Poseidon House
Castle Park
Cambridge CB3 0RD
United Kingdom

TELEPHONE: **Cambridge (01223) 515010**
INTERNATIONAL: **+44 1223 515010**
FAX: **+44 1223 359779**
E-MAIL: **apm@ansa.co.uk**

ANSA Phase III

A CORBA IDL Compiler for the World Wide Web

Nigel Edwards

Abstract

This presentation looks at the use of CORBA technology to extend the functionality of the World Wide Web. Current web technology makes accessing internet resources extremely easy; creating and managing these resources is much, much harder. The objective of this work is: to make it easier to manage and create new resources; and to make it easier to extend the functionality of the web.

A commercial application which uses current web technology is described; the stub compiler would have made building the application much easier — it provides a programming model for the web which is very close to CORBA.

The engineering put in place by the stub compiler and its associated libraries are described. The architecture of the Sun public domain CORBA IDL compiler and the lessons learned writing a back-end are also discussed.

An for a fully extensible web server based on the above technology is proposed— something which brings the web even closer to the CORBA model. Finally, opportunities for CORBA/Web interworking are discussed.

APM.1419.01 **Approved** 23rd February 1995

Project Management (confidential to ANSA consortium for 2 years)

Distribution:
Supersedes:
Superseded by:

A CORBA IDL compiler for the Web

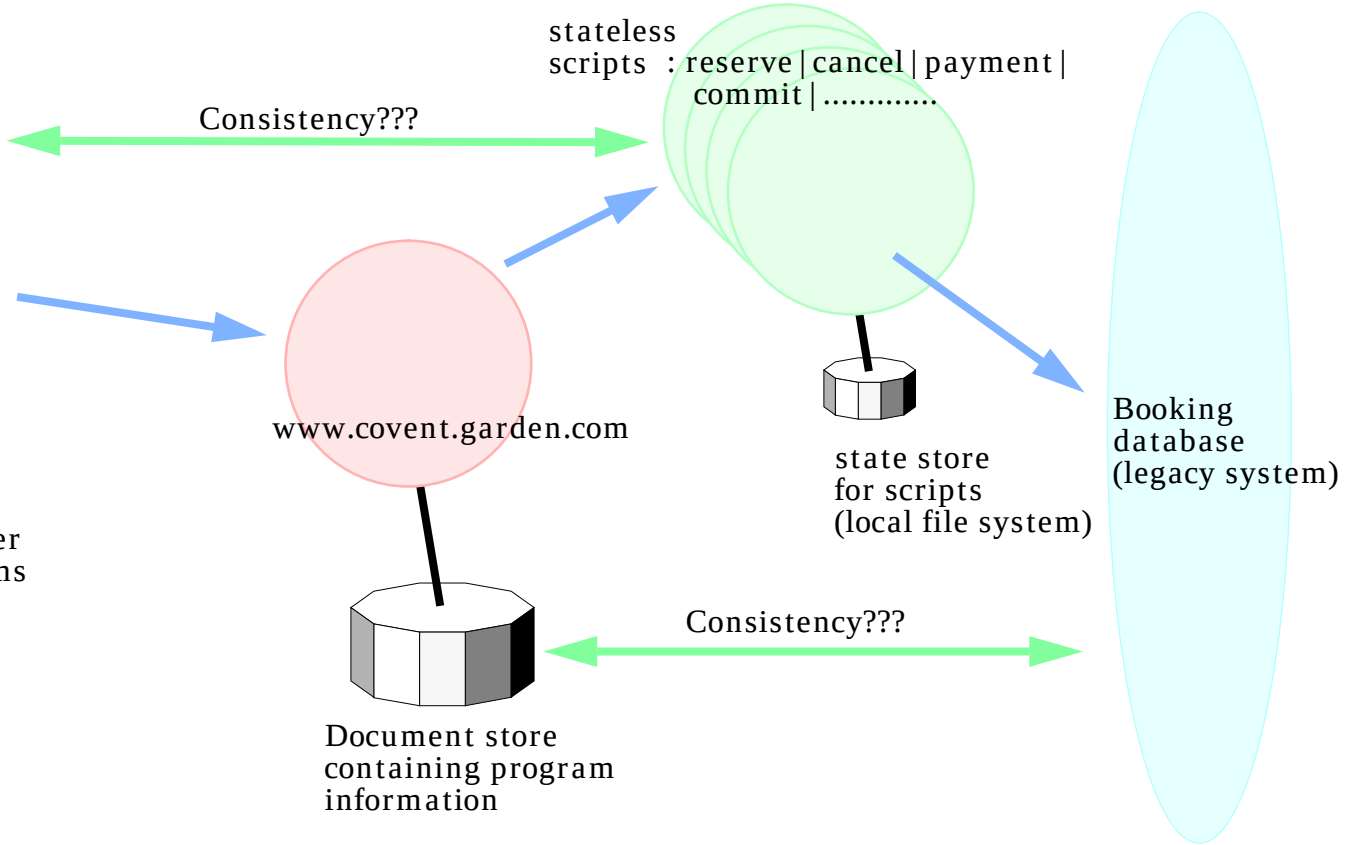
Nigel Edwards

Introduction

- Accessing information in the web is very easy
- Managing information and extending the functionality of the web is very difficult — for commercial applications solving this is crucial
- This presentation looks at:
  - A “commercial application” using current web technology
  - How the stub compiler would make it easier to build this application
  - The engineering put in place by the stub compiler and its libraries
  - The architecture of the stub compiler (Sun public domain front-end)
  - The lessons learned writing a back-end for the web
  - How to evolve the web towards the CORBA model
  - Some thoughts on CORBA - web interworking

A “commercial” application

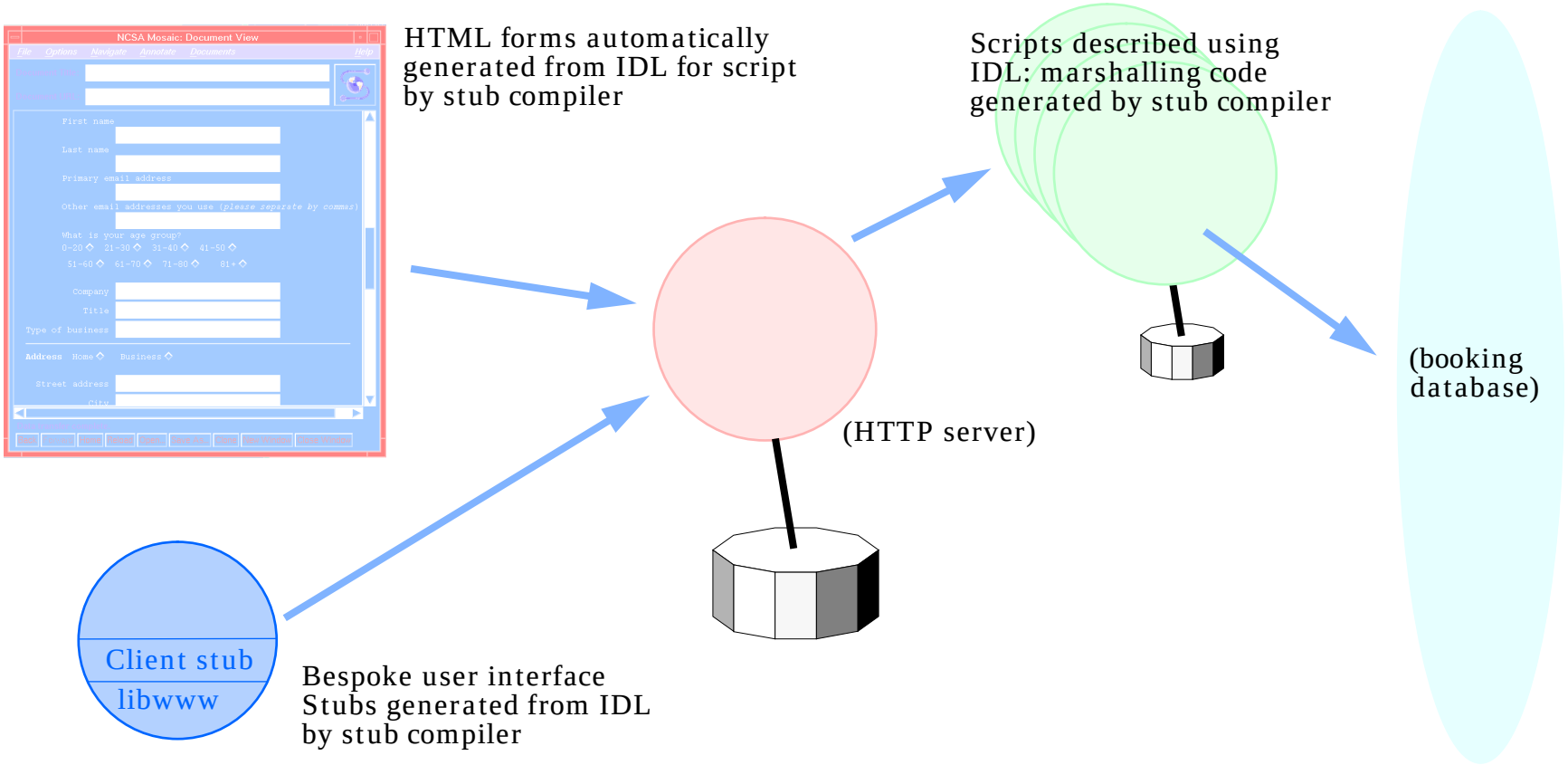
User interface: a web browser (e.g. Mosaic) using html forms



