

Cappulada: What we've learned And why binding C++ is hard

Johannes Kliemann FOSDEM, Brussels, 2020-02-01

Recall Cappulada 2019 Goals



- Automatically bind C++ APIs to Ada
- Maintain API layout and type safety
- Maintain semantically appropriate mappings
- Generate mangled symbols
- Detect and avoid name collisions
- Be SPARK compatible where possible
- Be better than existing solutions

Existing solutions (GCC)

- No template support
- No proper support to use nonvalid identifiers in Ada
- Generates uncompilable code
- No automatic handling of constructors/destructors
- Fixing requires maintaining a fork in the long term

Recall Cappulada 2019 Achievements and Shortcomings



Achievements

- Templates
- Classes, namespaces, nesting
- Inheritance, with virtual classes
- Builtin types, typedefs, enums, arrays, pointers, references
- Member functions, function pointers
- Private, public, protected scopes
- Mangling

Shortcomings

- Partial template specialization
- Typedefs on specific types
- Auto keyword
- Operator overloads
- Function templates
- Merging multiple equally named namespaces
- Destructors
- Multiple inheritance

Why is it hard? Complexity and Semantics



Both languages are complex

- Ada 2012 Standard has ~1300 pages
- C++17 Standard has ~1600 pages
- C++ builds upon C so we need to support C, too
- C11 Standard has ~700 pages

■ Inherent semantic differences

- Arrays: separate type in Ada, builtin construct in C++
- C++ templates can be used for meta programming, Ada generics cannot
- Both have different calling conventions



What doesn't work?





- Templates in C++ are static
- Linker symbol is generated from the template arguments
- Template arguments are always static

```
template <typename T>
class A
{
    void inc(T *t);
};

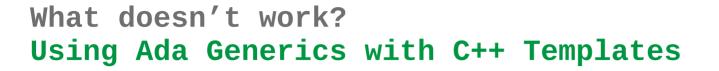
A<int>::inc(int *t);
_ZN1AIiE3incEPi
```



What doesn't work? Using Ada Generics with C++ Templates

```
generic
   type T is private
package A is
   type Class is limited null record with Convention => CPP;
   procedure Inc (This : in out Class; X : in out T) with
      Import, Convention => CPP,
      External_Name \Rightarrow "_ZN1AI" & M (X) & "E3incEP" & M (X);
      -- ZN1AIiE3incEPi
end A;
```

a.ads: entity for aspect "Import" must be a static expression a.ads: non-static function call (RM 4.9(6,18))





Theory

Overloading M for Mangling

Practice

- Overloading doesn't work on private types
- Return value of M is not static
- Generic formal parameters are never considered static

Potential Solution

Preprocessor

```
function M (X : Integer)
  return String
is ("i");
```

What doesn't work? C++ Pass by Value



```
type A is limited record
  X: Integer;
end record with
  Import,
  Convention => CPP;

function Con return A with
  Import, Convention => CPP,
  External_Name => "...";
pragma CPP_Constructor(Con);
```

```
class A
{
    public:
        int X;
        A();
};
```

What doesn't work? C++ Pass by Value



```
procedure Print (X : A) with
   Import,
   Convention => CPP,
   External_Name => "...";
```

```
class A
{
    public:
        int X;
        A();
};

void print(A a);
```

What doesn't work? C++ Pass by Value



- Problem: A will be passed by reference from Ada but expected by value in C++
- Considered Solution: Import Print with Convention C_Pass_By_Copy
- Problem: C_Pass_By_Copy Convention allowed only for record type
- Potential Solution: Define a record identical to the class
- Problem: Unable to convert between both safely (unlike in C++)

What doesn't work? Automatic destructor calling



- Automatically called destructors are not supported in the compiler
- Destructor could be called manually
- Controlled objects can implement this functionality

What have we learned?



- Even if everything fits it's much work
- Some things could in theory work
 - With really high effort
 - With additional tools
 - At the cost of usability and safety
- Some things just won't work at all



Don't you fear that you import the weirdness of C++ into Ada?



YES!

Questions?



Gneiss: A Nice Component Framework in SPARK Sunday 12:00 K.4.601 (Microkernel devroom)

Johannes Kliemann kliemann@componolit.com

@Componolit · componolit.com · github.com/Componolit