The *rol* compiler

Robotics Language Tutorial - IEEE IRC 2019

Prerequisites

EBNF

parsers

XML

grammars

epath

xpath

python

Jinja2

templates

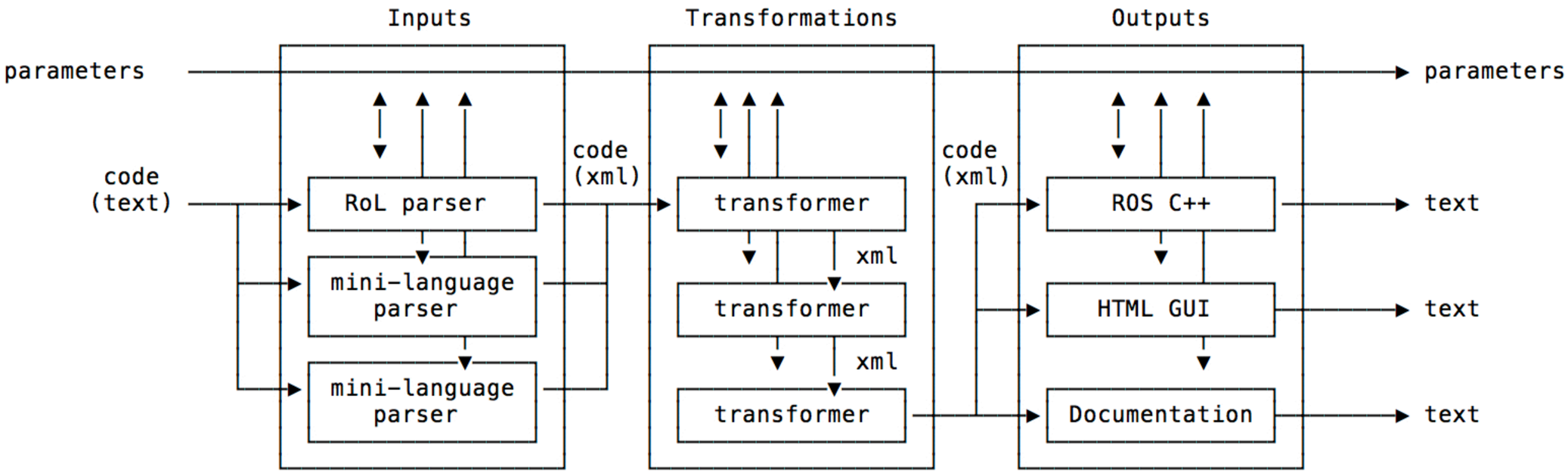
rol compiler

`rol` processes two types of information:

- **code**
- **parameters**

in three steps:

- **input**
- **transformations**
- **outputs**



rol compiler

Code: textual or abstract syntax tree representation of a program

Parameters: code-independent information that changes the behaviour of the compiler

rol compiler

Inputs:

Language parsers

Transformations:

Annotations on abstract syntax tree,
computations, decision making, file copying/creating

Outputs:

Serialisation, code generators

rol compiler

Parameters:

dictionary

Code:

text → **XML** → **text**

behaviour description

abstract syntax tree

programming language

Input phase

text code

```
node(
  name:"hello world",
  initialise:print("hello world!")
)
```

Input

Parsing

abstract syntax tree

```
<node p="64">
  <option p="26" name="name">
    <string p="26">hello world</string>
  </option>
  <option p="63" name="initialise">
    <print p="62">
      <string p="61">hello world!</string>
    </print>
  </option>
</node>
```

Input phase

Parser by file extension or by language

example.rol

```
node(name:'finite state machine',
     definitions: FiniteStateMachine<{
       name:machine
       initial:idle
       (idle) -start-> (running)
       -stop-> (idle)
       }>
    )
```

RoL parser

Finite state machine parser

example.fsm

```
name:machine
initial:idle
(idle) -start-> (running)
-stop-> (idle)
```

Input phase

Tools to automatically generate **grammar**

Type checking

```
'plus': {
        'definition': {
            'arguments': arguments('( real real+ | string string+)'),
            'returns': returns('same')
        },
        'input': {
            'RoL': {
                'infix': {'key': '+',
                          'order': 1100,
                          'flat': True}
            }
        },
```

