Índice general

Capítulo 1

Introducción

1.1. Definición y Objetivos

Time Oriented Language, en adelante `TOL`, es un lenguaje de programación diseñado para el análisis de series temporales y procesos estocásticos basado en la representación algebraica del tiempo y las series temporales que permite:

- Estructurar los datos procedentes de los sistemas operacionales, dándoles un soporte temporal y clasificador que los convierte en una información útil para comprender su comportamiento.
- Analizar la información dinámica, generar modelos estadísticos, identificar los factores que influyen en el comportamiento temporal y extraer conocimiento
- Facilitar la toma de decisiones a partir del conocimiento, de los modelos estadísticos, de las previsiones del comportamiento y de las funciones y opciones de decisión.

El tiempo es una magnitud continua representable como la recta real pero al tratar con series temporales de naturaleza social como ventas, llamadas telefónicas, movimientos bancarios, asistencia a eventos, tráfico, etc., se observan comportamientos ligados a diferentes características temporales como son los ciclos dentro de un periodo (que puede ser el día, la semana, el mes, el año ...); o la proximidad a festivos y cualesquiera otros eventos identificables con el momento en que ocurren.

Tradicionalmente estos fenómenos se han representado por extensión, es decir, como una enumeración de eventos fabricada más o menos artesanalmente; lo cual no supone un problema muy grave para series anuales o mensuales como máximo. Pero la explosión de la información ha traído series diarias, horarias o de frecuencias aún más altas, para las que se hace necesario un tratamiento algebraico de este tiempo social que, a partir de una base de conjuntos temporales primarios y una colección de funciones, permitan construir analíticamente

conjuntos temporales complejos que representen los comportamientos sociales más intrincados.

Por otro lado el lenguaje ha de ofrecer mecanismos de análisis estadístico de los datos, desde la lectura de los mismos en archivos o bases de datos a la producción de modelos predictivos pasando por los métodos de representación visual de la información que se hagan necesarios.

Dentro de este contexto nace TOL para intentar dar respuesta a todas estas necesidades y a otras que han ido surgiendo durante el proceso de implementación y maduración del lenguaje.

1.2. Diseño

Estas son algunas de las características técnicas más importantes del lenguaje y que más han influido en el diseño y la implementación de TOL .

- El lenguaje TOL ha sido esarrollado en C++ por ser uno de los lenguajes más rápidos, potentes y versátiles que existen. Desde luego, también es uno de los más complicados y de los que más tiempo de formación requieren.
- Se ha desarrollado con vocación multiplaforma y se ha compilado el código bajo Windows y bajo diferentes sistemas UNIX (LINUX, HPUX, AIX, IRIX y SOLARIS); intentando siempre que el código sea lo más compatible posible. No obstante, para algunas rutinas de bajo nivel no ha habido más remedio que hacer implementaciones ad hoc para cada sistema operativo.
- El lenguaje dispone, prácticamente desde su inicio, de un entorno gráfico llamado TOLBASE que ha facilitado la labor del analista permitiéndole obtener representaciones gráficas de los objetos programados, como series temporales, conjuntos temporales, matrices, funciones, conjuntos, etc. Esta particularidad ha influido en la estructura del lenguaje pues los objetos definidos deben sobrevivir en memoria después del tiempo de ejecución del programa, al contrario de lo que ocurre con muchos otros lenguajes de programación.
- TOL es un lenguaje de programación interpretado, lo cual facilita el uso y el aprendizaje de los analistas. En dos o tres meses se puede alcanzar un nivel de programación aceptable partiendo de personas sin formación informática específica y con poca o ninguna experiencia en programación.
- Es un lenguaje fuertemente tipado aunque existe un tipo comodín llamado `Anything` que permite construir funciones que traten objetos de diferentes tipos, característica ésta de la que no conviene abusar.
- TOL permite crear estructuras de datos y funciones de usuario con lo que el analista puede ampliar el lenguaje y adaptarlo a sus necesidades guardando en ficheros el código y organizándolo en librerías. El sistema TOL, de hecho, no sólo consta de código C++ sino que se complementa

con una serie de librerías básicas. Sólo por razones de eficiencia conviene a veces crear algunas nuevas funciones en C++.

- TOL admie recursividad en las funciones (monocíclica y policíclica); esto es, una función se puede llamarse a sí misma o a otras que a su vez llamen a otras y así sucesivamente hasta que alguna vuelva a llamar a la inicial. Esto ofrece grandes ventajas en cuanto a potencia y claridad del código pero no se debe abusar porque se carga demasiado la pila de llamadas y puede dar lugar a errores del tipo stack overflow.

