Tamara Dahlgren, Tom Epperly, Scott Kohn, & Gary Kumfert
Center for Applied Scientific Computing
Audience Calibration

- What is a component?
- Who writes code?
- Who uses code?
- What languages used?
- What platforms used?
- # 3\textsuperscript{rd} party libraries your code uses?
- # apps uses your libraries?
Outline

- Problem Motivation
- Babel Solution Strategy
- SIDL
- Using Babel
- Outstanding Problems
- Future R&D
Problem Motivation

- Code Reuse is Hard.
- Scientific Code Reuse is Harder!
- Barriers to Reuse...
  - Language Interoperability
  - Semantics
  - Software Portability
  - Lack of Standards
  - More...
What I mean by “Language Interoperability”

Simulation Framework (C)

Scripting Driver (Python)

Visualization System (Java)

Numerical Routines (f77)

Solver Library (C++)

Callback Handlers (Python)
Current Language Interoperability

- C
- C++
- Java
- Python
- f77
- f90
- cfortran.h
- SWIG
- JNI
- Silloon
- Chasm
- Platform Dependent
Babel Enabled
Language Interoperability

C

C++

Java

f77

f90

Python
Babel Enabled
Language Interoperability

What’s in version 0.6?

C

C++

Java

Python

f77

f90
Outline

- Problem Motivation
- Babel Solution Strategy
- SIDL
- Using Babel
- Outstanding Problems
- Future R&D
Developer Writes Interface

- SIDL: Scientific Interface Definition Language
- Similar to CORBA/COM IDLs...
  - Language/Platform Independent
- ...but tuned for scientific apps
  - complex numbers
  - dynamic, multidimensional arrays
version MySolverLib 0.1.0;

import ESI;

package MySolverLib {

    interface MatrixGenerator {
        ... }

    class OptionDatabase {
        void getOption( 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);
    }

}
Babel Generates Glue Code

- SIDL interface description
- XML repository interface description
- parser
- analyzer
- backend
- machine configuration database
- XML
- C
- C++
- F77
- Python
- Java
- F90
- Matlab?

CASC
Babel Provides
Uniform Object Model

- Universal Base Classes
- Virtual Method Dispatch
- Exception Handling
- Reference Counting

...It all happens here!
Babel Provides a Firewall Between Use and Implementation

Application

Stubs

IORs

Skels

Impls

SIDL interface description
Outline

- Problem Motivation
- Babel Solution Strategy
- SIDL
- Using Babel
- Outstanding Problems
- Future R&D
SIDL as a text-based design tool

- Express only the public API
- Prevents discussion drift into implementation details
- Amenable to email debates
- Easier to learn than UML
The SIDL Grammar

- Packages & Versions
- Interfaces & Classes
- Inheritance Model
- Methods
- Polymorphism Modifiers
- Intrinsic Data Types
- Parameter Modes
- Gotchas
Packages

version foo 1.0;
package foo {
   // ...
};

package gov {
   package llnl {
      package babel {
         // ...
      };
   };
};

- Use SIDL packages to prevent symbol conflicts
  - packages in Java
  - namespaces in C++
  - prefixes in C / Fortran
    (e.g. mpi_send() )
- must have version number
- lowercase symbols recommended
- Can be nested
Interfaces and Classes

- ObjectiveC and Java Inheritance Model
- Interfaces
  - pure abstract base classes in C++
  - define calling sequence only
  - provide no implementation
  - cannot be instantiated
  - can inherit ("extend") other interfaces
- Classes
  - inherit ("extend") from at most one class (including its implementation)
  - may inherit ("implement") multiple interfaces
Interfaces and Classes (Example)

```java
import their;
package my {
    interface Foo extends their.Foo {
    }
    class CFoo implements Foo {
    }
    class Bar extends CFoo implements their.Bar {
    }
    class Baz extends CFoo implements their.Bar, their.Baz {
    }
}
```
Inheritance Model

- Interfaces form contracts between implementor and user.

