Capítulo IV. El Sistema JProlog

El tercer paso de la estrategia utilizada se describe en este capítulo. Como se ha mencionado, NATLIN ya estaba implementado en XSB y se había considerado utilizar la interfaz de Interprolog, así que era importante desarrollar una metodología a partir de una implementación, si era necesario, para establecer una interfaz de calidad entre XSB y Java para que desde éste se pudieran recuperar resultados de procesamientos en el sistema XSB en forma controlada y sencilla. Por eso a continuación se describirá la metodología desarrollada para tal efecto.

4.1 Especificaciones del Sistema JProlog

4.1.1 Especificaciones generales

- Utilizar XSB desde una aplicación en Java mediante el paquete InterProlog.
- Facilitar desde Java la recuperación de términos y listas como resultados de un procesamiento en XSB.
- Enviar comandos desde Java a XSB.

4.1.2 Especificaciones de consultas

Mediante la interfaz de InterProlog, se redirecciona la salida de un proceso en XSB a una interfaz gráfica en Java en lugar de una consola desde la cual se ejecute XSB, sin embargo se necesitan obtener los resultados asociados a las variables que se colocan explícitamente en la consulta para después poder utilizarlos y procesarlos desde Java.

Con el Sistema JProlog debe poderse obtener el valor asociado a una variable de una consulta. Por ejemplo:

Sea la base de datos declarativa:

```
p(a, [c,d]).
p(b, [d,e]).
```

Si desde Java se enviara la consulta p(X, Y) al proceso XSB, debería poderse recuperar “a” y “[c,d]”, y no sólo eso, sino que el usuario de Java también debería conocer que la variable X se instanció por la constante “a” y la variable Y por la lista [c,d].

4.2 Análisis y diseño del sistema

Para resolver las especificaciones mencionadas en la sección anterior y debido a que el proceso de XSB se ejecutará desde Java se desarrolló el ciclo de vida mostrado en la figura 4.1 que tiene un objeto Prolog en Java.

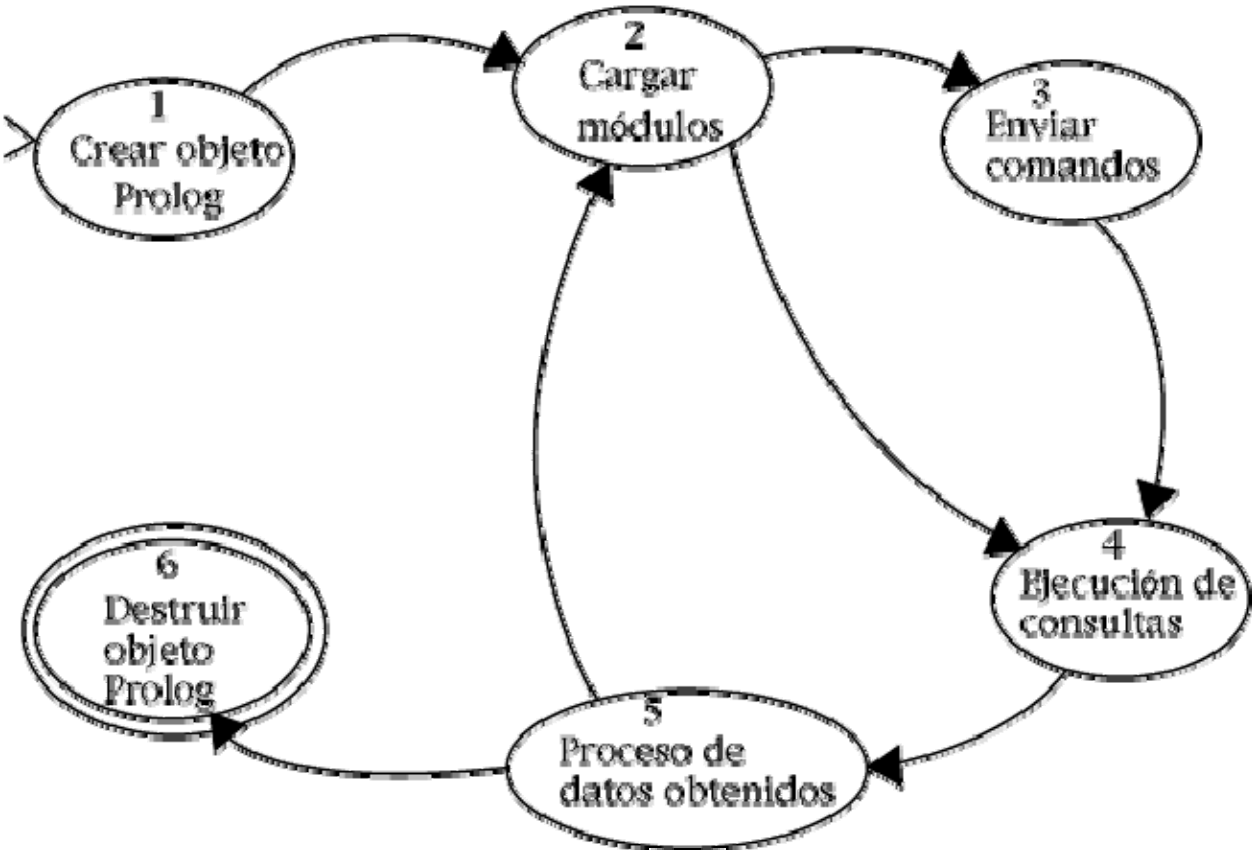

Figura 4.1 Ciclo de vida de un objeto Prolog en Java

Como se puede observar, desde Java el primer paso se refiere a la creación de un objeto Prolog, después en el segundo estado se pueden cargar módulos a la memoria de trabajo, una vez realizado esto se tienen dos alternativas, efectuar consultas a la máquina de inferencias y base declarativa (cuarto estado) o bien enviar comandos para emular aprendizaje con nuevas reglas en XSB (tercer estado).

Luego del tercer estado, se debería proceder a efectuar consultas según la nueva información existente en XSB.

Después de haber efectuado consultas, se asocian los resultados a las variables especificadas en la consulta, los cuales pueden procesarse desde Java. Una vez concluido el quinto estado se pueden cargar más módulos, o bien concluir el proceso de XSB y por tanto destruir el objeto Prolog.

Ahora bien, para cumplir con las especificaciones mencionadas en la sección anterior, se utilizó el paquete InterProlog y se desarrollaron las siguientes clases que facilitan la comunicación entre XSB y Java para un programador de Java: Prolog, Term, Terms, UndefinedPredicateException, además de StringVector, ésta última para fines de investigación en relación a conocer posteriormente a este trabajo el comportamiento de escritura de archivos en XSB y lectura de los mismos desde Java.