- Es un lenguaje autoevaluable, es decir, se puede usar el lenguaje para construir código TOL y evluarlo en tiempo de ejecución. Esta es una de las características más productivas de TOL a la hora de enfrentar problemas masivos en los que un código ha de ejecutarse muchas veces con variaciones derivadas de una cierta estructura de datos.

- TOL es básicamente declarativo pues como ya se ha dicho, los objetos no son el medio sino el fin del lenguaje. Incluso las funciones son objetos de tipo Code y pueden ser argumentos de otras funciones y almacenarse en estructuras permitiendo la construcción de algoritmos complicados de forma estructural y no secuencial.A pesar de ello se han ido introduciendo algunas capacidades de control del flujo propias de los lenguajes secuenciales para dotar al programador de mayor potencia de implementación algorítmica. Por ser un lenguaje interpretado, el control de flujo es necesariamente más lento que en los lenguajes compilados.

- Una de las tareas más complicadas de la programación es el manejo eficiente de la memoria por ser éste uno de los recursos más caros y limitados de la computadora. El manejo de memoria en TOL es dinámico y transparente al usuario, los objetos existen mientras son necesarios y desaparecen cuando nadie los usa, todo ello merced a un sistema de referencias.

- Implementa una representación algebraica del tiempo que permite manipular conjuntos temporales y series temporales como objetos virtualmente infinitos, que no son representables por extensión como tipos de datos en el sentido clásico de la programación, sino que más bien son clases definidas por funciones que determinan su comportamientorequiriéndose un lenguaje de programación orientado al objeto como lo es C++.

- Esto obliga a plantear un mecanismo de evaluación de tipo lazy, es decir, la evaluación del objeto se retrasa al momento en que es requerido por otro objeto, puesto que la información que lo constituye no puede ser enumerada sino que se construye la información necesaria cada vez que hace falta a partir de ciertos métodos intrínsecos. Este tipo de evaluación se ha extendido incluso a los tipos de datos que no lo precisarían, como los números, los textos y las fechas que si son almacenables en memoria de forma extensa.

Capítulo 2

Conceptos Básicos

2.1. Forma de un programa TOL

2.2. Palabras reservadas

2.3. Tipos de datos en TOL

Capítulo 3

Tipos de Datos en Tol

Capítulo 4

Constantes, Operadores y Expresiones

4.1. Constantes

4.2. Operadores

4.3. Expresiones TOL

4.3.1. Expresiones simples

4.3.2. Declaraciones de variable

4.3.3. Expresiones compuestas

4.3.4. Expresiones anidadas

Capítulo 5

Sentencias de control de flujo

5.1. Sentencias booleanas

5.2. Sentencias de iteración

5.3. Sentencias de inclusión

Capítulo 6

Funciones

6.1. Funciones internas del lenguaje

6.2. Funciones especiales

6.2.1. Funciones de aplicación de métodos

6.3. Funciones de usuario

6.3.1. Forma general de una función

6.3.2. Reglas de ámbito de las funciones

6.3.3. Reglas de ámbito de las funciones

6.3.4. Retorno de una función

6.3.5. Recursividad

Capítulo 7

Series temporales

7.1. El Tiempo

7.2. Series temporales

Capítulo 8

Acceso a Bases de Datos

TOl tiene una interfaz que permite efectuar múltiples operaciones sobre bases de datos.

Para efectuar una operación sobre una base de datos determinada, en primero lugar debe abrirse; Después se ejecuta la operación para terminar cerrándola. El proceso es sencillo y se especifica detalladamente a continuación.

8.1. Apertura: DBopen

La función de TOL que abre una base de datos es *DBOpen*. Su prototipo es como sigue:

```
Real DBopen (Text alias, Text usuario, Text clave [, Set estructuraDB]);
```

- Argumentos:

alias: Especifica el nombre del alias a través del cual se van a controlar las operaciones efectuadas en las bases de datos abiertas. Debe ser único para cada base de datos, de forma que no se confundan las bases de datos usadas posteriormente. En el caso de acceso ODBC este alias coincide con el alias especificado en el sistema ODBC.

usuario: Especifica el nombre del usuario con el que se accede a la base de datos.

clave: La clave que acompaña al usuario en el acceso.

estructuraDB: Argumento opcional que permite el acceso a bases de datos de forma nativa. Su declaración puede depender del gestor al cual se accede, si bien en la mayoría de los casos se limitará al nombre del servidor y de la base de datos.

