Achieving Language Interoperability with Babel

Gary Kumfert
with
Bill Bosl, Tammy Dahlgren, Tom Epperly, Scott Kohn, & Steve Smith
Babel’s Scope & CCA

Babel provides language interoperability, not components.

We collaborate with CCA to add parallel distributed support.

We also provide tools (Quorum & Alexandria) to facilitate component development and deployment.
Release Announcement

Babel

Beta 0.5

C, C++, F77, Python(client)
babel-0.5.0.tar.gz

Babel code generator
written in Java

Babel runtime library
written in ANSI C

Docs (minimal)
papers, talks, javadoc html
babel101 tutorial
Hand Coded Language Interoperability

 JNI
 Native
 SWIG
 Platform Dependent
Babel Enabled Language Interoperability

- Truly Object Oriented
- Reference Counting
- Exception Handling
- RMI (future)

Languages:
- C
- C++
- f77
- f90
- Python
- Java
Babel Enabled
Language Interoperability

What’s In This Release:

- C
- C++
- f77
- f90
- Python
- Java

Soon!!!
Babel Has Two Types of Customers

<table>
<thead>
<tr>
<th>Developers</th>
<th>Users</th>
</tr>
</thead>
<tbody>
<tr>
<td>Have a code</td>
<td>Have a problem</td>
</tr>
<tr>
<td>Want to increase their user base</td>
<td>Want to solve their problem</td>
</tr>
<tr>
<td>Will learn SIDL</td>
<td>May never see SIDL</td>
</tr>
<tr>
<td>Want Babel general and powerful</td>
<td>Want software that’s easy &amp; trustworthy</td>
</tr>
</tbody>
</table>
Babel’s Design Priorities

Performance
Developer/User dichotomy

What’s natural for each language?

Could expose C array structs in C++
C++ style would be SIDL::array<T>

```
template <>
array<item_t> : public array_mixin
  < array_t, item_t,
    item_cxx_wrapper_t>  { }
```
SIDL (Scientific Interface Definition Language)

Builds on Industry IDL technology
   CORBA
   COM
Designed for Scientific Apps
   complex types
   dynamic multidimensional arrays

version Hello 1.0;

package Hello {
   class World {
      string getMsg();
   }
}
version MySolverLib 0.1.0;

import ESI;

package MySolverLib {
    interface MatrixGenerator { ... } 
    class OptionDatabase {
        void option( in string name, 
                     out string val);
    }
    class Vector implements-all ESI.Vector {
        void setOptions( in OptionDatabase db );
    }
    class Bizarre implements MatrixGenerator {
        ... 
        void setData( in array<dcomplex,2> a );
    }
}
Many forms of language interoperable interfaces

**Human Compatible**

**SIDL**
Scientific Interface Definition Language

**IOR**
Internal Object Representation

**Web Compatible**

**XML**
eXtensible Markup Language

**BABE**

XML enables...

...Type Descriptions on Shared Repositories

...automated creation via higher-level tools

...automated search & discovery by advanced builders
Language Interoperability: How Babel Makes it Work

Application: user’s LOP (Language Of Preference)

If developer used Babel, they also have a SIDL file.

Implementation: developers LOP (Can be wrappers to legacy code)
Language Interoperability: How Babel Makes it Work

- Application: user’s LOP (Language Of Preference)
- Client Side Stubs: user’s LOP to C
- Internal Object Representation (IOR): Always in C
- Server Side Skeletons: translates IOR (in C) to developer’s LOP
- Implementation: developers LOP (Can be wrappers to legacy code)
How Much Language Interoperability Have We Achieved?

1431 test cases (and counting)
What we foresee, based on experience with our tests...

Language Interoperability

Developer Concerns: Configuration, Packaging, & Deployment

User Concerns: Installation Trust
These aren’t new problems...
But they are on a larger scale

- **Developer Concerns:** Configuration, Packaging, & Deployment

- **User Concerns:** Installation Trust
For Example...

Consider the following statements...

Java is more portable than C
C is more portable than C++
C++ is more portable than F77

Babel’s regression tests have the worst of all worlds!!!

Java can be ported, doesn’t need to be recompiled (like C)
C is mature, isn’t as hard to parse (as C++)
C++ can bind to C or Java easier than F77

BUT F77 is also more portable than C because of header/library issues
For Example...

