A Language Interoperability Tool for Scientific Computing

Gary Kumfert, Tamara Dahlgren, and Thomas Epperly

Center for Applied Scientific Computing

This work was performed under the auspices of the U.S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under Contract No. W-7405-Eng-48.

UCRL-200005-PRES
Outline

- Problem: Mixing Languages
- Babel Features
- Babel Performance/Overhead
  - Whole Application
  - Single Method Invocation
- Related Projects
  - IDL-based solutions
  - Source-parsing solutions
- Babel Customers/Collaborators
- Babel on AIX
- Conclusion
What I mean by “Language Interoperability”

- Scripting Driver (Python)
  - Simulation Framework (C)
  - Visualization System (Java)
  - Numerical Routines (f77)
  - Solver Library (C++)
  - Callback Handlers (Python)
Mixing Languages: hard, not portable, and unscalable
Babel makes all supported languages peers

This is not a Lowest Common Denominator Solution!

Once a library has been “Babelized” it is equally accessible from all supported languages
Library Developer Does This...

1. Write SIDL File
2. `babel --server=C++ greetings.sidl`
3. Add implementation details
4. Compile & Link into Library/DLL
Library User Does This...

1. `babel --client=F90 greetings.sidl`
2. Compile & Link generated Code & Runtime
3. Place DLL in suitable location
Babel Architecture

Client-side
- App
  - C
  - C++
  - F77
  - F90
  - Python
  - Java

Server-side
- Stub
- IOR
  - Intermediate Object Representation
- Skel
- Impl
  - C
  - C++
  - F77
  - F90
  - Python
  - Java
Features Tested Nightly

- **Basic Types**
  - bool
  - char
  - int
  - long
  - float
  - double
  - fcomplex
  - dcomplex
  - string
  - opaque

- **Extended Types**
  - Objects
  - enumerations
  - arrays of any the above
    - multidimensional
    - strided
    - dynamically allocated
    - no arrays of arrays

- **Modes**
  - in
  - out
  - inout
  - return value
Features Tested Nightly

- **Basic Types**
  - bool
  - char
  - int
  - long
  - float
  - double
  - fcomplex
  - dcomplex
  - string
  - opaque

- **Extended Types**
  - Objects
    - enumerations
    - arrays of any the above
      - multidimensional
      - strided
      - dynamically allocated
    - no arrays of arrays
  - OO Method Dispatch
    - regular
    - final
    - static
    - interfaces
    - classes
  - Exception Handling
  - Modes
    - in
    - inout
    - return
    - value

Features Tested Nightly

- Basic Types
- Extended Types
- OO Method Dispatch
- Modes
- Exception Handling
- For All Combinations of Languages
  - C
  - C++
  - F77
  - F90
  - Python
  - Java

- Linkage
  - Static
  - Run Time
Features Tested Nightly

- Basic Types
- Extended Types
- OO Method Dispatch
- Modes
  - in
- Exception Handling
- For All Combinations of Languages
  - C
  - C++
  - F77
  - F90
  - Python
  - Java
- 10,000+ test cases
- per platform
- per compiler set
- Linkage
  - Static
  - Run Time
Outline

• Problem: Mixing Languages
• Babel Features
• Babel Performance/Overhead
  ➤ Whole Application
  ➤ Single Method Invocation
• Related Projects
  ➤ IDL-based solutions
  ➤ Source-parsing solutions
• Babel Customers/Collaborators
• Babel on AIX
• Conclusion
Performance Impact on Whole Apps: Negligible

- **hype**: “Lost in the noise”

- **TAO/PETSc**: “overhead of using components is negligible and it does not affect the scalability of the algorithm”

Total execution time for a surface minimization problem using a fixed-sized 250x250 mesh. Dual 550 MHz Pentium III nodes with 1-Gb of RAM each, connected with Myrinet
Overhead on Single Function Call: Small & Variable


- “avg” Babel overhead \( \approx 3.8 \times F77 \)
  - Depends on argument modes, argument types and languages involved
  - All Babel calls are virtual
    - (C++ virtual \( \approx 2.2 \times F77 \))
- CORBA \( \approx 25 \times \) Babel
Babel Performance Models: Joint work /w PERC & TSTT

- Efficient Implementation of Babel
  - compulsory (Babel object model)
- Efficient Use of SIDL

also, how hard for customer to use SIDL efficiently?

not performance tuned yet

- compulsory (multi-language)

  e.g. no IOR shortcut if caller & callee in same language
  e.g. No C++-style inline
Outline

- Problem: Mixing Languages
- Babel Features
- Babel Performance/Overhead
  - Whole Application
  - Single Method Invocation
- Related Projects
  - IDL-based solutions
  - Source-parsing solutions
- Babel Customers/Collaborators
- Babel on AIX
- Conclusion
Other IDL Projects In Scientific Computing

- **ASE: Argonne SIDL Environment**
  - [http://www.mcs.anl.gov/ase](http://www.mcs.anl.gov/ase)
  - Knepley and Smith @Argonne
  - Based on Babel-0.6 (Dec’01)
  - Foundation for PETSc 3.0

