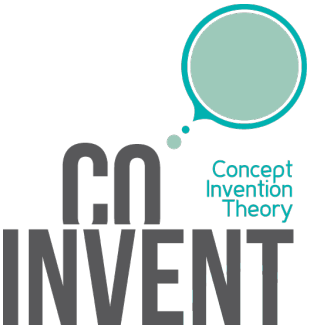

D8.5
D8.5 COINVENT system implementation and workflows

Authors	Ewen Maclean, Daniel Winterstein
Reviewers	Marco Schorlemmer

Grant agreement no.	611553
Project acronym	COINVENT - Concept Invention Theory
Date	October 1, 2013
Distribution	PU/RE/CO

Disclaimer

The project COINVENT acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under FET-Open Grant number 611553.

Abstract

This deliverable presents the implemented APIs for the Coinvent system –*Cobble*. It presents some examples and explains how access the APIs on an existing server, and how to install them locally so that the APIs can be accessed without the need for the internet. RESTful http web services have been implemented for HDTP, HETS and the amalgams system. The system allows iterative blending to take place by re-inserting computed blends as input theories. Keyword list: **specification**, **system implementation**

Executive Summary

The Coinvent system comprises of a set of RESTful interfaces for HDTP, HETS and amalgams. The system currently runs on a server and the API calls can be made remotely. Alternatively the system can be installed locally and the APIs used from the local machine. There is a simple interface to the system which can also be used as a front-end for computing blends and iteratively developing theories. This documents descibes the API specification for each component and gives examples of their use. Local installation instructions are also given.

Contents

1 API usage

Here we describe the function and specification of the three different APIs implemented for the Coinvent project. We give examples in §4

1.1 Blending with HDTP and HETS

The blend api can be installed locally or is running at the following address

`http://server.coinvent-project.eu`

or if run locally (see §2

`http://localhost:8300/cmd/blendui`

1.1.1 Post parameters

The post options are as follows:

"action" This is either

- **"amalgamscasl"** See §1.2 for a description of this part of the system
- **"amalgamsowl"** See§1.2 for a description of this part of the system
- **"hdtp"** This purely computes a generic space
- **"hets"** This computes a generic space and a blend
- **"next"** This gives the next answer for a given process id (pid)
- **"close"** This closes the given process (pid)
- **"list-concepts"** This returns a list of named concepts in the database
- **"save"** This saves a computed blend under a given name
- **"delete-all"** This deletes all saved concepts in the database

"input1" This defines an input concept. It has properties

- **"name"** If this matches an existing concept name in the database this is used
- **"url"** This is the url of an ontology - for example in ontohub
- **"text"** If the name does not match a concept in the database, then the definition can be given in this field

"input2" See input 1

"pid" This is the process id of a blend for which you want to get another answer from or close.

1.1.2 JSON response

Depending on the action requested the response is different. Each response is listed here under the action request heading. For any other action request the response carries an empty cargo and simply returns the success status of the command.

"list-concepts" This returns a cargo field which is an array of names concepts in the database.

"hdtp" The response cargo field is structured as follows:

"output" This is the output from running HDTP on the two input theories, **input1** and **input2**.

"input1" This is a copy of the input theory given by **input1**

"input2" This is a copy of the input theory given by **input2**

"theory" This is a file containing both input spaces and the morphisms and generic space calculated by HDTP.

"pid" This is a process id by which the process can be referenced; for example to obtain subsequent results from HDTP.

"hets" The response cargo field is structured as follows:

"blend" This is the blend theory output calculated by HETS as the colimit of the two input theories given by **input1** and **input2** with respect to the generic space calculated by HDTP.

"theoryurl" This is a url to the combined theory file produced by HETS

"theory" This is the whole theory given in theoryurl

"origtheoryurl" This is the url of the theory calculated and sent to HETS

"pid" This is the process if by which the process can be referenced.

