Mestrado Integrado em Engenharia Informática e Computação

Métodos Formais em Engenharia de Software

Scientific Paper Index System

Relatório Final

Grupo XII:
200702669 - André Humberto Trigo de Bordalo Morais - ei07122@fe.up.pt
201208217 - João Alberto Trigo de Bordalo Morais - ei12040@fe.up.pt

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

15 de Dezembro de 2015

Conteúdo

1 Descrição do Sistema

O nosso projeto consiste em modelar um sistema para indexação de artigos científicos (*Scientific Paper Index System* - SPIS) semelhante ao criado pela *Microsoft*. Ver `http://academic.research.microsoft.com` Neste projeto pretende-se inserir:

- Publicações;
- Autores;
- Afiliações;
- Referências Bibliográficas;
- Interesse por autor;

com o propósito de contar o número de citações que um determinado autor é citado noutras publicações, bem como o número de vezes que este se sita nas suas publicações. Para além desta funcionalidade, um SPIS deve também computar o caminho de co-autoria, isto é, computar as relações de co-autoria; e ainda computar o número de Erdos [1]

1.1 Lista de requisitos

Para cumprir o objectivo são necessários cumprir os seguintes requisitos:

Tabela 1: My caption

Identificador Requisito	Descriçao
Req01	Inserir uma nova publicação no sistema
Req02	Inserir autores no Sistema
Req03	Inserir novas filiações
Req04	adicionar referências a publicações
Req05	Adicionar interesses a um autor
Req06	Calcular o numero de auto citações (em publicações do próprio autor)
Req07	Calcular o número de citações de um autor por outros autores
Req08	Computar o co-author path (ligações de um autor a outros por co-autoria de artigos)
Req09	Computar o número de Erdös (distância relativa entre dois autores)
Req10	Garantir que uma publicação não referencia outra de data posterior

[1] Erdos Number

2 UML

Em seguida será apresentado o modelo de caso de uso e o diagrama de classes do nosso projeto:

2.1 Modelo de Caso de Uso

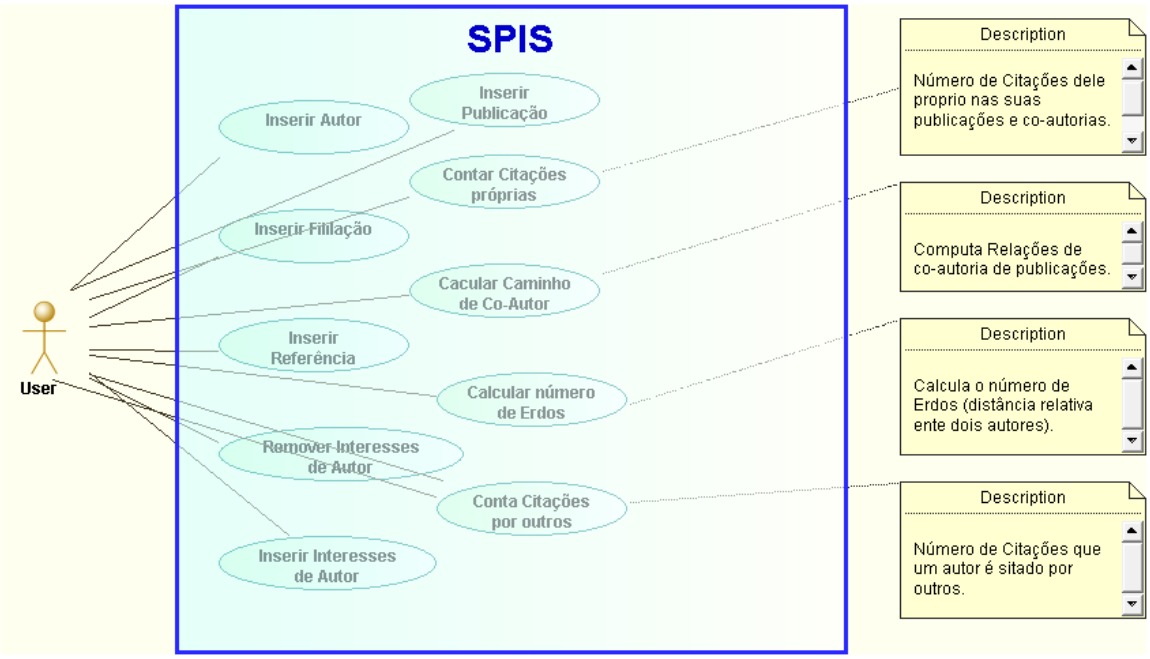

Figura 1: Diagrama de Caso de Uso

2.2 Diagramas de Classes

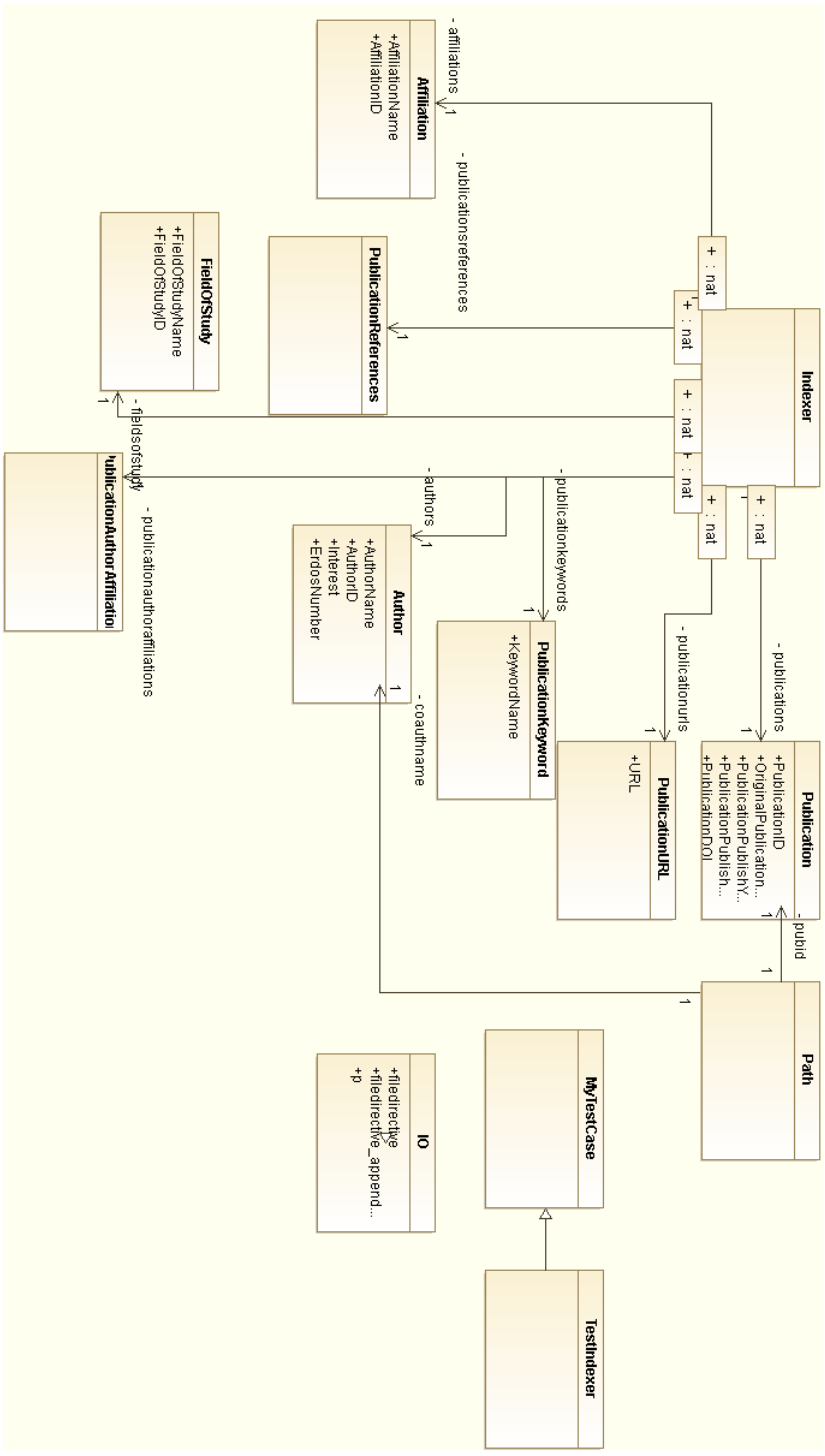

Figura 2: Diagrama de Classes

3 Modelo VDM++

3.1 Affiliation

```
class Affiliation
types
public AffiliationName = seq of char;
public AffiliationID= nat;


values
-- TODO Define values here
instance variables

 private name: AffiliationName := [];



operations

 public Affiliation : AffiliationName  ==> Affiliation
  Affiliation(an) ==
  (
  name := an;
  return self;

  );


 public changeName: AffiliationName ==> ()
 changeName(an) ==
 name :=  an;




 pure public getName: () ==> AffiliationName
  getName() == return name;



functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end Affiliation
```

3.2 Author

```
class Author
types
public AuthorName = seq of char;
public AuthorID= nat;
public Interest= seq of char;
public ErdosNumber= nat;


values
-- TODO Define values here
instance variables
 private name: AuthorName := [];
 private interests: set of Interest:= {};
private erdos: ErdosNumber := 9999;
```

```
operations
 public Author : AuthorName *
          set of Interest ==> Author
  Author(an, ints) ==
  (
  name := an;
  interests := ints;

  return self;


  );

  public Author : AuthorName *
          set of Interest * ErdosNumber  ==> Author

  Author(an, ints, erdsnum) ==
  (
  name := an;
  interests := ints;
  erdos := erdsnum;


  return self;

  );

 public changeName: AuthorName ==> ()

 changeName(an) ==
 name :=  an;



 public addInterest: Interest ==> ()
  addInterest(intr) ==
  interests:= interests union {intr}
  post card interests >= card interests~;


 public addInterests: set of Interest ==> ()
  addInterests(intr) ==

  (interests:= interests union intr;

  )
  post card interests >= card interests~;


 public removeInterests: set of Interest ==> ()
  removeInterests(intrs) ==
   ( interests := interests \ intrs;

   )
   pre intrs subset interests;


 pure public getErdos: () ==> ErdosNumber
 getErdos() == return erdos;

 public setErdos: ErdosNumber ==> ()
  setErdos(en) ==
  erdos:=en;


 pure public getName: () ==> AuthorName
  getName() == return name;
```

```
 pure public getInterests: () ==> set of Interest
  getInterests() == return interests;



functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end Author
```

3.3 FieldOfStudy

```
class FieldOfStudy
types
public FieldOfStudyName = seq of char;
public FieldOfStudyID= nat;


values
-- TODO Define values here
instance variables
 -- private id : FieldOfStudyID := 0;
 private name: FieldOfStudyName := [];



operations

 public FieldOfStudy : FieldOfStudyName  ==> FieldOfStudy
  FieldOfStudy( an) ==
  (
  name := an;
  return self;

  );


 public changeName: FieldOfStudyName ==> ()
 changeName(an) ==
 name :=  an;



 pure public getName: () ==> FieldOfStudyName
  getName() == return name;



functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end FieldOfStudy
```

