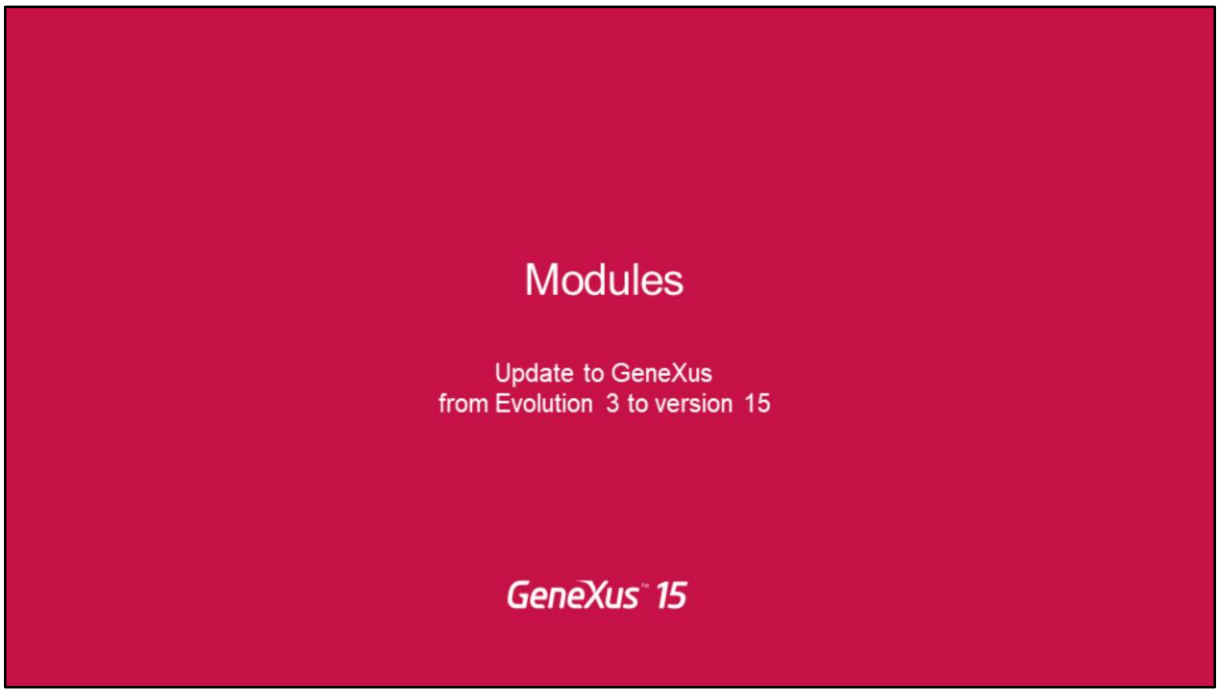
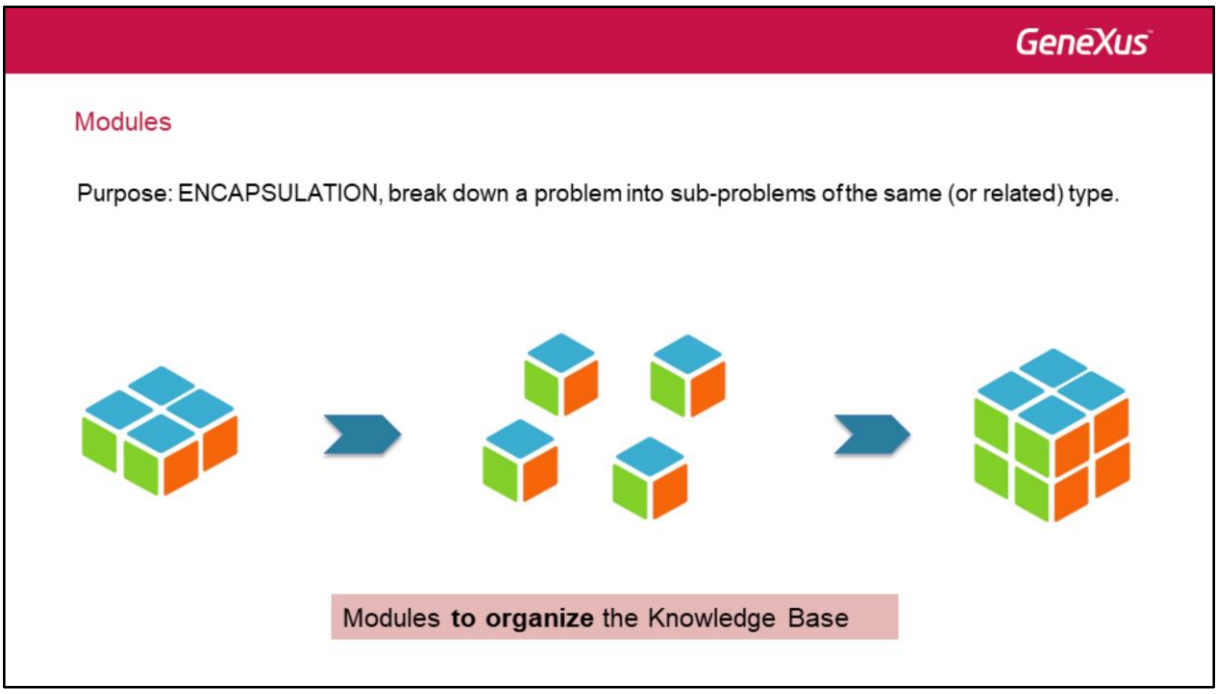


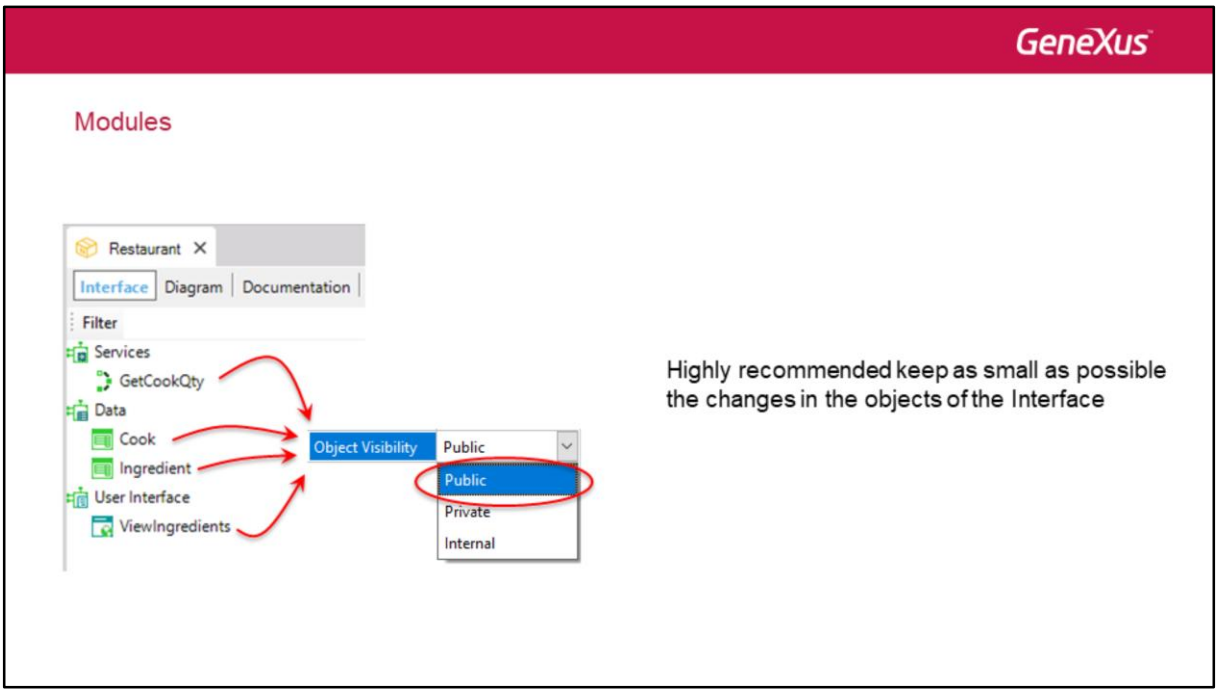
Material elaborado de acuerdo al Upgrade10 de GeneXus 15.

Empecemos por revisar lo que ya sabíamos. Recordemos que los módulos aparecen en GeneXus X Evolution 3.



Los módulos surgieron para hacer más fácil el proceso de diseñar aplicaciones, posibilitando **organizar la KB** en diferentes secciones –objetos módulo- que se concentran en proveer una serie reducida de servicios especializados. Todas esas secciones cooperan para proveer un servicio más complejo. Esto permite que los desarrolladores se concentren en desarrollar funcionalidades sin la necesidad de saber sobre el resto de la aplicación.