1.2 Blending with Amalgams

The blend api can be installed locally or is running at the following address

`http://server.coinvent-project.eu`

or if run locally (see §2

`http://localhost:8300/cmd/blendui`

1.2.1 Post parameters

"action" This is either*amalgamscasl* or *amalgamsowl* depending on whether the input concepts are expressed in CASL or OWL syntax.

"id" This is the process id of an amalgams process.

"content" This is a file with potentially many concepts defined in CASL.

"space_name1" This is the name of a specification in the content field.

"space_name2" This is the name of a specification in the content field.

"generic_space" A generic space can be given to the system

"exists_generic_space" This is a boolean which specifies if a generic space is given or should be calculated by amalgams

"request" This can be one of

- **"start"** This starts a new amalgams process
- **"next"** This request the next answer from amalgams for a given process id
- **"close"** This closes the process with given process id

1.2.2 JSON response

The response field cargo returns the output blend from amalgams computed on the space names given in the post parameters, in a field "output".

1.3 Blending Haikus

The interface the Coinvent System incorporates the Haiku blending system whose instructions can be found here:

`http://socrash.soda.sh:8642/static/haiku/index.html`

2 Local installation

In order to run the system locally, it suffices to download the code from

`https://github.com/coinvent/coinvent/tree/master/simple-server`

and to create a runnable jar file by importing the directory as a java project in eclipse. Alternatively, downloading the jar file from

`https://github.com/coinvent/coinvent/blob/master/simple-server/jar/simple-server.jar`

and running the command

java -jar simple-server.jar

will start a local server at

`http://localhost:8300`

The address for the front end is then

`http://localhost:8300/static/blendui/index.html`

The address for the implemented API servers are then

`http://localhost:8300/cmd/blendui`

3 User Interface

A front-end for the APIs is implemented which provides a graphical user interface for the underlying functionality. The Figures shown below demonstrate the interface and explain the functionality.

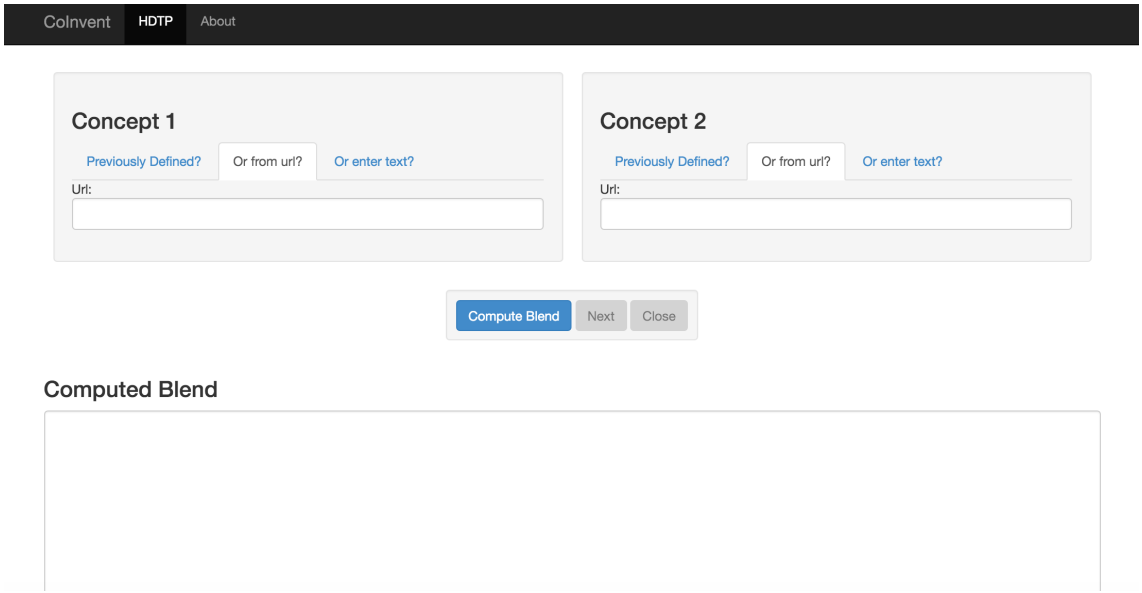

Figure 1: The Coinvent system main page, showing the two panes where the input concepts or theories can be selected

