A Generic SICStus Prolog COM Interface

By

Mamdouh Farouk

Ahmed Fouad

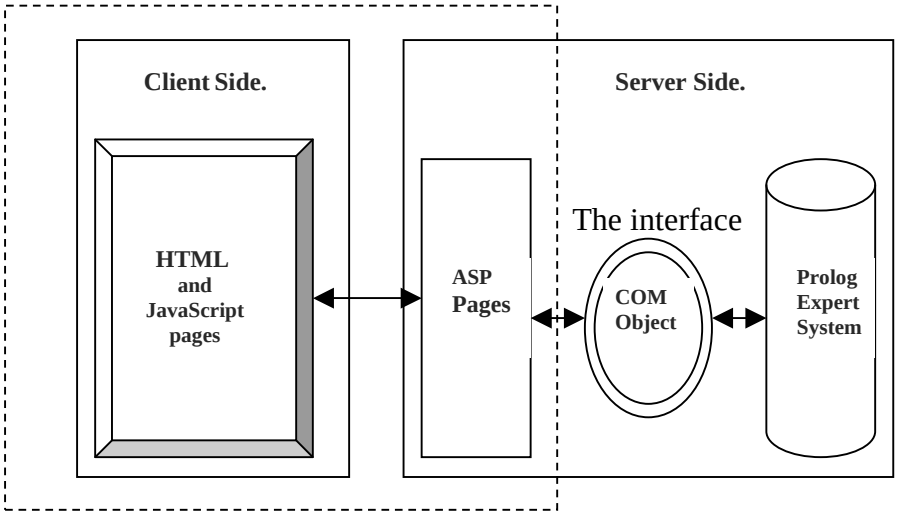
Dr.mahmoud Rafea

May, 2004

Table of Contents

Introduction

The Generic SICStus Prolog COM Interface is a bi-directional interface that acts as a wrapper to SICStus Prolog. Through this wrapper, any language that can provide access to a COM object can access SICStus Prolog. Thus, (ASP, Delphi, visual basic, … etc) applications can interact with SICStus Prolog programs. The motivation of this work was to allow ASP applications to interact with Prolog applications (something that is not currently directly supported) as there was a need to avail a number of agricultural expert systems implemented in SICStus Prolog, on the web. By using the developed interface, any Prolog program can be made available to the Internet users in an easy manner. The following figure shows the structure of the developed interface.



A COM object works as a bidirectional communication channel between the prolog system and the web application. The above figure clarifies the structure of the interface between ASP and Prolog. Note that all work is done at the server side and nothing is downloaded to the client.

Tow examples attached to show the power and usage of this interface. One example is designed to access prolog from web and the other is designed to show how VB program can bi-directionally integrated with Prolog.

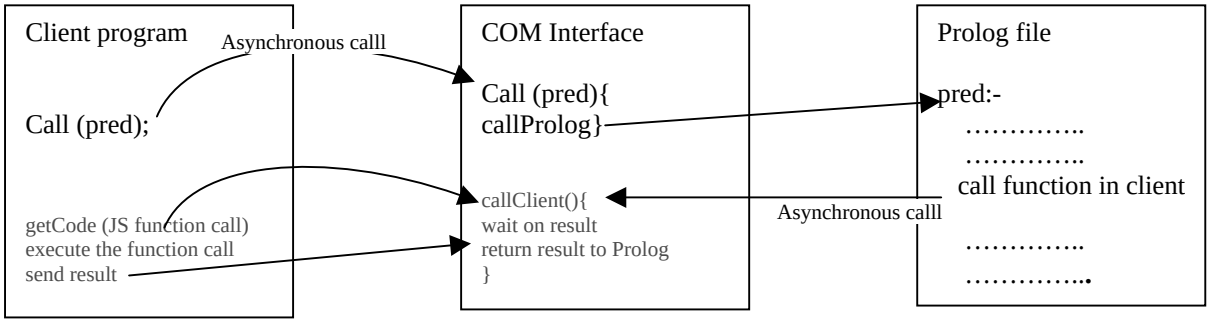
Implementation Problems

This section discusses some important implementation problems that influence the current design of the generic COM Interface. These problems are**:**

- **The already available Prolog code should not be changed**
- **The Multi-user Access Problem**

- **The already available Prolog code should not be changed**

In the avalilable Prolog code the data inputs are collected from user when needed. This is done through calls to user-interface code. Consequently, ASP application should support bidirectional interface. Further, the script (VBS or JS) code cannot be executed directly from a Prolog call because the script code should be executed at the client side. This can be implemented through coding of two C++ functions. The first call Prolog predicates while the second returns back with script function call that is passed from Prolog. This necessitates that the Prolog calling should be asynchronous, which is implemented throug creating a thread for this call.



