Proceso de limpieza de TOL previo al desarrollo de OISS

Víctor de Buen

9 de enero de 2006

Resumen

El propósito fundamental del presente proceso de limpieza de TOL es dejar una versión de TOL algo más despejada de elementos estériles y confusos así como prepararla para el desarrollo de sistemas OISS que le den una mayor agilidad y robustez a los proyectos desarrollados en TOL, de forma que dé tiempo al equipo de desarrollo a diseñar e implementar un nuevo lenguaje TOL sin bloquear por ello a los analistas de forma excesiva.

1. Objetivos

En términos del código C++, el objetivo principal era más que nada disponer los templates básicos adecuados para más adelante poder instanciar objetos TOL serializables de forma indexada y probablemente utilizando memoria externa, lo cual se consigue básicamente virtualizando los métodos principales de los objetos Name, Description, Contens, etc.; y esa ha sido la principal labor, afectando a la jerarquía de templates BRen*, BTmp*, BGraContens*, BGraObject* y en general a todas las clases derivadas de BSyntaxObject.

Sin embargo, aprovechando el tirón, se le ha dado un repasito general a TOL haciendo buena parte de esa limpieza de la que tanto habíamos hablado, eliminando campos superfluos e incorporándolos eventualmente en su lugar correspondiente en la jerarquía de clases.

A las clases BSet, BTimeSerie y derivadas, BSerieGroup y BSerieTable también se les ha dado un repaso importante pues son las que sustentan la mayor parte de las operaciones de TOL.

2. Resultados

Como efecto colateral se consigue que desaparezcan algunos errores en la recompilación, especialmente de funciones TOL de usuario, y una disminución de en torno a un 60 % de la memoria usada por TOL, desde un 20 % hasta más del 90 % dependiendo de los procsos, y un pequeño aumento de la velocidad que en modo alguno alguno justificarían el trabajo hecho, el cual cobra sentido sólo por el hecho de conseguir una jerarquía de clases más limpia y potente.

El principal problema de memoria se debe a que los objetos no se borran convenientemente a causa del lío que supone el tratamiento de las referencias vía BList y BSet, las cuales llevan bastante mala vida, pues se manipulan en exeso a veces, provocando interferencias entre ellas, además de las llamadas directas a InNRefs y DecNRefs. Se ha mejorado parcialmente esto devolviéndose ahora bastante más memoria que antes por lo que en algunas ocasiones, cuando se hacen procesos repetitivos muy pesados, el ahorro de memoria puede ser brutal.

Sin embargo, en el momento en el que se produzca cualquier pequeño desorden en los procesos de destrucción se empiezan a quedar objetos huérfanos de lista que les pueda ordenar destruirse y por eso pasa lo que pasa, aunque ahora mucho menos. Eso es realmente muy complicado de detectar sin tener casos de código TOL sencillo que reproduzcan y aislen los problemas por lo que no he dedicado apenas tiempo a ello aunque algo sí ha mejorado la situación. En cualquer caso es algo a lo que no merece la pena dedicarle tiempo a no ser que salte algo a la vista en cualquier momento.

Además, este problema quedará salvado, o al menos paliado en gran medida con el desarrollo de OISS, que está explicado en el documento OISS.pdf.

Aprovechando las ventajas del diseño modificado, o simplemente porque venía al paso, se han efectuado algunos otros pequeños cambios de menor alcance como por ejemplo:

- Los objetos de tipo autocontenido se declaraban internamente por referencia mediante la instanciación de un objeto BRenContens<Any>apuntando al objeto resultante de la parte izquierda del igual. Esta filosofía lazzy es muy útil cuando los objetos referenciados son muy grandes como pueden ser algunas matrices o conjuntos, pero al mismo tiempo obliga a mantener en memoria todo el árbol de objetos necesarios para su evaluación, lo cual puede ser que ocupe mucho más. Ahora se dispone de otra clase BCopyContens<Any>que copia el contenido del objeto y lo destruye en ese mismo momento liberando la memoria del árbol de evaluación de forma temprana. Este tipo de evaluación hungry se reserva a los objetos más bien pequeños que en principio serán Real, Complex, Date, Polyn, Ratio y Text; aunque habrá que estudiar cómo funciona mejor e incluso, ya para el nuevo TOL, se puede hacer parametrizable por el usuario si se observa que depende mucho de la naturaleza del proyecto.

