

Refactorización bucle MAPE-K como microservicios

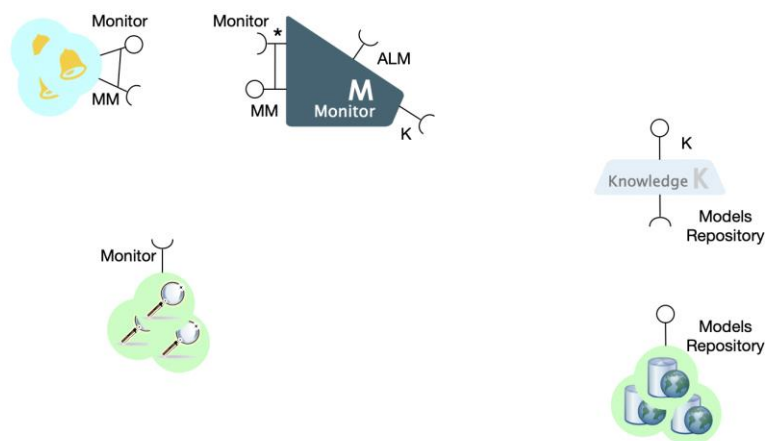
Identificación de APIs

Definición de estrategias de comunicación

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",
            Identifiable.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)

- arm.embalpack.monitors.connectors.mqtt
 - MQTTInterfaceConnector4Monitor.java
 - MQTTInterfaceConnector4MonitorMessageDispatcher.java
 - arm.embalpack.monitors.connectors.rest
 - RESTInterfaceConnector4Monitor.java
 - RESTInterfaceConnector4MonitorApplication.java
 - RESTInterfaceConnector4MonitorResource.java

```

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);
    }
}

```

La implementación de un monitor:

```
package arm.embalpack.monitors;

import org.osgi.framework.BundleContext;

import arm.embalpack.datatypes.EHealthStatus;
import arm.mapek.resources.ARMMMeasure;
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 {
            ARMMMeasure theMeasure = (ARMMMeasure)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 ( ((ARMMMeasure)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;
        }

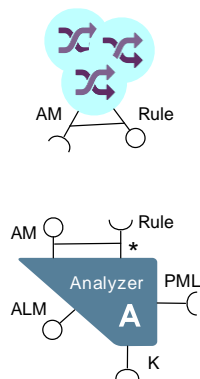
        return this;
    }
}
```


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

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

Módulo de Análisis

Existen 2 componentes principales:



- Analyzing Module: es el módulo del bucle MAPE-K que se encarga de gestionar la parte 'proceso' de la ejecución de las reglas de adaptación. Suele haber 1 por implementación, e igual que el de monitorización, también interactúa con el módulo de Knowledge.
- Rule Components (): este componente (AdaptationRule) se usa para crear nuevas reglas de adaptación de la solución a desarrollar. Estas reglas se dispararán cuando ocurran ciertos cambios sobre propiedades de adaptación (AdaptationProperties) que se reflejan en el Knowledge, sobre los que están interesados (y a los que se suscriben). Cuando se ejecuta la regla, en su cuerpo propone los cambios (adaptación) que debe ocurrir sobre el sistema.

Durante este proceso, los componentes se inter-relacionan de la siguiente manera:

- Analyzing Module y Knowledge: igual que entre el Monitoring Module y el Knowledge (del módulo anterior)
- Adaptation Rule y Analyzing Module : las reglas de adaptación se vinculan a un Módulo de Análisis. Al configurarlas ...

```
IAadaptationRule theMsHealthStatusAdaptationRule =  
    new MicroserviceHealthStatusProductionLineAdaptationRule(bundleContext, 1));  
  
IAaptiveReadyComponent theMsHealthStatusAdaptationRuleARC =  
    new AdaptationRuleARC(bundleContext, theMsHealthStatusAdaptationRule);  
theMsHealthStatusAdaptationRuleARC.start();  
  
theMsHealthStatusAdaptationRuleARC.bindService(  
    AdaptationRuleARC.REQUIRE_ANALYZINGMODULESERVICE,  
    theAnalyzingModuleARC.getServiceSupply(AnalyzingModuleARC.SUPPLY_ANALYZINGMODULESERVICE));  
theMsHealthStatusAdaptationRuleARC.deploy();
```

... primero se crea la regla (MicroserviceHealthStatusProductionLineAdaptationRule extendiende AdaptationRule) ...

... luego se configura un AdaptiveReadyComponent (ARC) para poder configurarla dinámicamente (y cambiarla en tiempo de ejecución con el bucle MAPE-K), y se inicia el ARC ...

