Enterprise Application Integration with OLE Automation and ArcGIS in the Telecommunications Industry

Bob Hake

With the ever growing number of point solutions that need to be integrated in a business workflow it is critical that data is able to easily flow from one application to another. This paper explains how this was achieved by developing an automation model with components that simplify feature operations inside ArcGIS and data transfer in/out of the geodatabase. This project used OLE automation technology to flow data through an ArcGIS-based application to a telecommunications provisioning system and it can be generalized to streamline data flow between any ArcGIS application and Office apps or any other third party solution.

Introduction

To be competitive, profitable and responsive to customer needs, service providers in the telecommunications industry today need to be able to integrate their complex network and operations infrastructures with a wide range of Operation Support System (OSS) applications. Gone are the days where point solutions can operate independently as increasingly the GIS database is becoming the center of all data activity for telecom providers. Achieving seamless data flow-through via application integration is a critical step in creating a successful enterprise software environment.

This paper will explain how this level of integration can be made easier with one solution that uses ArcGIS and a well-defined OLE automation model to streamline the development process. This paper will discuss what Automation is, the advantages of using it, lay out in detail an example of a telecom automation model and explain how it can be implemented.

Automation

Automation is a mature technology that has been around since the early days of COM over ten years ago. Automation is the COM technology based on the IDispatch interface that all Windows developers are familiar with as Visual Basic and all ActiveX controls expose their functionality through this. By implementing the IDispatch interface a client application is able to access functionality dynamically from the object without having to know the details of an object’s custom interface. When automation is implemented by a product, it is typically depicted in a drawing as an inverted tree-shaped hierarchy of components starting with the Application object at the top and branching down to the individual objects that perform the work (e.g. the Microsoft Excel automation model).

As regards GIS, automation formed the foundation of MapObjects and its simple, easy-to-understand automation model was a major reason for the popularity of MapObjects. If presented with a simple GIS programming project it was relatively straight forward for a developer to open up Visual Basic and use MapObjects to quickly create a solution.
Automation remains an important development tool but today newer tools such as ActiveX Template Library (ATL) have taken precedence. ArcGIS, an example of an application built with ATL technology, uses ArcObjects as the development platform for all customization. ArcObjects contains a vastly larger set of components - in ArcGIS 8.3 the type library esricore.olb alone exposes 3622 classes and enumerations. The sheer number and complexity of the ArcObjects components makes application development a more daunting task.

Add to this the fact that because ArcObjects components typically implement multiple custom interfaces they are inter-related in a much less rigid top-down fashion than the automation hierarchy as seen in MapObjects. When drawn in diagram form, the ArcObjects components are displayed in object model diagrams that resemble less a hierarchical tree structure and more a spider web of related objects. As a result, using ArcObjects requires a greater understanding where all the interfaces are defined and it is much more difficult to step through the objects and discover the functionality of the components without jumping all around the object model.

A well organized automation model provides a simple, auto-discoverable method of exposing application functionality that can be quickly implemented in a customization project.

**Telecom Data Model**

Data models are used in every application and serve as a way to organize data, both spatial and attribute information, in a meaningful way. Figure 1 shows an example of a telecom data model that encompasses both outside plant (OSP) and inside plant (ISP) functionality showing the relationships between the different objects. Each object in the diagram represents the telecom inventory items that combine to make up the network infrastructure.

![Figure 1 - Example telecom data model](image-url)
Using an automation model that is associated with a industry-specific data model and is layered on top of ArcObjects allows the application to expose the power of ArcGIS with its generic GIS functionality and also provide an industry-focused set of components.

Figure 2 Telecom Automation Model part 1
Figures 2 and 3 show an example automation model that uses the telecom data model in Figure 1. These diagrams depict the standard objects in solid gray color and the collection objects in a lighter dotted shade.

Figure 3 Telecom Automation Model part 2

This data model can be loosely divided into two parts: non-versioned data objects and versioned components. The objects in Figure 2 represent the non-versioned data objects with the exception of Circuits, Connections and WorkOrders which are versioned. The components in Figure 3 represent entirely versioned data objects which are implemented as feature and object classes in ArcGIS. From Figure 3 you can see that the Spans collection...
which contains a collection of conduit linear features from the geodatabase references a collection of duct and inner-ducts which are modeled as a SpanUnit object class in ArcGIS.