3.4 Indexer

```
class Indexer
types
-- TODO Define types here
values
```

```
-- TODO Define values here
instance variables
 private publications : map nat to Publication := { |->};
 private publicationIdCounter: nat:= 0;
 private authors : map nat to Author := { |->};
 private authorIdCounter : nat := 0;
 private affiliations : map nat to Affiliation := { |->};
 private affiliationIdCounter: nat := 0;
 private fieldsofstudy : map nat to FieldOfStudy := { |->};
 private fieldofstudyIdCounter: nat := 0;
 private publicationkeywords : map nat to PublicationKeyword := { |->};
 private publicationkeywordIdCounter: nat := 0;
 private publicationauthoraffiliations : map nat to PublicationAuthorAffiliation := { |->};
 private paaIdCounter: nat := 0;
 private publicationsreferences : map nat to PublicationReferences := { |->};
 private pReferencesIdCounter: nat := 0;
 private publicationurls : map nat to PublicationURL := { |->};
operations

 public Indexer: map nat to Publication * map nat to Author * map nat1 to Affiliation *
         map nat to FieldOfStudy * map nat to PublicationKeyword * map nat1 to
             PublicationAuthorAffiliation *
         map nat to PublicationReferences * map nat to PublicationURL ==> Indexer
     Indexer(pubs, auts, affs, fsos, pks, paas, prs, pus) ==
     (
      publications := pubs;
      authors := auts;
      affiliations := affs;
      fieldsofstudy := fsos;
      publicationkeywords := pks;
      publicationauthoraffiliations := paas;
      publicationsreferences := prs;
      publicationurls := pus;
      return self;
     );

 public Indexer: () ==> Indexer
     Indexer() ==
     (
      return self;
     );



 pure public getPublications: () ==> map nat to Publication
  getPublications() == return publications;


 pure public getAuthors: () ==> map nat to Author
  getAuthors() == return authors;


 pure public getAffiliations: () ==> map nat to Affiliation
  getAffiliations() == return affiliations;


 pure public getFieldsofstudy: () ==> map nat to FieldOfStudy
  getFieldsofstudy() == return fieldsofstudy;


 pure public getPublicationkeywords: () ==> map nat to PublicationKeyword
  getPublicationkeywords() == return publicationkeywords;


 pure public getPublicationauthoraffiliations: () ==> map nat to PublicationAuthorAffiliation
  getPublicationauthoraffiliations() == return publicationauthoraffiliations;


 pure public getPublicationsreferences: () ==> map nat to PublicationReferences
  getPublicationsreferences() == return publicationsreferences;
```

```
pure public getPublicationurls: () ==> map nat to PublicationURL
 getPublicationurls() == return publicationurls;

   --Req02 - Inserir um autor no sistema

public insertAuthor: Author`AuthorName * set of Author`Interest ==> ()
 insertAuthor(an, ints) ==
   (
       authorIdCounter := authorIdCounter+1;
       authors := authors munion {authorIdCounter|-> new Author(an, ints) };

    );


 Req03 - Inserir uma filiacao no sistema.
public insertAffiliation: Affiliation`AffiliationName  ==> ()

 insertAffiliation(affiname) ==
   (
       affiliationIdCounter := affiliationIdCounter+1;
       affiliations := affiliations munion {affiliationIdCounter|-> new Affiliation(affiname)
            };


    );




public getAuthorById: nat1 ==> Author
 getAuthorById(id) ==
  return authors(id);


  --Req05 - Adicionar interesses a um autor
public addInterestsToAuthor: nat1 * set of Author`Interest ==> ()
  addInterestsToAuthor(aid, intr) ==
  (dcl author: Author := getAuthorById(aid);
  author.addInterests(intr);
   authors := authors munion {aid |-> author};
   );


public removeInterestsFromAuthor:  nat1 * set of Author`Interest ==>()
 removeInterestsFromAuthor(aid, intrs) ==
 (dcl author: Author := getAuthorById(aid);
  author.removeInterests(intrs);
  authors:= authors ++ {aid |-> author};
  );

  --ReQ01 - inserir uma publicacao
public insertPublication: Publication`OriginalPublicationTitle * Publication`
    PublicationPublishYear * Publication`PublicationPublishDate *
             Publication`PublicationDOI * Publication`PublicationRank * Author`AuthorID * set
                  of Author`AuthorID *
             Affiliation`AffiliationID * set of Publication`PublicationID ==> ()
 insertPublication(ortitle, pubyear, pubdate, pubdoi, pubrank, idau, idcoauts, idaf, idrefs)
     ==
   (
    publicationIdCounter := publicationIdCounter+1;
    publications := publications munion {publicationIdCounter|-> new Publication(ortitle,
        pubyear, pubdate, pubdoi, pubrank)};


    paaIdCounter:= paaIdCounter+1;
    publicationauthoraffiliations:= publicationauthoraffiliations munion {paaIdCounter |->
        new PublicationAuthorAffiliation(publicationIdCounter, idau, idcoauts, idaf,
        affiliations(idaf).getName())};
```

```
    insertReferences(publicationIdCounter,idrefs);

    --Req09  Actualizacao do erdos number de cada autor
    (dcl everypubauthor: set of Author`AuthorID :={idau}union idcoauts ;
     dcl minerdos: Author`ErdosNumber:=999;

     dcl minerdosauthorid: Author`AuthorID:= 9999999;
     for all pubauthor in set everypubauthor do
     (
      if authors(pubauthor).getErdos() < minerdos
      then
       (
       minerdos:= authors(pubauthor).getErdos();
       minerdosauthorid:= pubauthor;
       )
     );

     for all pubauthor in set everypubauthor do
     (

      if pubauthor <> minerdosauthorid
      then authors(pubauthor).setErdos(minerdos+1);

     )
    )
   )
   pre idau in set dom authors and
   idaf in set dom affiliations and

   forall idcoau in set idcoauts &
   (
   idcoau in set dom authors
   );



public getPublicationsByAuthor: Author`AuthorID ==> set of PublicationAuthorAffiliation
getPublicationsByAuthor(ida) ==
(dcl pubs : set of PublicationAuthorAffiliation := {};
for all pub in set rng publicationauthoraffiliations do
 if pub.getAutid() = ida
  then pubs := pubs union {pub};
return pubs);

public getAuthorByPublication: Publication`PublicationID ==> Author`AuthorID
 getAuthorByPublication(idp) ==
 (dcl aid: Author`AuthorID :=0;
 for all pub in set rng publicationauthoraffiliations do

 if idp = pub.getPubid()
 then aid:= pub.getAutid();
 return aid);



public getReferencesFromAuthorPublications: set of PublicationAuthorAffiliation ==> set of
    Publication`PublicationID
getReferencesFromAuthorPublications(pubs) ==
   (dcl idpubs: set of Publication`PublicationID := {};
    for all pub in set pubs do
     idpubs:= idpubs union {pub.getPubid()};
    (dcl idrefs: set of Publication`PublicationID := {};
     for all id in set rng publicationsreferences do
      idrefs:= idrefs union id.getPrids();

      return idrefs));

   --Req08 co-author path - Todos as relacoes entre este autor e outros autores por via de co
       -autoria de publicacoes
```

```
public getCoauthorPath: Author`AuthorID ==> seq of Path
  getCoauthorPath(aid) ==
   (dcl pubs: set of PublicationAuthorAffiliation := getPublicationsByAuthor(aid);
    dcl coauthpath:  seq of Path:= [];
    for all pub in set pubs do
    (
     dcl coauths: set of Author`AuthorID:= pub.getCoautids();
      for all coauth in set coauths do

      (
       dcl path : Path := new Path(pub.getPubid() , authors(coauth).getName());
       coauthpath := coauthpath ^ [path];
      )
    );
    return coauthpath;

  );



 --Req06  numero de citacoes de um autor a si mesmo nas suas publicaceos
public getNumberSelfCitations: Author`AuthorID ==> nat
 getNumberSelfCitations(ida)==
 (dcl selfpubs: set of PublicationAuthorAffiliation := getPublicationsByAuthor(ida);
  (dcl idrefs: set of Publication`PublicationID := getReferencesFromAuthorPublications(
      selfpubs);
   (dcl idselfref: set of Publication`PublicationID := {};
    for all refs in set idrefs do
      if ida = getAuthorByPublication(refs)
      then  idselfref := idselfref union {refs};
     return card getReferencesFromAuthorPublications(selfpubs))));



   --Req07 - numero de vezes que um autor foi citado por outros em publicacoes
public getTimesCited: Author`AuthorID ==> nat
getTimesCited(ida) ==
(dcl authorpubs: set of PublicationAuthorAffiliation := getPublicationsByAuthor(ida);
dcl pubscited: seq of Publication`PublicationID := [];
for all pubs in set authorpubs do
(
 for all references in set rng publicationsreferences do
  if {pubs.getPubid()} subset references.getPrids()
  then pubscited:= pubscited ^ [pubs.getPubid()];
);
  return len pubscited);


 -- Req10 a pos e as pre condicoes garantem que este restricao e satisfeita
 -- Req04 adicionar referencias a publicacoes
 public insertReferences: Publication`PublicationID * set of Publication`PublicationID ==> ()
 insertReferences(idpub, idrefs) ==
   (pReferencesIdCounter := pReferencesIdCounter+1;
   publicationsreferences := publicationsreferences munion {pReferencesIdCounter |-> new
       PublicationReferences(idpub, idrefs)};
   )
   pre idpub in set dom publications and
   idrefs subset dom publications and
   forall refs in set idrefs &
   (
    publications(idpub).getPublishYear() >= publications(refs).getPublishYear() and
    publications(idpub).getPublishDate() >= publications(refs).getPublishDate()
   )
    ;
```

```
functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end Indexer
```

3.5 MyTestCase

```
class MyTestCase
/*
  Superclass for test classes, simpler but more practical than VDMUnit`TestCase.
  For proper use, you have to do: New -> Add VDM Library -> IO.
  FEUP, MFES, 2015/16.
*/

operations

 -- Simulates assertion checking by reducing it to pre-condition checking.
 -- If 'arg' does not hold, a pre-condition violation will be signaled.

 protected assertTrue: bool ==> ()
 assertTrue(arg) ==
  return
 pre arg;

 -- Simulates assertion checking by reducing it to post-condition checking.
 -- If values are not equal, prints a message in the console and generates
 -- a post-conditions violation.

 protected assertEqual: ? * ? ==> ()
 assertEqual(expected, actual) ==
  if expected <> actual then (
    IO`print("Actual value (");
    IO`print(actual);
    IO`print(") different from expected (");
    IO`print(expected);
    IO`println(")\n")
  )
 post expected = actual

end MyTestCase
```

3.6 Path

```
class Path
types
-- TODO Define types here
values
-- TODO Define values here
instance variables
private pubid : Publication`PublicationID := 0;
private coauthname : Author`AuthorName := [];
operations


 public Path : Publication`PublicationID * Author`AuthorName  ==> Path
  Path(pid, autname) ==
  (
  pubid := pid;
  coauthname := autname;
  return self;
```

```
  );


pure public getpubid: () ==> Publication`PublicationID
   getpubid() == return pubid;


  pure public getcoauthname: () ==> Author`AuthorName
   getcoauthname() == return coauthname;

functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end Path
```

3.7 Publication

```
class Publication
types
public PublicationID = nat;
public OriginalPublicationTitle = seq of char;
public PublicationPublishYear = nat;
public PublicationPublishDate=  nat;
public PublicationDOI = seq of char;
public PublicationRank = nat;


values
-- TODO Define values here
instance variables
private originalTitle: OriginalPublicationTitle := [];
private publishYear: PublicationPublishYear:= 0000; --YYYY
private publishDate: PublicationPublishDate:= 0000; --MMDD
private doi: PublicationDOI:= [];
private rank: PublicationRank:= 99999;

operations
-- TODO Define operations here

public Publication :  OriginalPublicationTitle *
           PublicationPublishYear * PublicationPublishDate *
           PublicationDOI * PublicationRank ==> Publication
 Publication(opt, ppy, ppd, pdoi, pr) == (
  originalTitle := opt;
  publishYear := ppy;
  publishDate := ppd;
  doi := pdoi;
  rank := pr;
  return self
 );


 public raiseRankBy: nat1 ==>()
 raiseRankBy(nRanks) == rank := rank-nRanks
 pre rank>1
 post rank>=1 and rank<rank~;


 public decreaseRankBy: nat1 ==>()
 decreaseRankBy(nRanks) == rank := rank+nRanks
 pre rank>=1
 post rank>rank~;
```

```
 pure public getOriginalTitle: () ==> OriginalPublicationTitle
  getOriginalTitle() == return originalTitle;


 pure public getPublishYear: () ==> PublicationPublishYear
 getPublishYear() == return publishYear;


 pure public getPublishDate: () ==> PublicationPublishDate
 getPublishDate() == return publishDate;


 pure public getDoi: () ==> PublicationDOI
 getDoi() == return doi;


 pure public getRank: () ==> PublicationRank
 getRank() == return rank;

functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end Publication
```

3.8 PublicationAuthorAffiliation

```
class PublicationAuthorAffiliation
types




values
-- TODO Define values here
instance variables
 private pubid : Publication`PublicationID := 0;
 private autid: Author`AuthorID := 0;
 private coautids: set of Author`AuthorID := {};
 private affid: Affiliation`AffiliationID := 0;
 private oriaffnam: Affiliation`AffiliationName := [];




operations

 public PublicationAuthorAffiliation : Publication`PublicationID * Author`AuthorID *
                   set of Author`AuthorID * Affiliation`AffiliationID *
                   Affiliation`AffiliationName  ==> PublicationAuthorAffiliation
  PublicationAuthorAffiliation(puid, auid, coaid, afid, oan) ==
  (
  pubid := puid;
  autid := auid;
  coautids := coaid;
  affid := afid;
  oriaffnam := oan;
  return self;

  );


 public changeOriAffName: Affiliation`AffiliationName ==> ()
 changeOriAffName(na) ==
 oriaffnam :=  na;
```

```
 pure public getPubid: () ==> Publication`PublicationID
 getPubid() == return pubid;


 pure public getAutid: () ==> Publication`PublicationID
 getAutid() == return autid;


 pure public getCoautids: () ==> set of Author`AuthorID
 getCoautids() == return coautids;


 pure public getAffid: () ==> Publication`PublicationID
 getAffid() == return affid;



 pure public getOriAffName: () ==> Affiliation`AffiliationName
 getOriAffName() == return oriaffnam;




functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end PublicationAuthorAffiliation
```

3.9 PublicationKeyword

```
class PublicationKeyword
types
public KeywordName = seq of char;


values


instance variables

 private pid : Publication`PublicationID := 1;
 private fosid: FieldOfStudy`FieldOfStudyID:= 1;
 private keyword: KeywordName := [];
operations


public PublicationKeyword : Publication`PublicationID * FieldOfStudy`FieldOfStudyID *
              KeywordName  ==> PublicationKeyword
  PublicationKeyword(id, fid, kw) ==
  (
  pid := id;
  fosid := fid;
  keyword := kw;
  return self;
  );


  public changeKeyword: KeywordName ==> ()
 changeKeyword(kw) ==
 keyword :=  kw;


 pure public getPid: () ==> Publication`PublicationID
```

```
  getPid() == return pid;


 pure public getFosid: () ==> FieldOfStudy`FieldOfStudyID
  getFosid() == return fosid;


 pure public getKeyword: () ==> KeywordName
  getKeyword() == return keyword;

functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end PublicationKeyword
```

