!

Javolution C++

They call him Ginger!

« It looks like Java, it tastes likes Java… but it is C++ »

October 20, 2012

What is the problem?

- More and more hybrid C++/Java projects
  - Developer expertise required in both Java and C++
- C++ total cost is significantly higher
  - But cost of migrating existing C++ components to Java is prohibitive.
- Standardized and well established software practices exist in the Java world
  - C++ developers are on their own (multiple solutions to address the same problems lead to additional complexity and issues in our systems)
- Many Open-Source implementations of Software Standards exist only in Java
  - OSGi, GeoAPI, UnitsOfMeasure, etc.

Many causes of variability.

- Developers expertise varies considerably.
- Testing performed at the end (integration) due to component inter-dependencies.
- Insufficient documentation.
- “Not Invented Here” Syndrome.
- Proprietary solutions not maintained which later become legacy burden.
- It is very beneficial to follow well-established standard specification.

“Doing the right thing is difficult, but doing it right is easier.”

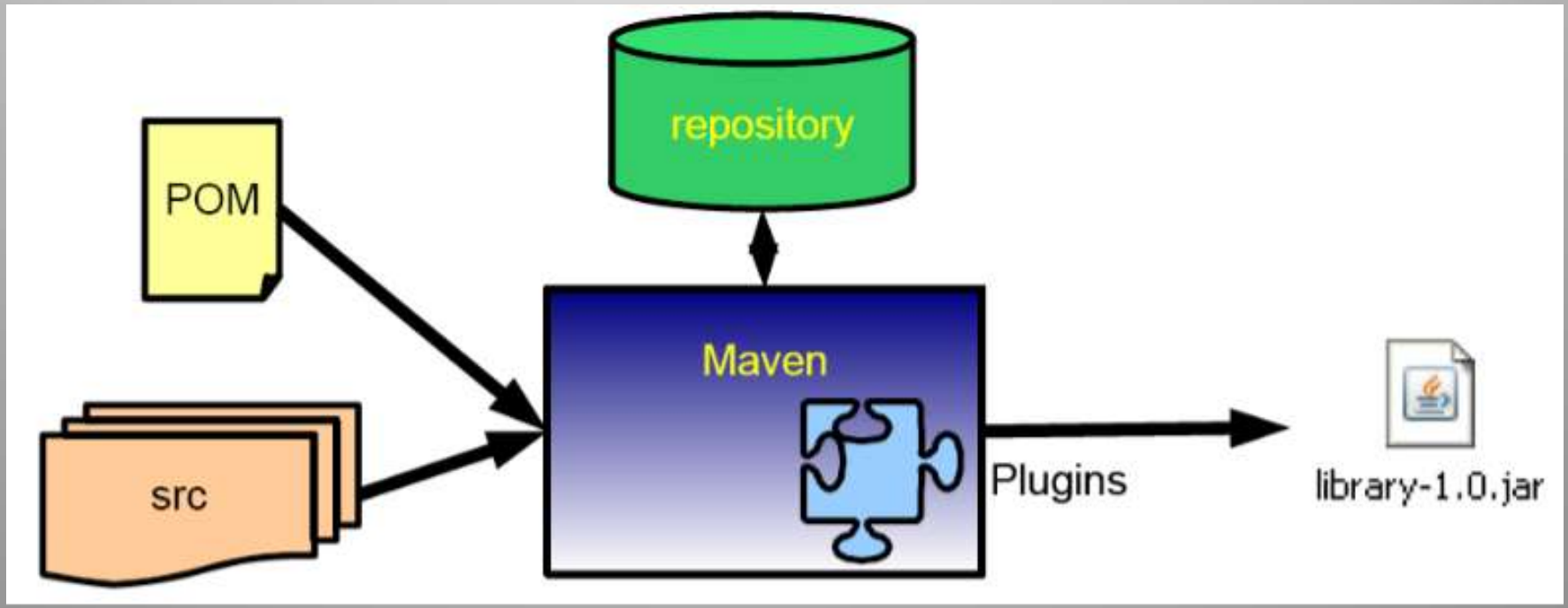
Our Solution.

- Uniformization of C++/Java development through the use of a common framework (Javolution) based on Java standard library specification.
- Facilitating the migration of Java OSS code to C++
- Promote the “Service Oriented Approach” by providing an OSGi framework for both Java/C++
- Reduce documentation by having the same specification/design for our Java and C++ components.
- Unification of code building for Java and C++ (maven used for both, e.g. maven native plugin).

