*	Can accept a null USA as 'no output' Default: N
seqoutall	name: *string*	Output base name (use of -osname preferred) Default: ""
	extension: *string*	Output extension (use of -osextension preferred) Default: ""
	features: *Y/N*	Write features if any Default: N
	type: *string*	Output sequence type (protein, gapprotein, etc.) Default: ""
	minseqs: *integer*	Minimum number of sequences Default: 1
	maxseqs: *integer*	Maximum number of sequences Default: *(INT_MAX)*
	nulldefault: *Y/N*	Defaults to 'no file' Default: N
	nullok: *Y/N*	Can accept a null USA as 'no output' Default: N
seqoutset	name: *string*	Output base name (use of -osname preferred) Default: ""
	extension: *string*	Output extension (use of -osextension preferred) Default: ""
	features: *Y/N*	Write features if any Default: N
	type: *string*	Output sequence type (protein, gapprotein, etc.) Default: ""
	minseqs: *integer*	Minimum number of sequences Default: 1
	maxseqs: *integer*	Maximum number of sequences Default: *(INT_MAX)*
	nulldefault: *Y/N*	Defaults to 'no file' Default: N
	nullok: *Y/N*	Can accept a null USA as 'no output' Default: N
	aligned: *Y/N*	Sequences are aligned Default: N

Tabel 8. Atribute specifice prntru tipurile de date grafice (Graph)

Data type	Attribute definition	Description
graph	nulldefault: *Y/N*	Defaults to 'no file' Default: N
	nullok: *Y/N*	Can accept a null graph type as 'no graph' Default: N
xygraph	multiple: *integer*	Number of graphs Default: 1
	nulldefault: *Y/N*	Defaults to 'no file' Default: N
	nullok: *Y/N*	Can accept a null graph type as 'no graph' Default: N

Tabel 9. Atribute calculate ce caracterizeaza tipurile de date

Data type	Calculated attributes	Description
distances	distancesize: *integer*	Number of distance rows Default:
	replicates: *Y/N*	Replicates data found in input Default:
	hasmissing: *Y/N*	Missing values found(replicates=N) Default:
features	fbegin: *integer*	Start of the features to be used Default: *(0 if unspecified)*
	fend: *integer*	End of the features to be used Default: *(0 if unspecified)*
	flength: *integer*	Total length of sequence (fsize is feature count) Default:
	fprotein: *Y/N*	Feature table is protein Default:
	fnucleic: *Y/N*	Feature table is nucleotide Default:
	fname: *string*	The name of the feature table Default: ""
	fsize: *string*	Integer, number of features Default: ""
frequencies	freqlength: *integer*	Number of frequency values per set Default:
	freqsize: *integer*	Number of frequency sets Default:
	freqloci: *integer*	Number of frequency loci Default:
	freqgenedata: *Y/N*	Gene frequency data Default:
	freqcontinuous: *Y/N*	Continuous frequency data Default:

	freqwithin: *Y/N*	Individual within species frequency data Default:
properties	propertylength: *integer*	Number of property values per set Default:
	propertysize: *integer*	Number of property sets Default:
regexp	length: *integer*	The length of the regular expression Default:
seqall	begin: *integer*	Start of the first sequence used Default:
	end: *integer*	End of the first sequence used Default:
	length: *integer*	Total length of the first sequence Default:
	protein: *Y/N*	Boolean, indicates if sequence is protein Default:
	nucleic: *Y/N*	Boolean, indicates if sequence is DNA Default:
	name: *string*	The name/ID/accession of the sequence Default: ""
	usa: *string*	The USA of the sequence Default: ""
seqset	begin: *integer*	The beginning of the selection of the sequence Default:
	end: *integer*	The end of the selection of the sequence Default:
	length: *integer*	The maximum length of the sequence set Default:
	protein: *Y/N*	Boolean, indicates if sequence set is protein Default:
	nucleic: *Y/N*	Boolean, indicates if sequence set is DNA Default:
	name: *string*	The name of the sequence set Default: ""
	usa: *string*	The USA of the sequence Default: ""
	totweight: *float*	Float, total sequence weight for a set Default:
	count: *integer*	Integer, number of sequences in the set Default:
seqsetall	begin: *integer*	The beginning of the selection of the sequence Default:
	end: *integer*	The end of the selection of the sequence Default:
	length: *integer*	The maximum length of the sequence set Default:
	protein: *Y/N*	Boolean, indicates if sequence set is protein Default:
	nucleic: *Y/N*	Boolean, indicates if sequence set is DNA

		Default:
	name: *string*	The name of the sequence set Default: ""
	usa: *string*	The USA of the sequence Default: ""
	totweight: *float*	Float, total sequence weight for each set Default:
	count: *integer*	Integer, number of sequences in each set Default:
	multicount: *integer*	Integer, number of sets of sequences Default:
sequence	begin: *integer*	Start of the sequence used Default:
	end: *integer*	End of the sequence used Default:
	length: *integer*	Total length of the sequence Default:
	protein: *Y/N*	Boolean, indicates if sequence is protein Default:
	nucleic: *Y/N*	Boolean, indicates if sequence is DNA Default:
	name: *string*	The name/ID/accession of the sequence Default: ""
	usa: *string*	The USA of the sequence Default: ""
string	length: *integer*	The length of the string Default:
tree	treecount: *integer*	Number of trees Default:
	speciescount: *integer*	Number of species Default:
	haslengths: *Y/N*	Branch lengths defined Default:

Tabel 10. Valorile posibile ale atributului *type* caracteristice tipului de date *sequence,* din categoria datelor de intrare