The Application object is at the top of the automation model and all other components are accessed through it. Figure 4 displays a table of the properties and methods that are exposed by the Application object. There are four types of items in the Application object:

1) feature class collections that are implemented as graphic feature classes in the geodatabase (e.g. Structures, Equipments, Locations)
2) telecom-specific collections that manage versioned data implemented as object classes in the geodatabase (e.g. Circuits, Connections, WorkOrders)
3) objects that used for telecom-specific functionality (e.g. ReferenceData, Contracts, Administration, Reports)
4) components that allow simple, quick access to ArcGIS functionality (e.g. IApplication, Editor, Document)

<table>
<thead>
<tr>
<th>Name</th>
<th>P/M/E</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbortEditOperation</td>
<td>M</td>
<td>Cancels edit operation</td>
</tr>
<tr>
<td>Administration</td>
<td>P</td>
<td>Core data management operations</td>
</tr>
<tr>
<td>Attachments</td>
<td>P</td>
<td>Attachment features</td>
</tr>
<tr>
<td>Circuits</td>
<td>P</td>
<td>Circuits collection</td>
</tr>
<tr>
<td>CommandLine</td>
<td>P</td>
<td>Command line arguments</td>
</tr>
<tr>
<td>Connections</td>
<td>P</td>
<td>Connections manager</td>
</tr>
<tr>
<td>Contracts</td>
<td>P</td>
<td>Contracts information</td>
</tr>
<tr>
<td>Database</td>
<td>P</td>
<td>Quick access to table metadata and data</td>
</tr>
<tr>
<td>DetailViews</td>
<td>P</td>
<td>Detail views of ISP graphics</td>
</tr>
<tr>
<td>Document</td>
<td>P</td>
<td>Map document functions</td>
</tr>
<tr>
<td>DropLines</td>
<td>P</td>
<td>DropLine features</td>
</tr>
<tr>
<td>Editor</td>
<td>P</td>
<td>ArcMap Editor</td>
</tr>
<tr>
<td>Equipments</td>
<td>P</td>
<td>Equipment features</td>
</tr>
<tr>
<td>IApplication</td>
<td>P</td>
<td>ArcMap IApplication interface</td>
</tr>
<tr>
<td>Landbase</td>
<td>P</td>
<td>Landbase manager</td>
</tr>
<tr>
<td>Locations</td>
<td>P</td>
<td>Location features</td>
</tr>
<tr>
<td>NetworkInterfaces</td>
<td>P</td>
<td>NetworkInterface features</td>
</tr>
<tr>
<td>Plots</td>
<td>P</td>
<td>Plot manager</td>
</tr>
<tr>
<td>ProductName</td>
<td>P</td>
<td>Current installed product</td>
</tr>
<tr>
<td>ReferenceData</td>
<td>P</td>
<td>Non-versioned data manager</td>
</tr>
<tr>
<td>Reports</td>
<td>P</td>
<td>Report manager</td>
</tr>
<tr>
<td>Spans</td>
<td>P</td>
<td>Span features</td>
</tr>
<tr>
<td>SpliceClosures</td>
<td>P</td>
<td>SpliceClosure features</td>
</tr>
<tr>
<td>StartEditOperation</td>
<td>M</td>
<td>Start workspace edit session</td>
</tr>
<tr>
<td>StopEditOperation</td>
<td>M</td>
<td>Stop edit session</td>
</tr>
<tr>
<td>Structures</td>
<td>P</td>
<td>Structure features</td>
</tr>
<tr>
<td>Transmedias</td>
<td>P</td>
<td>Transmedia features</td>
</tr>
<tr>
<td>WorkOrders</td>
<td>P</td>
<td>Work Order manager</td>
</tr>
</tbody>
</table>

Figure 4 Application object methods and properties
Data Integrity

Another important advantage to implementing an automation model that will serve as the primary customization interface is that it puts a protective programmatic layer between the custom application and the underlying ArcObjects/database tools that access the data. The automation model provides a level of security designed to preserve and enforce data integrity so invalid data is not allowed to be written to the database. The automation objects are aware of the telecom data model, object relationships, rules, exceptions and allowable customization features that the lower level ArcObjects and database functions are not.

One example of enforcing data integrity by means of the automation model is the process of data translation from one format to another. There are times when telecom data from a legacy system needs to be transferred to a new system. One effective solution is to dump the legacy data to a neutral format (e.g. XML) and develop a custom application that uses the new system’s automation objects to read the XML data and populate the database. The formats and data model constraints can vary greatly between the two systems but by programming with the automation model the new data will be validated as it is being converted ensuring data integrity.

Another benefit of using an industry-specific automation model in regards to data integrity is that it makes it easier to develop automated test procedures (ATPs) that use the automation objects to test each application software release to see if any new functionality is causing errors. ATPs are a great tools to flush out errors in the software build cycle and use of the automation objects makes developing ATPs easier to do.

A related benefit from a developer’s point of view is that programming against an automation model makes many operations simpler because a single function call can be used to perform and operation rather than issuing multiple lower-level calls. When using an automation model in a custom programming project the programmer will be required to do less coding and at the same time will be confident that the database will not be compromised.

Implementation – VB vs. VC++

It is an open question in application development circles what is the best development tool to use to implement an automation model – Visual Basic or Visual C++. There are advocates for both tools and sound arguments on both sides. In summary, VB provides a better environment for rapid application development and has better GUI development tools while VC++ supports a more customizable type library toolset using IDL, it has better error handling functions as well as a better debugging environment.

