Ecosystem infrastructure for smart and personalised inclusion
and PROSPERITY for ALL stakeholders

ARE REST API

Project Acronym	**Prosperity4All**
Grant Agreement number	**FP7-610510**
Deliverable number	**Work conducted for D203.1**
Work package number	**WP203**
Work package title	**Collaborative development tools/Environments**
	T203.3 Runtime Environment
Authors	**Marios Komodromos, Christos Mettouris**
Status	**Final**
Dissemination Level	**Public/Consortium**
Number of Pages	**13**

Table of Contents

List of Tables

List of Figures

No table of figures entries found.

Executive Summary

This document describes the usage of the ARE REST API developed by UCY in the context of Task 203.3 of WP203, Prosperity4All project.

1 REST API

To allow remote communication with the AsTeRICS Runtime Environment, the ARE REST API was developed. It allows manipulation of resources through a set of HTTP methods such as GET, POST, PUT and DELETE.

Apart from the regular REST functions, an event mechanism is provided (SSE). With this mechanism, ARE can broadcast messages to anyone who subscribes and inform when an event occurs.

The API uses HTTP status codes to declare an error in a call. Specifically, when an error occurs, the response will contain a 500 HTTP status code (Internal Server Error) with an ARE-produced error message inside the HTTP response body.

The table in the next page (table 1) describes these methods and provides the necessary information in order to call them.

1.1 REST API Functions

HTTP Method	Resource	Parameters	Consumes	Produces	Description
GET	/runtime/model	-	-	XML	Retrieves the currently deployed model in XML
PUT	/runtime/model	modelInXML (in body)	XML	TEXT	Deploys the model given as a parameter
PUT	/runtime/model/{filename}	filename	-	TEXT	Deploys the model contained in the given filename
PUT	/runtime/model/state/{state}	state	-	TEXT	Changes the state of the deployed model to STARTED, PAUSED, STOPPED
GET	/runtime/model/state	-	-	TEXT	Returns the state of the deployed model
PUT	/runtime/model/autorun/ {filename}	filename	-	TEXT	Deploys and starts the model in the given filename
GET	/runtime/model/components/ids	-	-	JSON	Retrieves all the component Ids contained in the currently deployed model
GET	/runtime/model/components/ {componentId}	componentId	-	JSON	Returns all property keys of the component with the given componentId in the currently deployed model
GET	/runtime/model/components/ {componentId}/{componentKey}	componentId, componentKey	-	TEXT	Retrieves property value of a specific component, in the currently deployed model

PUT	/runtime/model/components/{componentId}/{componentKey}	componentId, componentKey, value (in body)	TEXT	TEXT	Changes a property value of a specific component, in the currently deployed model
GET	/storage/models/{filename}	filename	-	XML	Returns an xml representation of a model in a specific file
POST	/storage/models/{filename}	filename, modelInXML (in body)	XML	TEXT	Stores a model in the given filename
DELETE	/storage/models/{filename}	filename	-	TEXT	Deletes the model with the given filename
GET	/storage/models/names	-	-	JSON	Retrieves the model names that are saved in the ARE repository
GET	/storage/components/descriptors/xml	-	-	XML	Returns an xml string containing the descriptors of the created components with some modifications in order to be used by the webACS
GET	/storage/components/descriptors/json	-	-	JSON	Retrieves the exact content of the component descriptors contained in the ARE repository
GET	/restfunctions	-	-	JSON	Returns a list with all the available rest functions
GET	/events/subscribe	-	-	-	Opens a persistent connection with ARE and listens for Server Sent Events.

Table 1: REST API functions

1.2 Path parameter encoding

As seen in table 1, there are some functions that expect parameters in the URI, the **path parameters**. It can be observed that the path parameters are part of the URI and are wrapped with curly brackets (for example, the “filename” in “/runtime/model/{filename}”). Caution: do not confuse **query parameters** with path parameters.

Before the function call, these parameters should be encoded based on the UTF-16 encoding table. Every character of the parameter should be replaced with the corresponding **decimal value** of the UTF-16 table, and every encoded character should be separated from the other characters with an un-encoded dash (“-”) character.