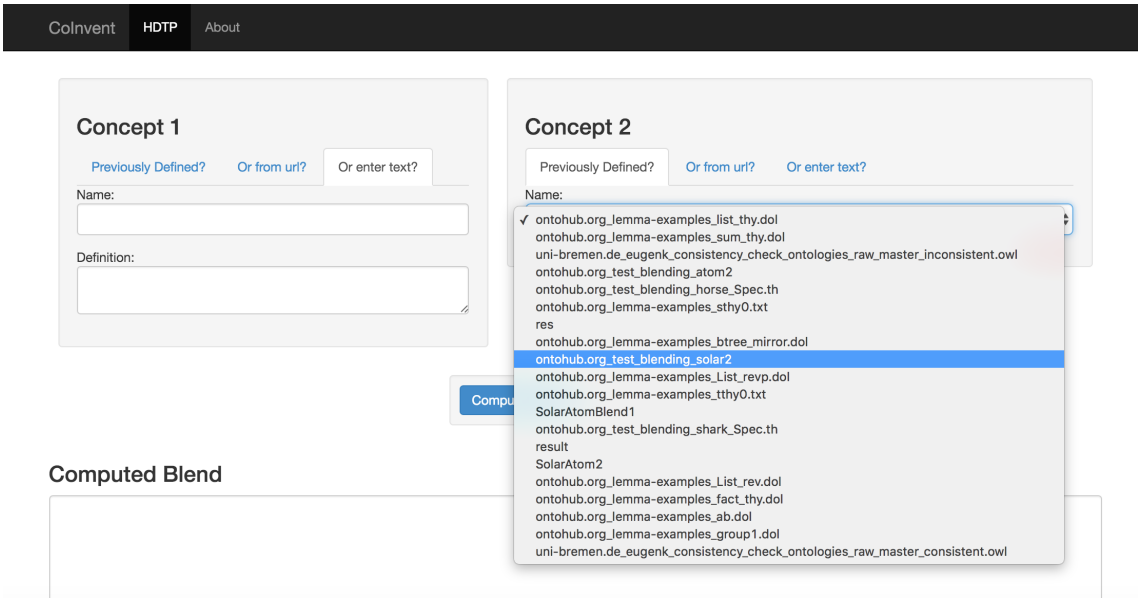

Figure 2: The Coinvent system main page, showing how a new theory can be defined in the text area on the left, and how a previous theory can be seleted in the right pane