- Valor devuelto:

DBopen devuelve un 1 (Verdadero) en caso de éxito, y un 0 (Falso) en caso de error. En caso de error, un mensaje por pantalla ayudará a identificar el problema (Acceso denegado, alias ya existente, servidor no accesible, etc.).

8.1.1. ODBC

La apertura de bases de datos ODBC bajo TOL es muy sencilla, ya que únicamente se hace necesario especificar el alias ODBC, un usuario y su clave de acceso. Por supuesto, la base de datos debe haber sido especificada previamente en el sistema ODBC.

- Estructura:

```
Struct DBStructODBC
{
  Text driver
};
```

Por compatibilidad con versiones anteriores de TOL, si el gestor no es especificado mediante el uso de esta estructura, se utilizará ODBC como gestor por defecto a la hora de abrir una base de datos, pero TOL avisará del uso erróneo mediante un mensaje.

- Ejemplo de uso:

```
Struct DBStructODBC
{
  Text driver
};
Set db = DBStructODBC("odbc");
Real resultado = DBOpen("Alias","usuario","passwd",db);
```

Como se ha explicado con anterioridad, por compatibilidad con versiones anteriores de TOL, también se permite el uso siguiente, que sería equivalente al anterior:

```
Real resultado = DBOpen("Alias","usuario","passwd");
```

Este uso se considera erróneo y debe evitarse.

8.1.2. Acceso nativo

Además de ODBC, TOL permite el acceso de forma nativa a múltiples gestores de bases de datos, consiguiendo de esta forma un acceso mas rápido y fiable.

Si bien cada uno de estos gestores tiene un comportamiento completamente distinto, la interfaz de acceso a bases de datos de TOL ayuda a hacerlo de forma transparente.

A continuación se especifica el comportamiento de cada uno de los gestores existentes.

8.1.2.1. mySQL

La interfaz mySQL de TOL requiere tan sólo especificar el gestor, servidor y base de datos en la estructura detallada en la sección 8.1.

- Estructura:

```
Struct DBStructMySQL {
  Text driver,
  Text database,
  Text host
};
```

driver: Especifica el gestor a utilizar. En este caso concreto, su valor será "mysql".

host: Especifica el nombre o dirección IP del servidor de la base de datos. Ejemplos válidos son "localhost", "server.domain.es" y "127.0.0.1".

database: Especifica el nombre de la base de datos. Ejemplo: "test".

- Ejemplo:

```
Struct DBStruct {
  Text driver,
  Text database,
  Text host
};
Set db1 = DBStruct("mysql", "test", "localhost");
Real DBOpen("alias","root","",db1);
```

Una vez abierta la base de datos, su comportamiento no varía con respecto al resto de gestores.

8.1.2.2. HuffmanByFields

(En desarrollo y documentación)

8.1.2.3. Otros gestores

La implementación de otros gestores de bases de datos está siendo llevada a cabo en estos momentos, y según sean considerados estables serán incorporados al lenguaje. Además, la interfaz permite el uso de cualquier tipo de soporte de almacenamiento en el acceso, que no tiene por qué ser una base de datos, como pueda ser un fichero con un determinado formato.

8.2. Cierre: DBClose

Una vez se haya terminado de usar una base de datos, ésta debe ser cerrada para liberar todos los recursos que TOL reserva para su utilización. Para hacer esto se cuenta con la función *DBClose:*

```
Real DBClose(Text aliasName);
```

El único parámetro de DBClose especifica el alias con el cual la base de datos fué abierta con anterioridad. Un ejemplo correcto de apertura y cierre de dos bases de datos mysql y una ODBC con TOL sería:

```
//Especificacion de las estructuras necesarias:
Struct DBStructMySQL {
  Text driver,
  Text database,
  Text host
};
Struct DBStructODBC {
  Text driver
};
//Apertura:
Set db1 = DBStructMySQL("mysql", "test", "localhost");
Real DBOpen("aliasTest","user","passwd",db1);
Set db2 = DBStructMySQL("mysql", "empresa", "myserver");
Real DBOpen("aliasEmpresa","user","passwd",db2);
Set db3 = DBStructODBC("odbc");
Real DBOpen("aliasName","user","passwd",db3);
//Cierre de las bases de datos abiertas:
Real DBClose(“aliasEmpresa”);
Real DBClose(“aliasTest”);
Real DBClose(“aliasName”);
```

