Метаклассы: воплощаем мечты в реальность

Сергей Садовников

Обо мне

Сергей Садовников

Архитектор ПО в компании “Лаборатория Касперского”

corehard.by

Метаклассы - что это?

Metaclasses

- **$class** denotes a metaclass.

```
$class interface { /*...public & pure virtual fns only + by default...*/ };
```

more specific than "class"

```
interface Shape { /*... public virtual enforced + default ...*/ };
```

- Typical uses:
  - Enforce rules (e.g., "all functions must be public and virtual")
  - Provide defaults (e.g., "functions are public and virtual by default")
  - Provide implicitly generated functions (e.g., "has virtual destructor by default," "has full comparison operators and default memberwise implementations")

15

https://www.youtube.com/watch?v=6nsyX37nsRs

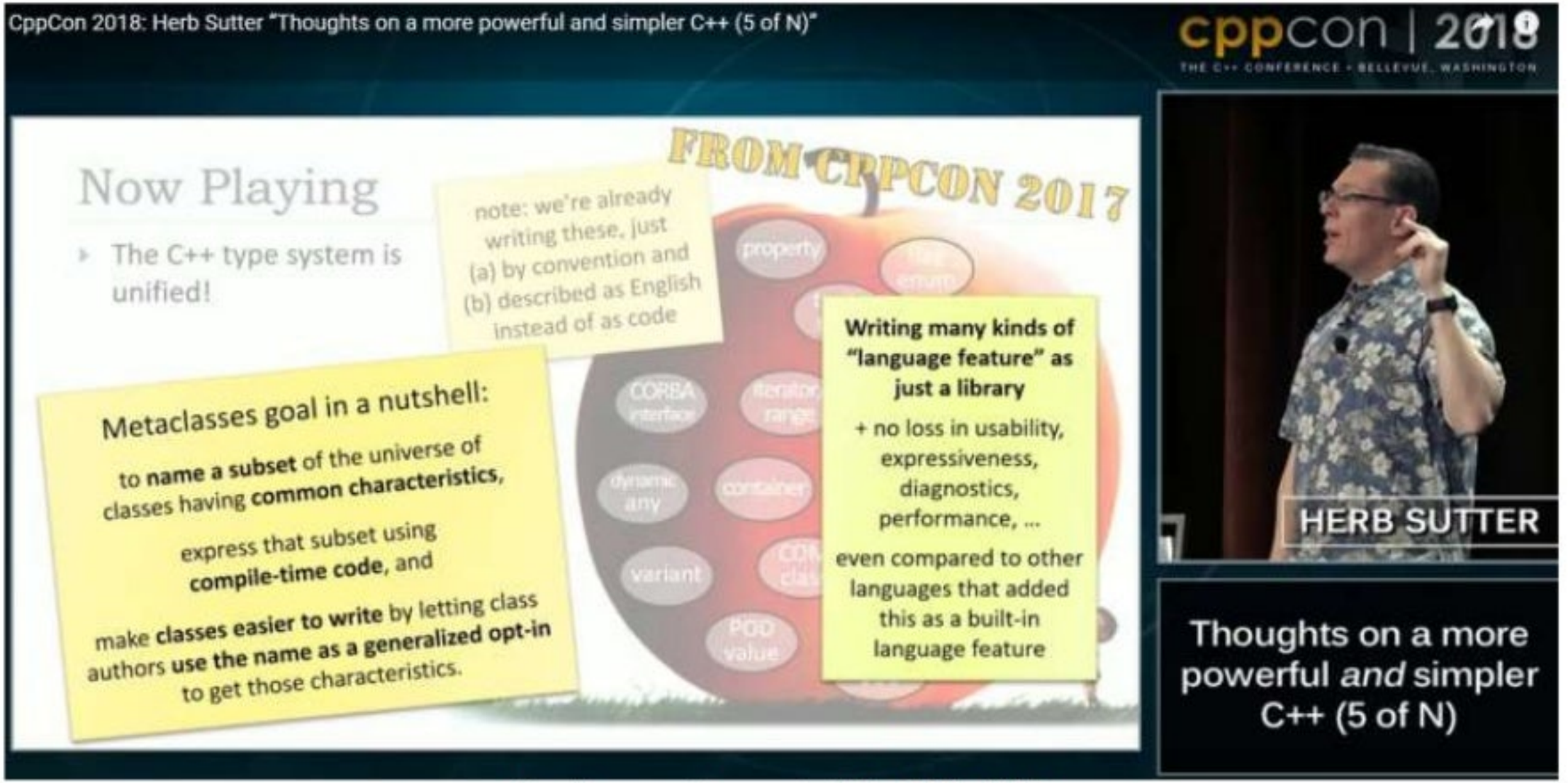
Метаклассы - что это?

https://www.youtube.com/watch?v=6nsyX37nsRs

Метаклассы - что это?

- Новая языковая сущность
- Управляет генерацией AST и кода
- “Метапрограммирование” в императивном стиле
- Потенциальная замена макросам и некоторым шаблонам

Метаклассы - зачем?



https://www.youtube.com/watch?v=80BZxujhY38

Метаклассы - зачем?

Основное назначение:

1. Ещё одна степень обобщения: описывать общие свойства целых групп классов (“**interface**”, “**value**”, “**property**” и т. д.), создавать библиотеки метаклассов
2. “Концепты на стероидах”: позволяют верифицировать свойства классов в императивном стиле
3. Inplace-расширение компилятора (и языка): непосредственная манипуляция AST на этапе компиляции с целью привести объявление класса к нужному виду (задание новых свойств, добавление методов, перегрузка операторов и т. п.

