Integrating generators EDSL's for Spirit X3 (WIP)

Felipe Magno de Almeida

Expertise Solutions
Who are we?

- Embedded Linux Development
- Provide training and consulting on various programming subjects
  - C, C++, Generic Programming, Template Meta Programming
  - Tizen, Enlightenment Foundation Libraries

Expertise Solutions
Who are we?

- Embedded Linux Development
- Provide training and consulting on various programming subjects
  - C, C++, Generic Programming, Template Meta Programming
  - Tizen, Enlightenment Foundation Libraries

EXPERTISE. SOLUTIONS

Overview

- Experiment with Spirit V2 (mORBid)
- CORBA EDSL for Spirit v2
- Motivation to work on Spirit X3
- Writing terminals and parsers for Spirit X3
- Generators for Spirit X3
- Extending Spirit X3 to other domains

EXPERTISE. SOLUTIONS

Overview

- Experiment with Spirit V2 (mORBid)
- CORBA EDSL for Spirit v2
- Motivation to work on Spirit X3
- Writing terminals and parsers for Spirit X3
- Generators for Spirit X3
- Extending Spirit X3 to other domains

EXPERTISE. SOLUTIONS

Overview

- Experiment with Spirit V2 (mORBid)
- CORBA EDSL for Spirit v2
- Motivation to work on Spirit X3
- Writing terminals and parsers for Spirit X3
- Generators for Spirit X3
- Extending Spirit X3 to other domains

EXPERTISE. SOLUTIONS

Overview

- Experiment with Spirit V2 (mORBid)
- CORBA EDSL for Spirit v2
- Motivation to work on Spirit X3
- Writing terminals and parsers for Spirit X3
- Generators for Spirit X3
- Extending Spirit X3 to other domains

EXPERTISE. SOLUTIONS

Overview

- Experiment with Spirit V2 (mORBid)
- CORBA EDSL for Spirit v2
- Motivation to work on Spirit X3
- Writing terminals and parsers for Spirit X3
- Generators for Spirit X3
- Extending Spirit X3 to other domains

EXPERTISE. SOLUTIONS

Overview

- Experiment with Spirit V2 (mORBid)
- CORBA EDSL for Spirit v2
- Motivation to work on Spirit X3
- Writing terminals and parsers for Spirit X3
- Generators for Spirit X3
- Extending Spirit X3 to other domains

Experiment with Spirit v2

- mORBid was an experiment to bring CORBA to modern C++
- It uses Spirit v2 for all generation and parsing.
  - Parsing IDL and binary communication
  - Generating C++ from IDL compiler and binary communication
- Boost.Type_Erasure for object references
- Boost.ASIO for Network
- An EDSL for binary communication in Spirit v2

EXPERTISE. SOLUTIONS

Experiment with Spirit v2

- mORBid was an experiment to bring CORBA to modern C++
- It uses Spirit v2 for all generation and parsing.
  - Parsing IDL and binary communication
  - Generating C++ from IDL compiler and binary communication
- Boost.Type_Erasure for object references
- Boost.ASIO for Network
- An EDSL for binary communication in Spirit v2

EXPERTISE. SOLUTIONS

Experiment with Spirit v2

- mORBid was an experiment to bring CORBA to modern C++
- It uses Spirit v2 for all generation and parsing.
  - Parsing IDL and binary communication
  - Generating C++ from IDL compiler and binary communication
- Boost.Type_Erasure for object references
- Boost.ASIO for Network
- An EDSL for binary communication in Spirit v2

EXPERTISE. SOLUTIONS

Experiment with Spirit v2

- mORBid was an experiment to bring CORBA to modern C++
- It uses Spirit v2 for all generation and parsing.
  - Parsing IDL and binary communication
  - Generating C++ from IDL compiler and binary communication
- Boost.Type_Erasure for object references
- Boost.ASIO for Network
- An EDSL for binary communication in Spirit v2

EXPERTISE. SOLUTIONS

Experiment with Spirit v2

- mORBid was an experiment to bring CORBA to modern C++
- It uses Spirit v2 for all generation and parsing.
  - Parsing IDL and binary communication
  - Generating C++ from IDL compiler and binary communication
- Boost.Type_Erasure for object references
- Boost.ASIO for Network
- An EDSL for binary communication in Spirit v2