La clase principal es Prolog, ya que al crear un objeto Prolog, se inicia un proceso XSB y entonces ya se pueden cargar módulos, enviar comandos o hacer consultas para recuperar resultados de procesamientos efectuados por implementaciones basadas en programación lógica desde Java.

Term es una clase que modela un término en XSB, un objeto será una representación en Java de un término en XSB, pudiendo ser constante, variable o lista.

Terms es una clase, cuyo objeto instanciado simplemente agrupa un conjunto de objetos Term. En otras, palabras es un conjunto de términos.

UndefinedPredicateException es una clase que permite la generación de una excepción cuando se pregunta por el valor de verdad de un predicado que no está definido en la memoria de trabajo actual de XSB.

En los diagramas representados en la figura 4.3, basados en la metodología UML [32], se muestra la relación existente entre las clases descritas y las clases del paquete InterProlog.

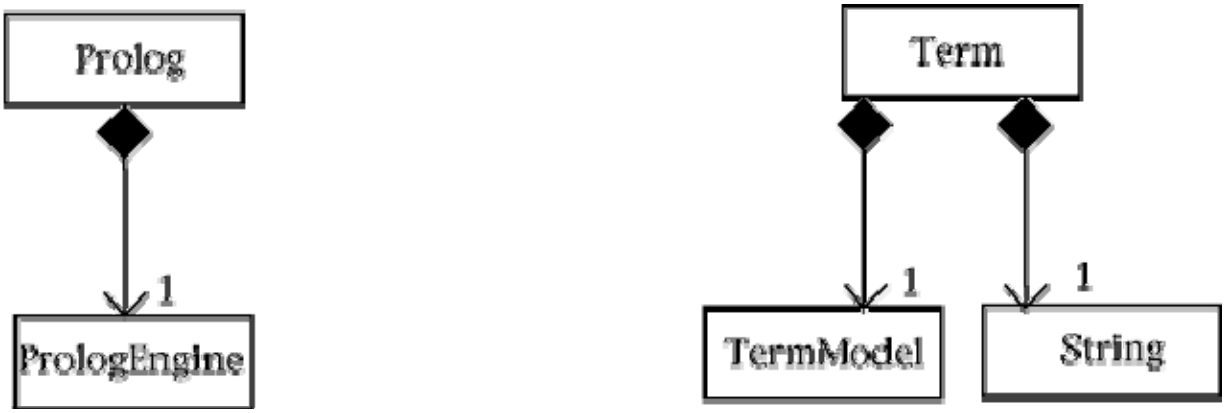

Figura 4.3 Relación de composición entre clases de InterProlog y JProlog

Hay una relación de composición según la figura, entre un objeto Prolog y uno PrologEngine, el primero tendrá una instancia del segundo, cuando se destruye el objeto Prolog, ocurre lo mismo con el objeto PrologEngine. PrologEngine es una clase del paquete InterProlog.

De manera análoga, Term tiene un TermModel, el cual también forma parte de InterProlog, y un String.

En la figura 4.4 se observa que un objeto de la clase Terms, está compuesto en parte por un objeto de clase Vector, dentro de ese Vector se guardan objetos Term. Con StringVector sucede algo parecido, sólo que el Vector que lo integra en lugar de contener objetos Term, tendrá objetos String.

Figura 4.4 Relación de composición entre Terms y StringVector con Vector

Además la excepción UndefinedPredicateException hereda de la clase excepción, esto se representa en la figura 4.5.

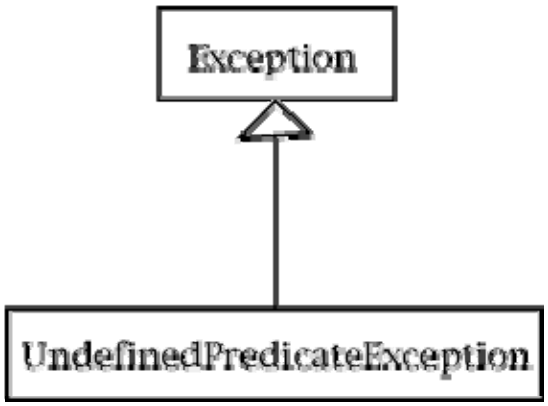

Figura 4.5 Relación de generalización entre Exception y UndefinedPredicateException

4.3 Metodología para recuperar desde Java información de procesamientos en XSB

Para utilizar XSB desde Java, se describe la siguiente forma de estructurar las consultas, y después se mencionan algunos ejemplos representativos como casos de prueba del Sistema JProlog, algunos en plataformas bajo el Sistema Operativo Windows 98 y otros en Solaris.

Sea el archivo grupo.P:

```
/*  para el mundial 2002 grupo(pais, nombre del grupo) */
grupo(italia, g).
grupo(croacia, g).
grupo(ecuador, g).
grupo(mexico,g ).
```

En XSB, se efectuaría una consulta así:

```
?- grupo(X,Y).
```

Desde Java, se haría de esta forma:

```
Terms resultados = prolog.retrieve("grupo (R1, R2)",2);
```

En el primer argumento del método retrieve de un objeto Prolog se envía la consulta con las variables denotadas por R1, R2, ... y así sucesivamente hasta Rn, según tantas variables diferentes se recuperen, el segundo argumento denota el número de variables diferentes a instanciar.

Es muy importante que el nombre de las variables empiecen con la letra “R”. Al efectuar el método mencionado, se regresan valores que se deben guardar en un objeto de la clase Terms, el cual tiene la capacidad de almacenar términos, así como recuperarlos como objetos de la clase Term que es una representación en Java de una constante, variable o lista.

4.3.1 Recuperación de un término formado por caracteres

En el siguiente ejemplo se puede notar cómo se puede recuperar un término formado por caracteres, desde Java. Sea el archivo perro.P en XSB:

```
perro(dodi).
perro(dingo).
perro(cloudy).
perro(rolipan).
perro(pirata).
perro(peki).
```

Entonces, desde Java se puede recuperar información de la siguiente forma, según el ciclo de vida de un objeto Prolog descrito en la figura :