- Default Inheritance:
  - reference counting
  - dynamic casting
  - introspection
  - reflection

```
SIDL.BaseInterface
```

```
SIDL.BaseClass
```

```
(classes)
```

```
(interfaces)
```

```
extends
```

```
extends
```

```
implements
```

```
BABEL
```

```
CASC
```

GKK 21
Abstract Class–Partially Implemented Class

interface Foo {
    int doThis( in int i );
    int doThat( in int i );
}

class Bar implements Foo {
    int doThis( in int i );
}

abstract class Bar implements Foo {
    int doThis( in int i );
}

class Grille implements Foo {
    int doThis( in int i );
    int doThat( in int i );
}
Methods (a.k.a. “member functions”)

- Belong to both Interfaces and Classes
- SIDL has no sense of method “access” specifiers
  - (e.g. private, protected, public)
  - All methods are public
  - Makes sense for an “Interface Definition Language”
- In classes only, methods can also be
  - static -- independent of an instance
  - final -- not overridden by derived classes
- No Method Overloading. (yet?)
Method Modifiers

- **static**
  - avoid OOP altogether:
    - make one class full of static methods.

```cpp
class Math {
  static double sin( in double x );
  static double cos( in double x );
};
```

- **final**
  - prevent function from being overridden
  - In C++
    - methods are final by default
    - must be declared “virtual” to be overridden
Intrinsic Data Types

- **Standard Types**
  - bool
  - char
  - int
  - long
  - float
  - double
  - fcomplex
  - dcomplex

- **Advanced Types**
  - string
  - enum
  - object (interface or class)
  - array< Type, Dimension >
  - opaque

**NOTES:**
- Mapped to different types in different languages
- No General Template Mechanism (maybe later?!?)
Parameter Modes

- Unique to IDLs
- Each parameter in a method call has a mode declared
  - in
  - out
  - inout
- Intent:
  - Communication optimization for distributed components
  - Copy minimization when copy is unavoidable
- Benefit:
  - Easy to understand intent when reading
Parameter Modes II

- “in”
  - pass by value semantics (not const!)

- “out”
  - pass by reference semantics
  - no initialization required
  - information returned

- “inout”
  - pass by reference semantics
  - initialization required
  - new information returned
  - instance may be destroyed and replaced
Parameter Modes III

package util { // SIDL FILE
    class String {
        static void reverse( inout string );
    };
};

#include <stdio.h>
#include "util_String.h"

int main () {
    char * hi = "Hello.");
    util_String_reverse( &hi );
    printf("%s\n", hi );
}

DANGER:
“inout” parameters may be destroyed and replaced under the covers.
Do you want to risk a “free(hi);” in the stubs???
package util { // SIDL FILE
    class String {
        static void appendReverse(inout string);
    }
}

#include <stdio.h>
#include “util_String.h”

int main () {
    char * hi = “Hello.”;
    util_String_appendReverse( &hi );
    printf(“%s\n”, hi );
}

Parameter Modes V

package util { // SIDL FILE
class String {
    static void appendReverse(inout string);
};

#include <stdio.h>
#include <string.h>
#include "util_String.h"

int main () {
    char * hi = strdup( "Hello." );
    util_String_appendReverse( &hi );
    printf("%s\n", hi );
    free( hi );
}
/** This is a DocComment for the package */
package Hello {

/**
 * This class has one method
 */
class World {

/**
 * result = “hello” + name */
  string getMsg( in string name );
};
};
SIDL Gotchas

- Case Sensitive
  - SIDL is
  - F77 is not

- Reserved Words:
  - union of C, C++, Fortran
  - C++ has 90+ reserved words!

