Language Framework for Optimal Schedulers (LFOS)

Guideline for Users

Güner Orhan

January 13, 2017

1 Required Modules

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

```
1 from LFOS.Scheduler.Scheduler import Scheduler
2 from LFOS.Resource.Resource import *
3 from LFOS.Task.Task import *
4 from LFOS.Scheduling.Characteristic.Time import Time
5 from LFOS.macros import *
```

Listing 1: Importing required modules

2 Scheduler

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

2.1 Resource Initialization for "cpu1"

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 ($\Re$): This attribute categorize the resources based on *Abstractions* and *Identifier*.
- Mode ($\mathcal{X}$): The mode of a resource may be either *Shared* or *Exclusive*.

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

```
1 cpu1_t = Type(ResourceTypeList.ACTIVE, 'cpu1_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.

```
1 cpu1 = ResourceFactory.create_instance(cpu1_t, 'cpu1')
```

Listing 3: Active resource instantiation using ResourceFactory class

```
cpu1.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.

```
cpu1.is_mode(mode) # mode --> ResourceTypeList::Enum
# returns True if the argument matches with the mode of the resource.

cpu1.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 "cpu1". 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.

You have selected discrete-state power consumption. Therefore, you can create your power consumption object with the code as follows:

```
power_type = PowerTypeList.DISCRETE_STATE_POWER_CONSUMPTION
cpu1_pc = PowerFactory.create_instance(power_type, scale, consumption)
# create_instance(_type, min_scale, min_pow_cons, max_scale=None,
    max_pow_cons=None) -> FixedStatePowerConsumption |
    DiscreteStatePowerConsumption | ContinuousStatePowerConsumption
#
#         Returns the corresponding instance for given _type.
#
#         :param _type:
#             PowerTypeList.FIXED_STATE_POWER_CONSUMPTION |
#             PowerTypeList.DISCRETE_STATE_POWER_CONSUMPTION |
#             PowerTypeList.CONTINUOUS_STATE_POWER_CONSUMPTION
#         :param min_scale: float -> minimum power scale
#         :param min_pow_cons: float -> minimum power consumption
#         :param max_scale: float -> maximum power scale
#         :param max_pow_cons: float -> maximum power consumption
```

```
15 #          :return: FixedStatePowerConsumption |
       DiscreteStatePowerConsumption | ContinuousStatePowerConsumption
```

Listing 6: Power consumption object is created for Discrete-State Power Consumption type.

All the other member functions for the object is shown in Listing 7:

```
1  DiscreteStatePowerConsumption.get_power_states() -> Numpy.array
2  DiscreteStatePowerConsumption.set_power_mode(scale) -> list
3  DiscreteStatePowerConsumption.range_check(scale) -> boolean
4  DiscreteStatePowerConsumption.get_max_power_state() -> list
5  DiscreteStatePowerConsumption.get_min_power_state() -> list
6  DiscreteStatePowerConsumption.get_active_power_state() -> dict
7  DiscreteStatePowerConsumption.remove_state(scale) -> float | None
8  DiscreteStatePowerConsumption.max_range_check(scale) -> boolean
9  DiscreteStatePowerConsumption.get_type() -> PowerTypeList
10 DiscreteStatePowerConsumption.get_power_consumption_w_scale(scale) -> float
11 DiscreteStatePowerConsumption.add_state(scale, pow_cons) -> boolean
12 DiscreteStatePowerConsumption.set_max_state(scale, pow_cons) -> boolean
13 DiscreteStatePowerConsumption.get_power_scale() -> float
14 DiscreteStatePowerConsumption.get_power_consumption() -> float
15 DiscreteStatePowerConsumption.set_min_state(scale, pow_cons) -> boolean
```

Listing 7: The member functions for DiscreteStatePowerConsumption module.

2.2 Resource Initialization for "memory1"

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

```
1 memory1_t = Type(ResourceTypeList.PASSIVE, 'memory1_t')
```

Listing 8: Passive 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 9. For passive resources, an object belonging to TerminalResource class is instantiated using factory method pattern to handle the optional feature under *Abstraction* sub-feature.

```
1 memory1 = ResourceFactory.create_instance(memory1_t, 'memory1')
```

Listing 9: Passive resource instantiation using ResourceFactory class

2.2.1 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:

```
memory1.set_mode(mode)
# mode --> ResourceTypeList::Enum
```

Listing 10: 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 11.