En este caso se recupera la primer ocurrencia de perro(R1) en la base declarativa.

```
/**

        Importar el paquete isc.mac.prolog

*/

import isc.mac.prolog.*;

public class PrimerPerro{

        public static void main(String args[]){

                /* Paso 1 ******************************/
                /* Crear objeto Prolog *****************/

                /*************************************************/
                /* Si es necesario coloque la ruta absoluta de su xsb */
                /*************************************************/
                String rutaProlog = new String("xsb");
                Prolog prolog = new Prolog(rutaProlog);

                /* Paso 2 ******************************/
                /* Cargar módulos  *********************/
                /* Se asume que el modulo en xsb no debe contener errores */
                prolog.load("[perro].");

                /* Paso 4 ******************************/
                /* Ejecucion de consultas */

                /* Se puede omitir paso 3: envio de comandos*/

                Terms resultados = prolog.retrieve("perro(R1)",1);
                String resultadosString = resultados.toString();
                System.out.println("String relativo a resultados:\n\t"+resultadosString);

                int numTerms = resultados.getNumTerms();

                /* Paso 5 ****************************
                /* Proceso de datos obtenidos */
                /* Las cadenas recuperadas se pueden procesar para diversos usos */
                /* Si numTerms es igual a cero significa que el predicado no esta definido */
                /* o bien no hay elementos con los cuales las variables hagan match, de tal */
                /* forma que el vector de terms es vacio */
                if(numTerms != 0){

                        Term term = resultados.getTerm(1);
                        String arg1 = term.toString();
                        System.out.println("String relativo a term:\n\t"+arg1);

                }

                /* Paso 6*************************************/
                /* Cierra proceso prolog y salir del sistema *********************/
                prolog.offProlog();
                System.exit(0);
```

```
        }
}
```

La salida en una consola de MS-DOS en Windows 98 se puede observar en la figura 4.6 al ejecutar “java PrimerPerro”.

4.3.2 Recuperación de enteros

Para este caso se utilizó el archivo “digitoPar.P”, descrito como:

```
digito(0).
digito(1).
digito(2).
digito(3).
digito(4).
digito(5).
digito(6).
digito(7).
digito(8).
digito(9).
par(0).
par(2).
par(4).
par(6).
par(8).

digitoPar(X):- digito(X), par(X).
```

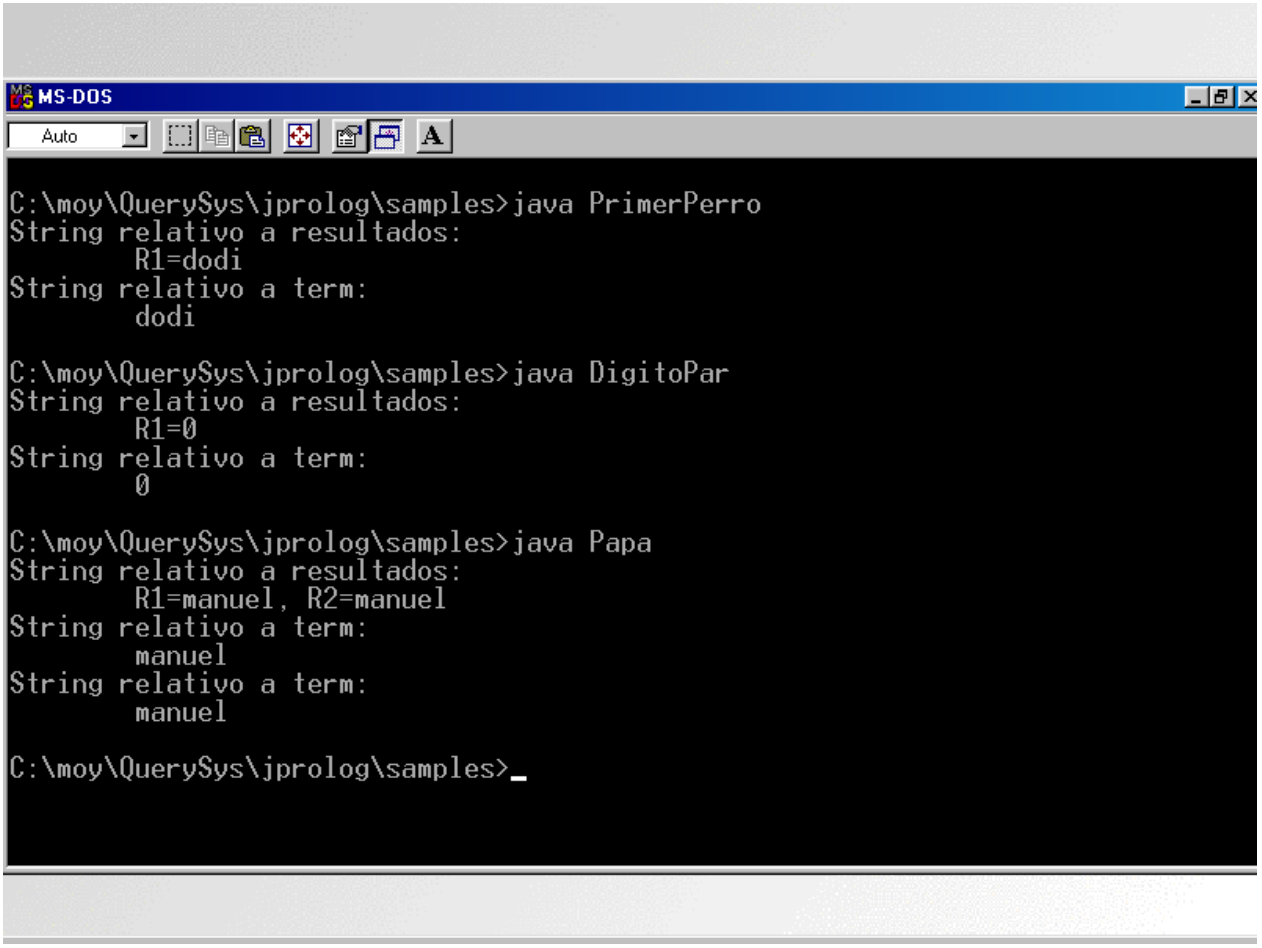

Figura 4.6 Casos de prueba en MS-DOS

Se recupera del archivo “digitopar.P”.