To support Python and Java
  All libraries must be shared (*.so) not
  statically linked (*.a)

C++ shared libraries are problematic
  Exception support is platform/compiler
  dependent
  Linking issues when interoperating with
  other languages

Can create valid shared library with
uncatchable exceptions
# Babel’s Configuration/Build

<table>
<thead>
<tr>
<th>GNU Make</th>
<th>python’s own build system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autoconf</td>
<td>java’s built-in make</td>
</tr>
<tr>
<td>configuration</td>
<td>(broken) make</td>
</tr>
<tr>
<td>Automake</td>
<td>helper scripts</td>
</tr>
<tr>
<td>build Makefiles</td>
<td>fixes to autoconf,</td>
</tr>
<tr>
<td></td>
<td>automake &amp; libtool</td>
</tr>
<tr>
<td>Libtool</td>
<td>lots of hacks</td>
</tr>
<tr>
<td>shared libraries</td>
<td></td>
</tr>
<tr>
<td>CUTE</td>
<td></td>
</tr>
<tr>
<td>custom testing</td>
<td></td>
</tr>
</tbody>
</table>
Test History
sparc-sun-solaris2.7-gcc

Configure/Build/Repository faults, not software failures!!!
Problems affect the User too...

How does a user get and install “language interoperable” software?

Binary: if supplied by developer

Source:

- Assume “configure; make install”? 

How to link into application?

If any C++ code, must use C++ linker

Which C++ to use?

C++ has no std binary interface
Crux of the problem

We’re building 21st century technology... using 30 year old tools.

C++  F77  Python  Java

C

Autoheader  libtool (perl/sh)

aclocal  Automake (perl)

Autoconf (M4)

Make

Bourne shell
Solution

Integrated config, build, package, test and management system.

no make inside!

- can have action create many files
- understands directories

uses real database

program all aspects in one language

MUST BE OPEN SOURCE
In the mean time...

Babel works on other platforms, just not automated config/testing

  Java code generator (precompiled)
  ANSI C runtime library (no problem)

Babel’s tests are (necessarily) pathological worst-case examples

We didn’t create these problems, we just exercise them aggressively
Future Babel: Will Provide More Build Help

“babel.make”
- currently lists code generated
- may add additional flags, macros, etc.

configure
- currently used for regression tests
- may generate artifacts useful for developers
  - helper scripts
  - warnings
Farther Future Babel: Will Do Distributed Computing
Closing Remarks

Babel Beta 0.5 is released
Babel enables language interoperability
   connect C, C++, F77, and Python
   provide a uniform object-model, even in non-OO languages.

Deploying & Installing Language Interoperable Code in General
   is still very hard
   has broken every tool we use
The End

babel-announce@llnl.gov
babel-users@llnl.gov

http://www.llnl.gov/CASC/components
components@llnl.gov

Bill Bosl, Tammy Dahlgren,
Tom Epperly, Scott Kohn,
Gary Kumfert, & Steve Smith
A.3. Can HPC and Component Technology REALISTICALLY be integrated?

Yes.

But HPC Components have huge (and unique) hurdles:
- Diverse Architectures
- Diverse OS’s
- Integration of SPMD and Dist. Comp.
- Archaic Pkg/Devel/Config/Build tools
- Non-CS trained (or interested) users
B.3. Can the HPC community really afford yet another compiler such as Babel?

Is language interoperability important?

How important?
B.3. How is the Java subset of C++ inadequate as an HPC IDL?

What is a “Java subset of C++”?  
How does one  
use it to bind to other languages?  
get a common inheritance model?  
get a common exception model?
B.5. What is the role of traditional (parallel) tools in component technology?

Hopefully, they’re replaced by modern parallel tools.
B.9. What will be the configuration issues for components...to be portable and high-performance?

Lots.

Lack of Configuration, Packaging, & Deployment tools

is the #1 Achilles heel for components

is the #1 day-to-day pain in Babel development

#1 cause for failure in regression tests

CASC
C.1. Will anyone actually make the effort to componentize their applications software?

Yes.

But it will be messy.

Efforts to Babelize at LLNL:

- hypre - want OOP in ANSI C & automatic F77 bindings
- ALPS - want scripting interface for laser plasma physics
- SAMRAI - framework used in ALPS
D.4. Should components be viewed as mostly a library/runtime developer technology?

No.

I used components in this PowerPoint Presentation.

Users can use components without knowing they’re using them.

This is harder to achieve in UNIX than other platforms.
Work performed under the auspices of the U. S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under Contract W-7405-Eng-48