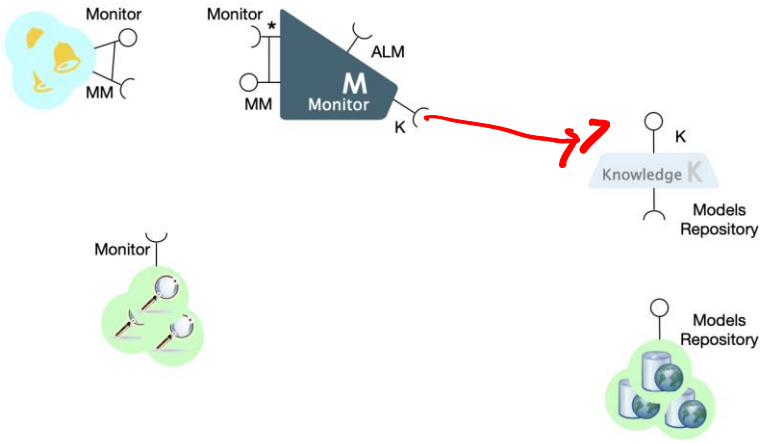
Módulo de Monitorización

Existen 3 componentes principales:
- Monitor Module: es el módulo del bucle MAPE-K que se encarga de mantener 'el proceso' de adaptación. Suele haber 1 por implementación (lo cambiaremos gracias a este trabajo) e interactúa con el módulo de Knowledge. En la Figura es el 'M Monitor'.



- Monitores ('campanitas' en la figura): representan a los monitores específicos creados para una solución. Si por ejemplo, se necesita comprobar cierto estado del sistema (por ejemplo, que debe operar en un entorno con unos niveles de temperatura y presión adecuados), se crea un monitor 'ParámetrosEntornoOperativo_Monitor' que recibirá datos de estos factores. Son los encargados de filtrar, limpiar, ordenar, etc. las lecturas que reciben para identificar que todo va bien, o intuir que se deba/pueda tomar alguna acción.

- Probes o Sondas ('lupas' en la figura): miden 'cosas de interés' (lo que sea) y reportan estas mediciones 'raw' a los monitores. Estas sondas en realidad están 'embebidas' (o adheridas) dentro del sistema que monitorizan (y no se despliegan junto al bucle).

Viendo este comportamiento, existen dependencias entre los componentes:

- Sondas (Probes) y Monitores : una sonda debe reportar a uno (o varios) monitores una lectura.

Al configurar un Monitor, se le conecta con un Módulo de Monitorización:

```
IMonitor theMsHealthStatusMonitor = new MicroserviceHealthStatus_Monitor(bundleContext);
IAdaptiveReadyComponent theMsHealthStatusMonitorARC =
        new MonitorARC(bundleContext, theMsHealthStatusMonitor);
theMsHealthStatusMonitorARC.start();
theMsHealthStatusMonitorARC.bindService(
        MonitorARC.REQUIRE_MONITORINGMODULESERVICE,
        theMonitoringModuleARC.getServiceSupply(
                MonitoringModuleARC.SUPPLY_MONITORINGMODULESERVICE));
theMsHealthStatusMonitorARC.deploy();
```

dónde previamente ya hemos obtenido el módulo de monitorización (a través de OSGi):

```
IAdaptiveReadyComponent theMonitoringModuleARC =
        OSGiUtils.getService(bundleContext,
                                        IAdaptiveReadyComponent.class,
                                        String.format("(%s=%s)",
                                        IIdentifiable.ID,
                                        MonitoringModuleARC.MODULE_ID));
```

A ver, de momento no te líes mucho con este código (ya te comentaré con más detalle más adelante, pero para que te familiarices con la dinámica), pero básicamente lo que está haciendo es crear un monitor 'MicroserviceHealthStatus_Monitor' (que es un IMonitor), y lo configura dinámicamente:
1) lo crea (start)
2) lo conecta con el theMonitoringModuleARC
3) lo despliega para que empiece a trabajar

En la actualidad, hacemos que los Monitores ofrezcan varias APIs (REST y MQTT)


```
MQTTInterfaceConnector4Monitor theHealthStatusMonitorMqttConnector =
        new MQTTInterfaceConnector4Monitor(bundleContext, theMsHealthStatusMonitor, mqttGateway);
theHealthStatusMonitorMqttConnector.start();

RESTInterfaceConnector4Monitor theHealthStatusMonitorRESTConnector =
        new RESTInterfaceConnector4Monitor();
theHealthStatusMonitorRESTConnector.setPort(8100).
                        setTheMonitor(theMsHealthStatusMonitor).
//                      setTheMonitorProxy(mqttGateway, monitorsBaseTopic).
                        start();
```

* Te prometo que no me acordaba que esto de los conectores lo había implementado ya (cosas de la pandemia ... ;-)

