

mOSAIC

Open Source API and Platform for Multiple Clouds

Georgiana Macariu
eAustria Research Institute
Timisoara, Romania

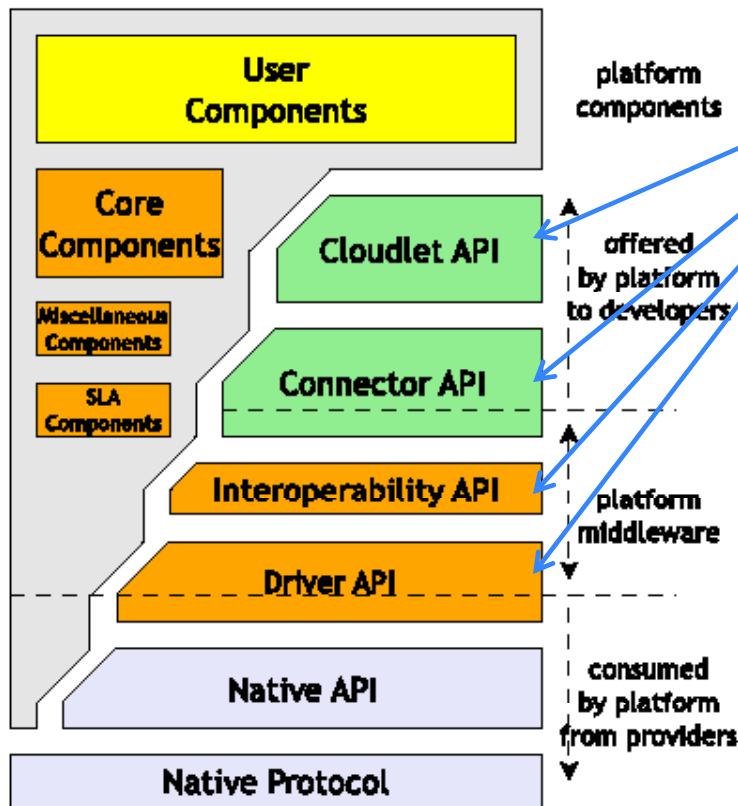


mOSAIC API and Platform

- Targets mainly Web 2.0 applications, but can be used for others, too
- Outcomes
 - API for at least two programming languages
 - Application tools
 - Provisioning platform
 - Application management platform



API layers



mOSAIC API

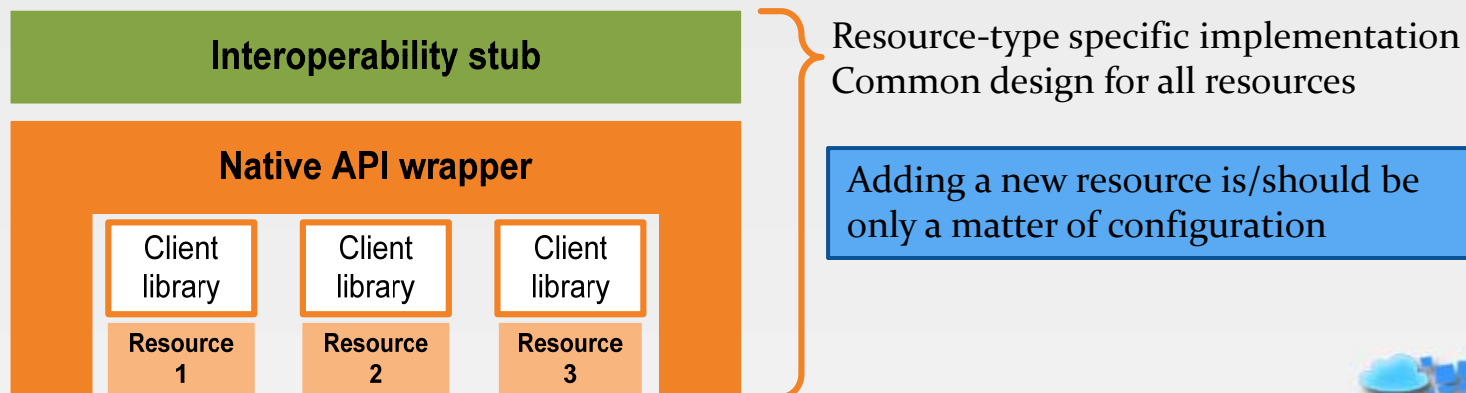


Now:
**Java implementation
of the API**



The Driver API

- We have drivers for
 - Key-value store
 - AMQP-based queuing systems
- Driver implementation based on external client libraries
- Drivers completely implemented by us:
 - Amazon S3