EXPERTISE. SOLUTIONS

Experiment with Spirit v2

- mORBid was an experiment to bring CORBA to modern C++
- It uses Spirit v2 for all generation and parsing.
  - Parsing IDL and binary communication
  - Generating C++ from IDL compiler and binary communication
- Boost.Type_Erasure for object references
- Boost.ASIO for Network
- An EDSL for binary communication in Spirit v2

CORBA EDSL with Spirit v2

- Spirit v2 uses Boost.Proto
  - The natural way to use Boost.Proto is doing Proto transformations
  - Spirit v2 has a meta_compiler for transformation and grammar matching
    - Compile time Switch/case over syntax
    - Calls meta functions based on syntax. E.g, make_terminal, make_binary, make_directive
    - meta_compiler and meta functions are templated on a domain tag

CORBA EDSL with Spirit v2

- Spirit v2 uses Boost.Proto
  - The natural way to use Boost.Proto is doing Proto transformations
  - Spirit v2 has a meta_compiler for transformation and grammar matching
    - Compile time Switch/case over syntax
    - Calls meta functions based on syntax. E.g, make_terminal, make_binary, make_directive
    - meta_compiler and meta functions are templated on a domain tag

EXPERTISE. SOLUTIONS

CORBA EDSL with Spirit v2

- Spirit v2 uses Boost.Proto
  - The natural way to use Boost.Proto is doing Proto transformations
  - Spirit v2 has a meta_compiler for transformation and grammar matching
    - Compile time Switch/case over syntax
    - Calls meta functions based on syntax. E.g, make_terminal, make_binary, make_directive
    - meta_compiler and meta functions are templated on a domain tag

EXPERTISE. SOLUTIONS

CORBA EDSL with Spirit v2

- Spirit v2 uses Boost.Proto
  - The natural way to use Boost.Proto is doing Proto transformations
  - Spirit v2 has a meta_compiler for transformation and grammar matching
    - Compile time Switch/case over syntax
    - Calls meta functions based on syntax. E.g, make_terminal, make_binary, make_directive
    - meta_compiler and meta functions are templated on a domain tag

EXPERTISE. SOLUTIONS

CORBA EDSL with Spirit v2

- Spirit v2 uses Boost.Proto
  - The natural way to use Boost.Proto is doing Proto transformations
  - Spirit v2 has a meta_compiler for transformation and grammar matching
    - Compile time Switch/case over syntax
    - Calls meta functions based on syntax. E.g, make_terminal, make_binary, make_directive
    - meta_compiler and meta functions are templated on a domain tag

EXPERTISE. SOLUTIONS

CORBA EDSL with Spirit v2

- Spirit v2 uses Boost.Proto
  - The natural way to use Boost.Proto is doing Proto transformations
  - Spirit v2 has a meta_compiler for transformation and grammar matching
    - Compile time Switch/case over syntax
    - Calls meta functions based on syntax. E.g, make_terminal, make_binary, make_directive
    - meta_compiler and meta functions are templated on a domain tag

EXPERTISE. SOLUTIONS

CORBA EDSL with Spirit v2

- Specialized the make meta-functions
- The meta-compilation creates a parser or generator
- Giving new meanings to terminals
- Composability

EXPERTISE. SOLUTIONS

CORBA EDSL with Spirit v2

- Specialized the make meta-functions
- The meta-compilation creates a parser or generator
- Giving new meanings to terminals
- Composability

```
bool r = parse(first, last,
    giop::compile<iiop::parser_domain>
    (giop::endianness
     ["GIOP"
      & octet
      & octet
      & string
      & giop::ushort_
      & sequence[octet]
     ])
     , attr
    );
```

EXPERTISE. SOLUTIONS

CORBA EDSL with Spirit v2

- Specialized the make meta-functions
- The meta-compilation creates a parser or generator
- Giving new meanings to terminals
- Composability

EXPERTISE. SOLUTIONS

CORBA EDSL with Spirit v2

- Specialized the make meta-functions
- The meta-compilation creates a parser or generator
- Giving new meanings to terminals
- Composability

