Language Framework for Optimal Schedulers (LFOS)

Guideline for Users

Güner Orhan

December 18, 2016

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*.

- 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:

```
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 11. 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

2.2 Resource Initialization for "cpu2"

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

```
1 cpu2_t = Type(ResourceTypeList.ACTIVE, 'cpu2_t')
```

Listing 4: 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 13. For active resources, an object belonging to TerminalResource class is instantiated using factory method pattern to handle the optional feature under *Abstraction* sub-feature.

```
cpu2 = ResourceFactory.create_instance(cpu2_t, 'cpu2')
```

Listing 5: Active resource instantiation using ResourceFactory class

2.3 Resource Initialization for "cpu1"

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

```
cpu1_t = Type(ResourceTypeList.ACTIVE, 'cpu1_t')
```

Listing 6: 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 11. For active resources, an object belonging to TerminalResource class is instantiated using factory method pattern to handle the optional feature under *Abstraction* sub-feature.

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

Listing 7: Active resource instantiation using ResourceFactory class

2.4 Resource Initialization for "cpu2"

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

```
cpu2_t = Type(ResourceTypeList.ACTIVE, 'cpu2_t')
```

Listing 8: 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 13. For active resources, an object belonging to TerminalResource class is instantiated using factory method pattern to handle the optional feature under *Abstraction* sub-feature.

```
cpu2 = ResourceFactory.create_instance(cpu2_t, 'cpu2')
```

Listing 9: Active resource instantiation using ResourceFactory class

2.5 Resource Initialization for "cpu1"

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

```
cpu1_t = Type(ResourceTypeList.ACTIVE, 'cpu1_t')
```

Listing 10: 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 11. 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 11: Active resource instantiation using ResourceFactory class

2.6 Resource Initialization for "cpu2"

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

```
1 cpu2_t = Type(ResourceTypeList.ACTIVE, 'cpu2_t')
```

Listing 12: 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 13. 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 cpu2 = ResourceFactory.create_instance(cpu2_t, 'cpu2')
```

Listing 13: Active resource instantiation using ResourceFactory class