Maven Build

- **Apache Maven** is used to produce artifacts (dynamic libraries, static libraries, executable) and to perform unit tests and test coverage.

- Profiles and packaging classifiers are used to address platform variability (windows, linux, etc.)



What is Javolution C++ (overview)?

- A Javolution mirrored C++ library sharing the same specifications, documentation and unit testing.
- A “behind-the-scenes” C++ infrastructure based on smart pointers (real-time garbage collection through reference counting).
- Integrated memory cache making small, short lived objects (e.g. value types) very efficient.
- C++ packages/classes derived from standard Java (e.g. java::lang::Thread, java::lang::* collections)
- A C++ dynamic execution and testing framework (OSGi & JUnit) identical to Java.

C++ Class Definition

- **The general pattern for class/interface is as follow:**

```
#include "javolution/lang/Object.hpp"
namespace com {  namespace bar {
    class Foo_API;
    typedef SDK::Handle<Foo_API> Foo; // Object reference.
}}

class com::bar::Foo_API : public virtual javolution::lang::Object_API {
private:
    Param param;

protected:
    Foo_API(Param param) { // Constructor visible to sub-classes.
        this->param = param;
    }

public:
    static Foo newInstance(Param param) {
        return Foo(new Foo_API(param));
    };
    virtual void fooMethod () { ... };

}
```

C++ Parameterization – Better than Java!

- **Unlike Java, C++ class parameterization is not syntactic sugar but efficient use of C++ templates!**
- **All javolution.util collections are parameterized.**

```
List<String> list = FastTable_API<String>::newInstance();
list->add(L"First");
list->add(Type::Null);
list->add(L"Second");
```

- **Also used for Java-Like Enums**

Synchronization

- **Supported through a macro: synchronized(Object) mimicking the Java synchronized keyword.**

- **Can be performed on any instances of Javolution collections and Class (for static synchronization).**

```
synchronized (trackedServices) {// trackedServices instance of FastMap
   for (int i = 0; i < serviceReferences.length; i++) {
       Object service
           = actualCustomizer->addingService(serviceReferences[i]);
       trackedServices->put(serviceReferences[i], service);
   }
   trackingCount = 0;
}
```

Miscellaneous

- **Limited reflection support through RTT**
- **Auto-boxing of primitive types (boolean, integer, float, wide strings).**

```
Integer32 i = 23;
Float64 f = 3.56;
Boolean b = true;
String s = L"xx"
```

- **All variables are initialized to SDK::Null (NullPointerException if not set before use).**
- **Wide-String (literal) concatenation supported.**

```
throw RuntimeException_API::newInstance(
    L"Bundle " + symbolicName + L" not in a resolved state");
```

- **Dynamic length array SDK::Array<type>**

```
Type::Array<ServiceReference> serviceReferences
    = context->getServiceReferences(serviceName, Type::Null);
if (serviceReferences.length == 0) return;
```

Minor differences with Java

- Generalized use of static factory methods (such as newInstance(), valueOf()) instead of constructor.
- No ‘finally’ keyword in C++ (but try…catch same as Java).
- Static methods are called using the name of the class with the suffix ‘_API’
- Synchronization not supported on every object but only on those whose class implements the Object_API::getMutex() virtual methods

OSGi Port

- Done in one week.
- Conversion of Java classes provided by the OSGi alliances (e.g. ServiceTracker)
- C++ only execution framework (no mix of Java and C++ bundles).
- Dynamic library activation possible through JNA
- Javolution Java & C++ OSGi-light implementations can be used to perform JUnit (Java or C++) with bundle activations.

JUnit Port

- **Done in two days.**
- **Integrate with Hudson/Jenkins**

```
class javolution::util::FastTableTest_API
        : public junit::framework::TestCase_API {
   javolution::util::List<javolution::lang::String> list;
public:
   BEGIN_SUITE(javolution::util::FastTableTest_API)
   ADD_TEST(testIsEmpty)
   ADD_TEST(testIterator)
   ADD_TEST(testSubList)
   END_SUITE()

   void testIsEmpty();
   void testIterator();
   void testSubList();
protected:
   void setUp();
};
```

What next?

- More ports of Java Open source libraries (e.g. units of measurement, geo transforms)
- Math routines with GPU Acceleration.
- More standard Java library conversion (from OpenSDK source code ?)