- Ya no se pueden modificar los objetos constantes del sistema. Cuando se hacen asignaciones de constantes se crea una copia interna del objeto con BCopyContens<Any>de forma que no se produzcan los efectos colaterales impredictibles y muy peligrosos a que nos tiene acostumbrados TOL. En el siguiente ejemplo, XXX queda como [[1,2]] mientras que YYY y Empty permanecen vacíos y la última línea no tiene ningún efecto salvo dar un mensaje de error al usuario

```
Set XXX = Empty;
Set (XXX:=[[1,2]]);
Set YYY = Empty;
Set (Empty:=[[1,2,3]]);
```

- Ya se pueden crear estructuras locales. El problema era simplemente que la función FindLocal no estaba preparada para buscar por Mode sino sólo por nombre y encontraba primero la función asociada a la estructura que tiene tipo Code por lo que no se reconocía como un Struct. Lo único que hay que hacer es tomar la pila local y hacer la búsqueda ad-hoc del Struct.

- Los elementos de un conjunto S se pueden referenciar como S[x] donde x es un objeto de tipo Real o de tipo Text. Cuando x es una constante de texto necesariamente ha de comenzar por el carácter comilla doble <">, por lo que se puede hacer esa comprobación inicial para ahorrarse el intento de evaluación de x como Real, ahorrando un tiempo que puede llegar a ser importante en algunas circunstancias.

- Los objetos TOL de usuario tienen ahora Description y Expression como funciones separadas, pero sólo la descripción es modificable por el usuario mientras que la expresión la rellenará TOL sólo cuando UnparseNodes sea cierta. Cuando no hay descripción el usuario recibirá la expresión, si es que tiene. La función Identify devuelve el primer texto no vacío de la lista Name(), LocalName(), Expression(), Description().

- El nombre local sólo existe en los objetos referenciantes BCopyContens<Any>, BRenContens<Any>, BRenObject<Any>

- En la función Frecuency se estaba dividiendo en una parte de más que luego no se devolvía por lo que los análisis de frecuencias estaban incompletos.

- En la función ARMAAutoCovarianzeVector había un acceso inválido de memoria cuando no hay parte AR en un modelo y se solicita un número de autocovarianzas menor o igual que el grado MA, lo cual puede dar lugar a caídas o comportamientos extraños de forma aleatoria e irreproducible.

- En el AIA no se destruía la copia de la serie de residuos construida para no modificar la serie pasada como argumento. Esto no traía consecuencias en cuanto a las referencias pero sí que ocupaba el espacio correspondiente al área de datos duplicada que nunca se devolvía.
- Cada mensaje de error o de aviso llevará adjunto un número correlativo para que el usuario pueda encontrar más facilmente el primero de ellos y seguirles la pista más cómodamente.
- Se han creado las variables TOL Real NError y Real NWarning que devuelven el número de erores y avisos emitidos hasta el momento en la sesión, para que los usuarios puedan comprobar en tiempo de ejecución si un proceso está funcionando correctamente.
- Se han hecho algunos cambios a nivel interno en modo de depuración para facilitar la búsqueda de fallos de memoria creando una tabla en un fichero de texto con los eventos de creación y destrucción de objetos de forma que se pueda cargar en una base de datos para investigar lo que está ocurriendo cuando algo falla. Esto es tranasparente al usuario TOL y afecta sólo al desarrollador.
- También pensando en el desarrollador y en el usuario avanzado se ha mejorado la sintaxis y la ayuda en línea de los comandos de TOL Interactivo (tol.exe -d ...)
  - <tol expression>: ejecuta una expresión TOL
  - $HELP : muestra esta ayuda
  - $SYS <command>: ejecuta un comando del sistema operativo
  - $GRAMMAR : especifica la gramática por defecto
  - $DELETE : borra objetos TOL creados previamente
  - $DUMP : muestra un objeto TOL
  - $STATUS : muestra algunas estadísticas acerca del estado de TOL
  - $SHOW_CLASS_SIZES : muestra el tamaño de algunas clases C++ internas
  - $END : sale del modo interactivo de TOL