- **PIDL: Parallel Interface Definition Language**
  - Damevski & Parker @SCI Institute, Utah
  - C++ only
  - Parallel RMI
SWIG v. Babel

(David Beazley @ U Chicago)

- Call from Tcl, Perl, Python, Java, Ruby, mzscheme, or Guile
- Implement in C, C++

- Reads existing code
  - Library User can do independently
  - C++ “type system”
  - Auxiliary .i files fill in details
- Better suited for fast prototyping

- Call from C, C++, F77, F90, Python, and Java
- Implement in C, C++, F77, F90, and Python
- Hand-written SIDL
  - Library Developer task (or “motivated” user?)
  - SIDL “object model”
  - SIDL is self contained, no extra hints needed
- Better suited for production use
Projects Citing Babel In Their Pubs
(see [www.llnl.gov/CASC/components/gallery.html](http://www.llnl.gov/CASC/components/gallery.html) for more)

---

I implemented a Babel-based interface for the hypre library of linear equation solvers. The Babel interface was straightforward to write and gave us interfaces to several languages for less effort than it would take to interface to a single language.

--Jeff Painter, LLNL.

---

CCA
Common Component Architecture

[www.llnl.gov/CASC/components/gallery.html](http://www.llnl.gov/CASC/components/gallery.html)

NWChem
High Performance Computational Chemistry Software

research.cs.vt.edu/lacsa

BABEL

SAMRAI
Structured Adaptive Mesh Refinement Application Infrastructure

Tuning and Analysis Utilities

ALPS

TOPS
Babel 0.8.6 supports IBM compilers on AIX

- “support” = `gtar; ./configure; make check`

- Major barrier was the build, not code
  - use autoconf, automake, libtool, & distutils

- Run-time linking (aka dlopen()) on AIX remains a challenge
  - needed for server-side Java and Python
  - still having trouble building libpython.so on AIX
Critical Resources for AIX Port

- Cobb, Hook, Strauss, Ambati, Govindjee, Huang & Kumar. *AIX Linking and Loading Mechanisms*  

- GNU libtool 1.5 (or better)
Conclusion: Babel makes software easier to use

- “Babelizing” a library is generally ____ than hand crafting wrappers
  - more scalable
  - easier / more sustainable
  - less error-prone
  - more portable

- Our customers also like
  - having polymorphism in non-OO languages
  - stronger encapsulation than even C++
  - producing new interfaces without modifying legacy code
  - SIDL for specifying API standards
  - Easier incremental evolution via looser coupling
Contact Info

- Project: [http://www.llnl.gov/CASC/components](http://www.llnl.gov/CASC/components)

- Project Team Email: [components@llnl.gov](mailto:components@llnl.gov)

- Mailing Lists: [majordomo@lists.llnl.gov](mailto:majordomo@lists.llnl.gov)
  - subscribe babel-users [email address]
  - subscribe babel-announce [email address]
greetings.sidl: A Sample SIDL File

```idl
package greetings version 1.0 {

    interface Hello {
        void setName( in string name );
        string sayIt( );
    }

    class English implements all Hello {
    }
}
```
Library Developer Does This...

1. Write SIDL File
2. `babel --server=C++ greetings.sidl`
3. Add implementation details
4. Compile & Link into Library/DLL
Adding the Implementation

```cpp
namespace greetings {
    class English_impl {
    private:
        // DO-NOT-DELETE splicer.begin(greetings.English._impl)
        ::std::string d_name;
        // DO-NOT-DELETE splicer.end(greetings.English._impl)

        ::std::string greetings::English_impl::sayIt() throw () {
            // DO-NOT-DELETE splicer.begin(greetings.English.sayIt)
            ::std::string msg("Hello ");
            return msg + d_name + "!");
            // DO-NOT-DELETE splicer.end(greetings.English.sayIt)
        };
    }
```
namespace greetings {
class English_impl {
    private:
    // DO-NOT-DELETE splicer.begin(greetings.English._impl)
    ::std::string d_name;
    // DO-NOT-DELETE splicer.end(greetings.English._impl)

    ::std::string
    greetings::English_impl::sayIt() throw () {
        // DO-NOT-DELETE splicer.begin(greetings.English.sayIt)
        ::std::string msg("Hello ");
        return msg + d_name + ";
        // DO-NOT-DELETE splicer.end(greetings.English.sayIt)
    }
}

package greetings version 1.0 {
    interface Hello {
        void setName( in string name );
        string sayIt( );
    }
    class English implements-all Hello {
    }
}
Library User Does This...

1. `babel --client=F90 greetings.sidl`
2. Compile & Link generated Code & Runtime
3. Place DLL in suitable location
**F90/Babel “Hello World” Application**

```f90
program helloclient
  use greetings_English
  implicit none
  type(greetings_English_t) :: obj
  character (len=80) :: msg
  character (len=20) :: name
  name = 'World'
  call new(obj)
  call setName(obj, name)
  call sayIt(obj, msg)
  call deleteRef(obj)
  print *, msg
end program helloclient
```

*These subroutines come from directly from the SIDL*

*Some other subroutines are “built in” to every SIDL class/interface*
These subroutines come from directly from the SIDL

Some other subroutines are “built in” to every SIDL class/interface