Precedence order

Transformation phase

abstract syntax tree

```
<node p="64">
  <option p="26" name="name">
    <string p="26">hello world</string>
  </option>
  <option p="63" name="initialise">
    <print p="62">
      <string p="61">hello world!</string>
    </print>
  </option>
</node>
```

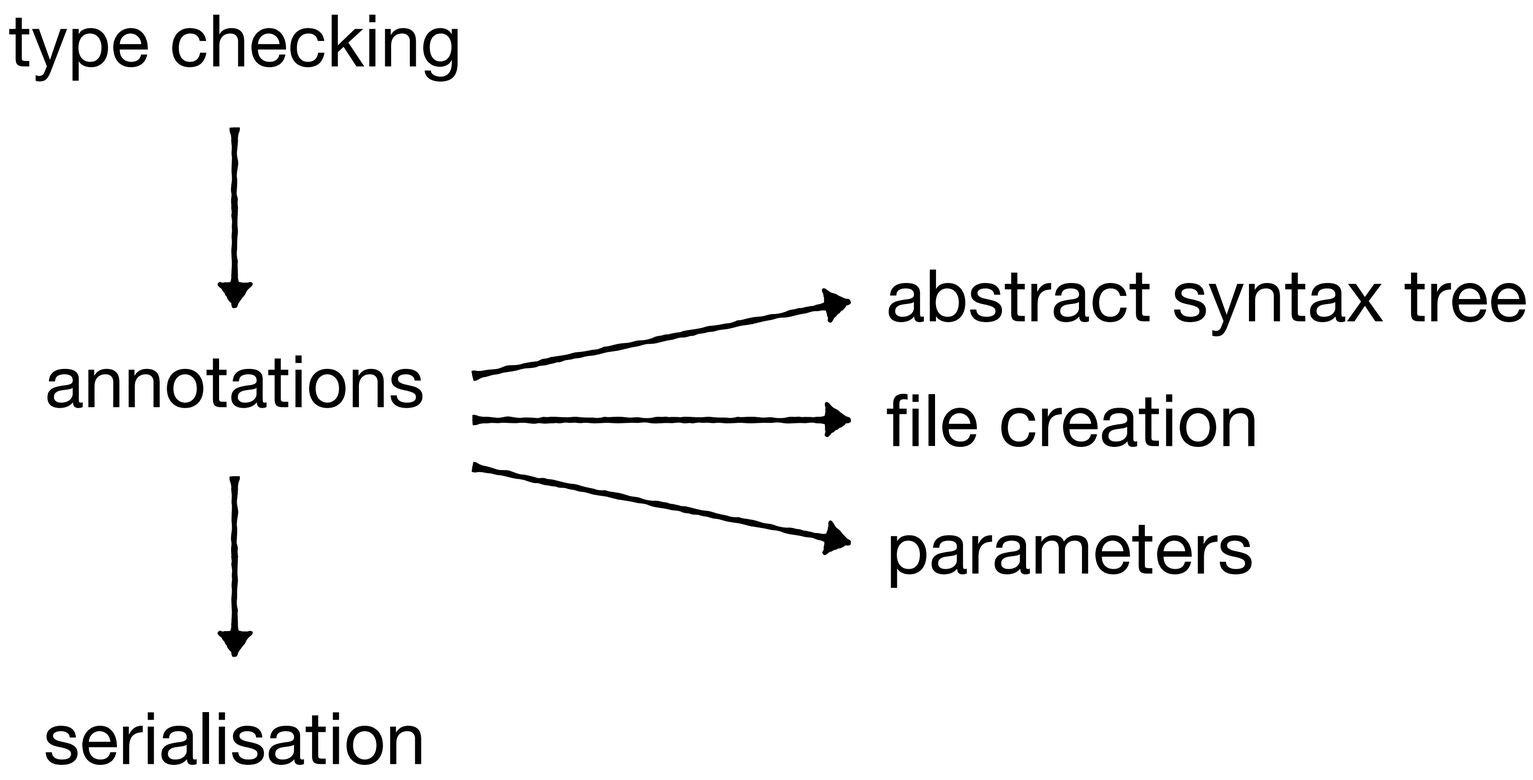
Transform

annotated abstract syntax tree

```
<node p="64">
  <option p="26" name="name" RosCpp="&quot;hello world&quot;">
    <string p="26" RosCpp="&quot;hello world&quot;">hello world</str
  </option>
  <option p="63" name="initialise" RosCpp="ROS_INFO_STREAM(&quot;hell
    <print p="62" RosCpp="ROS_INFO_STREAM(&quot;hello world!&quot;)">
      <string p="61" RosCpp="&quot;hello world!&quot;">hello world!</
      <option name="level" RosCpp="&quot;info&quot;">
        <string RosCpp="&quot;info&quot;">info</string>
      </option>
    </print>
  </option>
  <option name="definitions" RosCpp=""/>
  <option name="rate" RosCpp="25">
    <real RosCpp="25">25</real>
  </option>
  <option name="finalise" RosCpp=""/>
  <option name="cachedComputation" RosCpp=""/>
</node>
```

Code snippets

Transformation phase



Transformation phase

code snippets, output inheritance

C++

```
'print': {
    'output':
    {
        'Cpp': 'std::cout << {{children|join(" << ")}} << std::endl',
    },
}
```

Abstract syntax tree annotations

RosCpp

```
'print': {
    'output':
    {
        'RosCpp': 'ROS_INFO_STREAM({{children|join(" << ")}})',
    },
}
```

Output phase

annotated abstract syntax tree

```
<node p="64">
  <option p="26" name="name" RosCpp="&quot;hello
    <string p="26" RosCpp="&quot;hello world&quo
  </option>
  <option p="63" name="initialise" RosCpp="ROS_I
    <print p="62" RosCpp="ROS_INFO_STREAM(&quot;
      <string p="61" RosCpp="&quot;hello world!&
      <option name="level" RosCpp="&quot;info&qu
        <string RosCpp="&quot;info&quot;">info</