2.1. Robustez

De momento se han probado a través de la VPN procesos, principalmente de tipo pestimation.tol, de los SADD's almacenados en el CVS de Bayes, de proyectos bastante variados

1. Banco do Brasil
   a) Test_BB_000_InitSADD: Sólo carga el SADD del proyecto y la lista de agencias desde la base de datos
   b) Test_BB_001_ARIMATFEst: Modelo ARIMATF de la serie de Variación de efectivo de una agencia
   c) Test_BB_002_ARIMAIdentif: Identificación de modelos ARIMA más probables para la serie de Variación de Efectivo de una agencia
   d) Test_BB_003_ARIMASearch: Estimación de los modelos ARIMA alternativos más probables para la serie de Variación de Efectivo de una agencia
   e) Test_BB_004_VARMA: Modelo modelo VARMA aproximado linealmente de la serie de Variación de efectivo de un grupo de agencias
   f) Test_BB_005_SAM: Simulación del Saldo Autónomo Mínimo
2. Cocacola
   a) Estimación ARIMA de un producto

b) Estimación bayesiana

3. Voz de Galicia

   a) Estimación de una serie global
   b) Estimación de un punto de venta
   c) Previsión

4. Feda

   a) Combinación de previsiones: Los resultados son idénticos pero utiliza 50 MB de RAM en la versión nueva y 80 MB en la vieja y tarda 369 segundos en lugar de 690, con lo que los ahorros son del 47 % en tiempo y del 38 % en memoria.

Los resultados de la versión de desarrollo de limpieza DEV-VERSION son idénticos a los de la versión CVS-20051220 que se ha tomado como referencia en todos los casos.

Sería necesario probar otros tipos de procesos, tanto de estimación de todos los tipos, (VARMA, bayesianos, jerárquicos, ...), como de procesos que no sean de modelización para estar seguros de que no hay errores de importancia antes de subir los cambios al CVS.

Yo veo en los proyectos que hay muchos programas TOL que podría probar, pero no me atrevo para no fastidiar, por lo que es necesario que desde los proyectos se seleccione una colección de procesos lo más variados que sea posible e intentando que no sean demasiado pesados de forma gratuita, para poder hacer muchas pruebas en el menor tiempo posible.

Con esta colección de procesos se creará una batería de tests que nos servirá más adelante para probar de forma sistemática nuevas versiones de TOL, o incluso el nuevo desarrollo de TOL, adaptando el código si es necesario, pero con el mayor grado de compatibilidad que sea razonable conseguir.

2.2. Eficiencia

Tomando como caso de comparación la estimación de modelos de agencia del Banco do Brasil se han obtenido los siguientes resultados.

Se ha utilizado el tol.exe y se ha separado el proceso en módulos correspondientes a los tests del Banco do Brasil por ser los que se han podido hacer en local y en los que tiene sentido medir el tiempo de proceso, que está dominado por la interconexión en el caso de acceder de forma remota por la VPN al código TOL y/o la base de datos. Las mediciones abarcan

1. Cores: Cualquier instancia de cualquier clase heredada de BCore, lo cual incluye todos los elementos del lenguaje, las listas, los árboles y otros heredados de BObject que carecen normalmente de importancia relativa. Estos han descendido notablemente al no heredar BText de BAtom y emplearse muchas menos listas que antes ya que BSet ya no contiene una lista, ni tampoco BSerieGroup ni BSerieTable.

2. Objects: Objetos heredados de BSyntaxObject, es decir, los propiamente relacionados con el lenguaje. Estos han descendido porque se borran los objetos de los tipos pequeños con el operdor de declaración y sobre todo porque hay menos errores que dejen referencias de más y se borran más objetos innecesarios.