En este caso se recuperará un término que es entero, específicamente el primer dígito par según la regla “digitoPar”del archivo “digitopar.P”. A continuación se muestra el código apropiado desde Java para que desde una aplicación se pueda recuperar lo mencionado.

```
/**
        Importar el paquete isc.mac.prolog
*/
import isc.mac.prolog.*;
public class DigitoPar{
        public static void main(String args[]){
                /*************************************************/
                /* Si es necesario coloque la ruta absoluta de su xsb */
                /*************************************************/
```

```
                String rutaProlog = new String("xsb");
                Prolog prolog = new Prolog(rutaProlog);

                /* Cargar módulos  ********************/
                prolog.load("[digitopar].");

                /************************************************/
                /*R1 es la variable a instanciar, 1 es el número de variables */
                /*El resultado se almacena en el objeto resultados de clase Terms*/
                /************************************************/

                Terms resultados = prolog.retrieve("digitoPar(R1)",1);

                /************************************************/
                /*En el objeto resultadosString de clase String se almacena */
                /*la representación del objeto resultados */
                /************************************************/

                String resultadosString = resultados.toString();
                System.out.println("String relativo a resultados:\n\t"+resultadosString);

                int numTerms = resultados.getNumTerms();

                /************************************************/
                /*Si el número de términos recuperados en el objeto resultados, */
                /*es diferente de cero, quiere decir que la recuperación fue exitosa */
                /************************************************/

                if(numTerms != 0){

                        /* Se recupera el primer término instanciado desde XSB*/
                        /* y se almacena en un objeto term de clase Term*/
                        Term term = resultados.getTerm(1);
                        String arg1 = term.toString();
                        System.out.println("String relativo a term:\n\t"+arg1);

                }

                /* Cerrar proceso prolog y salir del sistema **********************/
                prolog.offProlog();
                System.exit(0);


        }

}
```

La salida en una consola de MS-DOS, se puede apreciar en la figura 4.6 al ejecutar "java DigitoPar"

4.3.3 Recuperación de listas

En el siguiente ejemplo se describe cómo se puede recuperar una lista desde Java, en esta prueba se recupera el código y costo del agua en un término lista.

Sea el archivo “producto.P” en XSB:

```
/* producto([codigo, precio unitario]) */

azucar([12, 11.5]).
leche([13, 12.3]).
miel([7, 5.2]).
agua([20, 17.7]).
```

Y la aplicación en Java:

```
import isc.mac.prolog.*;

public class Producto{

	public static void main(String args[]){

		/*************************************************/
		/* Si es necesario coloque la ruta absoluta de su xsb */
		/*************************************************/
		String rutaProlog = new String("/archivos/vol07/is104286/XSB/config/sparc-sun-
solaris2.6/bin/xsb");
		Prolog prolog = new Prolog(rutaProlog);

		/* Cargar módulo  *********************/
		prolog.load("[producto].");

		Terms resultados = prolog.retrieve("agua(R1)",1);
		String resultadosString = resultados.toString();
		System.out.println("String relativo a resultados:\n\t"+resultadosString);

		int numTerms = resultados.getNumTerms();

		/*************************************************/
		/* Se recupera el primer término en un objeto term de clase Term */
		/*************************************************/

		if(numTerms != 0){

			Term term = resultados.getTerm(1);
			String arg1 = term.toString();
			System.out.println("String relativo a term:\n\t"+arg1);

		}

		prolog.offProlog();
		System.exit(0);


	}
```

}

La salida obtenida en consola se puede apreciar en la figura 4.7 al momento de ejecutar “java Producto”.

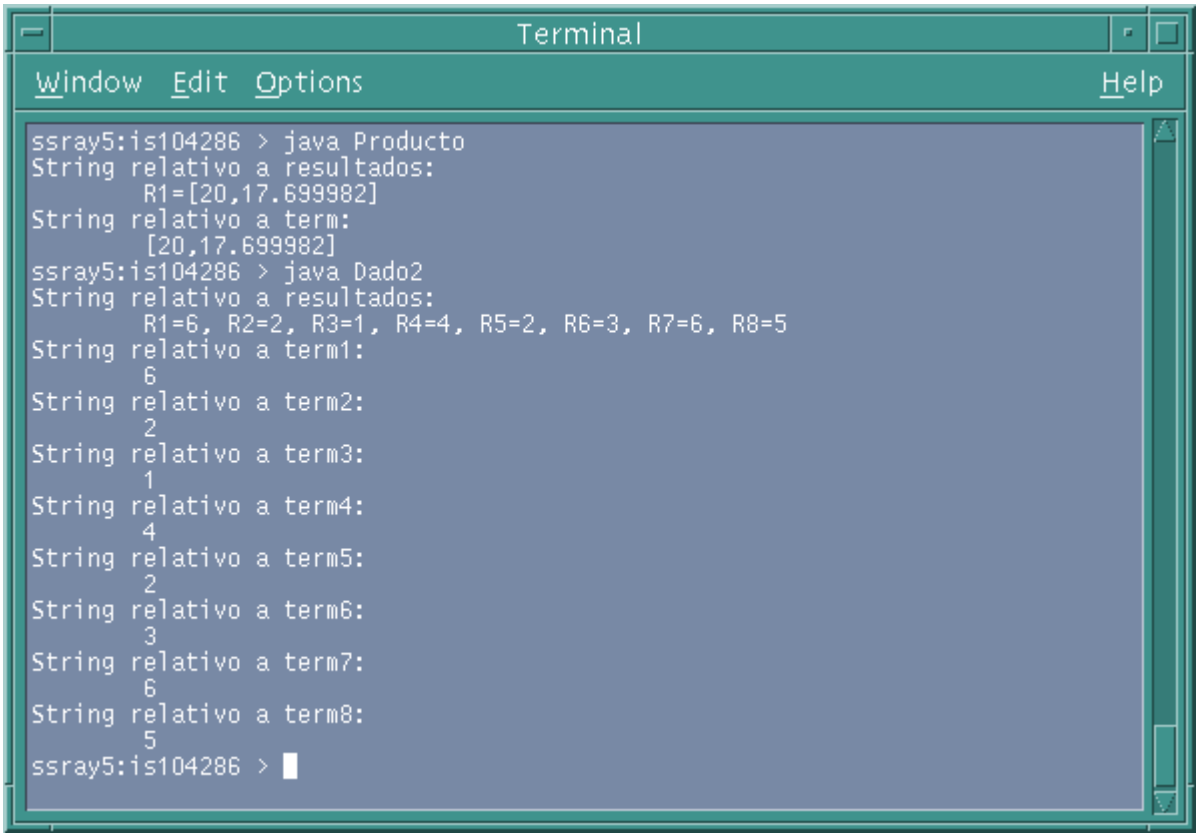

Figura 4.7 Casos de prueba en Solaris

4.3.4 Recuperación de dos o más términos

Sea el archivo “papa.P”:

```
/* papa(X,Y) X es papa de Y */

papa(manuel, manuel).
papa(manuel, luis).
papa(juan, pablo).
papa(mario, esteban).
papa(miguel, omar).
papa(jose, angel).
papa(juan, omar).
papa(felipe, hector).
```

En esta situación al método “retrieve” del objeto “prolog”, se le envía la consulta con las variables que se quieren recuperar, las cuales se identifican por R1, R2, ..., Rn, según las diferentes variables que se pretenden recuperar.

