# Lifecycle Parallel Shutdown

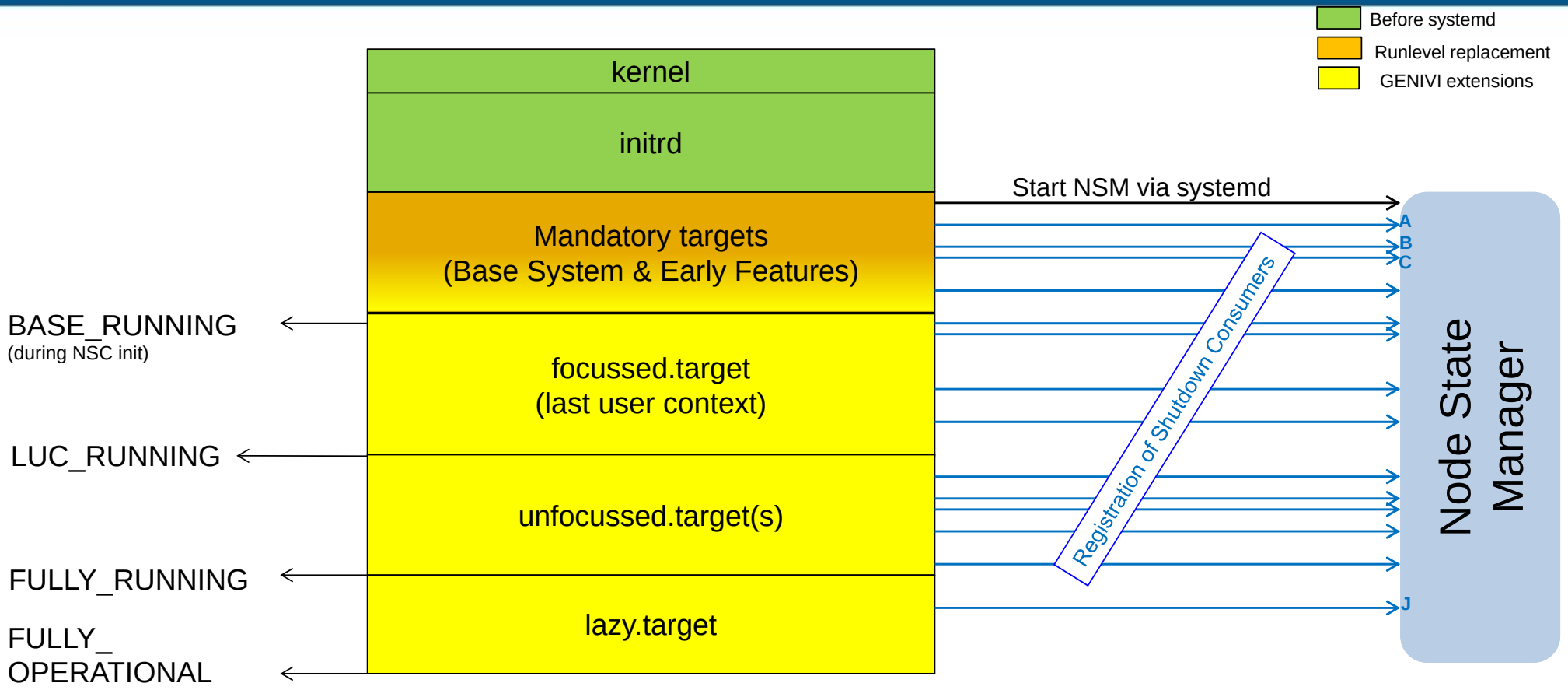
27/04/2016 15.00

David Yates
Software Architect/Lifecycle Topic Lead
Continental Automotive

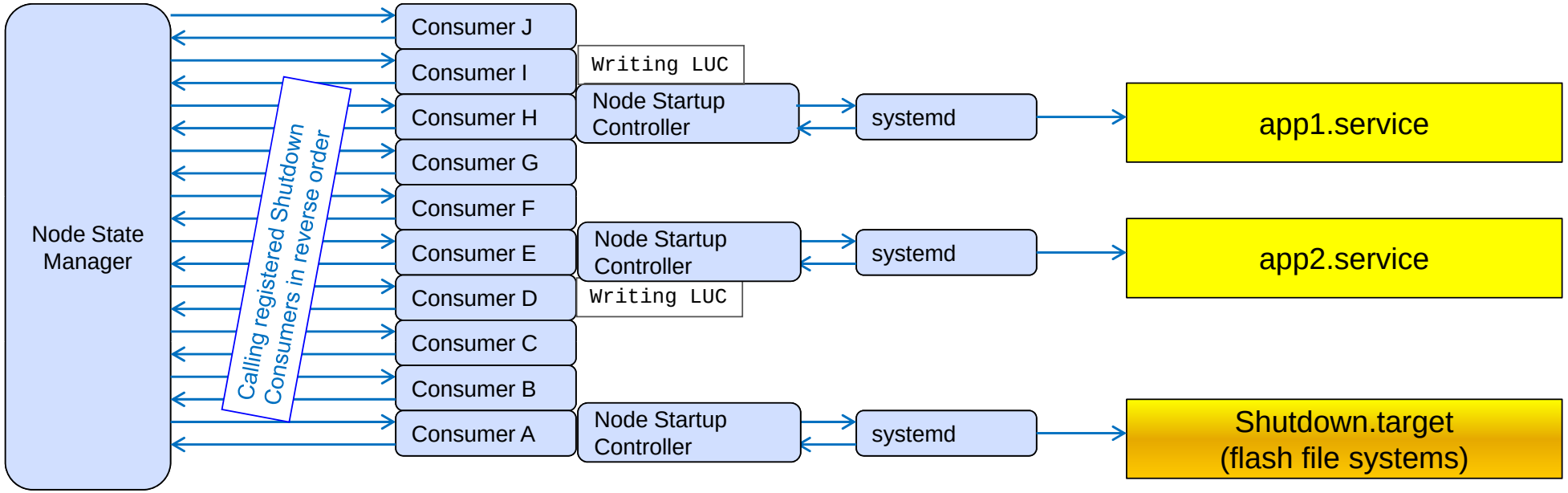
# Overview of the problem

- The Node State Manager (NSM) currently only performs a sequential shutdown
- Clients can specify a timeout value for their max shutdown time
- Experience shows that all clients will over estimate the time they need and if multiple clients then require their full timeout value it can lead to a really long shutdown time
- In worst cases the long shutdown can be overridden in a product where the “Entertainment” node is monitored and controlled by an Automotive/Vehicle Controller node that simply turns the power off if shutdown takes too long
- This can lead to a loss of data as clients are never informed and cached persistency data is never written

# Current shutdown preparation in startup phase



# Current shutdown execution




Enables:

1. Shutdown activities are trigger able without unloading the components.
2. Legacy components can be shut down in their traditional way.
3. Full flexibility on where to integrate systemd based shutdown units.

# New requirements and way of working

- Reduce the time required to shutdown the system
- Remove the possibility for one or more misbehaving clients to block the system shutdown
- Remove the situation where critical persistent components never get the chance to flush data from the current lifecycle
- and provide a way to do this without affecting backwards compatibility
- Provide more flexibility for a product/platform integrator to define and alter the behavior during the shutdown