Базируются на:

1. Static reflection (p0712)
2. Generative metaprogramming: code generation and injection (p0712, Code generation in C++)

Описано в p0707 (revision 3)

Метаклассы - пример

```
struct IEnumerator : public IUnknown
{
  virtual ~IEnumerator() {}

  virtual HRESULT Reset() = 0;
  virtual HRESULT MoveNext(BOOL* succeeded) = 0;
  virtual HRESULT Current(IUnknown** item) = 0;
};
```

Метаклассы - пример

```
interface IEnumerator
{
   void Reset();
   BOOL MoveNext();
   IUnknown* Current();
};
```

Метаклассы - пример

Метакласс

```
interface IEnumerator
{
   void Reset();
   BOOL MoveNext();
   IUnknown* Current();
};
```

Метаклассы - пример

```
$class interface {
  virtual ~interface() {}
  constexpr {
    compiler.require($interface.variables().empty(), "interfaces may not contain data");
    for (auto f : $interface.functions()) {
      compiler.require(!f.is_copy() && !f.is_move(),
                       "interfaces may not copy or move; consider a virtual clone() instead");

      if (!f.has_access())
        f.make_public();

      compiler.require(f.is_public(), "interface functions must be public");
      f.make_pure_virtual();
      f.parameters().add($f.return_type() *);
      f.set_type(HRESULT);
      f.bases().push_front(IUnknown);
    }
  }
};
```

Метаклассы - пример

```
interface IEnumerator
{
  void Reset();
  BOOL MoveNext();
  IUnknown* Current();
};
```

```
$class interface {
  virtual ~interface() {}
  constexpr {
    compiler.require($interface.variables().empty(), "interfaces may not contain data");
    for (auto f : $interface.functions()) {
      compiler.require(!f.is_copy() && !f.is_move(),
                       "interfaces may not copy or move; consider a virtual clone()
instead");

      if (!f.has_access())
        f.make_public();

      compiler.require(f.is_public(), "interface functions must be public");
      f.make_pure_virtual();
      f.parameters().add($f.return_type() *");
      f.set_type(HRESULT);
      f.bases().push_front(IUnknown);
    }
  }
};
```

```
struct IEnumerator : public IUnknown
{
  virtual ~IEnumerator() {}

  virtual HRESULT Reset() = 0;
  virtual HRESULT MoveNext(BOOL* succeeded) = 0;
  virtual HRESULT Current(IUnknown** item) = 0;
};
```

Метаклассы - пример

В соответствии с p0707r1/p0707r2

```
$class interface {
  virtual ~interface() {}
  constexpr {
    compiler.require($interface.variables().empty(), "interfaces may not contain data");
    for (auto f : $interface.functions()) {
      compiler.require(!f.is_copy() && !f.is_move(),
                       "interfaces may not copy or move; consider a virtual clone() instead");

      if (!f.has_access())
        f.make_public();

      compiler.require(f.is_public(), "interface functions must be public");
      f.make_pure_virtual();
      f.parameters().add($f.return_type() *);
      f.set_type(HRESULT);
      f.bases().push_front(IUnknown);
    }
  }
};
```

Метаклассы - пример

В соответствии с p0707r3

```
template<typename T, typename S>
constexpr void interface(T target, const S source) {
  compiler.require(source.variables().empty(), "interfaces may not contain data");
  for (auto f : source.member_functions()) {
    compiler.require(!f.is_copy() && !f.is_move(), "interfaces may not copy or move; consider a virtual clone() instead");
    if (!f.has_access())
      f.make_public();
    compiler.require(f.is_public(), "interface functions must be public");
    auto ret = f.return_type();
    if (ret == $void) {
      -> (target) class {
        virtual HRESULT idexpr(f)(__inject(f.parameters())) = 0;
      };
    } else {
      -> (target) class {
        virtual HRESULT idexpr(f)(__inject(f.parameters()), typename(ret)* retval) = 0;
      };
    }
  }
}
```

Метаклассы - пример

```
constexpr void basic_enum(meta::type target, const meta::type source) {
  value(target, source); // a basic_enum is-a value
  compiler.require(source.variables().size() > 0, "an enum cannot be empty");
  if (src.variables().front().type().is_auto())
    ->(target) { using U = int; } // underlying type
    else ->(target) { using U = (src.variables().front().type())$; }
    for (auto o : source.variables()) {
      if (!o.has_access()) o.make_public();
      if (!o.has_storage()) o.make_constexpr();
      if (o.has_auto_type()) o.set_type(U);
      compiler.require(o.is_public(), "enumerators must be public");
      compiler.require(o.is_constexpr(), "enumerators must be constexpr");
      compiler.require(o.type() == U, "enumerators must use same type");
      ->(target) o;
    }
  ->(target) { U value; } // the instance value
  compiler.require(source.functions().empty(), "enumerations must not have functions");
  compiler.require(source.bases().empty(), "enumerations must not have base classes");
}
```

Метаклассы

Проблемы:

- Базируется на ещё не принятых пропозалах -> предлагаемый синтаксис может меняться
- Неизвестно, когда всё это примут. По некоторым оценкам - не раньше C++2z
- Не все возможные варианты использования детально рассмотрены в публично доступных документах

Метаклассы

Проблемы:

- Базируется на ещё не принятых пропозалах -> предлагаемый синтаксис может меняться
- Неизвестно, когда всё это примут. По некоторым оценкам - не раньше C++2z
- Не все возможные варианты использования детально рассмотрены в публично доступных документах

А если хочется раньше?