It is possible to use both VB and VC++ and take advantage of the strengths of both tools as was done recently in the development of a telecom automation model. In this case, VB was considered to be preferable for implementing the automation objects because of its ease of use and rapid development features but was seen as inferior for producing a type library that will be used by all custom developers of the automation model. In this case the type library was implemented in a VC++ project which defined the top-level Application object while the implementation for all the methods/properties of the Application as well as all child objects was done in a VB project. The only trick for this implementation was to make sure that the GUIDs defined in the VB project are referenced correctly in the VC++ type library project.
This issue is solved by using the OLE/COM Object Viewer applet in Visual Studio to view the VB dll and copy the object declarations from the ITypeLib Viewer dialog to the .idl file in the VC++ project. Doing this keeps the type library in synch with the VB implementation project.

Following is a code snippet from VB showing how the WorkOrders.Remove automation method was implemented. Below that is the object definition code from the OLE/COM Object Viewer for the WorkOrders object that was copied to the Application.idl file in the VC++ project.

```vbnet
Public Sub Remove(vKey As Variant)
    On Error GoTo ErrorHandler
    If m_pWOMgr Is Nothing Then Set m_pWOMgr = SystemMgr.WorkOrderMgr
    m_pWOMgr.DeleteWorkOrder (vKey)
    m_colWorkOrders.Remove vKey
    Exit Sub
ErrorHandler:
    ProcessError "WorkOrders::Remove"
End Sub
```

```idl
[odl,
uuid(1C834DFA-583D-4A2D-8A36-7B4DF91064C3),
version(1.0),
hidden,
dual,
nonextensible,
oleautomation]
interface _WorkOrders : IDispatch {
    [id(0x68030001), propget]
    HRESULT Count([out, retval] long* );
    [id(00000000), propget]
    HRESULT Item{
        [in] VARIANT pKey,
        [out, retval] _WorkOrder**
    }
    [id(0xffffffffc), propget, hidden]
    HRESULT NewEnum([out, retval] IUnknown** );
    [id(0x60030004)]
    HRESULT Add{
        [in] BSTR sWOName,
        [in, out] BSTR* sTOType,
        [out, retval] _WorkOrder**
    }
    [id(0x60030005)]
    HRESULT Remove([in, out] VARIANT* vKey);
    [id(0x68030000), propget]
};
```

```idl
[uuid(4B3AED63-933A-4BCB-8778-8A05BE973C30),
version(1.0)]
coclass WorkOrders {
    [default] interface _WorkOrders;
    [default, source] dispinterface ___WorkOrders;
};
```
Implementation Issues

One of the primary implementation issues when creating a telecom automation model is naming conventions. When designing an automation model of any complexity beyond the most simple the work of naming each object, property, method and event is a non-trivial task. There are three naming convention rules that should be followed when it comes to creating a good automation model: 1) coherence 2) consistency and 3) completeness.

Automation object names should be coherent in the sense that the names should be succinct, common (i.e. minimize jargon) and clearly identify the object, property, method or event. A good automation model should read like a book to a new programmer as they descend the automation hierarchy. It should not require a developer to know the intricacies of the data model and how the database has been laid out in order to determine what object to use in a custom program. Long names are ok if needed for clarification but all things being equal a shorter name is better. Use of common words is good as many times they convey implied functionality (e.g. most developers are familiar with a Document object) but care must be taken to avoid reserved words (e.g. Type is not allowed in VB as a property name).

Automation object names should be consistent so that functionality with a property name in one object is named the same if found in another object. A programmer’s job is made much easier when it looks like the automation model is developed as a consistent unit rather than each object independently. Rules for consistent naming should be created and adhered to. For example, all collection names should be the plural of the object in the collection, the word “Is” should precede boolean properties if they are read-only (e.g. object.IsEditable) but should not be there if it is read-write (e.g. object.Visible) etc.

Automation objects should be complete so that no functionality is missing which will prevent the custom programmer from doing all that he needs to do to complete the project. As mentioned earlier, sometimes in order to make objects complete it is necessary to expose interfaces from underlying objects that the developer can access from the automation model (e.g. Toolbar has a ICommandBar property). This allows the model to be complete without duplicating functionality.

Conclusion

In a telecom enterprise computing environment no software application can exist as an island – it must be able to easily share data with other applications or it will quickly become obsolete. One way to allow for quick customization and streamlined data flow-through is to publish an OLE automation model of easy-to-use components that can be used by novice developers for rapid application development projects. The advantages of implementing this type of automation model are:

1) functionality is auto-discoverable
2) objects are industry focused
3) objects are simpler to use
4) data integrity of the underlying network is enforced
5) testing with these objects can flush out errors quicker
References

1. ArcObjects OnLine, ESRI

Author Information

Bob Hake
Senior Software Consultant
MESA Solutions, Inc.
7800 Madison Blvd.
Huntsville, AL 35807
256.864.0400 (tel)
256.864.0251 (fax)
rhake@mesahq.com