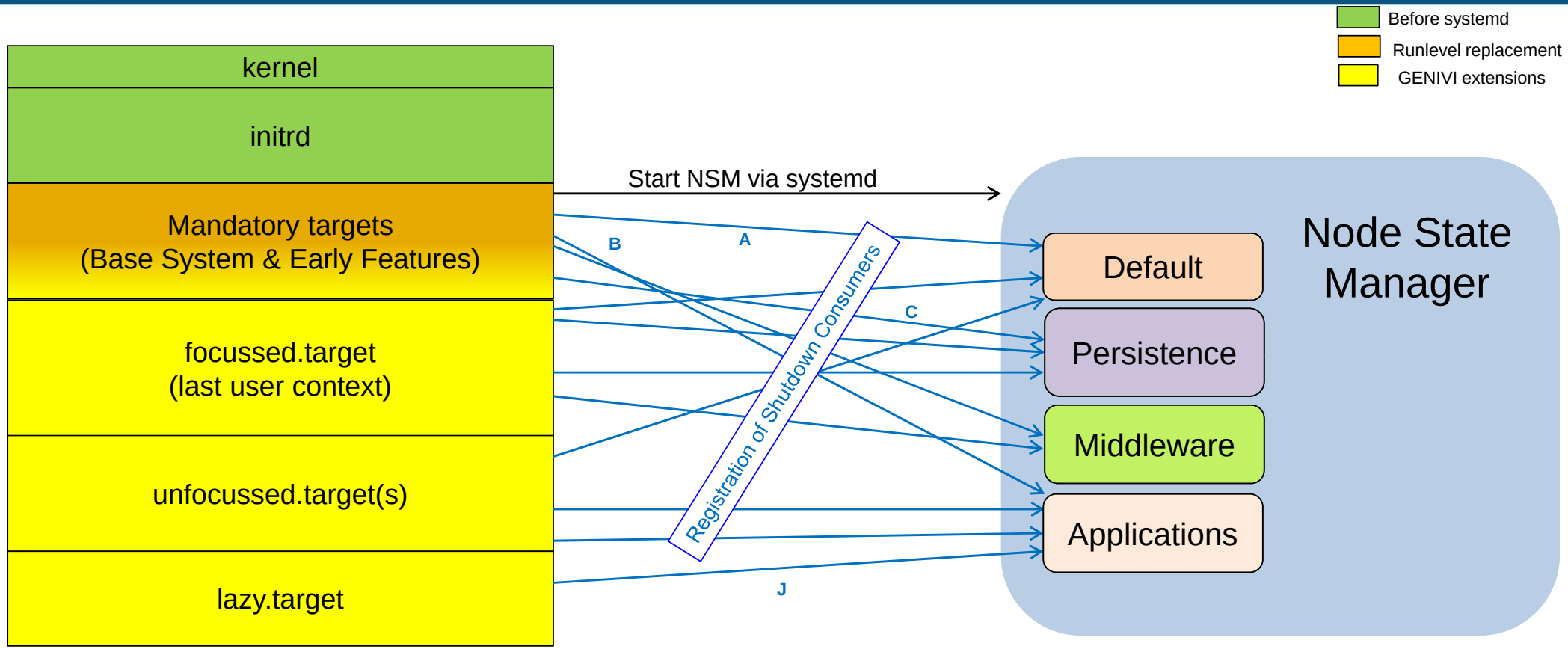
# Support for older clients/platforms

As mentioned previously, it is mandatory that older clients should work with the new NSM, and also that existing platform configurations should work as they do currently. We handle this by :

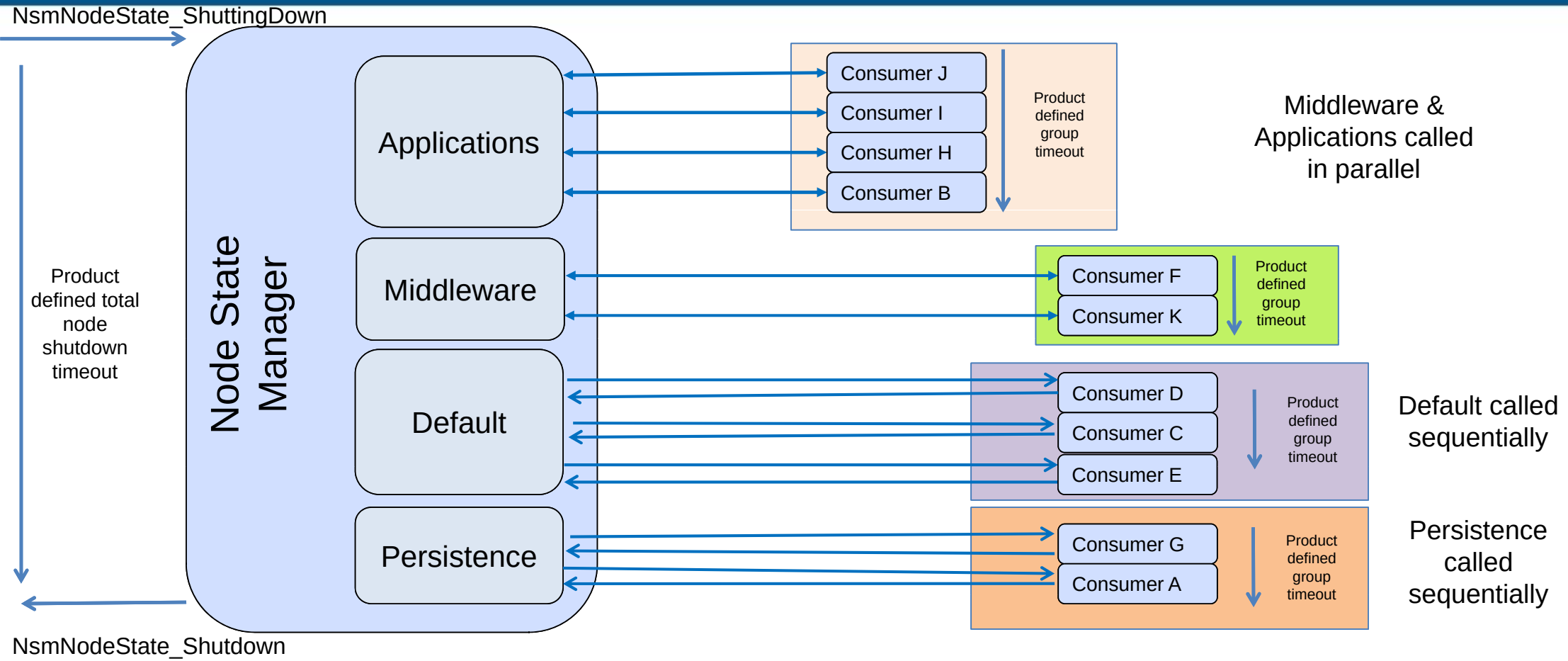
- leaving the existing interface as is and adding a new one
- adding a new configuration file which can be used to define how the NSM should handle the registrations
- introducing a concept of “groups” that can be staged
- all registration calls to the NSM using the older interface will by *default* be added in the “Default” group which is always handled sequentially (order defined by the order of registration)

NOTE: It is intended that the “Default” group is additionally used for Core Infrastructure components (i.e. Persistency and Lifecycle) in the system that normally have dependencies on them

# New shutdown preparation in startup phase



# New shutdown process



# Interface change

To maintain backwards compatibility companies that have already implemented NSM Clients, a new interface *org.genivi.NodeStateManager.Shutdown* has been added to the Node State Manager and all old interfaces remain unchanged.