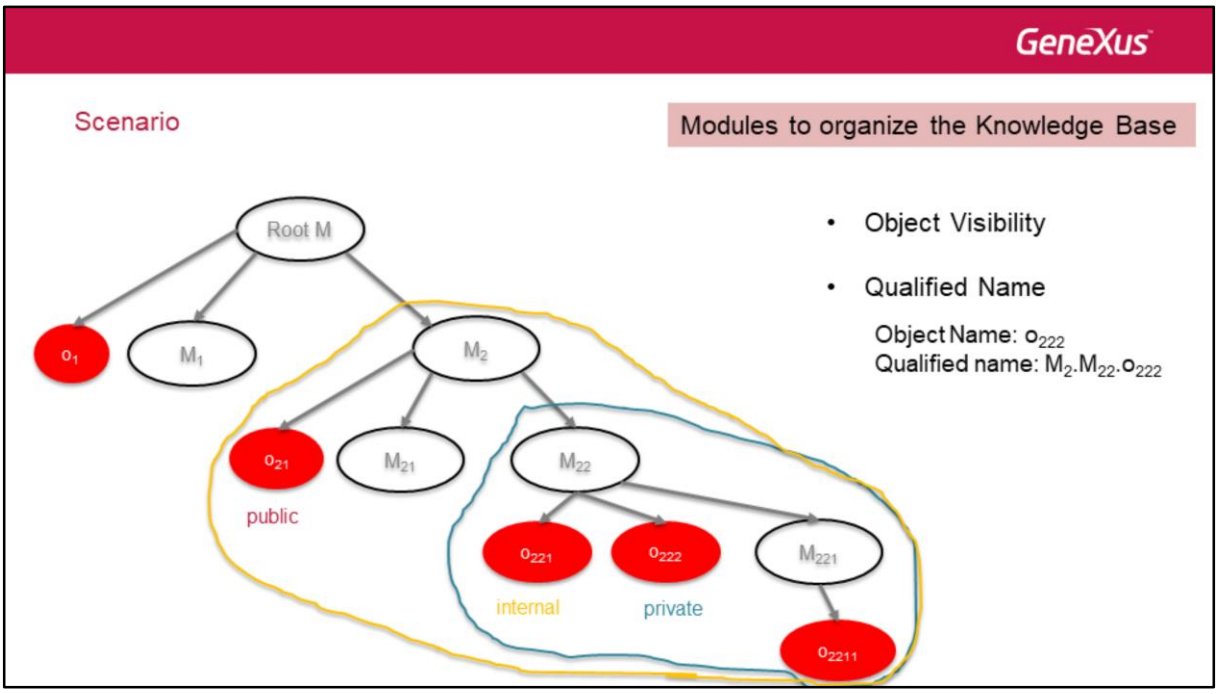
Por tanto, el objeto módulo permite partir el problema en sub-problemas del mismo (relativo) tipo. Estos sub-problemas son suficientemente simples como para permitir una solución directa, que es luego combinada con las soluciones de los otros sub-problemas utilizando las Interfaces definidas para encontrar una solución para el problema original.



La interfaz de un módulo consistirá de aquellas funcionalidades que se ofrecen para afuera, es decir, las que otros podrán ejecutar.

Aquí tenemos un ejemplo de un módulo Restaurant. En su interfaz se encuentran los objetos que vemos. De todos los objetos que estén dentro del módulo, aquellos que tengan las propiedad Object Visibility con el valor “Public” serán los que aparecerán formando parte de la Interfaz.

Se recomienda al diseñar la interfaz de un módulo concentrarse en pensar bien y definir esos objetos públicos para intentar que no cambien demasiado de modo de no afectar a otros módulos que los utilizan.



Recordemos que un objeto **private** sólo puede ser accedido por objetos del mismo módulo o por objetos de módulos hijos de éste, pero no por objetos que estén “más arriba” en la jerarquía o desciendan de módulos que están más arriba.

En el ejemplo, los únicos objetos que pueden invocar al objeto o_{222} privado son el o_{221} y el o_{2211}.

Los objetos **internos** se agregaron en esta versión 15 para que puedan ser utilizados entre objetos de una rama del Root Module. Es decir, un objeto que tenga la propiedad Object Visitility en **Internal** podrá ser accedido por todo objeto con el que comparta un módulo raíz que no sea el Root Module sino algún descendiente de éste.

En el ejemplo, o_{21} puede acceder (y no porque sea público, en esto lo mismo da su visibilidad) al objeto o_{221}. Si o_{221} fuera private, no podría.

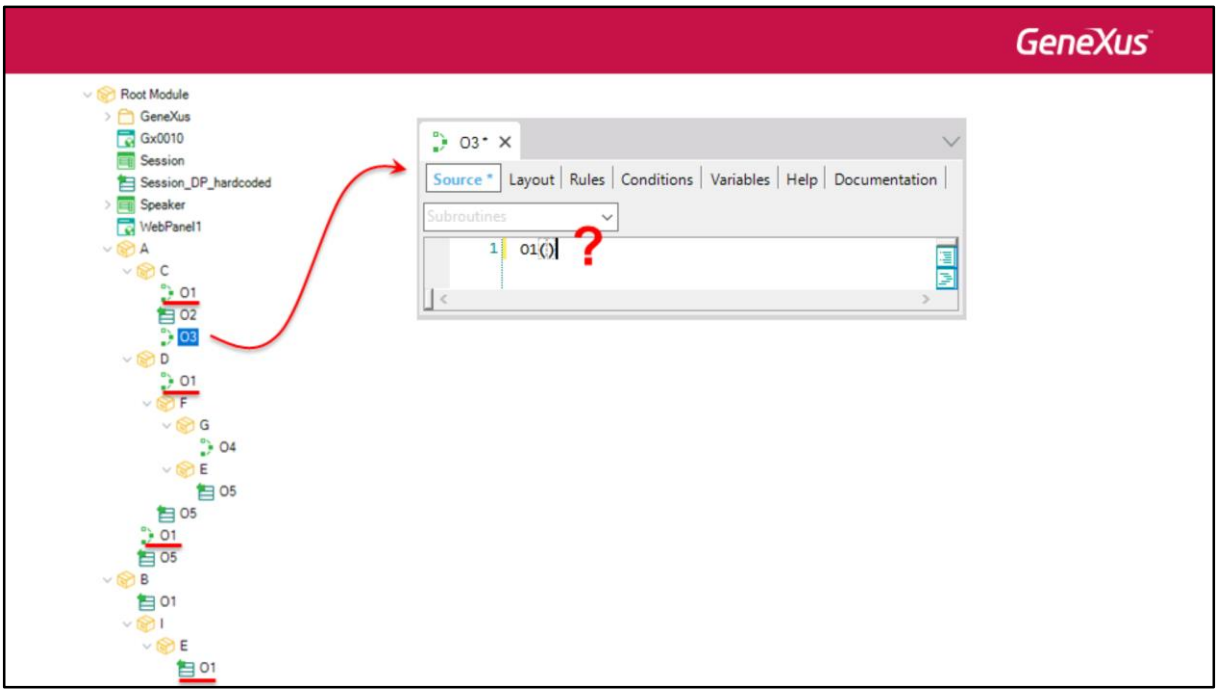
A los objetos públicos puede acceder cualquier objeto de la KB. Recordemos que los objetos públicos son los que compondrán su interfaz. Por tanto, si bien o_{221} no será visible en la interfaz del módulo M_{22}, de todas maneras un desarrollador que esté trabajando en un objeto del módulo M_2 o de M_{21} podrá invocar al objeto o_{221}.

Por otro lado recordemos que en distintos módulos podemos repetir el nombre de objeto, dado que a partir de que los objetos se salvan dentro de un módulo, su nombre queda calificado automáticamente con el path. Los objetos del módulo Root son los únicos para los que el Qualified name es el propio name.

Esto provoca un problema cuando se trata de transacciones paralelas en distintos módulos, ya que si bien las transacciones pueden pertenecer a módulos diferentes, las tablas son físicamente únicas, así como los atributos. Aunque a partir del upgrade 10 aparezcan entre las propiedades de cada tabla una Module/Folder que indica el módulo al que pertenece y una Object Visibility; ambas readonly. Esto se implementó para empezar a buscar una solución en la que las tablas puedan también ser propias del módulo. En un momento presentaremos este caso, y su complejidad, que está aún en etapa de investigación.

