Multimethods as dynamic C++ overloading.

Or:

How to stop worrying about Multimethod declarations and instead let the caller decide.

Julian Smith
jsmith@undo-software.com

Multimethod syntax

Stroustrup D&E and 2007

```
bool intersect( virtual Shape&,     virtual Shape&); // open method
bool intersect( virtual Rectangle&, virtual Circle&); // overrider
```

WG21/N1529 2003

```
bool    overlap( virtual shape&   a, virtual shape& b);
bool    overlap( static square&   a, static triangle& b) {...}
bool    overlap( static triangle& a, static square&   b) {…}
```

Jean-Louis Leroy 2015 (this conference)

Problems with existing proposals

- Is declaration of the 'base' implementation special ?
  - Why ?
  - Needed by implementation to decide when to generate dynamic dispatch code.
- Callers need to have seen multimethod declaration first.
  - Could cause different behaviour in different compilation units.
- Difficult to call a specific implementation function without virtual dispatch.
  - Might be useful inside an implementation.

Issues with implementations

- Don't complicate the linker.
- Cope with loading/unloading of dynamic libraries.
- What the minimum we can put into the language, that will allow the rest to be implemented in library code?

Cmm – Adds Multimethods to C++

- Written in parallel with WG21/N1529
- Shows that multimethods can be implemented:
  - Without special linker support.
  - Mostly in a library, with specific language additions.
- Gives direct access to pointer to multimethod function that would be called for a particular set of parameters.
  - E.g. avoid lookup in tight loop.
- Constant-time dispatch if classes are assigned unique small integers.
- Easy to extend to shared pointers as well as references.
- Caller dispatch.

Implementation - Cmm

- Other:
  - Worlds worst C++ parser.
  - Optionally supports Stroustrup alternative declaration syntax (see D&E):
    - main: (argc: int, argv: []->char) int ...
  - Optionally supports 'autoblocks' – python-style block-structure-from-indentation for C++.

Caller dispatch

- Multimethod dispatch algorithms for C++:
  - Are like compile-time overloading.
  - Except that they use dynamic types instead of static types.
- So... instead of declaring something is a multimethod...
- … can we let the *caller* decide whether to use static type or dynamic type?

Caller dispatch – Cmm syntax

```
Base& x = . . .;
Base& y = . . .;
bool a = foo( x, y); // resolve using static types.
bool b = foo( virtual x, virtual y); resolve using dynamic types.
```

- Very similar to conventional C++ overloading.
  - No special declarations required.
  - Base-implementation is not special.
  - Easy to call specific implementations directly.
- Differences:
  - Uses dynamic types, not static types.
  - Gives access to all functions in executable, not just the ones visible to this compilation unit.
  - Throws exception if no match or ambiguous match.

Caller dispatch - implementation

- Implementation is easy as long as we have:
  - A compiler that knows about new calling syntax.
    - E.g. compile to a call to special dispatch function.
  - A library with runtime access to:
    - Prototypes of all functions in the programme.
    - Inheritance information.
  - Will C++ introspection gives us these things?

Multimethods and caller dispatch - summary

- Generalisation of C++ overloading, not virtual functions.
  - Little mention of classes.
- No need for member functions.
  - what have they ever done for us?
    - (apart from destructors.)
- Programme differently:
  - Use plain structs for data.
  - Use free functions for access and manipulation of this data.
  - Use caller-dispatch as required to make behaviour depend on dynamic types.
- Simpler lookup rules.
- Almost like a new language...

New language?

C++ Without Classes.