The new interface will provide new methods and signals as defined in the following slides.

The benefit of the new interfaces are:

- removal of DBUS specific parameters (Bus Name & Obj  Name)
- support for sequential and parallel shutdown methods

# Interface change cont..

```
<method name="RegisterShutdownClient">
        <arg name="GroupName" type="s" direction="in" />
        <arg name="ShutdownMode" type="u" direction="in" />
                <arg name="TimeoutMs" type="u" direction="in" />
        <arg name="ClientId" type="s" direction="out" />
        <arg name="ErrorCode" type="i" direction="out" />
</method>
<method name="UnRegisterShutdownClient">
        <arg name="ClientId" type="s" direction="out" />
        <arg name="ShutdownMode" type="u" direction="in" />
        <arg name="ErrorCode" type="i" direction="out" />
</method>
<method name="LifecycleRequestComplete">
        <arg name=" ClientId " type="s" direction="in" />
        <arg name="Status" type="i" direction="in" />
        <arg name="ErrorCode" type="i" direction="out" />
    </method>

<signal name="LifecycleRequest">
        <arg name="ClientPath" type="s" />
         <arg name="Request" type="u" direction="in" />
</signal>
```

Here it is shown that the DBUS Bus and Object name are no longer needed.

The GroupName defines the group that the client wants to be a member of

The *ClientId* is a unique id allocated by the NSM to the registering client

The DBUS Bus and Object name are no longer required as the

the *ClientId* is a unique id allocated by the NSM during the registration

The *ClientId* is the unique id allocated by the NSM during the registration above

The parameter "*ClientPath*" is a path string that indicates the request is valid for a Group (parallel) or a specific client (seq).

The "*Request*" will be one of the existing NSM_SHUTDOWNTYPE_XXX types (i.e. shutdown, runup).

# Interface change cont…

A client will register for the signal using the "*ClientPath*" which is created from their GroupName and unique ClientId (out parameter from the *RegisterShutdownClientGroup* method call).

For instance if the "*ClientPath*" is "/Middleware/123456" then the NSM could signal

| | |
|---|---|
| / | -> Complete node (unlikely to be used) |
| /Middleware | -> Group specific (Parallel) |
| /Middleware/123456 | -> Client Specific (Sequential) |

and the client would receive the signal in all 3 cases

As signals have no return value, every client that has received such a signal has to call the LifecycleRequestComplete method within a certain timeout to notify their completion of the request.

NOTE: For backwards compatibility any clients registering with the old interface will still be notified using the old method and not the new signal.

# Configuration file

In the example configuration file shown on the right, it can be seen that 4 groups have been defined and each has it's own configuration:

- the total group time can be defined so that if a component in one group fails, it does not block the complete shutdown. Rather the next group will be started
- each group can define whether to be handled in parallel or sequentially
- there is always a "Default" group which will be used by *default* when the old registration interface is used
- an optional tag is allowed called *MaxClientTimeout* which allows the system designer to define the max. timeout allowed for components in a certain group. When included the NSM will override the requested value passed by the client in the interface (debug output will be provided in this use case)
- the shutdown order of the groups is defined by the dependency tags *Before* and *After*. In the example here it is shown that the parallel groups *Applications* and *Middleware* will actually themselves be done in parallel

```
[Applications]
ShutdownTimeout=25000
FastShutdownTimeout=3000
Order=parallel
Before=Default

[Middleware]
ShutdownTimeout=25000
FastShutdownTimeout=3000
Order=parallel
Before=Default

[Default]
ShutdownTimeout=25000
FastShutdownTimeout=3000
MaxClientTimeout=5000
Order=sequential
Before=Persistence

[Persistence]
ShutdownTimeout=25000
FastShutdownTimeout=3000
Order=sequential
After=Default
```

# Advantages of new approach

The advantages seen with the proposed change, based on real product development, are the ability to :

- perform parallel shutdown actions to reduce the time needed for the shutdown phase
- reduce the risk of one misbehaving component from blocking all components in the system from getting a shutdown notification and hence increase chances that customer data from that lifecycle is not lost
- have a higher granularity of timeouts via being able to skip a group that is timing out and move on to the next group in the list, hence increasing the chances of a successful shutdown before a hard power off is performed
- better handle failing/recovering components during normal run-time
- removal of DBUS specific parameters in the interface