3.10 PublicationReferences

```
class PublicationReferences
types
-- TODO Define types here
values
-- TODO Define values here
instance variables
 private pid : Publication`PublicationID := 0;
 private prids: set of Publication`PublicationID:= {};

operations


 public PublicationReferences : Publication`PublicationID * set of Publication`PublicationID
                ==> PublicationReferences
    PublicationReferences(id, rids) ==
    (
     pid := id;
     prids := rids;
     return self;
    )
    pre id not in set rids;


 public addReference: Publication`PublicationID ==> ()
 addReference(pubid) ==
 prids:= prids union {pubid}
 pre pubid not in set prids
 post pubid in set prids;


 public removeReference: Publication`PublicationID ==> ()
 removeReference(prid) ==
 prids:= prids \ {prid}
 pre prid in set prids
 post prid not in set prids;


 pure public getPid: () ==> Publication`PublicationID
 getPid() == return pid;


 pure public getPrids: () ==> set of Publication`PublicationID
 getPrids() == return prids;


functions
-- TODO Define functiones here
traces
```

```
-- TODO Define Combinatorial Test Traces here
end PublicationReferences
```

3.11 PublicationURL

```
class PublicationURL
types
public URL = seq of char;
values
-- TODO Define values here
instance variables
 private pid : Publication`PublicationID := 0;
 private url : URL := [];

operations

 public PublicationURL : Publication`PublicationID * URL  ==> PublicationURL
  PublicationURL(id, u) ==
  (
   pid := id;
   url := u;
   return self;
  );


  pure public getPid: () ==> Publication`PublicationID
   getPid() == return pid;


  pure public getUrl: () ==> URL
   getUrl() == return url;
functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end PublicationURL
```

3.12 TestIndexer

```
class TestIndexer is subclass of MyTestCase

operations
-- TODO Define operations here



 public testAffiliation: () ==> ()
  testAffiliation() ==
  (dcl af: Affiliation := new Affiliation("FEUP");
   assertEqual("FEUP", af.getName());
  );


 public testAffiliationChangeName: () ==> ()
  testAffiliationChangeName() ==
  (dcl af: Affiliation := new Affiliation("FEUP");
   assertEqual("FEUP", af.getName());
   af.changeName("MIT");
   assertEqual("MIT", af.getName());
  );
```

```
 public testAuthor: () ==> ()
  testAuthor() ==
  (dcl a: Author := new Author("Andre", {"amputacoes", "desmembramentos"});
   assertEqual("Andre", a.getName());
   assertEqual({"amputacoes", "desmembramentos"}, a.getInterests());
  );




public testAuthorChangeName: () ==> ()
 testAuthorChangeName() ==
 (dcl a: Author := new Author("Andre", {"amputacoes", "desmembramentos"});
  assertEqual("Andre", a.getName());
   a.changeName("Banana");
   assertEqual("Banana", a.getName());
  );


 public testAddInterestsAuthor: () ==> ()
  testAddInterestsAuthor() ==
  (dcl a: Author := new Author("Andre", {"amputacoes", "desmembramentos"});
   assertEqual({"amputacoes", "desmembramentos"}, a.getInterests());
   a.addInterests({"mutilacoes", "imolacoes"});
   assertEqual({"amputacoes", "desmembramentos", "mutilacoes", "imolacoes"}, a.getInterests()
       );
  );


  public testRemoveInterestsAuthor: () ==> ()
   testRemoveInterestsAuthor() ==
   (dcl a: Author := new Author("Andre", {"amputacoes", "desmembramentos"});
    assertEqual({"amputacoes", "desmembramentos"}, a.getInterests());
    a.removeInterests({"amputacoes"});
    assertEqual({"desmembramentos"}, a.getInterests());
    a.removeInterests({"desmembramentos"});
    assertEqual({}, a.getInterests());
   );


  public testFieldOfStudy: () ==> ()
   testFieldOfStudy() ==
   (dcl fos: FieldOfStudy:= new FieldOfStudy("Necromancia");
    assertEqual("Necromancia", fos.getName());
   );


  public testPublication: () ==> ()
   testPublication() ==
   (dcl pub: Publication := new Publication("A Specifier's Introduction to Formal Methods",
                        1990, 0901, "11.111.11/ISBN", 100);
    assertEqual("A Specifier's Introduction to Formal Methods", pub.getOriginalTitle());
    assertEqual(1990, pub.getPublishYear());
    assertEqual(0901, pub.getPublishDate());
    assertEqual("11.111.11/ISBN", pub.getDoi());
    assertEqual(100, pub.getRank());
   );


  public testRaiseRankBy: () ==> ()
   testRaiseRankBy() ==
   (dcl pub: Publication := new Publication("A Specifier's Introduction to Formal Methods",
                        1990, 0901, "11.111.11/ISBN", 100);
    assertEqual(100, pub.getRank());
    pub.raiseRankBy(50);
    assertEqual(50, pub.getRank());
```

```
 );


public testDecreaseRankBy: () ==> ()
 testDecreaseRankBy() ==
 (dcl pub: Publication := new Publication("A Specifier's Introduction to Formal Methods",
                      1990, 0901, "11.111.11/ISBN", 100);
  assertEqual(100, pub.getRank());
  pub.decreaseRankBy(50);
  assertEqual(150, pub.getRank());

 );


public testPublicationAuthorAffiliation: () ==> ()
 testPublicationAuthorAffiliation() ==
 (dcl paa: PublicationAuthorAffiliation := new PublicationAuthorAffiliation(1, 2, {1, 3,
     4}, 3 , "FEUP");
  assertEqual(1, paa.getPubid());
  assertEqual(2, paa.getAutid());
  assertEqual(3, paa.getAffid());
  assertEqual("FEUP", paa.getOriAffName());

 );


 public testPublicationAuthorAffiliationChangeName: () ==> ()
 testPublicationAuthorAffiliationChangeName() ==
 (dcl paa: PublicationAuthorAffiliation := new PublicationAuthorAffiliation(1, 2, {1, 3,
     4}, 3,  "FEUP");
  assertEqual("FEUP", paa.getOriAffName());
  paa.changeOriAffName("MIT");
  assertEqual("MIT", paa.getOriAffName());

 );


 public testPublicationKeyword: () ==> ()
 testPublicationKeyword() ==
 (dcl pk: PublicationKeyword := new PublicationKeyword(1, 2, "MFES");
  assertEqual(1, pk.getPid());
  assertEqual(2, pk.getFosid());
  assertEqual("MFES", pk.getKeyword());
 );


 public testPublicationKeywordChangeKeyword: () ==> ()
 testPublicationKeywordChangeKeyword() ==
 (dcl pk: PublicationKeyword := new PublicationKeyword(1, 2, "MFES");
  pk.changeKeyword("SEFM");
  assertEqual("SEFM", pk.getKeyword());
 );


 public testPublicationReferences: () ==> ()
 testPublicationReferences() ==
 (dcl pr: PublicationReferences := new PublicationReferences(1,{2, 3});
      assertEqual(1, pr.getPid());
      -- da para fazer assert error ou assim?
      -- assertEqual({1, 2}, pr.getPrids());
      assertEqual({2, 3}, pr.getPrids());


 );


 public testPublicationReferencesAddReference: () ==> ()
 testPublicationReferencesAddReference() ==
 (dcl pr: PublicationReferences := new PublicationReferences(1,{2, 3});
```

```
      assertEqual(1, pr.getPid());
      -- da para fazer assert error ou assim?
      -- assertEqual({1, 2}, pr.getPrids());
      assertEqual({2, 3}, pr.getPrids());
      --pr.addReference(2);
      pr.addReference(4);
      assertEqual({2, 3, 4}, pr.getPrids());

 );


 public testPublicationReferencesRemoveReference: () ==> ()
 testPublicationReferencesRemoveReference() ==
 (dcl pr: PublicationReferences := new PublicationReferences(1,{2, 3, 4});
      assertEqual(1, pr.getPid());
      -- da para fazer assert error ou assim?
      -- assertEqual({1, 2}, pr.getPrids());
      assertEqual({2, 3, 4}, pr.getPrids());
      --pr.addReference(2);
      pr.removeReference(4);
      assertEqual({2, 3}, pr.getPrids());

 );


 public testPublicationURL: () ==> ()
 testPublicationURL() ==
 (dcl purl: PublicationURL := new PublicationURL(1,"mypublicatonurl.com" );
     assertEqual(1, purl.getPid());
     assertEqual("mypublicatonurl.com", purl.getUrl());
);


  public testIndexerEmpty: () ==> ()
  testIndexerEmpty() ==
  (dcl idx: Indexer := new Indexer();
   assertEqual({|->}, idx.getPublications());
   assertEqual({|->}, idx.getAuthors());
   assertEqual({|->}, idx.getAffiliations());
   assertEqual({|->}, idx.getFieldsofstudy());
   assertEqual({|->}, idx.getPublicationkeywords());
   assertEqual({|->}, idx.getPublicationauthoraffiliations());
   assertEqual({|->}, idx.getPublicationsreferences());
   assertEqual({|->}, idx.getPublicationurls());

  );


  public testIndexerAuthors: () ==> ()
  testIndexerAuthors() ==
  (dcl idx: Indexer := new Indexer();
   idx.insertAuthor("autor1", {"cozinhar", "escrever"});
   assertEqual( "autor1", idx.getAuthorById(1).getName());
   assertEqual( {"cozinhar", "escrever"}, idx.getAuthorById(1).getInterests());


   idx.insertAuthor("autor2", {"estudar", "voar"});
   assertEqual( "autor2", idx.getAuthorById(2).getName());
   assertEqual( {"estudar", "voar"}, idx.getAuthorById(2).getInterests());

   idx.addInterestsToAuthor(2, {"futebol", "praia"});
   idx.addInterestsToAuthor(2, {"golf"});
   assertEqual( {"estudar", "voar", "futebol","praia", "golf"}, idx.getAuthorById(2).
       getInterests());

   idx.removeInterestsFromAuthor(2, {"voar"});
   assertEqual( {"estudar", "futebol","praia", "golf"}, idx.getAuthorById(2).getInterests()
       );
   idx.removeInterestsFromAuthor(2, {"futebol","praia", "golf"});
   assertEqual( {"estudar"}, idx.getAuthorById(2).getInterests());
```

```
    );



public testCoautherPath: () ==> ()
   testCoautherPath() ==
   (dcl idx : Indexer := new Indexer();

    --inserir autores
    idx.insertAuthor("autor1", {"cozinhar", "escrever"});
    idx.insertAuthor("autor2", {"estudar", "voar"});

    idx.insertAuthor("autor3", {"correr", "cantar"});
    idx.insertAuthor("autor4", {"estudar", "voar"});
    idx.insertAuthor("autor5", {"brincar", "animar"});
    idx.insertAuthor("autor6", {"calcular", "cantar"});

    --inserir afiliacoes
    idx.insertAffiliation("FEUP");
    idx.insertAffiliation("FAC");

    assertEqual(2,card dom idx.getAffiliations());

    --inserir publicacoes
    idx.insertPublication("A Specifier's Introduction to Formal Methods", 1990, 0901, "
        11.111.11/ISBN", 100, 1, {2,3}, 1,{});
    idx.insertPublication("A Specifier's Introduction to Informal Methods", 1991, 0902, "
        11.111.12/ISBN", 10, 1, {3,4}, 2,{1});
    idx.insertPublication("A Specifier's Introduction to Informal Math", 1993, 0903, "
        11.111.13/ISBN", 13, 1, {5,6,2}, 2, {1,2});


    assertEqual(3,card dom idx.getPublications());
    assertEqual(3,card idx.getPublicationsByAuthor(1));
    assertEqual(7,len idx.getCoauthorPath(1));

  );



   public testCountSelfCitations: () ==> ()
   testCountSelfCitations() ==
   (dcl idx : Indexer := new Indexer();

   --inserir autores
    idx.insertAuthor("autor1", {"cozinhar", "escrever"});

    --inserir afiliacoes
    idx.insertAffiliation("FEUP");
    idx.insertAffiliation("FAC");

    --inserir publicacoes
    idx.insertPublication("A Specifier's Introduction to Formal Methods", 1990, 0901, "
        11.111.11/ISBN", 100, 1, {}, 1,{});
    idx.insertPublication("A Specifier's Introduction to Informal Methods", 1991, 0902, "
        11.111.12/ISBN", 10, 1, {}, 2,{1});
    idx.insertPublication("A Specifier's Introduction to Informal Math", 1993, 0903, "
        11.111.13/ISBN", 13, 1, {}, 2, {1,2});


    assertEqual(2,card idx.getReferencesFromAuthorPublications(idx.getPublicationsByAuthor(1)
        ));
   );



   public testCountOthersCitations: () ==> ()
   testCountOthersCitations() ==
   (dcl idx : Indexer := new Indexer();
```

```
    --inserir autores
     idx.insertAuthor("autor1", {"cozinhar", "escrever"});
     idx.insertAuthor("autor2", {"estudar", "voar"});
     idx.insertAuthor("autor3", {"correr", "cantar"});

     --inserir afiliacoes
     idx.insertAffiliation("FEUP");
     idx.insertAffiliation("FAC");

     --inserir publicacoes
     idx.insertPublication("A Specifier's Introduction to Formal Methods", 1990, 0901, "
         11.111.11/ISBN", 100, 1, {}, 1,{});
     idx.insertPublication("A Specifier's Introduction to Informal Methods", 1991, 0902, "
         11.111.12/ISBN", 10, 2, {}, 2,{1});
     idx.insertPublication("A Specifier's Introduction to Informal Math", 1993, 0903, "
         11.111.13/ISBN", 13, 3, {}, 2, {1,2});

     assertEqual(1, idx.getTimesCited(2));
     assertEqual(2, idx.getTimesCited(1));

    );



    public testNumberErdosAuthor: () ==> ()
    testNumberErdosAuthor() ==
    (dcl idx : Indexer := new Indexer();

     --inserir autores
     idx.insertAuthor("autor1", {"cozinhar", "escrever"});
     idx.insertAuthor("autor2", {"estudar", "voar"});
     idx.insertAuthor("autor3", {"correr", "cantar"});

     idx.getAuthors()(1).setErdos(1);
     assertEqual(1,idx.getAuthors()(1).getErdos());

     idx.getAuthors()(2).setErdos(100);
     idx.getAuthors()(3).setErdos(3000);

          --inserir afiliacoes
     idx.insertAffiliation("FEUP");

     idx.insertPublication("A Specifier's Introduction to Formal Methods", 1990, 0901, "
         11.111.11/ISBN", 100, 1, {2,3}, 1,{});


      assertEqual(2,idx.getAuthors()(2).getErdos());
       assertEqual(2,idx.getAuthors()(3).getErdos());
    );



public testAll: () ==> ()
  testAll() == (
   testAffiliation();
   testAffiliationChangeName();
   testAuthor();
   testAuthorChangeName();
   testAddInterestsAuthor();
   testRemoveInterestsAuthor();
   testFieldOfStudy();
   testPublication();
   testRaiseRankBy();
   testDecreaseRankBy();
   testPublicationAuthorAffiliation();
   testPublicationAuthorAffiliationChangeName();
   testPublicationKeyword();
   testPublicationKeyword();
```

```
   testPublicationReferences();
   testPublicationReferencesAddReference();
   testPublicationReferencesRemoveReference();
   testPublicationURL();
   testIndexerEmpty();
   testIndexerAuthors();
   testCoautherPath();
   testCountSelfCitations();
   testCountOthersCitations();
   testNumberErdosAuthor();

  );

end TestIndexer
```