Value	**Type(s)**	**Gaps**	**Ambiguity codes**	**Conversions**	**Description**
any	Nucleotide or protein	Removed	Yes	'?'=>'X'	any valid sequence
gapany	Nucleotide or protein	Kept	Yes	'?'=>'X'	any valid sequence with gaps
dna	Nucleotide only	Removed	Yes	'?'=>'N' 'X'=>'N' 'U'=>'T'	DNA sequence
puredna	Nucleotide only	Removed	No	'U'=>'T'	DNA sequence, bases ACGT only
gapdna	Nucleotide only	Kept	Yes	'?'=>'N' 'X'=>'N' 'U'=>'T'	DNA sequence with gaps

gapdnaphylo	Nucleotide only	Kept	Yes	'U'=>'T'	DNA sequence with gaps and queries
rna	Nucleotide only	Removed	Yes	'?'=>'N' 'X'=>'N' 'T'=>'U'	RNA sequence
purerna	Nucleotide only	Removed	No	'T'=>'U'	RNA sequence, bases ACGU only
gaprna	Nucleotide only	Kept	Yes	'?'=>'N' 'X'=>'N' 'T'=>'U'	RNA sequence with gaps
gaprnaphylo	Nucleotide only	Kept	Yes	'T'=>'U'	RNA sequence with gaps and queries
nucleotide	Nucleotide only	Removed	Yes	'?'=>'N' 'X'=>'N'	nucleotide sequence
purenucleotide	Nucleotide only	Removed	No		nucleotide sequence, bases ACGTU only
gapnucleotide	Nucleotide only	Kept	Yes	'?'=>'N' 'X'=>'N'	nucleotide sequence with gaps
gapnucleotidephylo	Nucleotide only	Kept	Yes		nucleotide sequence with gaps and queries
protein	Protein only	Removed	Yes	'?'=>'X' '*'=>'X'	protein sequence
pureprotein	Protein only	Removed	No		protein sequence without BZ U X or *
stopprotein	Protein only	Removed	Yes	'?'=>'X'	protein sequence with possible stops
gapprotein	Protein only	Kept	Yes	'?'=>'X' '*'=>'X'	protein sequence with gaps
gapstopprotein	Protein only	Kept	Yes	'?'=>'X'	protein sequence with gaps and possible stops
gapproteinphylo	Protein only	Kept	Yes		protein sequence with gaps, stops and queries
proteinstandard	Protein only	Removed	Yes	'?'=>'X' '*'=>'X' 'U'=>'X' 'J'=>'X' 'O'=>'X'	protein sequence with no selenocysteine
stopproteinstandard	Protein only	Removed	Yes	'?'=>'X' 'U'=>'X' 'J'=>'X' 'O'=>'X'	protein sequence with a possible stop but no selenocysteine
gapproteinstandard	Protein only	Kept	Yes	'?'=>'X' '*'=>'X' 'U'=>'X' 'J'=>'X' 'O'=>'X'	protein sequence with gaps but no selenocysteine

Tabel 11. Calificatorii globali

Qualifier definition	**Description**
-acdlog	Turns on logging of ACD file processing
-acdpretty	Outputs a nicely formatted ACD file

-auto	Turns off any prompting of the user
-debug	Turns on debugging with ajDebug calls
-filter	Reads from stdin and writes to stdout and implies -auto
-stdout	Writes by default to stdout, but still prompts the user
-help	Will give usage information of this program. See also *-verbose* below.
-verbose	When used with *-help* also gives the associated qualifiers and the general qualifiers (this list)
-options	Program will also prompt for optional qualifiers.

Tabel 12. Variabilele de mediu

Environment variable	Associated global qualifier
EMBOSS_ACDLOG	-acdlog
EMBOSS_ACDPRETTY	-acdpretty
EMBOSS_AUTO	-auto
EMBOSS_DEBUG	-debug
EMBOSS_FILTER	-filter
EMBOSS_STDOUT	-stdout
EMBOSS_HELP	-help
EMBOSS_VERBOSE	-verbose
EMBOSS_OPTIONS	-options

Tabel 13. Calificatori specifici tipurilor de date de intrare (Input)

Data type	**Qualifier definition**	**Description**
codon	-format: *string*	Data format Default: ""
cpdb	-format: *string*	Data format Default: ""
features	-fformat: *string*	Features format Default: ""
	-fopenfile: *string*	Features file name Default: ""
	-fask: *Y/N*	Prompt for begin/end/reverse Default: N
	-fbegin: *integer*	Start of the features to be used Default: 0
	-fend: *integer*	End of the features to be used Default: 0
	-freverse: *Y/N*	Reverse (if DNA) Default: N
pattern	-pformat: *string*	File format Default: ""
	-pmismatch: *integer*	Pattern mismatch Default:
	-pname: *string*	Pattern base name

		Default: ""
regexp	-pformat: *string*	File format Default: ""
	-pname: *string*	Pattern base name Default: ""
scop	-format: *string*	Data format Default: ""
sequence	-sbegin: *integer*	Start of the sequence to be used Default: 0
	-send: *integer*	End of the sequence to be used Default: 0
	-sreverse: *Y/N*	Reverse (if DNA) Default: N
	-sask: *Y/N*	Ask for begin/end/reverse Default: N
	-snucleotide: *Y/N*	Sequence is nucleotide Default: N
	-sprotein: *Y/N*	Sequence is protein Default: N
	-slower: *Y/N*	Make lower case Default: N
	-supper: *Y/N*	Make upper case Default: N
	-sformat: *string*	Input sequence format Default: ""
	-sdbname: *string*	Database name Default: ""
	-sid: *string*	Entryname Default: ""
	-ufo: *string*	UFO features Default: ""
	-fformat: *string*	Features format Default: ""
	-fopenfile: *string*	Features file name Default: ""
seqall	-sbegin: *integer*	Start of each sequence to be used Default: 0
	-send: *integer*	End of each sequence to be used Default: 0
	-sreverse: *Y/N*	Reverse (if DNA) Default: N
	-sask: *Y/N*	Ask for begin/end/reverse Default: N
	-snucleotide: *Y/N*	Sequence is nucleotide Default: N
	-sprotein: *Y/N*	Sequence is protein Default: N
	-slower: *Y/N*	Make lower case Default: N

