Anexos

1.1 Anexo A

Código en Jape del recurso identificador de frases nominales:

```
Phase:TokenUse
Input: Token
Options: control = appelt

Rule: Candidatos
(
      (
      {Token.category == JJ}
      |
      {Token.category == JJS}
      )*
      (
      {Token.category == NN}
      |
      {Token.category == NNS}
      |
      {Token.category == NP}
      |
      {Token.category == NPS}
      |
      {Token.category == NNP}
      |
      {Token.category == NNPS}
      )+

)
:CandidatosPC
-->
 :CandidatosPC.Candidato= {rule = "Candidatos"}
```

1.2 Anexo B

Código en Jape del recurso identificador de abreviaciones:

```
Phase:TokenUse
Input: Token
Options: control = brill

Rule: Abrev
(
       (
              (
              {Token.kind == word , Token.length == 1 , Token.orth ==
upperInitial}
              )
              (
              {Token.string == "."}
              )
              (
                     (
                     {Token.kind == word, Token.length == 1 , Token.orth ==
upperInitial}
                     )
                     (
                     {Token.string == "."}
                     )
              )+
       )
       |
       (
              {Token.kind == word , Token.length > 1 , Token.orth ==
allCaps, Token.string ==~
"[abcdefghijklmnñopqrstuvwxyzABCDEFGHIJKLMNÑOPQRSTUVWXYZ-]+"}
       )
       |
       (
              (
                     (
                     {Token.kind == word , Token.orth == allCaps,
Token.string ==~
"[abcdefghijklmnñopqrstuvwxyzABCDEFGHIJKLMNÑOPQRSTUVWXYZ]+"}
                     )
                     (
                     {Token.kind == number, Token.string ==~ "[1234567890]+"}
                     )
                     (
                     {Token.kind == word , Token.orth == allCaps,
Token.string ==~
"[abcdefghijklmnñopqrstuvwxyzABCDEFGHIJKLMNÑOPQRSTUVWXYZ]+"}
                     )+
              )+
       )

)
:Abreviaciones
-->
 :Abreviaciones.Abreviacion= {rule = "Abrev"}
```

1.3 Anexo C

Código en Java del proceso usado para encontrar las abreviaciones mapeadas en el diccionario de sinónimos:

```
public void procesarAnotaciones(String nombreAnotacion)throws Exception{

        Map anotaciones=documento.getNamedAnnotationSets();
        DocumentContent contenido= documento.getContent();
        AnnotationSet setAnotacion =
(AnnotationSet)(anotaciones.get(nombreAnotacion));
        String nombreTipo = "Candidato";
        AnnotationSet candidato= setAnotacion.get(nombreTipo);
        Object[] arregloAnotacion=candidato.inDocumentOrder().toArray();
        int nabrev=0;
        Annotation abreviacion=
this.abreviaturas.size()>nabrev?this.abreviaturas.get(nabrev):null;
        for (int numAn=0; numAn<arregloAnotacion.length ; numAn++){
            Object anotacion= arregloAnotacion[numAn];
            Annotation candidatoAnot=((Annotation)anotacion);
            Long start=candidatoAnot.getStartNode().getOffset() ;
            Long end=candidatoAnot.getEndNode().getOffset();
            String textocandidato=contenido.getContent(start,
end).toString();
            textocandidato=procesaPalabra(textocandidato);
            if (textocandidato==null) continue;
            //Verifico el diccionario de sinónimos
            while(true && abreviacion!=null){
                if (abreviacion.withinSpanOf(candidatoAnot) ){
textocandidato=textocandidato.replace(this.textoAbreviaturas.get(nabrev) ,
this.cambioAbreviaturas.get(nabrev));
                    break;
                }else{
                    if (Long.compare(start,
abreviacion.getStartNode().getOffset())>0){
                        nabrev++;
                        abreviacion=
this.abreviaturas.size()>nabrev?this.abreviaturas.get(nabrev):null;
                        continue;
                    }
                    if (Long.compare(start,
abreviacion.getStartNode().getOffset())<0){
                        break;
                    }
                }
            }
            this.agregarPalabraContenido(textocandidato);
        }
    }
```