4 Validação do Modelo - Testes

Os testes de seguida apresentados foram executados com sucesso na totalidade.

4.1 TestIndexer

```
class TestIndexer is subclass of MyTestCase

operations
-- TODO Define operations here



 public testAffiliation: () ==> ()
  testAffiliation() ==
  (dcl af: Affiliation := new Affiliation("FEUP");
   assertEqual("FEUP", af.getName());
  );


 public testAffiliationChangeName: () ==> ()
  testAffiliationChangeName() ==
  (dcl af: Affiliation := new Affiliation("FEUP");
   assertEqual("FEUP", af.getName());
   af.changeName("MIT");
   assertEqual("MIT", af.getName());
  );




  public testAuthor: () ==> ()
   testAuthor() ==
   (dcl a: Author := new Author("Andre", {"amputacoes", "desmembramentos"});
    assertEqual("Andre", a.getName());
    assertEqual({"amputacoes", "desmembramentos"}, a.getInterests());
   );




 public testAuthorChangeName: () ==> ()
  testAuthorChangeName() ==
  (dcl a: Author := new Author("Andre", {"amputacoes", "desmembramentos"});
   assertEqual("Andre", a.getName());
    a.changeName("Banana");
    assertEqual("Banana", a.getName());
   );
```

```
public testAddInterestsAuthor: () ==> ()
 testAddInterestsAuthor() ==
 (dcl a: Author := new Author("Andre", {"amputacoes", "desmembramentos"});
  assertEqual({"amputacoes", "desmembramentos"}, a.getInterests());
  a.addInterests({"mutilacoes", "imolacoes"});
  assertEqual({"amputacoes", "desmembramentos", "mutilacoes", "imolacoes"}, a.getInterests()
      );
 );


 public testRemoveInterestsAuthor: () ==> ()
  testRemoveInterestsAuthor() ==
  (dcl a: Author := new Author("Andre", {"amputacoes", "desmembramentos"});
   assertEqual({"amputacoes", "desmembramentos"}, a.getInterests());
   a.removeInterests({"amputacoes"});
   assertEqual({"desmembramentos"}, a.getInterests());
   a.removeInterests({"desmembramentos"});
   assertEqual({}, a.getInterests());
  );


 public testFieldOfStudy: () ==> ()
  testFieldOfStudy() ==
  (dcl fos: FieldOfStudy:= new FieldOfStudy("Necromancia");
   assertEqual("Necromancia", fos.getName());
  );


 public testPublication: () ==> ()
  testPublication() ==
  (dcl pub: Publication := new Publication("A Specifier's Introduction to Formal Methods",
                       1990, 0901, "11.111.11/ISBN", 100);
   assertEqual("A Specifier's Introduction to Formal Methods", pub.getOriginalTitle());
   assertEqual(1990, pub.getPublishYear());
   assertEqual(0901, pub.getPublishDate());
   assertEqual("11.111.11/ISBN", pub.getDoi());
   assertEqual(100, pub.getRank());
  );


 public testRaiseRankBy: () ==> ()
  testRaiseRankBy() ==
  (dcl pub: Publication := new Publication("A Specifier's Introduction to Formal Methods",
                       1990, 0901, "11.111.11/ISBN", 100);
   assertEqual(100, pub.getRank());
   pub.raiseRankBy(50);
   assertEqual(50, pub.getRank());

  );


 public testDecreaseRankBy: () ==> ()
  testDecreaseRankBy() ==
  (dcl pub: Publication := new Publication("A Specifier's Introduction to Formal Methods",
                       1990, 0901, "11.111.11/ISBN", 100);
   assertEqual(100, pub.getRank());
   pub.decreaseRankBy(50);
   assertEqual(150, pub.getRank());

  );


 public testPublicationAuthorAffiliation: () ==> ()
  testPublicationAuthorAffiliation() ==
  (dcl paa: PublicationAuthorAffiliation := new PublicationAuthorAffiliation(1, 2, {1, 3,
      4}, 3 , "FEUP");
   assertEqual(1, paa.getPubid());
   assertEqual(2, paa.getAutid());
```

```
 assertEqual(3, paa.getAffid());
 assertEqual("FEUP", paa.getOriAffName());

);


public testPublicationAuthorAffiliationChangeName: () ==> ()
testPublicationAuthorAffiliationChangeName() ==
(dcl paa: PublicationAuthorAffiliation := new PublicationAuthorAffiliation(1, 2, {1, 3,
    4}, 3,  "FEUP");
 assertEqual("FEUP", paa.getOriAffName());
 paa.changeOriAffName("MIT");
 assertEqual("MIT", paa.getOriAffName());

);


public testPublicationKeyword: () ==> ()
testPublicationKeyword() ==
(dcl pk: PublicationKeyword := new PublicationKeyword(1, 2, "MFES");
 assertEqual(1, pk.getPid());
 assertEqual(2, pk.getFosid());
 assertEqual("MFES", pk.getKeyword());
);


public testPublicationKeywordChangeKeyword: () ==> ()
testPublicationKeywordChangeKeyword() ==
(dcl pk: PublicationKeyword := new PublicationKeyword(1, 2, "MFES");
 pk.changeKeyword("SEFM");
 assertEqual("SEFM", pk.getKeyword());
);


public testPublicationReferences: () ==> ()
testPublicationReferences() ==
(dcl pr: PublicationReferences := new PublicationReferences(1,{2, 3});
     assertEqual(1, pr.getPid());
     -- da para fazer assert error ou assim?
     -- assertEqual({1, 2}, pr.getPrids());
     assertEqual({2, 3}, pr.getPrids());


);


public testPublicationReferencesAddReference: () ==> ()
testPublicationReferencesAddReference() ==
(dcl pr: PublicationReferences := new PublicationReferences(1,{2, 3});
     assertEqual(1, pr.getPid());
     -- da para fazer assert error ou assim?
     -- assertEqual({1, 2}, pr.getPrids());
     assertEqual({2, 3}, pr.getPrids());
     --pr.addReference(2);
     pr.addReference(4);
     assertEqual({2, 3, 4}, pr.getPrids());

);


public testPublicationReferencesRemoveReference: () ==> ()
testPublicationReferencesRemoveReference() ==
(dcl pr: PublicationReferences := new PublicationReferences(1,{2, 3, 4});
     assertEqual(1, pr.getPid());
     -- da para fazer assert error ou assim?
     -- assertEqual({1, 2}, pr.getPrids());
     assertEqual({2, 3, 4}, pr.getPrids());
     --pr.addReference(2);
     pr.removeReference(4);
     assertEqual({2, 3}, pr.getPrids());
```

```
    );


    public testPublicationURL: () ==> ()
    testPublicationURL() ==
    (dcl purl: PublicationURL := new PublicationURL(1,"mypublicatonurl.com" );
        assertEqual(1, purl.getPid());
        assertEqual("mypublicatonurl.com", purl.getUrl());
   );


     public testIndexerEmpty: () ==> ()
     testIndexerEmpty() ==
     (dcl idx: Indexer := new Indexer();
      assertEqual({|->}, idx.getPublications());
      assertEqual({|->}, idx.getAuthors());
      assertEqual({|->}, idx.getAffiliations());
      assertEqual({|->}, idx.getFieldsofstudy());
      assertEqual({|->}, idx.getPublicationkeywords());
      assertEqual({|->}, idx.getPublicationauthoraffiliations());
      assertEqual({|->}, idx.getPublicationsreferences());
      assertEqual({|->}, idx.getPublicationurls());

     );


     public testIndexerAuthors: () ==> ()
     testIndexerAuthors() ==
     (dcl idx: Indexer := new Indexer();
      idx.insertAuthor("autor1", {"cozinhar", "escrever"});
      assertEqual( "autor1", idx.getAuthorById(1).getName());
      assertEqual( {"cozinhar", "escrever"}, idx.getAuthorById(1).getInterests());


      idx.insertAuthor("autor2", {"estudar", "voar"});
      assertEqual( "autor2", idx.getAuthorById(2).getName());
      assertEqual( {"estudar", "voar"}, idx.getAuthorById(2).getInterests());

      idx.addInterestsToAuthor(2, {"futebol", "praia"});
      idx.addInterestsToAuthor(2, {"golf"});
      assertEqual( {"estudar", "voar", "futebol","praia", "golf"}, idx.getAuthorById(2).
          getInterests());

      idx.removeInterestsFromAuthor(2, {"voar"});
      assertEqual( {"estudar", "futebol","praia", "golf"}, idx.getAuthorById(2).getInterests()
          );
      idx.removeInterestsFromAuthor(2, {"futebol","praia", "golf"});
      assertEqual( {"estudar"}, idx.getAuthorById(2).getInterests());
      );



public testCoautherPath: () ==> ()
    testCoautherPath() ==
    (dcl idx : Indexer := new Indexer();

     --inserir autores
     idx.insertAuthor("autor1", {"cozinhar", "escrever"});
     idx.insertAuthor("autor2", {"estudar", "voar"});

     idx.insertAuthor("autor3", {"correr", "cantar"});
     idx.insertAuthor("autor4", {"estudar", "voar"});
     idx.insertAuthor("autor5", {"brincar", "animar"});
     idx.insertAuthor("autor6", {"calcular", "cantar"});

     --inserir afiliacoes
     idx.insertAffiliation("FEUP");
     idx.insertAffiliation("FAC");
```

```
  assertEqual(2,card dom idx.getAffiliations());

  --inserir publicacoes
  idx.insertPublication("A Specifier's Introduction to Formal Methods", 1990, 0901, "
      11.111.11/ISBN", 100, 1, {2,3}, 1,{});
  idx.insertPublication("A Specifier's Introduction to Informal Methods", 1991, 0902, "
      11.111.12/ISBN", 10, 1, {3,4}, 2,{1});
  idx.insertPublication("A Specifier's Introduction to Informal Math", 1993, 0903, "
      11.111.13/ISBN", 13, 1, {5,6,2}, 2, {1,2});


  assertEqual(3,card dom idx.getPublications());
  assertEqual(3,card idx.getPublicationsByAuthor(1));
  assertEqual(7,len idx.getCoauthorPath(1));

);



 public testCountSelfCitations: () ==> ()
 testCountSelfCitations() ==
 (dcl idx : Indexer := new Indexer();

 --inserir autores
  idx.insertAuthor("autor1", {"cozinhar", "escrever"});

  --inserir afiliacoes
  idx.insertAffiliation("FEUP");
  idx.insertAffiliation("FAC");

  --inserir publicacoes
  idx.insertPublication("A Specifier's Introduction to Formal Methods", 1990, 0901, "
      11.111.11/ISBN", 100, 1, {}, 1,{});
  idx.insertPublication("A Specifier's Introduction to Informal Methods", 1991, 0902, "
      11.111.12/ISBN", 10, 1, {}, 2,{1});
  idx.insertPublication("A Specifier's Introduction to Informal Math", 1993, 0903, "
      11.111.13/ISBN", 13, 1, {}, 2, {1,2});


  assertEqual(2,card idx.getReferencesFromAuthorPublications(idx.getPublicationsByAuthor(1)
      ));
 );



 public testCountOthersCitations: () ==> ()
 testCountOthersCitations() ==
 (dcl idx : Indexer := new Indexer();

 --inserir autores
  idx.insertAuthor("autor1", {"cozinhar", "escrever"});
  idx.insertAuthor("autor2", {"estudar", "voar"});
  idx.insertAuthor("autor3", {"correr", "cantar"});

  --inserir afiliacoes
  idx.insertAffiliation("FEUP");
  idx.insertAffiliation("FAC");

  --inserir publicacoes
  idx.insertPublication("A Specifier's Introduction to Formal Methods", 1990, 0901, "
      11.111.11/ISBN", 100, 1, {}, 1,{});
  idx.insertPublication("A Specifier's Introduction to Informal Methods", 1991, 0902, "
      11.111.12/ISBN", 10, 2, {}, 2,{1});
  idx.insertPublication("A Specifier's Introduction to Informal Math", 1993, 0903, "
      11.111.13/ISBN", 13, 3, {}, 2, {1,2});

  assertEqual(1, idx.getTimesCited(2));
  assertEqual(2, idx.getTimesCited(1));
```

```
    );



    public testNumberErdosAuthor: () ==> ()
    testNumberErdosAuthor() ==
    (dcl idx : Indexer := new Indexer();

     --inserir autores
     idx.insertAuthor("autor1", {"cozinhar", "escrever"});
     idx.insertAuthor("autor2", {"estudar", "voar"});
     idx.insertAuthor("autor3", {"correr", "cantar"});

     idx.getAuthors()(1).setErdos(1);
     assertEqual(1,idx.getAuthors()(1).getErdos());

     idx.getAuthors()(2).setErdos(100);
     idx.getAuthors()(3).setErdos(3000);

          --inserir afiliacoes
     idx.insertAffiliation("FEUP");

     idx.insertPublication("A Specifier's Introduction to Formal Methods", 1990, 0901, "
         11.111.11/ISBN", 100, 1, {2,3}, 1,{});


      assertEqual(2,idx.getAuthors()(2).getErdos());
       assertEqual(2,idx.getAuthors()(3).getErdos());
    );



public testAll: () ==> ()
  testAll() == (
   testAffiliation();
   testAffiliationChangeName();
   testAuthor();
   testAuthorChangeName();
   testAddInterestsAuthor();
   testRemoveInterestsAuthor();
   testFieldOfStudy();
   testPublication();
   testRaiseRankBy();
   testDecreaseRankBy();
   testPublicationAuthorAffiliation();
   testPublicationAuthorAffiliationChangeName();
   testPublicationKeyword();
   testPublicationKeyword();
   testPublicationReferences();
   testPublicationReferencesAddReference();
   testPublicationReferencesRemoveReference();
   testPublicationURL();
   testIndexerEmpty();
   testIndexerAuthors();
   testCoautherPath();
   testCountSelfCitations();
   testCountOthersCitations();
   testNumberErdosAuthor();

  );

end TestIndexer
```

