version 0.99.0  (aka 1.0rc1)

This changes everything…

…and change is GOOD

Gary Kumfert, James Leek

& Thomas Epperly

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-PRES-222291
We’re celebrating!

- With 0.99.0, We’ve satisfied every item on our 1.0 release criteria
- The 1.0 Release Criteria document has been our roadmap since Dec 2003
0.99 is a major change

1. Complete rewrite of Parser
2. Changed Type Resolution
3. Modifications to SIDL
4. Improved babel-\{cc,cxx,f77,f90\} scripts
5. Significant RMI & multithreading improvements
6. A new feature we haven’t found a name for yet
1. Complete Rewrite of the Parser
1. Complete Rewrite of the Parser

- Better error messages!
- Change type resolution (more on this later)
- Easier to adapt in the future (structs are coming!)
- Easier for 3rd parties to participate.
Before: Parsing, Checks, & Resolution were tightly interwoven
Now: Decoupled the stages & visitor pattern*

* Visitor pattern was first suggested by Matt Knepley back in 2001
FUTURE: Backends will migrate off of Symbol Table to AST

- Backends will get smaller in the process!
- Symbol Table will eventually go away
2. Changed Type Resolution (it was too aggressive)
2. Changed Type Resolution
(it was too aggressive)

Now: These two files are now equivalent

No longer need special attention which order SIDL files appear on the command line.
3. Modifications to SIDL

a. Added a global scope indicator
b. Added a “from clause” to resolve multiple-inheritance induced collisions
c. Broadened rarray extents from single variables to expressions
d. Allow leading underscore or digit in method suffix
e. Added `%attrib{ }` blocks to add arbitrary user data for custom bindings
package foo
version 0.0 {
class A {
    package foo {
        class A {
            foo.A bar();
        }
    }
}
}

Q: What does bar() return?
- Before: foo.foo.A
- Before: foo.A was not addressable from that scope
- Now: use “.foo.A” to specify top level scope
3.b. The new (and novel) FROM Clause

interface I1 { init( in int i );}
interface I2 { init( in float f );}
class C implements-all I1, I2 { }

☀ Before:

- would throw a signature Conflict...
- and print 37 lines of text stderr/stdout

☀ Now:

Signature conflict between method
"abstract void init( in double d) throws sidl.RuntimeException"
from "pkg.I2" and method
" void init( in int i) throws sidl.RuntimeException"
from "pkg.C".
3.b. The new (and novel) FROM Clause

- New syntax to resolve the conflict

```java
interface I1 { init( in int i ); }
interface I2 { init( in float f ); }
class C implements-all I1, I2 {
    init[f]( in float f ) from I2.init;
}
```

- Restriction: can only introduce new suffix! (langs that support overloading can't handle more)
- Python: methods can be removed! May want to upcast to expected type.
3.c. **Broader extents of Raw Arrays**

- **Before:**
  ```cpp
  void foo( in rarray<int,2> A(m,n),
    in int m, in int n );
  ```

- **Now:** Allow simple arithmetic expressions & constants
  ```cpp
  void foo( in rarray<int,3>
            A(2*m,2*n+3*(n+1), 3),
    in int m, in int n );
  ```

- **Limitation:** max one variable per expression in a dimension
  (Why? #eqns == #unknowns)
3.d. Allow leading underscore or digit in method suffix

Now: following inits are all legal

```cpp
interface Iface {
    init( in int i );
    init[2]( in int i, in int j );
    init[2a]( in int i, in char a );
    init[_]( in bool not_recommended );
    init[_2yikes]( inout Iface scary );
}
```

Warnings issued if/when you stumble on an internal suffix.
(e.g. [f])
3.e. The extensible `%attrib{}` blocks

- **WARNING**: This feature matters iff you are writing a new backend, or parsing Babel's XML

```plaintext
%attrib{ key1 }
%attrib{ key2="some value" }
%attrib{ key1, key2="some value", keyN }
```

- **Intention is**
  - to make SIDL more extensible
  - Support development of innovative features
What's an attribute?

- Metadata associated with Types, Methods, or Arguments in SIDL

- Before: Only supported “built-in” attributes
  - Types could be `final` or `abstract`
  - Methods could be `local`, `static`, `abstract`, and/or `final`
  - Args can be `copy`

- Now: can add arbitrary attributes with the `%attrib{ }` command.
Possible Uses

Specify a default value for an argument

```cpp
void foo( %attrib{ default="1.0" } in double d );
```

Specify a parallel operation that returns the max of all processes' values

```cpp
%attrib{ collective } void foo( %attrib{ reduce="max" } out double d );
```
Interesting properties

- For all built-in attributes, X: “%attrib{ X }” is equivalent to “X”
- For all SIDL, C/C++, Fortran, Python, and Java keywords, K: %attrib{ K } is not precluded (separate tokenizer avoids collisions)
- Attributes are preserved in XML
- Backends should quietly ignore attributes they don't understand
0.99 is a major change