8.3. Operaciones sobre bases de datos

Existen varias funciones en TOL para efectuar operaciones sobre bases de datos. Los prototipos son:

```
Real DBExecQuery(Text consulta);
Set DBTable (Text consulta [, Text nombreEstructura]);
Matrix DBMatrix(Text consulta);
Set DBSeries(Text consulta, timeSet fechado, Set nombres
  [, Set Set descripciones, Real valorDefecto=0]);
Set DBSeriesTable (Text consulta , TimeSet fechado, Set nombres
  [, Set descripciones, Real valorDefecto=0]);
Set DBSeriesColumn (Text consulta , TimeSet fechado
```

```
  [, Real valorDefecto=0]);
Real DBSpool (Text consulta, Text fichero
  [, Text cabecera="",Text separadorColumnas=";",
  Text separadorFilas=";",Real formateo=CIERTO ]);
Real DBCreateSeriesTable (Text nombreTabla, Set series
  [, Text nombreFecha , Text tipoFecha, Text formatoFechas,
  Set nombres]);
```

8.3.1. DBExecQuery

- Qué hace: Ejecuta un comando en la base de datos activa. En el caso de comandos de extracción de datos no es posible acceder al contenido devuelto, por lo que se recomienda limitar esta función al uso de comandos de inserción, creación y borrado.
- Valor devuelto: Devuelve el número de filas afectadas o (-1) en caso de error. Nótese que existen comandos SQL, como son *drop*, *create* o similares, en los que el valor devuelto será cero aunque la consulta se haya efectuado correctamente. Además de estos casos, una consulta puede devolver cero filas y aún así tratarse de una llamada correcta. DBExecQuery devuelve un error (-1) únicamente en caso de imposibilidad de ejecución del comando.
- Ejemplo de uso:

```
Real DBExecQuery("create table prueba ( campo1 VARCHAR(5),
                                        campo2 INT,
                                        fechado DATE)");
```

8.3.2. DBTable

- Qué hace: Ejecuta un comando en la base de datos activa. Al contrario que *DBExecQuery*, *DBTable* devuelve un conjunto con los datos leídos. Estos datos deben ser de tipo numérico, texto o fecha.
- Valor devuelto: Devuelve un conjunto con los datos leídos o un conjunto vacío en caso de error.
- Ejemplo de uso:

```
Set conjunto = DBTable("select dni, nombre, apellido from empleados");
```

8.3.3. DBMatrix

- Qué hace: Ejecuta un comando en la base de datos activa. Su funcionamiento es similar a *DBTable*, si bien *DBMatrix* devuelve los datos leídos en una matriz en vez de un conjunto, y además los datos de ese conjunto se encuentran limitados a campos de tipo numérico.

- Valor devuelto: Devuelve una matriz con los datos leídos o una matriz de tamaño cero en caso de error.
- Ejemplo de uso:

```
Matrix matriz = DBMatrix("select * from tabla");
```

8.3.4. DBSeries

- Qué hace: Ejecuta una consulta en la base de datos activa y devuelve un conjunto de series con los nombres dados y haciendo uso del fechado indicado. La consulta obligatoriamente debe devolver en el primer campo una fecha y a continuación un dato numérico, y las filas deben estar ordenadas por fecha.
- Valor devuelto: El conjunto de las series temporales generadas por la consulta.
- Ejemplo de uso:

```
// Devuelve las series de ingresos y gastos de los
// movimientos de una cuenta bancaria, tomando como
// valor por defecto cero:
Set seriesMovimientos = DBseries("select fecha, ingresos, gastos
                                  from movimientos
                                    where id_cuenta = 12345
                                    order by fecha",
                      Diario,
                      [[ingresos, gastos]],
                      [[ing, gast]],0);
```

8.3.5. DBSeriesTable

- Qué hace: De forma similar a la función DBSeries, ejecuta una consulta en la base de datos activa y devuelve un conjunto de conjuntos de series con los nombres dados y haciendo uso del fechado indicado. La consulta obligatoriamente debe devolver en el primer campo un prefijo para la serie, una fecha y a continuación un dato numérico. La consulta debe estar ordenada por los campos prefijo y fecha.
- Valor devuelto: Un conjunto bidimensional consistente en los conjuntos de series temporales generadas por la consulta a a base de datos.
- Ejemplo de uso:

```
// Devuelve el conjunto bidimensional de las series generadas por
//los ingresos y gastos de cada cuenta bancaria existente:
```

```
Set seriesMov = DBSeriesTable("select fecha, id_cuenta, ingresos, gastos
                               from movimientos
                               order by id_cuenta, fecha",
                              Diario,
                              [[ingresos, gastos]]);
```