- Ждать принятия в стандарт и поддержки компиляторами
- Использовать шаблонную магию и макросы
- Реализовать и использовать сторонние утилиты

Метаклассы

Проблемы:

- Базируется на ещё не принятых пропозалах -> предлагаемый синтаксис может меняться
- Неизвестно, когда всё это примут. По некоторым оценкам - не раньше C++2z
- Не все возможные варианты использования детально рассмотрены в публично доступных документах

А если хочется раньше?

- Ждать принятия в стандарт и поддержки компиляторами
- Использовать шаблонную магию и макросы
- **Реализовать и использовать сторонние утилиты**

Метаклассы - сторонняя утилита

Достоинства:

- Не требуется особой поддержки со стороны компиляторов
- Реализация в рамках актуальных стандартов
- Может быть легко встроена в сборочный тулчейн
- Может сделать кодовую базу проектов проще и упростить поддержку

Метаклассы - сторонняя утилита

Достоинства:

- Не требуется особой поддержки со стороны компиляторов
- Реализация в рамках актуальных стандартов
- Может быть легко встроена в сборочный тулчейн
- Может сделать кодовую базу проектов проще и упростить поддержку

Недостатки:

- Не соответствует одной из целей создания метаклассов (“Eliminate the need to invent non-C++ “side languages” and special compilers, such as Qt moc, COM MIDL, and C++/CX”)
- Собственный “птичий” язык для описания специальных конструкций
- Не всегда удобное использование предлагаемых возможностей
- Потенциальная несовместимость с итоговым вариантом пропозала

Метаклассы - сторонняя утилита

Основные возможности:

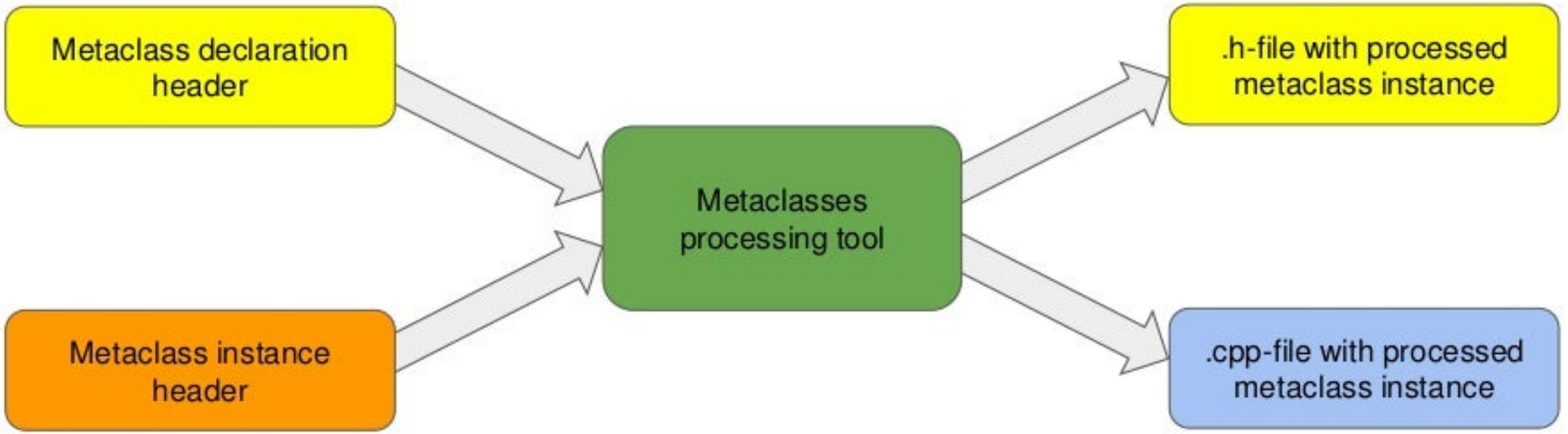
- Позволяет объявлять метаклассы
- Позволяет объявлять инстансы конкретных метаклассов
- В метаклассах позволяет проверять свойства инстансов и генерировать диагностику
- Позволяет внедрять в инстансы метаклассов новый код/изменять существующий
- Всё это в рамках одной кодовой базы

Требования:

- Совместимость со стандартами C++14/17 (идеально - с C++11)
- Совместимость с популярными компиляторами (gcc/clang/MSVC)
- Лёгкость интеграции в сборочный тулчейн
- Относительная простота использования возможностей
- C++ на входе - C++ на выходе, ничего больше

Метаклассы - сторонняя утилита

Основная идея реализации:



Метаклассы - использование утилиты

```
$_metaclass(Interface) {
  static void GenerateDecl() {
    compiler.message("Hello world from metaprogrammer!");
    compiler.require($Interface.variables().empty(), "Interface may not contain data members");

    $_inject(public) {
      enum {InterfaceId = 100500};
    }

    for (auto& f : $Interface.functions()) {
      compiler.require(f.is_implicit() || (!f.is_copy_ctor() && !f.is_move_ctor()),
        "Interface can't contain copy or move constructor");

      compiler.require(f.is_public(), "Inteface function must be public");

      f.make_pure_virtual();
    }
  }
};
```

