DELIVERABLE

Project Acronym: F4W

Grant Agreement nbr: 000000

Project Title: FACTS4WORKERS

RESTdesc – Status update 06/2015

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 Other documents

Besides this document we also have a document describing the demo system more in general[1] and an in-depth description of the communication protocol[2]. Both of these documents are slightly outdated but still mostly relevant.

There is also a paper completely describing the RESTdesc system[3].

2 Current state

The current Hidria demo shows that the RESTdesc system can already be used to solve use cases. The Hidria use case is also interesting since it contains a loop back to a previous state if certain conditions are not fulfilled (i.e. the machine is not calibrated correctly). Simple linear use cases should be no problem for the current implementation and more complex use cases will have to be studied on a case by case basis; it is impossible to predict where there are still potential problems without seeing the full use cases.

3 Requirements

The system itself is quite lightweight. The server only needs a running EYE instance[4] and our RESTdesc interface in front of the reasoner, which is a Node.js application. In the future a database might be added to store the session information, but this will again be a lightweight database.

To actually communicate with RESTdesc, a network connection to the server is required at the time the request is executed. The worker does not have to be connected to the network when he is doing work for the next step of the workflow, the connection is only necessary at the time he sends his work results to get the next step from the system.

4 Current/future work

We are currently working on implementing the second use case (Thermolympic) and will update the demo site when this is finished. The result of this is closely tied to our next point here.

We are focussing on making the system more robust. One of the important parts is converting the JSON to and from the N3 format that is used by the EYE reasoner. There is currently no system that does this automatically so we have to do research on a robust solution that can handle all possible

[1] http://f4w.restdesc.org/demo/documentation/documentation.pdf
[2] http://f4w.restdesc.org/demo/documentation/dataflow.pdf
[3] http://ruben.verborgh.org/publications/verborgh_mtap_2013/
[4] http://eulersharp.sourceforge.net/

cases here. A format closely related to these two is JSON-LD, which is similar to N3 in the data it can represent, but is stored in the JSON format. We are looking into the viability of doing a N3 <-> JSON-LD <-> JSON conversion which would ease this process a lot.

Beside the data conversion part we also want to improve the error handling part. It is planned to enable the system to deal with unavailable APIs or incomplete user input. If it detects that an API is unavailable it will either retry the API or simply search for an alternative part towards the goal, depending on the circumstances. If there is no alternative possible, the HMI should be informed with an extensive description of the problem. Currently this is already partially implemented, but the informing of the HMI isn't complete yet.

As mentioned before we will also add a database to store user session data. Currently all state information gets sent back to the user, who then has to send this back to the server for the next step. This can grow quite big depending on the use case which would slow down the system. The solution for this is to set up a database that stores this information and only sends a session ID to the user, limiting the amount of data that has to be sent in each call.

Another direction we are focussing on is the nature of the rules themselves. We want to be able to give more concrete guidelines for the user/maintainer of the system on how to write API descriptions. They have to be expressive enough to be reusable, but not that expressive that they can significantly slow down the reasoning process. We already have experience on that based on the implementation of simple use cases. The complex use cases of the project will enable us to refine our knowledge and to give more concrete recommendations.

Our main focus will depend on the special needs of the project and dependencies of other building blocks once the use cases are defined.

5 Advantages

One of the main goals of RESTdesc is to make the workflow engine completely modular. This makes the life of a developer a lot easier and prevents ugly duplication. API changes can be handled easily newly added APIs will be handled automatically. If an API needs to or could be reused for multiple use cases, no additional code has to be written. The API description will automatically be found by the reasoner, even if both the previous and next step are different for each use case. If, for example, a new use case needs to be added that can be completely solved with APIs of which we already have the description from other use cases, the only addition required is a description of the use case's goal. Using this description the reasoner will be able to find a path to the goal if the current APIs permit it.

The API descriptions can also be quite flexible in handling APIs that can return multiple types of output or output that is incomplete. The output will automatically be handled and based on the contents of that, the system can see which is the next API that can handle that specific set of data.

Another advantage of RESTdesc is how generic it is: it supports all REST APIs (currently only JSON APIs, but even this could be updated) and completely handles the calling of these APIs with the correct input. This reduces some of the complexity for the HMI, since that now only has to do a simple call to RESTdesc to ask for the next step, without having to know anything about the current state.

Because the system is so generic, error handling is also a lot more flexible. The developer doesn't have to take every possible failure into account and then tell the system what to do in those situations. RESTdesc will simply look if there is another path that can be taken and request additional information from the HMI if it is necessary for this new path.