Virtual Infrastructure Operating System (VIOS)

EF-PI SMA inverter driver and manager

Status: Concept
Version: 1.1
Date: 10-6-2015

Contents

1 SMA Inverter (PV panel)

This document describes the integration of the SMA inverter within EF-pi.

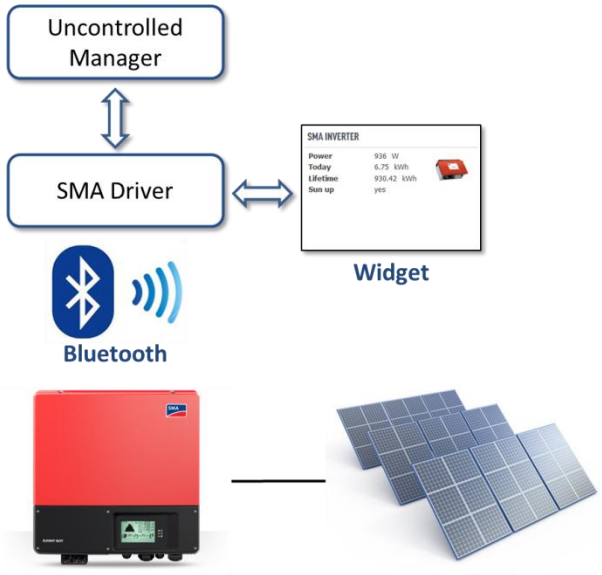

Figure 1: Overview of components for SMA inverter

Figure 1 presents an overview of all components (hard- and software) involved. The PV panel are connected via a cable with the SMA inverter. The inverter provides information on PV generation such as current power and the total amount of energy that has been produced.

The SMA inverter features a Bluetooth interface which is used by the SMA driver to obtain the information about the PV performance. The driver makes this information available for the Uncontrolled Manager that translate this into the messages that are specified by the Energy Flexibility Interface.

1.1 SMA resource driver

1.1.1 Configuration

```
    public interface Config {
        @Meta.AD(description = "The unique resource identifier", deflt = "sma-
inverter")
        String resourceId();

        @Meta.AD(description = "The MAC Address of the bluetooth receiver on the SMA
inverter", deflt = "00802529EC47")
        String macAddress();

        @Meta.AD(description = "The password of the bluetooth receiver (default
0000)", deflt = "0000")
        String password();
```

```
        @Meta.AD(description = "The latitude of the inverter location, used for
sunset/rise", deflt = "0")
        String latitude();

        @Meta.AD(description = "The longitude of the inverter location, used for
sunset/rise", deflt = "0")
        String longitude();

        @Meta.AD(description = "Offset in seconds to start before sunrise and end
after sunset", deflt = "900")
        int sunUpOffset();

        @Meta.AD(description = "The timezone identifier of the machine running this
code, used for sunset/rise",
                 deflt = "Europe/Amsterdam")
        String timezone();

        @Meta.AD(description = "The update frequency at which the inverter will be
read in seconds", deflt = "60")
        int updateFrequency();
    }
```

The code block shows the configuration parameters of the SMA driver. The most important parameters are the MAC address and the password that are required to communicate with the SMA inverter.

1.1.2 Power State

```
package org.flexiblepower.ral.drivers.uncontrolled;

import javax.measure.Measurable;
import javax.measure.quantity.Power;

import org.flexiblepower.ral.ResourceState;

public interface PowerState extends ResourceState {
    /**
     * @return The current power consumption (or production if negative).
     */
    Measurable<Power> getCurrentUsage();
}
```

The driver sends a PowerState object to the Uncontrolled Manager. This PowerState class is not part of the SMA Resource Driver itself. It is a generic interface that can be used by various kind of uncontrolled resource drivers. This interface only has one method (getCurrentUsage) that simply returns the power that is currently being consumed or produced.

1.1.3 Control Parameters

The SMA inverter cannot be controlled, therefore there are no control parameters required for this driver.