Метаклассы - использование утилиты

```
$_metaclass(Interface) {
  static void GenerateDecl() {
    compiler.message("Hello world from metaprogrammer!");
    compiler.require($Interface.variables().empty(), "Interface may not contain data members");

    $_inject(public) {
      enum {InterfaceId = 100500};
    }

    for (auto& f : $Interface.functions()) {
      compiler.require(f.is_implicit() || (!f.is_copy_ctor() && !f.is_move_ctor()),
        "Interface can't contain copy or move constructor");

      compiler.require(f.is_public(), "Inteface function must be public");

      f.make_pure_virtual();
    }
  }
};
```

Макрос для объявление метакласса

Метод, отвечающий за генерацию декларации

Интерфейс к “компилятору”

Макрос для внедрения нового кода в класс

Ссылка на обрабатываемый инстанс метакласса

Метаклассы - использование утилиты

```
$_class(TestIface, Interface)
{
  void TestMethod1();
  std::string TestMethod2(int param)
const;
};
```

Метаклассы - использование утилиты

```
$_class(TestIface, Interface)
{
  void TestMethod1();
  std::string TestMethod2(int param)
const;
};
```

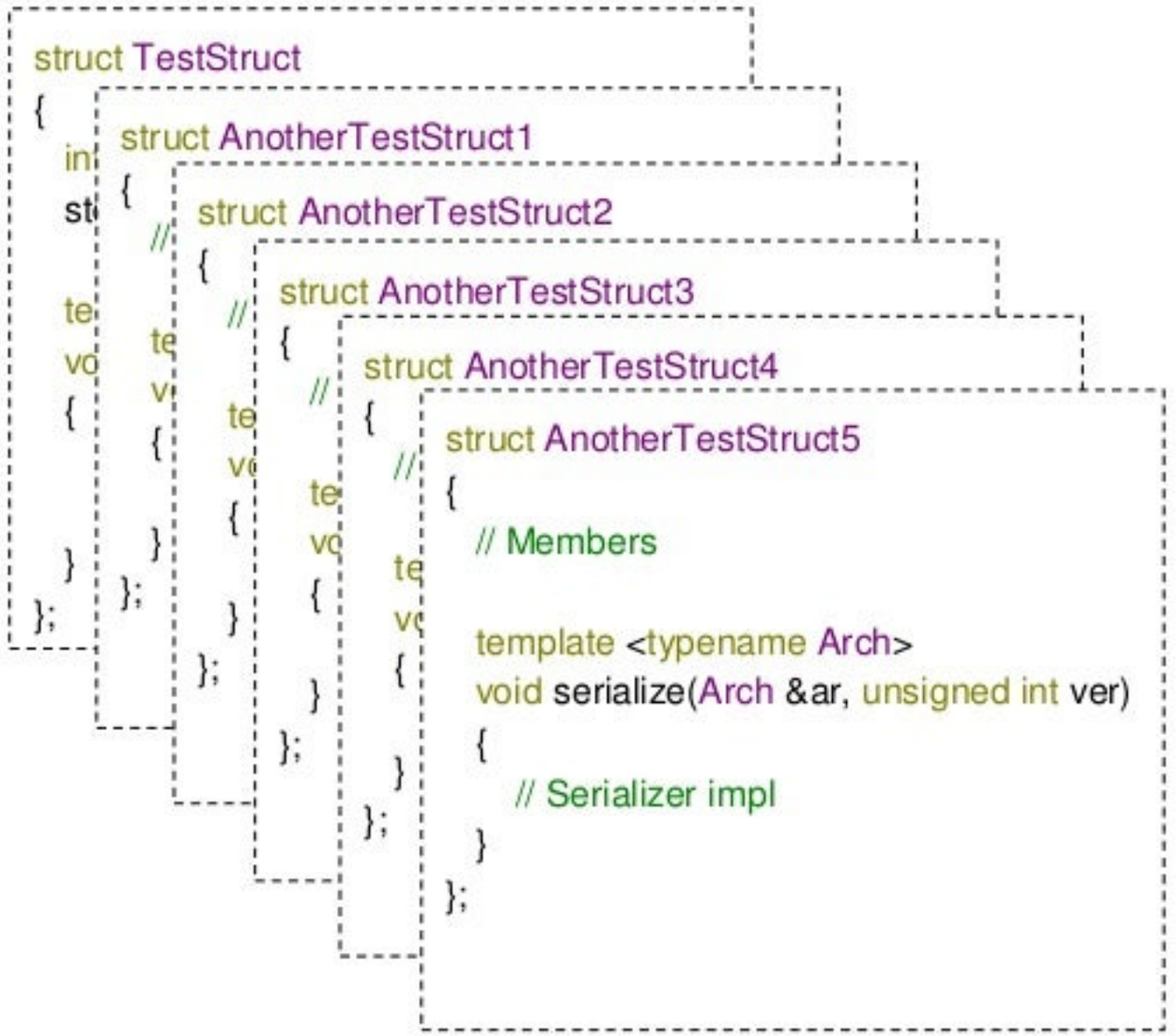
```
class TestIface {
public:
  enum { InterfaceId = 100500 };

  virtual void TestMethod1() = 0;
  virtual std::string TestMethod2(int param) const = 0;

protected:
private:
};
```

```
$_metaclass(Interface) {
  static void GenerateDecl() {
    compiler.message("Hello world from metaprogrammer!");
    compiler.require($Interface.variables().empty(), "Interface may not contain data members");

    $_inject(public) {
      enum {InterfaceId = 100500};
    }

    for (auto& f : $Interface.functions()) {
      compiler.require(f.is_implicit() || (!f.is_copy_ctor() && !f.is_move_ctor()),
        "Interface can't contain copy or move constructor");

      compiler.require(f.is_public(), "Inteface function must be public");

      f.make_pure_virtual();
    }
  }
};
```