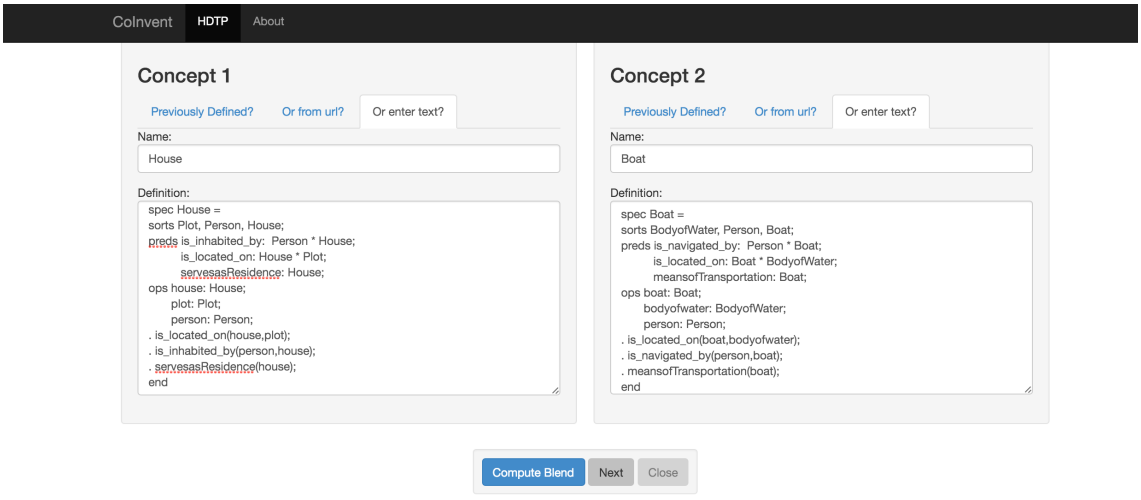

Figure 3: Defining new concepts in the text entry box in the Coinvent System

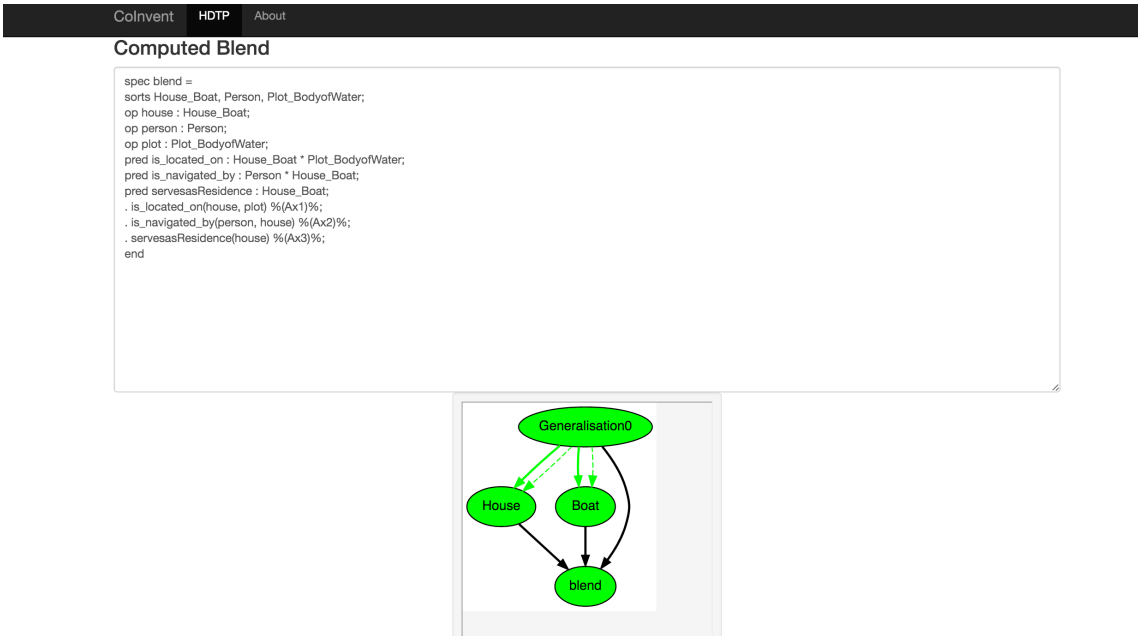

Figure 4: The result of computing the blend in the Coinvent System showing the graphical and text depiction of the blend

