Multi-Language Struct Support in Babel

Dietmar Ebner

October 2009

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

UCRL “XXXXXXXXXXXXX”

About Me

Introduction

- Dietmar Ebner
- Recent post-doc at LLNL (August 2009)
- Academic credentials from the Vienna University of Technology (Austria)
- Background
  - Compilers / Code Generators
  - Embedded Systems
  - Combinatorial Optimization

Struct Support in Babel

Goal:

Provide access to native structured data types for Babel-generated interfaces.

Motivation:

- Performance (eliminating Babel calls for getters/setters)
- Reduced development effort
- Completeness ("natural" way of grouping semantically related data)
- Compatibility with existing interfaces
- Compatiblity with related systems (CORBA, WSDL)

Example

SIDL Class

```
class Date {
  int getMonth();
  void setMonth(in int month);
  int getDay();
  void setDay(in int day);
  int getYear();
  void setYear(in int year);
}
```

SIDL Struct

```
struct date_t {
  int month;
  int day;
  int year;
}
```

Cornerstones

- SIDL structs can contain any data type, including raw arrays and structs
- There is no support for arrays of structs
- Structs are not reference counted by Babel
- Babel automatically generates code for (de)serialization
- No copies when passing between C, C++, and Fortran 2003

Example: SIDL Struct Declaration

```
enum Color { red, blue, green }

struct MyOtherStruct {
  ...
}

struct MyStruct {
  int                 d_int;
  dcomplex            d_dcomplex;
  Color               d_enum;
  sidl.BaseClass      d_object;
  MyOtherStruct       d_struct;
  array<string>       d_string_array;
  rarray<double,1>    d_rarrayRaw(d_int);
  rarray<double,1>    d_rarrayFix(3);
}
```

C Bindings

```
struct pkg_MyStruct__data {
  int32_t                              d_int;
  struct sidl_dcomplex                 d_dcomplex;
  int64_t                              d_enum;
  struct sidl_BaseClass__object*       d_object;
  struct pkg_MyOtherStruct__data       d_struct;
  struct sidl_string__array*           d_string_array;
  double*                              d_rarrayRaw;
  double                               d_rarrayFix[3];
};

pkg_MyStruct__init(...);
pkg_MyStruct__copy(...);
pkg_MyStruct__serialize(...);
...
```

C++ Bindings

```
struct MyStruct : pkg_MyStruct_data {
  MyStruct();
  MyStruct(const ::pkg::MyStruct &src);

  void serialize(::sidl::io::Serializer &pipe,
                 const ::std::string &name,
                 const bool copyArg);
  ...

  ::sidl::BaseClass get_d_object() const;
  void set_d_object(const ::sidl::BaseClass &val);
  ...
};
```

Python Bindings

- Implemented as a Python C extension type
  - Allows to directly access the underlying IOR representation
  - Appears like a regular Python object with correctly named attributes
  - Also correctly converts Python objects to Babel's IOR

Fortran 90

- Implemented as a derived data type

```
type :: pkg_MyStruct_t
  integer (kind=sidl_int) ::       d_int
  complex (kind=sidl_dcomplex) :: d_dcomplex
  integer (kind=sidl_enum) ::      d_enum
  type(sidl_BaseClass_t) ::        d_object
  type(sidl_string_1d) ::          d_string_array
  type(pkg_MyOtherStruct_t) ::     d_struct

  //TODO: (fixed size) rarrays not yet supported

end type pkg_MyStruct_t
```

Java Bindings NEW!

```
package pkg;
public class MyStruct {
  public int                    d_int;
  public sidl.DoubleComplex     d_dcomplex;
  public long                   d_enum;
  public sidl.BaseClass         d_object;
  public MyOtherStruct          d_struct;
  public sidl.String.Array1     d_string_array;
  public sidl.Double.Array1     d_rarrayRaw;
  public sidl.Double.Array1     d_rarrayFix;

  public MyStruct() { ... }
  public void serialize(sidl.io.Serializer pipe,
                        final String name,
  ...                   boolean copyArg) { ... }
}
```

Peculiarities of the Java Bindings

- Babel automatically generates a public inner class named `Holder` that has to be used for out/inout Arguments

```
MyStruct.Holder h = new MyStruct.Holder(myStruct);
foo.passInOutStruct(h)
MyStruct retVal = h.get();
```

- Most data is copied when converting from IOR structs to the Java representation
  - Arrays and Objects are wrapped in the usual way
  - Simple data types and raw arrays are duplicated
  - No distinction between raw arrays and standard arrays from Java point of view
  - ☺ No JNI penalty for reads/writes
  - ☹ Relatively large call overhead

Structs and Babel RMI

- Babel automatically generates code for (de)serialization
- User-defined classes implementing the sidl.io.Serializable interface can use these methods to pack/unpack struct data members
- Regression test suite is currently extended to test RMI automatically

Current State (as of Oct. 2009)

	C	C++	Python	Java	F77	F90	F03
simple types	☺	☺	☺	☺	-	-	☺
objects / ifcs	☺	☺	☺	☺	-	-	☺
raw arrays	☺	☺	☹	☺	-	-	☺
RMI	☺	☺	?	-	☹	☹	?

Thank You!

Questions?