Метаклассы - использование утилиты

```
struct TestStruct
{
   int a;
   std::string b;

   template <typename Arch>
   void serialize(Arch &ar, unsigned int ver)
   {
      ar & a;
      ar & b;
   }
};
```

Метаклассы - использование утилиты



Метаклассы - использование утилиты

```
$_struct(TestStruct, BoostSerializable)
{
  int a;
  std::string b;
};
```

Метаклассы - использование утилиты

```
$_metaclass(BoostSerializable) {
  static void GenerateDecl()  {
    $_inject(public) [&, name="serialize"](auto& ar, const unsigned int ver) -> void
    {
      $_constexpr for (auto& v : $BoostSerializable.variables())
        $_inject(_) ar & $_v(v.name());
    };
  }
};
```

Метаклассы - использование утилиты

```
$_struct(TestStruct, BoostSerializable)
{
  int a;
  std::string b;
};
```

```
$_metaclass(BoostSerializable) {
  static void GenerateDecl()  {
    $_inject(public) [&, name="serialize"](auto& ar, const unsigned int ver) -> void
    {
      $_constexpr for (auto& v : $BoostSerializable.variables())
        $_inject(_) ar & $_v(v.name());
    };
  }
};
```

```
class TestStruct {
public:
  template <typename T0> void serialize(T0 &ar, const unsigned int ver) {
    ar & a;
    ar & b;
  }
  int a;
  std::string b;
};
```

Метаклассы - использование утилиты

```
$_metaclass(BoostSerializableSplitted) {
  static void GenerateDecl()  {
    $_inject(public) [&, name="load"](auto& ar, unsigned int ver) -> void
    {
      $_constexpr for (auto& v : $BoostSerializableSplitted.variables())
        $_inject(_) ar >> $_v(v.name());
    };

    $_inject(public) [&, name="save", is_const=true](auto& ar, unsigned int ver) -> void
    {
      $_constexpr for (auto& v : $BoostSerializableSplitted.variables())
        $_inject(_) ar << $_v(v.name());
    };

    $_inject(public) [&, name="serialize"](auto& ar, const unsigned int ver) -> void
    {
      boost::serialization::split_member(ar, $_safe(*this), ver);
    };
  }
};
```

Метаклассы - использование утилиты

```
class TestStruct {
public:
  template <typename T0> void load(T0 &ar, unsigned int ver) {
     ar << a;
     ar << b;
  }
  template <typename T0> void save(T0 &ar, unsigned int ver) const {
     ar >> a;
     ar >> b;
  }
  template <typename T0> void serialize(T0 &ar, const unsigned int ver) {
     boost::serialization::split_member(ar, *this, ver);
  }
  int a;
  std::string b;

protected:
private:
};
```

Метаклассы - использование утилиты

$_metaclass(Name) - макрос, с помощью которого объявляется метакласс
$_class(Name, MetaclassName) - макрос, с помощью которого объявляется инстанс метакласса
$_inject(Visibility) - макрос-префикс, который приводит к вставке следующего за ним блока в тело инстанса метакласса
$_constexpr() - макрос-префикс, помечающий следующий за ним оператор как вычисляемый на этапе компиляции
$_inject_xxx(Name, Visibility), где XXX - это **enum**, **struct**, **class** или **union** - макрос-префикс, который приводит к вставке следующего за ним блока в тело инстанса метакласса в виде отдельной структуры, enum'а, класса или объединения
$_n(expr) - макрос, который заменяется значением выражения expr, вычисленного на этапе компиляции
$_t(expr) - макрос, который заменяется типом выражения expr, вычисленного на этапе компиляции
$_safe(expr) - макрос, который заменяется на expr