For a better understanding check the example below:

Rest call:

PUT	/runtime/model/{filename}	filename	-	TEXT	Deploys the model contained in the given filename

Un-encoded URI: http://localhost:8081/runtime/model/foobar

Encoded URI: http://localhost:8081/runtime/model/*102-111-111-98-97-114*

1.3 Event Types

As said before, the API allows subscription to specific ARE event types. To consume SSE events, the client must be able to achieve communication using SSE technology. SSE library implementations are available for almost any well-known programming language (java, C#, JavaScript etc).

A subscription can be conducted per event type. Each event received by a client, will contain a message from a pre-defined set. The client should check this message in order to identify which specific event has occurred. The event types and their corresponding messages can be found in table 2. More information for each event message can be found in AsTeRICS Developer manual, section 5.10.

Note that the event type parameter, is passed as a part of the SSE mechanism and not as part of the corresponding REST API function.

event type	event messages	Description
model_state_changed	pre_start_event post_start_event pre_stop_event post_stop_event pre_pause_event post_pause_event pre_resume_event post_resume_event	Notifies the subscribers that an event occurred, which has affected or is going to affect the runtime model state
model_changed	pre_deploy_event post_deploy_event	Notifies the subscribers that a new model was deployed or is going to be deployed
model_event *		Notifies the subscribers that an ARE runtime model event was triggered

* NOT YET IMPLEMENTED

Table 2: Event Types

2 REST API Client libraries

To enable easier REST API accessibility, communication libraries were created that simplify the whole procedure.

2.1 JavaScript Client library

To install the JavaScript library in a webpage these steps have to be followed:

1) Import the 'AREcommunicator.js' file in the html page.

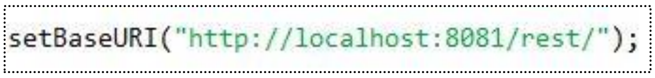
2) Import 'JSmap.js' file in the html page.

3) Import a script that provides jQuery functionality.

   (i.e. "http://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js")

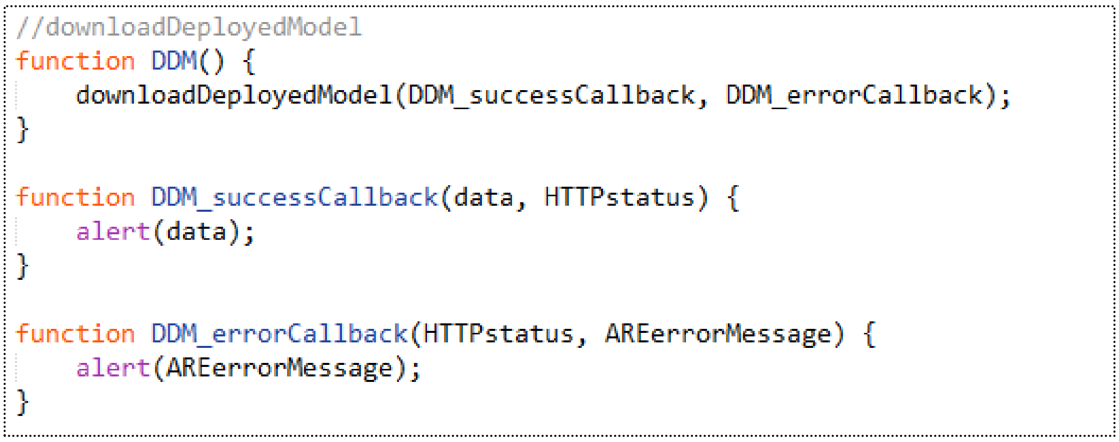
(For testing purposes, a simple implementation of a JavaScript client was created and it can be found here: https://github.com/asterics/AsTeRICS/tree/master/bin/ARE/data/webservice/api/clientExample/client.html)

Before calling ARE functions, the baseURI has to be set. This is the URI where ARE runs at. For example:



```
setBaseURI("http://localhost:8081/rest/");
```

To call any REST function, we have to provide two callback functions: a successCallback and an errorCallback such as the example below



```
//downloadDeployedModel
function DDM() {
    downloadDeployedModel(DDM_successCallback, DDM_errorCallback);
}

function DDM_successCallback(data, HTTPstatus) {
    alert(data);
}

function DDM_errorCallback(HTTPstatus, AREerrorMessage) {
    alert(AREerrorMessage);
}
```

Furthermore, the 'subscribe' function is opening a persistent connection with ARE. Using an event mechanism based on Server Sent Events (SSE) specifications, it listens to the connection for broadcasted messages. Additionally, the event type (Table 2) name must be provided, to specify what type of events to listen for. The concept still remains the same as a successCallback function and an errorCallback function must be provided. The unsubscribe function does not use any rest calls since it closes the connection from the browser's side.

