UNIVERSITY
OF TRENTO

DEPARTMENT OF INFORMATION ENGINEERING AND COMPUTER SCIENCE

38050 Povo – Trento (Italy), Via Sommarive 14
http://www.disi.unitn.it

PLUG-IN COMPONENT SUPPORTING QUERY ANSWERING

Coordinator: Pavel Shvaiko

with contributions from: Fausto Giunchiglia, Paolo Besana, Juan Pane and Mikalai Yatskevich

January 2008

Technical Report # DISI-08-006

OpenKnowledge

FP6-027253

Plug-in component supporting query answering

Coordinator: Pavel Shvaiko[1]
with contributions from
Fausto Giunchiglia[1], Paolo Besana[2], Juan Pane[1], Mikalai Yatskevich[1]

[1] Department of Information and Communication Technology (DIT),
University of Trento, Povo, Trento, Italy
{pavel|fausto|pane|yatskevi}@dit.unitn.it
[2] The University of Edinburgh, Edinburgh, UK
p.besana@ed.ac.uk

Report Version: final
Report Preparation Date: 17.12.2007
Classification: deliverable 4.3
Contract Start Date: 1.1.2006 Duration: 36 months
Project Co-ordinator: University of Edinburgh (David Robertson)
Partners: IIIA(CSIC) Barcelona
Vrije Universiteit Amsterdam
University of Edinburgh
KMI, Open University
University of Southampton
University of Trento

Abstract

This deliverable provides a brief documentation for the implementation of a plug-in component supporting query answering. Specifically, it discusses (i) the purpose and functionality of the component, (ii) its usage example, and finally (iii) plans for its future development.

1 Purpose and functionality

There are two purposes of the plug-in component supporting query answering. The first one is to verify what interaction model, among those found by the Discovery Service [1], best fits the plug-in components (called OKCs in [1]) available to a peer. The second purpose includes a creation of a bridge, if possible, between the components and the interaction. In turn, the bridge is created in two steps: (i) match the elements of the constraints expressed in lightweight coordination calculus (LCC) [2] to a method in the OKC and (ii) create an adaptor that is used by the LCC interpreter to access the actual values of the constraints. This deliverables focuses only on the second step, i.e., the adaptor creation, while the first step is described in [3]. Specifically, the adaptors are created by using the correspondences returned by the ontology matching component [3].

2 Adaptors usage example

The following web site `http://www.few.vu.nl/OK/wiki/` provides the OpenKnowledge (OK) client installation guidelines, while the source code is available at the project revision subversion control system (SVN) `http://fountain.ecs.soton.ac.uk/ok/repos/openk/trunk`[1]. Here we provide only a usage example of a plug-in component supporting query answering, which in turn is implemented by means of adaptors, and located at openK>src>org.openk.core.OKC within the SVN project.

Let us consider an example of a vendor and customer roles, see Figure 1. Suppose we have to match a customer method, such as desire_wine(Origin(Country, Region, Area), Colour) on an OKC to want(Region, Country, Colour), which is an LCC constraint; and a vendor method, for example, get_wine(Region(Country, Area), Colour, Cost, Year) to an LCC constraint, such as get_wine(Region, Country, Colour, Price). The created adaptors are shown as (red) rectangles.

[1] Authorization required, contact David Dupplaw (dpd@ecs.soton.ac.uk) for an account set up.