8.3.6. DBSeriesColumn

- Qué hace: De forma similar a la función DBSeries, ejecuta una consulta en la base de datos activa, pero únicamente devuelve un conjunto de series con los nombres dados y haciendo uso del fechado indicado. La consulta obligatoriamente debe devolver en el primer campo el nombre de la serie, seguido por una fecha y a continuación un dato numérico. La consulta debe estar ordenada por los campos serie y fecha.
- Valor devuelto: El conjunto de las series temporales consultadas.
- Ejemplo de uso:

```
// Devuelve el conjunto de las series generadas por los
// ingresos y los gastos de cada cuenta bancaria existente:
Set seriesCol = DBSeriescolumn("select fecha, id_cuenta, ingresos, gastos
                               from movimientos
                               order by id_cuenta, fecha",
                               Diario,
                               [[ingresos, gastos]]);
```

8.3.7. DBCreateSeriesTable

- Qué hace: Crea una tabla en la base de datos activa con las series indicadas como contenido. La existencia de dicha tabla es independiente del funcionamiento de TOL, por lo que, si no se desea conservarla al terminar un programa, hay que borrarla explícitamente mediante la orden: "drop table tablaSeries".
- Valor devuelto:
- Ejemplo de uso:

```
// Guardamos las series devueltas por DBSeriesTable:
DBCreateSeriesTable(tablaSeries, seriesMov, "fecha","","",[[,,]]);
```

8.4. Manejo simultáneo de múltiples bases de datos

TOL permite el uso simultáneo de varias bases de datos de forma que es posible alternar su uso sin tener que abrirlas y cerrarlas entre operaciones.

8.4.1. DBActivate

Mediante el uso de la función *DBActivate* se puede hacer uso de una determinada base de datos mientras el resto siguen abiertas.

El prototipo de DBActivate es:

```
Real DBActivate (Text alias);
```

- Comportamiento de activación de las bases de datos:

Es importante conocer el comportamiento de estado de activación de las bases de datos abiertas, puesto que no siempre puede parecer evidente. Básicamente existen dos reglas básicas:

1. Después de abrir una base de datos ésta se activa automáticamente.
2. En caso de cierre de una base de datos activa, la primera de la lista de bases de datos abiertas será activada.

- Valor devuelto:

DBActivate devuelve 1 (Verdadero) en caso de éxito y 0 (Falso) en caso de error.

- Ejemplo de uso:

```
// Abrimos dos bases de datos:
Real DBopen("alias1", "usuario","clave",db1);
Real DBopen("alias2", "usuario","clave",db2);
// Operamos sobre la última abierta:
Set tabla2 = DBTable("select * from tabla");
// Activamos la primera:
Real DBActivate("alias1");
// Operamos sobre la primera:
Set tabla1 = DBTable("select * from tabla");
// Cerramos las dos bases de datos:
Real DBClose("alias1");
Real DBClose("alias2");
```

8.4.2. DBGetOpened

Mediante el uso de la función DBGetOpened es posible examinar el estado de todas las bases de datos abiertas, ya que devuelve un conjunto de estructuras de información sobre cada base de datos abierta, incluyendo alias usado en TOL, gestor, servidor y nombre de la base de datos.

Su prototipo es:

Set DBGetOpened(Text tipo);

- Valor devuelto:

DBGetOpened devuelve un conjunto bidimensional con la información de cada base de datos que se encuentre en condiciones de uso.

El parámetro "tipo" especifica el gestor cuya coincidencia filtrará el conjunto. Para consultar la lista completa, basta con dar una cadena vacía como argumento.

- Ejemplo de uso:

```
// Abrimos tres bases de datos:
Real DBOpen("alias1", "usuario","clave",db1);
Real DBOpen("alias2", "usuario","clave",db2);
Real DBOpen("alias3", "usuario","clave",db3);
// Leemos la información sobre bases de datos abiertas:
Set dataBases = DBGetOpened("");
// ¿Cuántas bases de datos hay?
Real numDataBases = Card(dataBases);
// ¿De qué tipo es la segunda?
Text type = dataBases[2][2];
// ¿La primera está activa?
Real active = dataBases[1][3];
// ¿Qué alias tiene la tercera?
Text alias = dataBases[3][1];
// Cerramos las bases de datos:
Real DBClose("alias1");
Real DBClose("alias2");
Real DBClose("alias3");
```