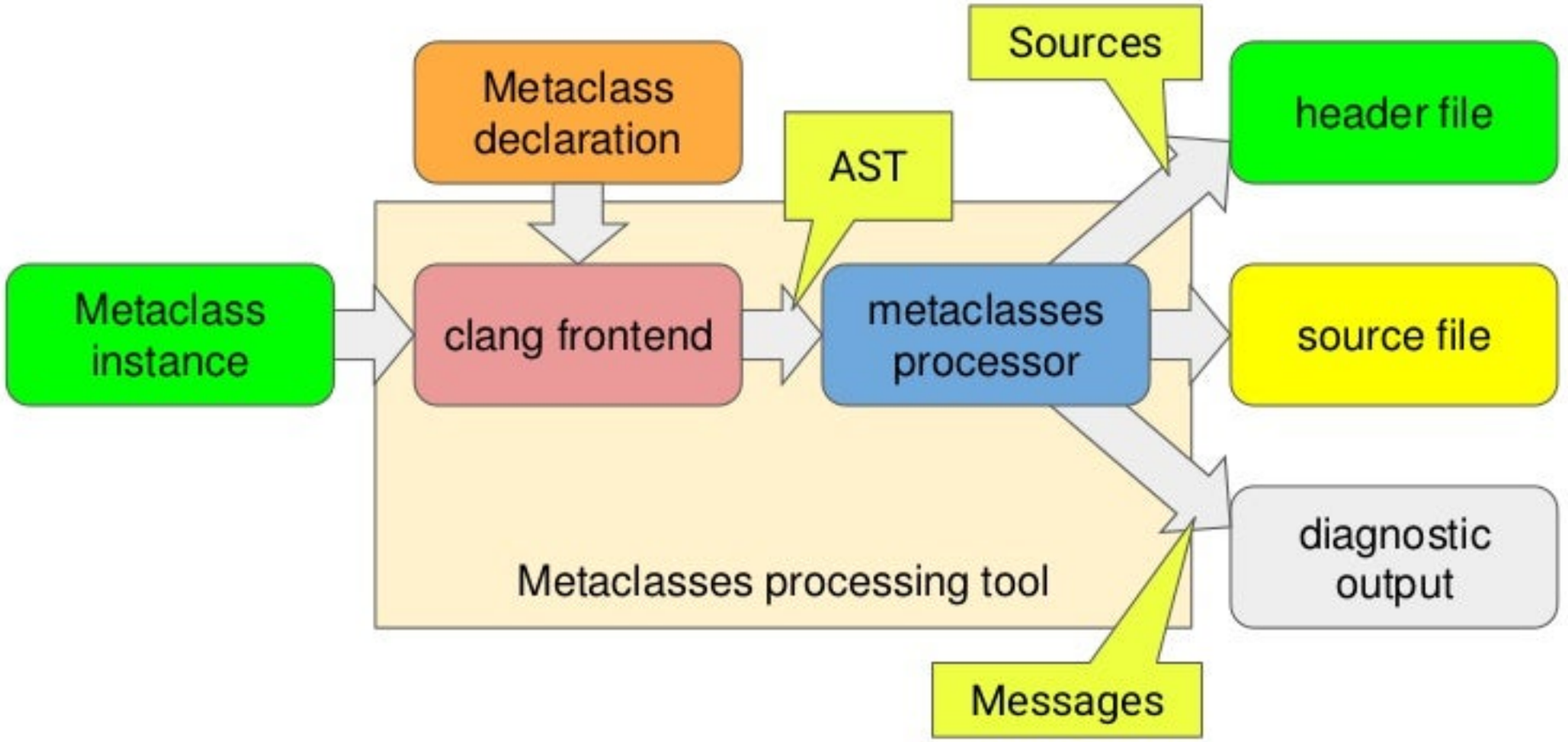
Метаклассы - использование утилиты

compiler - экземпляр класса **meta::CompilerImpl**, предоставляющий доступ к глобальным методам типа *require*, *error*, *message* и т. п.
meta::TypeInfo - класс, предоставляющий различную информацию о конкретном типе
meta::MemberInfo - класс, предоставляющий различную информацию о конкретном члене класса
meta::MethodInfo - класс, предоставляющий различную информацию о конкретном методе класса
meta::ClassInfo - класс, предоставляющий различную информацию о конкретном классе. Видимые в методах реализации метакласса переменные вида *$Interface* - именно этого типа

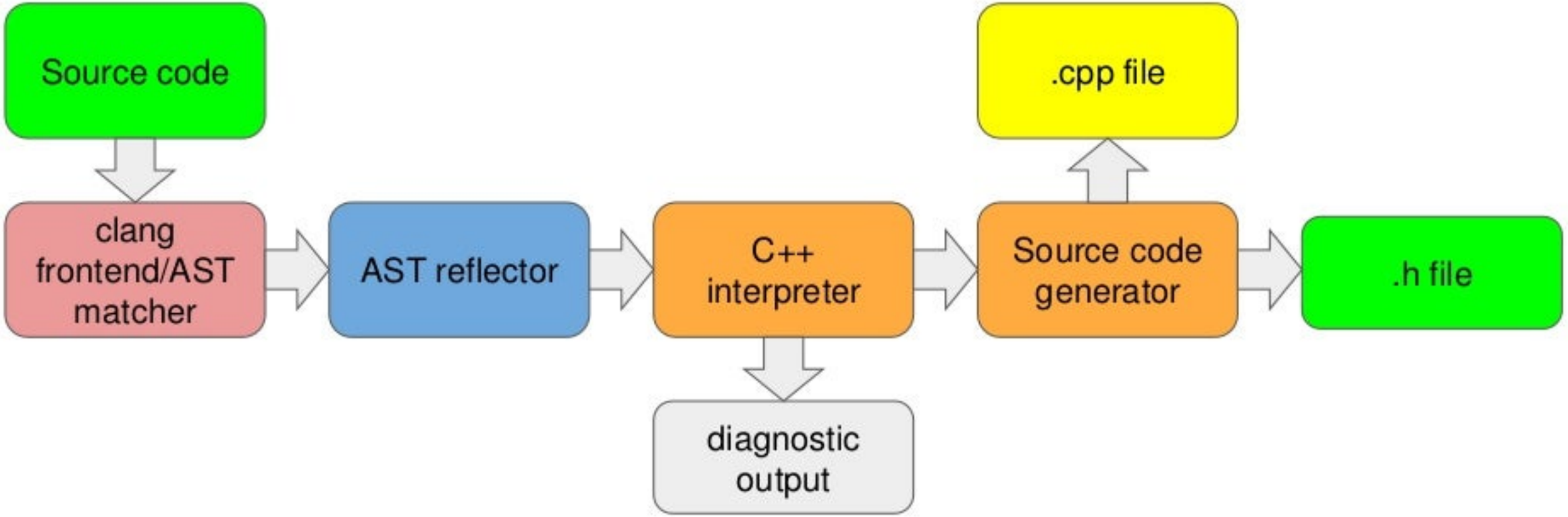
Метаклассы - использование утилиты

Перечисленные типы (и переменные), по сути - read/write-интерфейс к AST и компилятору