1.4 Anexo D

Código en Java del proceso usado para preparar los candidatos para el proceso de enriquecimiento:

```
public void agregarPalabraContenido(String palabra){

        if (palabra==null || palabra.matches(".*[/'-
_{}#$%\n]+")
                || palabra.contains("(") ||
palabra.contains(")") || palabra.contains("{")||
palabra.contains("}")
                || palabra.length()==1)return;

        palabra=palabra.replaceAll("-\n", "");
        palabra=palabra.replaceAll("\n", " ");
        while (palabra.endsWith(" "))
palabra=palabra.substring(0, palabra.length()-1);
        while (palabra.contains("  "))
palabra=palabra.replaceAll("  ", " ");

        if (!palabra.matches("[1234567890a-zA-Z '´-]*"))
return;


this.setCandidatosNoEnriquecidos(this.getCandidatosNoEnriqueci
dos().concat(palabra + " .\n"));

    }
```

1.5 Anexo E

Código en Jape del recurso encargado del proceso de enriquecimiento de candidatos:

```
Phase:TokenUse
Input: Token
Options: control = all

Rule: Enriquecimiento
(
      (
      {Token.category == JJ}
      |
      {Token.category == JJS}
      )*
      (
      {Token.category == NN}
      |
      {Token.category == NNS}
      |
      {Token.category == NP}
      |
      {Token.category == NPS}
      |
      {Token.category == NNP}
      |
      {Token.category == NNPS}
      )+

)
:NuevosCandidatosPC
-->
 :NuevosCandidatosPC.NuevoCandidato= {rule =
"Enriquecimiento"}
```

1.6 Anexo F

Código en Java del proceso usado para realizar el proceso de enriquecimiento:

```
public void procesarEnriquecimiento(Document documentoTemp)throws Exception{
        Map anotaciones=documentoTemp.getNamedAnnotationSets();
        DocumentContent contenido= documentoTemp.getContent();
        AnnotationSet setAnotacion =
(AnnotationSet)(anotaciones.get(Constantes.anotacionCandidatosEnriquecidos))
;
        String nombreTipo = "NuevoCandidato";
        AnnotationSet candidato= setAnotacion.get(nombreTipo);
        Object[] arregloAnotacion=candidato.inDocumentOrder().toArray();
        for (int numAn=0; numAn<arregloAnotacion.length ; numAn++){
            Object anotacion= arregloAnotacion[numAn];
            Annotation candidatoAnot=((Annotation)anotacion);
            Long start=candidatoAnot.getStartNode().getOffset() ;
            Long end=candidatoAnot.getEndNode().getOffset();
            String textocandidato=contenido.getContent(start,
end).toString();
            this.agregarPalabra(textocandidato);
        }
    }
    public String procesaPalabra(String palabra){
        if (palabra==null || palabra.matches(".*[/'-_{}#$%\n]+")
                || palabra.contains("(") || palabra.contains(")") ||
palabra.contains("{")|| palabra.contains("}")
                || palabra.length()==1)return null;
        palabra=palabra.replaceAll("-\n", "");
        palabra=palabra.replaceAll("\n", " ");
        while (palabra.endsWith(" ")) palabra=palabra.substring(0,
palabra.length()-1);
        while (palabra.contains("  ")) palabra=palabra.replaceAll("  ", "
");
        if (!palabra.matches("[1234567890a-zA-Z '´-]*")) return null;
        String[] palabras= palabra.split(" ");
        //Verifico que no sea un stopword
        if (palabras!=null && palabras.length >0){
            for (String p: palabras)
            if (CFinderData.getStopwordList().contains(p.toLowerCase()))
                return null;
        }else
            if
(CFinderData.getStopwordList().contains(palabra.toLowerCase()))
                return null;
        return palabra;
    }
```