... luego se conecta (bindservice) su REQUIRE_ANALYZINGMODULE (interfaz 'Rule' en la figura) con la interfaz que ofrece el ANALYZING_MODULE. A través de esta inferfaz se comunicarán ambos. La interfaz IAnalyzingModule es la siguiente ...

```
public interface IAnalyzingModule extends ILoopModule {

    public IAdaptationReport startingAdaptationProcessRequest();
    public IAnalyzingModule requestPlanning(String correlationId, IRuleSystemConfiguration configuration);

    public IAnalyzingModule setThePlanningModule(IPlannerModule theModule);
    public IPlannerModule getThePlanningModule();

    public IAnalyzingModule setTheSystemConfigurationModelOperator(ISystemConfigurationModelOperator
theProjector);
    public ISystemConfigurationModelOperator getTheSystemConfigurationModelOperator();

    public IAnalyzingModule onAdaptationProcessChange(IAdaptationReport theReport);
}
```

La parte del proceso de adaptación referente a esta parte del bucle ocurre de la siguiente manera:

1) Las reglas de adaptación, al configurarse, han indicado a qué propiedades de adaptación quieren atender cuando cambien éstas. Ejemplo:

```
public class MicroserviceHealthStatusProductionLineAdaptationRule extends AdaptationRule {

    protected int number = 0;
    protected String kpDevPL = null;
    protected String kpDevTL = null;
    public MicroserviceHealthStatusProductionLineAdaptationRule
        (BundleContext context, int number) {

        super(context, "id-" + number + "-adaptationrule");
        this.number = number;
        this.kpDevPL = "DevPL" + number + "_HealthStatus";
        this.kpDevTL = "DevTL" + number + "_HealthStatus";
        this.setListenToKnowledgePropertyChanges(kpDevPL);
        this.setListenToKnowledgePropertyChanges(kpDevTL);
    }

    @Override
    public boolean checkAffectedByChange(IKnowledgeProperty property) {
        ...
    }

    @Override
    public IRuleSystemConfiguration onExecute(IKnowledgeProperty property)
        throws RuleException {
        ...
    }
}
```

```
}
```

La operación `onExecute` la invocará el `AdaptationRule` component cuando requiera que se calculen los cambios que provocará la regla.

2) Ahora el sistema, durante su funcionamiento, provoca que alguna probe reporte 'algo', que envíe a su Monitor, y que éstos decidan cambios sobre propiedades de adaptación (en el knowledge)

3) El knowledge genera un evento (en la implementación actual, se produce un evento OSGi) a los que se suscriben las Adaptation Rules. * Aquí deberíamos pensar/establecer otro mecanimo, posiblemente a través de colas de mensajería (aunque me gustaría que el mecanismo fuera abstracto)

4) Cuando una propiedad de adaptación de interés para una regla cambia en el Knowledge, la regla de adaptación lo 'caza':

```
public abstract class AdaptationRule
    extends LoopResource
    implements IAdaptationRule, ServiceListener {
```

```
...
```

```
@Override
```

```
public void serviceChanged(ServiceEvent event) {
```

```
    IKnowledgeProperty property =
    (IKnowledgeProperty) this.getBundleContext().getService(event.getServiceReference());
```

```
    if ( !this.checkAffectedByChange(property) )
        return;
```

```
    IRuleSystemConfiguration theNextConfiguration = null;
```

```
    IAnalyzingModule theAnalyzingModule = this.getTheAnalyzingModule();
    IAdaptationReport theReport = theAnalyzingModule.startingAdaptationProcessRequest();
    theReport.reportSymptom(new Symptom(property.getId(), property.getValue()));
```

```
    try {
        logger.trace(String.format("<Rule %s> Asking next System Configuration", this.getId()));
        theNextConfiguration = this.onExecute(property);
    } catch (RuleNotAffectedException ex) {
        theAnalyzingModule.abortAdaptation(theReport.getCorrelationId(), "Nothing to do!");
        return;
    } catch (RuleException ex) {
        // Nothing to do ...
        theAnalyzingModule.abortAdaptation(theReport.getCorrelationId(), ex.getCausa());
        return;
    }
}
```

```
if ( theAnalyzingModule != null ) {
    logger.trace(String.format("<Rule %s> Request Planning (%s)", this.getId(), theReport.getCorrelationId()));
    theAnalyzingModule.requestPlanning(theReport.getCorrelationId(), theNextConfiguration);
}
```



```
}
```

Aquí existen 4 métodos importantes que la regla (como parte del proceso) controla:

- `startingAdaptationProcessRequest`: la regla indica al módulo de adaptación que el desea iniciar un proceso de adaptación
- `onExecute`: la regla solicita que se calcule el cambio concreto por la implementación específica de esta regla (en el ejemplo que seguimos, a `MicroserviceHealthStatusProductionLineAdaptationRule`)
- `abortAdaptation`: si se genera una excepción al ejecutar la regla específica (puede que porque finalmente no corresponda realizar la adaptación), se le indica al módulo de análisis que debe abortar la adaptación que se ha solicitado anteriormente (`startingAdaptationProcessRequest`).
- `requestPlanning`: finalmente, si todo ha ido bien, la regla de adaptación específica ha calculado el nuevo modelo, y le solicita al módulo de análisis que puede enviar a planificación la los cambios que ha calculado

A continuación se muestra una versión simplificada del método `onExecute`:

```
public class MicroserviceHealthStatusProductionLineAdaptationRule extends AdaptationRule {  
  
    ...  
  
    @Override  
    public IRuleSystemConfiguration onExecute(IKnowledgeProperty property) throws RuleException {  
  
        IRuleMicroservicesSystemConfiguration theNextSystemConfiguration = RuleMicroservicesSystemConfiguration.build(this.getId() + "_" +  
ITimeStamped.getCurrentTimeStamp(), REFERENCE_MODEL);  
  
        IMicroservicesSystemConfiguration theKnowledge =  
            this.getTheAnalyzingModule().getTheKnowledgeModule().getTheKnowledge();  
  
        IMicroservicesSystemConfiguration theCurrentSystemConfiguration = (IMicroservicesSystemConfiguration)  
theKnowledge.getCurrentSystemConfiguration();  
  
        try {  
  
            IKnowledgeProperty DevPL_HealthStatus_Property = theKnowledge.getKnowledgeProperty(this.kpDevPL);  
            IKnowledgeProperty DevTL_HealthStatus_Property = theKnowledge.getKnowledgeProperty(this.kpDevTL);  
  
            ...  
  
            // Añadimos a theNextSystemConfiguration lo que la regla considere oportuno,  
            // o se lanza una RuleException si no procede  
  
        } catch (Exception e) {  
            logger.trace("Abortamos ejecución de la regla ...");  
            throw new RuleException();  
        }  
  
        return theNextSystemConfiguration;  
    }  
}
```

5) Finalmente, si la regla devuelve la 'next system configuration' (cómo debe quedar el sistema tras la adaptación), la AdaptationRule ejecuta el método 'requestPlanning', que en resumen:

1) crea un 'job'

```
Thread t = new Thread(new AnalyzingJob(theNextRequest, this));
```

2) ejecuta el 'job'

```
t.start();
```

que lo que hace es (te lo muestro para que veas otras invocaciones al API del knowledge)

```
@Override
```

```
public void run() {
```

```
String correlationId = theNextAdaptationRequestToExecute.getCorrelationId();
```

```
IRuleSystemConfiguration theAdaptationRulesSystemConfiguration =
```

```
theNextAdaptationRequestToExecute.theAdaptationRulesSystemConfiguration;
```

```
logger.info(String.format(" >>> ## STARTING NEW ADAPTATION PROCESS (%s) <<<", correlationId));
```

```
IKnowledgeModule theKnowledgeModule = theAnalyzingModule.getTheKnowledgeModule();
```

```
IKnowledge theKnowledge = theKnowledgeModule.getTheKnowledge();
```

```
IAdaptationReport theReport = theKnowledge.getAdaptationReport(correlationId);
```

```
theReport.reportStatus(EAdaptationProcessStatus.IN_PROGRESS, null);
```

```
theKnowledge.saveAdaptationReport(theReport);
```

```
ISystemConfiguration theCurrentSystemConfiguration =
```

```
theKnowledge.getCurrentSystemConfiguration();
```

```
ISystemConfiguration theNextSystemConfiguration = ...;
```

```
theKnowledge.setFuturibleSystemConfiguration(theNextSystemConfiguration);
```

```
if ( theReport != null ) {
```

```
    theReport.reportTheCurrentSystemConfiguration(theCurrentSystemConfiguration);
```

```
    theReport.reportTheNextSystemConfiguration(theNextSystemConfiguration);
```

```
    theReport.reportTheAdaptationRulesProposal(theAdaptationRulesSystemConfiguration);
```

```
    theKnowledge.saveAdaptationReport(theReport);
```

```
}
```

```
IPlannerModule planningModule = theAnalyzingModule.getThePlanningModule();
```

```
if ( planningModule != null ) {
```

```
    planningModule.plan(correlationId, theAdaptationRulesSystemConfiguration);
```

```
}
```

```
return;
```

```
}
```

La operación importante, según el proceso, es la rellamada a la planificación de la configuración del sistema que ha definido la regla de adaptación.

Por el camino ha anotado (en el knowledge, a través del concepto de AdaptationReport) cuál es el estado de la adaptación (en progreso), cuál es la configuración actual, la siguiente y qué regla ha disparado el cambio.