

Automated Component Bridge Generator

Dominik Glaser
docufy GmbH

D-96047 Bamberg
Email: dominik@docufy.de

Gregor Fischer
Würzburg University
Institute for Informatics
D-97074 Würzburg

Email: fischer@informatik.uni-wuerzburg.de

Jürgen Wolff von Gudenberg
Würzburg University
Institute for Informatics
D-97074 Würzburg

Email: wolff@informatik.uni-wuerzburg.de

Abstract—This position paper describes an automatic component bridge generator for embedding COM components in the Eclipse rich client platform. While embedding of ActiveX controls (graphical COM components) is in principle possible in Eclipse, the data model of the control is not easily accessible, and if done anyway, the task is quite tedious and error-prone.

Therefore an automated component bridge generator was developed based on a model transformation framework, that analyses information about the component and automatically generates a bridge for the component to be used in Java. Special care must be taken of resource management, event handling, type mapping and constructs in COM that are not directly available in Java like optional or out parameters.

Using the generator bridges were created within minutes that previously took several months to code. They allow numerous components like Word or Nero Burning Rom to be used, embedded and controlled from Java applications.

I. INTRODUCTION

When developing an application for the Eclipse platform, the core Java library and of course the components of the Eclipse platform itself (plug-ins) are available. Although this accumulates to quite a collection of reusable software, these are mostly rather abstract. More specific components like a spreadsheet are not easily available (yet).

On the other hand, the missing components are often available outside of Eclipse, e.g. as applications. Often applications export all or parts of their functionality as components to be embedded in other applications. Unfortunately it is quite tedious and error-prone to create bridges from the Eclipse/Java-World to the component model of the system. Therefore an automatic component bridge generator was developed. It builds upon a model transformation framework that simplifies transformations of component models. Using the framework the required transformers were implemented to generate bridges for general COM components and ActiveX controls in particular to be used from Java/Eclipse.

COM

The Component Object Model (COM) is currently the most widely used model for reusing components throughout the Windows platform. This might change once its designated successor .NET is widely established and broadly used. But for now most components on Windows are available as COM components.

The interfaces and types for a COM component are defined in a type library (as a binary standard). These are usually

generated from an (M)IDL description file. An eclipse-plugin to recreate the IDL-file from the binary type library was developed in this project as a side-product.

Stubs for e.g. C/C++ can be generated from the IDL-file, so that the components of the type library can be used in those languages. Scripting languages like Visual Basic can also explore and use COM components dynamically.

Eclipse as execution-environment for COM components

The Eclipse Rich Client Platform runs on the Java Platform. Hence direct interaction with the underlying system and its components is not possible.

Although running on the Java platform, Eclipse generally provides means to embed an ActiveX control. But up to now the code to access the component model of the ActiveX control had to be manually implemented. To automate this process it was necessary to implement a library that allows interaction with the COM system. A configurable provider, that gives access to the installed components, was developed in particular. It encapsulates the native code required. The provider can of course be implemented natively using JNI, but as the SWT (which is a part of Eclipse) already contains the required functionality, a SWT provider was realized.

II. THE MODEL TRANSFORMATION FRAMEWORK

In order not to be restricted to COM as input and Java as the target for the generated bridge, a more general purpose model transformation framework was developed.

It allows to transform arbitrary models given by model providers to target models used by model consumers, as long as a chain of transformations by model transformers can be derived. All possible chains of transformation are automatically inferred from the registered model transformers, so that the user only needs to choose which chain to use. Model providers are always at the beginning of a chain of transformations. They provide a certain model. The model can be created by arbitrary means, it does not refer to a meta model. Model transformers accept a model from a given set of model classes. This model is then transformed to an instance of another model class. Model consumers finally consume a model and create some kind of output.

Transformation wizard

The different parts of the transformation framework are automatically composed for the usage in the import wizard

of Eclipse. The framework therefore first lets the user choose a model provider from the registered ones. Next, the possible chains of transformation and the matching model consumers are displayed for the user to choose from.

Once the complete transformation has been selected, each part of the chain is configured if necessary and executed. Configuration of the parts is very flexible, as the parts can provide their own wizard pages. In order to simplify creation of "filter transformations", general purpose SelectionWizardPages are provided.

III. BUILDING BRIDGES FROM ECLIPSE TO COM

To be able to use COM components from Eclipse, Java wrappers have to be created for the elements of the COM type library. This is realized by first providing a model for COM components in the model transformation framework, which is then transformed and finally consumed by a Java bridge generator.