```
memory1.is_mode(mode) # mode --> ResourceTypeList::Enum
# returns True if the argument matches with the mode of the resource.

memory1.is_exclusive()
# returns True if the mode of the resource is any one of the exclusive mode
    .
```

Listing 11: The functions for resource mode check.

According to your specification, you have selected at least CB_EXCLUSIVE mode for your resource "memory1". Since the resource is set to this mode initially, you do not need to set it again.

2.2.2 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
memory1_pc = PowerFactory.create_instance(power_type, scale, consumption)
# create_instance(_type, min_scale, min_pow_cons, max_scale=None,
    max_pow_cons=None) -> FixedStatePowerConsumption |
    DiscreteStatePowerConsumption | ContinuousStatePowerConsumption
#
#         Returns the corresponding instance for given _type.
#
#         :param _type:
```

```
8  #                PowerTypeList.FIXED_STATE_POWER_CONSUMPTION |
9  #                PowerTypeList.DISCRETE_STATE_POWER_CONSUMPTION |
10 #                PowerTypeList.CONTINUOUS_STATE_POWER_CONSUMPTION
11 #           :param min_scale: float -> minimum power scale
12 #           :param min_pow_cons: float -> minimum power consumption
13 #           :param max_scale: float -> maximum power scale
14 #           :param max_pow_cons: float -> maximum power consumption
15 #           :return: FixedStatePowerConsumption |
       DiscreteStatePowerConsumption | ContinuousStatePowerConsumption
16
17 memory1.set_power_consumption(memory1_pc)
```

Listing 12: 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:

```
1 memory1.get_power_consumption()
```

Listing 13: Getting power consumption module

All the other member functions for the object is as follows:

```
1  FixedStatePowerConsumption.set_max_state(self, scale, pow_cons) -> boolean
       (Interface Function)
2  FixedStatePowerConsumption.max_range_check(scale) -> boolean
3  FixedStatePowerConsumption.get_type() -> PowerTypeList
4  FixedStatePowerConsumption.get_power_consumption_w_scale(scale) -> float (
       Interface Function)
5  FixedStatePowerConsumption.set_power_consumption(consumption) -> None
6  FixedStatePowerConsumption.get_active_power_state() -> dict
7  FixedStatePowerConsumption.get_max_power_state() -> list
8  FixedStatePowerConsumption.get_min_power_state() -> list
9  FixedStatePowerConsumption.get_power_states() -> Numpy.array
10 FixedStatePowerConsumption.set_min_state(scale, pow_cons) -> boolean (
       Interface Function)
11 FixedStatePowerConsumption.remove_state(scale) -> boolean (Interface
       Function)
12 FixedStatePowerConsumption.range_check(scale) -> boolean
13 FixedStatePowerConsumption.get_power_consumption() -> float
14 FixedStatePowerConsumption.set_power_mode(scale) -> boolean (Interface
       Function)
15 FixedStatePowerConsumption.add_state(self, scale, pow_cons) -> boolean (
       Interface Function)
16 FixedStatePowerConsumption.get_power_scale() -> float
```

Listing 14: The member functions for FixedStatePowerConsumption module.

3 Task Initialization for t1

In order to create a task instance, a programmer first needs to specify the granularity of the task. A task can be specified as either *terminal* or *composite*. This information comes from the feature diagram. Secondly, she has to define each mandatory keyword

Keyword	Type	Default Value	Description
name	string	-	The name of a task
periodicity	LFOS.Task.Periodicity.PeriodicityTypeList	-	The periodicty type of a task
deadline_type	LFOS.Task.Requirement.DeadlineRequirementTypeList	DeadlineRequirementTypeList.HARD	The deadline satisfaction type of a task
priority	int	0	The priority of a task
deadline	LFOS.Scheduling.Characteristic.Time	-	The deadline of a task
token_number	list(int)	[1]	The list of numbers of a specific token that would be fired. There exist one-to-one relation between the list and token_name list with respect to index.
phase	LFOS.Scheduling.Characteristic.Time	-	The first release time of a task
preemptability	LFOS.Task.Preemptability.PreemptionTypeList	PreemptionTypeList.FULLY_PREEMPTABLE	The preemptability of a task
type	string	-	The type of a task
token_name	list(str)	list("__<name>__"]	The list of names of the token which would be fired after completion of the task instance.

Table 2: *Keywords* to instantiate a task object.

that is inevitable to create task instance. The task model consists of many sub-feature models. It is necessary to explain these branches to make a programmar familiar with the terminology.

- Granularity ($\mathcal{G}$): The granularity of a task (*terminal* or *composite*) that is termed as TaskType in the implementation.

```
1 TaskTypeList.TERMINAL
2 TaskTypeList.COMPOSITE
```

Listing 15: The enumeration of the type of tasks.

- Timing (T): This attribute includes all the relevant time-related task properties such as *release time*, *execution time*, *deadline*, and *period* information.
- Requirement ($\mathcal{R}$): The requirements of tasks that are *Resource Requirement* and *Deadline Requirement*.
- Priority (ρ): The priority information of a task.
- Dependency (Δ): The dependency relation with respect to other tasks, which is optional.
- Preemptable ($\mathcal{P}_\tau$): The preemptability property of a task, which is optional.
- Objective ($\mathcal{O}_\tau$): This attribute holds the task-related objective information, which is optional.

In the second phase, a programmer should specify the values for each attribute in the feature model. The keywords and correlated information is represented in Table 2

3.1 Specifying the Granularity of task "t1"

There are two types of tasks, namely *Terminal* and *Composite*. These are enumerated within TaskTypeList as follows:

Since you have defined the task "t1" as *Terminal*, you should the following line:

```
task_type_t1 = TaskTypeList.TERMINAL
```

Listing 16: Terminal task type definition.