	-supper: *Y/N*	Make upper case Default: N
	-sformat: *string*	Input sequence format Default: ""
	-sdbname: *string*	Database name Default: ""
	-sid: *string*	Entryname Default: ""
	-ufo: *string*	UFO features Default: ""
	-fformat: *string*	Features format Default: ""
	-fopenfile: *string*	Features file name Default: ""
seqset	-sbegin: *integer*	Start of each sequence to be used Default: 0
	-send: *integer*	End of each sequence to be used Default: 0
	-sreverse: *Y/N*	Reverse (if DNA) Default: N
	-sask: *Y/N*	Ask for begin/end/reverse Default: N
	-snucleotide: *Y/N*	Sequence is nucleotide Default: N
	-sprotein: *Y/N*	Sequence is protein Default: N
	-slower: *Y/N*	Make lower case Default: N
	-supper: *Y/N*	Make upper case Default: N
	-sformat: *string*	Input sequence format Default: ""
	-sdbname: *string*	Database name Default: ""
	-sid: *string*	Entryname Default: ""
	-ufo: *string*	UFO features Default: ""
	-fformat: *string*	Features format Default: ""
	-fopenfile: *string*	Features file name Default: ""
seqsetall	-sbegin: *integer*	Start of each sequence to be used Default: 0
	-send: *integer*	End of each sequence to be used Default: 0
	-sreverse: *Y/N*	Reverse (if DNA) Default: N
	-sask: *Y/N*	Ask for begin/end/reverse

		Default: N
	-snucleotide: *Y/N*	Sequence is nucleotide Default: N
	-sprotein: *Y/N*	Sequence is protein Default: N
	-slower: *Y/N*	Make lower case Default: N
	-supper: *Y/N*	Make upper case Default: N
	-sformat: *string*	Input sequence format Default: ""
	-sdbname: *string*	Database name Default: ""
	-sid: *string*	Entryname Default: ""
	-ufo: *string*	UFO features Default: ""
	-fformat: *string*	Features format Default: ""
	-fopenfile: *string*	Features file name Default: ""

Tabel 14. Calificatori specifici tipurilor de date de iesire (Output)

Data type	**Qualifier definition**	**Description**
align	-aformat: *string*	Alignment format Default: ""
	-aextension: *string*	File name extension Default: ""
	-adirectory: *string*	Output directory Default: ""
	-aname: *string*	Base file name Default: ""
	-awidth: *integer*	Alignment width Default: 0
	-aaccshow: *Y/N*	Show accession number in the header Default: N
	-adesshow: *Y/N*	Show description in the header Default: N
	-ausashow: *Y/N*	Show the full USA in the alignment Default: N
	-aglobal: *Y/N*	Show the full sequence in alignment Default: N
featout	-offormat: *string*	Output feature format Default: ""
	-ofopenfile: *string*	Features file name Default: ""
	-ofextension: *string*	File name extension

		Default: ""
	-ofdirectory: *string*	Output directory Default: ""
	-ofname: *string*	Base file name Default: ""
	-ofsingle: *Y/N*	Separate file for each entry Default: N
outcodon	-odirectory: *string*	Output directory Default: ""
	-oformat: *string*	Output format specific to this data type Default: ""
outdata	-odirectory: *string*	Output directory Default: ""
	-oformat: *string*	Output format specific to this data type Default: ""
outdiscrete	-odirectory: *string*	Output directory Default: ""
	-oformat: *string*	Output format specific to this data type Default: ""
outfile	-odirectory: *string*	Output directory Default: ""
outfileall	-odirectory: *string*	Output directory Default: ""
outfreq	-odirectory: *string*	Output directory Default: ""
	-oformat: *string*	Output format specific to this data type Default: ""
outmatrix	-odirectory: *string*	Output directory Default: ""
	-oformat: *string*	Output format specific to this data type Default: ""
outmatrixf	-odirectory: *string*	Output directory Default: ""
	-oformat: *string*	Output format specific to this data type Default: ""
outproperties	-odirectory: *string*	Output directory Default: ""
	-oformat: *string*	Output format specific to this data type Default: ""
outscop	-odirectory: *string*	Output directory Default: ""
	-oformat: *string*	Output format specific to this data type Default: ""
outtree	-odirectory: *string*	Output directory Default: ""
	-oformat: *string*	Output format specific to this data type Default: ""
report	-rformat: *string*	Report format Default: ""

	-rname: *string*	Base file name Default: ""
	-rextension: *string*	File name extension Default: ""
	-rdirectory: *string*	Output directory Default: ""
	-raccshow: *Y/N*	Show accession number in the report Default: N
	-rdesshow: *Y/N*	Show description in the report Default: N
	-rscoreshow: *Y/N*	Show the score in the report Default: Y
	-rusashow: *Y/N*	Show the full USA in the report Default: N
	-rmaxall: *integer*	Maximum total hits to report Default: 0
	-rmaxseq: *integer*	Maximum hits to report for one sequence Default: 0
seqout	-osformat: *string*	Output seq format Default: ""
	-osextension: *string*	File name extension Default: ""
	-osname: *string*	Base file name Default: ""
	-osdirectory: *string*	Output directory Default: ""
	-osdbname: *string*	Database name to add Default: ""
	-ossingle: *Y/N*	Separate file for each entry Default: N
	-oufo: *string*	UFO features Default: ""
	-offormat: *string*	Features format Default: ""
	-ofname: *string*	Features file name Default: ""
	-ofdirectory: *string*	Output directory Default: ""
seqoutall	-osformat: *string*	Output seq format Default: ""
	-osextension: *string*	File name extension Default: ""
	-osname: *string*	Base file name Default: ""
	-osdirectory: *string*	Output directory Default: ""
	-osdbname: *string*	Database name to add Default: ""
	-ossingle: *Y/N*	Separate file for each entry