In the following sections we will show the most challenging tasks that needed to be solved to achieve this.

Communication

In order to allow communication between Java and COM, an abstract core system (comcore) was implemented. It allows instantiation of components and interaction with them through abstract classes COMUtils and COMFactory. A concrete implementation is realised based on Eclipse's SWT implementation, which already includes basic functionality to communicate with COM components. Other implementations, e.g. based on AWT are conceivable.

Type mapping

For the components to be usable from Java, a mapping of data structures and types had to be realized, so that information could be exchanged between the systems.

In COM, all "simple" types are wrapped inside a Variant. A Variant can therefore be an integer, a real number, a date, a color, and so on. These types can be mapped to standard Java types, e.g. int, float, java.util.Calendar and a specifically created OleColor-Class.

For complex types, that are represented as classes and interfaces in COM, matching classes are created in Java. COM enumerations are matched with a subclass of a dedicated abstract class that provides type-safe constants in Java 1.4 style.

Parameters

While normal (in-) parameters to functions can be easily mapped, in COM parameters can also be declared to be out-parameters. These can be modified by the component and must be passed back to the Java environment. Because primitive types cannot be modified (as seen from the caller) when passed in as parameters in Java, these parameters are mapped to an array of the type. When an array is passed as parameter, the values within the array can be modified, and the modification is also visible to the caller.

COM also allows for optional parameters. In order to compensate for this, multiple methods are created in the generated Java code.

Events

Besides the method invocation that is initiated from the client to the server (the component), it is also possible that the server needs to invoke methods on the client. This is used when the server wants to notify the client of events.

In order for the COM server to call methods on the client-side, the client registers for these events on component creation. This is automatically done by the generated code. Also Listener-, Adapter- and Event-classes are created, so that classes can easily be written to use this functionality.

Parameter mapping is in principle done the same way as when invoking methods the other way around. However, the parameters are not directly passed to the Listener, but embedded as attributes of the Event-class.

Resource Handling

Because COM employs a cooperative memory management, users must increment a reference counter when starting to use a component, and decrement it, when the component is no longer needed.

The created wrapper classes try to manage this as far as possible by themselves. The reference is increased on construction. Also there is a dispose method to deallocate the component.

But because Java does automatic memory management, usually no explicit destruction of an object is done. Objects are destructed only when the garbage collector runs. At that time each reclaimed object's finalize method gets invoked and the component can be disposed of.

If however the Java Virtual Machine is shut-down, no destruction of the objects is done. In order to prevent memory leaks for components running outside the current process's memory space, it is necessary to employ a shut-down-hook to be able to release the components.

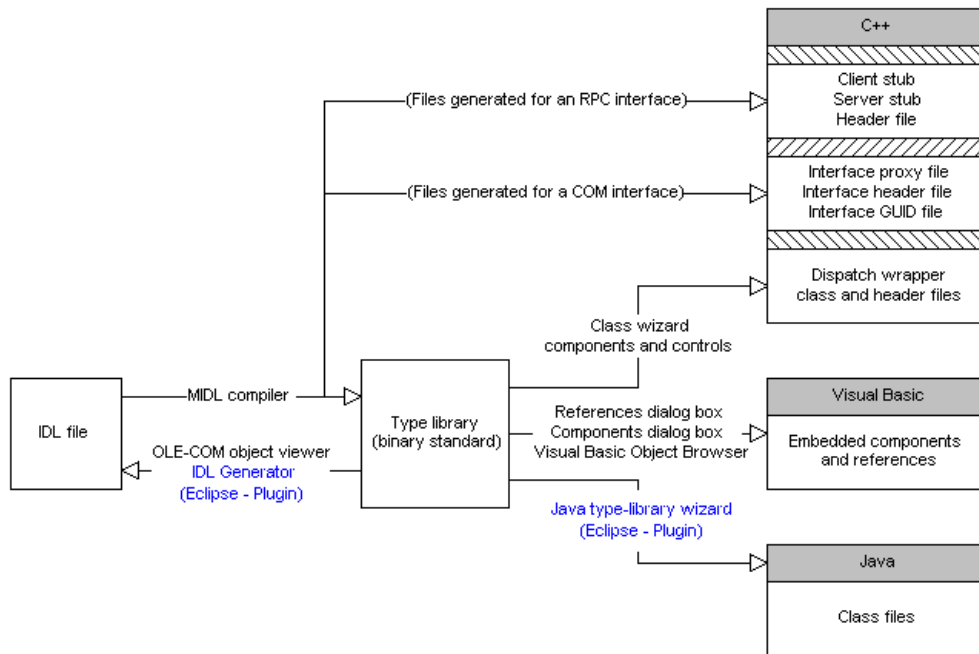
Although this technique works well, we recommend to explicitly dispose of no longer used components for efficiency reasons and for a clear separation of responsibilities.

