How to Create CORBA IDL using Enterprise Architect

Introduction

Although largely superseded by Web Services the use of CORBA as an implementation platform for services is still very relevant in today’s systems. This is particularly true when components are communicating with legacy systems or for languages that do not support Web Services directly.

The services in a CORBA system are described using a language known as IDL (Interface Definition Language) (OMG, 2007).

Enterprise Architect from Sparx Systems does not provide IDL support directly, however Sparx Systems do provide an Add-In for IDL free of charge. This Add-In is available for download at http://www.sparxsystems.com/downloads/mdg_technology/EACorba.exe. Once downloaded and installed, IDL capabilities can then be added to an Enterprise Architect project.

This “Add-in” as provided does not generate IDL code that is completely accurate and the use of the CORBA elements is not entirely straightforward. It is the intention of this paper to describe:

1) What is provided by the Add-In.
2) The limitations in using the add-in unmodified.
3) The modifications necessary to ensure that accurate IDL code is generated.
4) How to use the CORBA elements in a design model ready for IDL code generation.

The fourth item will be illustrated by the use of a simple example based upon the book by Fintan Bolton (Bolton, 2002).

The IDL Add-In

After downloading the Add-In and double-clicking the file, it will be installed and upon re-starting Enterprise Architect, an extra line item CORBA will be added to the Add-Ins menu. This item results in four options as shown below:

- Install MDG Technology for CORBA
- Generate CORBA elements
- Help
- About
The first option adds CORBA modelling and IDL code generation to an existing Enterprise Architect project. The second option allows the generation of CORBA modelling elements from an existing class diagram. The last two options are obvious as to their function, however the Help option provides very limited help indeed!

**Install MDG Technology for CORBA**

Selecting this option simply adds the facility for CORBA IDL elements to be modelled and for IDL code to be generated from such a model. This is achieved by adding a new set of elements to the toolbox and the project resources. In addition some customized templates for code generation are also added to the project. The bulk of this paper is devoted to the use of the CORBA modelling elements and further customization of the code generation templates.

**Generate CORBA Elements**

This option is intended to generate and add to the current project, CORBA elements that correspond to classes/interfaces that have been defined already in the model.

This does have serious limitations and this process cannot be customized, in this paper I will describe what the effect of performing this operation is, you can decide whether or not it is worth it or not.

The effect of Generating CORBA elements from existing classes / interfaces is summarized below. This should be sufficient to give you a guide as the effectiveness (or not) of this process. The most appropriate way to see the effect is to experiment.

<table>
<thead>
<tr>
<th>UML Element</th>
<th>CORBA Element</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>CORBAInterface</td>
<td>Attributes defined in the class are lost</td>
</tr>
<tr>
<td>&lt;&lt;struct&gt;&gt; Class</td>
<td>CORBAStruct</td>
<td>Correct mapping No conversion of types</td>
</tr>
<tr>
<td>&lt;&lt;enum&gt;&gt; Class</td>
<td>CORBAEnum</td>
<td>Correct mapping</td>
</tr>
<tr>
<td>&lt;&lt;interface&gt;&gt; Class</td>
<td>CORBAInterface</td>
<td>Correct mapping, but any attributes defined are lost</td>
</tr>
<tr>
<td>&lt;&lt;exception&gt;&gt; Class</td>
<td>CORBAInterface</td>
<td>Incorrect mapping</td>
</tr>
<tr>
<td>&lt;&lt;type&gt;&gt; Class</td>
<td>CORBAInterface</td>
<td>Incorrect mapping</td>
</tr>
<tr>
<td>&lt;&lt;typedef&gt;&gt; Class</td>
<td>CORBAInterface</td>
<td>Incorrect mapping</td>
</tr>
<tr>
<td>&lt;&lt;sequence&gt;&gt; Class</td>
<td>CORBAInterface</td>
<td>Incorrect mapping</td>
</tr>
</tbody>
</table>

**Note:** In some attempt to achieve an accurate generation, Classes were stereotyped as shown above to ascertain as to whether or not the correct CORBA elements would be generated (they were not!)

In addition to the above, a Dependency link stereotyped with represents, is created between the generated CORBA element and the existing class / interface.