1.1.4 Widget

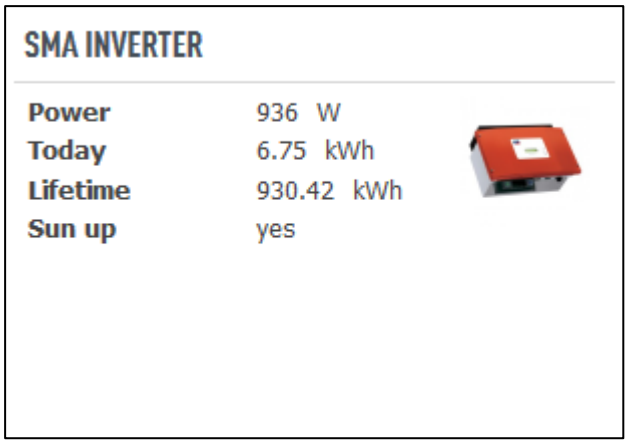

Figure 2: The widget for the SMA resource driver

The widget for the SMA inverter shows the following information:

- Current Power production in Watts.
- The total amount of energy produced today in kiloWattHours.
- The total amount of energy produced in the lifetime of the inverter in kiloWattHours.
- A boolean to indicate whether the sun is up or not. The driver uses the latitude and longitude to determine when the sun rises and sets. Latitude and longitude are configurable (see section 1.1.1). When the sun is down the SMA inverter goes into standby modus in order to save energy.

1.1.5 Bluetooth communication

Communication with the SMA inverter is made possible with Bluetooth. However the Bluetooth implementation of the SMA inverter does not completely follow the standard. This has to do with the way the messages are formatted. Because of these deviations a lot effort had to be put in to make the Bluetooth communication work.

1.2 Uncontrolled resource manager

The Uncontrolled resource manager is a generic manager for all kinds of uncontrolled appliances. As long as the driver of an appliance uses the PowerState class (see section 1.1.2), it can be coupled to this resource manager.

This generic uncontrolled resource manager does not support curtailment. However, this is not an issue for the SMA inverter that was used within the VIOS project as it does not have a curtailment option.

1.2.1 Configuration

```
    @Meta.OCD
    interface Config {
        @Meta.AD(deflt = "uncontrolled", description = "Resource identifier")
        String resourceId();

        @Meta.AD(deflt = "20", description = "Expiration of the ControlSpaces [s]",
required = false)
```

```
        int expirationTime();

        @Meta.AD(deflt = "false", description = "Show simple widget")
        boolean showWidget();
    }
```

The code snippet above shows the configuration options, which are quite limited.

- A resource ID can be entered to uniquely identifty this resource manager.
- An expiration time for the control spaces can be set.
- One can choose to show a widget. This widget is very simple; it only shows the current production or consumption of the uncontrolled appliance. In this case the widget is not being used as the SMA driver already features a widget that provides more information than this one.

1.2.2 Registration

The code snippet below shows the registration of the Uncontrolled resource manager.

```
    protected List<? extends ResourceMessage> startRegistration(PowerState state) {
        changedState = context.currentTime();
        allocationDelay = Measure.valueOf(5, SI.SECOND);
        ConstraintListMap constraintList = ConstraintListMap.electricity(null); //
this version of the uncontrolled
                                                                                 //
manager does not support
                                                                                 //
curtailments...
        UncontrolledRegistration reg = new UncontrolledRegistration(getResourceId(),
                                                                    changedState,
                                                                    allocationDelay,

CommoditySet.onlyElectricity, constraintList);
        UncontrolledUpdate update = createUncontrolledUpdate(state);
        return Arrays.asList(reg, update);
    }
```

As can be seen in the code this resource manager only supports electricity and cannot handle curtailments.

1.2.3 Control Space updates

```
    private UncontrolledUpdate createUncontrolledUpdate(PowerState state) {
        Measurable<Power> currentUsage = state.getCurrentUsage();
        CommodityMeasurables measurables =
CommodityMeasurables.electricity(currentUsage);
        UncontrolledUpdate update = new UncontrolledMeasurement(getResourceId(),
                                                                changedState,

context.currentTime(),
                                                                measurables);
        return update;
    }
```

A new Control Space update is constructed every time a new PowerState object is received. It simply takes the current power usage (generation is negative usage) and puts that in the Control Space update.

1.2.4 Allocations

The Uncontrolled resource manager has no method to process allocations since it does support curtailments.