3.1.1 Setting the time attributes

The time-related attributes for a task is kept under Timing class that inherits Periodicity class. Since the task class also inherits from the Timing class, a programmer can set and get the values directly from an intance of a task. The following member functions are utilized to interact with the task instance to access the timing attributes defined above:

```
Timing.set_deadline(new_deadline) -> None
Timing.str(object='') -> string
Timing.set_release_time(new_release_time) -> None
Timing.get_release_time() -> LFOS.Scheduling.Characteristic.Time
Timing.get_deadline() -> LFOS.Scheduling.Characteristic.Time
Timing.set_period(new_period) -> boolean
Timing.set_execution_time(new_exec_time) -> None
Timing.get_period_type() -> LFOS.Scheduling.Characteristic.Time
Timing.get_period() -> LFOS.Scheduling.Characteristic.Time
Timing.get_execution_time() -> LFOS.Scheduling.Characteristic.Time
Timing.set_periodicity(_type) -> None
```

Listing 17: The member functions for Timing module.

The periodicity types are as follows:

```
PeriodicityTypeList.APERIODIC
PeriodicityTypeList.PERIODIC
PeriodicityTypeList.SPORADIC
```

Listing 18: The Periodicity type enumeration.

Since you have classified task "t1" as PERIODIC, you can set a parameter for this attribute as follows:

```
periodicity_t1 = PeriodictyTypeList.PERIODIC
```

Listing 19: Periodicity for parameter is stored in the variable to use it later in instantiation.

3.1.2 Setting the priority

The priority class is a compact small class consists of three class vairables:

- MIN_PRIORITY: the minimum value for priority value.
- MAX_PRIORITY: the maximum value for priority value.
- IMPORTANCE_RANKING: the importance criteria stating whether ascending or descending priority values have higher privilege for a task. PriorityRanking class includes this enumeration attributes as class variables:

```
1 PriorityRanking.ASCENDING
2 PriorityRanking.DESCENDING
```

Listing 20: The PriorityRanking enumeration.

This class also includes three functions, two of which setting and getting the priority value, and one of which is responsible for setting the class variables:

```
1 Priority.get_priority() -> int
2 Priority.set_class_vars(min_prio=None, max_prio=None, ranking=None) -> None
      (class method)
3 Priority.set_priority(new_prio) -> boolean
```

Listing 21: The member functions for Priority module.

4 Task Initialization for t2

4.1 Specifying the Granularity of task "t1"

There are two types of tasks, namely *Terminal* and *Composite*. These are enumerated within TaskTypeList as follows:

Since you have defined the task "t1" as *Terminal*, you should the following line:

```
1 task_type_t1 = TaskTypeList.TERMINAL
```

Listing 22: Terminal task type definition.

4.1.1 Setting the time attributes

The time-related attributes for a task is kept under Timing class that inherits Periodicity class. Since the task class also inherits from the Timing class, a programmer can set and get the values directly from an intance of a task. The following member functions are utilized to interact with the task instance to access the timing attributes defined above:

```
1  Timing.set_deadline(new_deadline) -> None
2  Timing.str(object='') -> string
3  Timing.set_release_time(new_release_time) -> None
4  Timing.get_release_time() -> LFOS.Scheduling.Characteristic.Time
5  Timing.get_deadline() -> LFOS.Scheduling.Characteristic.Time
6  Timing.set_period(new_period) -> boolean
7  Timing.set_execution_time(new_exec_time) -> None
8  Timing.get_period_type() -> LFOS.Scheduling.Characteristic.Time
9  Timing.get_period() -> LFOS.Scheduling.Characteristic.Time
10 Timing.get_execution_time() -> LFOS.Scheduling.Characteristic.Time
11 Timing.set_periodicity(_type) -> None
```

Listing 23: The member functions for Timing module.

The periodicity types are as follows:

```
1 PeriodicityTypeList.APERIODIC
2 PeriodicityTypeList.PERIODIC
3 PeriodicityTypeList.SPORADIC
```

Listing 24: The Periodicity type enumeration.

Since you have classified task "t1" as APERIODIC, you can set a parameter for this attribute as follows:

```
1 periodicity_t1 = PeriodictyTypeList.APERIODIC
```

Listing 25: Periodicity for parameter is stored in the variable to use it later in instantiation.

4.1.2 Setting the priority

The priority class is a compact small class consists of three class vairables:

- MIN_PRIORITY: the minimum value for priority value.
- MAX_PRIORITY: the maximum value for priority value.
- IMPORTANCE_RANKING: the importance criteria stating whether ascending or descending priority values have higher privilege for a task. PriorityRanking class includes this enumeration attributes as class variables:

```
1 PriorityRanking.ASCENDING
2 PriorityRanking.DESCENDING
```

Listing 26: The PriorityRanking enumeration.

This class also includes three functions, two of which setting and getting the priority value, and one of which is responsible for setting the class variables:

```
1 Priority.get_priority() -> int
2 Priority.set_class_vars(min_prio=None, max_prio=None, ranking=None) -> None
      (class method)
3 Priority.set_priority(new_prio) -> boolean
```

Listing 27: The member functions for Priority module.