No es obligatorio cuando se invoca a un objeto de un módulo brindar el qualified name. GeneXus

tiene algoritmos para saber a qué objeto se está haciendo referencia cuando hay más de uno con el mismo nombre. Sin embargo, habrá casos de ambigüedad donde GeneXus no puede resolver a cuál de los objetos que se llaman de esa manera se está queriendo invocar. Y nos lo advierte. Allí habrá que realizar la invocación con el qualified name (completo o parcial, lo suficiente para desambiguar). Calificar el nombre de objeto que se está invocando no nos impide mover el objeto de lugar sin tener que preocuparnos de modificar las invocaciones de forma acorde. GeneXus lo hace automáticamente por nosotros. Califica el objeto invocado con el nuevo path.
El único problema que puede surgir es que
Aquí en nuestro wiki encontrará la información acerca de cómo se resuelven las invocaciones parcialmente qualificadas:
https://wiki.genexus.com/commwiki/servlet/wiki?22438,How%20are%20partially%20qualified%20object%20names%20resolved

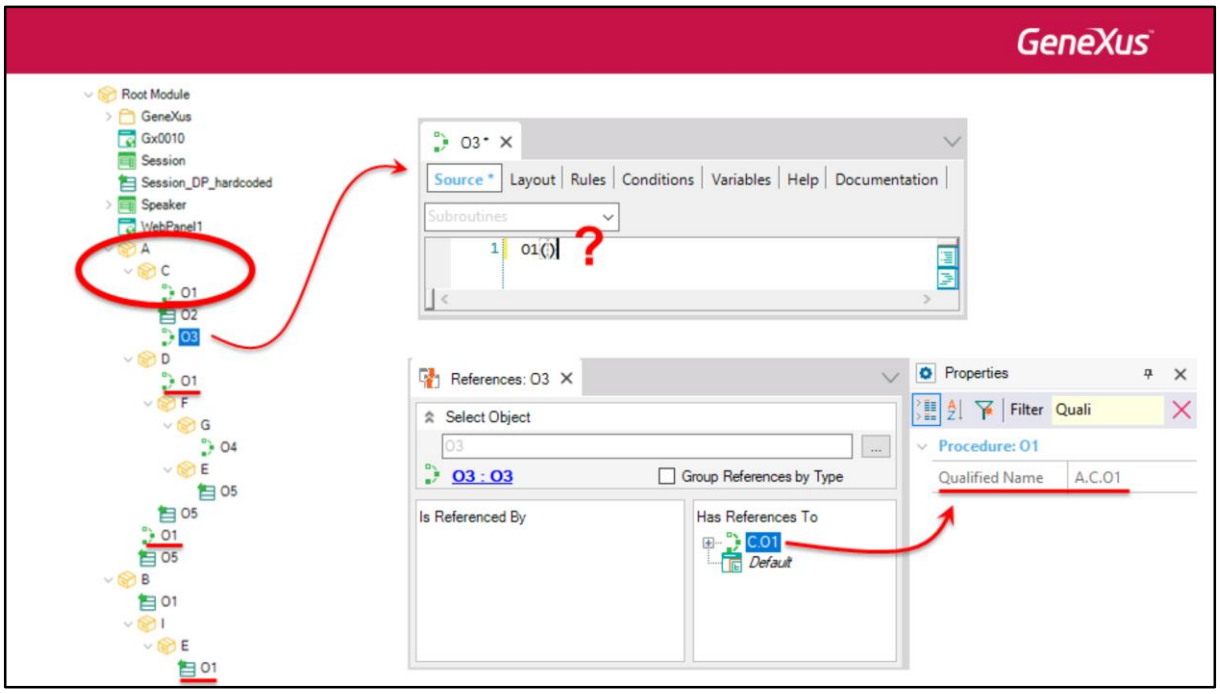


No es obligatorio cuando se invoca a un objeto de un módulo brindar el qualified name. GeneXus tiene algoritmos para saber a qué objeto se está haciendo referencia cuando hay más de uno con el mismo nombre.

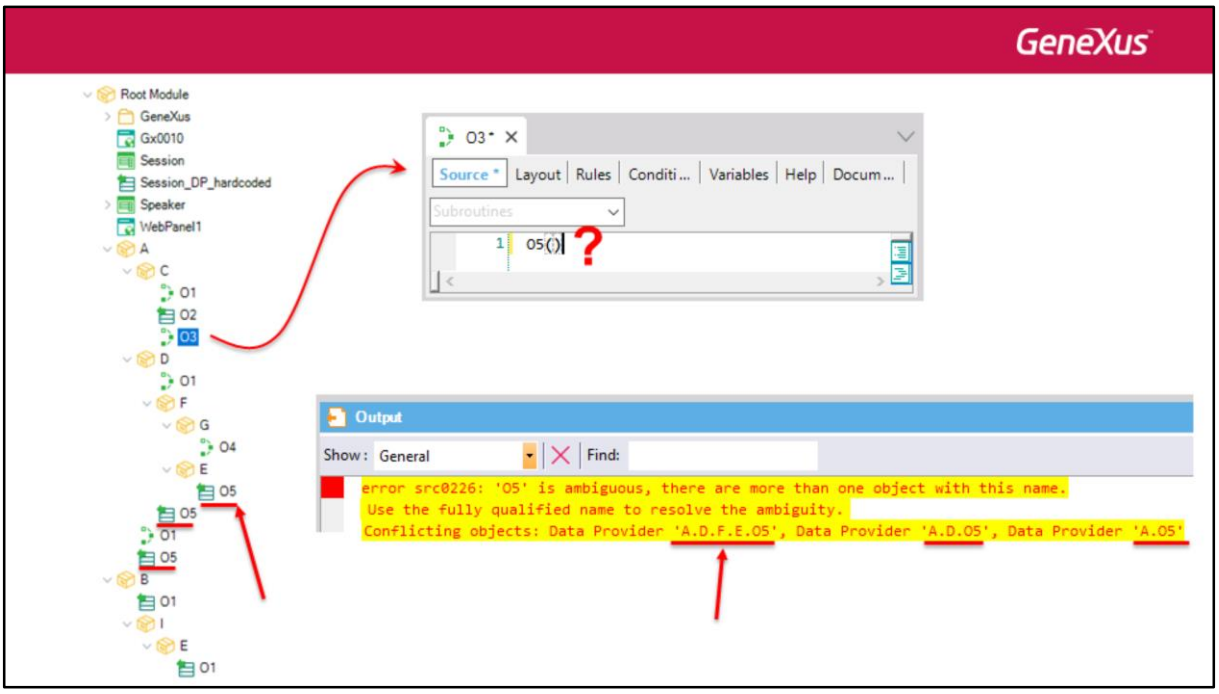
Por ejemplo, teniendo esta estructura de módulos, si desde el procedimiento O3 del módulo C, invocamos al objeto O1, ¿a cuál de todos los O1 se estará invocando?

[En el otro esquemita que puede ser más claro, si desde aquí llamamos a O1, ¿es a este, a este, a este, a este, o a este otro? Seguro es a este al que queríamos llamar. Veámoslo en GeneXus...]

Nota al instructor: puede saltearse esta parte sobre name qualification si lo desea. Es simplemente un repaso, pero puede que los alumnos pidan verlo.



Si pedimos ver las References del objeto O3, veremos que GeneXus eligió invocar al O1 hermano del objeto O3 que realiza la invocación. Esto se lo determinó el algoritmo del que hablábamos.



Sin embargo, habrá casos de ambigüedad donde GeneXus no puede resolver a cuál de los objetos que se llaman de esa manera se está queriendo invocar. Y nos lo advierte.
Por ejemplo, si desde O3 ahora queremos invocar a O5...

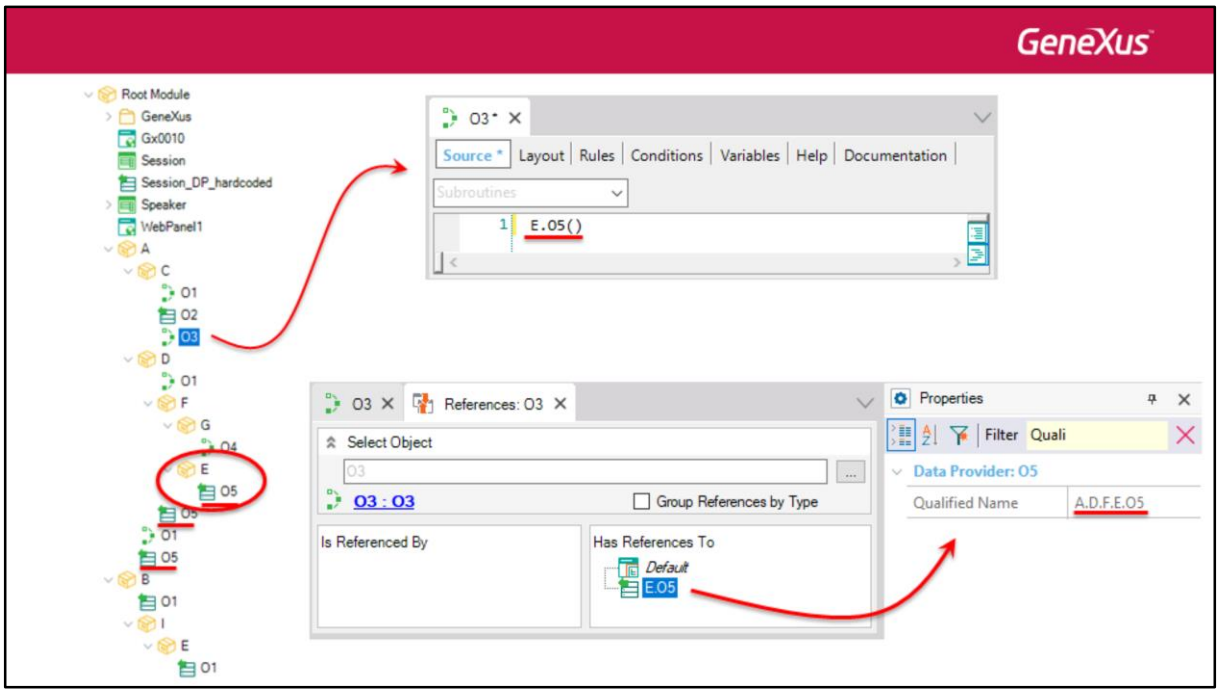
[en nuestro esquema...]

Al intentar grabar nos arrojará el error que vemos, indicándonos cuáles son los objetos en conflicto.