Como en el ejemplo se quiere obtener el valor asociado a dos variables diferentes, se envía un 2 como segundo parámetro (indica el número de variables diferentes de la consulta), al método retrieve.

```
/**

        Importar el paquete isc.mac.prolog

*/

import isc.mac.prolog.*;

public class Papa{

        public static void main(String args[]){

                String rutaProlog = new String("xsb");

                Prolog prolog = new Prolog(rutaProlog);
                Term tmpTerm;
                String tmpStrTerm;
                prolog.load("[papa].");

                Terms resultados = prolog.retrieve("papa(R1, R2)",2);
                String resultadosString = resultados.toString();
                System.out.println("String relativo a resultados:\n\t"+resultadosString);

                int numTerms = resultados.getNumTerms();

                /************************************************/
                /* Se recuperan los términos asociados al predicado papa con aridad 2*/
                /************************************************/
                for(int i=1; i<=numTerms; i++){

                        tmpTerm = resultados.getTerm(i);
                        tmpStrTerm = tmpTerm.toString();
                        System.out.println("String relativo a term:\n\t"+tmpStrTerm);

                }

                prolog.offProlog();
                System.exit(0);

        }
}
```

La salida en consola al ejecutar “java Papa”, se puede apreciar en la figura 4.6.

Otro ejemplo en esta sección es el de Dado.java con el cual se recupera el resultado de 10 lanzamientos de un dado, en consecuencia se necesitan 10 variables diferentes, R1, R2 ..., R10, y el número de variables diferentes en el segundo argumento es 10.

A continuación se muestra el archivo “dado.P”, y el código de la aplicación en Java:

```
/* 10 tiradas de un dado en 7 ocasiones*/
eventosDado(1,3,2,5,6,4,3,2,1,2).
eventosDado(3,2,1,1,1,4,5,4,3,6).
eventosDado(5,4,3,2,3,6,1,5,3,2).
eventosDado(4,5,6,5,3,2,3,1,5,4).
eventosDado(6,3,5,2,5,1,4,6,2,3).
eventosDado(6,2,6,1,4,2,3,6,2,5).
eventosDado(3,6,4,5,1,2,1,3,6,5).
```

```
/**

        Importar el paquete isc.mac.prolog

*/

import isc.mac.prolog.*;

public class Dado{

        public static void main(String args[]){

                /*************************************************/
                /* Si es necesario coloque la ruta absoluta de su xsb */
                /*************************************************/
                String rutaProlog = new String("xsb");
                Prolog prolog = new Prolog(rutaProlog);
                Term tmpTerm;
                String tmpStrTerm;

                prolog.load("[dado].");




                /*************************************************/
                /* En la consulta se coloca el predicado con las variables R como un String como primer
argumento*/
                /* En el segundo argumento se especifica el número de variables diferentes a recuperar */
                Terms resultados = prolog.retrieve("eventosDado(R1, R2, R3, R4, R5, R6, R7, R8,R9,
R10)", 10);
                String resultadosString = resultados.toString();
                System.out.println("String relativo a resultados:\n\t"+resultadosString);

                int numTerms = resultados.getNumTerms();

                for(int i=1; i<=numTerms; i++){

                        tmpTerm = resultados.getTerm(i);
                        tmpStrTerm = tmpTerm.toString();
```

```
                    System.out.println("String relativo a term"+i+":\n\t"+tmpStrTerm);

                }

                prolog.offProlog();
                System.exit(0);


        }

}
```

El resultado al ejecutar “java Dado” en una consola de MS-DOS, se obtiene el resultado que se observa en la figura 4.8.

Otro ejemplo es el de Dado2.java, en el cual se exige que el primer y tercer tiro sean iguales, y también que lo sean el sexto tiro con respecto al noveno, por ello, el número de variables diferentes en este ejemplo disminuye y es de 8, sin embargo después de las variables que hacen referencia a las anteriores se continúa con el crecimiento progresivo del número que acompaña a la R.

Este ejemplo utiliza el mismo archivo dado.P del caso de prueba anterior, sin embargo el código Java asociado sí es diferente:

```
/**

        Importar el paquete isc.mac.prolog

*/

import isc.mac.prolog.*;

public class Dado2{

        public static void main(String args[]){

                String rutaProlog = new String("/archivos/vol07/is104286/XSB/config/sparc-sun-
solaris2.6/bin/xsb");
                Prolog prolog = new Prolog(rutaProlog);
                Term tmpTerm;
                String tmpStrTerm;

                prolog.load("[dado].");

                /*************************************************/
                /* En este ejemplo el primer término debe ser igual al tercero y el sexto igual al noveno*/
                /* por tanto son ocho variables diferentes las que se recuperaran*/
                /*************************************************/

                Terms resultados = prolog.retrieve("eventosDado(R1, R2, R1, R3, R4, R5, R6, R7, R5,
R8)", 8);
                String resultadosString = resultados.toString();
                System.out.println("String relativo a resultados:\n\t"+resultadosString);
```

```
                int numTerms = resultados.getNumTerms();

                for(int i=1; i<=numTerms; i++){

                        tmpTerm = resultados.getTerm(i);
                        tmpStrTerm = tmpTerm.toString();
                        System.out.println("String relativo a term"+i+":\n\t"+tmpStrTerm);

                }

                prolog.offProlog();
                System.exit(0);


        }

}
```

