Language Framework for Optimal Schedulers (LFOS)

Guideline for Users

Güner Orhan

December 20, 2016

1 Required Modules

In order to use LFOS frameowrk API, a programmer should import the required module:

```
from LFOS.Scheduler.Scheduler import Scheduler
from LFOS.Resource.Resource import *
from LFOS.Task.Task import *
from LFOS.Scheduling.Characteristic.Time import Time
from LFOS.macros import *
```

Listing 1: Importing required modules

2 Scheduler

Based on the selected instance, the scheduler instance, namely "tsp_scheduler", can be generated by following the following procedures one-by-one:

2.1 Resource Initialization for "machine"

Since some of the task specifications are based on the resources, in the framework, a programmer is expected to define the resources, initially. As explained in the article, some of the specifications are inevitable for a resource. Therefore, it should be defined for each resource. Based on the feature model, the following attributes are inevitable for a resource:

- Capacity ($\mathcal{C}$): The capacity of the resource are required to determine the maximum amount of capacity which can be utilized per time unit.
- Type ($\mathfrak{R}$): This attribute categorize the resources based on *Abstractions* and *Identifier*.
- Mode ($\mathcal{X}$): The mode of a resource may be either *Shared* or *Exclusive*.

- Power Consumption ($\mathcal{V}$): A resource consumes power based on this attribute. The resource is either *Scalable* or not.
- Objective ($\mathcal{O}_\alpha$): This attribute is related with the resource-related objectives.

The only required specification for the instantiation is *Type* of the resource. The default values for all specification belonging to a resource are shown in Table 1.

Feature Name	**Variable Type**	**Initial Value**
Capacity ($\mathcal{C}$)	float	0.0
Type ($\Re$)	LFOS.Resource.Type.ResourceTypeList::Enum	proc_t
Mode ($\mathcal{X}$)	LFOS.Resource.Mode.ModeTypeList::Enum	CB_EXCLUSIVE
Power Consumption ($\mathcal{V}$)	LFOS.Resource.Power	None
Objective ($\mathcal{O}_\alpha$)	LFOS.Objective	None

Table 1: Default instance variables of the *Resource* module and their default values.

For *abstraction*, ACTIVE is selected. Therefore, you can create the type object using the following code segment:

```
machine_t = Type(ResourceTypeList.ACTIVE, 'machine_t')
```

Listing 2: Active resource type object instantiation

According to the specification, a programmer can create the resource giving type object and a name of the resource as arguments to the class method of the ResourceFactory class shown in Listing 3. For active resources, an object belonging to TerminalResource class is instantiated using factory method pattern to handle the optional feature under *Abstraction* sub-feature.

```
machine = ResourceFactory.create_instance(machine_t, 'machine')
```

Listing 3: Active resource instantiation using ResourceFactory class

2.1.1 Setting mode

There are three possible types for this attribute. These are:

- ModeTypeList.SHARED
- ModeTypeList.CB_EXCLUSIVE
- ModeTypeList.SB_EXCLUSIVE
- ModeTypeList.CB_AND_SB_EXCLUSIVE

The functionality of these modes are discussed in the article.

As shown in Table 1, the mode is initially set to ModeTypeList.CB_EXCLUSIVE. A programmer can change the mode of a resource by using the following code segment:

```
machine.set_mode(mode)
# mode --> ResourceTypeList::Enum
```

Listing 4: Setting the mode of a resource after creating a resource.

In order to check the mode of the resource, you can use the functions depicted in Listing 5.

```
machine.is_mode(mode) # mode --> ResourceTypeList::Enum
# returns True if the argument matches with the mode of the resource.

machine.is_exclusive()
# returns True if the mode of the resource is any one of the exclusive mode
    .
```

Listing 5: The functions for resource mode check.

According to your specification, you have selected at least CB_EXCLUSIVE mode for your resource "machine". Since the resource is set to this mode initially, you do not need to set it again.

2.1.2 Setting Power Consumption

There are three possible types of power consumption:

- PowerTypeList.FIXED_STATE_POWER_CONSUMPTION
- PowerTypeList.DISCRETE_STATE_POWER_CONSUMPTION
- PowerTypeList.CONTINUOUS_STATE_POWER_CONSUMPTION

Each of these types have their corresponding classes inheriting Resource class. Therefore, we have utilized factory method design pattern.

Since you have selected non-scalable power consumption for the resource. The following instantiation can be applied:

```
power_type = PowerTypeList.FIXED_STATE_POWER_CONSUMPTION
machine_pc = PowerFactory.create_instance(power_type, scale, consumption)
# Arguments: power_type --> PowerTypeList::Enum
#            scale --> float // a scale value within [0.0, 1.0]
#            consumption --> float // a power consumption of one-capacity
    per unit time
# it returns the instance belonging

machine.set_power_consumption(machine_pc)
```

Listing 6: Initializing power consumption module and setting it to the resource.

If you want to get the instance of power, then you can use the following function:

```
machine.get_power_consumption()
```

Listing 7: Getting power consumption module

All the other member functions for Power module is as follows:

```
FixedStatePowerConsumption.add_state(self, scale, pow_cons) -> boolean (
    Interface Function)
FixedStatePowerConsumption.get_active_power_state() -> dict
FixedStatePowerConsumption.get_max_power_state() -> list
FixedStatePowerConsumption.get_min_power_state() -> list
FixedStatePowerConsumption.get_power_consumption() -> float
FixedStatePowerConsumption.get_power_consumption_w_scale(scale) -> float (
    Interface Function)
FixedStatePowerConsumption.get_power_scale() -> float
FixedStatePowerConsumption.get_power_states() -> Numpy.array
FixedStatePowerConsumption.max_range_check(scale) -> boolean
FixedStatePowerConsumption.range_check(scale) -> boolean
FixedStatePowerConsumption.remove_state(scale) -> boolean (Interface
    Function)
FixedStatePowerConsumption.set_max_state(self, scale, pow_cons) -> boolean
    (Interface Function)
FixedStatePowerConsumption.set_min_state(scale, pow_cons) -> boolean (
    Interface Function)
FixedStatePowerConsumption.set_power_consumption(consumption) -> None
FixedStatePowerConsumption.set_power_mode(scale) -> boolean (Interface
    Function)
```

Listing 8: The member functions for FixedStatePowerConsumption module.