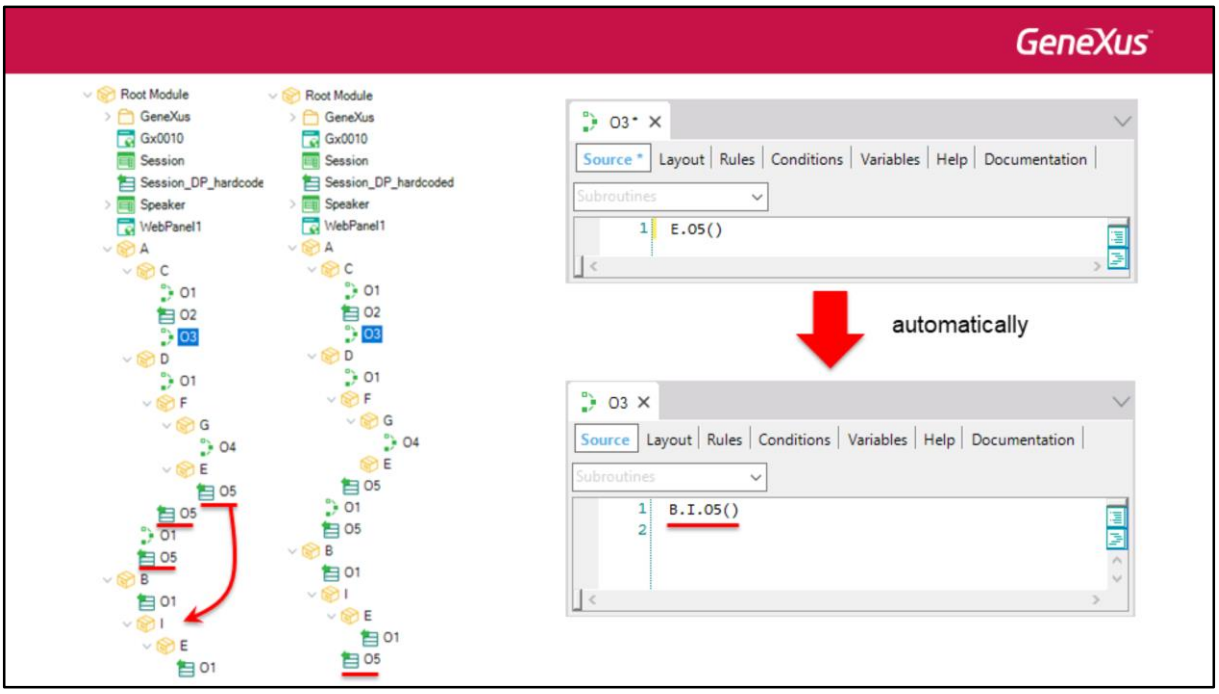
Allí habrá que realizar la invocación con el qualified name (completo o parcial, lo suficiente para desambiguar).

Aquí en nuestro wiki encontrará la información acerca de cómo se resuelven las invocaciones parcialmente qualificadas:
https://wiki.genexus.com/commwiki/servlet/wiki?22438,How%20are%20partially%20qualified%20object%20names%20resolved

Por ejemplo, si al que quisiéramos invocar fuera el Data provider A.D.F.E.O5, alcanzaría con modificar la invocación...

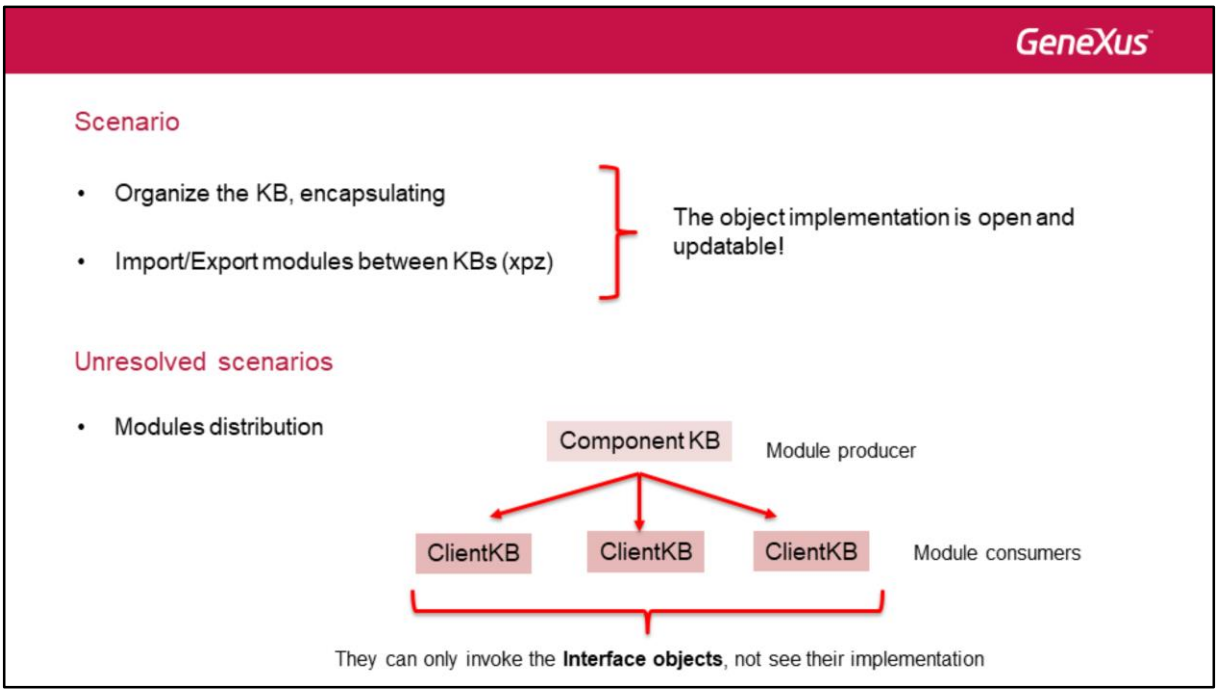


... de esta manera. Vemos que calificamos parcialmente al objeto. Si observamos las References, vemos que el Qualified Name corresponde al objeto que queríamos.



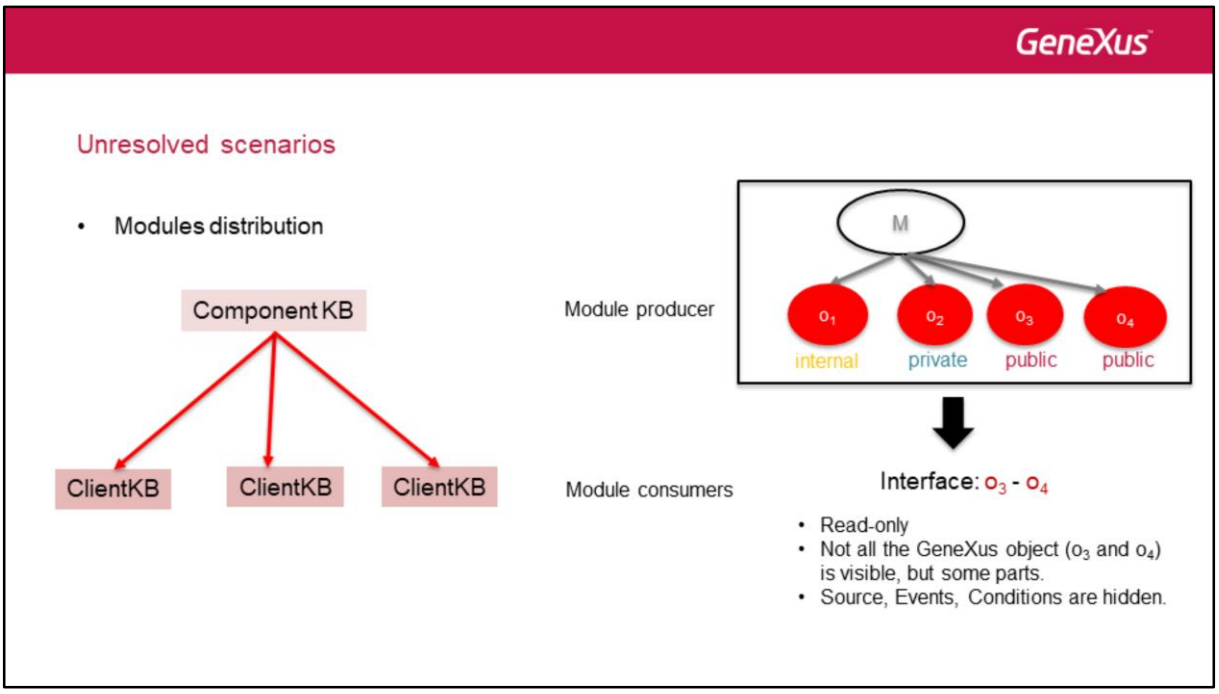
Calificar el nombre de objeto que se está invocando no nos impide mover el objeto de lugar sin tener que preocuparnos de modificar las invocaciones de forma acorde. GeneXus lo hace automáticamente por nosotros. Califica el objeto invocado con el nuevo path.

Así, si en el ejemplo movemos el objeto O5 que estábamos invocando desde O3 al módulo I, veremos que GeneXus cambia automáticamente la invocación, con el qualified name que corresponde.