4.2 MyTestCase

```
class MyTestCase
/*
  Superclass for test classes, simpler but more practical than VDMUnit`TestCase.
  For proper use, you have to do: New -> Add VDM Library -> IO.
  FEUP, MFES, 2015/16.
*/

operations

 -- Simulates assertion checking by reducing it to pre-condition checking.
 -- If 'arg' does not hold, a pre-condition violation will be signaled.

 protected assertTrue: bool ==> ()
 assertTrue(arg) ==
  return
 pre arg;

 -- Simulates assertion checking by reducing it to post-condition checking.
 -- If values are not equal, prints a message in the console and generates
 -- a post-conditions violation.

 protected assertEqual: ? * ? ==> ()
 assertEqual(expected, actual) ==
  if expected <> actual then (
    IO`print("Actual value (");
    IO`print(actual);
    IO`print(") different from expected (");
    IO`print(expected);
    IO`println(")\n")
  )
 post expected = actual

end MyTestCase
```

4.3 Resultados

Em seguida serão apresentados os quatro testes mais importantes, corridos em separado, bem como a prova de que todos os testes são bem sucedido, quando corridos simultâneamente.

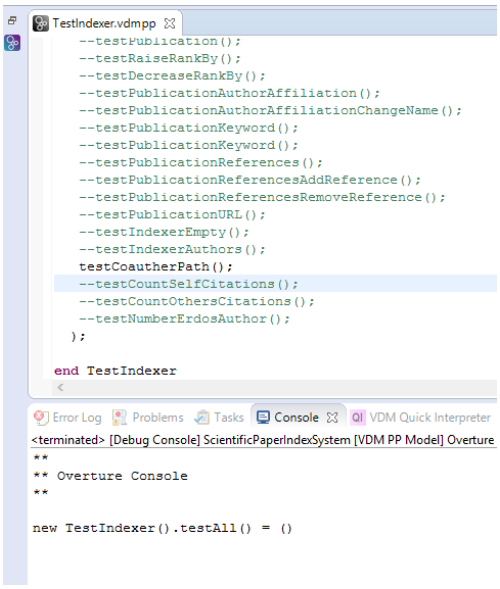

Figura 3: Teste do caminho do co-autor

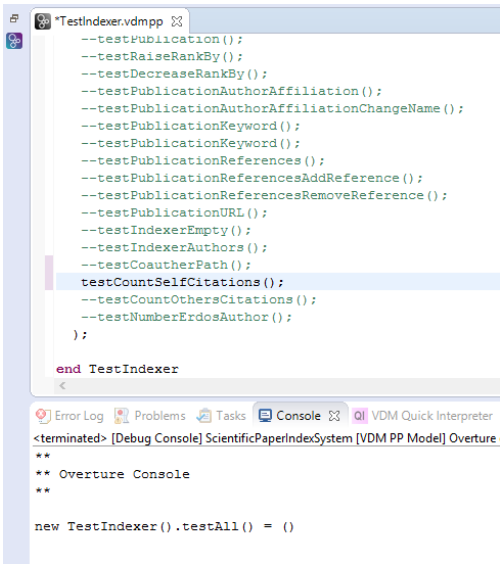

Figura 4: Teste da contagem das citações do autor nas próprias publicações

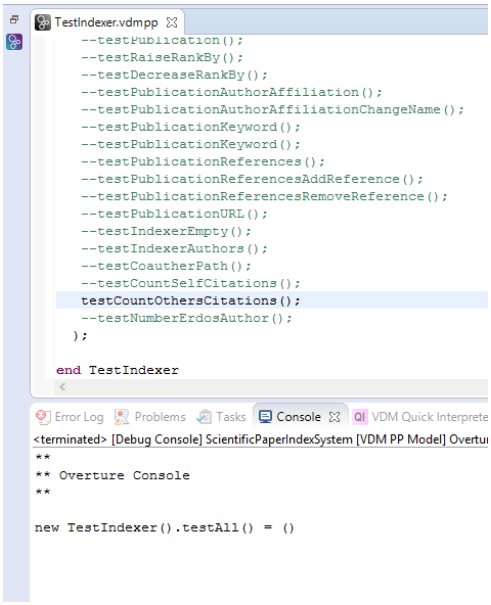

Figura 5: Teste da contagem das minhas citações noutras publicações

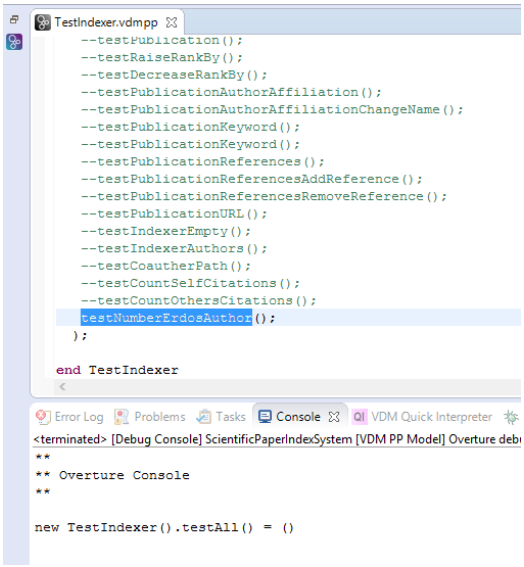

Figura 6: Teste da verificação do numbero de Erdős

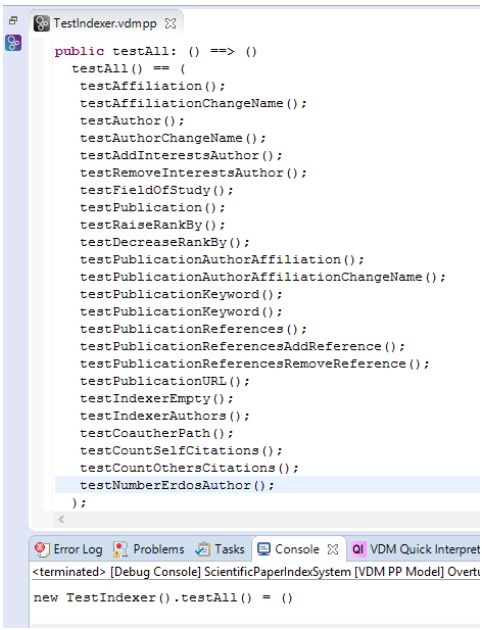

Figura 7: Execução e aceitação de todos os testes elaborados

5 Verificação do Modelo - Análise de Consistência

Para analisar a consistência do programa, isto é, verificar que as *pre*, *post* e *invariants* condições são realmente compridas foram feitos testes que tentam violar as mesmas e foi obtido insucesso nos mesmos testes o que significa que o programa é realmente consistente.
Abaixo pode-se encontrar algumas imagens que refletem estes mesmos testes:

No seguinte teste:

```
public testPublicationReferencesAddReference: () ==> ()
testPublicationReferencesAddReference() ==
(dcl pr: PublicationReferences := new PublicationReferences(1,{2, 3});
        assertEqual(1, pr.getPid());
        -- da para fazer assert error ou assim?
        assertEqual({1, 2}, pr.getPrids());
        assertEqual({2, 3}, pr.getPrids());
        pr.addReference(2);
        pr.addReference(4);
        assertEqual({2, 3, 4}, pr.getPrids());

);
```

Figura 8: Execução e aceitação de todos os testes elaborados

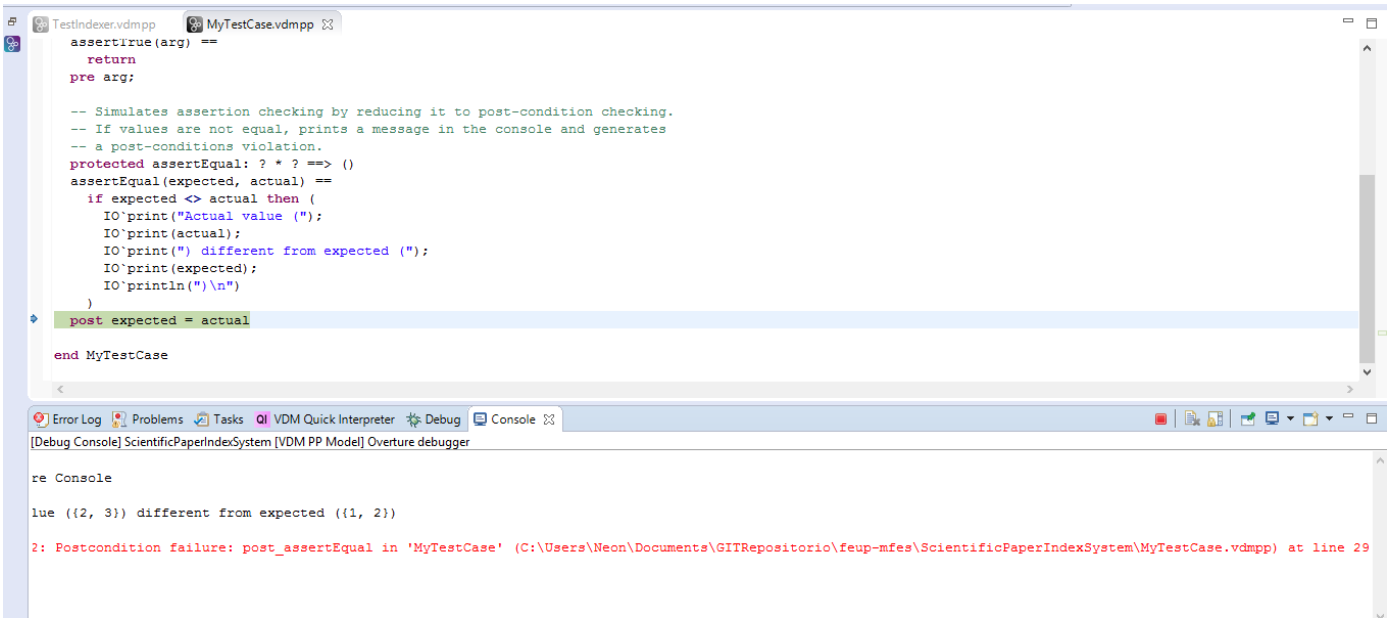

Figura 9: Execução e aceitação de todos os testes elaborados

A pós-condição de uma publicação não pode fazer referência de outra publicação se a data desta for depois da primeira é violada

6 Transformação do Código para Java

6.1 Affiliation

```
package ScientificPaperIndexSystem;

import org.overture.codegen.runtime.*;

import java.util.*;


@SuppressWarnings("all")
public class Affiliation {
    private String name = SeqUtil.toStr(SeqUtil.seq());

    public Affiliation(final String an) {
        cg_init_Affiliation_1(an);
    }

    public Affiliation() {
    }

    public void cg_init_Affiliation_1(final String an) {
        name = an;

        return;
    }

    public void changeName(final String an) {
        name = an;
    }

    public String getName() {
        return name;
    }

    public String toString() {
        return "Affiliation{" + "name := " + Utils.toString(name) + "}";
    }
}
```