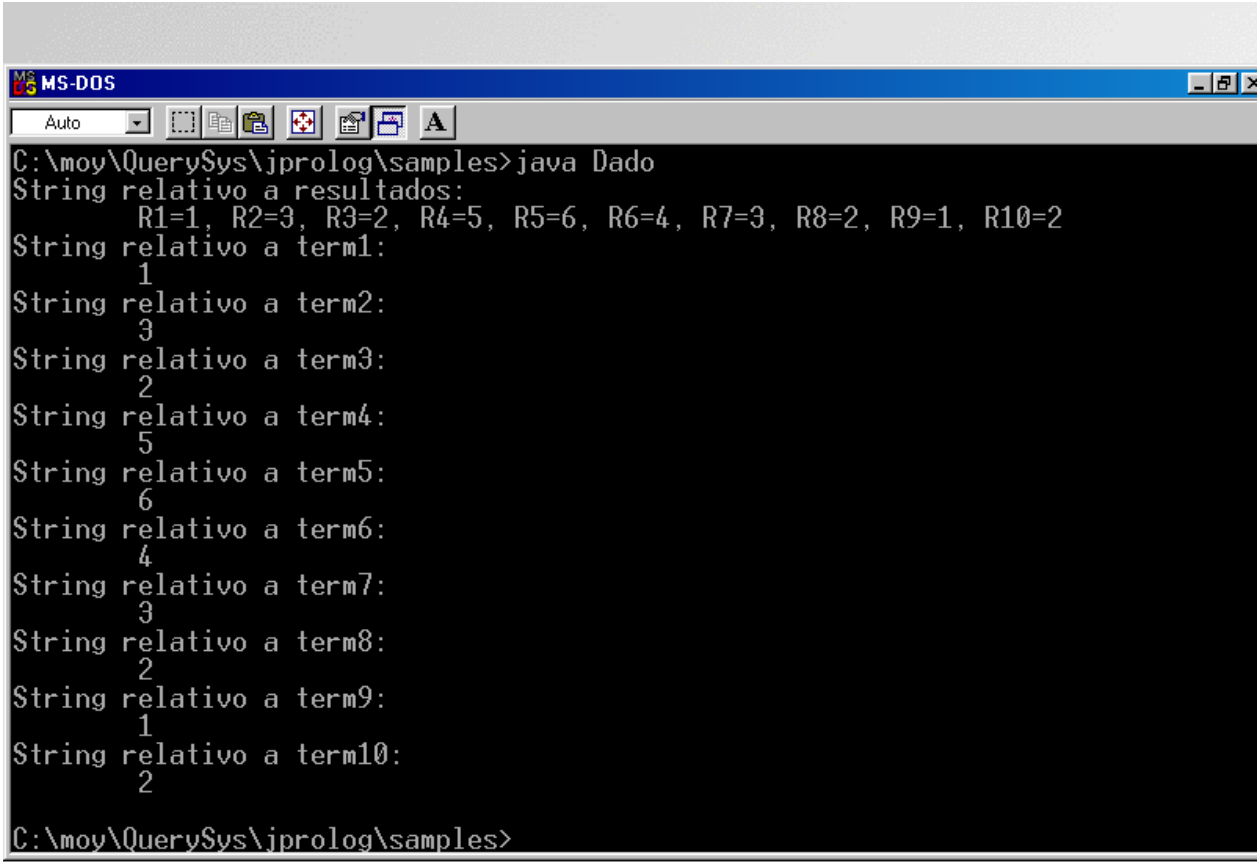

Figura 4.8 Casos de prueba en Windows 98 (2)

Al ejecutar en una terminal de Solaris en una máquina sunray “java Dado2”, se obtiene la respuesta que se aprecia en la figura 4.7.

4.3.5 Recuperación de todas las respuestas asociadas a un predicado

Cuando se necesitan obtener todas las respuestas asociadas a un predicado en una consulta, se debe utilizar el predicado “setof” de la forma que se describe en este ejemplo. La primer variable es auxiliar y por tal razón nos interesa recuperar la segunda variable, que contiene las instancias que satisfacen X, en una lista.

Este ejemplo utiliza el archivo “perro.P”, utilizado en la sección 4.3.1. La aplicación codificada en Java queda de la siguiente manera:

```
import isc.mac.prolog.*;

public class TodoPerro{

        public static void main(String args[]){

                String rutaProlog = new String("/archivos/vol07/is104286/XSB/config/sparc-sun-
solaris2.6/bin/xsb");
                Prolog prolog = new Prolog(rutaProlog);

                prolog.load("[perro].");

                /************************************************/
                /*Para recuperar todos los posibles valores para los cuales se cumple */
                /*el predicado, se utiliza el predicado setof con la variable auxiliar R1*/
                /*Desde Java sólo será útil el valor de R2 que será una lista */
                /************************************************/
                Terms resultados = prolog.retrieve("setof(R1,perro(R1),R2)",2);
                String resultadosString = resultados.toString();
                System.out.println("String relativo a resultados:\n\t"+resultadosString);

                int numTerms = resultados.getNumTerms();

                if(numTerms != 0) /* Se instanciaron las variables R1 y R2 */{

                        /*****************************/
                        /* Nos interesa el resultado de R2 */
                        /*****************************/
                        Term term = resultados.getTerm(2);
                        Term head;
                        int i=1;

                        while(term.isList()){

                                        head = term.getHead();

                                        System.out.print("\n\tCabeza "+i+":"+head);

                                        term = term.getTail();
                                        i++;
                        }
                }

                System.out.println();
```

```
                prolog.offProlog();
                System.exit(0);
        }

}
```

Y los resultados que se observan en consola se pueden observar en la figura 4.9.

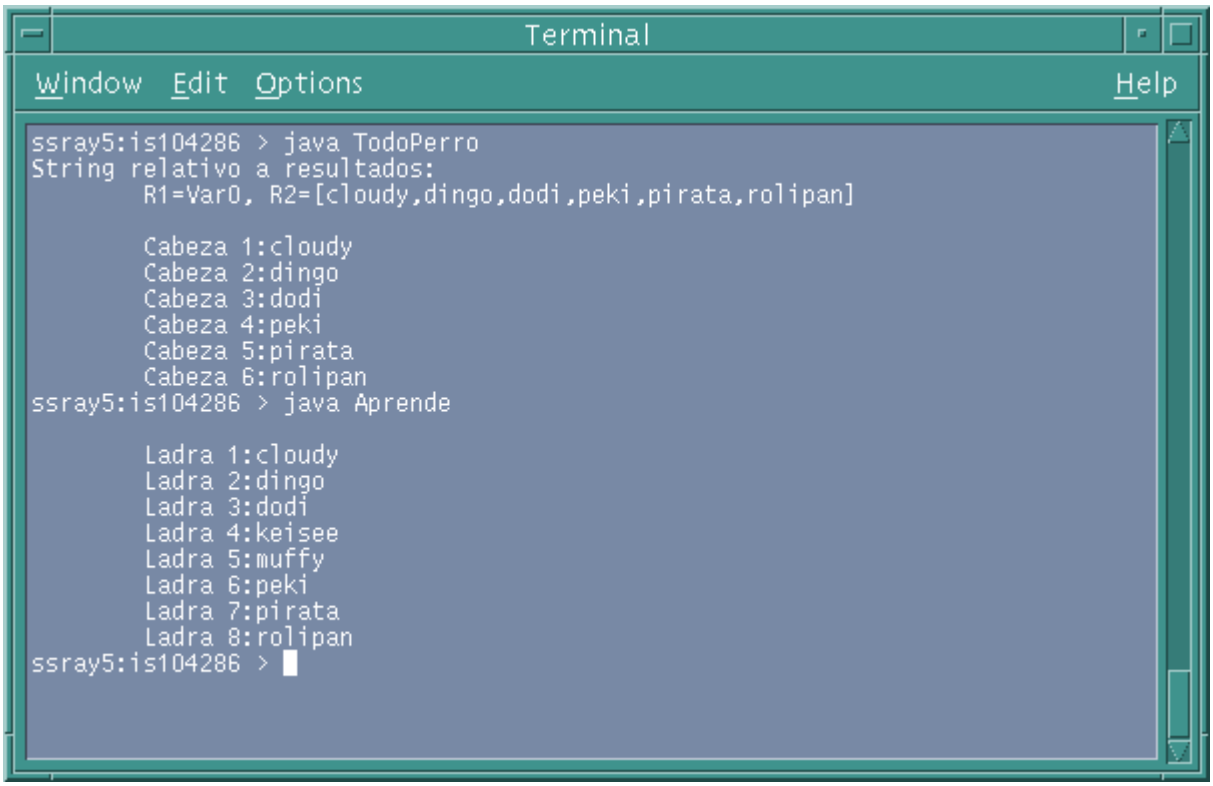

Figura 4.9 Casos de prueba en Solaris (2)