1.7 Anexo G

Código en Java del proceso usado para buscar candidatos dependientes de otros:

```
public void guardarDatosCandidatos(){

        for (Object cand: this.candidatos.keySet().toArray()){
            CandidatoData candidatoData= new
CandidatoData((String)cand,(int)this.candidatos.get(cand) );
            this.datosCandidatos.add(candidatoData);
            this.candidatosObjetos.put(candidatoData.getPalabra(),
candidatoData);
        }
        //Se calcula para cada candidato sus dependientes
        Object[]candidatoArray= this.candidatos.keySet().toArray();
        for (int i=0; i<candidatoArray.length;i++ ){
            ArrayList<CandidatoData> dependencia= new ArrayList();
            String candidato= (String)(candidatoArray[i]);
            String[] candidatoSplit=candidato.split(" ");
            CandidatoData candidatoData=
((CandidatoData)(this.candidatosObjetos.get(candidato)));
            if (candidatoData.getNumPalabras()>1 &&
candidatoData.getDependientes().isEmpty() )
                for (int j=0; j<candidatoArray.length;j++ ){
                    String posibleDependiente=
(String)(candidatoArray[j]);
                    if (!candidato.equals(posibleDependiente) &&
candidato.contains(posibleDependiente)){
                        if
(!dependencia.contains((CandidatoData)(this.candidatosObjetos.get(po
sibleDependiente)))){
                            String[]posibleDependienteSplit=
posibleDependiente.split(" ");
                            if
(java.util.Arrays.asList(candidatoSplit).containsAll(java.util.Array
s.asList(posibleDependienteSplit)))

dependencia.add((CandidatoData)(this.candidatosObjetos.get(posibleDe
pendiente)));
                        }
                    }
                }
            candidatoData.setDependientes(dependencia);
        }
    }
```

1.8 Anexo H

Código en Java del proceso usado para calcular los pesos de los candidatos por documento:

```
public void calculaPesoCandidato(HashMap palabrasDominio, int
maxPalabrasDominio){
        //Se ordenana los candidatos por número de palabras
que lo conforman
        Collections.sort(this.datosCandidatos);
        //Se calcula el peso para los candidatos conformados
por una palabra
        int i=0;
        for (i=0;
i<this.datosCandidatos.size()&&this.datosCandidatos.get(i).get
NumPalabras()<=1 ;i++){
            double tf=
this.datosCandidatos.get(i).calculartf(this.maxRepienciaCandid
ato);
            double wd=
this.calcularDomainSpecifc(palabrasDominio,
maxPalabrasDominio, this.datosCandidatos.get(i).getPalabra());
            this.datosCandidatos.get(i).setPeso(tf*wd);
        }
        // Se calcula el peso de aquellos que son multi-
palabra
        for (;i<this.datosCandidatos.size();i++){
            CandidatoData candidato=
this.datosCandidatos.get(i);
            if (candidato.getNumPalabras()>1 &&
candidato.getNumRepitencia()<=2) continue;
            ArrayList<CandidatoData> maxsubsets=
candidato.obtenerMaxSubsets();
            double peso=0;
            for (CandidatoData c: maxsubsets)
                peso= peso + c.getPeso();
            candidato.setPeso(peso);
        }
        Collections.sort(this.datosCandidatos);
    }
```

1.9 Anexo I

Código en Java del proceso usado para calcular los pesos de los candidatos para todo el corpus:

```
public void sumarTodosDocumentos(){

        for ( Object documento :
this.documentos.values().toArray()){

            ArrayList<CandidatoData> candidatosDoc
=((DocumentData)documento).getDatosCandidatos();
            for(CandidatoData c:candidatosDoc){
                if
(this.conceptosClaveObjetos.containsKey(c.getPalabra())){
                    CandidatoData
concepto=(CandidatoData)(this.conceptosClaveObjetos.get(c.getP
alabra()));

concepto.setPeso(c.getPeso()+concepto.getPeso());
                }else{

                    CandidatoData nuevoConcepto= new
CandidatoData(c.getPalabra());
                    nuevoConcepto.setPeso(c.getPeso());

this.conceptosClaveObjetos.put(c.getPalabra(), nuevoConcepto);
                    this.conceptosClave.add(nuevoConcepto);
                }
            }
        }

        Collections.sort(this.conceptosClave);
    }
```