Notes for previous slide (not for presentation)

- Allowing customers to browse marketing material on forthcoming attractions would be easy — the web is well suited to this.
- Extending the functionality to allow bookings (possibly for multiple performances) is much more difficult — the only way of doing it now is to use scripts.
- Creating and managing the marketing material would be difficult (scripts which are external to the server would be needed to keep the document store & database consistent).
- Even though HTTP has “PUT” and “POST” methods for creating and replacing resources, these are usually implemented by scripts external to the server, because the precise semantics of creation etc. is usually very dependent on the local environment. It is harder to implement with a “one-size” fits all implementation inside the server itself.
- It is hard to write the code to unmarshal the parameters to the scripts.
- A script gets “forked” each time it is invoked, so state has to be stored externally.
- Scripts are driven by forms technology; no notion of bespoke client application.
- Effort is required to keep scripts and the html forms which drive them consistent with each other.

New features



A simple example — the ANSAware echo service

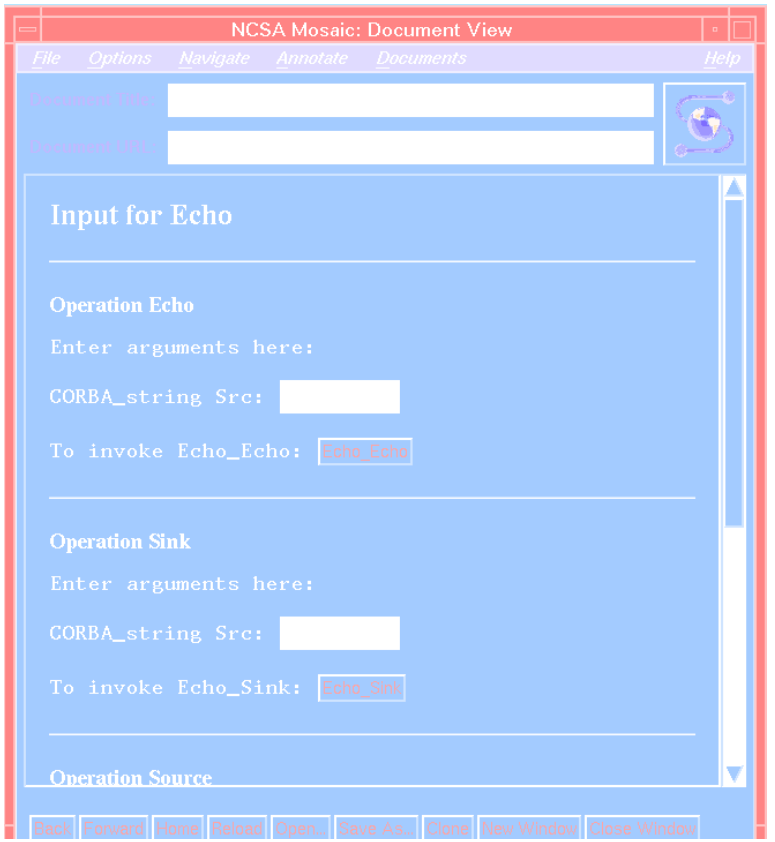
```
interface Echo{
  string Echo(in string Src);
  void Sink(in string Src);
  string Source(in long Length);
  string Reverse(in string Src);
};
```

```
#include <Stub.h>
PUBLIC char * HTAppName = “Echo server”; /* Application name */
PUBLIC char * HTAppVersion = “0”;  /* Application version */
static char buffer[32000];
CORBA_string Echo_Echo(CORBA_string Src){
        return Src;
}
CORBA_void Echo_Sink(CORBA_string Src){
        return;
}

CORBA_string Echo_Source(CORBA_long Length){
  register char *sp;
  for (sp = buffer; Length-- > 0; )
    *sp++ = ‘a’;
  *sp = ‘\0’;
  return buffer;
}

CORBA_string Echo_Reverse(CORBA_string Src){
  register char *sp;
  int n;
  for (sp = buffer, n = strlen(Src) - 1; n >= 0; n--)
    *sp++ = Src[n];
  *sp = ‘\0’;
  return buffer;
}
```