		Default: N
	-oufo: *string*	UFO features Default: ""
	-offormat: *string*	Features format Default: ""
	-ofname: *string*	Features file name Default: ""
	-ofdirectory: *string*	Output directory Default: ""
seqoutset	-osformat: *string*	Output seq format Default: ""
	-osextension: *string*	File name extension Default: ""
	-osname: *string*	Base file name Default: ""
	-osdirectory: *string*	Output directory Default: ""
	-osdbname: *string*	Database name to add Default: ""
	-ossingle: *Y/N*	Separate file for each entry Default: N
	-oufo: *string*	UFO features Default: ""
	-offormat: *string*	Features format Default: ""
	-ofname: *string*	Features file name Default: ""
	-ofdirectory: *string*	Output directory Default: ""

Tabel 15. Calificatori specifici tipurilor de date grafice (Graphics)

Data type	Qualifier definition	Description
graph	-gprompt: *Y/N*	Graph prompting Default: N
	-gdesc: *string*	Graph description Default: ""
	-gtitle: *string*	Graph title Default: ""
	-gsubtitle: *string*	Graph subtitle Default: ""
	-gxtitle: *string*	Graph x axis title Default: ""
	-gytitle: *string*	Graph y axis title Default: ""
	-goutfile: *string*	Output file for non interactive displays Default: ""

	-gdirectory: *string*	Output directory Default: ""
xygraph	-gprompt: *Y/N*	Graph prompting Default: N
	-gdesc: *string*	Graph description Default: ""
	-gtitle: *string*	Graph title Default: ""
	-gsubtitle: *string*	Graph subtitle Default: ""
	-gxtitle: *string*	Graph x axis title Default: ""
	-gytitle: *string*	Graph y axis title Default: ""
	-goutfile: *string*	Output file for non interactive displays Default: ""
	-gdirectory: *string*	Output directory Default: ""

Anexa 2

Continutul fisierului fseqbootall.acd

```
application: fseqbootall [
        documentation: "Bootstrapped sequences
algorithm"
        groups: "Phylogeny:Molecular sequence"
        embassy: "phylipnew"
]

section: input [
        information: "Input section"
        type: "page"
]

  seqset: infile [
        parameter: "Y"
        type: "gapany"
        aligned: "Y"
  ]

  properties: categories [
        additional: "Y"
        characters: ""
        information: "File of input categories"
        nullok: "Y"
        size: "1"
        length: "$(infile.length)"
  ]

  properties: mixfile [
        additional: "Y"
        characters: ""
        information: "File of mixtures"
        nullok: "Y"
        size: "1"
        length: "$(infile.length)"
  ]

  properties: ancfile [
        additional: "Y"
        characters: ""
        information: "File of ancestors"
        nullok: "Y"
        size: "1"
        length: "$(infile.length)"
  ]

  properties: weights [
        additional: "Y"
        characters: "01"
        information: "Weights file"
        help: "Weights file"
        length: "$(infile.length)"
        nullok: "Y"
  ]

  properties: factorfile [
        additional: "Y"
        information: "Factors file"
        nullok: "Y"
        length: "$(infile.length)"
        size: "1"
  ]

endsection: input

section: additional [
        information: "Additional section"
        type: "page"
]

  list: datatype [
        additional: "y"
        minimum: "1"
        maximum: "1"
        header: "Datatype"
        values: "s:Molecular sequences; m:Discrete
Morphology;
        r:Restriction Sites; g:Gene Frequencies"
        information: "Choose the datatype"
        default: "s"
  ]

  list: test [
        additional: "y"
        minimum: "1"
        maximum: "1"
        header: "Test"
        values: "b:Bootstrap; j:Jackknife; c:Permute
species for each
        character; o:Permute character order;
s:Permute within species;
        r:Rewrite data"
        information: "Choose test"
        default: "b"
  ]

  toggle: regular [
        additional: "@( $(test) == { b | j } )"
        information: "Altered sampling fraction"
        default: "N"
  ]

  float: fracsample [
```

```
        additional: "@(!$(regular))"
        information: "Samples as percentage of
sites"
        default: "100.0"
        minimum: "0.1"
        maximum: "100.0"
  ]

  list: rewriteformat [
        additional:
"@(@($(test)==r)&&@($(datatype)==s))"
        minimum: "1"
        maximum: "1"
        header: "test"
        values: "p:PHYLIP; n:NEXUS; x:XML"
        information: "Output format"
        default: "p"
  ]

  list: seqtype [
        additional: "@( @( $(datatype) == s ) & @(
$(rewriteformat)
                == {n | x} ))"
        minimum: "1"
        maximum: "1"
        header: "test"
        values: "d:dna; p:protein; r:rna"
        information: "Output format"
        default: "d"
  ]

  list: morphseqtype [
        additional: "@( @( $(datatype) == m ) &
@( $(test) == r ))"
        minimum: "1"
        maximum: "1"
        header: "Output format"
        values: "p:PHYLIP; n:NEXUS"
        information: "Output format"
        default: "p"
  ]

  integer: blocksize [
        information: "Block size for bootstraping"
        additional: "@($(test) == b)"
        default: "1"
        minimum: "1"
  ]

  integer: reps [
        additional: "@($(test) != r)"
        information: "How many replicates"
        minimum: "1"
        default: "100"
  ]

  list: justweights [
        additional: "@( $(test) == { b | j } )"
        minimum: "1"
        maximum: "1"
        header: "Write out datasets or just weights"
        values: "d:Datasets; w:Weights"
        information: "Write out datasets or just
weights"
        default: "d"
  ]

  boolean: enzymes [
        additional: "@($(datatype) == r)"
        information: "Is the number of enzymes
present in input
                file"
        default: "N"
  ]

  boolean: all [
        additional: "@($(datatype) == g)"
        information: "All alleles present at each
locus"
        default: "N"
  ]

  integer: seed [
        additional: "@($(test) != r)"
        information: "Random number seed between
1 and 32767 (must
                be odd)"
        minimum: "1"
        maximum: "32767"
        default: "1"
  ]

endsection: additional

section: output [
        information: "Output section"
        type: "page"
]

  outfile: outfile [
        parameter: "Y"
        knowntype: "seqboot output"
        information: "Phylip seqboot program output
file"
  ]

  boolean: printdata [
        additional: "Y"
        default: "N"
        information: "Print out the data at start of
run"
  ]
```