Client: calls prolog predicate P asynchronously and does not wait on P

P: get information from Client synchronously and waits on W.

In the above figure, an ASP page calls a prolog predicate which contains a callback to the Client program to get some data from the user. The Prolog predicate waits until the desired data is receieved. The Client will send data to the prolog.

Notice that the caller (ASP) continues execution so that it can handle the callback (if any).It should be stressed that the Scripit code is indirectly called through a

notification message that is passed from Prolog to a function in the COM Interface that returns with the received notification message

✓ **The multi-user access problem:**

This problem can occur in a multi-user environment such as web environment; when more than one user wants to access a prolog application at the same time. Prolog run time is loaded once for all users. When more than one user access at the same time this cause conflict in their sessions.

To overcome this problem we can use one of the following solutions:

1. Multiple SICStus runtimes.
2. Implement access control.
3. Multiple virtual applications.

What is implemented now is access control, because it is easy to implement and sufficient enough for our prolog applications[1]. But in general, access control is not an efficient solution. The optimal solution is to provide multiple SICStus runtimes such that when a user wants to access a Prolog program, a new instance of SICStus run time is created and when the user session ends the instance is deleted (under consideration).

User manual

The COM object provides an interface that lets you load and call SICStus Prolog programs from a client program and allows the Prolog program to callback the client program.

The control structure of this interface is rather similar to that of the foreign language interface. The queries to be passed to Prolog have to be given as strings on the client program and the terms outputted by Prolog are received as strings in client variables.

The interface provides functions for:

- Passing a query to Prolog

[1] Our programs are considered stateless (have no state).

- Retrieving a result value (string) of the Prolog query
- Passing a query to a client from Prolog
- Passing the result to Prolog

The full description of the interface functions provides at Appendix in the end of the document.

How to use the interface

- **Setting up the interface**

The interface consists of the following files:

• 'web.dll'

• 'webscript.pl'

In order to use the interface, perform the following steps:

1- Register the web.dll file. Note, the sprt310.dll must be in the same directory with web.dll to achieve successful registration.
   Ex. Regsvr32 C:\interface\web.dll

2- Include the file 'webscript.pl' in your Prolog program.

- **Initializing the Prolog engine from the client**

  After creating an object from COM interface (web.interact) you can call *init(BSTR bootPath ,BSTR plFile):* this method is used to initialize SICStus Prolog run time and to load the pl file specified in argument two (*plFile*).

  Note `bootPath` is the SICStus Prolog installation directory.

Example: Hello world

This is a very simple example in which we will try to show 'Hello world' string from a pl file on the web page.

ASP page

```
<% @Language = "VBScript" %>
<%
set webObj=Server.CreateObject("web.interact")
```

```
webObj.init "C:/SICStusProlog3.10.1/bin","D:\moh.pl"
webObj.callPred "test(Obj)."
webObj.getResult()
%>
```

%pl file

```
:-use_module(library(webscript)).
test(Obj):-
   sendToClient(Obj,"<body bgcolor=blue> <H1><font
color=red> Hallo world </H1></font> </body>").
```

In the ASP page, we call the test/1 predicate in line 5. Because the test predicate contains html to be shown on the client, we should call the getResult method as in line 6. Note, when calling the getResult method, the component will show the Prolog result on the page directly (this is done through the output buffer of the asp page).

The first line in the pl file is for enabling the use of the functionality of the module webscript. The pl file contains only one predicate 'test/1' which calls the `sendToClient`/2' predicate, `sendToClient`/2 take script code to be show in the client as a second argument.

Appendix

Passing a query to Prolog

Prolog queries are represented in the client program in a textual form, i.e. as a string containing the query followed by a fullstop.

Prolog can be queried synchronously using the following function:
callPred (BSTR goal): where *goal* is the predicate to be called

Prolog can be queried asynchronously using the following function:
setPredName (BSTR goal): (Asynchronous call)

passing a query to the client from prolog

There are two ways to call the client from Prolog.

1- *callscript/3*: this predicate takes 3 arguments. The first is the object[2] passed from the caller, the second is the function that should be executed on the client side, and the third is the return value in string format (chars list).

2- *sendToClient/2*: this predicate send a data (string) from Prolog to the client program. First argument is the object, and the second is the string to be sending to the client.

Retrieving a value (string) result of the Prolog query (in client)

BSTR getResult(): use this function to get result of Prolog query. This method should be called when the programmer know that Prolog will send the result to the client side.

Passing a result to Prolog (in client)

setResultToPl (BSTR resl): This method used to send a result to Prolog. Use this method when Prolog program calls callscript/3 predicate, which need a return value from client.

[2] This object created in the COM interface and pass to the prolog to send back on the same object.