1.10 Anexo J

Código en Java del proceso usado para la obtención de los conceptos clave y número de documentos en donde aparecen:

```
public HashMap<String, NodoData> obtenerNodosCorpus()throws
Exception{

        Connection con= ConexionBD.conn;
        String sql= "";
        PreparedStatement pstmt= null;
        ResultSet rset=null ;
        HashMap <String, NodoData> conceptos= new HashMap();
        try{
            sql=" SELECT A.TEXTO,COUNT(B.IDDOCUMENTO) FROM
TESIS.CONCEPTOS_CLAVE A, TESIS.CANDIDATO B "
             + " WHERE A.IDCORPUS=? AND A.TEXTO=B.TEXTO AND
A.IDCORPUS=B.IDCORPUS "
             + " GROUP BY A.TEXTO ORDER BY 2 DESC ";

            pstmt= con.prepareStatement(sql);
            pstmt.setInt(1, this.idCorpus);
            rset=pstmt.executeQuery();
            while (rset.next()){
                NodoData nuevoConcepto= new NodoData();
                nuevoConcepto.setTexto(rset.getString(1));
                nuevoConcepto.setAparicionDocumentos(rset.getInt(2))
;

                conceptos.put(rset.getString(1), nuevoConcepto);
            }
        }catch(Exception e){
            System.out.println("GRAVE ERROR: " +e.getMessage());
            throw e;
        }finally{
          if (pstmt!=null) pstmt.close();
          if (rset!=null)  rset.close();
        }
        this.conceptosNodo=conceptos;
        return conceptos;
    }
```

1.11 Anexo K

Código en Java del proceso usado para la obtención de pares de conceptos clave y número de documentos en donde aparecen:

```
public  HashMap<String,ArrayList<RelacionJerarquiaData>>
obtenerRelacionPosiblesPadres()throws Exception{
        Connection con= ConexionBD.conn;
        String sql= "";
        PreparedStatement pstmt= null;
        ResultSet rset=null ;
        ArrayList<RelacionJerarquiaData> posiblesPadres= new ArrayList();
        HashMap<String,ArrayList<RelacionJerarquiaData>>
relacionesPosiblesPadres= new HashMap();
        try{
            sql=" SELECT B.TEXTO, D.TEXTO, COUNT(DISTINCT B.IDDOCUMENTO) "
              + " FROM TESIS.CONCEPTOS_CLAVE A, TESIS.CANDIDATO B,
TESIS.CONCEPTOS_CLAVE C, TESIS.CANDIDATO D "
              + " WHERE A.IDCORPUS = ? "
              + " AND A.TEXTO=B.TEXTO AND A.IDCORPUS= B.IDCORPUS AND
C.TEXTO=D.TEXTO AND C.IDCORPUS= D.IDCORPUS "
              + " AND A.IDCORPUS=C.IDCORPUS AND B.IDDOCUMENTO=D.IDDOCUMENTO
AND NOT B.TEXTO = D.TEXTO "
              + " GROUP BY B.TEXTO, D.TEXTO ORDER BY 1,2 ";
            pstmt= con.prepareStatement(sql);
            pstmt.setInt(1, this.idCorpus);
            rset=pstmt.executeQuery();
            String actual="";
            String padre="";
            String hijo="";
            while (rset.next()){
                padre=rset.getString(2);
                hijo=rset.getString(1);
                if (!actual.equals(padre) && !actual.trim().equals("")){
                    relacionesPosiblesPadres.put(hijo, posiblesPadres);
                    posiblesPadres= new ArrayList();
                    actual=padre;
                }
                RelacionJerarquiaData relacionData = new
RelacionJerarquiaData();
                relacionData.setNodoPadre(this.conceptosNodo.get(padre));
                relacionData.setNodoHijo(this.conceptosNodo.get(hijo));
                relacionData.setAparicionDocumentos(rset.getInt(3));
                if (relacionData.verificaRelacion())
                    posiblesPadres.add(relacionData);
            }
            if (!posiblesPadres.isEmpty())
relacionesPosiblesPadres.put(hijo, posiblesPadres);
        }catch(Exception e){
            System.out.println("GRAVE ERROR: " +e.getMessage());
            throw e;
        }finally{
          if (pstmt!=null) pstmt.close();
          if (rset!=null)  rset.close();
        }
        return relacionesPosiblesPadres;
    }
```