- Forbidden Method Names
  - same as class name (reserved in C++)

- Built-in method names start with “_” to avoid collisions with user defined names.
Outline

- Problem Motivation
- Babel Solution Strategy
- SIDL
- Using Babel
- Outstanding Problems
- Future R&D
Getting The Software

- **Grab tarball**
  - [http://www.llnl.gov/CASC/components/software.html](http://www.llnl.gov/CASC/components/software.html)
  - Current release: babel-0.6.0.tar.gz

- **Typical build/install** (using VPATH)
  - `gtar zxvf babel-0.6.0.tar.gz`
  - `cd babel-0.6.0-build/`
  - `../babel-0.6.0/configure --prefix=${HOME}/babel`
  - `gmake all check install`

- **Platforms Tested Nightly:**
  - Linux ( GNU )
  - Solaris ( GNU, Sun, KCC )
The Babel Compiler – commandline options

- Choose exactly one of the following:
  - --help: Display more info
  - --version: Babel Version
  - --parse-check: Parse SIDL, no output
  - --xml: Generate XML
  - --client=[lang]: User of Babel Object
  - --server=[lang]: Developer of Babel Object

- Other Options
  - --output-directory=[dir]: Default = .
  - --repository-path=[path]: Semicolon separated URLs
  - --generate-subdirs
Babel from a developer’s POV

Application

- Stubs
- IORs
- Skels
- Impls

SIDL interface description
/** This is a DocComment for the package */
version hello 1.0;

package hello {

    class World {
        void setName( in string name );
        /** result = “hello” + name */
        string getMsg( );
    };
};
# Babel Generates LOTS of Code!!!

<table>
<thead>
<tr>
<th>hello.sidl</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generated C/C++ code (wc -l *)</td>
<td>4,107</td>
</tr>
<tr>
<td>Hand added Implementation</td>
<td>4</td>
</tr>
</tbody>
</table>
Adding the Implementation

```cpp
namespace hello {
    class World_impl {
        private:
            // DO-NOT-DELETE splicer.begin(hello.World._implementation)
            // Put additional implementation details here...
            // DO-NOT-DELETE splicer.end(hello.World._implementation)
    
    string
    hello::World_impl::getMsg ()
    throw ()
    {
        // DO-NOT-DELETE splicer.begin(hello.World.getMsg)
        // insert implementation here
        // DO-NOT-DELETE splicer.end(hello.World.getMsg)
    }
```
```cpp
namespace hello {
    class World_impl {
        private:
            // DO-NOT-DELETE splicer.begin(hello.World._implementation)
            string d_name;
            // DO-NOT-DELETE splicer.end(hello.World._implementation)

    string
    hello::World_impl::getMsg ()
    throw ()
    {
        // DO-NOT-DELETE splicer.begin(hello.World.getMsg)
        string msg("hello ");
        return msg + d_name + "!";
        // DO-NOT-DELETE splicer.end(hello.World.getMsg)
    }
```
Methods Beginning with “_”

- Method names cannot start with “_” in SIDL.
- Babel uses leading underscores for internal stuff:
  - e.g. IOR-level methods “_create()”
  - e.g. binding specific methods “PKG::CLASS::_get_ior()”
- Note: Things that look like a double underscore:
  - e.g. hello_World__create() is really normal convention with internal method.
Babel from a user’s POV

Application

Stubs

libfoo.so

SIDL interface description
A driver in C

```c
#include <stdio.h>
#include "SIDL.h"
#include "hello.h"

int main(int argc, char ** argv ) {
    hello_World hw;
    hw = hello_World__create();
    hello_World_setName( hw, argv[1] );
    fprintf(stdout, "%s", hello_World_getMsg( hw ) );
    hello_World_deleteReference( hw );
}
```
A driver in Python

```python
import hello.world

if __name__ == '__main__':
    h = hello.world.World()
    h.setName('Gary')
    print h.getMsg()
```
Outline

- Problem Motivation
- Babel Solution Strategy
- SIDL
- Using Babel
- Outstanding Problems
- Future R&D
Common Problems

- $CLASSPATH$ not set
- Compilers not found ($CC$, $CXX$, $F77$)
- Python or NumPy not installed
- Server-side Python requires libpython.so (Not in standard distributions)
- LD_LIBRARY_PATH issues with shared libraries
- C++ and shared libraries
Achilles’ Heel

- Babel Generates Correct Code
- It does nothing about correct compilation
How Much Language Interoperability Have We Achieved?