Terminals

```
bool r = parse(first, last,
    giop::compile<iiop::parser_domain>
    (giop::endianness
     ["GIOP" // new meaning
      & octet
      & octet
      & string
      & giop::ushort_
      & sequence[octet]
     ])
      , attr
    );
```

EXPERTISE. SOLUTIONS

CORBA EDSL with Spirit v2

- Specialized the make meta-functions
- The meta-compilation creates a parser or generator
- Giving new meanings to terminals
- Composability

CORBA EDSL with Spirit v2

- Specialized the make meta-functions
- The meta-compilation creates a parser or generator
- Giving new meanings to terminals
- Composability

Composability

```
bool r = parse(first, last,
    giop::compile<iiop::parser_domain>
    (giop::endianness
     ["GIOP"
      & octet
      & octet
      & string
      & giop::ushort_
      & sequence[octet]
     ])
     % giop::compile<iiop::parser_domain>
     (octet('\0'))
      , attr
    );
```

Motivation

- Spirit X3 is the future
- Make format specifications easy to write on X3
- Write Spirit Karma for Spirit X3

EXPERTISE. SOLUTIONS

Motivation

- Spirit X3 is the future
- Make format specifications easy to write on X3
- Write Spirit Karma for Spirit X3

EXPERTISE. SOLUTIONS

Motivation

- Spirit X3 is the future
- Make format specifications easy to write on X3
- Write Spirit Karma for Spirit X3

Spirit X3

- Spirit X3
  - How is it better than Spirit v2?
  - Why is it faster (compile-time)?
    - It doesn't use Boost.Proto and has a much leaner ET
  - Extending Spirit X3
    - Writing terminals and operators for new parsers
    - Writing generators

EXPERTISE. SOLUTIONS

Spirit X3

- Spirit X3
  - How is it better than Spirit v2?
  - Why is it faster (compile-time)?
    - It doesn't use Boost.Proto and has a much leaner ET
  - Extending Spirit X3
    - Writing terminals and operators for new parsers
    - Writing generators

Spirit X3

- Spirit X3
  - How is it better than Spirit v2?
  - Why is it faster (compile-time)?
    - It doesn't use Boost.Proto and has a much leaner ET
  - Extending Spirit X3
    - Writing terminals and operators for new parsers
    - Writing generators

EXPERTISE. SOLUTIONS

Spirit X3

- Spirit X3
  - How is it better than Spirit v2?
  - Why is it faster (compile-time)?
    - It doesn't use Boost.Proto and has a much leaner ET
  - Extending Spirit X3
    - Writing terminals and operators for new parsers
    - Writing generators

EXPERTISE. SOLUTIONS

Spirit X3

- Spirit X3
  - How is it better than Spirit v2?
  - Why is it faster (compile-time)?
    - It doesn't use Boost.Proto and has a much leaner ET
  - Extending Spirit X3
    - Writing terminals and operators for new parsers
    - Writing generators

EXPERTISE. SOLUTIONS

Spirit X3

- Spirit X3
  - How is it better than Spirit v2?
  - Why is it faster (compile-time)?
    - It doesn't use Boost.Proto and has a much leaner ET
  - Extending Spirit X3
    - Writing terminals and operators for new parsers
    - Writing generators

EXPERTISE. SOLUTIONS

Writing terminals in Spirit X3

```
struct eol_parser : parser<eol_parser>
{
 typedef unused_type attribute_type;
 static bool const has_attribute = false;

 template <typename I, typename C, typename A>
 bool parse(I& first, I last, C const& context
   , unused_type, Attribute& /*attr*/) const;
};

auto const eol = eol_parser{};
```

EXPERTISE. SOLUTIONS

Writing operators in Spirit X3

```
template <typename L, typename R>
inline sequence<
 typename as_parser<L>::value_type
 , typename as_parser<R>::value_type>
operator>>(L const& lhs, R const& rhs)
{
 return { as_parser(lhs), as_parser(rhs) };
}
```

Parsers

- Parser concept
  - SFINAE operators
  - To convert constants and terminals into parsers. e.g., char_ >> '5'
  - as_parser converts int to a parser before passing to sequence<> parser.
  - Customization points