3. Kb: Memoria adjudicada al proceso por el sistema operativo Windows una vez ejecutado el test restándole la correspondiente a la inicialización del proyecto SADD (Test BB000 InitSADD) para calcular el consumo debido exclusivamente a cada test. Como estos se ejecutan en una pila local (o sea entre llaves {}), y devolviendo copia de los resultados para la estadística de tests, TOL debería devolver prácticamente toda la memoria. Aunque tiene cierto componente aleatorio pues el S.O. puede otorgar más de lo solicitado, esto no supone demasiada variación. El ahorro depende mucho del tipo de proceso que ejecuta el test ya que el TOL del CVS puede perder más o menos memoria según las operaciones que se realizan.

Figure 1: TOL performance

Number of Cores				
test name	**CVS**	**DEV**	**Saving**	**Saving %**
test_000_InitSADD	157	31	126	80.25%
test_001_ARIMATFEst	49,540	8,677	40,863	82.48%
test_002_ARIMAIdentif	114,659	3,958	110,701	96.55%
test_003_ARIMASearch	117,429	23,206	94,223	80.24%
test_004_VARMA_	180,504	7,832	172,672	95.66%
test_005_SAM	10,371	3,092	7,279	70.19%
Total general	***472,660***	***46,796***	***425,864***	***90.10%***
Number of Objects				
test name	**CVS**	**DEV**	**Saving**	**Saving %**
test_000_InitSADD	26	14	12	46.15%
test_001_ARIMATFEst	6,689	4,392	2,297	34.34%
test_002_ARIMAIdentif	15,398	1,046	14,352	93.21%
test_003_ARIMASearch	14,132	7,413	6,719	47.54%
test_004_VARMA_	27,708	2,569	25,139	90.73%
test_005_SAM	1,102	703	399	36.21%
Total general	***65,055***	***16,137***	***48,918***	***75.19%***
Non Freed Memory (KBytes)				
test name	**CVS**	**DEV**	**Saving**	**Saving %**
test_000_InitSADD				
test_001_ARIMATFEst	18,328	4,632	13,696	74.73%
test_002_ARIMAIdentif	4,120	310	3,810	92.48%
test_003_ARIMASearch	90,724	16,072	74,652	82.28%
test_004_VARMA_	35,312	24,520	10,792	30.56%
test_005_SAM	2,760	2,200	560	20.29%
Total general	***151,244***	***47,734***	***103,510***	***68.44%***
Time Elapsed (Seconds)				
test name	**CVS**	**DEV**	**Saving**	**Saving %**
test_000_InitSADD	0.000	0.000	0.000	0.00%
test_001_ARIMATFEst	11.672	8.453	3.219	27.58%
test_002_ARIMAIdentif	22.391	15.282	7.109	31.75%
test_003_ARIMASearch	69.391	64.719	4.672	6.73%
test_004_VARMA_	12.313	12.750	-0.437	-3.55%
test_005_SAM	103.563	71.687	31.876	30.78%
Total general	***219***	***173***	***46***	***21.17%***

4. Time Elapsed: Tiempo empleado por cada proceso medido en segundos. En algunos procesos no se mejora prácticamente nada pero en otros se gana un 30 % en velocidad y en otros que se han probado aparte mucho más, especialmente cuando la memoria consumida desciende mucho y se evita el uso de memoria virtual.

3. Resumen de los cambios introducidos en el código

Los cambios afectan a 87 de los 267 ficheros *.c, *.cpp; *. h; *.hpp que componen el proyecto TOL y afectan a las áreas más importantes del lenguaje aunque también hay algunos cambios menores que se han hecho necesarios o que venían al paso.

A continuación se detallan por categorías conceptuales los cambios realizados seguidos de los ficheros afectados por cada uno o a veces de forma conjunta para un grupo de cambios consecutivos íntimamente relacionados entre sí.

