UNO:
Anecdotal Evidence

Stephan Bergmann
Software Engineer
Sun Microsystems, Inc.

If art is the tip of the iceberg
I'm the part sinking below
—Lou Reed, John Cale, *Songs for Drella*

UNO

- An object-oriented framework to bring together:
  - > different programming languages (C++, Java, Python, ...)
  - > different environments
    - – within a process (C++ runtime, JVM, ...)
    - – across processes (named pipes, sockets)
    - – across machines (sockets)
- Using *bridges* among the environments to marshal method invocations on objects as remote procedure calls.
  - > Where the shortcomings of this approach are well known.

Breaking no new ground
—Bongwater

Minimal Type System

- UNO type system reflects capabilities of target languages:
  - void, boolean, char and string, various integral and floating point types (minimal?)
  - enums (modeled after C)
  - structs (records), supporting parametric polymorphism
  - interfaces (references to objects, incl. null)
  - sequences
  - exceptions (somewhat second class)
  - any (type + value)

No Algebraic Data Types, etc.

- Overall, a rather mediocre type system:
  - Encoding sums with products,
    `struct Optional<T>{boolean isPresent; T Value;};`
    instead of
    `Optional<T> = NotPresent | Value T`
  - No parametric polymorphism for interfaces:
    `XInterface getElement();` or
    `any getElement();`
- Then again, how to map the good concepts to the relevant target languages?

Java Binding Oddities

- The base UNO XInterface is mapped to java.lang.Object. But UNO objects still have to implement XInterface.
- UNO any is mapped to java.lang.Object, too:

long	→ java.lang.Integer or com.sun.star.uno.Any
unsigned long	→ com.sun.star.uno.Any
XInterface reference	→ java.lang.Object or com.sun.star.uno.Any
XFoo reference	→ com.sun.star.uno.Any

Identifiers

- Naively modeled after C.
- Problems with name clashes in language bindings (keywords, methods named the same as containing interfaces, etc.).
- Restrictions retrofitted to allow escape mechanisms in language bindings (`goto` → `method_goto`).
- What about languages where case has meaning?

Object Life-Cycle

- Implemented with reference counting.
  - C++ Reference<T> calls acquire/release.
  - Java finalize calls release across bridge.
- C++ programmers confused when their destructors are called late (only after JVM garbage collection).
- Cyclic references cause problems, of course.
  - XComponent.dispose
  - XCloseBroadcaster, XCloseable, and CloseVetoException

URP Oddities

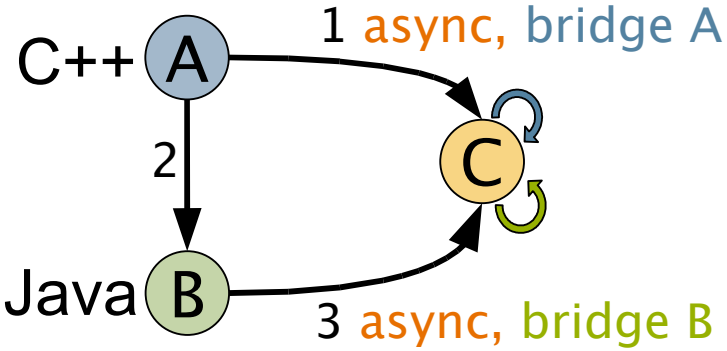
- Compact wire representation.
- Marshalling types:

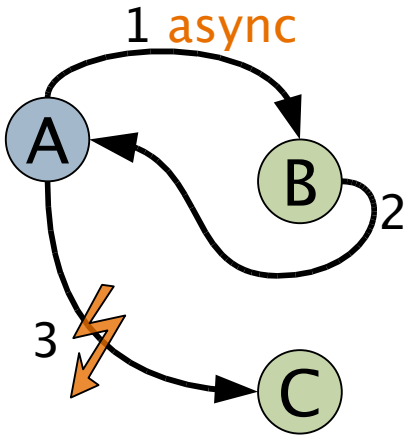
  long: 0x06
  com.sun.star.uno.XInterface: 0x16 "com.sun.st..."
  sequence<long>: 0x14 "[]long"
  instead of 0x14 0x06

- Negotiating protocol properties during startup:
  - If both sides want to set the same properties, both send the same marshalled blob plus a random number. Unnecessary life-lock if both sides use identical random numbers.