EXPERTISE. SOLUTIONS

Parsers

- Parser concept
  - SFINAE operators
  - To convert constants and terminals into parsers. e.g., char_ >> '5'
  - as_parser converts int to a parser before passing to sequence<> parser.
  - Customization points

EXPERTISE. SOLUTIONS

Parsers

- Parser concept
  - SFINAE operators
  - To convert constants and terminals into parsers. e.g., char_ >> '5'
  - as_parser converts int to a parser before passing to sequence<> parser.
  - Customization points

EXPERTISE. SOLUTIONS

Parsers

- Parser concept
  - SFINAE operators
  - To convert constants and terminals into parsers. e.g., char_ >> '5'
  - as_parser converts int to a parser before passing to sequence<> parser.
  - Customization points

EXPERTISE. SOLUTIONS

Parsers

- Parser concept
  - SFINAE operators
  - To convert constants and terminals into parsers. e.g., char_ >> '5'
  - as_parser converts int to a parser before passing to sequence<> parser.
  - Customization points

EXPERTISE. SOLUTIONS

Creating parsers for Spirit x3

- Customization points
  - specialize x3 :: extension :: as_parser.
    template <> class as_parser<int>
    - To allow 'c'har constants
  - Overload the ADL-enabled as_spirit_parser function
  - Inherit from x3 :: parser_base.
    - ADL-enables operators from Spirit X3
  - http://talesofcpp.fusionfenix.com/post-8/true-story-i-will-always-find-you

EXPERTISE. SOLUTIONS

Creating parsers for Spirit x3

- Customization points
  - specialize x3 :: extension :: as_parser.
    template <> class as_parser<int>
    - To allow 'c'har constants
  - Overload the ADL-enabled as_spirit_parser function
  - Inherit from x3 :: parser_base.
    - ADL-enables operators from Spirit X3
  - http://talesofcpp.fusionfenix.com/post-8/true-story-i-will-always-find-you

EXPERTISE. SOLUTIONS

Creating parsers for Spirit x3

- Customization points
  - specialize x3 :: extension :: as_parser.
    template <> class as_parser<int>
    - To allow **'c'har** constants
  - Overload the ADL-enabled as_spirit_parser function
  - Inherit from x3 :: parser_base.
    - ADL-enables operators from Spirit X3
  - http://talesofcpp.fusionfenix.com/post-8/true-story-i-will-always-find-you

EXPERTISE. SOLUTIONS

Creating parsers for Spirit x3

- Customization points
  - specialize x3 :: extension :: as_parser.
    template <> class as_parser<int>
    - To allow **'c'har** constants
  - Overload the ADL-enabled as_spirit_parser function
  - Inherit from x3 :: parser_base.
    - ADL-enables operators from Spirit X3
  - http://talesofcpp.fusionfenix.com/post-8/true-story-i-will-always-find-you

EXPERTISE. SOLUTIONS

Overload as_spirit_parser

```
namespace foo {

class point { ··· };
class point_parser { ··· };

point_parser as_spirit_parser(point const&);

}
```

EXPERTISE. SOLUTIONS

Creating parsers for Spirit x3

- Customization points
  - specialize x3 :: extension :: as_parser.
    template <> class as_parser<int>
    - To allow **'c'har** constants
  - Overload the ADL-enabled as_spirit_parser function
  - Inherit from x3 :: parser_base.
    - ADL-enables operators from Spirit X3
  - http://talesofcpp.fusionfenix.com/post-8/true-story-i-will-always-find-you

EXPERTISE. SOLUTIONS

Creating parsers for Spirit x3

- Customization points
  - specialize x3 :: extension :: as_parser.
    template <> class as_parser<int>
    - To allow **'c'har** constants
  - Overload the ADL-enabled as_spirit_parser function
  - Inherit from x3 :: parser_base.
    - ADL-enables operators from Spirit X3
  - http://talesofcpp.fusionfenix.com/post-8/true-story-i-will-always-find-you

Creating parsers for Spirit x3

- Customization points
  - specialize x3 :: extension :: as_parser.
    template <> class as_parser<int>
    - To allow **'c'har** constants
  - Overload the ADL-enabled as_spirit_parser function
  - Inherit from x3 :: parser_base.
    - ADL-enables operators from Spirit X3
  - http://talesofcpp.fusionfenix.com/post-8/true-story-i-will-always-find-you