1.12 Anexo L

Código en Java del proceso que verifica si los pares de conceptos clave conforman una relación de posible parentesco:

```
public boolean verificaRelacion(){

        boolean esRelacionValida=false;

        calculaProbabilidad();

        //x padre de y
        esRelacionValida = (pXY>=Constantes.benchmark) &&
(pYX<Constantes.benchmark);
        return esRelacionValida;
    }

    public void calculaProbabilidad(){

        this.setpXY((double)((double)this.aparicionDocumentos/(doubl
e)this.nodoHijo.getAparicionDocumentos()));
        this.setpYX((double)((double)this.aparicionDocumentos/(doubl
e)this.nodoPadre.getAparicionDocumentos()));

    }
```

1.13 Anexo M

Código en Java del proceso que identifica el mejor padre de un nodo:

```
public void identificaPadreNodo(NodoData nodo)throws Exception{
        //Si ya tiene padre
        if (nodo.getNodoPadre()!=null) return;
        //Si es el nodo raiz
        if (nodo.getAparicionDocumentos()==-100) return;
        //Si no tiene padre posible
        if (!this.posiblesPadres.containsKey(nodo.getTexto())){
nodo.setNodoPadre(nodoRaiz);nodoRaiz.agregaNodoHijo(nodo);return;}
        ArrayList<RelacionJerarquiaData> relacionesPadre=
this.posiblesPadres.get(nodo.getTexto());
        if (relacionesPadre==null || relacionesPadre.isEmpty()){
nodo.setNodoPadre(nodoRaiz);nodoRaiz.agregaNodoHijo(nodo);return ;}
        //Caso normal
        double maxScore=0;
        RelacionJerarquiaData relacionPadre=null;
        for (RelacionJerarquiaData r:relacionesPadre){
            double score=this.calculaScore(r,1);
            if (score>maxScore){
                relacionPadre=r;
                maxScore=score;
            }
        }
        this.getRelacionesPadres().put(nodo.getTexto(),
relacionPadre);
        nodo.setNodoPadre(relacionPadre.getNodoPadre());
        relacionPadre.getNodoPadre().agregaNodoHijo(nodo);
    }
```

1.14 Anexo N

Código en Java del proceso que obtiene el puntaje de una posible relación de parentesco:

```
public double calculaScore(RelacionJerarquiaData relacion, int
distancia )throws  Exception{
        double score=0;

        if (relacion.getNodoPadre()==null) return 0;
        if (relacion.getNodoPadre().getAparicionDocumentos()==-100)
return 0;
        if (relacion.getNodoPadre().getNodoPadre()==null)
identificaPadreNodo(relacion.getNodoPadre()) ;

        RelacionJerarquiaData relaciontemp= new
RelacionJerarquiaData();
        relaciontemp.setNodoHijo(relacion.getNodoHijo());
        relaciontemp.setNodoPadre(relacion.getNodoPadre().getNodoPad
re());
        relaciontemp.obtenerAparicionDocumento(this.corpus.getIdCorp
us());
        relaciontemp.calculaProbabilidad();
        score= calculaScore(relaciontemp, distancia+1) +
relacion.getpXY();

        return score;
    }
```