4.3.6 Recuperación de cabeza y cola de un término lista

Sea el archivo “deportes.P” definido así:

```
/* paises calificados al munidal 2002, segun la region a la que pertenecen */
/* futbol(zona, paises calificados) */

futbol(afc, [corea_del_sur, japon, china, arabia_saudita]).
futbol(caf, [sudafrica, camerun, senegal, tunisia, nigeria]).
futbol(concacaf, [costa_rica, estados_unidos, mexico]).
futbol(conmebol, [argentina, paraguay, ecuador, brasil, uruguay]).
futbol(ofc, []). /* Australia fue eliminado por Uruguay */
futbol(uefa, [francia, polonia, suecia, espana, rusia, portugal, dinamarca, croacia, italia, inglaterra, eslovenia,
turquia, belgica, alemania, irlanda]).
```

La siguiente aplicación muestra cómo se pueden obtener todos los elementos de un término lista, sólo hay que almacenar en una lista temporal la cola de la lista y luego obtener la cabeza de la lista temporal, después se guarda en la lista temporal la cola de la lista temporal y así sucesivamente, tantas veces como el término que se obtenga continúe siendo lista.

```
import isc.mac.prolog.*;

public class Deportes{

	public static void main(String args[]){


		/************************************************/
		/* En su caso coloque la ruta absoluta de su xsb */
		/************************************************/

		String pathProlog = new
String("/archivos/vol07/is104286/XSB/config/sparc-sun-solaris2.6/bin/xsb -m2000");
		Prolog prolog = new Prolog(pathProlog);

		prolog.load("[deportes].");

		Terms resultados = prolog.retrieve("futbol(uefa,R1)",1);

		System.out.println("Calificados de uefa: "+resultados);

		int nt = resultados.getNumTerms();

		Term term = resultados.getTerm(1);
		Term head;
		int i=1;



		if(nt!=0) /* Se instanció la variable R1*/
		{
			/*********************************************/
			/*Se recuperan los términos que conforman una lista */
			/*********************************************/
			while(term.isList()){

				head = term.getHead();

				System.out.print("\n\tCabeza "+i+":"+head);

				term = term.getTail();
				i++;

			}
		}

		resultados = prolog.retrieve("futbol(ofc, R1)",1);

		System.out.println("\nCalificados de ofc: "+resultados);
```

```
                nt = resultados.getNumTerms();

                term = resultados.getTerm(1);
                i=1;

                /*************************************************************/
                /* En nt diferente a cero se sugiere para tener una aplicación más robusta */
                /*************************************************************/

                while(term.isList()){

                        head = term.getHead();

                        System.out.print("\n\tCabeza "+i+":"+head);

                        term = term.getTail();
                        i++;
                }
                prolog.offProlog();
                System.exit(0);
        }
}
```

La salida de la aplicación se observa en la figura 4.10 al ejecutar “java Deportes”

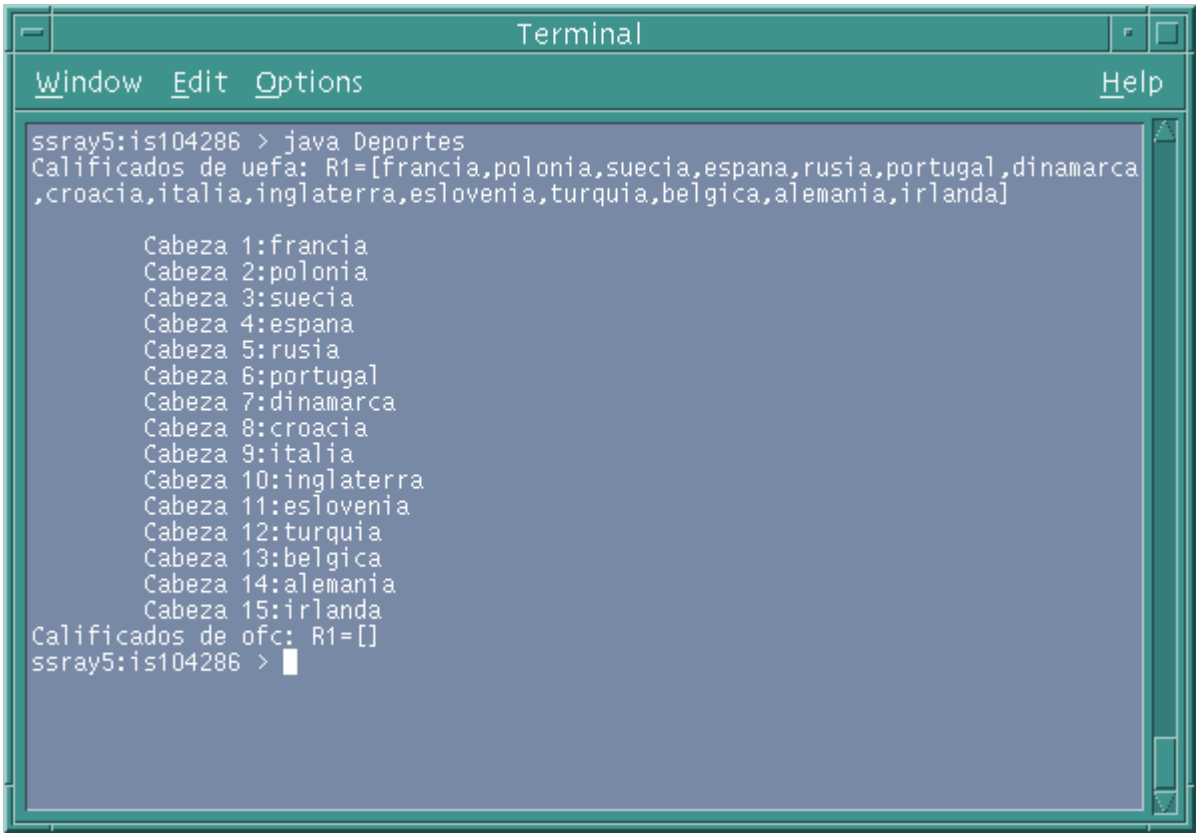

Figura 4.10 Casos de prueba en Solaris (3)

