Collabora Productivity

Improving LibreOffice as a MSO replacement for Automation and VB Clients: COLEAT

Tor Lillqvist

Software Engineer at Collabora Productivity

About us

Collabora Ltd.

- Leading Open Source Consultancy
- 10 years of experience. 100+ People.

Collabora Productivity Ltd.

- Dedicated to Enterprise LibreOffice
- Provides Level-3 support (code issues) to SUSE LibreOffice clients
- Architects of OpenXML filters in LibreOffice

What problem are we trying to solve

- Customer wants to switch from some old MSO version to Collabora Office
- But has legacy software that connects to MSO using OLE Automation or COM
- Such legacy software might be 3rd-party without support, in-house without sources or without development environment, etc

What is (OLE) Automation

Subset of more generic COM/OLE for scripting languages

- Quoting Wikipedia: “ inter-process communication mechanism created by Microsoft. It is based on a subset of Component Object Model (COM) that was intended for use by scripting languages – originally Visual Basic – but now is used by several languages on Windows”
- The more modern form of what used to be called OLE or COM
- Uses “late binding” (introspection at run-time)
- Typical client code: VBScript, which is included in current Windows. No IDE, and not really for creating GUIs though (I might be wrong)

Sample VBScript code

```
Set w = Wscript.CreateObject("Word.Application")
```

- Note how the class of object created is a string literal. The interpreter has no a priori knowledge about the interfaces that service contains

```
Set d = w.Documents
Set doc = d.Open("C:\foo\u1.odt")
For Each i in doc.Paragraphs
    Set range = i.Range
    nchars = nchars + (range.End-range.Start)
Next
```

- That Documents property, Open call, Paragraphs property, Range property etc are all looked up using “introspection” at run-time

What is (old-fashioned) OLE or COM then

Similar, but not the same

- Uses compile-time binding; binaries have hardcoded knowledge of interface structure, what methods, at what vtable offsets, etc
- Typical client code: Visual Basic, or C++
  - Rumours of VB's death exaggerated
  - Microsoft considers it dead, sure
  - But probably still used a lot, and a lot of applications written in it in use
  - Last release was VB6, in 1998 [sic]
  - Even sane people who used it claim VB6 "was very good". (But maybe they have forgotten the actual horrors...)

Sample VB6 code

```
Set w = New Word.Application
```

- Note how the class of the object created is specified as an identifier, not a string. The compiler must have access to the Word type library at compilation time

```
Set doc = w.Documents.Open("C:\foo\u1.odt",False,True)
For Each i in doc.Paragraphs
    Set range = i.Range
    nchars = nchars + (range.End-range.Start)
Next
```

- Not very different from the corresponding VBScript
- But the vtable offset of the Open call, and of the getters and setters for the properties, are fixed at compile time.

VB vs. VBA, then. Sounds similar, same thing?

No.

- VB is a freestanding GUI tool to create GUI programs
  - The code in such programs can communicate with COM servers
  - Lots of “intelligent” features to “reduce typing” when coding, using type libraries of those COM services
  - Various products, including MSO, offers an API through COM
- VBA (Visual Basic for Applications) is the built-in Basic interpreter in Word and Excel
  - Word and Excel documents can contain VBA macros
  - The API offered to such macros is (not surprisingly) very similar to that offered to external COM or OLE Automation clients.

What did LibreOffice already have?

Quite some bits and pieces in the right direction

- Support for OLE Automation clients: the “oleautobridge” library, code in extensions/source/ole
  - Allows OLE Automation clients to use UNO services
- The “option vbasupport” in our Basic interpreter to make the language behave more like VBA. (Not relevant for this work.)
- UNO APIs similar to what MSO offers to VBA code (and also to out-of-process clients) in Word and Excel, in oovbaapi/ooo/vba, ooo::vba namespace.

What needed to be done?

A lot

- Improve the oleautobridge. Many language features easily usable through a VBScript client were broken or missing
- Those legacy “compile-time binding” clients, typically produced by VB6, are not supported at all because obviously LibreOffice doesn’t offer ABI compatible interfaces, at the same GUIDs, etc
  - Some kind of translating tool to be inserted between such a client and the server (LibreOffice) was needed
- Actual clients typically use much of even fairly trivial APIs that we did not have (as part of oovbaapi).

What was done, examples

oleautobridge

- UNO does not have optional parameters. OLE has. That impedance mismatch needed to be fixed
  - Pass incoming missing parameters to UNO as empty Any
- Implement enumerations of collections (DISPID_NEWENUM). Maps to css::container::Xenumeration
- Properties in our “VBA” support were weird. Apparently we use that for stuff like named form controls. COM interfaces on the other hand have properties just like they have methods. (Like UNO interfaces have attributes.)
- Case (in)sensitivity issues.