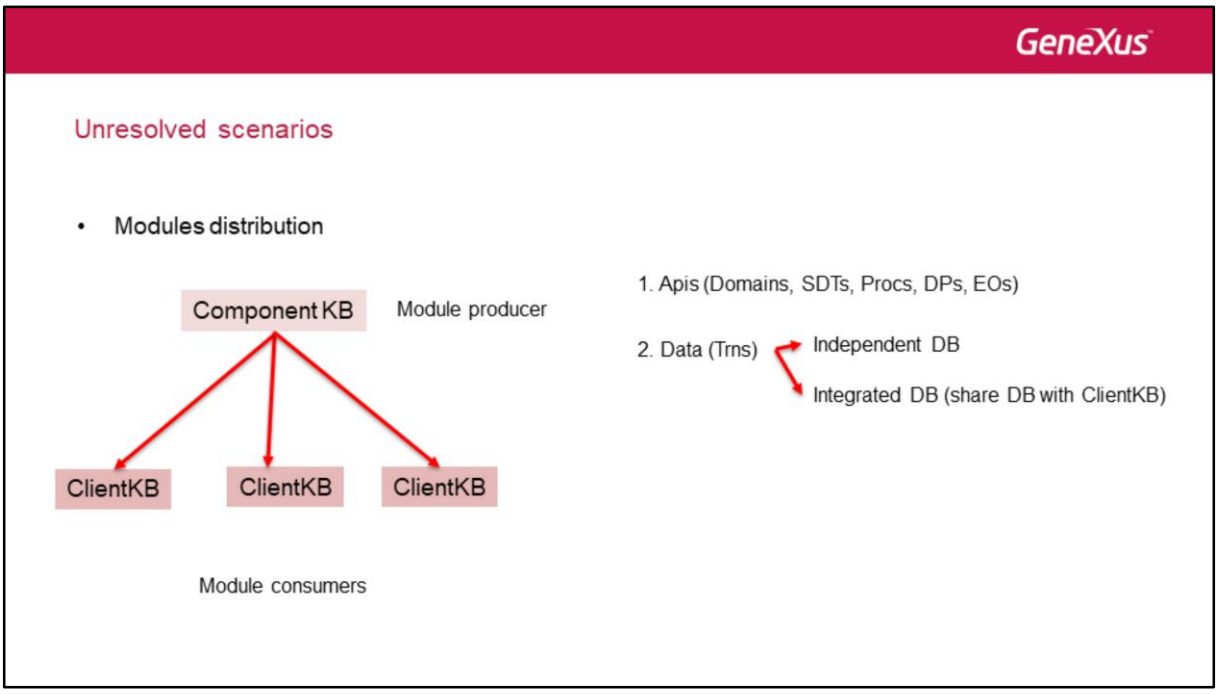


El escenario que teníamos hasta GeneXus 15 era uno donde los módulos se utilizaban solamente como forma de organizar la KB, encapsulando. A lo sumo podíamos exportar un módulo como xpz e importarlo en otra KB. Y el comportamiento del módulo iba a ser exactamente el mismo a que si se hubiera creado directamente en esa KB destino. De hecho, todos sus objetos pueden ser abiertos por cualquier desarrollador de la KB destino y actualizados. La propiedad Object Visibility solamente aplica a la hora de que un objeto invoque a otro y nada más.

No teníamos manera de distribuir módulos de forma tal que funcionen en cierto sentido como apis, donde el cliente no pueda ver la implementación de los objetos del módulo proveedor sino que solamente pueda utilizarlos a través de su interfaz, invocando a los servicios y pantallas definidas (sin poder ver su código). Y sin poder enterarse siquiera de la existencia de los objetos privados o internos del módulo.



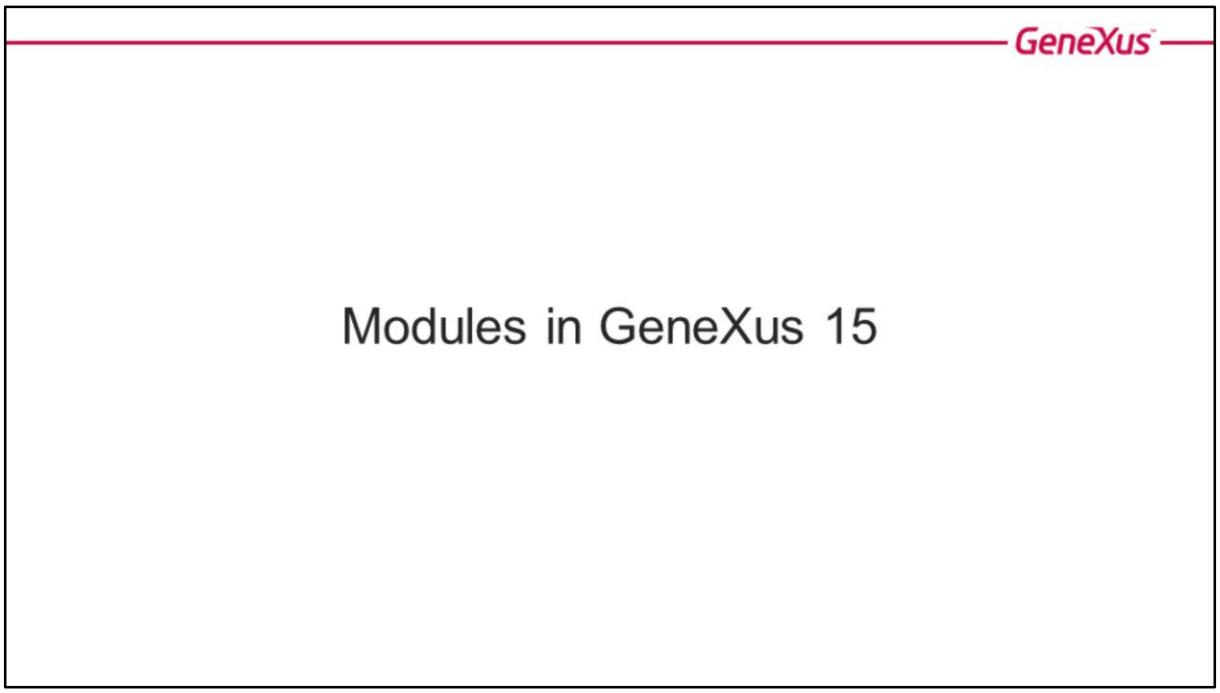
Es decir, no teníamos resuelto este escenario donde queremos que una KB desarrolle el módulo y otras puedan consumirlo, es decir, utilizar lo que el módulo provee, pero sin poder tocar nada de allí, ni siquiera ver la implementación (salvo la de las partes necesarias para poder utilizar el objeto, como la regla parm, el form y las variables).



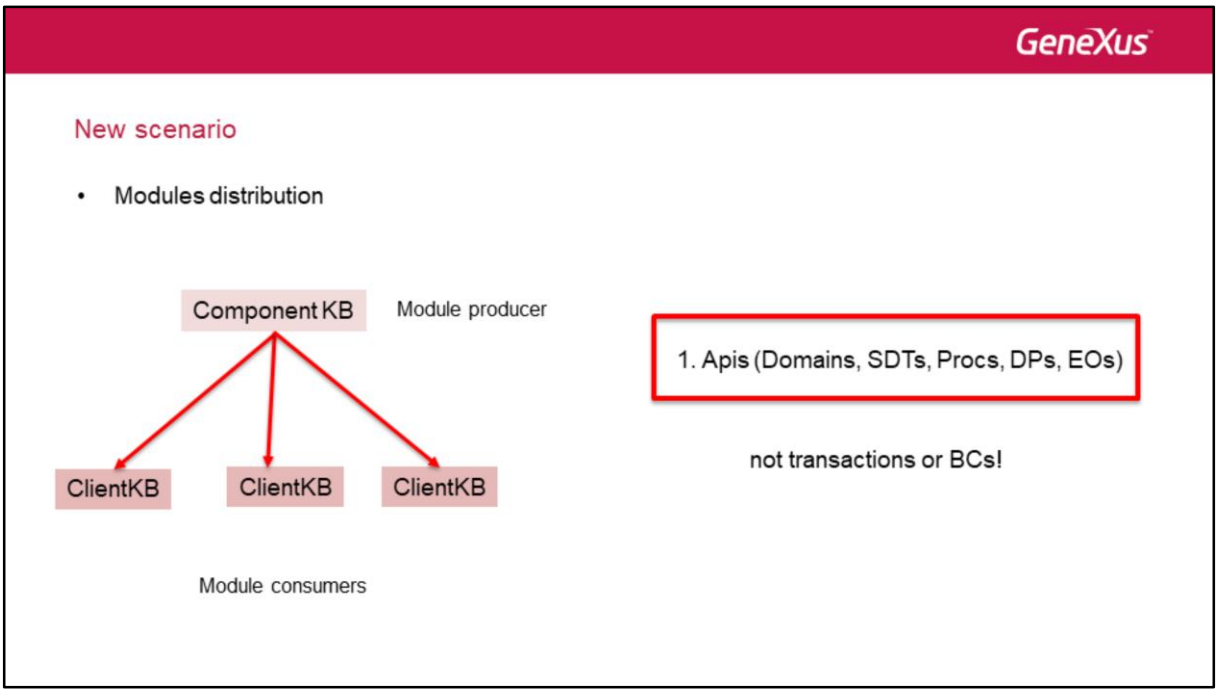
Este escenario a su vez se abre en tres:

1. Módulos que implementan apis que no requieren de acceso a base de datos (no contienen transacciones). Este es el escenario que se ha resuelto en GeneXus 15 y que abordaremos en lo que sigue.