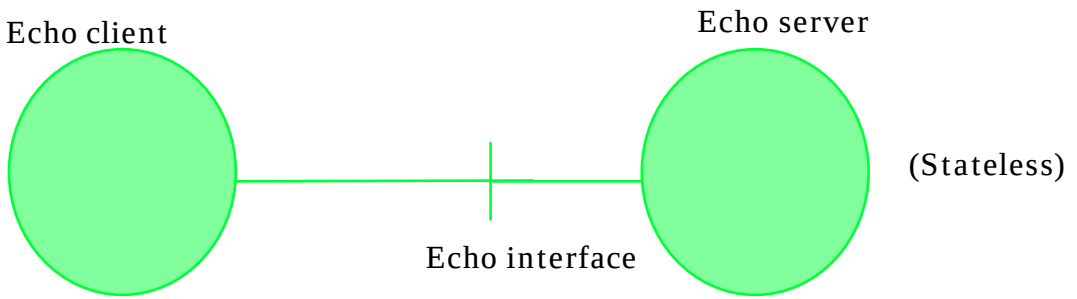
The Echo client — a bespoke web client program

```
#include <Stub.h>
#include <CEcho.h>
PUBLIC char * HTAppName = "Echo client"; /* Application name */
PUBLIC char * HTAppVersion = "0";  /* Application version */
CORBA_string obuf;
char ibuf[1024];

int main(int argc, char* argv[]){
  Echo ref = NULL;
  int nbytes = 1000;
  int i, n;
  char* p;
  /*Argument processing deleted*/
  ref = Stub_bind("http://socrates.ansa.co.uk:8080/cgi-bin/Echo");
  printf("> "); fflush(stdout);
  while(n = read(0, (void *) ibuf, sizeof(ibuf))) {
    ibuf[n] = '\0';
    obuf = Echo_Echo(ref, ibuf);
    printf("%s", obuf);
    printf("> ");
    fflush(stdout);
  }
  /*remainder of program deleted*/
}
```

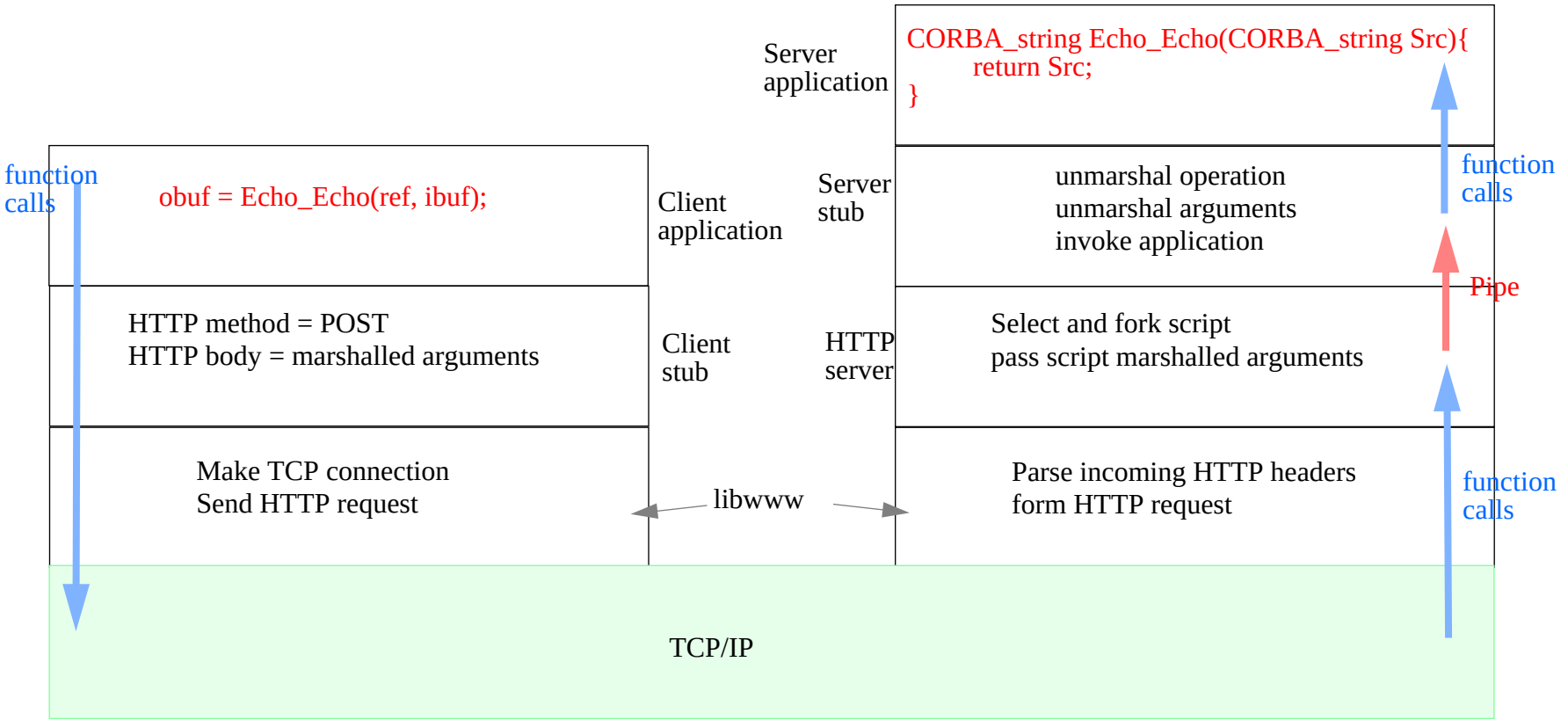
The Echo html form automatically generated for browsers



