DELIVERABLE

Project Acronym: F4W

Grant Agreement nbr: 000000

Project Title: FACTS4WORKERS

RESTdesc – Status update 10/2015

Revision:

Authors:

Joachim Van Herwegen (iMinds – UGent – Multimedia Lab)

Dörthe Arndt (iMinds – UGent – Multimedia Lab)

REVISION HISTORY

Revision	Date	Author	Organisation	Description

Table of Contents

1 Introduction

This document describes the current status of the iMinds RESTdesc framework as of October 2015. This includes specific changes since the previous status update and a general description of how to use the system. Afterwards we will go more deeply into the future work that is still necessary.

2 RESTdesc framework

In this chapter we will describe how to use the interface provided by the framework to discover which path to take.

2.1 Initial call

The initial call should be a POST to f4w.restdesc.org/next with the following JSON body:

```
{  "goal": "calibration"  }
```

The “goal” field indicates which use case needs to be solved; in this case “calibration” corresponds to the initial Hidria use case. Currently, if no goal value is provided, the value “calibration” will be used (to stay compatible with the previous version we provided).

The available goals at the moment are:

- calibration
- thermolympics_operator
- thermolympics_teamleader

2.2 Framework response

After the first call (for the calibration use case) the server will return the following response:

```
{
  "http:methodName": "GET",
  "http:requestURI": "start",
  "http:body": {
    "message": "Please input the starting information. 'id'
                corresponds to the part ID.",
    "partIDList": [ 100, 123 ] },
  "http:resp": {"http:body": {"contains": {"id": "_:sk99_1"}}},
  "http:headers": [{
      "http:fieldName": "Content-Type",
      "http:fieldValue": "application/json" }],
  "data": "c614ee06-6400-4a08-9698-509518b401a5",
  "output": ...,
```

```
  "proofs": [ ... ]
}
```

http:methodname will always be GET (we interpret the worker actions as GET APIs)

http:requestURI corresponds to the action we expect worker to take. These should change to more descriptive names but have stayed the same so far for backwards compatibility.

http:body provides the information that might be necessary for the HMI. In this case it contains a message of what is expected and a list of part IDs.

http:resp describes what we expect from the HMI. In this case we expect the 'body' to at least contain an "id" field. The value "_:sk99_1" in this case simply corresponds with the internal variable we use until we have an actual value.

http:headers indicates that we expect JSON. This value will probably always be the same and can be ignored.

data is the database key we use to store the intermediate step values that are necessary to progress in the workflow. This value is necessary to keep the framework stateless.

output and **proofs** are used by our own testing interface and can safely be ignored.

2.3 Follow-up calls

All the next calls to the RESTdesc API should always contain the response previously received (see previous step 2.2). The entire JSON object the HMI received from the server needs to be put in a nested JSON object with the key "eye". The actual response values should be in a nested object indicated with the key "json". In this case the HMI should send the following JSON object to /next:

```
{
  "goal": "calibration",
  "json": { "id": 100 },
  "eye": { "http:methodName": "GET", ... }
}
```

2.4 Finishing up

Once the workflow has finished the framework will return the following JSON object:

```
{
  "status": "DONE",
  "data": "c614ee06-6400-4a08-9698-509518b401a5",
  "output": ...,
  "proofs": [ ... ]
}
```

At this point the HMI can send this JSON object directly to /clear to clear the database entry (more on this later).

3 New features

This chapter will describe some of the new features that have been added to the framework since the previous update.

3.1 Goals

As indicated in 2.1, it is now possible to indicate what the goal will be.

3.2 Redis backend

To reduce the amount of data that gets sent back and forth, the framework now uses a Redis database to store the intermediate data necessary to continue the workflow. Currently this data gets deleted automatically after 24 hours; this can still change depending on the requirements.

3.3 /clear

To prevent the server from storing unneeded data, it is possible to indicate that intermediate steps of a workflow can be deleted. This could happen when the steps are finished or the worker decides to cancel the process. Clearing the data can be done by sending a JSON object to /clear containing the Redis key:

```
{ "data": "c614ee06-6400-4a08-9698-509518b401a5" }
```

3.4 /back

It is possible to go back to a previous step in the workflow, if the user made a mistake in a previous step for example. This is done by sending a request to /back with the Redis key (similar as /clear). The next call to /next will then be the previous step again with all the information added after that erased.

3.5 Errors

Should one of the APIs fail during the workflow, the framework will return the following response:

```
{
  "error": {
    "statusCode": 400,
    "error": "theError",
    "body": "theErrorBody"
  }
  ...
```

```
}
```

3.6 Rule changes

The N3 rules to describe the APIs have changed at some points. If you are interested you can find the actual rules at /n3.

4 Future work

This chapter contains some of the changes we still need to do for the framework in this project.

4.1 Error handling

Currently the server simply returns a message if one of the APIs produced an error. The RESTdesc system has been designed in such a way that these error-producing APIs can easily be removed so the system can find another path through the APIs. This still needs to be implemented. There is also still the open question of what to do exactly with these error-producing APIs. This might depend on what the other systems expect.

- Should an API immediately be removed if there is an error or should there be multiple retries?
- When removing such an API, should it be removed for all requests during a period of time? Should someone be alerted of this?
- Should we ask the user for new input?
- etc.

4.2 Special cases

There are multiple ways to handle non-standard APIs and workflows, such as APIs returning multiple possible formats of output, workflows splitting and looping etc. For our framework we have to define a general approach for these situations so they don't get solved differently every time.

4.3 N3 rules

Currently it's still quite hard for someone that is less knowledgeable about the system to actually implement API rules and descriptions. This needs to be solved with better documentation or even by providing an interface that allows easier rule creation.