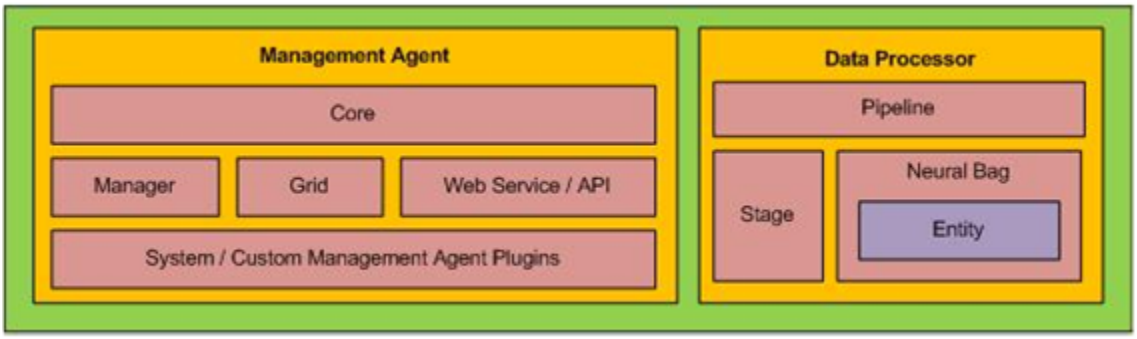
UCW Daemon

The daemon represents the main system service responsible for the node management. The daemon has simple functionality allowing to do basic operations such as load/unload plugins, daemon management, remote upgrades, and access to the host resources. The management agent plugins extend the daemon functionality.



UCW Daemon Architecture

The Management Agent is the first of the two major subsystems hosted within the daemon, and its responsibility is to manage the plugins (component Manager). The first subsystem provides access to the host resources (component Core), is interacting with other daemons through web service (REST, SOAP), and providing cooperation with the Data Processor subsystem. The second subsystem Data Processor does the data processing for the particular plugin. The *NeuralBag* (also known as the **bag**) is a data structure used to exchange data between connected daemons. The plugin can access the bag in a **pipeline** that allows data processing in multiple stages. Each **stage** returns the true or false which means whether the Data Processor will to continue in the processing or not. The pipeline allows you to modify the bag during stage processing. It's recommended to keep the functionality of the stage as simple as possible. We should have one stage for one operation. It can be for example a stage to read data from the sensor or send the bag to another daemon for next processing.

The bag represents basics data structure that daemon understand, and it's used by the daemon to exchange information between connected daemons. The management agent is processing via component Grid all incoming bags that have been sent by other daemons.

How to create NeuralBag

```
protected NeuralBag getBag(String payload) {
    Map<String, String> headers = new HashMap<String, String>();
    headers.put("data-type", "random");
    headers.put("sender", "generator");
    return NeuralBag.createBag(headers, payload);
}
```

NeuralBag example as JSON

```
{"sender":"generator","payload":"{\"name\": \"This is a message for host node!\", \"id\": 9}","data-type":"
random"}
```

Each *Management Agent* plugin can create a pipeline with multiple stages and run it in the *Data Processor* based on a trigger such as incoming bag matching a set of rules or can run as the scheduled task.

Data Processor stage example

```
public class SnmpCollectorProcessDataStage extends AbstractDataProcessorStage {

    @Override
    public boolean process(final NeuralBag bag, final DataProcessingThread thread) {
        getLog().debug("Entering CollectorProcessData stage...");

        if (bag.containsKey(NeuralBagName.PAYLOAD)) {
            getLog().debugFormat("Payload: %s", bag.get(NeuralBagName.PAYLOAD));

        } else {
            getLog().debug("NeuralBag does not contain attribute PAYLOAD!");
        }

        getLog().debug("Exiting CollectorProcessData stage...");

        return true;
    }
}
```

Data Processor pipeline configuration - Scheduled task

```
@Override
public DataProcessorConfig getDataProcessorConfig() {
    DataProcessorConfig config = new DataProcessorConfig();
    config.setRunnerJobEnabled(true);
    config.setRunnerJobDelay(60 * 1000); // starts after one minute
    config.setRunnerJobInterval((60 * 1000) * 60); // runs once per hour
    return config;
}

@Override
public List<DataProcessorStage> getDataProcessorStages() {
    List<DataProcessorStage> stages = new ArrayList<DataProcessorStage>();
    stages.add(new CollectorGetDataStage());
    stages.add(new CollectorValidateDataStage());
    stages.add(new CollectorStoreDataStage());
    return stages;
}
```

The good example of usage of the *Data Processor* is a plugin implementing the web crawler functionality. As we have mentioned earlier, the pipeline stage should do one simple task.

The plugin creates a pipeline with three stages.

1) Stage ***CollectorGetDataStage*** downloads a web page for a URI address defined in the bag and stores the HTML data in the bag.

CollectorGetDataStage.java

```
public class CollectorGetDataStage extends AbstractDataProcessorStage {

    @Override
    public boolean process(final NeuralBag bag, final DataProcessingThread thread) {
        getLog().debug("Entering CollectorGetDataStage stage...");

        getLog().info("Getting Collector data...");

        getLog().debug("Exiting CollectorGetDataStage stage...");

        return true;
    }
}
```

2) Stage ***CollectorValidateDataStage*** gets web page data stored in the bag and runs the analysis.

CollectorValidateDataStage.java

```
public class CollectorValidateDataStage extends AbstractDataProcessorStage {

    @Override
    public boolean process(final NeuralBag bag, final DataProcessingThread thread) {
        getLog().debug("Entering CollectorValidateDataStage stage...");

        getLog().info("Validating Collector data...");

        getLog().debug("Exiting CollectorValidateDataStage stage...");

        return true;
    }
}
```

3) Stage ***CollectorStoreDataStage*** gets the results of analysis from the bag, and save them in the storage.

CollectorStoreDataStage.java

```
public class CollectorStoreDataStage extends AbstractDataProcessorStage {

    @Override
    public boolean process(final NeuralBag bag, final DataProcessingThread thread) {
        getLog().debug("Entering CollectorStoreDataStage stage...");

        getLog().info("Storing Collector data to database...");

        getLog().debug("Exiting CollectorStoreDataStage stage...");

        return true;
    }
}
```

The UCW Daemon was designed based on the concept of the Nsys Daemon that is part of the Nsys Platform project.

Related Topics

- UCW Daemon Plugin