2. Módulos que sí contienen acceso a base de datos. Allí se nos abren a su vez dos escenarios: módulos cuyas Trns deben trabajar sobre bases de datos independientes de las KBs que vayan a consumir esos módulos, y módulos cuyas Trns deben integrar la misma Base de Datos que la KB que consume el módulo. Estos escenarios siguen en investigación, por lo que serán resueltos en próximas versiones de GeneXus.

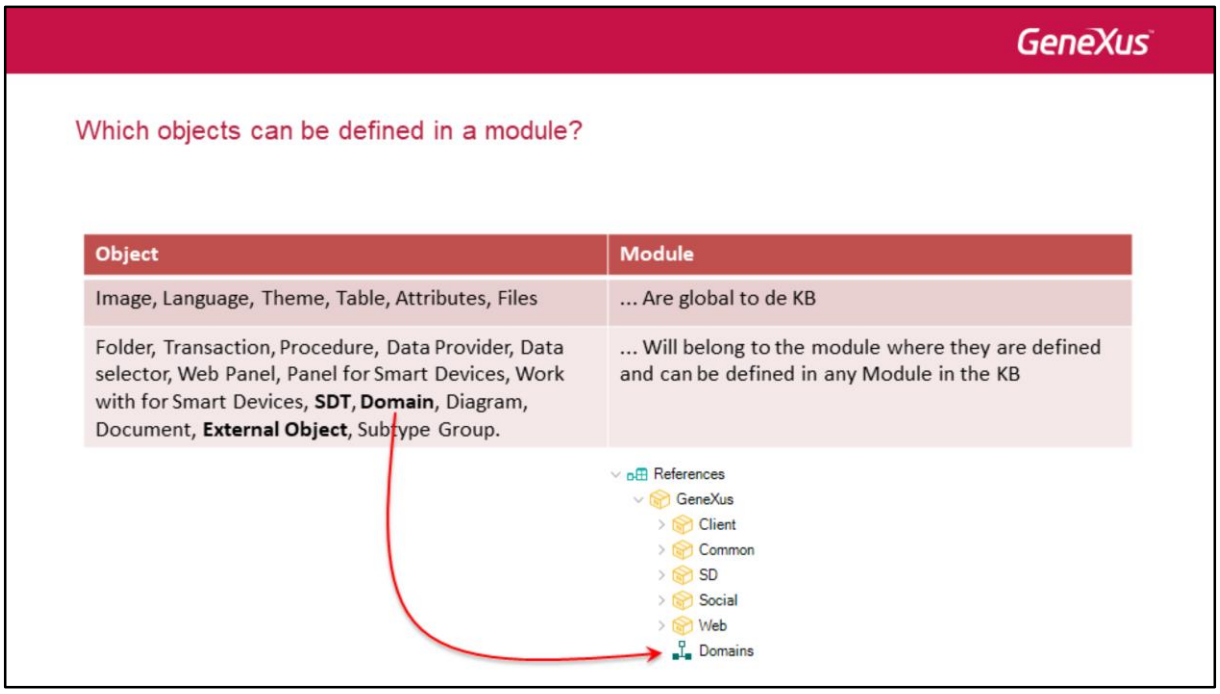


Entonces ahora pasaremos a estudiar el escenario que sí está resuelto en GeneXus 15.



Decíamos que GeneXus 15 resuelve el escenario en el que el módulo implementa una o varias apis. Eso significa que provee servicios para resolver algunas funcionalidades, pero que no requieren de datos de tablas propias.

Es decir, los únicos objetos del módulo serán Dominios, SDTs, Procs, DPs y EOs. Podrían haber paneles. Pero no transacciones o business components!

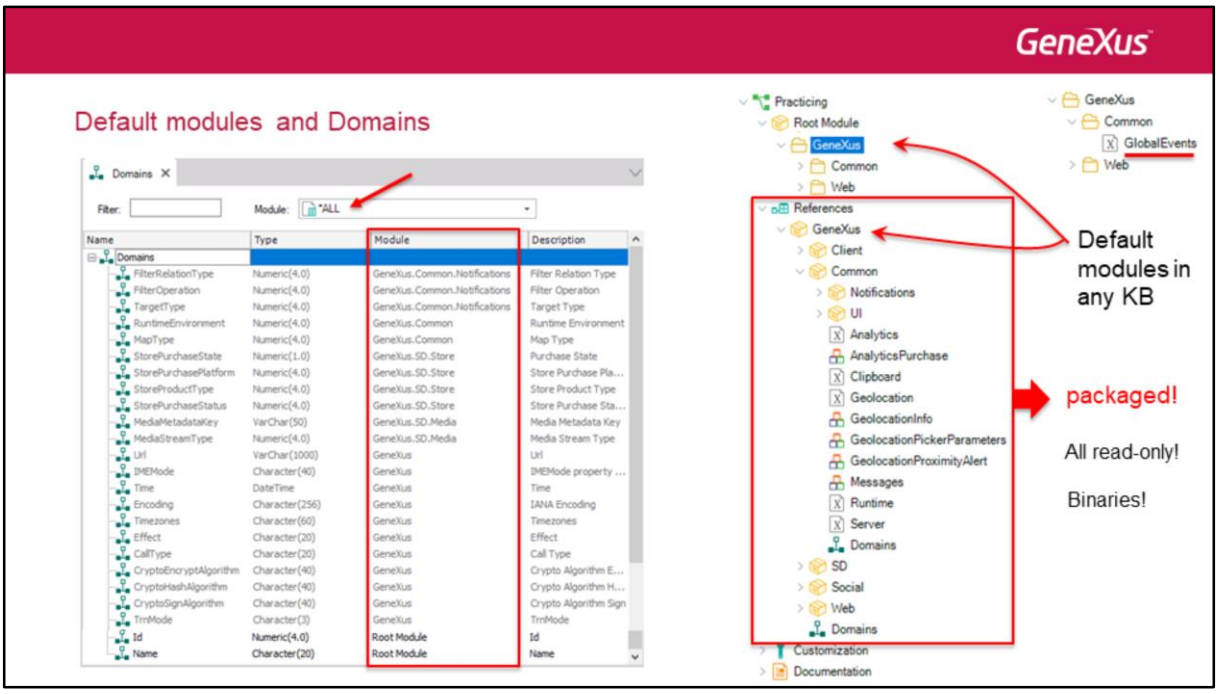


Aquí mostramos en una tablita qué objetos van a pertenecer al módulo donde estén definidos y cuáles, por el contrario, serán globales a toda la KB.

Los SDTs, External Objects y dominios antes eran globales y ya no. Por esa razón tendremos un nodo Domains en el KB Explorer por cada módulo.

[Nota: en Upgrade 10 no está apareciendo el nodo, pero es un error que será corregido]

Todavía no mencionamos qué es ese módulo GeneXus bajo el nodo References. Ya lo veremos.



En toda KB creada con GeneXus 15 habrá dos módulos default:

- el módulo Root, bajo el que se encuentra el folder GeneXus.
- El módulo GeneXus bajo el nodo **References**.

Este nodo References es en el que se van a encontrar todos los módulos que en esta KB se vayan instalando para ser consumidos. Si observamos, ya existe aquí un módulo con varios submódulos. Este módulo, conocido internamente como GeneXus core, contiene todas aquellas apis (objetos externos y sus SDTs) que en la versión previa se encontraban bajo el nodo Root de la KB.

¿Cuál es la gran diferencia? Que este módulo (y sus sub-módulos) no está abierto sino empaquetado. Esto significa que no pueden editarse los objetos dentro de él como hacíamos antes. Todo lo que puede verse puede utilizarse, pero ¡es read-only! Y no solo eso: lo que aquí se muestra de los objetos es solo su interfaz. En verdad lo que aquí se importa es... ¡los binarios! De esta manera se ahorra una considerable cantidad de tiempo de spec, generación y ejecución.

Este es el primer caso de un módulo que es producido por una KB (del equipo de desarrollo de GeneXus) y distribuido automáticamente (instalado) a todas las KBs que utilicen GeneXus 15.

Recomendamos que cree sus propios dominios o SDTs dado que ya no será posible personalizar aquellos distribuidos por GeneXus.