**Using the CORBA Elements for Modelling**

A full set of CORBA elements is available in the toolbox as shown below (version 7.0 of Enterprise Architect)
Most of these elements are used in the same manner as any other UML element, the exception is switch (used for CORBA unions) which if used from the toolbox, does not result in the correct IDL being generated.

The class diagram below represents a simple example of the common CORBA Elements
CORBA IDL Modelling Details

The following details the procedures to enable the correct model to be created using combination of both CORBA IDL elements and UML associations.

1) CORBAModule – this is used to create a stereotyped package with the Enterprise Architecture project to be created in addition to the visual element on the class diagram. This is necessary for the generation of the module { } construct in the generated IDL code. The construct module { } is not generated for a package constructed in the usual manner in an Enterprise Architect project.

2) CORBATypedef – this is to model a CORBA typedef IDL construction, if a typedef is being used like an alias for a CORBA primitive, for example

```plaintext
typedef long ISBN;
```

then the CORBATypedef is linked to the CORBAPrimitive via a UML Generalization link as shown in the diagram above. If preferred the CORBAPrimitive element can be deleted from the diagram.

3) CORBAPrimitive – this is used to model a CORBA primitive. All CORBA primitives are supported in Enterprise Architect, the most common scenario for including a
CORAB primitive in a model is when used in conjunction with a CORBA typedef (see 2 above)

4) **CORBAStruct** – this is used to model a CORBA struct element. Attributes can be added to this element in the same manner as when using Enterprise Architect to model classes. The visibility setting is ignored by the IDL code generation process.

5) **CORBACl asses** – this is used to model constants in IDL. When adding attributes to this element ensure that the **const** checkbox on the attributes properties dialog is checked and that an initial value is specified.

6) **CORBAEnum** – this is used to model enumerations in the IDL and is modelled in a similar fashion to enumerations in a UML class diagram. No initial values are required.

7) **CORBASequence** – this is used to model a typedef sequence in IDL, to set the correct type a UML Generalization link to a CORBAPrimitive or CORBAStruct element is used.

8) **CORBAInterface** – the most used CORBA modelling element and is used to model CORBA IDL interfaces. Attributes and operations can be added and this is exactly the same as when using Enterprise Architect to model UML classes. When specifying that an attribute is **readonly**, add a **readonly** tag to the attribute and set that tag’s value to true. This does not show up in the diagram, but can be viewed in the Tagged Values View. For convenience consider adding a **readonly** tag to the project via the **Settings | UML | Tagged Value Types** in Enterprise Architect (Remember that this setting is for the current project only). Inheritance hierarchies of CORBAInterfaces can be created using UML Generalization links. In a similar manner, for any operation that raises an exception add a **raises** tag to the operation and set the tag’s value to the CORBAException that is being raised.

9) **CORBAException** – this is used to model an exception in the IDL. Note that when using Enterprise Architect, CORBA exceptions are defined as separate entities within the IDL and not nested within a CORBA interface.

10) **CORBAUnion** – this is used to model a CORBA union element within the IDL. Attributes are added as usual. The CORBA union makes use of tags both for the CORBAUnion element and the attributes. For the CORBAUnion element the tag used is **switchType** and its value is the type to be used for the IDL switch construction.
    Each attribute has a **cases** tag associated with it, the value of this tag is 1 for the first attribute, 2 for the second and so on. These tags and their values are used in the IDL code generation process.

11) **UML Association** – this is used in exactly the same manner as associations in a UML Class diagram. When one CORBA element has knowledge (uses) another CORBA element then these CORBA elements are connected with a directed association. The target role name should be set and will be used to generate the correct reference in the source CORBA element. If a UML Association is used between a CORBAInterface element and some other element (for example a CORBAStruct) is read only, the adding a **readonly** tag and setting its value to true (the same process as for an attribute) will result in the correct IDL being generated.
Generating the IDL Code

There are several problems with generating IDL using the CORBA MDG extension in Enterprise Architect:

1) By default, each CORBA element will generate its own individual *.idl file.
2) There is no recognition of the UML associations; hence attributes for these associations are not generated in the IDL.
3) Although the CORBAConstants element could be used in a model, no IDL code is generated for this element. (However, attributes defined in an interface with the const option check and an initial value defined will result in IDL code being generated, but this code is incorrect!)
4) Syntax error in the IDL code generated for an operation that raises an exception. (The Exception name is not enclosed in brackets)
5) Incorrect code generated for a CORBA union; the keyword case is omitted in the generated code.
6) Each value in a CORBA enumeration is listed on a separate line — not an error, but an unconventional arrangement of code;
7) The terminating ; for CORBA typedef is on a newline. No changes to the template are possible to modify this, at least no if you desire the code/model synchronization to work! After generating the IDL, simple modify the code manually.
8) No ; is generated for a CORBASEquence element

The following are details of how these errors in the IDL generation are corrected by modifying the code generation templates for IDL. Again, these modifications will only affect the current project, so it may be appropriate to create an Enterprise Architect base project, with these customization applied and use this as a base for all future projects which require CORBA IDL.

Generating a Single IDL file for all CORBA elements

1) Right-click the package containing the CORBA elements in the Project Browser and select Code Engineering | Generate Source Code...
2) Uncheck the option Auto Generate Files
3) Select Overwrite Code from the DropDown list
4) Upon clicking Generate, you will be prompted for a filename for each separate CORBA element, simply choosing the same file for each will result in a single file for the generated IDL code. This only has to be performed once, since Enterprise Architect “remembers” the filename name for all future IDL code generation.
5) Note that, as usual, if Synchronize Model & Code is chosen from the DropDown list, this process will fail if there are any syntax errors in the underlying IDL code.

IDL Code Generation template modifications for UML associations

In the IDL code generation templates provided in the IDL extension, no code is present in the macros LinkedAttribute, LinkedAttributeNotes and LinkedAttributeDeclaration. The following is the code that needs to be added to each macro respectively.
• **LinkedAttribute**
  
  `%LinkedAttributeNotes%
  `%LinkedAttributeDeclaration%

• **LinkedAttributeNotes**
  
  `%WRAP_COMMENT(associationNotes,genOptWrapComment,"","// ")%`

• **LinkedAttributeDeclaration**
  
  `%PI=""%
  %if classStereotype == "CORBAInterface" and linkTag:"readonly" == "true"% readonly %endif%
  %if classStereotype == "CORBAInterface"% attribute %endif%
  $type = %linkAttName%
  $type %if linkAttRole != ""% linkAttRole%; %else% %REPLACE(genOptDefaultAssocAttName,"$LinkClass",linkAttName)% %endif%

These changes will ensure that the appropriate reference will be added to the CORBA element, if this element is a CORBA interface, then prefix attribute or readonly attribute will added accordingly.

**Ensuring correct IDL for constants in an interface, or for a CORBAConstants element**

The IDL code for an interface attribute is achieved by a stereotype override for the template AttributeDeclaration. The change for this template to generate correct IDL for constant attributes is summarized below (the changes are highlighted):

<table>
<thead>
<tr>
<th>Original Macro</th>
</tr>
</thead>
</table>
| `%PI=""%
%if attConst=="T" and attInitial != ""% const %else% %if attTag:"readonly" == "true"% readonly %endif%
attribute %endif%
%attType%
%PI=""%
%attName%
%if attConst=="T" and attInitial != ""% = %attInitial%
%endif% ; |

<table>
<thead>
<tr>
<th>Modified Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>`%PI=&quot;&quot;%</td>
</tr>
</tbody>
</table>
To generate code for the CORBAConstants element

1) The first line of the macro **Class** is modified to the following (written on a single line)

```idl
%if classStereotype != "CORBAArray" and classStereotype != "CORBAEnum" and classStereotype != "CORBAException" and classStereotype != "CORBAInterface" and classStereotype != "CORBASequence" and classStereotype != "CORBAStruct" and classStereotype != "CORBATypedef" and classStereotype != "CORBAUnion" and classStereotype != "CORBAConstants"
```

2) The following code is added to the template **ClassDeclaration**

```idl
%if classStereotype == "CORBAConstants"
   // CONSTANTS
   //
%endTemplate
%endif
```

This places a comment into the IDL code

3) A new Stereotyped Override for CORBAConstants is added to the template **ClassBody**. This override has the following line of code

```idl
%list="Attribute" @separator="\n" @indent="\t"
```