- 3213 test cases
  - per platform
  - per compiler set
Babel Development Tools

- Development platforms
  - sun-sparc-solaris2.7
  - intelx86-redhat-linux
  - cygwin

- Compilers
  - Python 2.1
  - Sun jdk-1.3
  - gcc 2.95.x
  - sunpro 5.0
  - KCC

- Build Tools
  - make
  - autoconf
  - automake
  - libtool

- Testing
  - in-house tool

- Bug-Tracking
  - in-house/bugzilla mods
Outline

- Problem Motivation
- Babel Solution Strategy
- SIDL
- Using Babel
- Outstanding Problems
- Future R&D
Platform Independence

- Encourage Locality for Maximum Performance
- Connect to separate process space
  - to avoid symbol conflicts at link time
- Connect to separate machine
  - to utilize special hardware
  - to use platform specific code
    (Babel doesn’t get Windows apps to run on UNIX!)
  - To distribute work
Same Process Space Components

- Application
  - Stubs
    - IORs
      - Skels
        - Impl
Out of Process Components

Application

<table>
<thead>
<tr>
<th>Stubs</th>
<th>IORs</th>
<th>IPC</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>IPC</th>
<th>IORs</th>
<th>Skels</th>
<th>Impl</th>
</tr>
</thead>
</table>

Diagram shows the flow of components from Application to Stubs, then to IORs and IPC, and finally to IPC, IORs, Skels, and Impl.
Remote Components

Application → Stubs → IORs → Marshaler

Line Protocol → Unmarshaler → IORs → Skels → Impl
Parallel Components: MxN Communication
Problem Motivation

- Code Reuse is Hard.
- Scientific Code Reuse is Harder!
- Barriers to Reuse...
  - Language Interoperability
  - Semantics
  - Software Portability
  - Lack of Standards
  - More...

Tammy Dahlgren’s PhD Thesis
Problem Motivation

- Code Reuse is Hard.
- Scientific Code Reuse is Harder!
- Barriers to Reuse...
  - Language Interoperability
  - Semantics
  - Software Portability
  - Lack of Standards
  - More...

Whitepaper
How do I find out more???

- Website  http://www.llnl.gov/CASC/components
- User’s Guide
- Download Code
- Email Reflector (subscribe via majordomo@lists.llnl.gov)
  - babel-users@llnl.gov
  - babel-announce@llnl.gov
- Email the team
  - components@llnl.gov
- Tutorial for CASC tomorrow
  - B451 White Room at 3:30pm  Fri, Nov 9
The End
Business Component Frameworks

- CORBA
  - Language Independent
  - Wide Industry Acceptance
  - Primarily Remoting Architecture

- Enterprise Java Beans (EJB)
  - Platform Independent
  - Runs wherever Java does

- COM
  - Language Independent
  - Most Established
  - In Process Optimization
  - Network Transparent
Business Component Frameworks

- **CORBA**
  - Language Independent
  - Wide Industry Acceptance
  - Primarily Remoting Architecture
  - Huge Standard
  - No In-Process Optimization

- **COM**
  - Language Independent
  - Most Established
  - In Process Optimization
  - Network Transparent
  - not Microsoft Transparent
  - Relies on sophisticated development tools

- **Enterprise Java Beans (EJB)**
  - Platform Independent
  - Runs wherever Java does
  - Language Specific
  - Potentially highest overhead

- **All The Above**
  - No Complex Intrinsic Datatype
  - No Dynamic Multidimensional Arrays
  - No Fortran77/90/95 bindings
  - No Parallel Components
Key to Babel’s Interoperability...

**SIDL**
Scientific Interface Definition Language

**IOR**
Intermediate Object Representation

**XML**
eXtensible Markup Language

Human Compatible

Compiler Compatible

Web Compatible
CASC

Work performed under the auspices of the U. S. Department of Energy by Lawrence Livermore National Laboratory under Contract W-7405-Eng-48