(Aún faltan de reportar los últimos cambios, para ello tengo que hacer un diff en local del TOL del CVS actual con la versión de desarrollo)

3.1. Basic objects related changes

1. Virtual methods IsDeadObj and System are been downloaded from BSyntaxObject to this level to allow a best memory tracing

   a) bbasic/tol_batom.h

2. Changed memory trace handling

   a) bbasic/atm.cpp

3. Constant factor 2 for Add reallocation replaced by ReallocMarginFactor defined as 1.2

   a) bbasic/tol_barray.h :

4. BText::defectSize_ = 0 instead of old unused value of 64, due to constant 1 was used

   a) bbasic/txt.cpp :

5. Now BText doesn't inherites from BAtom. To insert strings in lists BObject will be used.

   a) bbasic/tol_btext.h
   b) bbasic/txt.cpp
   c) bbasic/tol_bdir.h
   d) bbasic/dir.cpp
   e) bbasic/dirtext.cpp
   f) bbasic/tol_bopt.h
   g) bbasic/opt.cpp
   h) bbasic/opttext.cpp
   i) bparser/par.cpp

6. New access functions to miliseconds and string formated of process elapsed timebbasic/tol_btimer.h

   a) bbasic/timer.cpp

3.2. Math objects related changes

1. Frequency(...) must divide in just n parts instead of n+1

   a) bmath/bstat/stat.cpp

2. Fixed invalid memory access in ARMAAutoCovarianzeVector function, when no AR is defined and number of required autocovariances is less or equal than MA degree that could cause random secondary effects, even system crashes.

   a) bmath/bstat/barma.cpp

3.3. Kernel-grammar related changes

1. BField inherites from BObject instead of BSyntaxObject

   a) btol/bgrammar/tol_bstruct.h

2. New IsAutoContens() method of grammars to be used in futurebtol/bgrammar/str.cpp :

   a) btol/bgrammar/tol_bgrammar.h
   b) btol/bgrammar/graacc.cpp

3. Unparsing is used just in Evaluate(BText expression) instead of Evaluate(BTree tree) to savememory and process time.

   a) btol/bgrammar/graimp.cpp

3.4. Syntax-hierarchy related changes

1. Unused class BSyntaxImage and related functions has been eliminated

   a) btol/bgrammar/tol_bsyntax.h
   b) btol/bgrammar/syn.cpp
   c) btol/bgrammar/tol_bgrammar.h
   d) btol/bgrammar/graacc.cpp
   e) btol/bgrammar/tol_bmethod.h
   f) btol/bgrammar/met.cpp

2. Boolean members system_, released_, calculated_, and calculating_ are compacted in a protected bit-struct flags_ member at BSyntaxObject using just a byte all together

   a) btol/bgrammar/tol_bsyntax.h
   b) btol/bgrammar/tol_bgencon.h
   c) btol/bgrammar/tol_bgenobj.h
   d) btol/lang/language.cpp

3. BSyntaxObject Mode constants are moved to tol_bsyntax.h

   a) tol_bcommon.h,
   b) btol/bgrammar/tol_bsyntax.h
   c) btol/bgrammar/syn.cpp

4. BSyntaxObject::flags_.system_ is set to true for all BSyntaxObjects until TOLHasBeenInitialized() return true, when InitGrammars ends

   a) btol/bgrammar/tol_bsyntax.h

   b) lang/language.cpp

5. Old BSyntaxObject tolPath_ is restricted to BSetFromFile to be used in new method ParentPath() to access to path of the file where the object was created in. New BSynObjOptInfo class to optional storement of name_, description_, localName_ and parent_ in BSyntaxObject just when needed. Changed memory trace handling

   a) btol/bgrammar/tol_bsyntax.h

   b) btol/bgrammar/syn.cpp

6. Class BRenamed is eliminated, BRenamedGen is renamed as BRefObject. In BTmpObject args_ member is now a BList* instead of a BSet and a short card_, and BSyntaxObject** array_ are used to accelerate accesses to arguments.

7. New template classes has been inserted in *Contens* template hierarchy. A new template BGraContensBase<Any>without contens_ to save memory when is this member is not needed and allowing virtual access to it. A

8. New template BGraContensFND<Any,Name_,Description_>usefull to returned objects by built-in functions saving memory in each instance.