Metaclasses processing tool



Metaclasses processing tool



Интерпретатор C++

1. Детально анализирует AST декларации метакласса
2. Фактически исполняет код декларации метакласса используя в качестве исходных данных отрефлексированный AST конкретного инстанса
3. Обеспечивает печать диагностических сообщений и проверку предикатов
4. Модифицирует отрефлексированный AST в соответствии с прописанной в меткалассе логикой
5. Инжектирует в инстанс метакласса заданные фрагменты кода с учётом constexpr-конструкций

Интерпретатор C++

Интерпретатор C++

```
compiler.require($Interface.variables().empty(), "Interface may not contain data members");
```

Интерпретатор C++

```
compiler.require($Interface.variables().empty(), "Interface may not contain data members");
```

1. Интерпретатор операторов определяет, что это выражение

Интерпретатор C++

```
compiler.require($Interface.variables().empty(), "Interface may not contain data members");
```

1. Интерпретатор операторов определяет, что это выражение
2. Вычислитель определяет, что выражение - это вызов члена класса

Интерпретатор C++

```
compiler.require($Interface.variables().empty(), "Interface may not contain data members");
```

1. Интерпретатор операторов определяет, что это выражение
2. Вычислитель определяет, что выражение - это вызов члена класса
3. Объект, к которому применяется вызов метода, отображается на реализацию интерфейса к компилятору

Интерпретатор C++

```
compiler.require($Interface.variables().empty(), "Interface may not contain data members");
```

1. Интерпретатор операторов определяет, что это выражение
2. Вычислитель определяет, что выражение - это вызов члена класса
3. Объект, к которому применяется вызов метода, отображается на реализацию интерфейса к компилятору
4. Начинают вычисляться параметры

Интерпретатор C++

```
compiler.require($Interface.variables().empty(), "Interface may not contain data members");
```

1. Интерпретатор операторов определяет, что это выражение
2. Вычислитель определяет, что выражение - это вызов члена класса
3. Объект, к которому применяется вызов метода, отображается на реализацию интерфейса к компилятору
4. Начинают вычисляться параметры
5. Первый параметр - вызов члена класса

Интерпретатор C++

```
compiler.require($Interface.variables().empty(), "Interface may not contain data members");
```

1. Интерпретатор операторов определяет, что это выражение
2. Вычислитель определяет, что выражение - это вызов члена класса
3. Объект, к которому применяется вызов метода, отображается на реализацию интерфейса к компилятору
4. Начинают вычисляться параметры
5. Первый параметр - вызов члена класса
6. ‘$Interface’ отображается на дескриптор инстанса метакласса

Интерпретатор C++

```
compiler.require($Interface.variables().empty(), "Interface may not contain data members");
```

1. Интерпретатор операторов определяет, что это выражение
2. Вычислитель определяет, что выражение - это вызов члена класса
3. Объект, к которому применяется вызов метода, отображается на реализацию интерфейса к компилятору
4. Начинают вычисляться параметры
5. Первый параметр - вызов члена класса
6. ‘$Interface’ отображается на дескриптор инстанса метакласса
7. Последовательно вызываются: метод, возвращающий коллекцию переменных, а затем - ‘empty()’

Интерпретатор C++

```
compiler.require($Interface.variables().empty(), "Interface may not contain data members");
```

1. Интерпретатор операторов определяет, что это выражение
2. Вычислитель определяет, что выражение - это вызов члена класса
3. Объект, к которому применяется вызов метода, отображается на реализацию интерфейса к компилятору
4. Начинают вычисляться параметры
5. Первый параметр - вызов члена класса
6. ‘$Interface’ отображается на дескриптор инстанса метакласса
7. Последовательно вызываются: метод, возвращающий коллекцию переменных, а затем - ‘empty()’
8. Второй параметр, строковый литерал, конвертируется в строку

Интерпретатор C++

```
compiler.require($Interface.variables().empty(), "Interface may not contain data members");
```

1. Интерпретатор операторов определяет, что это выражение
2. Вычислитель определяет, что выражение - это вызов члена класса
3. Объект, к которому применяется вызов метода, отображается на реализацию интерфейса к компилятору
4. Начинают вычисляться параметры
5. Первый параметр - вызов члена класса
6. ‘$Interface’ отображается на дескриптор инстанса метакласса
7. Последовательно вызываются: метод, возвращающий коллекцию переменных, а затем - ‘empty()’
8. Второй параметр, строковый литерал, конвертируется в строку
9. Вызывается метод ‘require’, который в зависимости от значения первого параметра либо выводит диагностическое сообщение, либо нет

Интерпретатор C++

Нюансы реализации:

- ОЧЕНЬ СЛОЖНОЕ И ПОДРОБНОЕ AST
- Необходимость следить за областями видимости и, соответственно, временем жизни объектов и промежуточных результатов вычислений
- Логика вычисления некоторых конструкций нетривиальна. Например, range-based for loop
- Необходимость отделять то, что нужно интерпретировать, от того, что нужно инжектировать в результирующий код

Интерпретатор C++

- Связка boost::variant + boost::apply_visitor + C++14 + SFINAE - очень мощная. Позволяет упростить ряд конструкций.
- Атрибуты, введённые в язык, очень сильно помогают в разметке узлов AST и их последующем анализе
- Нет видимой необходимости поддерживать весь набор базовых типов в C++

Интеграция с системой сборки

```
include_directories(
  ${CODEGEN_DIR}
  ${CODEGEN_ROOT_DIR}/include
  )

set (CODEGEN_DIR ${CMAKE_CURRENT_BINARY_DIR}/codegen)
file (MAKE_DIRECTORY ${CODEGEN_DIR}/generated)
set (METACLASSES_GEN_FILE ${CODEGEN_DIR}/generated/structs.meta.h)
set (CODEGEN_ROOT_DIR "d:/projects/work/Personal/autoprogrammer/autoprogrammer")

set (CODEGEN_TOOL "d:/projects/work/Personal/autoprogrammer/autoprogrammer-build/MSVC2015-Debug/fl-codegen.exe")

add_custom_command(OUTPUT ${METACLASSES_GEN_FILE}
  COMMAND ${CODEGEN_TOOL} ARGS -gen-metaclasses -show-clang-diag -ohdr ${METACLASSES_GEN_FILE} -input
${CMAKE_CURRENT_SOURCE_DIR}/structs.h -- clang-cl -std=c++14 -x c++ ${CMAKE_CXX_FLAGS} -I ${CODEGEN_ROOT_DIR}/include -I
${BOOST_ROOT}
  MAIN_DEPENDENCY ${CMAKE_CURRENT_SOURCE_DIR}/structs.h
  DEPENDS ${CODEGEN_BIN_NAME} serialization_metaclass.h
  COMMENT "Generating metaclasses implementation for ${CMAKE_CURRENT_SOURCE_DIR}/structs.h"
  )

add_executable(${PROJECT_NAME} "main.cpp" "serialization_metaclass.h" "structs.h" ${METACLASSES_GEN_FILE})
target_link_libraries(${PROJECT_NAME} Boost::serialization)
```

Интеграция с системой сборки

```
include_directories(
  ${CODEGEN_DIR}
  ${CODEGEN_ROOT_DIR}/include
  )

set (CODEGEN_DIR ${CMAKE_CURRENT_BINARY_DIR}/codegen)
file (MAKE_DIRECTORY ${CODEGEN_DIR}/generated)
set (METACLASSES_GEN_FILE ${CODEGEN_DIR}/generated/structs.meta.h)
set (CODEGEN_ROOT_DIR "d:/projects/work/Personal/autoprogrammer/autoprogrammer")

set (CODEGEN_TOOL "d:/projects/work/Personal/autoprogrammer/autoprogrammer-build/MSVC2015-Debug/fl-codegen.exe")

add_custom_command(OUTPUT ${METACLASSES_GEN_FILE}
  COMMAND ${CODEGEN_TOOL} ARGS -gen-metaclasses -ohdr ${METACLASSES_GEN_FILE} -input
${CMAKE_CURRENT_SOURCE_DIR}/structs.h -- clang-cl -std=c++14 -x c++ ${CMAKE_CXX_FLAGS} -I ${CODEGEN_ROOT_DIR}/include -I
${BOOST_ROOT}
  MAIN_DEPENDENCY ${CMAKE_CURRENT_SOURCE_DIR}/structs.h
  DEPENDS ${CODEGEN_BIN_NAME} serialization_metaclass.h
  COMMENT "Generating metaclasses implementation for ${CMAKE_CURRENT_SOURCE_DIR}/structs.h"
  )

add_executable(${PROJECT_NAME} "main.cpp" "serialization_metaclass.h" "structs.h" ${METACLASSES_GEN_FILE})
target_link_libraries(${PROJECT_NAME} Boost::serialization)
```

Дальнейшие планы

- Доработка основных возможностей интерпретатора
- Отладка и настройка сборки под разные системы и компиляторы
- Докер-образы с уже установленной утилитой
- Расширение возможностей frontend'а:
  - манипуляция с типами
  - возможность генерации не только в .h-файлы, но и в .cpp
  - возможность задания нескольких метаклассов для одного инстанса
  - приведение в соответствие с последней (r4) ревизии p0707

Полезные ссылки

- Пропозал по метаклассам: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p0707r3.pdf
- Пропозал по static reflection (один из):http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/p0712r0.pdf
- Доклад Герба Саттера с ACCU 2017: https://www.youtube.com/watch?v=6nsyX37nsRs
- Доклад Герба Саттера с CppCon 2018: https://www.youtube.com/watch?v=80BZxujhY38
- Реализация metaclasses processing tool: https://github.com/flexferrum/autoprogrammer/tree/metaclasses
- Презентация по “автопрограммисту”: https://docs.google.com/presentation/d/1v0lJqkQVrDUbaKy7_79Vg67EcrNKpBB9yruZIvo8f0c/edit?usp=sharing

Спасибо!

Вопросы? :)

Сергей Садовников flexferrum@gmail.com