EXPERTISE. SOLUTIONS

Creating parsers for Spirit x3

- Customization points
  - specialize x3 :: extension :: as_parser.
    template <> class as_parser<int>
    - To allow **'c'har** constants
  - Overload the ADL-enabled as_spirit_parser function
  - Inherit from x3 :: parser_base.
    - ADL-enables operators from Spirit X3
  - http://talesofcpp.fusionfenix.com/post-8/true-story-i-will-always-find-you

EXPERTISE. SOLUTIONS

Generators

- Generator concept
- Allowing same-syntax to build both generators and parsers
- Implementing generators in already developed parsers
  - Creates classes that are parsers and generators at the same time
  - Less typing
- Using Spirit.Karma test cases as basis for development

EXPERTISE. SOLUTIONS

Generators

- Generator concept
- Allowing same-syntax to build both generators and parsers
- Implementing generators in already developed parsers
  - Creates classes that are parsers and generators at the same time
  - Less typing
- Using Spirit.Karma test cases as basis for development

EXPERTISE. SOLUTIONS

Same syntax

- Spirit v2 had symmetric operators ($>>$ and $<<$) for parsers and generators
- This symmetry causes unnecessary incompatibility
- This enables the user to write parsers and generators as a format specification

EXPERTISE. SOLUTIONS

Same syntax

- Spirit v2 had symmetric operators ($>>$ and $<<$) for parsers and generators
- This symmetry causes unnecessary incompatibility
- This enables the user to write parsers and generators as a format specification

EXPERTISE. SOLUTIONS

Same syntax

- Spirit v2 had symmetric operators ($>>$ and $<<$) for parsers and generators
- This symmetry causes unnecessary incompatibility (so I chose $>>$ for everything)
- This enables the user to write parsers and generators as a format specification

EXPERTISE. SOLUTIONS

Same syntax

- Spirit v2 had symmetric operators ($>>$ and $<<$) for parsers and generators
- This symmetry causes unnecessary incompatibility (so I chose $>>$ for everything)
- This enables the user to write parsers and generators as a format specification

Format specification

```
auto status_line = "HTTP/1.1 "
  >> int_ >> ' ' >> +char_ >> "\r\n";

generate(output_it, status_line
  , fusion::make_vector(200, "OK"));

fusion::vector2<int, std::string> res;
parse(first, last, status_line, res);
```

EXPERTISE. SOLUTIONS

Generators

- Generator concept
- Allowing same-syntax to build both generators and parsers
- Implementing generators in already developed parsers
  - Creates classes that are parsers and generators at the same time
  - Less typing
- Using Spirit.Karma test cases as basis for development

EXPERTISE. SOLUTIONS

Generators

- Generator concept
- Allowing same-syntax to build both generators and parsers
- Implementing generators in already developed parsers
  - Creates classes that are parsers and generators at the same time
  - Less typing
- Using Spirit.Karma test cases as basis for development

EXPERTISE. SOLUTIONS

Generators

- Generator concept
- Allowing same-syntax to build both generators and parsers
- Implementing generators in already developed parsers
  - Creates classes that are parsers and generators at the same time
  - Less typing
- Using Spirit.Karma test cases as basis for development

EXPERTISE. SOLUTIONS

Generators

- Generator concept
- Allowing same-syntax to build both generators and parsers
- Implementing generators in already developed parsers
  - Creates classes that are parsers and generators at the same time
  - Less typing
- Using Spirit.Karma test cases as basis for development

Parsers and generators

```
template <typename Derived>
struct char_parser : parser<Derived>
  , generator<Derived>
{
 template <typename I, typename C, typename A>
 bool parse(I& first, I last, C const& context
  , unused_type, A& attr) const;

 template <typename I, typename C, typename A>
 bool generate(I sink, C const& context
  , unused_type, A const& attr) const;

 ...
```

EXPERTISE. SOLUTIONS

Generators

- Generator concept
- Allowing same-syntax to build both generators and parsers
- Implementing generators in already developed parsers
  - Creates classes that are parsers and generators at the same time
  - Less typing
- Using Spirit.Karma test cases as basis for development