boolean: dotdiff [
	additional: "$(printdata)"
	default: "Y"
	information: "Use dot-differencing"
]

boolean: progress [
	additional: "Y"
	default: "Y"
	information: "Print indications of progress of run"
  ]

endsection: output

Anexa 3

Tabel 1. Tipuri de formate ale secventelor din categoria de date Input

Input Format	Comments
abi	ABI trace file format. This is the format of file produced by ABI sequencing machines. It contains the 'trace data' i.e. the probabilities of the 4 bases along the sequencing run, together with the sequence, as deduced from that data. The sequence information is what is normally read in and used by EMBOSS programs, although the trace data is available and may be utilised by some specialised EMBOSS programs. The code for this is heavily based on David Mathog's fortran library with a description of ABI trace file format (abi.txt): ftp://saf.bio.caltech.edu/pub/software/molbio/abitools.zip
acedb	ACeDB format
clustal aln	ClustalW ALN (multiple alignment) format.
codata	CODATA format.
dbid	Odd FASTA format with Database name first, then ID name then an optional accession number eg: >database name description or >database name accession description
embl em	EMBL entry format, or at least a minimal subset of the fields. The Staden package and others use EMBL or similar formats for sequence data.
experiment	The Staden package stores single sequencing experiment reads in a format derived from EMBL. All EMBL tags are allowed, plus many extras. Unusually, the extra tags are allowed to continue beyond the '//' line which only marks the end of the sequence. The "EX" experiment line is used to create a sequence description. Accuracy values are stored, or at least the largest value for each sequence position. To date no EMBOSS program is using these values.
fasta ncbi	FASTA format with optional accession number and database name in NCBI style included as part of the sequence identifier. eg >database|accession|id description or >name description or >name accession description (and other variants on this theme!)
gcg gcg8	GCG 9.x and 10.x format with the format and sequence type identified on the first line of the file. GCG 8.x format where anything up to the first line containing ".." is considered as heading, and the remainder is sequence data.
genbank gb ddbj	GENBANK entry format, including the feature table..
gff	GFF format. Normally used as a pure feature format, but can hold the sequence as part of the structured header.
hennig86	Hennig86 format
ig	IntelliGenetics format.

jackknifer	Jackknifer format
jackknifernon	Jackknifernon format
mega	Mega format
meganon	Meganon format
msf	Wisconsin Package GCG's MSF multiple sequence format.
nbrf pir	NBRF (PIR) format, as used in the PIR database sequence files. This format was used for some years as an interchange format with the reference data followed by the sequence data. This unofficial PIR format is what EMBOSS supports. If there is enough interest, we can also use NBRF database format with separate files for sequence (the main EMBOSS input/output) and for features. Documentation of this format is hard to find, but we do have a copy from PIR. The sequence files include the ID and description but no citation or feature information.
nexus paup	Nexus/PAUP format
nexusnon paupnon	Nexusnon/PAUPnon format
pearson	FASTA format with no further processing of the "ID" eg: >name description Used where fasta or ncbi format interprets the ID in an unwanted way, this format skips the further ID parsing stage of reading these files.
pfam stockholm	Pfam format
phylip phylipnon	PHYLIP interleaved multiple alignment format.
raw	Like text/plain format except that it removes any whitespace or digits, accepts only alphabetic characters and rejects anything else. This means that it is safer to use this format than plain format. If you have digits and spaces or TAB characters, these are removed and ignored. If you have other non-alphabetic characters (for example, punctuation characters), then the sequence will be rejected as erroneous. Gap characters, '-', and translated STOP codon characters '*' are legal.
selex	SELEX format is used by Sean Eddy's HMMER package. It can store RNA secondary structure as part of the sequence annotation.
staden	This format is actually obsolete, the latest version of the Staden package does not support it anymore (see "experiment" format for the new Staden package format). Staden format was a just the sequence in simple text with, optionally, comments at any position in the sequence. When EMBOSS reads in "staden" format, it recognizes a comment at the top of the sequence as the sequence dientifier and removes any comments inside the sequence. Some alternative nucleotide ambiguity codes are used and should be converted.
strider	DNA Strider format
swissprot swiss sw	SWISSPROT entry format, or at least a minimal subset of the fields.
text plain	Plain text. This is the format with no format. The whole of the file is read in as a sequence. No attempt is made to parse the file contents in any way. Anything is acceptable in this format. This means that any character will be included in the sequence, even digits and punctuation. Use this format only when you are sure that the input sequence file is correct and contains only what you want to be considered as your 'sequence'.
treecon	Treecon format
asis	This is not so much a sequence format as a quick way of entering a sequence on the command line, but it is included here for completeness. Where a filename would normally be given, in asis format

	there is the sequence itself. An example would be: asis::atacgcagttatctgaccat In 'asis' format the name is the sequence so no file needs to be opened. This is a special case. It was intended as a joke, but could be quite useful for generating command lines.