In the next page, Table 3 describes each method provided by the library.

JavaScript Library Functions

Function Signature	Description
downloadDeployedModel(**sCB1**, eCB)	Retrieves the currently deployed model in XML
uploadModel(**sCB1**, eCB, modelinXML)	Deploys the model given as a parameter
deployModelFromFile(**sCB1**, eCB, filename)	Deploys the model contained in the given filename

startModel(**sCB1**, eCB)
stopModel(**sCB1**, eCB)
pauseMolel(**sCB1**, eCB)
	Changes the state of the deployed model to STARTED, PAUSED, STOPPED

getModelState(**sCB1**, eCB)	Returns the state of the deployed model
autorun(**CB1**, eCB, filename)	Deploys and starts the model in the given filename
getRuntimeComponentIds(**sCB1**, eCB)	Retrieves all the component ids contained in the currently deployed model (as JSON array)
getRuntimeComponentPropertyKeys(**sCB2**, eCB, componentId)	Returns all property keys of the component with the given componentId in the currently deployed model (as JSON array)
getRuntimeComponentProperty(**sCB1**, eCB, componentId, componentKey)	Retrieves property value of a specific component, in the currently deployed model
setRuntimeComponentProperty(**sCB1**, eCB, componentId, componentKey, value)	Changes a property value of a specific component, in the currently deployed model
downloadModelFromFile(**sCB1**, eCB, filename)	Returns an xml representation of a model in a specific file
storeModel(**sCB1**, eCB, filename, modelinXML)	Stores a model in the given filename
deleteModelFromFile(**sCB1**, eCB, filename)	Deletes the model with the given filename
listStoredModels(**sCB2**, eCB)	Retrieves the model names that are saved in the ARE repository (as JSON array)
getComponentDescriptorsAsXml(**sCB2**, eCB)	Returns an xml string containing the descriptors of the created components with some modifications in order to be used by the webACS
getComponentDescriptorsAsJSON(**sCB2**, eCB) **	Retrieves the exact content of the component descriptors contained in the ARE repository (as JSON array)
getRestFunctions(**sCB2**, eCB) ***	Retrieves the information for all the available rest functions provided by the Restful API (as JSON array with Function objects)
subscribe(**sCB1**, eCB, eventType)	Opens a persistent connection with ARE and listens for Server Sent Events.
unsubscribe(eventType)	Closes the connection for Server Sent Events. Returns true if the unsubscription

	was successful and false otherwise

Table 3: JavaScript Client Functions

sCB1: successCallback(textData, HTTPstatus)

sCB2: successCallback(array, HTTPstatus)

eCB: errorCallback(HTTPstatus, AREerrorMessage)

**: Component object (see JSON objects section)

***: Function object (see JSON objects section)

JSON OBJECTS

Object Name	Example
Function	{ "path": "/runtime/model", "description": "Retrieves the currently deployed model in XML", "httpRequestType": "GET", "bodyParameter": "", "consumes": "", "produces": "text/xml" }
Component	{ "canonicalName":"eu.asterics.component.processor….", "type":"PROCESSOR", "id":"asterics.StringDispatcher", "description":"Send text from chosen slot", "singleton":false, "inputPorts":[{ "type":"INPUT", "multiplicity":null, "description":"Send the string from the slot defined by the incoming value", "portID":"slotDispatch", "dataType":"INTEGER", "propertyNames":null }], "outputPorts":[{ "type":"OUTPUT", "description":"Output text", "portID":"output", "dataType":"STRING", "propertyNames":null }], "eventTriggererPorts":[ ], "ports":[{ "type":"INPUT", "multiplicity":null, "description":"Send the string from the slot defined by the incoming value",

```
            "portID":"slotDispatch",
            "dataType":"INTEGER",
            "propertyNames":null
         },
         {
            "type":"OUTPUT",
            "description":"Output text",
            "portID":"output",
            "dataType":"STRING",
            "propertyNames":null
         }
      ],
      "eventPorts":[
         {
            "id":"dispatchSlot1",
            "description":"Send text from slot 1"
         }
      ],
      "eventListenerPorts":[
         {
            "id":"dispatchSlot1",
            "description":"Send text from slot 1"
         }
      ],
      "propertyNames":[
         "delay",
         "slot1"
      ]
   }
```