EXPERTISE. SOLUTIONS

Generators

- Generator concept
- Allowing same-syntax to build both generators and parsers
- Implementing generators in already developed parsers
  - Creates classes that are parsers and generators at the same time
  - Less typing
- Using Spirit.Karma test cases as basis for development

Test cases (WIP)

```
BOOST_TEST(test_gen("x"
  , char_('x') | char_('i')));
BOOST_TEST(test_gen("xi"
  , char_('x') >> char_('i') | char_('i')));
BOOST_TEST(test_gen("i"
  , char_('i') | char_('x') >> char_('i')));

boost::variant<int, char> v (10);
BOOST_TEST(test_gen("10", char_ | int_, v));
BOOST_TEST(test_gen("10", int_ | char_, v));
```

EXPERTISE. SOLUTIONS

Extending Spirit x3 to other domains

- Limitations of Spirit X3
- Domain specialization by new terminals and directives
- Using context for orthogonal aspects
- Using attribute matching for format specification
  - Semantic actions are not compatible with this approach
- Using ADL with Context for multiple backends
- Putting it all together

Limitations Spirit x3

- It has only one compilation phase
- The only phase is during the ET construction
- We can only extend terminals meanings, but not give another meaning

Limitations Spirit x3

- It has only one compilation phase
- The only phase is during the ET construction
- We can only extend terminals meanings, but not give another meaning

EXPERTISE. SOLUTIONS

Only phase

```
template <typename L, typename R>
inline sequence<
 typename as_parser<L>::value_type
 , typename as_parser<R>::value_type>
operator>>(L const& lhs, R const& rhs)
{
 return { as_parser(lhs), as_parser(rhs) };
}
```

Limitations Spirit x3

- It has only one compilation phase
- The only phase is during the ET construction
- We can only extend terminals meanings, but not give another meaning

Limitations Spirit x3

- It has only one compilation phase
- The only phase is during the ET construction
- We can only extend terminals meanings, but not give another meaning

EXPERTISE. SOLUTIONS

Extending Spirit x3 to other domains

- Limitations of Spirit X3
- Domain specialization by new terminals and directives
- Using context for orthogonal aspects
- Using attribute matching for format specification
  - Semantic actions are not compatible with this approach
- Using ADL with Context for multiple backends
- Putting it all together

Extending Spirit x3 to other domains

- Limitations of Spirit X3
- Domain specialization by new terminals and directives
- Using context for orthogonal aspects
- Using attribute matching for format specification
  - Semantic actions are not compatible with this approach
- Using ADL with Context for multiple backends
- Putting it all together

EXPERTISE. SOLUTIONS

Terminals and specialization

```
struct octet_partor
  : x3::parser<octet_partor>
  , x3::generator<octet_partor>
{
   typedef int attribute_type;
   static bool const has_attribute = true;

   template <typename I, typename C, typename A>
   bool parse(I& first, I last, C const& context
     , x3::unused_type, A& attr) const
   {
      return octet_parse(*this, first, last
        , context, x3::unused, attr);
   }
```

Terminals and specialization

```
    template <typename I, typename C, typename A>
    bool generate(O sink, C const& context
      , x3::unused_type, A& attr) const
    {
      return octet_generate(*this, sink, context
        , x3::unused, attr);
    }
};

octet_partor octet = {};
```

Extending Spirit x3 to other domains

- Limitations of Spirit X3
- Domain specialization by new terminals and directives
- Using context for orthogonal aspects
- Using attribute matching for format specification
  - Semantic actions are not compatible with this approach
- Using ADL with Context for multiple backends
- Putting it all together

EXPERTISE. SOLUTIONS

Extending Spirit x3 to other domains

- Limitations of Spirit X3
- Domain specialization by new terminals and directives
- Using context for orthogonal aspects
- Using attribute matching for format specification
  - Semantic actions are not compatible with this approach
- Using ADL with Context for multiple backends
- Putting it all together

Endianness and alignment in CORBA

- Endianness obeys scopes
- Alignment obeys some scopes
- Both are runtime information that changes aspects of the format layout