This simply creates a line of IDL code for each attribute defined in the CORBAConstants element and re-uses the existing template **Attribute**

**Corrections for IDL code for an operation that raises an exception**

The IDL code for an operation is handled in the macro **OperationDeclaration**. The change for this template to generate correct IDL for raising an exception is summarized below (the changes are highlighted)

```
<table>
<thead>
<tr>
<th>Original Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>%PI=&quot;&quot;%</td>
</tr>
<tr>
<td>$qualifiers = %opTag:&quot;EATypeQualifiers&quot;%</td>
</tr>
<tr>
<td>%if $qualifiers != &quot;&quot;%</td>
</tr>
</tbody>
</table>
```
%REPLACE($qualifiers,";","::")%
%endIf%
-opReturnType=""? "void" : value%
-opName%
(%list="Parameter" @separator="", "")
$raises = %opTag:"raises"
if $raises != ""%
$raises = %REPLACE($raises, ",", ",\n\t\t")%
\n$raises $raises
%endIf%
;

Modified Macro

$PI=""
$qualifiers = %opTag:"EATypeQualifiers"
if $qualifiers != ""
%REPLACE($qualifiers,";","::")%
%endIf%
-opReturnType=""? "void" : value%
-opName%
(%list="Parameter" @separator="", "")
$raises = %opTag:"raises"
if $raises != ""
$raises = %REPLACE($raises, ",", ",\n\t\t")%
\n$raises ($raises
%endIf%
;

Corrections for IDL code for a CORBAUnion element
The IDL code for CORBAUnion is handled in a Stereotype Override CORBAUnion for the macro AttributeDeclaration. The change for this template to generate correct IDL for a CORBAUnion is summarized below (the changes are highlighted)

Original Macro

$PI=""
$cases = %attTag:"cases"
$cases = %REPLACE($cases, ":", ":\n")
$cases = %TRIM_RIGHT($cases, "\n")
if $cases != ""
$cases
$attType=""? "any" : value%
$attName%;

Modified Macro

$PI=""
$cases = %attTag:"cases"
$cases = %REPLACE($cases, ":", ":\n")
$cases = %TRIM_RIGHT($cases, "\n")
if $cases != ""
case $cases:
$attType=""? "any" : value%
$attName%;

Reformatting the IDL code for a CORBAEnumeration element
The IDL code for a CORBAEnumeration is handled in a Stereotype Override CORBAEnum for the macro ClassBody. The change for this template to generate correct IDL for CORBAEnumeration is summarized below (the changes are highlighted)
Note the modified code for the template is entered on a single line.

Adding the ; for IDL for a CORBASequence element

This is achieved very simply by adding a Stereotype Override for the CORBASequence in the ClassBody template, this override just contains a single ;

This is same technique as the existing Stereotype Override for CORBATypedef.

IDL Generated

After all the modifications listed above are applied, the following is the IDL file that is generated for the model shown in Figure 1.

```idl
module IDLS
{
    module BookRepository
    {
        struct BookDetails
        {
            string title;
            Date publication_date;
            PersonName author;
            ISBN book_id;
        };

        typedef sequence<BookDetails> BookDetailsSeq;

        interface BorrowableCollection : Collection
        {
            const attribute LOAN_PERIOD = 21;
            void borrow_book(in ISBN book_id, in PersonName borrower, out Date return_date)
                raises (Unavailable);
        };

        interface Collection
        {
            readonly attribute long number_of_books;
            // name of collection
            attribute string name_of_collection;
            FuzzyBoolean is_in_collection(in ISBN book_id);
        };

        // CONSTANTS
        //
        short MIN_DISCOUNT_PERCENT;
        short MAX_DISCOUNT_PERCENT;

        struct Date
```
Generating a Model from IDL

This process works reasonably well, any errors that may be detected will have to be corrected manually, since this process is not customizable.

Conclusion

The CORBA IDL add-in from Sparx Systems does add the ability of using UML to model CORBA elements, however to generate correct IDL code, some modifications to the IDL code generation templates are necessary. This paper has detailed the changes that are necessary to achieve accurate IDL code to be generated.
Works Cited