What was done, more examples

oleautobridge

- OLE Automation expects that the server is able to do callbacks to the client, that is, invoke event handlers in the client code. We had nothing for that
- More parameter number issues: A client might also pass in more parameters than the UNO-defined “VBA” API expects. In that case ignore them as long as they are empty.
- Optional parameters in OLE can also be in the middle of the list and left out
- OLE has named arguments. UNO does not
- For details, look in my commits to extensions/source/ole.

What was done, more examples

oovbaapi

- New stuff relevant only for Automation (or COM) clients, and not related to “VBA” (in our StarBasic) added here anyway. For instance things related to the callbacks (events). Slightly confusing, yes
- Add new properties (UNO attributes) to various interfaces. Might be useful to macros ported from MSO that expect them to be present, too.

What was done, more

sw and sc

- Much largely boilerplate work when adding new properties or methods. Many of the additions have dummy implementations, but are needed, as typical (?) clients wants to use them
- Most complicated was the event callback stuff.

COLEAT

Collabora OLE Automation Tool

- A standalone program that goes between the client and the server (LibreOffice)
- Redirects attempted use of MSO services to LibreOffice instead
- Translates the “compile time binding” of VB6 clients to the OLE Automation style (“late binding”, run-time introspection) that LibreOffice supports
  - To do that, COLEAT needs to be built with access to the type libraries for Word and/or Excel
  - At its build time proxy code is generated from those type libraries.

How COLEAT works

At COLEAT build time: genproxy

- A separate program, in COLEAT sources, that reads type libraries and generates proxy code for the interfaces in them
- Takes various parameters to restrict which interfaces proxies are generated for
- One need to run genproxy and build the actual COLEAT tool separately for each set of type libraries one wants to handle. The default build is a COLEAT for redirecting Word and Excel use to LibreOffice.

How COLEAT works

At COLEAT run time

- Top-level executable coleat.exe
- Takes as parameter some options, and the name of the OLE (Automation or binary OLE) client program to run, the "wrapped process"
- Starts the wrapped process suspended, checks whether it is 32- or 64-bit
- Starts another program that is part of COLEAT, exewrapper. Either exewrapper-x86.exe or exewrapper-x64.exe
- Waits for the exewrapper process to finish.

How COLEAT works

At COLEAT run time: exewrapper

- Gets a handle to the wrapped process from coleat.exe
- Injects a small executable code snippet into it
- That injected code snippet loads a DLL into the wrapped process that does most of the actual work, either exewrapper-x86-injected.dll or exewrapper-x64-injected.dll
- Waits for the wrapped process to finish.

How COLEAT works

At COLEAT run time: the DLL loaded into the wrapped process

- Hooks calls to interesting functions in the wrapped process's main executable, like those that create COM objects: CoCreateInstance and friends
- Also hooks LoadLibrary so that it can hook the same functions in each loaded library
- The COM object creation hook checks if the object class to be created is one of those for which we have generated a proxy, and calls that proxy
- Lots of interesting and complicated details.

How to use COLEAT: other uses

Tracing client process against actual MSO

- With the right option, coleat does not redirect to LibreOffice, but instead traces what calls the client performs, what interfaces it creates, what methods it calls, what properties it gets and sets.
- This is so that one knows what potentially missing API needs to be implemented in LibreOffice to handle this client.

```
Usage: coleat [options] program [arguments...]

  Options:
    -n                          no redirection to replacement app
    -o file                     output file (default: stdout, in new console if necessary)
    -t                          terse trace output
    -v                          verbose logging of internal operation
    -V                          print COLEAT version information
```

Sample trace output

```
new Word.Application -> 0088FAA8
Word._Application<0088FAA8>.Visible = True
Word._Application<0088FAA8>.Documents -> 008A2858
Word.Documents<008A2858>.Open("C:\\cygwin64\\home\\Tor\\lo\\
u1.odt")
    Word.ApplicationEvents4.DocumentOpen
 -> 008A2258
Word._Application<0088FAA8>.WordBasic -> 008A22D8
Word.?<008A22D8>.32827() -> "u1.odt"
Word._Application<0088FAA8>.ActiveWindow -> 008A2718
Word.Window<008A2718>.Caption = "Foo Bar"
Word._Application<0088FAA8>.Documents -> 008A2398
Word.Paragraphs<008A2558>._NewEnum -> 008A2798
Enum<008A2798>.Next(1): S_OK
... 0: 008A958C
```

Where to find COLEAT

- On GitHub: https://github.com/CollaboraOnline/COLEAT

Collabora Productivity

Thank you

... and Keep Calm and Crush the Patriarchy

Tor Lillqvist

tml@collabora.com