PyORBit

an ORBit2 binding for Python

GUADEC 2003
Dublin, Ireland

James Henstridge <james@daa.com.au>

Introduction

PyORBit is a Python binding for ORBit2

- Was written for use by the Python bindings for Gnome.
  - Should be useful on its own too though.
- Aims to follow the standard Python Language Mapping for CORBA.
- Uses ORBit2 type information to generate client stubs and server skeletons.

History

ORBit 0.5.x bindings:

- orbit-python
  - Written by Jason Tackaberry
  - Took ideas from Owen Taylor's Perl bindings
  - Generated stubs and skeletons at runtime by parsing IDL
  - Was the basis for Johan Dahlin's 1.x Bonobo bindings

ORBit2 bindings:

- orbit2-python
  - A minimal port of orbit-python by Johan.
  - Used by earlier releases of the Gnome 2 Python bindings.
  - Did not take advantage of many of ORBit2's newer features.
  - Manually marshaled data to and from GIOP buffers.

PyORBit

Due to a number of shortcomings in orbit2-python, I started looking at doing a new binding that better suited the needs of the Gnome Python bindings. Some features include:

- Make use of ORBit2 type-code information.
- Can generate stubs from ORBit2 type-libs, parsing IDL or by introspection.
- Can marshal from Python representation to and from C representation.
  - C code can invoke methods on in-process Python objects.
  - Python code can invoke methods on in-process C objects.
- Makes use of new features in Python 2.2.
- About 33% smaller than orbit-python.

Type Codes

In the CORBA framework, type codes are used to represent the various types of data that that can be transmitted. They hold all the information the ORB needs to marshal and transmit data over the wire plus some metadata.

This includes most of the information that a language binding needs to marshal data:

- fundamental type of data
- for enumerations, the list of value names.
- for structures and unions, the list of field names and types.
- for sequences and arrays, the type of the items.

Unfortunately, type codes do not provide much information about interfaces.

ORBit2 Interface Information

ORBit2 introduces a new way to represent information about interfaces through the use of the `IInterface` type:

```
struct IInterface {
  TypeCode tc;
  sequence<IMethod> methods;
  sequence<string> base_interfaces;
};
```

The fields are:

- The type code for this interface.
- a sequence of structures describing the interface's methods.
- a sequence of base interface names.

ORBit2 Interface Information

IMethod structures contain the following:

- name of method
- names and types of arguments, along with flags (in, out, inout).
- The type code for the return type.

How to Get Interface Information

ORBit2 provides a number of ways to obtain interface information:

- **Type libraries:** these are shared libraries containing the IInterface structures for a set of interfaces as static data. Can be loaded with the `ORBit_small_get_iinterface()` function.
- **IDL files:** ORBit2 provides an API to read in an IDL file and return interface information in the `IInterface` format.
- **Introspection:** servers implemented using ORBit2 automatically provide an `ORBit_get_iinterface` method. This can be invoked to retrieve interface information about a particular server.

PyORBit can use any of these methods to get interface information.

Method Invocation

- Method invocation goes through the `ORBit_small_invoke_stub()` function.
  - Takes an `IMethod` structure that describes how to perform the method invocation.
  - Is responsible for marshaling the argument vector from C layout to a GIOP buffer for remote invocations.
  - For local invocations, no marshaling required.
- For language bindings, it is only necessary to implement functions to marshal to and from the language's native format and C format.
  - For local calls to C servers, ORBit2 can pass the values directly to the server.
  - For remote calls, there are two translations, but this gets lost in the noise.

Comparison with Interface Repository

CORBA standard defines an interface repository service.

- Never supported well by ORBit

Some benefits of ORBit's approach include:

- No need for a separate server to hold interface info.
- No need to prime the interface repository with IDL before use.
- Interface information can be retrieved directly from servers.

If interface repository support is desired, it should be possible to implement as a bridge that generates `IInterface` structures from the information in the repository.

PyORBit Stubs and Skeletons

- Like its predecessor, PyORBit generates stubs and skeletons at runtime.
- However, PyORBit doesn't hook `__import__`.
  - You must manually load type-libs or IDL.
  - Some modules may automatically load up type-libs, such as `bonobo`.

PyORBit Stubs

- Stubs are new style Python classes.
- CORBA methods are represented by a "CORBA method" type.
  - wraps a `IMethod`.
  - Acts as a descriptor, as with standard methods.
- Provides automatically generated doc strings, so that `help()` works.

PyORBit Skeletons

- Follows the standard Python mapping.
- Like the client side stubs, skeletons are generated at runtime.
  - As with stubs, `IInterface` data is used to generate the skeleton
- ORBit2 allows a servant to provide an "implementation finder" function responsible for finding the implementation of a method.
- ORBit2 passes arguments to the servant in the same format given to `ORBit_small_invoke_stub()`, so we can reuse the same marshaling code.

Simple Scripting Example

In this example, we will use PyORBit to tell the running Nautilus process to open a window at a particular location.

First we need to get the IOR of the Nautilus shell. We could use the Gnome Python bindings for this, but to reduce the amount of magic, I'll use the standard `activation-client` program.

```
$ activation-client -s "iid == 'OAFIID:Nautilus_Shell'"
Query spec is "iid == 'OAFIID:Nautilus_Shell'"
Activation ID "OAFAID:[OAFIID:Nautilus_Factory,james,example.net]" RESULT_OBJECT
IOR:01daffbf17000000...
$
```

Simple Scripting Example

First we start Python:

```
>>> import ORBit
>>> import CORBA
>>> orb = CORBA.ORB_init()
```

Now we create an object reference from the IOR (which will also generate a stub for the Nautilus::Shell interface):

```
>>> shell = orb.string_to_object('IOR:01daffbf17000000...')
```

Now we can call methods on the object reference:

```
>>> shell.open_windows(['fonts:///'], '')
```

More Advanced Scripting Example

Nautilus metadata monitoring: metadata-demo.py.

- Uses the `Nautilus::MetafileFactory` interface.
  - This is a private interface, subject to change
- Loading interface data from typelibs or from IDL files.
- Implements the Nautilus::MetafileMonitor interface.
- Asks for metadata change notification for files in the home directory.
- Prints the list of emblems for files that change.

Future

- Further development of the CORBA related Gnome bindings.
- Custom interfaces on Bonobo components.
- Integrate better with threaded applications.
  - Looking at the new Python 2.3 threading APIs.