Tabel 2. Tipuri de formate ale secventelor din categoria de date Input

Output Format	Single/ Multiple	Comments
acedb	multiple	ACeDB format
asn1	multiple	A subset of ASN.1 containing entry name, accession number, description and sequence, similar to the current ASN.1 output of readseq
clustal aln	multiple	Clustal multiple sequence format.
codata	multiple	CODATA format.
debug	multiple	EMBOSS sequence object report for debugging showing all available fields. Not all fields will contain data - this depends very much on the input format used.
embl em	multiple	EMBL entry format with available fields filled in and others with no infomation omitted. The EMBOSS command line allows missing data such as accession numbers to be provided if they are not obtainable from the input sequence.
fasta pearson	multiple	Standard Pearson FASTA format, but with the accession number included after the identifier if available.
fitch	multiple	Fitch format
gcg gcg8	single	Wisconsin Package GCG 9.x and 10.x format with the sequence type on the first line of the file. GCG 8.x format where anything up to the first line containing ".." is considered as heading, and the remainder is sequence data.
genbank gb	multiple	GENBANK entry format with available fields filled in and others with no infomation omitted. The EMBOSS command line allows missing data such as accession numbers to be provided if they are not obtainable from the input sequence.
gff	multiple	GFF format. Normally used as a pure feature format, but can hold the sequence as part of the structured header.
hennig86	multiple	Hennig86 format
ig	multiple	Intelligenetics format, as used by the Intelligenetics package
jackknifer	multiple	Jackknifer format
jackknifernon	multiple	Jackknifernon format
mega	multiple	Mega format
meganon	multiple	Meganon format
msf	multiple	Wisconsin Package GCG's MSF multiple sequence format.
nbrf pir	multiple	NBRF (PIR) format, as used in the PIR database sequence files.
ncbi	multiple	NCBI style FASTA format with the database name, entry name and accession number separated by pipe ("|") characters.
nexus	multiple	Nexus/PAUP format

paup		
nexusnon paupnon	multiple	Nexusnon/PAUPnon format
phylip	multiple	PHYLIP interleaved format.
phylipnon	multiple	PHYLIP non-interleaved format that was used in Phylip version 3.2. Also called phylip3 for back compatibility with earlier EMBOSS versions.
selex	multiple	SELEX format.
staden	single	This format is actually obsolete, the latest version of the Staden package does not support it anymore. Staden format is a just the sequence in simple text with, optionally, comments at any position in the sequence. When EMBOSS reads in "staden" format, it recognizes only a comment at the top of the sequence but considers comments inside the sequence as part of the sequence. Some alternative nucleotide ambiguity codes are used and must be converted.
strider	multiple	DNA strider format
swiss sw	multiple	SwisProt entry format with available fields filled in and others with no infomation omitted. The EMBOSS command line allows missing data such as accession numbers to be provided if they are not obtainable from the input sequence.
text plain raw	single	Plain sequence, no annotation or heading.
treecon	multiple	Treecon format

Anexa 4

Clasa responsabila de import (XML -> baza de date BioDesc)