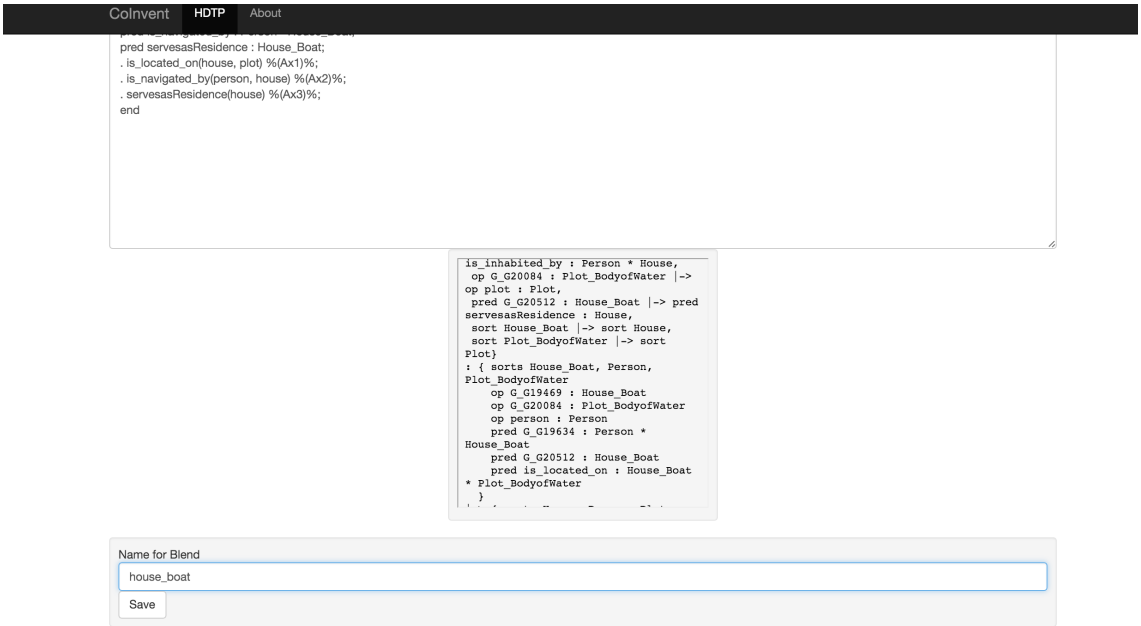

Figure 5: Investigating the calculated morphisms and saving the resulting blend

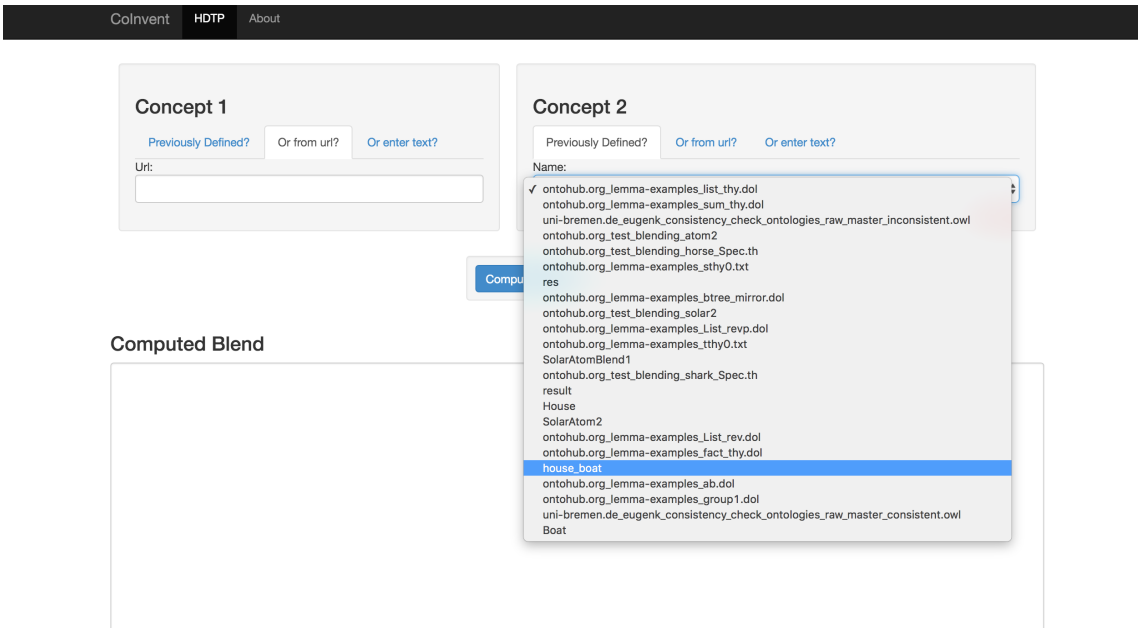

Figure 6: Selecting a saved blend as an input theory

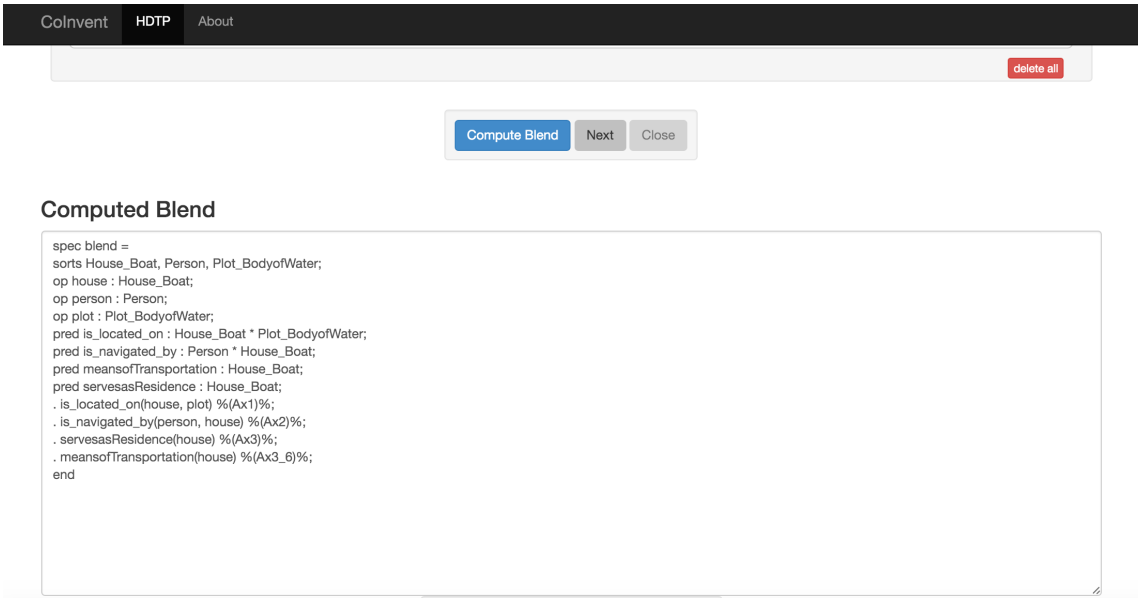

Figure 7: Requesting a new answer for the generic space and morphisms from HDTP, and recalculating the blend

4 Example Usage

In this section we give some example usages of calling the APIs in common languages.

4.1 javascript

```
var pid = $('input[name=pid]').val();
$.ajax({
url: '/cmd/blend',
data: {
action:action,
input1: JSON.stringify(input1),
input2: JSON.stringify(input2),
pid:pid
}
})
.then(function(a,b){
console.warn(a,b);
var output = ""+a.cargo.output;
$('textarea[name=output]').val(output);
if (a.cargo.pid) {
$('input[name=pid]').val(a.cargo.pid);
}
if ( ! output) {
// Fail :(
alert("Sadly this blend failed.");
}
})
```

4.2 Ruby

```
require 'rest-client'
require 'json'
require 'uri'


def execute_request()
  post_params = {action: 'hdtp', input1:
    {url: ARGV[0]}.to_json, input2: {url: ARGV[1]}.to_json}
  response = RestClient::Request.execute({method: :post,url:
      "148.251.85.37:8300/cmd/blend",payload: post_params})
    return response
end

x = execute_request()
```

```
my_hash = JSON.parse(x)

my_hash.each do |k,v|
  puts k + ':' + v.to_s
end
```