MarGotAspect - XML Configuration File User Manual

Arcari Leonardo, Galimberti Andrea

June 27, 2017

1   About

This document is intended to be used as a reference for the elements of the XML configuration file. It describes the use and meaning of each element providing examples. MarGotAspect is intended as a code generator for the mARGOt autotuning framework, therefore it uses concepts and constructs for which to refer to the mARGOt framework documentation.

2   Document Type Definition

```
<!ELEMENT margot (aspect*)>
<!ELEMENT aspect (function-monitor*, region-monitor*, goal-tuner*, state-tuner*)>
<!ATTLIST aspect block_name CDATA #REQUIRED>

<!ELEMENT function-monitor (function-name, return-type, argument*, configure-call)>
<!ELEMENT function-name (#PCDATA)>
<!ELEMENT return-type (#PCDATA)>
<!ELEMENT argument (type, name)>
<!ATTLIST argument sw-knob CDATA #REQUIRED>
<!ELEMENT type (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT configure-call (#PCDATA)>

<!ELEMENT region-monitor (argument+, configure-call)>

<!ELEMENT goal-tuner (control-var, goal-name, rule+)>
<!ELEMENT control-var (type, name)>
<!ELEMENT goal-name (#PCDATA)>
<!ELEMENT rule ((predicate|and|or|not), value)>
<!ELEMENT predicate (#PCDATA)>
<!ATTLIST predicate type CDATA #REQUIRED>
<!ELEMENT and ((predicate|and|or|not), (predicate|and|or|not))>
<!ELEMENT or ((predicate|and|or|not), (predicate|and|or|not))>
<!ELEMENT not ((predicate|and|or|not))>
<!ELEMENT value (#PCDATA)>

<!ELEMENT state-tuner (control-var, rule+)>
```

3 Aspect

The most external element *margot* can contain multiple *aspect* elements, each with a block_name attribute corresponding to a block declared in the mARGOt framework configuration files.

Each *aspect* element can contain any number of function monitors, region monitors, goal tuners and state tuners, and will be mapped to a single aspect (with its related pointcuts and advices) inside the generated aspect file.

It has to be noted that, aside from the XML configuration file contents and the related aspects that will be generated, a "special" aspect is always created containing an advice such that a margot::init() call is performed at the start of the main function of the application program.

4 Monitors

Monitors can either be function monitors or region monitors.

The *argument* elements correspond to variables which can be configured in the mARGOt framework as knobs, or be standard parameters passed to the monitored function; this is specified with the *sw-knob* attribute of each argument, which can take values "*yes*" or "*no*".

The optional *configure-call* subelement corresponds to a function that has to be called whenever the values of the knobs are modified; this function has to be written in the XML document as:

```
function_name(arg1_type arg1_name, arg2_type, arg2_name, ...)
```

4.1 Function Monitor

A function monitor is characterized by the name (*function-name*), the return type (return-type*) and the arguments (*argument*) of the function that has to be monitored.

The generated aspect code will then start and stop the monitors for the corresponding knob parameters of the function, as configured in the mARGOt framework configuration files; logging activity for the monitors is also performed.

4.1.1 Example

Let's say we want to monitor the execution time of the function *do_work*. It takes as arguments two parameters, one is a normal parameter, the other one is a software knob in the mARGOt framework. This is the functional code the user wrote:

```cpp
int main() {
  int trials = 100; // Normal argument
  int knob = 2; // Sofware knob
  for (int i = 0; i < REPETITIONS; ++i) {
    do_work(trials, knob);
  }
} // Functional code
```

So we write an XML file describing our intention to monitor a function named *do_work* taking two parameters, one of which is a *software knob* in the block named *foo*.

```xml
<margot>
  <aspect block_name="foo">
    <function-monitor>
      <function-name>do_work</function-name>
      <return-type>void</return-type>
      <argument sw-knob="no">
```

```
      <type>int</type>
    <name>trials</name>
    </argument>
    <argument sw-knob="yes">
      <type>int</type>
    <name>knob</name>
    </argument>
  </function-monitor>
  </aspect>
</margot>
```

And here is the AspectC++ code that MarGotAspect generates for us. Note that it performs a check on the value of our software knob and then starts the monitor before calling *do_work()*. Lastly, after *do_work()* returned, it stops the monitor and logs the results.

```
aspect GeneralAspect {
  pointcut main_exec() = execution("int main(...)");
  advice main_exec() : before() {
    margot::init();
  }
};
aspect fooAspect {
  pointcut do_work_exec(int trials, int knob) = execution("void do_work(...)") &&
      args(knob);
  advice do_work_exec(trials, knob) : before(int knob) {
    if (margot::foo::update(knob)) {
      margot::foo::manager.configuration_applied();
    }
    margot::foo::start_monitor();
  }

  advice do_work_exec(knob) : after(int knob) {
    margot::foo::stop_monitor();
    margot::foo::log();
  }
};
```

4.2 Region Monitor

Not only function calls can be monitored, but also well-delimited blocks of code (i.e., the regions of interest); the block delimitation has to be manually performed inside the application code, by applying at the start of the block the function call:

margot_<block_name>_start_roi(knob1_name, knob2_name, ...)

where all the knobs to monitor are listed as its parameters and <block_name> corresponds to the attribute with the same name for the aspect element, and at the end of the block the function call:

margot_<block_name>_end_roi()

3

5   Tuners

Goal tuners and state tuners corresponds to advice code which allows to change at run-time the goal values or the state of the mARGOt framework, according to some variable (not just a software knob, as already automatically possible for the framework).

Both of them share the concept of rule, which allows associating each particular state or goal value to a logical predicate to be verified at run-time.

5.1   Rule

Rules are a key aspect for both tuners, as they couple together predicates and goal/state values.

Each *rule* element has indeed an associated *value* element, which corresponds to a possible value for either a state or a goal, and a predicate element.

Predicates can be either simple predicates (*predicate*) or composite predicates (*and*, *or*, *not* elements).

Simple predicates are assigned a predicate value (a numerical value, or a boolean) and they can be of "*eq*", "*neq*", "*gt*", "*gte*", "*lt*", "*lte*" types, as reported in the corresponding *type* attribute; the simple predicate will be satisfied when the associate variable, i.e., the *control-var* element, will have a value such that the *<control-var> <predicate-type> <predicate-value>* expression is true.

Composite predicates instead use the *and*, *or* and *not* elements to compose (both simple and composite) predicates with the corresponding boolean logic functions.

5.2   Goal Tuner

To each goal tuner, a goal previously defined in the mARGOt configuration files is associated through the *goal-name* element.

In the context of the goal tuners, the simple predicate element values correspond to possible values of this goal, and the value of the goal is updated to the value of some rule whose predicate is verified.

5.3   State Tuner

In the context of the state tuners, the simple predicate element values correspond to the names of states as defined in the mARGOt configuration files, and the state of the framework is updated to the value of some rule whose predicate is verified.