Asynchronous One-Way Calls

- UNO methods marked as [oneway], implemented as (semi-)asynchronous URP calls.
  - > All async calls executed in order.
  - > All async calls executed before next synchronous call.
  - > For each UNO thread, there is at most one concurrent thread executing the async calls.
- Does not really work:





Evolution

- User-defined UNO types want to evolve over time.
- This causes problems in two dimensions:
  - Interfaces have both clients and implementations.
  - Implementation languages offer limited support.
- UNO only offers mechanisms, no policies.
  - Evolve `XFoo` to `XFoo2` to `XFoo3`.
  - Published vs. unpublished types.

Everyone wants the honey but not the sting

—Mark Edwards

Runtime Type Information

- UNO types described in textual `.idl` files.
- Translated and combined into binary `.rdb` files.
- Read at runtime:
  - Packing values (type description + `void*`) in `any`s.
  - Marshalling data across bridges.
  - Creating vtables of C++ proxy objects.
- For Java, `.class` files are used instead.
- So there are two (three, even) descriptions for each type.

.idl Files

- Are C-style preprocessed, complicating things without need.
  - Preprocessor used to handle includes.
  - But code makers have to be careful to not generate code for merely included types.
  - Could have used the Java way instead, as (almost) each type is in its own .idl file, anyway.

.rdb Files

- Back when StarOffice still had mail/news functionality, it used storages to store mails in. Those often broke and caused data loss.
- A new implementation solved the stability issues.
  - (But it is unsound, reducing arbitrary-size string keys to fixed length hash values.)
- This was re-used for the UNO type database.
  - Complex code (no direct access from Java).
  - Files are unnecessarily large, due to lots of padding.

Active Component Registration

- Each UNO component (collection of UNO service implementations; shared library or .jar) must be registered in a services.rdb.
- This is done by calling code in the component.
- OOo extensions are uninstalled as follows:
  - Call code in extension to register its services into a temporary .rdb.
  - Iterate entries in temporary .rdb and remove them from the services.rdb.
  - What if extension code no longer loads after an OOo upgrade?

Dynamically Generated Code

- `cppumaker` only generates headers, no code.
- C++ proxy objects with vtables are generated on the fly.
- The relevant code is compiler and platform/CPU dependent.
  - Probably the biggest obstacle for porters.
  - Subtle bugs, plaguing us for years.
- GCC exception handling needs unique RTTI.
  - Implemented via weak symbols, which are expensive at library load time.

Q?

- throw SQLException(...);
  ... UNO bridge in between ...
  catch (SQLException e) ...

kept mysteriously failing under Solaris (catch (...) worked).

Q? Ah!

- throw SQLException(...);
  ... UNO bridge in between ...
  catch (SQLException e) ...

  kept mysteriously failing under Solaris (catch (...) worked).
- In the Solaris C++ ABI, mangled names use the letter Q as an escape mechanism.
- The Solaris C++ bridge ignored that detail when generating mangled names for exception handling.

Instruction Manual Details

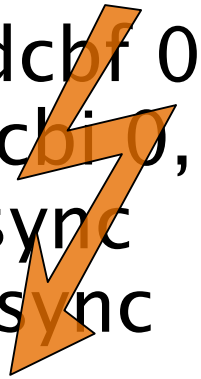
- On Mac OS X PowerPC, is it

```
dcbf 0, p        or        dcbst 0, p
icbi 0, p                  sync
sync                       icbi 0, p
isync                      isync          ?!?
```

Instruction Manual Details

- On Mac OS X PowerPC, is it

```
dcbf 0, p        or        dcbst 0, p
icbi 0, p                  sync
sync                       icbi 0, p
isync                      isync
```



?!?

- The difference is random crashes when the C++ bridge dynamically generates code and does not correctly synchronize data and instruction memory.

Security-Enhanced Linux

- Dynamically generating code under SELinux:

  Instead of

```
p = mmap(0, n, PROT_READ|PROT_WRITE,
    MAP_PRIVATE|MAP_ANON, -1, 0);
mprotect(p, n,
    PROT_READ|PROT_WRITE|PROT_EXEC);
```

  one now needs

```
fd = mkstemp(name);
unlink(name);
ftruncate(fd, n);
p1 = mmap(0, n, PROT_READ|PROT_WRITE,
    MAP_SHARED, fd, 0);
p2 = mmap(0, n, PROT_READ|PROT_EXEC,
    MAP_SHARED, fd, 0);
```

French music sucks but it is nice

—Sonny Vincent