La Sonda, a su vez, se configura:

```
IAdaptiveReadyComponent theMsHealthStatusProbeARC =
        new ProbeARC(bundleContext, new MicroserviceHealthStatus_Probe(bundleContext));
theMsHealthStatusProbeARC.start();
theMsHealthStatusProbeARC.bindService(
        ProbeARC.REQUIRE_MONITORSERVICE,
        theMsHealthStatusMonitorARC.getServiceSupply(MonitorARC.SUPPLY_MONITORSERVICE));
theMsHealthStatusProbeARC.deploy();
```

Igual que antes, se configura dinámicamente, pero asociándolo con un monitor (con el MsHealthStatusMonitor que hemos configurado antes).

La implementación de una sonda:

```
package arm.embalpack.probes;

import org.osgi.framework.BundleContext;

import arm.mapek.resources.ARMMeasure;
import es.upv.pros.tatami.adaptation.mapek.lite.artifacts.components.Probe;

public class MicroserviceHealthStatus_Probe extends Probe {

	public MicroserviceHealthStatus_Probe(BundleContext context) {
		super(context, "Microservice-HealthStatus-Probe");
	}

	public void sendHealthStatus(ARMMeasure theMeasure) {
		this.reportMeasure(theMeasure);
	}


}
```

(no sé porqué me ha copiado sin formato de colores ... :-(

Dónde el reportMeasure() está implementado en la case Probe, y es donde se utilizaría alguno de los conectores de comunicación (actualmente no lo hace, se ve que me quedé ahí).

Por cierto, el método reportMeasure() recibe Object. Si lo vamos a enviar, tendremos que decidir alguna estrategia para 'serializar/des-serializar' la información (*marshaling / unmarshaling*).

```
protected void reportMeasure(Object measure) {
    if ( this.getTheMonitors() != null && this.getTheMonitors().size() > 0 ) {
      logger.info(String.format("(%s) Reporting measure: %s", this.getId(), measure.toString()));
      for(IMonitor theMonitor : this.getTheMonitors())
      theMonitor.report(measure);
    }
}
```

¿ Cómo Serializar / Deserializar?

¿ Tenemos un concepto de Measurement?

I Measurement <Pressure>

La implementación de un monitor:

```
package arm.embalpack.monitors;

import org.osgi.framework.BundleContext;

import arm.embalpack.datatypes.EHealthStatus;
import arm.mapek.resources.ARMMeasure;
import es.upv.pros.tatami.adaptation.mapek.lite.artifacts.components.Monitor;
import es.upv.pros.tatami.adaptation.mapek.lite.artifacts.interfaces.IKnowledge;
import es.upv.pros.tatami.adaptation.mapek.lite.artifacts.interfaces.IKnowledgeProperty;
import es.upv.pros.tatami.adaptation.mapek.lite.artifacts.interfaces.IMonitor;

public class MicroserviceHealthStatus_Monitor extends Monitor {

    public MicroserviceHealthStatus_Monitor(BundleContext context) {
        super(context, "Microservice-HealthStatus-monitor");
    }

    @Override
    public IMonitor report(Object measure) {

        logger.trace(String.format("(" + this.getId() + ") Received measure: %s",
                measure.toString()));

        try {
            ARMMeasure theMeasure = (ARMMeasure)measure;

            IKnowledge knowledge =
                this.getTheMonitoringModule().getTheKnowledgeModule().getTheKnowledge();

            EHealthStatus status = (EHealthStatus) theMeasure.getMeasure();
            String microservice_id = (String) theMeasure.getThing();
            String knowledgePropertyId = microservice_id+"_HealthStatus";
            IKnowledgeProperty kp = knowledge.getKnowledgeProperty(knowledgePropertyId);
            if ( kp == null ) {
                logger.trace(String.format("(" + this.getId() + ") Received measure: %s",
                            measure.toString()));
                logger.trace("Creating new Knowledge Property " + knowledgePropertyId +
                            " with value " + measure);
                kp = knowledge.createKnowledgeProperty(knowledgePropertyId, measure);
            } else if ( ((ARMMeasure)kp.getValue()).compareTo(theMeasure) != 0 ) {
                logger.trace(String.format("(" + this.getId() + ") Received measure: %s",
                    measure.toString()));
                logger.trace("Updating Knowledge Property " + knowledgePropertyId +
                            " with value " + measure);
                kp.setValue(theMeasure);
            }


        } catch (Exception e) {
            return this;
        }

        return this;
    }

}
```

Si te fijas, como te comentaba, un Monitor accede al (Módulo de)º Knowledge a través del Módulo de Monitorización al que está conectado.

```
IKnowledge knowledge =
    this.getTheMonitoringModule().getTheKnowledgeModule().getTheKnowledge();
```