9. A new temlpate BCopyContens<Any>to use instead of BRenContens<Any>to low size types as Real, Complex, Date, Text, Polyn, Ration. This objects copy the contens of referenced object and try to destroy it. If isn't posible, destroying will be delayed to the live end of encapsulating object.

10. BGraConstant<Any>modifications of name, description or value are disabled

11. BRenContens<Any>::New(const BText& name, BSyntaxObject* obj) method will call BCopyContens<Any>::New when obj were a System variable to avoid involuntary constant objects modifications

    a) btol/bgrammar/tol_bgentmp.h

    b) btol/bgrammar/tol_bgencon.h

    c) btol/bgrammar/tol_bgenobj.h

    d) lang/language.cpp

12. Access to BSet BTmpObject<Any>::Args() must be replaced by BList* ArgList() or DupArgList()

    a) btol\matrix_type\matgra.cpp

    b) btol\set_type\setgra.cpp

    c) btol\bgrammar\tol_bgentmp.h

13. Access to BSet BTmpObject<Any>::Args().Card() must be replaced by NumArgs()

    a) btol\bmonte\mc-integrator.cpp

    b) btol\matrix_type\matgra.cpp

    c) btol\real_type\armseval.cpp

    d) btol\real_type\datgrapr.cpp

e) btol\real_type\datgrast.cpp
f) btol\real_type\datgrav.cpp
g) btol\set_type\setgra.cpp
h) btol\timeset_type\tmsgrav.cpp
i) btol\bgrammar\tol_bgentmp.h:
j) lang\language.cpp

14. Access to BSyntaxObject or inherited boolean members system_, released_, calculated_, and calculating_ must be changed to flags_.* or BSyntaxObject::Put* functions

    a) btol\bmonte\mc-integrator.cpp
    b) btol\cseries_type\ctsrgra.cpp
    c) btol\matrix_type\multimin.cpp
    d) btol\serie_type\tsrgra.cpp
    e) btol\set_type\setgra.cpp
    f) btol\bgrammar\tol_bgencon.h
    g) btol\bgrammar\tol_bgenobj.h
    h) btol\bgrammar\tol_bsyntax.h
    i) btol\set_type\tol_bsetgra.h:
    j) lang\language.cpp

15. BSystem* objects that shouldn't be system constants are changed to BUser* to inherite from BGraContens<Any>instead of undeletable system class BGraConstant<Any>

    a) btol\matrix_type\matgra.cpp
    b) btol\real_type\datgra.cpp
    c) btol\real_type\datgsrst.cpp
    d) btol\text_type\txtgra.cpp
    e) lang\language.cpp

3.5. TOL Functions and Code-type related changes

1. New BOperator::AddSystemOperator() to control different creating processes of built-in and user functions that were never deleted internally

2. New bool BuildingMethod() function to special treatment of overloading of method's functions

    a) btol/bgrammar/tol_boper.h
    b) btol/bgrammar/opr.cpp

3. Fixed forbidden accesses to code->Contens() for non BUSerFunction objects

4. When BuildingMethod() is false Mode() must be equal to so.Mode() to match

    a) btol/bgrammar/txthash.cpp :
    b) btol/bgrammar/met.cpp :

5. Internal BCode::operator_ is changed from BUserFunction* to BStandardOperator*

6. btol/code_type/tol_bcodgra.h :
7. BUserFunCode::Mode() is changed to BOBJECTMODE instead of BUSERFUNMODE that is reserved to internal operators and unused class BCodeCreator has been eliminated
   a) btol/code_type/tol_bcode.h
   b) btol/code_type/cod.cpp