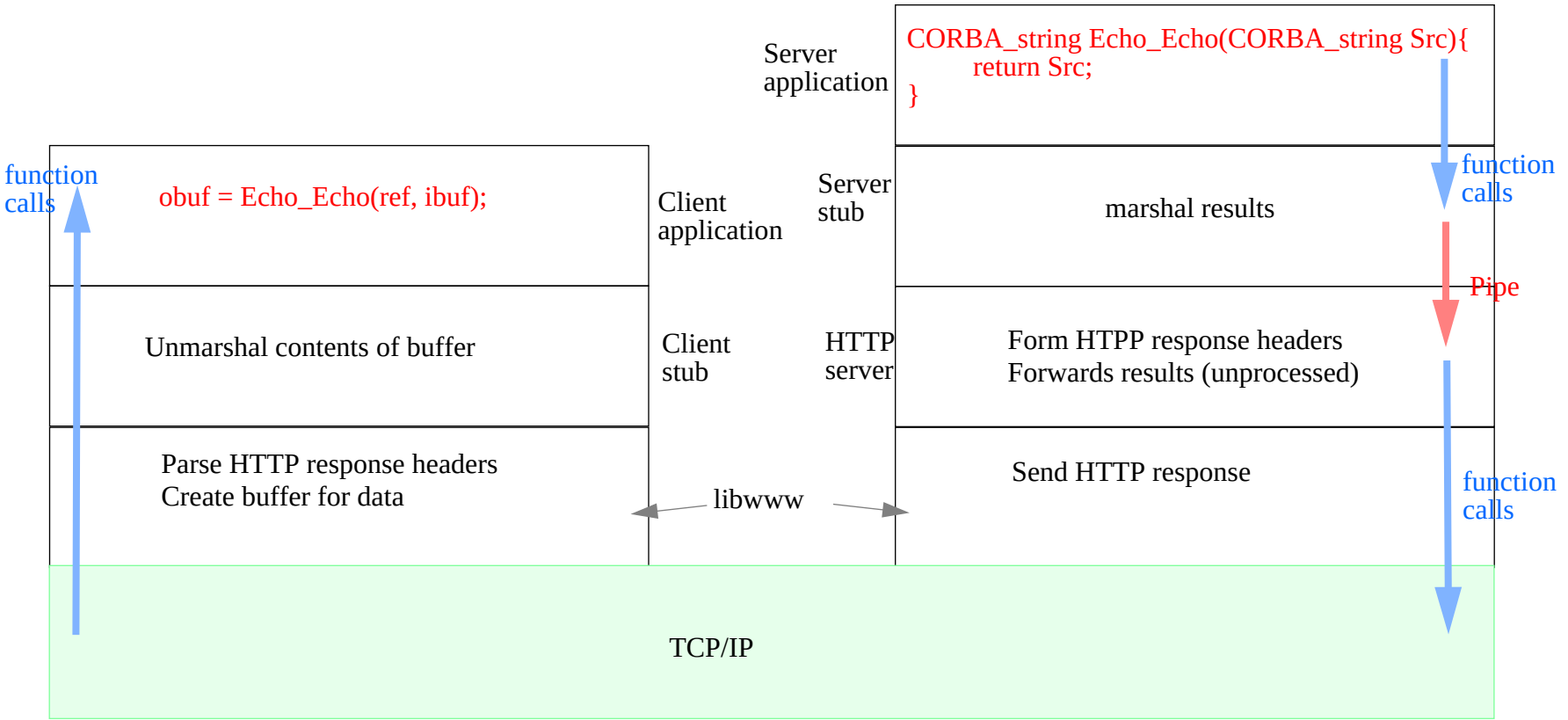
The computational view for the application programmer