      </option>
    </print>
  </option>
  <option name="definitions" RosCpp=""/>
  <option name="rate" RosCpp="25">
    <real RosCpp="25">25</real>
  </option>
  <option name="finalise" RosCpp=""/>
  <option name="cachedComputation" RosCpp=""/>
</node>
```

Output →

c++ code

```
namespace hello_world
{
  /****************************** constructor ***********************************
  HelloWorldClass::HelloWorldClass() :
    nh_("~")

  {
  }

  /****************************** initialise ************************************
  void HelloWorldClass::initialise()
  {
    /* initialisation */
    ROS_INFO_STREAM("hello world!");
  }

  /****************************** finalise **************************************
  void HelloWorldClass::finalise()
  {
  }

  /****************************** spin ******************************************
  void HelloWorldClass::spin()
  {
    // Sets the spin rate
    ros::Rate r(25);

    while(ros::ok() )
    {
      ros::spinOnce();

      r.sleep();
    }
  }
}
```

Output phase

templates, code injection, output inheritance

transformer
c++

```
{% set initialise %}
std::string s = ‘hello’
{% endset %}
```

transformer
RosCpp

```
{% set initialise %}
std_msgs::String s = ‘hello’
{% endset %}
```

output: RosCpp
nodename.cpp

```
int main()
{
  <<<initialise>>>
  ros::Rate r({{rate}})
```

Output phase

Code snippets

```
'Cpp': 'std::cout << {{children|join(" << ")}} << std::endl',
```

↓

```
'RosCpp': 'ROS_INFO_STREAM({{children|join(" << ")}})',
```

Templates

output
c++

transformer
RosCpp

output
RosCpp

hello_world.cpp

rol compiler

supporting tools

Inputs:

parser plug-in generator: generic, xml, yaml, json
grammar generator: precedence order
xml namespaces

Transformations:

plugin generator
plugin inheritance
code injection
xpath, dpath search support

Outputs:

plugin generator
template engine

make your own language!
make your own compiler!