6.2 appendQuote

```
package ScientificPaperIndexSystem.quotes;

import org.overture.codegen.runtime.*;

import java.util.*;


@SuppressWarnings("all")
public class appendQuote {
    private static int hc = 0;
    private static appendQuote instance = null;

    public appendQuote() {
```

```
        if (Utils.equals(hc, 0)) {
            hc = super.hashCode();
        }
    }

    public static appendQuote getInstance() {
        if (Utils.equals(instance, null)) {
            instance = new appendQuote();
        }

        return instance;
    }

    public int hashCode() {
        return hc;
    }

    public boolean equals(final Object obj) {
        return obj instanceof appendQuote;
    }

    public String toString() {
        return "<append>";
    }
}
```

6.3 Author

```
package ScientificPaperIndexSystem;

import org.overture.codegen.runtime.*;

import java.util.*;


@SuppressWarnings("all")
public class Author {
    private String name = SeqUtil.toStr(SeqUtil.seq());
    private VDMSet interests = SetUtil.set();
    private Number erdos = 9999L;

    public Author(final String an, final VDMSet ints) {
        cg_init_Author_1(an, Utils.copy(ints));
    }

    public Author(final String an, final VDMSet ints, final Number erdsnum) {
        cg_init_Author_2(an, Utils.copy(ints), erdsnum);
    }

    public Author() {
    }

    public void cg_init_Author_1(final String an, final VDMSet ints) {
        name = an;
        interests = Utils.copy(ints);
```

```
        return;
    }

    public void cg_init_Author_2(final String an, final VDMSet ints,
        final Number erdsnum) {
        name = an;
        interests = Utils.copy(ints);
        erdos = erdsnum;

        return;
    }

    public void changeName(final String an) {
        name = an;
    }

    public void addInterest(final String intr) {
        interests = SetUtil.union(Utils.copy(interests), SetUtil.set(intr));
    }

    public void addInterests(final VDMSet intr) {
        interests = SetUtil.union(Utils.copy(interests), Utils.copy(intr));
    }

    public void removeInterests(final VDMSet intrs) {
        interests = SetUtil.diff(Utils.copy(interests), Utils.copy(intrs));
    }

    public Number getErdos() {
        return erdos;
    }

    public void setErdos(final Number en) {
        erdos = en;
    }

    public String getName() {
        return name;
    }

    public VDMSet getInterests() {
        return Utils.copy(interests);
    }

    public String toString() {
        return "Author{" + "name := " + Utils.toString(name) +
        ", interests := " + Utils.toString(interests) + ", erdos := " +
        Utils.toString(erdos) + "}";
    }
}
```

6.4 FieldOfStudy

```
package ScientificPaperIndexSystem;

import org.overture.codegen.runtime.*;
```

```
import java.util.*;


@SuppressWarnings("all")
public class FieldOfStudy {
    private String name = SeqUtil.toStr(SeqUtil.seq());

    public FieldOfStudy(final String an) {
        cg_init_FieldOfStudy_1(an);
    }

    public FieldOfStudy() {
    }

    public void cg_init_FieldOfStudy_1(final String an) {
        name = an;

        return;
    }

    public void changeName(final String an) {
        name = an;
    }

    public String getName() {
        return name;
    }

    public String toString() {
        return "FieldOfStudy{" + "name := " + Utils.toString(name) + "}";
    }
}
```

6.5 Indexer

```
package ScientificPaperIndexSystem;

import org.overture.codegen.runtime.*;

import java.util.*;


@SuppressWarnings("all")
public class Indexer {
    private VDMMap publications = MapUtil.map();
    private Number publicationIdCounter = 0L;
    private VDMMap authors = MapUtil.map();
    private Number authorIdCounter = 0L;
    private VDMMap affiliations = MapUtil.map();
    private Number affiliationIdCounter = 0L;
    private VDMMap fieldsofstudy = MapUtil.map();
    private Number fieldofstudyIdCounter = 0L;
    private VDMMap publicationkeywords = MapUtil.map();
    private Number publicationkeywordIdCounter = 0L;
    private VDMMap publicationauthoraffiliations = MapUtil.map();
```

```
private Number paaIdCounter = 0L;
private VDMMap publicationsreferences = MapUtil.map();
private Number pReferencesIdCounter = 0L;
private VDMMap publicationurls = MapUtil.map();

public Indexer(final VDMMap pubs, final VDMMap auts, final VDMMap affs,
    final VDMMap fsos, final VDMMap pks, final VDMMap paas,
    final VDMMap prs, final VDMMap pus) {
    cg_init_Indexer_1(Utils.copy(pubs), Utils.copy(auts), Utils.copy(affs),
        Utils.copy(fsos), Utils.copy(pks), Utils.copy(paas),
        Utils.copy(prs), Utils.copy(pus));
}

public Indexer() {
    cg_init_Indexer_2();
}

public void cg_init_Indexer_1(final VDMMap pubs, final VDMMap auts,
    final VDMMap affs, final VDMMap fsos, final VDMMap pks,
    final VDMMap paas, final VDMMap prs, final VDMMap pus) {
    publications = Utils.copy(pubs);
    authors = Utils.copy(auts);
    affiliations = Utils.copy(affs);
    fieldsofstudy = Utils.copy(fsos);
    publicationkeywords = Utils.copy(pks);
    publicationauthoraffiliations = Utils.copy(paas);
    publicationsreferences = Utils.copy(prs);
    publicationurls = Utils.copy(pus);

    return;
}

public void cg_init_Indexer_2() {
    return;
}

public VDMMap getPublications() {
    return Utils.copy(publications);
}

public VDMMap getAuthors() {
    return Utils.copy(authors);
}

public VDMMap getAffiliations() {
    return Utils.copy(affiliations);
}

public VDMMap getFieldsofstudy() {
    return Utils.copy(fieldsofstudy);
}

public VDMMap getPublicationkeywords() {
    return Utils.copy(publicationkeywords);
}
```

```
public VDMMap getPublicationauthoraffiliations() {
    return Utils.copy(publicationauthoraffiliations);
}

public VDMMap getPublicationsreferences() {
    return Utils.copy(publicationsreferences);
}

public VDMMap getPublicationurls() {
    return Utils.copy(publicationurls);
}

public void insertAuthor(final String an, final VDMSet ints) {
    authorIdCounter = authorIdCounter.longValue() + 1L;
    authors = MapUtil.munion(Utils.copy(authors),
            MapUtil.map(
                new Maplet(authorIdCounter, new Author(an, Utils.copy(ints)))));
}

public void insertAffiliation(final String affiname) {
    affiliationIdCounter = affiliationIdCounter.longValue() + 1L;
    affiliations = MapUtil.munion(Utils.copy(affiliations),
            MapUtil.map(
                new Maplet(affiliationIdCounter, new Affiliation(affiname))));
}

public Author getAuthorById(final Number id) {
    return ((Author) Utils.get(authors, id));
}

public void addInterestsToAuthor(final Number aid, final VDMSet intr) {
    Author author = getAuthorById(aid);
    author.addInterests(Utils.copy(intr));
    authors = MapUtil.munion(Utils.copy(authors),
            MapUtil.map(new Maplet(aid, author)));
}

public void removeInterestsFromAuthor(final Number aid, final VDMSet intrs) {
    Author author = getAuthorById(aid);
    author.removeInterests(Utils.copy(intrs));
    authors = MapUtil.override(Utils.copy(authors),
            MapUtil.map(new Maplet(aid, author)));
}

public void insertPublication(final String ortitle, final Number pubyear,
    final Number pubdate, final String pubdoi, final Number pubrank,
    final Number idau, final VDMSet idcoauts, final Number idaf,
    final VDMSet idrefs) {
    publicationIdCounter = publicationIdCounter.longValue() + 1L;
    publications = MapUtil.munion(Utils.copy(publications),
            MapUtil.map(
                new Maplet(publicationIdCounter,
                    new Publication(ortitle, pubyear, pubdate, pubdoi,
                        pubrank))));
    paaIdCounter = paaIdCounter.longValue() + 1L;
    publicationauthoraffiliations = MapUtil.munion(Utils.copy(
```

```
                publicationauthoraffiliations),
            MapUtil.map(
                new Maplet(paaIdCounter,
                    new PublicationAuthorAffiliation(publicationIdCounter,
                        idau, Utils.copy(idcoauts), idaf,
                        ((Affiliation) Utils.get(affiliations, idaf)).getName()))));
        insertReferences(publicationIdCounter, Utils.copy(idrefs));

        {
            VDMSet everypubauthor = SetUtil.union(SetUtil.set(idau),
                    Utils.copy(idcoauts));
            Number minerdos = 999L;
            Number minerdosauthorid = 9999999L;

            for (Iterator iterator_3 = everypubauthor.iterator();
                    iterator_3.hasNext();) {
                Number pubauthor = (Number) iterator_3.next();

                if (((Author) Utils.get(authors, pubauthor)).getErdos()
                        .longValue() < minerdos.longValue()) {
                    minerdos = ((Author) Utils.get(authors, pubauthor)).getErdos();
                    minerdosauthorid = pubauthor;
                }
            }

            for (Iterator iterator_4 = everypubauthor.iterator();
                    iterator_4.hasNext();) {
                Number pubauthor = (Number) iterator_4.next();

                if (!(Utils.equals(pubauthor, minerdosauthorid))) {
                    ((Author) Utils.get(authors, pubauthor)).setErdos(minerdos.longValue
                        1L);
                }
            }
        }
    }

    public VDMSet getPublicationsByAuthor(final Number ida) {
        VDMSet pubs = SetUtil.set();

        for (Iterator iterator_5 = MapUtil.rng(Utils.copy(
                        publicationauthoraffiliations)).iterator();
                iterator_5.hasNext();) {
            PublicationAuthorAffiliation pub = (PublicationAuthorAffiliation) iterator_5

            if (Utils.equals(pub.getAutid(), ida)) {
                pubs = SetUtil.union(Utils.copy(pubs), SetUtil.set(pub));
            }
        }

        return Utils.copy(pubs);
    }

    public Number getAuthorByPublication(final Number idp) {
        Number aid = 0L;
```

```
        for (Iterator iterator_6 = MapUtil.rng(Utils.copy(
                        publicationauthoraffiliations)).iterator();
                iterator_6.hasNext();) {
            PublicationAuthorAffiliation pub = (PublicationAuthorAffiliation) iterator_6

            if (Utils.equals(idp, pub.getPubid())) {
                aid = pub.getAutid();
            }
        }

        return aid;
    }

    public VDMSet getReferencesFromAuthorPublications(final VDMSet pubs) {
        VDMSet idpubs = SetUtil.set();

        for (Iterator iterator_7 = pubs.iterator(); iterator_7.hasNext();) {
            PublicationAuthorAffiliation pub = (PublicationAuthorAffiliation) iterator_7
            idpubs = SetUtil.union(Utils.copy(idpubs),
                    SetUtil.set(pub.getPubid()));
        }

        {
            VDMSet idrefs = SetUtil.set();

            for (Iterator iterator_8 = MapUtil.rng(Utils.copy(
                            publicationsreferences)).iterator();
                    iterator_8.hasNext();) {
                PublicationReferences id = (PublicationReferences) iterator_8.next();
                idrefs = SetUtil.union(Utils.copy(idrefs), id.getPrids());
            }

            return Utils.copy(idrefs);
        }
    }

    public VDMSeq getCoauthorPath(final Number aid) {
        VDMSet pubs = getPublicationsByAuthor(aid);
        VDMSeq coauthpath = SeqUtil.seq();

        for (Iterator iterator_9 = pubs.iterator(); iterator_9.hasNext();) {
            PublicationAuthorAffiliation pub = (PublicationAuthorAffiliation) iterator_9
            VDMSet coauths = pub.getCoautids();

            for (Iterator iterator_10 = coauths.iterator();
                    iterator_10.hasNext();) {
                Number coauth = (Number) iterator_10.next();
                Path path = new Path(pub.getPubid(),
                        ((Author) Utils.get(authors, coauth)).getName());
                coauthpath = SeqUtil.conc(Utils.copy(coauthpath),
                        SeqUtil.seq(path));
            }
        }

        return Utils.copy(coauthpath);
    }
```

```
public Number getNumberSelfCitations(final Number ida) {
    VDMSet selfpubs = getPublicationsByAuthor(ida);

    {
        VDMSet idrefs = getReferencesFromAuthorPublications(Utils.copy(
                selfpubs));

        {
            VDMSet idselfref = SetUtil.set();

            for (Iterator iterator_11 = idrefs.iterator();
                    iterator_11.hasNext();) {
                Number refs = (Number) iterator_11.next();

                if (Utils.equals(ida, getAuthorByPublication(refs))) {
                    idselfref = SetUtil.union(Utils.copy(idselfref),
                            SetUtil.set(refs));
                }
            }

            return getReferencesFromAuthorPublications(Utils.copy(selfpubs))
                    .size();
        }
    }
}

public Number getTimesCited(final Number ida) {
    VDMSet authorpubs = getPublicationsByAuthor(ida);
    VDMSeq pubscited = SeqUtil.seq();

    for (Iterator iterator_12 = authorpubs.iterator();
            iterator_12.hasNext();) {
        PublicationAuthorAffiliation pubs = (PublicationAuthorAffiliation) iterator_

        for (Iterator iterator_13 = MapUtil.rng(Utils.copy(
                    publicationsreferences)).iterator();
                iterator_13.hasNext();) {
            PublicationReferences references = (PublicationReferences) iterator_13.n

            if (SetUtil.subset(SetUtil.set(pubs.getPubid()),
                    references.getPrids())) {
                pubscited = SeqUtil.conc(Utils.copy(pubscited),
                        SeqUtil.seq(pubs.getPubid()));
            }
        }
    }

    return pubscited.size();
}

public void insertReferences(final Number idpub, final VDMSet idrefs) {
    pReferencesIdCounter = pReferencesIdCounter.longValue() + 1L;
    publicationsreferences = MapUtil.munion(Utils.copy(
                publicationsreferences),
            MapUtil.map(
```

```
                    new Maplet(pReferencesIdCounter,
                        new PublicationReferences(idpub, Utils.copy(idrefs)))));
    }

    public String toString() {
        return "Indexer{" + "publications := " + Utils.toString(publications) +
        ", publicationIdCounter := " + Utils.toString(publicationIdCounter) +
        ", authors := " + Utils.toString(authors) + ", authorIdCounter := " +
        Utils.toString(authorIdCounter) + ", affiliations := " +
        Utils.toString(affiliations) + ", affiliationIdCounter := " +
        Utils.toString(affiliationIdCounter) + ", fieldsofstudy := " +
        Utils.toString(fieldsofstudy) + ", fieldofstudyIdCounter := " +
        Utils.toString(fieldofstudyIdCounter) + ", publicationkeywords := " +
        Utils.toString(publicationkeywords) +
        ", publicationkeywordIdCounter := " +
        Utils.toString(publicationkeywordIdCounter) +
        ", publicationauthoraffiliations := " +
        Utils.toString(publicationauthoraffiliations) + ", paaIdCounter := " +
        Utils.toString(paaIdCounter) + ", publicationsreferences := " +
        Utils.toString(publicationsreferences) + ", pReferencesIdCounter := " +
        Utils.toString(pReferencesIdCounter) + ", publicationurls := " +
        Utils.toString(publicationurls) + "}";
    }
}
```