How the engineering supports an invocation NOW

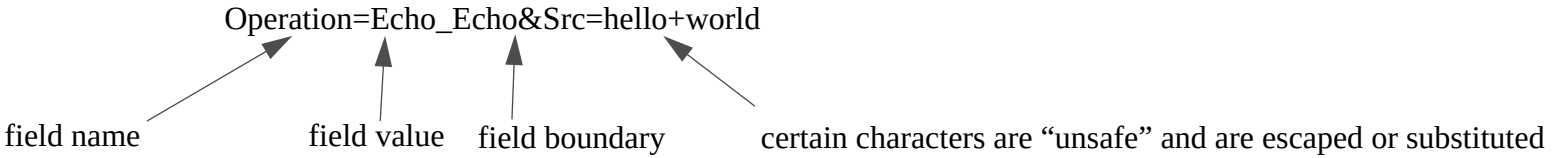


How the engineering supports a Termination NOW

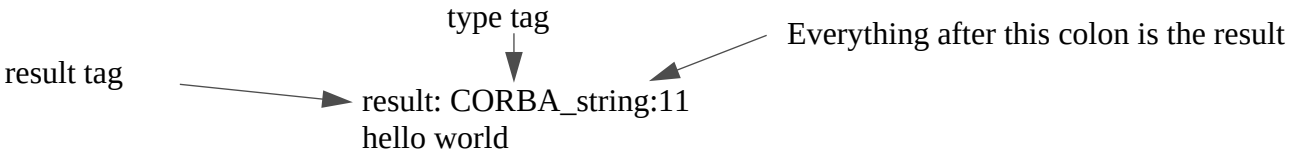


“On-the-wire” Format for invocations and terminations

- For “obuf = Echo_Echo(ref, ibuf);”
- The request is in the format that browsers send when processing forms:



- The response is more human readable so it can be wrapped in html



- Server stubs look at request headers and wrap response in html if the client is a browser (i.e. wants html rather than HTCorba)
- Two formats doubles the number of marshalling routines