Endianness and alignment in CORBA

- Endianness obeys scopes
- Alignment obeys some scopes
- Both are runtime information that changes aspects of the format layout

Endianness and alignment in CORBA

- Endianness obeys scopes
- Alignment obeys some scopes
- Both are runtime information that changes aspects of the format layout

```
template <typename C, typename E, typename A>
inline bool generate(C& container
  , E const& expr, A& attr)
{
 typedef forward_back_insert_iterator<C>
  iterator;
 iterator sink(container);
 ...
 alignment_attribute<iterator>
  alignment_attr{sink, 0u};
 auto align_context =
  boost::spirit::x3::make_context
  <alignment_tag>(alignment_attr, x3::unused);
 return as_generator(expression).generate
   (sink, align_context, x3::unused, attr);
}
```

EXPERTISE. SOLUTIONS

Extending Spirit x3 to other domains

- Limitations of Spirit X3
- Domain specialization by new terminals and directives
- Using context for orthogonal aspects
- Using attribute matching for format specification
  - Semantic actions are not compatible with this approach
- Using ADL with Context for multiple backends
- Putting it all together

Extending Spirit x3 to other domains

- Limitations of Spirit X3
- Domain specialization by new terminals and directives
- Using context for orthogonal aspects
- Using attribute matching for format specification
  - Semantic actions are not compatible with this approach
- Using ADL with Context for multiple backends
- Putting it all together

EXPERTISE. SOLUTIONS

Attributes, parsers and generators

- If attributes have a inherent sequence when parsing and generating
- Compatible syntax
- Let's forget parsing and generators
- Format specification

EXPERTISE. SOLUTIONS

Attributes, parsers and generators

- If attributes have a inherent sequence when parsing and generating
- Compatible syntax
- Let's forget parsing and generators
- Format specification

EXPERTISE. SOLUTIONS

Attributes, parsers and generators

- If attributes have a inherent sequence when parsing and generating
- Compatible syntax
- Let's forget parsing and generators
- Format specification

Attributes, parsers and generators

- If attributes have a inherent sequence when parsing and generating
- Compatible syntax
- Let's forget parsing and generators
- Format specification

Extending Spirit x3 to other domains

- Limitations of Spirit X3
- Domain specialization by new terminals and directives
- Using context for orthogonal aspects
- Using attribute matching for format specification
  - Semantic actions are not compatible with this approach
- Using ADL with Context for multiple backends
- Putting it all together

EXPERTISE. SOLUTIONS

Extending Spirit x3 to other domains

- Limitations of Spirit X3
- Domain specialization by new terminals and directives
- Using context for orthogonal aspects
- Using attribute matching for format specification
  - Semantic actions are not compatible with this approach
- Using ADL with Context for multiple backends
- Putting it all together

EXPERTISE. SOLUTIONS

Extending Spirit x3 to other domains

- Limitations of Spirit X3
- Domain specialization by new terminals and directives
- Using context for orthogonal aspects
- Using attribute matching for format specification
  - Semantic actions are not compatible with this approach
- Using ADL with Context for multiple backends
- Putting it all together

ADL with Context

- GIOP is a generic specification for CORBA messages
- IIOP is an Internet/TCP specification (with endianness and alignment)
- Grammar can be written in generic GIOP
- Parsing and generation must be with specific protocol (IIOP or other)
- We can use overload and ADL

ADL with Context

- GIOP is a generic specification for CORBA messages
- IIOP is an Internet/TCP specification (with endianness and alignment)
- Grammar can be written in generic GIOP
- Parsing and generation must be with specific protocol (IIOP or other)
- We can use overload and ADL

ADL with Context

- GIOP is a generic specification for CORBA messages
- IIOP is an Internet/TCP specification (with endianness and alignment)
- Grammar can be written in generic GIOP
- Parsing and generation must be with specific protocol (IIOP or other)
- We can use overload and ADL

EXPERTISE. SOLUTIONS

ADL with Context

- GIOP is a generic specification for CORBA messages
- IIOP is an Internet/TCP specification (with endianness and alignment)
- Grammar can be written in generic GIOP
- Parsing and generation must be with specific protocol (IIOP or other)
- We can use overload and ADL