6.6 Main

```
package ScientificPaperIndexSystem;

import org.overture.codegen.runtime.*;

import java.util.*;


@SuppressWarnings("all")
public class Main {
    public Main() {
    }

    public static void Run() {
        new TestIndexer().testAll();
    }

    public String toString() {
        return "Main{}";
    }

    public static void main(String[] args) {
        Run();
        IO.println(Utils.toString(Utils.VOID_VALUE));
    }
}
```

6.7 MyTestCase

```
package ScientificPaperIndexSystem;
```

```
import org.overture.codegen.runtime.*;

import java.util.*;


@SuppressWarnings("all")
public class MyTestCase {
    public MyTestCase() {
    }

    protected void assertTrue(final Boolean arg) {
        return;
    }

    protected void assertEqual(final Object expected, final Object actual) {
        if (!(Utils.equals(expected, actual))) {
            IO.print("Actual value (");
            IO.print(((Object) actual));
            IO.print(") different from expected (");
            IO.print(((Object) expected));
            IO.println(")\n");
        }
    }

    public String toString() {
        return "MyTestCase{}";
    }
}
```

6.8 Path

```
package ScientificPaperIndexSystem;

import org.overture.codegen.runtime.*;

import java.util.*;


@SuppressWarnings("all")
public class Path {
    private Number pubid = 0L;
    private String coauthname = SeqUtil.toStr(SeqUtil.seq());

    public Path(final Number pid, final String autname) {
        cg_init_Path_1(pid, autname);
    }

    public Path() {
    }

    public void cg_init_Path_1(final Number pid, final String autname) {
        pubid = pid;
        coauthname = autname;

        return;
```

```
    }

    public Number getpubid() {
        return pubid;
    }

    public String getcoauthname() {
        return coauthname;
    }

    public String toString() {
        return "Path{" + "pubid := " + Utils.toString(pubid) +
        ", coauthname := " + Utils.toString(coauthname) + "}";
    }
}
```

6.9 Publication

```
package ScientificPaperIndexSystem;

import org.overture.codegen.runtime.*;

import java.util.*;


@SuppressWarnings("all")
public class Publication {
    private String originalTitle = SeqUtil.toStr(SeqUtil.seq());
    private Number publishYear = 0L;
    private Number publishDate = 0L;
    private String doi = SeqUtil.toStr(SeqUtil.seq());
    private Number rank = 99999L;

    public Publication(final String opt, final Number ppy, final Number ppd,
        final String pdoi, final Number pr) {
        cg_init_Publication_1(opt, ppy, ppd, pdoi, pr);
    }

    public Publication() {
    }

    public void cg_init_Publication_1(final String opt, final Number ppy,
        final Number ppd, final String pdoi, final Number pr) {
        originalTitle = opt;
        publishYear = ppy;
        publishDate = ppd;
        doi = pdoi;
        rank = pr;

        return;
    }

    public void raiseRankBy(final Number nRanks) {
        rank = rank.longValue() - nRanks.longValue();
    }
```

```
    public void decreaseRankBy(final Number nRanks) {
        rank = rank.longValue() + nRanks.longValue();
    }

    public String getOriginalTitle() {
        return originalTitle;
    }

    public Number getPublishYear() {
        return publishYear;
    }

    public Number getPublishDate() {
        return publishDate;
    }

    public String getDoi() {
        return doi;
    }

    public Number getRank() {
        return rank;
    }

    public String toString() {
        return "Publication{" + "originalTitle := " +
        Utils.toString(originalTitle) + ", publishYear := " +
        Utils.toString(publishYear) + ", publishDate := " +
        Utils.toString(publishDate) + ", doi := " + Utils.toString(doi) +
        ", rank := " + Utils.toString(rank) + "}";
    }
}
```

6.10 PublicationAuthorAffiliation

```
package ScientificPaperIndexSystem;

import org.overture.codegen.runtime.*;

import java.util.*;


@SuppressWarnings("all")
public class PublicationAuthorAffiliation {
    private Number pubid = 0L;
    private Number autid = 0L;
    private VDMSet coautids = SetUtil.set();
    private Number affid = 0L;
    private String oriaffnam = SeqUtil.toStr(SeqUtil.seq());

    public PublicationAuthorAffiliation(final Number puid, final Number auid,
        final VDMSet coaid, final Number afid, final String oan) {
        cg_init_PublicationAuthorAffiliation_1(puid, auid, Utils.copy(coaid),
            afid, oan);
    }
```

```
    public PublicationAuthorAffiliation() {
    }

    public void cg_init_PublicationAuthorAffiliation_1(final Number puid,
        final Number auid, final VDMSet coaid, final Number afid,
        final String oan) {
        pubid = puid;
        autid = auid;
        coautids = Utils.copy(coaid);
        affid = afid;
        oriaffnam = oan;

        return;
    }

    public void changeOriAffName(final String na) {
        oriaffnam = na;
    }

    public Number getPubid() {
        return pubid;
    }

    public Number getAutid() {
        return autid;
    }

    public VDMSet getCoautids() {
        return Utils.copy(coautids);
    }

    public Number getAffid() {
        return affid;
    }

    public String getOriAffName() {
        return oriaffnam;
    }

    public String toString() {
        return "PublicationAuthorAffiliation{" + "pubid := " +
        Utils.toString(pubid) + ", autid := " + Utils.toString(autid) +
        ", coautids := " + Utils.toString(coautids) + ", affid := " +
        Utils.toString(affid) + ", oriaffnam := " + Utils.toString(oriaffnam) +
        "}";
    }
}
```

6.11 PublicationKeyword

```
package ScientificPaperIndexSystem;

import org.overture.codegen.runtime.*;

import java.util.*;
```

```
@SuppressWarnings("all")
public class PublicationKeyword {
    private Number pid = 1L;
    private Number fosid = 1L;
    private String keyword = SeqUtil.toStr(SeqUtil.seq());

    public PublicationKeyword(final Number id, final Number fid, final String kw) {
        cg_init_PublicationKeyword_1(id, fid, kw);
    }

    public PublicationKeyword() {
    }

    public void cg_init_PublicationKeyword_1(final Number id, final Number fid,
        final String kw) {
        pid = id;
        fosid = fid;
        keyword = kw;

        return;
    }

    public void changeKeyword(final String kw) {
        keyword = kw;
    }

    public Number getPid() {
        return pid;
    }

    public Number getFosid() {
        return fosid;
    }

    public String getKeyword() {
        return keyword;
    }

    public String toString() {
        return "PublicationKeyword{" + "pid := " + Utils.toString(pid) +
        ", fosid := " + Utils.toString(fosid) + ", keyword := " +
        Utils.toString(keyword) + "}";
    }
}
```

6.12 PublicationReferences

```
package ScientificPaperIndexSystem;

import org.overture.codegen.runtime.*;

import java.util.*;


@SuppressWarnings("all")
```

```
public class PublicationReferences {
    private Number pid = 0L;
    private VDMSet prids = SetUtil.set();

    public PublicationReferences(final Number id, final VDMSet rids) {
        cg_init_PublicationReferences_1(id, Utils.copy(rids));
    }

    public PublicationReferences() {
    }

    public void cg_init_PublicationReferences_1(final Number id,
        final VDMSet rids) {
        pid = id;
        prids = Utils.copy(rids);

        return;
    }

    public void addReference(final Number pubid) {
        prids = SetUtil.union(Utils.copy(prids), SetUtil.set(pubid));
    }

    public void removeReference(final Number prid) {
        prids = SetUtil.diff(Utils.copy(prids), SetUtil.set(prid));
    }

    public Number getPid() {
        return pid;
    }

    public VDMSet getPrids() {
        return Utils.copy(prids);
    }

    public String toString() {
        return "PublicationReferences{" + "pid := " + Utils.toString(pid) +
        ", prids := " + Utils.toString(prids) + "}";
    }
}
```

6.13 PublicationURL

```
package ScientificPaperIndexSystem;

import org.overture.codegen.runtime.*;

import java.util.*;


@SuppressWarnings("all")
public class PublicationURL {
    private Number pid = 0L;
    private String url = SeqUtil.toStr(SeqUtil.seq());

    public PublicationURL(final Number id, final String u) {
```

```
        cg_init_PublicationURL_1(id, u);
    }

    public PublicationURL() {
    }

    public void cg_init_PublicationURL_1(final Number id, final String u) {
        pid = id;
        url = u;

        return;
    }

    public Number getPid() {
        return pid;
    }

    public String getUrl() {
        return url;
    }

    public String toString() {
        return "PublicationURL{" + "pid := " + Utils.toString(pid) +
        ", url := " + Utils.toString(url) + "}";
    }
}
```

6.14 startQuote

```
package ScientificPaperIndexSystem.quotes;

import org.overture.codegen.runtime.*;

import java.util.*;


@SuppressWarnings("all")
public class startQuote {
    private static int hc = 0;
    private static startQuote instance = null;

    public startQuote() {
        if (Utils.equals(hc, 0)) {
            hc = super.hashCode();
        }
    }

    public static startQuote getInstance() {
        if (Utils.equals(instance, null)) {
            instance = new startQuote();
        }

        return instance;
    }

    public int hashCode() {
```

```
        return hc;
    }

    public boolean equals(final Object obj) {
        return obj instanceof startQuote;
    }

    public String toString() {
        return "<start>";
    }
}
```