XMLv2bisToModel.java

```
package control;

import java.io.File;
import java.sql.Statement;
import java.util.Iterator;
import java.util.List;
import javax.swing.JOptionPane;
import model.Parameter;
import model.Service;
import model.Type;
import org.jdom.Document;
import org.jdom.Element;
import org.jdom.input.SAXBuilder;
import bioside.common.BiosideEntityResolver;

public class XMLv2bisToModel {
	static Document document;
	private Element root;


	public XMLv2bisToModel(String file) {
		SAXBuilder sxb = new SAXBuilder();

		sxb.setIgnoringElementContentWhitespace(true);
		sxb.setEntityResolver(new BiosideEntityResolver());

		try {
		   document = sxb.build(new File(file));
		  root = document.getRootElement();
		}
		catch (Exception e) {
		  System.err.println("Error occurred: " + e.getMessage());
		}
	  }

	public Service getService (Statement stmt) {
		Service service;
		Parameter parameter;
		String ID, name, description = "";
		DBToModel dbToModel = new DBToModel(stmt);

		try {
			ID = root.getAttributeValue("serviceKey");
			if (dbToModel.selectService(ID) != null) {
				int id = dbToModel.getHigherServiceID() + 1;
				ID  = "S" + String.valueOf(id);
				JOptionPane.showMessageDialog(null,
				   "The service ID in the XML file already exists.\n" +
				   "A new service ID has been generated. ID = '" + ID +
```

```
                                        "'.",
                                    "Existing service ID",
                                    JOptionPane.INFORMATION_MESSAGE);
                        }
                        else if (!ID.matches("S[1-9][0-9]*")) {
                                int id = dbToModel.getHigherServiceID() + 1;
                                ID  = "S" + String.valueOf(id);
                                JOptionPane.showMessageDialog(null,
                                    "The service ID in the XML file is not valid.\n" +
                                    "A new service ID has been generated. ID = '" + ID + "'.",
                                    "Invalid service ID",
                                    JOptionPane.INFORMATION_MESSAGE);
                        }
                }
                catch (Exception e) {
                        int id = dbToModel.getHigherServiceID() + 1;
                        ID  = "S" + String.valueOf(id);
                }

        name = root.getChildText("name_s");

        List descriptionList = root.getChildren("description_s");
        Iterator i = descriptionList.iterator();
        while(i.hasNext())
        {
                Element descriptionElement = (Element)i.next();
                description += descriptionElement.getText();
        }

        service = new Service(ID, name, description);

        String serviceIDExt = root.getChildText("ext_id");
        service.setIDExt(serviceIDExt);

        String serviceBelongs_to = root.getChildText("belongs_to");
        service.setBelongsTo(serviceBelongs_to);

        String serviceCode = root.getChildText("code_s");
        service.setCode(serviceCode);

        String serviceComments = root.getChildText("comments_s");
        service.setComments(serviceComments);

        String serviceState = root.getChildText("state");
        service.setState(serviceState);

        String serviceURL = root.getChildText("url");
        service.setURL(serviceURL);

        Element parameters = root.getChild("parameters");
        List list = parameters.getChildren("parameter");
        i = list.iterator();
        while(i.hasNext())
        {
                Element parameterElement = (Element)i.next();
                parameter = getParameter(parameterElement, service);
```

```
            service.addParameter(parameter);
        }
        return service;
    }


    private Parameter getParameter (Element parameterElement, Service service) {
            Parameter parameter;
            Type type;
            String parameterName, parameterDescription, parameterStatus = "regular";
            int parameterOrder;

            String parameterId = parameterElement.getAttributeValue("id");
            parameterOrder = Integer.parseInt(parameterElement.getChildText("order"));
            parameterName = parameterElement.getChildText("name");
            parameterDescription = parameterElement.getChildText("description");
            parameterStatus = parameterElement.getChildText("status");
            parameter = new Parameter(parameterId, parameterName, parameterDescription, parameterStatus,
                                        service);

            parameter.setOrder(parameterOrder);

            String strVisible = parameterElement.getAttributeValue("ishidden");
            if (strVisible != null && strVisible.equals("1"))
                            parameter.setVisible(false);
                    else
                            parameter.setVisible(true);

            String strUpdatable = parameterElement.getAttributeValue("isupdatable");
            if (strUpdatable != null && strUpdatable.equals("1"))
                            parameter.setUpdatable(false);
                    else
                            parameter.setUpdatable(true);


            int parameterPosition = Integer.parseInt(parameterElement.getChildText("position"));
            parameter.setPosition(parameterPosition);

            String parameterComments = parameterElement.getChildText("comments");
            parameter.setComments(parameterComments);

            Element types = parameterElement.getChild("types");
            List list = types.getChildren("type");
             Iterator i = list.iterator();
             while(i.hasNext())
             {
                    Element typeElement = (Element)i.next();
                    type = getType(typeElement, parameter);
                    parameter.addType(type);
             }

            String parameterDependences = parameterElement.getChildText("dep");
            parameter.setDependences(parameterDependences);

            String parameterBioDependences = parameterElement.getChildText("bio_dep");
            parameter.setBioDependences(parameterBioDependences);
```

```
    return parameter;
}

private Type getType (Element typeElement, Parameter parameter) {
        Type type;
        String value, base, nvalue, code_t,
        nreference, bio_depend, depend, bio_res, res, comments_t, format;

        String typeId = typeElement.getAttributeValue("id");
        base = typeElement.getChildText("base");
        type = new Type(typeId, base, parameter);

        String strDefault = typeElement.getAttributeValue("default");
        if (strDefault != null && strDefault.equals("0"))
                type.setDefault(false);
            else
                type.setDefault(true);


        String strIsCode = typeElement.getAttributeValue("iscode");
        if (strIsCode != null && strIsCode.equals("0"))
                type.setIsCode(false);
            else
                type.setIsCode(true);


        String strConstant = typeElement.getAttributeValue("isconstant");
        if (strConstant != null && strConstant.equals("0"))
                type.setConstant(false);
            else
                type.setConstant(true);


        String strReference = typeElement.getAttributeValue("isreference");
        if (strReference != null && strReference.equals("0"))
                type.setReference(false);
            else
                type.setReference(true);


        String strFormatter = typeElement.getAttributeValue("isformatter");
        if (strFormatter != null && strFormatter.equals("true"))
                type.setFormatter(true);
            else
                type.setFormatter(false);


        value = typeElement.getChildText("value");
        type.setValue(value);


        nvalue = typeElement.getChildText("nvalue");
        type.setNatureOfValue(nvalue);
```

```
                nreference = typeElement.getChildText("nreference");
                type.setNatureOfReference(nreference);


                code_t = typeElement.getChildText("code_t");
                type.setCode(code_t);


                bio_depend = typeElement.getChildText("bio_depend");
                type.setBioDependences(bio_depend);


                depend = typeElement.getChildText("depend");
                type.setDependences(depend);


                bio_res = typeElement.getChildText("bio_res");
                type.setBioRestrictions(bio_res);


                res = typeElement.getChildText("res");
                type.setRestrictions(res);


                format = typeElement.getChildText("formats");
                type.addFormat(format);


                comments_t = typeElement.getChildText("comments_t");
                type.setComments(comments_t);


                return type;
        }

}
```

Clasa responsabila de export (baza de date BioDesc -> XML)