ADL with Context

- GIOP is a generic specification for CORBA messages
- IIOP is an Internet/TCP specification (with endianness and alignment)
- Grammar can be written in generic GIOP
- Parsing and generation must be with specific protocol (IIOP or other)
- We can use overload and ADL

ADL with Context

```
namespace giop {

struct octet_partor : x3::parser<octet_partor>
                    , x3::generator<octet_partor>
{
 typedef int attribute_type;
 static bool const has_attribute = true;

 template <typename I, typename C, typename A>
 bool parse(I& first, I const& last, C const&
  context, x3::unused_type, A& attr) const
 {
  return octet_parse(*this, first, last
    , context, x3::unused, attr);
 }
...
```

ADL with Context

```
namespace iiop {

template <typename I, typename C, typename A>
bool octet_parse(giop::octet_partor const&
  , I& first, I const& last
  , C const& context, x3::unused_type, A& attr)
{
  ...
```

ADL with Context

```
template <typename I, typename E, typename A>
inline bool
parse(I& first, I last, E const& expr, A& attr)
{
 I saved = first;
 auto iiop_context =
  boost::spirit::x3::make_context
  <iiop::iiop_parse_tag>(x3::unused);
 auto align_context
  = boost::spirit::x3::make_context
  <alignment_tag, I>(saved, iiop_context);
 return as_parser(expr).parse
  (first, last, align_context, x3::unused, attr);
}
```

EXPERTISE. SOLUTIONS

Extending Spirit x3 to other domains

- Limitations of Spirit X3
- Domain specialization by new terminals and directives
- Using context for orthogonal aspects
- Using attribute matching for format specification
  - Semantic actions are not compatible with this approach
- Using ADL with Context for multiple backends
- Putting it all together

EXPERTISE. SOLUTIONS

Extending Spirit x3 to other domains

- Limitations of Spirit X3
- Domain specialization by new terminals and directives
- Using context for orthogonal aspects
- Using attribute matching for format specification
  - Semantic actions are not compatible with this approach
- Using ADL with Context for multiple backends
- Putting it all together

EXPERTISE. SOLUTIONS

Putting it all together

- Let's make a call

```
object1->is_a
  ("IDL:omg.org/CORBA/Object:1.0");
```

EXPERTISE. SOLUTIONS

Putting it all together

```
template <typename Body>
auto message_1_0(Body const& body)
{
  return
    raw("GIOP")
    >> octet('\1') // major version
    >> octet('\0') // minor version
    >> endianness  // one byte
    [
      octet(MessageType)
      >> raw_size(ulong_)[body]
    ]
  ;
}
```

Putting it all together

```
template <std::size_t MessageType, typename B>
auto request_1_0(B const& body)
{
  return
     sequence[service_context]
    >> ulong_                  // request_id
    >> bool_                   // response_expected
    >> sequence[octet]         // object key
    >> string                  // operation
    >> sequence[octet]         // requesting_principal
    >> body
  ;
}
```

Putting it all together

```
auto isa_call = message_1_0<0u>
 (request_1_0(giop::string /* args */));
std::vector<std::vector<char>> empty;
std::vector<char> buffer;
iiop::generate(buffer, isa_call
 , fusion::make_vector
  (empty                          // service context
   , 0u                           // request id
   , true                         // expect response
   , "POARoot#object1"            // object key
   , "is_a"                       // operation name
   , ""                           // request principal
   , "IDL:omg.org/CORBA/Object:1.0");

send(socket, buffer);
```

EXPERTISE. SOLUTIONS

Putting it all together

```
auto isa_reply = message_1_0<1u>(reply_1_0
  (giop::bool_ /* return and out params */));
std::vector<std::vector<char>> empty;
std::vector<char> buffer;
iiop::generate(buffer, isa_reply
  , fusion::make_vector
   (empty                    // service context
    , 0u                     // request id
    , 0u                     // reply status
    , true);

send(socket, buffer);
```

Q&A

- Questions?
- Expertise Solutions - Felipe Magno de Almeida felipe@expertisesolutions.com.br
- https://github.com/expertisesolutions/spirit
- https://github.com/expertisesolutions/giop
- https://git.enlightenment.org/ for GUI in C++