6.15 TestIndexer

```
package ScientificPaperIndexSystem;

import org.overture.codegen.runtime.*;

import java.util.*;


@SuppressWarnings("all")
public class TestIndexer extends MyTestCase {
    public TestIndexer() {
    }

    public void testAffiliation() {
        Affiliation af = new Affiliation("FEUP");
        super.assertEqual("FEUP", af.getName());
    }

    public void testAffiliationChangeName() {
        Affiliation af = new Affiliation("FEUP");
        super.assertEqual("FEUP", af.getName());
        af.changeName("MIT");
        super.assertEqual("MIT", af.getName());
    }

    public void testAuthor() {
        Author a = new Author("Andre",
                SetUtil.set("amputacoes", "desmembramentos"));
        super.assertEqual("Andre", a.getName());
        super.assertEqual(SetUtil.set("amputacoes", "desmembramentos"),
            a.getInterests());
    }

    public void testAuthorChangeName() {
        Author a = new Author("Andre",
                SetUtil.set("amputacoes", "desmembramentos"));
        super.assertEqual("Andre", a.getName());
        a.changeName("Banana");
        super.assertEqual("Banana", a.getName());
    }

    public void testAddInterestsAuthor() {
        Author a = new Author("Andre",
```

```
            SetUtil.set("amputacoes", "desmembramentos"));
    super.assertEqual(SetUtil.set("amputacoes", "desmembramentos"),
        a.getInterests());
    a.addInterests(SetUtil.set("mutila oes", "imola oes"));
    super.assertEqual(SetUtil.set("amputacoes", "desmembramentos",
            "mutila oes", "imola oes"), a.getInterests());
}

public void testRemoveInterestsAuthor() {
    Author a = new Author("Andre",
            SetUtil.set("amputacoes", "desmembramentos"));
    super.assertEqual(SetUtil.set("amputacoes", "desmembramentos"),
        a.getInterests());
    a.removeInterests(SetUtil.set("amputacoes"));
    super.assertEqual(SetUtil.set("desmembramentos"), a.getInterests());
    a.removeInterests(SetUtil.set("desmembramentos"));
    super.assertEqual(SetUtil.set(), a.getInterests());
}

public void testFieldOfStudy() {
    FieldOfStudy fos = new FieldOfStudy("Necromancia");
    super.assertEqual("Necromancia", fos.getName());
}

public void testPublication() {
    Publication pub = new Publication("A␣Specifier's␣Introduction␣to␣Formal␣Methods'
            1990L, 901L, "11.111.11/ISBN", 100L);
    super.assertEqual("A␣Specifier's␣Introduction␣to␣Formal␣Methods",
        pub.getOriginalTitle());
    super.assertEqual(1990L, pub.getPublishYear());
    super.assertEqual(901L, pub.getPublishDate());
    super.assertEqual("11.111.11/ISBN", pub.getDoi());
    super.assertEqual(100L, pub.getRank());
}

public void testRaiseRankBy() {
    Publication pub = new Publication("A␣Specifier's␣Introduction␣to␣Formal␣Methods'
            1990L, 901L, "11.111.11/ISBN", 100L);
    super.assertEqual(100L, pub.getRank());
    pub.raiseRankBy(50L);
    super.assertEqual(50L, pub.getRank());
}

public void testDecreaseRankBy() {
    Publication pub = new Publication("A␣Specifier's␣Introduction␣to␣Formal␣Methods'
            1990L, 901L, "11.111.11/ISBN", 100L);
    super.assertEqual(100L, pub.getRank());
    pub.decreaseRankBy(50L);
    super.assertEqual(150L, pub.getRank());
}

public void testPublicationAuthorAffiliation() {
    PublicationAuthorAffiliation paa = new PublicationAuthorAffiliation(1L,
            2L, SetUtil.set(1L, 3L, 4L), 3L, "FEUP");
    super.assertEqual(1L, paa.getPubid());
    super.assertEqual(2L, paa.getAutid());
```

```
        super.assertEqual(3L, paa.getAffid());
        super.assertEqual("FEUP", paa.getOriAffName());
    }

    public void testPublicationAuthorAffiliationChangeName() {
        PublicationAuthorAffiliation paa = new PublicationAuthorAffiliation(1L,
                2L, SetUtil.set(1L, 3L, 4L), 3L, "FEUP");
        super.assertEqual("FEUP", paa.getOriAffName());
        paa.changeOriAffName("MIT");
        super.assertEqual("MIT", paa.getOriAffName());
    }

    public void testPublicationKeyword() {
        PublicationKeyword pk = new PublicationKeyword(1L, 2L, "MFES");
        super.assertEqual(1L, pk.getPid());
        super.assertEqual(2L, pk.getFosid());
        super.assertEqual("MFES", pk.getKeyword());
    }

    public void testPublicationKeywordChangeKeyword() {
        PublicationKeyword pk = new PublicationKeyword(1L, 2L, "MFES");
        pk.changeKeyword("SEFM");
        super.assertEqual("SEFM", pk.getKeyword());
    }

    public void testPublicationReferences() {
        PublicationReferences pr = new PublicationReferences(1L,
                SetUtil.set(2L, 3L));
        super.assertEqual(1L, pr.getPid());
        super.assertEqual(SetUtil.set(2L, 3L), pr.getPrids());
    }

    public void testPublicationReferencesAddReference() {
        PublicationReferences pr = new PublicationReferences(1L,
                SetUtil.set(2L, 3L));
        super.assertEqual(1L, pr.getPid());
        super.assertEqual(SetUtil.set(2L, 3L), pr.getPrids());
        pr.addReference(4L);
        super.assertEqual(SetUtil.set(2L, 3L, 4L), pr.getPrids());
    }

    public void testPublicationReferencesRemoveReference() {
        PublicationReferences pr = new PublicationReferences(1L,
                SetUtil.set(2L, 3L, 4L));
        super.assertEqual(1L, pr.getPid());
        super.assertEqual(SetUtil.set(2L, 3L, 4L), pr.getPrids());
        pr.removeReference(4L);
        super.assertEqual(SetUtil.set(2L, 3L), pr.getPrids());
    }

    public void testPublicationURL() {
        PublicationURL purl = new PublicationURL(1L, "mypublicatonurl.com");
        super.assertEqual(1L, purl.getPid());
        super.assertEqual("mypublicatonurl.com", purl.getUrl());
    }
```

```
public void testIndexerEmpty() {
    Indexer idx = new Indexer();
    super.assertEqual(MapUtil.map(), idx.getPublications());
    super.assertEqual(MapUtil.map(), idx.getAuthors());
    super.assertEqual(MapUtil.map(), idx.getAffiliations());
    super.assertEqual(MapUtil.map(), idx.getFieldsofstudy());
    super.assertEqual(MapUtil.map(), idx.getPublicationkeywords());
    super.assertEqual(MapUtil.map(), idx.getPublicationauthoraffiliations());
    super.assertEqual(MapUtil.map(), idx.getPublicationsreferences());
    super.assertEqual(MapUtil.map(), idx.getPublicationurls());
}

public void testIndexerAuthors() {
    Indexer idx = new Indexer();
    idx.insertAuthor("autor1", SetUtil.set("cozinhar", "escrever"));
    super.assertEqual("autor1", idx.getAuthorById(1L).getName());
    super.assertEqual(SetUtil.set("cozinhar", "escrever"),
        idx.getAuthorById(1L).getInterests());
    idx.insertAuthor("autor2", SetUtil.set("estudar", "voar"));
    super.assertEqual("autor2", idx.getAuthorById(2L).getName());
    super.assertEqual(SetUtil.set("estudar", "voar"),
        idx.getAuthorById(2L).getInterests());
    idx.addInterestsToAuthor(2L, SetUtil.set("futebol", "praia"));
    idx.addInterestsToAuthor(2L, SetUtil.set("golf"));
    super.assertEqual(SetUtil.set("estudar", "voar", "futebol", "praia",
            "golf"), idx.getAuthorById(2L).getInterests());
    idx.removeInterestsFromAuthor(2L, SetUtil.set("voar"));
    super.assertEqual(SetUtil.set("estudar", "futebol", "praia", "golf"),
        idx.getAuthorById(2L).getInterests());
    idx.removeInterestsFromAuthor(2L,
        SetUtil.set("futebol", "praia", "golf"));
    super.assertEqual(SetUtil.set("estudar"),
        idx.getAuthorById(2L).getInterests());
}

public void testCoautherPath() {
    Indexer idx = new Indexer();
    idx.insertAuthor("autor1", SetUtil.set("cozinhar", "escrever"));
    idx.insertAuthor("autor2", SetUtil.set("estudar", "voar"));
    idx.insertAuthor("autor3", SetUtil.set("correr", "cantar"));
    idx.insertAuthor("autor4", SetUtil.set("estudar", "voar"));
    idx.insertAuthor("autor5", SetUtil.set("brincar", "animar"));
    idx.insertAuthor("autor6", SetUtil.set("calcular", "cantar"));
    idx.insertAffiliation("FEUP");
    idx.insertAffiliation("FAC");
    super.assertEqual(2L, MapUtil.dom(idx.getAffiliations()).size());
    idx.insertPublication("A␣Specifier's␣Introduction␣to␣Formal␣Methods",
        1990L, 901L, "11.111.11/ISBN", 100L, 1L, SetUtil.set(2L, 3L), 1L,
        SetUtil.set());
    idx.insertPublication("A␣Specifier's␣Introduction␣to␣Informal␣Methods",
        1991L, 902L, "11.111.12/ISBN", 10L, 1L, SetUtil.set(3L, 4L), 2L,
        SetUtil.set(1L));
    idx.insertPublication("A␣Specifier's␣Introduction␣to␣Informal␣Math",
        1993L, 903L, "11.111.13/ISBN", 13L, 1L, SetUtil.set(5L, 6L, 2L),
        2L, SetUtil.set(1L, 2L));
    super.assertEqual(3L, MapUtil.dom(idx.getPublications()).size());
```

```
        super.assertEqual(3L, idx.getPublicationsByAuthor(1L).size());
        super.assertEqual(7L, idx.getCoauthorPath(1L).size());
    }

    public void testCountSelfCitations() {
        Indexer idx = new Indexer();
        idx.insertAuthor("autor1", SetUtil.set("cozinhar", "escrever"));
        idx.insertAffiliation("FEUP");
        idx.insertAffiliation("FAC");
        idx.insertPublication("A Specifier's Introduction to Formal Methods",
            1990L, 901L, "11.111.11/ISBN", 100L, 1L, SetUtil.set(), 1L,
            SetUtil.set());
        idx.insertPublication("A Specifier's Introduction to Informal Methods",
            1991L, 902L, "11.111.12/ISBN", 10L, 1L, SetUtil.set(), 2L,
            SetUtil.set(1L));
        idx.insertPublication("A Specifier's Introduction to Informal Math",
            1993L, 903L, "11.111.13/ISBN", 13L, 1L, SetUtil.set(), 2L,
            SetUtil.set(1L, 2L));
        super.assertEqual(2L,
            idx.getReferencesFromAuthorPublications(idx.getPublicationsByAuthor(
                    1L)).size());
    }

    public void testCountOthersCitations() {
        Indexer idx = new Indexer();
        idx.insertAuthor("autor1", SetUtil.set("cozinhar", "escrever"));
        idx.insertAuthor("autor2", SetUtil.set("estudar", "voar"));
        idx.insertAuthor("autor3", SetUtil.set("correr", "cantar"));
        idx.insertAffiliation("FEUP");
        idx.insertAffiliation("FAC");
        idx.insertPublication("A Specifier's Introduction to Formal Methods",
            1990L, 901L, "11.111.11/ISBN", 100L, 1L, SetUtil.set(), 1L,
            SetUtil.set());
        idx.insertPublication("A Specifier's Introduction to Informal Methods",
            1991L, 902L, "11.111.12/ISBN", 10L, 2L, SetUtil.set(), 2L,
            SetUtil.set(1L));
        idx.insertPublication("A Specifier's Introduction to Informal Math",
            1993L, 903L, "11.111.13/ISBN", 13L, 3L, SetUtil.set(), 2L,
            SetUtil.set(1L, 2L));
        super.assertEqual(1L, idx.getTimesCited(2L));
        super.assertEqual(2L, idx.getTimesCited(1L));
    }

    public void testNumberErdosAuthor() {
        Indexer idx = new Indexer();
        idx.insertAuthor("autor1", SetUtil.set("cozinhar", "escrever"));
        idx.insertAuthor("autor2", SetUtil.set("estudar", "voar"));
        idx.insertAuthor("autor3", SetUtil.set("correr", "cantar"));
        ((Author) Utils.get(idx.getAuthors(), 1L)).setErdos(1L);
        super.assertEqual(1L,
            ((Author) Utils.get(idx.getAuthors(), 1L)).getErdos());
        ((Author) Utils.get(idx.getAuthors(), 2L)).setErdos(100L);
        ((Author) Utils.get(idx.getAuthors(), 3L)).setErdos(3000L);
        idx.insertAffiliation("FEUP");
        idx.insertPublication("A Specifier's Introduction to Formal Methods",
            1990L, 901L, "11.111.11/ISBN", 100L, 1L, SetUtil.set(2L, 3L), 1L,
```

```
            SetUtil.set());
        super.assertEqual(2L,
            ((Author) Utils.get(idx.getAuthors(), 2L)).getErdos());
        super.assertEqual(2L,
            ((Author) Utils.get(idx.getAuthors(), 3L)).getErdos());
    }

    public void testAll() {
        testAffiliation();
        testAffiliationChangeName();
        testAuthor();
        testAuthorChangeName();
        testAddInterestsAuthor();
        testRemoveInterestsAuthor();
        testFieldOfStudy();
        testPublication();
        testRaiseRankBy();
        testDecreaseRankBy();
        testPublicationAuthorAffiliation();
        testPublicationAuthorAffiliationChangeName();
        testPublicationKeyword();
        testPublicationKeyword();
        testPublicationReferences();
        testPublicationReferencesAddReference();
        testPublicationReferencesRemoveReference();
        testPublicationURL();
        testIndexerEmpty();
        testIndexerAuthors();
        testCoautherPath();
        testCountSelfCitations();
        testCountOthersCitations();
        testNumberErdosAuthor();
    }

    public String toString() {
        return "TestIndexer{}";
    }
}
```

7 Conclusão

Conseguiu-se cumprir com os objetivos e requisitos esperados e propostos, respetivamente, ficando por refinar do algoritmo utilizado para computar o número de Erdős pois quando uma publicação é introduzida e possa itreferir com co-autores, o número destes co-autores não é alterado se houver um co-autor número de Erdős inferior a todos participantes na publicação. Supostamente, se a cituação anterior se sucedesse, os números de todos os intervenientes (autores e co-autores) diretos e indiretos, o número de Erdős deveria mudar.
Assim sendo, recorreu-se às seguintes estruturas de dados:

- Maps;
- Sets;
- Seqs;

de tipos de variáveis nativas e de classes implmentadas por nós para implmentar as inserções; computação do Erdős e caminho do co-autor; e contagem de citações.
Quanto à participação dos elementos do grupo no trabalho, ditribui-se de igual modo 50% quer para o André, quer para o João.

8 Bibliofrafia

Vienna Development Method, https://en.wikipedia.org/wiki/Vienna_Development_Method. accessed on 14-12-2015.

Overture, Overture Tool website, http://overturetool.org/. accessed on 14-12-2015.

Peter G. Larsen and Kenneth Lausdahl and Nick Battle and John Fitzgerald and Sune Wolff. VDM-10 Language Manual. Overture. February 2011.

Microsoft Academic Graph - 2015/11/06, https://academicgraph.blob.core.windows.net/graph-2015-11-06/index.html. accessed on 14-12-2015.