3.6. Set-type related changes

1. At BSet class BList* element_ is eliminated.
2. When a list is needed a copy is available through ToList().
3. Access to BSet BTmpObject<Any>::Args() must be replaced by BList* ArgList() or DupArgList(). New class BGraContensFNDSet<BSet, Name_, Description_>inherited from BGraContensFND<Any,Name_,Description_>to allow special constructors of bulit-in functions returned sets.
4. BSet constructor from BList* and operator= to BList* are eliminated to replace in this case by PutElement or RobElement.
5. Allocation methods: DeleteBuffer() method is renamed as Delete(). ReallocBuffer is renamed as Realloc and protected and is added a new protected Alloc method and the only one public will be PrepareStore, in order to avoid invalid memory accesses.
   a) btol\bdb\bdb.cpp
   b) btol\bdb\bdspool.cpp
   c) btol\bgrammar\graimp.cpp
   d) btol\bgrammar\spfuninst.cpp
   e) btol\bmodel\aia.cpp
   f) btol\bmodel\estim.cpp
   g) btol\bmodel\foread.cpp
   h) btol\bmodel\modcalc.cpp
   i) btol\bmodel\model.cpp
   j) btol\bmonte\mc-integrator.cpp
   k) btol\matrix_type\matgra.cpp
   l) btol\matrix_type\multimin.cpp
   m) btol\polynomial_type\polgra.cpp
   n) btol\real_type\datgrast.cpp
   ñ) btol\set_type\set.cpp
   o) btol\set_type\setgra.cpp
   p) btol\set_type\tol_bset.h
   q) btol\text_type\txtgra.cpp
   r) lang\language.cpp

3.7. Real-type related changes

1. New controls about number of errors and warnings exported to TOL user
   a) btol/real_type/datgra.cpp
   b) bbasic/out.cpp

3.8. Complex-type related changes

1. BCmpConstant* i_ is changed to BParamCmp

   a) btol/complex_type/cmpgra.cpp

3.9. Serie-type related changes

1. Dating_->nRefs_ is not touched by BTimeSerie because is non needed feature. Unused hashDat_ member has been eliminated

   a) btol/serie_type/tol_btmser.h

   b) btol/serie_type/tsrgra.cpp

2. BData data_ is now really protected eliminating GetDataBuffer() access. New BTimeSerie member functions PutBufDat to change a dat, or Realloc, Alloc and Delete to manage memory

   a) btol\bdb\bdb.cpp

   b) btol\bdb\bdspool.cpp

   c) btol\bmodel\modprev.cpp

   d) btol\serie_type\srg.cpp

   e) btol\serie_type\tsr.cpp:

   f) btol\serie_type\tsrgra.cpp:

   g) btol\serie_type\tsrgrav.cpp:

   h) btol\serie_type\tol_btmser.h:

   i) btol\serie_type\tol_btsrgrp.h

   j) btol\set_type\setgra.cpp

3. New BData class inherited from BArray<BDat>to control Time Series used volume

   a) bmath/mathobjects/tol_bdat.h

   b) bmath/mathobjects/dat.cpp

4. BSerieGroup and BSerieTable use now a BArray<BUserTimeSerie*>instead of a BSet to store Series in order to avoid secondary effects of references over them.

5. Now the only one creator for BSerieGroup and BSerieTable has no arguments. Then you can add a Serie, a BList*, a BSet or another BSerieGroup or BSerieTable. After you can use BSerieGroup to prepare operations between series or test dating and date bounds of a collection of series or you can fill a BSerieTable with data from collected series.

   a) btol\bdb\bdspool.cpp

   b) btol\bmodel\estim.cpp

   c) btol\bmodel\modcalc.cpp

   d) btol\bmodel\tol_bmodel.h

   e) btol\matrix_type\matgra.cpp

   f) btol\real_type\datgra.cpp

   g) btol\real_type\datgsrst.cpp

   h) btol\serie_type\srg.cpp

i) btol\serie_type\tol_btsrgrp.h

j) btol\serie_type\tsrgrai.cpp

k) btol\text_type\txtgra.cpp

l) lang\language.cpp

6. BList* outLst_ and inLst_ are eliminated from BModel to apply new BSerieTable advantages

   a) btol/bmodel/foread.cpp

   b) btol/bmodel/modcalc.cpp

   c) btol/bmodel/model.cpp

   d) btol/bmodel/tol_bmodel.h

3.10. Console related changes

1. Revisited commands and user help for interactive console mode of tol.exe

   a) lang/language.cpp