4.3.7 Envío de comandos en tiempo de ejecución: assert

En este ejemplo se describe cómo es posible utilizar el predicado “assert” para enriquecer las reglas de un programa. En este ejemplo se considera el archivo “perro.P” que se utilizó en la sección 4.3.1

```
import isc.mac.prolog.*;

public class Aprende{

	public static void main(String args[]){

		/* Paso 1 ******************************/
		/* Crear objeto Prolog *****************/

		/*************************************************/
		/* En su caso coloque la ruta absoluta de su xsb */
		/*************************************************/
		String pathProlog = new String("/archivos/vol07/is104286/XSB/config/sparc-sun-
solaris2.6/bin/xsb -m2000");
		Prolog prolog = new Prolog(pathProlog);

		prolog.load("[perro].");

		/* Paso 3 ******************************/
		/* Envio de comandos  ******************/

		prolog.send("assert(ladra(keisee)).");
		prolog.send("assert(ladra(muffy)).");

		/*************************************/
		/*Aunque no se utilizarán la variable R1, el assert de efectuarse así*/
		/*************************************/
		Terms tmpResultados = prolog.retrieve("assert((ladra(R1) :-
perro(R1)))", 1);

		Terms resultados = prolog.retrieve("setof(R1, ladra(R1), R2)",2);
		int nt = resultados.getNumTerms();

		Term term = resultados.getTerm(2);
		Term head;
		int i=1;

		if(nt!=0) /* Se instanciaron las variables R1 y R2 */
		{
			while(term.isList()){

				head = term.getHead();

				System.out.print("\n\tLadra "+i+":"+head);
```

```
                    term = term.getTail();
                    i++;

                }
            }

            System.out.println();
            prolog.offProlog();
            System.exit(0);


        }

}
```

En la salida de consola mostrada en la figura 4.9 al ejecutar “java Aprende”, se puede notar que se incorporaron adecuadamente tanto los dos hechos como la nueva regla enviadas a XSB desde Java.

4.4 Metodología general sugerida

En diagrama mostrado en la figura 4.11, se resume la metodología general sugerida, primero colocar en un String el valor de la rutaProlog, de ahí enviar esa ruta al objeto de la clase Prolog para que conozca en dónde encontrar a XSB, después es necesario cargar los módulos en memoria.

Una vez realizado lo anterior, se pueden recuperar los resultados en un objeto de la clase Terms, y de éste se puede obtener cada término Term para convertirlo a una cadena y tener la posbilidad de procesar esa cadena desde el lenguaje Java.

De ahí que ya se tiene una herramienta para establecer una comunicación de calidad entre Java y XSB y por tal razón se procede a utilizar la interfaz con una aplicación más sofisticada: NATLIN para el Volcán Popocatépetl.

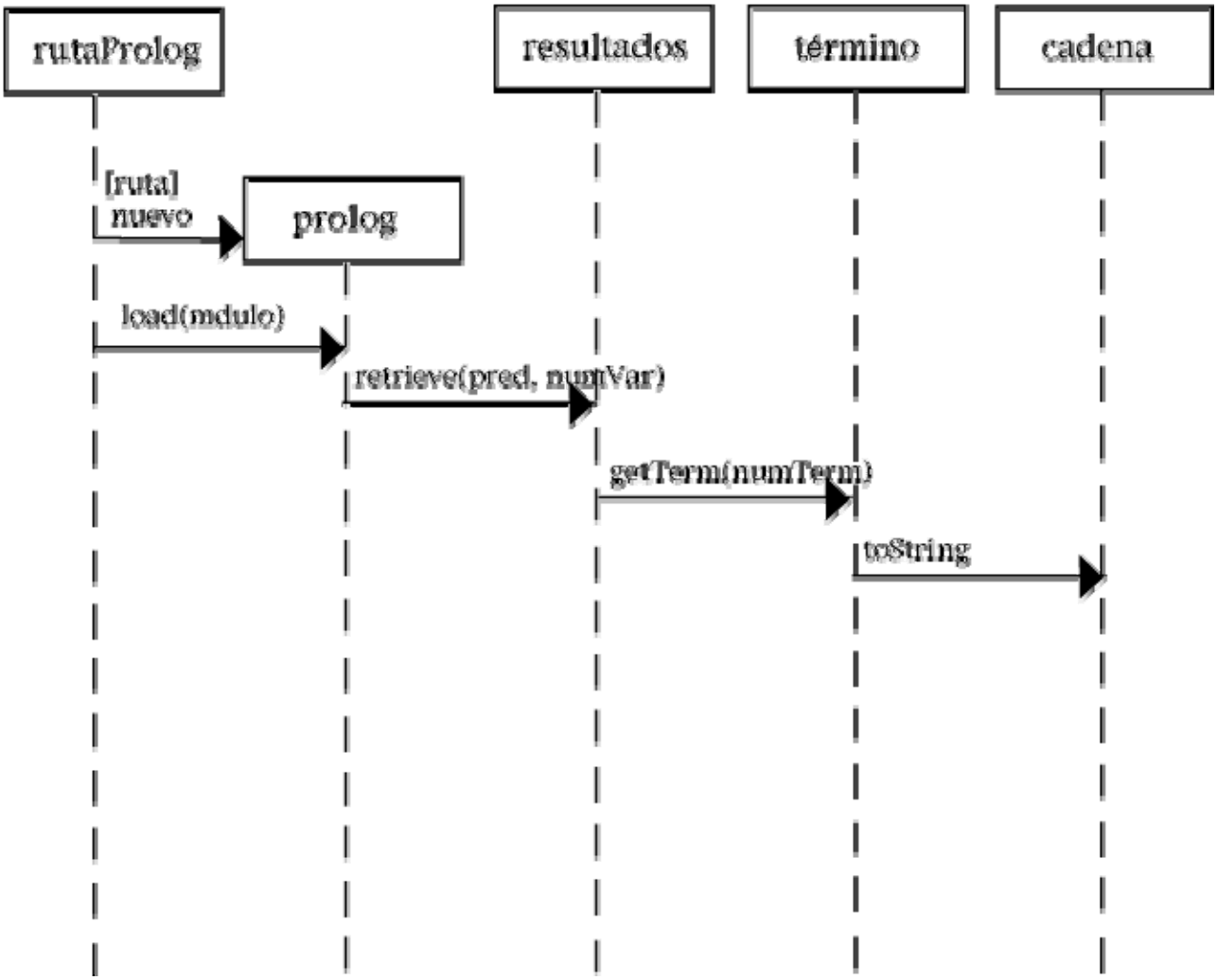

Figura 4.11 Diagrama de secuencia