Reading type-libraries

In order to first find COM components and then read the associated type libraries, the Windows API must be employed. This is done by a native library written in Delphi. It allows to read information about the COM components from the Windows registry and the referenced binary files.

From the gathered information a model for the type library is derived. This model can then be transformed and finally consumed in order to create a Java bridge, but e.g. also to recreate an IDL file.

The following figure shows the general usage of the type library and the newly created use cases:



Generating code

The code finally is generated using the Java Emitting Templates (JET) from Eclipse. The code generation is highly configurable using name generators. This way generated code can seamlessly be integrated in projects without breaking coding conventions.

Furthermore the generated classes include automatically generated Javadoc-comments. These are automatically extracted from the type library and the associated help-strings. This way the created classes blend in the usual coding style, and the documentation is also readily available right inside the IDE.

The generated code depends only on the comcore library and is therefore independent of Eclipse.

For each element of a component two classes are generated, an internal that manages the communication and a public class that optionally provides user defined functionality. A regeneration of the bridge only recreates the internal classes, thus the functionality is kept.

For graphical COM components (ActiveX controls) a SWT component is automatically created as a wrapper for the control. That allows seamless integration of the ActiveX controls within SWT applications.

IV. EVALUATION AND CASE STUDIES

The bridge generator has been successfully used to create bridges to numerous COM-components. These include: Excel, Word, XMetaL, Internet Explorer, MSHTML, Nero Burning Rom etc., see table 1.

A bridge to the XMetaL component, that was previously implemented manually by Docufy within about 2 man-months, can now be generated within seconds with better documentation and (probably) less errors. The potential reduction of costs is therefore enormous.

V. CONCLUSION AND FUTURE WORK

We have developed a framework to create bridges between different component models and implemented a bridge from the Java platform to the binary COM model. This bridge enables the developer to use and integrate well established, extensively tested, widely known components within a Java application as if they were native Java components.

While the development of these tools is already considered a great success, there are some points where future work can be useful:

- Embedding ActiveX-Control in AWT
While being able to almost automatically embed ActiveX-Controls in SWT applications is already very useful, it would still be desirable to also be able to do so in AWT to reduce dependency on external libraries. This can be done, but requires natively implemented helpers, because the necessary tasks cannot be achieved with standard Java.
- Integration of .NET
The Component Object Model has recently been superseded by the .NET framework. While currently all major components are still available as COM components and will remain so for quite some time, embedding components of a .NET architecture in Java is the next logical straight forward step.
When specific .NET components are defined in a future version, the easiest way of transformation probably will be to wrap the .NET components in COM components. This can most easily be done by the author of the component, because here it only requires adding a special marking to the class that instructs the compiler to create the necessary type library.
If this is not possible, e.g. because the source code is

TABLE I

CASE STUDIES: TIMES WERE MEASURED ON A PIV DUAL 3 GHZ WITH 3 GB MEMORY.

Type Library	LOC	Classes	Methods	Time
XMetaL Editor	13597	183	1281	1 sec
MS Word 11.0 Object Library	121056	1239	12484	9 sec
MS Excel 11.0 Object Library	222756	1613	22791	11 sec
MS HTML Object Library 4.0	525597	3823	62330	22 sec
Nero 1.4 Type Library	14108	291	1444	1 sec
MS Internet Explorer	12092	121	1148	1 sec

not available, a new .NET component can be created that is derived from the desired component. This newly created component can then once again be attributed to be available as COM-component, too.

REFERENCES

- [1] Eclipse Corner Article: ActiveX Support In SWT. www.eclipse.org/articles/
- [2] <http://www.ezjcom.com/>
- [3] Automatische Einbindung von existierenden COM - Komponenten in Eclipse, Institut für Informatik, Universität Würzburg, Diplomarbeit, 2006
- [4] <http://www.nevaobject.com>
- [5] Microsoft Windows Platform SDK Collection for Windows Server 2003 SP1
- [6] MOF Queries, Views, Transformations. <http://www.omg.org/cgi-bin/doc?ptc/2005-11-01>