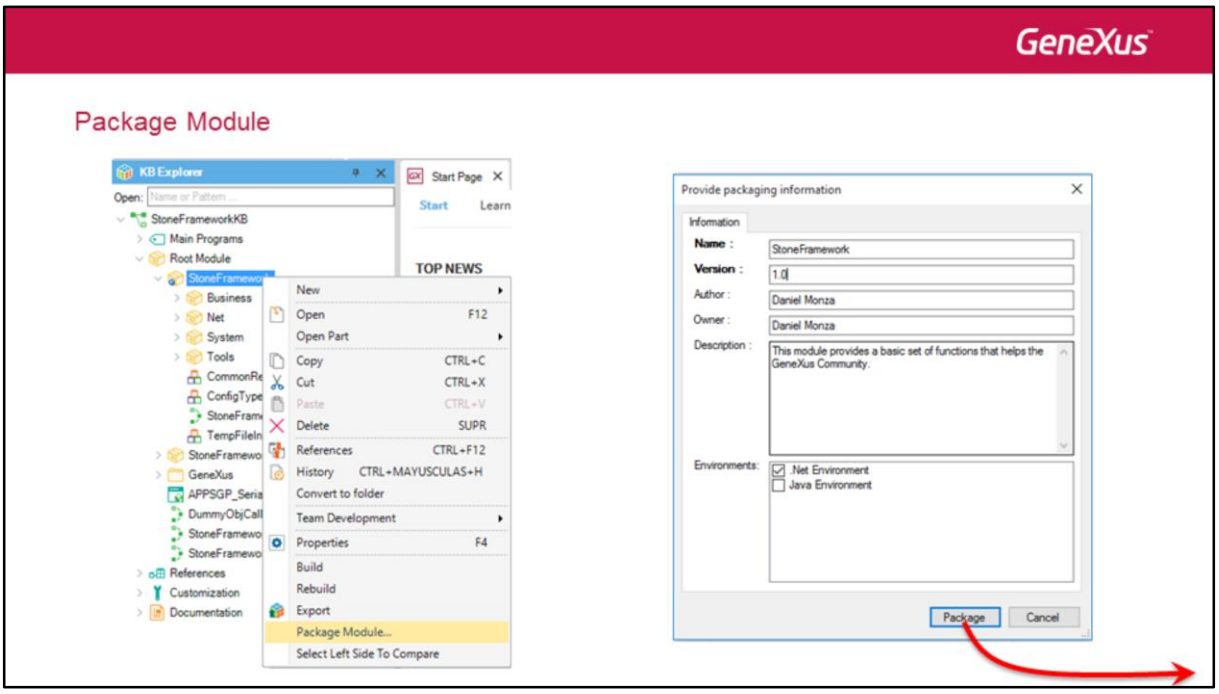
Este módulo GeneXus nos permite saber que cuando un cliente está trabajando con una versión específica de GeneXus tiene la versión correcta del módulo y sus objetos (APIs, SDTs, Domains).

En el editor de dominios hay ahora una nueva columna que muestra a qué modulo pertenece cada dominio listado. Los dominios que pertenecen a aquellos módulos distribuidos directamente por GeneXus aparecen como read-only, y lo mismo sucederá con dominios que provengan de módulos que los desarrolladores instalen en la KB.

¿Por qué se puede querer que dominios de un módulo sean públicos? Porque si alguno de los objetos públicos tienen regla parm definida, y alguno de los parámetros de esa regla parm está basado en un dominio, ese dominio debe ser conocido por la KB que esté queriendo trabajar con ese modulo.

¿Por qué el folder GeneXus del Root Module contiene un folder Common con un external object GlobalEvents? ¿Por qué ese EO no está también en el módulo GeneXus, empaquetado, como los demás? La respuesta es simple: si bien es un objeto predefinido, dentro de su función está el que el desarrollador lo modifique. No puede ser read-only.

Lo mismo sucederá con algunos dominios y EO del GAM.



Como decíamos antes, a partir de GeneXus 15 podemos empaquetar nuestros propios módulos para distribuirlos a otras KBs que los consuman.

Supongamos que en nuestra KB, bajo Root Module hemos desarrollado un módulo StoneFramework con objetos y submódulos que implementan algunas funcionalidades básicas como una función que transforma una cantidad de dinero en una moneda determinada, en texto, de acuerdo al idioma que se elija.

Queremos hacer un paquete con ese módulo para distribuirlo a otras KBs, que no puedan modificar esos objetos, ni ver su implementación completa, sino que sólo puedan usarlos. Para ello hacemos botón derecho sobre el módulo, y vemos que aparece la opción Package Module... que nos abre una ventana para proveer la información que vemos. El desarrollador debe configurar, entre toda la info, en cuáles de los environments de la KB va a querer que los objetos se generen. ¿Por qué? Porque lo que se va a distribuir no son los objetos GeneXus, sino la interfaz y ¡los binarios ejecutables!

En el ejemplo estamos indicando que queremos se genere el módulo en .NET únicamente. Observe que la KB podría tener también environment Android e iOS.
¡Pero al hacer esto, solamente una KB con environment .NET podrá hacer uso del módulo! En general, si usted desarrolla un módulo para ser consumido por otros, seguramente querrá generarlo en todos los lenguajes posibles.

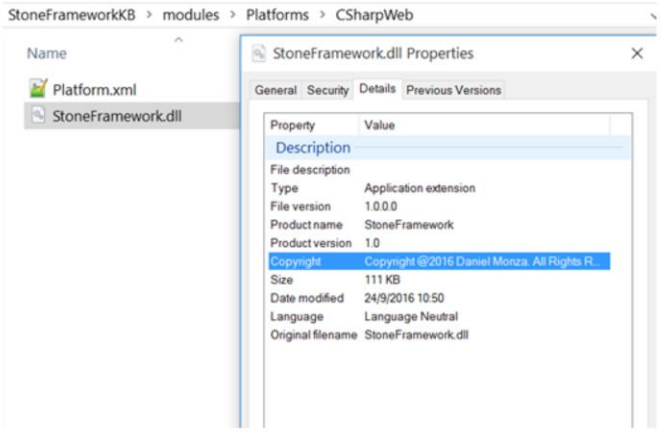
Observe también que entre la información del módulo aparece un campo Description, que corresponderá a la info que se le brindará a quien desee instalarse el módulo en otra KB, antes de hacerlo. Allí se pueden especificar limitaciones, etc.

Al presionar Package...

GeneXus

Package Module

Specification
Generation
Compilation



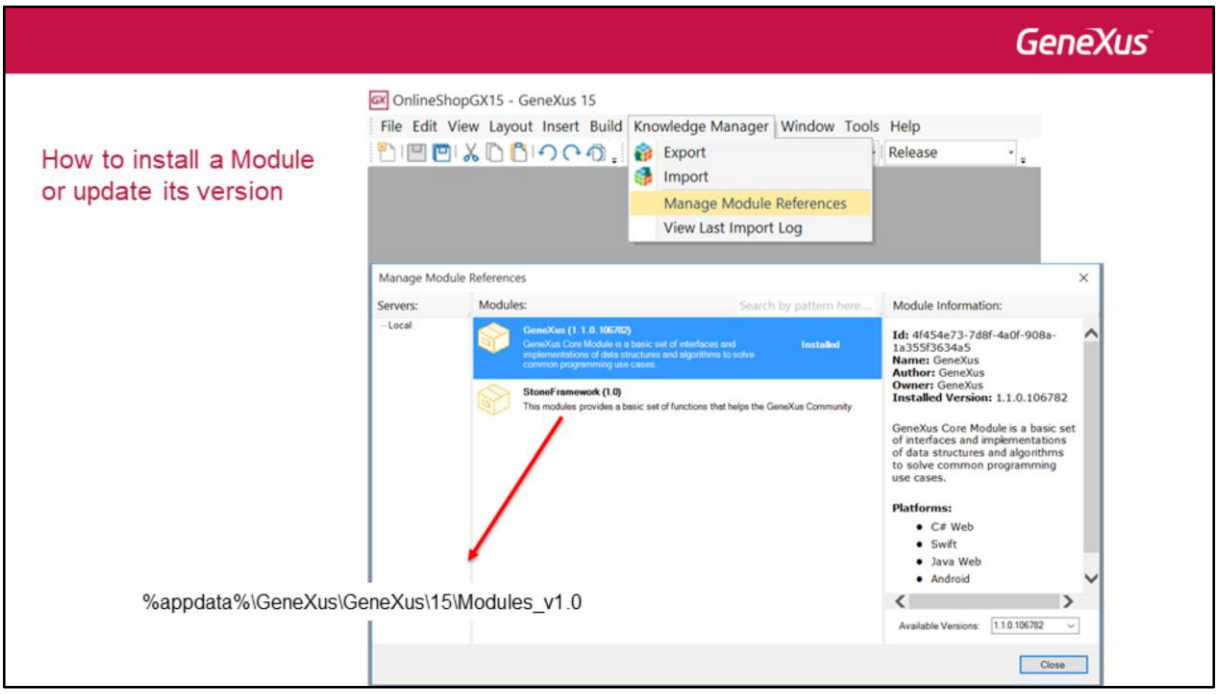
%appdata%\GeneXus\GeneXus\15\Modules_v1.0

GeneXus especifica, genera y compila todos los objetos dentro del modulo (hace rebuild) y una vez que el proceso culmina, crea en este caso una dll (pues le pedimos que armara el paquete solo para .Net; si hubiéramos seleccionado además Java habría generado un .class también).

El paquete contiene los binarios (e.g. dll) y los archivos de definición del módulo. Lo genera con extensión opc que significa Open Packaging Convention. Es un zip. En este caso en: "<Knowledge Base Directory>\modules\StoneFramework_1.0.opc“

El binario es creado proveyendo version e información de copyright.