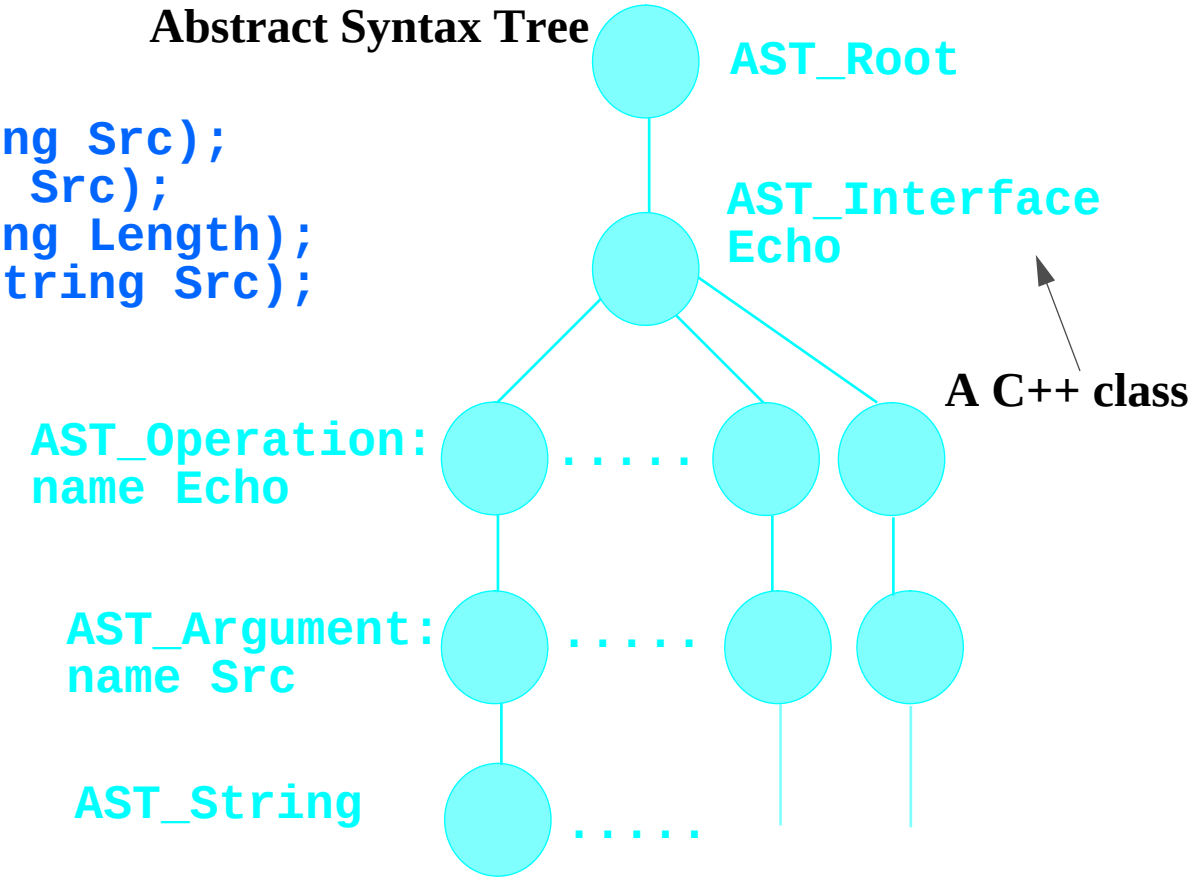
The basic principles of the Sun IDL compiler “front-end”

IDL file

```
interface Echo{
  string Echo(in string Src);
  void Sink(in string Src);
  string Source(in long Length);
  string Reverse(in string Src);
};
```

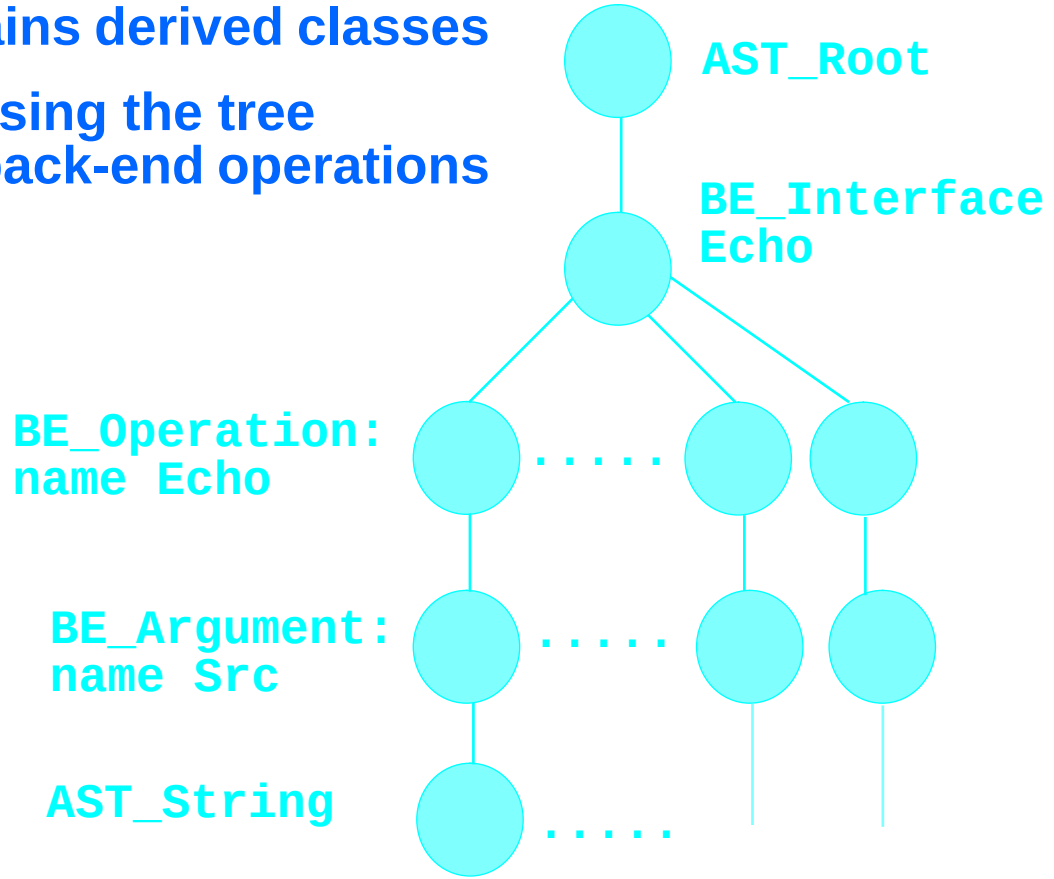
Parser
(Yacc grammar
+ lex specification