Table 4: JSON objects

2.2 Java Client library

Environment specs:

1) **Recommended IDE:** eclipse
2) **Recommended Java version**: 7

To import, test or modify the Java library in an IDE, follow these steps:

1) Create a simple java project in your IDE.
2) Navigate to the destination where the Java library is located and copy the 'lib' and 'models' folders to the root of your project.
3) Copy the contents of ‘src’ folder to the ‘src’ folder of your project.
4) Add all the jar files which are located inside 'lib' folder to your project build path.
5) Run 'JavaClient.java' class located inside the ‘tester’ package to test that everything works as expected.

To use the Java library in our own project, follow these steps:

1) Add ‘ARECommunicator.jar’ file to the build path of our project.
2) Add the jar files contained in the ‘lib’ folder to the build path of our project.

When installation is completed, the procedure of communicating with ARE is reduced to plain calls of Java methods of an object.

As with JavaScript library, the baseURI has to be set:

```
ARECommunicator areCommunicator = new ARECommunicator("http://localhost:8081/rest/");
```

and when this is done, you are able to call any method you desire:

```
areCommunicator.startModel();
```

Furthermore, the ‘subscribe’ function is opening a persistent connection with the ARE. Using an event mechanism based on Server Sent Events (SSE) specifications, it listens to the connection for broadcasted messages. Additionally, the eventType name must be provided, to specify what type of events to listen for. To achieve this functionality, the Jersey SSE java library was used.

In the next page, Table 5 describes each method provided by the library.

Java Library Methods

Function Signature	Description
String downloadDeployedModel()	Retrieves the currently deployed model in XML
String uploadModel(**String** modelinXML)	Deploys the model given as a parameter
String deployModelFromFile(**String** filename)	Deploys the model contained in the given filename
String startModel() **String** stopModel() **String** pauseModel()	Changes the state of the deployed model to STARTED, PAUSED, STOPPED
String getModelState()	Retrieves the state of the deployed model
String autorun(**String** filename)	Deploys and starts the model in the given filename
String[] getRuntimeComponentIds()	Retrieves all the components contained in the currently deployed model
String[] getRuntimeComponentPropertyKeys(**String** componentId)	Retrieves all property keys of the component with the given componentId in the currently deployed model
String getRuntimeComponentProperty(**String** componentId, **String** componentKey)	Retrieves property value of a specific component, in the currently deployed model
String setRuntimeComponentProperty(**String** componentId, **String** componentKey, **String** value)	Changes a property value of a specific component, in the currently deployed model
String downloadModelFromFile(**String** filename)	Retrieves an xml representation of a model in a specific file
String storeModel(**String** filename**, String** modelinXML)	Stores a model in the given filename
String deleteModelFromFile(**String** filename)	Deletes the model with the given filename
String[] listStoredModels()	Retrieves a list with all the model that are saved in the ARE repository
String getComponentDescriptorsAsXml()	Returns an xml string containing the descriptors of the created components with some modifications in order to be used by the webACS
List<String> getComponentDescriptorsAsJSON()	Retrieves the exact content of the component descriptors contained in the ARE repository (as JSON array)
ArrayList<RestFunction> functions()	Retrieves a list with all the available rest functions
subscribe(**String** eventType)	Subscribes the IP that sent the request to the event mechanism
unsubscribe(**String** eventType)	Unsubscribes the IP that sent the request to the event mechanism

Table 5: Java Client Functions