The Interoperability API

- Ensures communication between the Driver Layer and the Connector Layer
 - Stubs on Driver side
 - Proxies on Connector side
- Based on ZeroMQ and Google Protocol Buffers



The Connector API

- We have connectors for
 - Key-value stores
 - AMQP-based queuing systems
- One connector for each resource type and programming language
- Upon initialization the connector looks for a driver using the mOSAIC platform

Per resource Connector structure

Connector

Interoperability proxy

When adding a new resource type a new proxy and Connector must be implemented...
... but they need to implement only resource specific methods



The Cloudlet API

- Cloudlets \approx Servlets
- How to develop your mOSAIC application?
 - Identify application components, a.k.a. identify your cloudlets
 - Implement each cloudlet
 - Event-based programming model
 - A cloudlet = a collection of callback classes: Life cycle callback and one callback per resource used
- Cloudlets execute within a container which is a mOSAIC Component
- We have a container per cloudlet type
 - in a container can be many instances of the same cloudlet
 - Cloudlet instances are independent
- Cloudlet state
 - Only references to resource connectors and constants



Callbacks: life cycle

```
public static final class LifecycleHandler extends
    DefaultCloudletCallback<CloudletState> {

    public void initialize(CloudletState state,
        CallbackArguments<CloudletState> arguments) {
        // do init stuff, e.g. configure cloudlet, create resource
        connectors
    }

    public void initializeSucceeded(CloudletState state,
        CallbackArguments<CloudletState> arguments) {
        ...
    }

    public void destroy(CloudletState state,
        CallbackArguments<CloudletState> arguments) {
        ...
    }

    public void destroySucceeded(CloudletState state,
        CallbackArguments<CloudletState> arguments) {
        ..
    }
}
```



Callbacks: key-value store

```
public static final class KeyValueCallback extends
DefaultKeyValueAccessorCallback<CloudletState> {

    public void initializeSucceeded(CloudletState state,
        CallbackArguments<CloudletState> arguments) {
        ...
    }

    public void destroySucceeded(CloudletState state,
        CallbackArguments<CloudletState> arguments) {
        state.kvStore = null;
        // do other destroy stuff
    }

    public void setSucceeded(CloudletState state,
        KeyValueCallbackArguments<CloudletState> arguments) {
        ...
    }
}
```



Callbacks: queue consumer

```
public static final class AmqpConsumerCallback extends
DefaultAmqpConsumerCallback<CloudletState, LoggingData> {
    public void registerSucceeded(CloudletState state,
        CallbackArguments<CloudletState> arguments) {
        ...
    }
    public void unregisterSucceeded(CloudletState state,
        CallbackArguments<CloudletState> arguments) {
        ...
    }
    public void acknowledgeSucceeded(CloudletState state,
        CallbackArguments<CloudletState> arguments) {
        ...
    }
    public void consume(CloudletState state,
        AmqpQueueConsumeCallbackArguments<CloudletState, LoggingData> arguments) {
        AmqpQueueConsumeMessage<LoggingData> message =
            arguments.getMessage();
        // process message
        message.acknowledge();
    }
    ...
}
```



Callbacks: queue publisher

```
public static final class AmqpPublisherCallback
extends
DefaultAmqpPublisherCallback<CloudletState, AuthenticationToken> {

    public void registerSucceeded(CloudletState state,
    CallbackArguments<CloudletState> arguments) {
        ...
    }
    public void unregisterSucceeded(CloudletState state,
    CallbackArguments<CloudletState> arguments) {
        // if unregistered as publisher is successful then destroy resource
        ICloudletController<LoggingCloudletState> cloudlet = arguments
        .getCloudlet();
        cloudlet.destroyResource(state.publisher, this);
    }

    public void publishSucceeded(CloudletState state,
    AmqpQueuePublishCallbackArguments<CloudletState, AuthenticationToken>
    arguments) {
        state.publisher.unregister();
    }
}
```



The workflow (1)

- Start your virtual machines
- Start mOSAIC controller on each machine
 - `/opt/mosaic-node-boot/bin/run`
- Access the web interface
 - e.g.: `http:// i-48C406E5-
pub.vms.mosaic.ieat.ro:31810/`



The workflow (2)

- Start the resources
 - RabbitMQ server (queuing system)
 - Riak (key-value system)
- Start the drivers for resources
- Start the cloudlet



www.mosaic-cloud.eu

Code available at:

<http://bitbucket.org/mosaic/mosaic-java-platform>