Ya tenemos listo el paquete para ser instalado en cualquier otra KB. Por ahora, para que el módulo pueda ser instalado debe ser copiado en un lugar específico del disco (en el AppData del usuario). Si no se copia allí no será visto automáticamente por el Module Manager. Lo vemos en el siguiente paso.

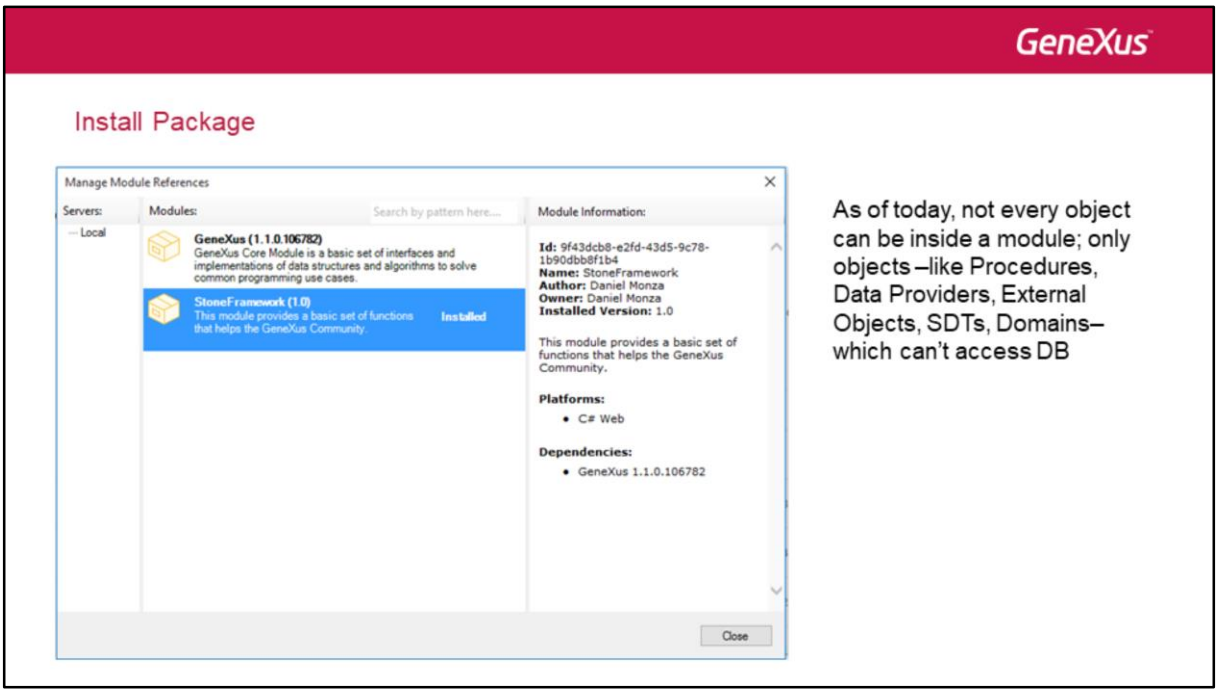


Ahora bajo la opción Knowledge Manager del menú de GeneXus podemos ver la opción Manage Module References.

Aquí veremos todos los módulos que la KB tiene instalados (siempre habrá por defecto uno: el módulo GeneXus) y los que tiene a disposición para instalar.
Por ahora son encontrables para instalar los paquetes que se encuentren en el directorio AppData del usuario. Estamos trabajando en una opción para compartir módulos, ya sea a través del marketplace o de una red de la empresa.

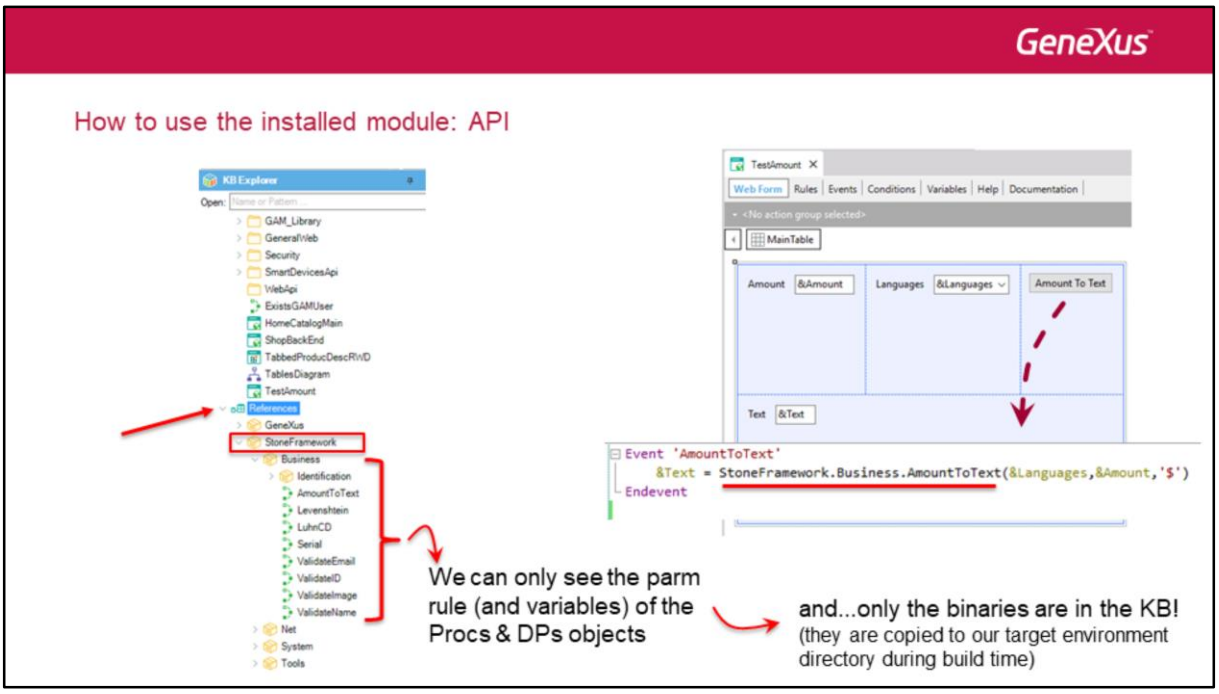
Para módulos ya instalados, como el GeneXus, nos ofrecerá actualizar la versión si quedamos desactualizados. De hecho podemos ver en el combo de la derecha las versiones disponibles del módulo.

Algunas veces puede pasar que no se vea en el Manager un módulo correspondiente a un paquete que tenemos correctamente copiado para instalar. Esto se resuelve ejecutando **GeneXus.exe /install** que automáticamente lo copia al lugar correcto y ahora sí el modulo aparecerá en la ventana del manager.



Como decíamos al principio, solo podemos resolver un escenario actualmente y es aquel en el que los objetos del módulo no acceden a la base de datos. Esto era porque teníamos que resolver el siguiente conflicto: tengo una DB en mi app, pero traigo objetos de un módulo de otra KB, por lo que no sabemos a qué DB acceden esos objetos del módulo. Tenemos que resolver cómo manejar este tipo de conexión y si el módulo puede acceder a otra DB o si tiene que ser la misma. Allí hay que resolver el tema de la reorg para crear las tablas, proveer data stores para cada DBMS, etc. Estos otros escenarios (módulo con trns/BCs) está bajo estudio.

El ejemplo que veníamos presentando era el de un módulo con procedimientos que resuelven un requerimiento simple, como validar una imagen o email, o enviar una cantidad o texto (que son herramientas que usamos muchas veces en nuestras KBs y que no tienen necesidad de acceder a la DB). También podrían acceder a servicios externos.



Al importar un módulo quedará bajo el nodo **References** del KB Explorer. La pregunta ahora es ¿cómo podemos utilizar los objetos de su interfaz (api)?

Como sabemos, al igual que los objetos del módulo GeneXus, estos objetos serán read-only. Sólo debemos saber cómo invocar a los objetos y qué es lo que cada objeto hace. Es decir, solamente tendremos una API disponible de esos objetos y lo único que veremos será su regla parm, dado que debemos conocer qué parámetros utilizar al invocar los objetos, y sus tipos de datos. En el módulo también podrán venir Eos, SDTs y Domains que serán visibles, claro.

En el ejemplo tenemos un procedimiento del módulo, AmountToText, que devuelve un texto formateado de acuerdo a los valores que le son enviados por parámetro. En la KB que instaló el módulo se define un web panel con tres variables en pantalla y un botón. El evento de este botón invoca a la API del proc del modulo StoneFramework.Business. Invocar a una API es lo mismo que invocar a cualquier objeto GeneXus. Podríamos no haber incluido el qualified name del objeto AmountToText si sabemos que AmountToText es único.

Cuando ejecutemos, GeneXus tomará al proc AmountToText como un objeto normal en la KB en lo que hace a las invocaciones (solo que AmountToText no necesitará ser generado, pues tenemos los binarios)

Todo sobre módulos aquí: https://wiki.genexus.com/commwiki/servlet/wiki?22414

GeneXus™

Videos	training.genexus.com
Documentation	wiki.genexus.com
Certificactions	training.genexus.com/certifications