1. Complete rewrite of Parser
2. Changed Type Resolution
3. Modifications to SIDL
4. Improved babel-\{cc,cxx,f77,f90\} scripts
5. Significant RMI & multithreading improvements
6. A new feature we haven’t found a name for yet
4. Improved the *babel-{cc,cxx,f77,f90} scripts*

- These scripts orchestrate the compiler, babel-config, and babel-libtool for you.

```bash
% babel-cc -c -n pkg_cls_Impl.c
$(bindir)/babel-libtool --quiet --tag=CC --mode=compile
gcc -c -I$(includedir) -I/usr/include/libxml2
pkg_cls_Impl.c
```

- There will be more work here for 1.0.
- Will support “--with-mpi”
5. Significant RMI and Multithreading Improvements

Thanks to PSI…

w/ PSI, build support for PRMI and Nonblocking RMI

- PSI Overlord (C++)
- Input Deck (Python)
- PSI Daemons
- Ale3d (ANSI C)
- Fine Scale Response Compute Farm (F90)

Not shown: All processes can RMI. Overlord & Overlord has table of all rank 0 processes.
6. A new feature we haven’t found a name for yet
New constructor capabilities

- Useful for temporarily wrapping a native language structure as a Babel object.
- For C and Fortran, it can act like a C++ placement new. You can initialize the private data struct before creating the object.
- Requires tight coupling between client and implementation.
**(Temporarily wrapping native objects (C++))**

Assume a C++ Mesh called `myMesh` & SIDL class `MeshWrap`

```
#include "foo_MeshWrap_Impl.hxx"
....numerous lines skipped....
{
    // create a Babel Impl object to wrap MyMesh
    MeshWrap_Impl m = new MeshWrap_Impl();
    m.setMesh(myMesh); // call a non-Babel method on
        // the Impl class
    // pass m to a Babel object meshRefiner through
    // a Babel method call
    meshRefiner.refineMesh(m);
} // m goes out of scope and is garbage collected
    // myMesh was temporarily wrapped up for a Babel
    // call and can now be used by the rest of the C++ app
```
Temporarily wrapping native objects (Java)

Assume a Java Mesh called myMesh & SIDL class MeshWrap

```java
{
    // create a Babel Impl object to wrap MyMesh
    MeshWrap_Impl m = new MeshWrap_Impl();
    m.setMesh(myMesh); // call a non-Babel method on
    // the Impl class
    // pass m to a Babel object meshRefiner through
    // a Babel method call
    meshRefiner.refineMesh(m);
} // m goes out of scope and is garbage collected
// myMesh was temporarily wrapped up for a Babel
// call and can now be used by the rest of the
// Java app
```
Temporarily wrapping native objects (Python)

- You can new the Impl in Python or...
- You can wrap any Python object that implements the required methods! (DANGEROUS but very Pythonic)

```python
from foo.MeshWrap import MeshWrap
babelMesh = MeshWrap(impl = myMesh)
# babelMesh is a Python object wrapping
# myMesh. RuntimeException's will occur
# if myMesh doesn't implement all the
# expected methods
```
Example of Dangerous Python

 SIDL file

```python
package f version 1.0 {  class S {  
    void sayHello(in string hello);
}}
```

Any Python instance that implements sayHello can be wrapped as follows:

```python
>>> from f.S import S
>>> s = S()
>>> s.sayHello("Tom")
>>> class Override:
...    def sayHello(self, name):
...       print "Python says hello to " + name
...
>>> o = Override()
>>> s = S(impl = o)
>>> s.sayHello("Tom")
Python says hello to Tom
```
**Temporarily wrapping native objects (C, F77)**

- For C, pass a pointer to the private struct defined in the _Impl.h file to the `_wrapObj(void *data, _sdl_BaseInterface _ex)` method.
- For F77, pass an opaque to the `_wrapObj` method.
- These values are stored in the IOR and ctor2 is called instead of ctor.
Temporarily wrapping native objects (F90)

```fortran
use x_y_z_impl

type(x_y_z_wrap) :: myData

type(x_y_z_t) :: myObj

allocate(myData%d_private_data)

! ... ! initialize myData%d_private_data
! ... 
call wrapObj(myData, myObj, exception)
```
In case you hadn't heard...

- Original (D)C++ binding is gone.
- UC++ binding is now the default C++ binding.
- See Tom's Jan 2006 talk on what's involved in upgrading.
Conclusion

- Babel 0.99.0 is our first release candidate for Babel 1.0
  - No new features planned between now and 1.0.
  - Bugfixes and Documentation fixes still in the works
  - Babel 1.0 will be out before SciDAC meeting
- Babel 0.99.0 is a big change from Babel 0.11.x series.
- Change is good!