ModelToXMLv2bis.java

```
package control;

import java.io.FileOutputStream;
import model.Parameter;
import model.Service;
import model.Type;
import org.jdom.DocType;
import org.jdom.Document;
import org.jdom.Element;
import org.jdom.output.Format;
import org.jdom.output.XMLOutputter;


public class ModelToXMLv2bis {
```

```
private Element root;

public ModelToXMLv2bis (Service service, String file) {
      Element serviceName, serviceDescription, serviceIDExt, serviceBelongs_to, serviceCode,
      serviceParameters, parameterElement, serviceComments, serviceState, serviceURL;

            root = new Element("program");
            root.setAttribute("id",service.getID());

            serviceName = new Element("name_s");
            serviceName.setText(service.getName());

            serviceIDExt = new Element("ext_id");
            serviceIDExt.setText(service.getIDExt ());

            serviceBelongs_to = new Element("belongs_to");
            serviceBelongs_to.setText(service.getBelongsTo ());

            serviceDescription = new Element("description_s");
            serviceDescription.setText(service.getDescription());


            serviceCode = new Element("code_s");
            serviceCode.setText(service.getCode());

            serviceComments = new Element("comments_s");
            serviceComments.setText(service.getComments());

            serviceState = new Element("state");
            serviceState.setText(service.getState());

            serviceURL = new Element("url");
            serviceURL.setText(service.getURL());

            root.addContent(serviceName);
            root.addContent(serviceIDExt);
            root.addContent(serviceBelongs_to);
            root.addContent(serviceDescription);
            root.addContent(serviceCode);
            root.addContent(serviceComments);
            root.addContent(serviceState);
            root.addContent(serviceURL);
            root.setAttribute("id", service.getName());

               /*
                * First create "command parameter"
                */

            serviceParameters = new Element("parameters");

            for (Parameter parameter: service.parameters())
               {
                        parameterElement = buildParameterElement(parameter);
                        serviceParameters.addContent(parameterElement);
               }
```

```
            root.addContent(serviceParameters);
            createXMLFile(file);

}



private Element buildParameterElement(Parameter parameter) {
      Element parameterElement, parameterName, parameterPosition, parameterOrder,
      parameterDescription, parameterComments, parameterStatus, parameterTypes, typeElement,
      dependances, biodependances;//, types;

            parameterElement = new Element("parameter");

            parameterName = new Element("name");
            parameterName.setText(parameter.getName());

            parameterDescription = new Element("description");
            parameterDescription.setText(parameter.getDescription());

            parameterPosition = new Element("position");
            parameterPosition.setText(String.valueOf(parameter.getPosition()+1));

            parameterOrder = new Element("order");
            parameterOrder.setText(String.valueOf(parameter.getOrder()+1));

            parameterComments = new Element ("comments");
            parameterComments.setText(parameter.getComments());

            parameterStatus = new Element("status");
            parameterStatus.setText(parameter.getStatus());


            parameterElement.addContent(parameterName);
            parameterElement.addContent(parameterDescription);
            parameterElement.addContent(parameterPosition);
            parameterElement.addContent(parameterOrder);
            parameterElement.addContent(parameterComments);
            parameterElement.addContent(parameterStatus);

            parameterTypes = new Element("types");

            for (Type type: parameter.types()) {
                  typeElement = buildTypeElement(type);
                  parameterTypes.addContent(typeElement);
             }

               parameterElement.addContent(parameterTypes);

             if (parameter.getDependences() != null) {
                  dependances = new Element("dep");
                  dependances.setText(parameter.getDependences());
                  parameterElement.addContent(dependances);
               }
```

```
            if (parameter.getBioDependences() != null) {
                    biodependances = new Element("bio_dep");
                    biodependances.setText(parameter.getBioDependences());
                    parameterElement.addContent(biodependances);
            }

               parameterElement.setAttribute("id", parameter.getID());
               if (!parameter.isVisible())
                    parameterElement.setAttribute("ishidden","1");

               if (!parameter.isUpdatable())
                    parameterElement.setAttribute("isupdatable","1");

               parameterElement.setAttribute("type", "command");
               parameterElement.setAttribute("ismandatory", "1");

        return parameterElement;
    }


private Element buildTypeElement (Type type) {
          Element typeElement, value, base, nvalue, nreference, bio_depend, depend, bio_res, res,
          formatElement, comments_t;

            String formats[]= type.formats();


            typeElement = new Element("type");

            if (!type.isConstant())
                    typeElement.setAttribute("isconstant","0");

                    if (!type.isDefault())
                            typeElement.setAttribute("default", "0");

                    typeElement.setAttribute("id", type.getID());

                    base = new Element("base");
                    base.setText(type.getBase());
                    typeElement.addContent(base);

                    if (type.getValue() != null) {
                            value = new Element("value");
                            value.setText(type.getValue());
                            typeElement.addContent(value);
                    }

                    if (type.getNatureOfValue() != null) {
                            nvalue = new Element("nvalue");
                            nvalue.setText(type.getNatureOfValue());
                            typeElement.addContent(nvalue);
                    }

                    if (type.getNatureOfReference() != null){
                            nreference = new Element("nreference");
                            nreference.setText(type.getNatureOfReference());
```

```
                typeElement.addContent(nreference);
        }

        Element code_t = new Element("code_t");
        if (type.getCode() != null) {
                code_t.setText(type.getCode());
                typeElement.addContent(code_t);
        }

        if (type.isFormatter())
                code_t.setAttribute("code","true");

        if (!type.isCode())
                typeElement.setAttribute("iscode", "0");

        if (type.getBioDependences() != null) {
                bio_depend = new Element("bio_depend");
                bio_depend.setText(type.getBioDependences());
                typeElement.addContent(bio_depend);
        }

        if (type.getDependences() != null) {
                depend = new Element("depend");
                depend.setText(type.getDependences());
                typeElement.addContent(depend);
        }

        if (type.getBioRestrictions() != null) {
                bio_res = new Element("bio_res");
                bio_res.setText(type.getBioRestrictions());
                typeElement.addContent(bio_res);
          }

        if (type.getRestrictions() != null) {
                res = new Element("res");
                res.setText(type.getRestrictions());
                typeElement.addContent(res);
        }

        for (String format:formats) {
        formatElement = new Element ("formats");
        formatElement.setText(format.toString());
        typeElement.addContent(formatElement);
        }

        if (!(type.getNatureOfReference() != null))
                typeElement.setAttribute("isreference","0");


        comments_t = new Element("comments_t");
        comments_t.setText(type.getComments());
        typeElement.addContent(comments_t);

   return typeElement;

}
```

```
	private void createXMLFile (String file) {

		Document document = new Document(root);
		 /*
		  * <!DOCTYPE program PUBLIC "-//ENSTBr//DTD Bioside Program 2.0//EN" "http://perso.enst-
		 bretagne.fr/~picouet/projets/genomer/bioside/program_description.dtd">
		  *
		  */



		 DocType docType = new DocType("program", "-//ENSTBr//DTD Bioside Program
		 		2.0//EN", "file://C:\\program_description1.dtd");

		 document.setDocType(docType);
		 FileOutputStream os = null;

		 try {
		 	os = new FileOutputStream(file);
		 	XMLOutputter sortie = new XMLOutputter (Format.getPrettyFormat());
		 	sortie.output(document, os);
		 	os.close();
		 }
		 catch (java.io.IOException e) {
		 	e.printStackTrace();
		 }
	}

}
```