Abstract Syntax Tree



Generating stubs by sub-classing AST classes

- Abstract syntax tree contains derived classes
- Create the stubs by traversing the tree invoking the appropriate back-end operations



Derived from AST_Operation new operations to dump stubs

Lessons learned

- Quite a steep learning curve (for me)
  - complex inheritance hierarchy
  - four years since I last wrote any C++ & no compiler writing experience
- Easy to write a back-end once the basic concepts are mastered
- Decision to generate C code caused some minor difficulties where C and CORBA are not well aligned (C is more mature and more widely accepted in the Web community than C++)
  - C cannot return arrays from operations
  - There is no first class entity in C analogous to CORBA's sequences
- Marshalling everything as ascii and escaping certain characters will hurt performance — but performance is not a priority at present
- Separate browser and program marshalling routines

Current status

- **Some limitations**
  - **CORBA Types “Exception”, “Union” and “Any” not supported**
  - **Stateless servers**
  - **No threads**
  - **“Interfaces” are URLs so no location transparency or relocation**
- **Size of sources (not including .hh, .h and .idl files):**
  - **Back-end 5000 lines of C++**
  - **Stub library 2000 lines of C**
  - **Example and test applications 1500 lines of C**

What next — a fully extensible stateful HTTP server?

Browser

stateless
scripts : PUT | POST | reserve | cancel | payment | commit | .....

state store
for scripts

New application

Document store

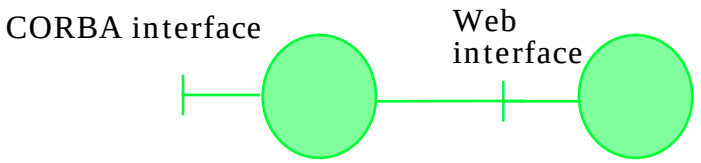
Move these scripts into the HTTP server as new HTTP methods

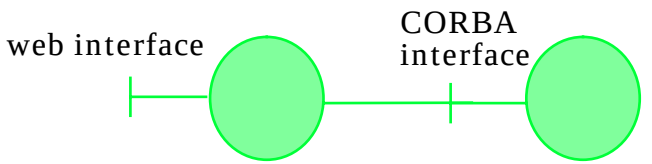
Extensible HTTP server — requirements and benefits

- **Requirements**
  - Mapping of HTTP methods (GET|HEAD|PUT|POST|DELETE|LINK|UNLINK) to CORBA IDL
  - Some redesign and re-implementation of current (CERN) HTTP server
  - Modifications to stub compiler back-end
- **Benefits**
  - Extensibility for management as well as new services and applications.
  - Extending the functionality of a Web server and implementing standard HTTP methods (like PUT and POST) will be exactly like extending the functionality of a CORBA server (note could still use scripts if you want to)
  - Improved performance
  - Can choose between stateless and stateful interaction between client and web server
  - Easier to write new applications

Some thoughts on interworking

- **Two experiments:**
  - **Write CORBA (e.g. Orbix or ANSAware) server which is also a web client (Hence CORBA to web gateway)**
  - **Write web server which is also a CORBA (e.g. Orbix or ANSAware) client (Hence web to CORBA gateway).**





- **Some likely difficulties:**
  - **Can the two protocol stacks exist side by side (even in the limited fashion in which a server is not required to be both a CORBA and a web server)**
  - **The interception problem — mapping between the type representations.**

Summary

- Motivation (1): build extensible services in the internet — extensibility is essential for market differentiation and manageability
- Motivation (2): examine the relationship between WWW and CORBA
- Result: have been able to provide a web programming environment which is close to CORBA by using stub compiler technology
- Need to map HTTP to CORBA IDL (on going) and redesign HTTP server to move the model even closer to (fully consistent with?) CORBA
- Hopefully we will soon be able to perform some interworking experiments.