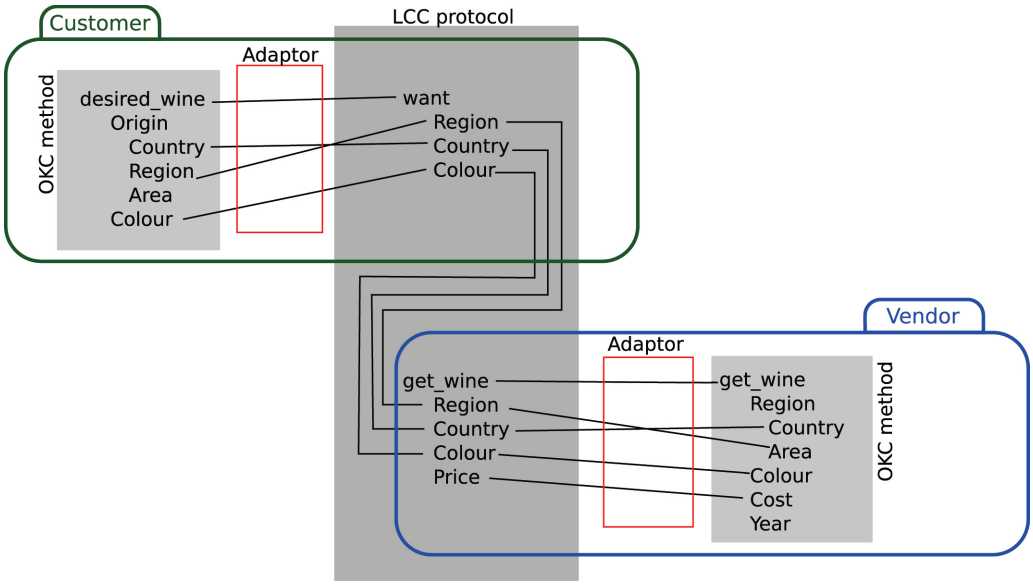

Figure 1: Use of adaptors to map content in a protocol to methods in peers.

The comparison between the constraints in the received interaction models and the methods in the OKCs available to the peer is performed by the class IMOKCComparatorImpl that calls the ontology matching component [3] and returns an object of type SubscriptionAdaptor. The SubscriptionAdaptor object contains a list of ConstraintAdaptor objects: each of them maps an LCC constraint and its arguments to a method in one of the available OKCs.

The parameters in methods and constraints are semantically marked up with their ontological types. This mark-up is made using the Java 5.0. annotations[2]. For example, for the method of a vendor OKC with the following signature Get_Wine(Argument R, Argument Col, Argument Cos, Argument Y) its Java annotation is as follows:

```
@MethodSemantic(language = "tag",
args = {"Region(Country, Area)", "Colour", "Cost", "Year"} )
```

The language attribute in the annotation specifies what mark-up language is used. In this case the ontological types of the parameters are defined by keywords. Another option is to use URIs of terms defined in ontologies available on the web. The parameters are accessed within the method using the Argument adaptors. For example, to access the country of a requested wine, the method will call getValue() of the first argument: R.getValue("/Region/Country[0]"). The numbers in square brackets (for instance, [0]) are used to index the elements of the constraints, see [3] for details.

[2] http://java.sun.com/j2se/1.5.0/docs/guide/language/annotations.html

This decouples how data is exchanged between peers in the LCC protocol and how data is accessed in the peers' components. The constraint of the example above can be used in an interaction model [2] as shown in Figure 2.

$a(customer, C)$ ::
$ask(Region, Country, Color) \Rightarrow a(vendor, V) \leftarrow$
$want(Region, Country, Color)$
$then\ reply(Price) \Leftarrow a(vendor)$

$a(vendor, V)$ ::
$ask(Region, Country, Color) \Leftarrow a(customer, C)$
$then$
$reply(Price) \Rightarrow a(customer, C) \leftarrow$
$Get\_Wine(Region, Country, Color, Price)$

Figure 2: A buying-selling interaction model.

Similarly, the peer in the customer role have an OKC with a method, such as desired_wine(Argument Origin, Argument Colour) along with the following Java annotation:

```
@MethodSemantic(language="tag",
args={"Origin(Country, Region, Area)", "Colour"})
```

The method results in a pop up window that asks the user for the values. Inside the method, the received arguments are accessed to set in the LCC constraint the values returned by the user. The actual correspondences between the LCC constraints and methods in the OKCs are shown as lines (connecting the semantically related elements) in Figure 1. The arguments in the method work as bridges towards the parameters as defined in the LCC constraint. A possible implementation of the customer method is as follows:

```
public boolean desired_wine(Argument Origin, Argument Colour){
    String cntr = JOptionPane.showInputDialog("Country?");
    String regio = JOptionPane.showInputDialog("Region?");
    String area = JOptionPane.showInputDialog("Area?");
    String clr = JOptionPane.showInputDialog("Colour?");
    Origin.setValue("/Origin/Country[0]", cntr);
    Origin.setValue("/Origin/Region[1]", regio);
    Origin.setValue("/Origin/Area[2]", area);
    Colour.setValue("Colour", clr);
}
```

where the calls to the setValue() method on the arguments set the value of the atomic parameters Country, Region and Colour in the LCC constraint. Note that, since the

LCC constraint want() in the customer role does not have the parameter Area, the call to setValue("/Origin/Area[2]", area) will not have any effect, since the Area parameter in the OKC is not matched to any parameter in the LCC constraint (see Figure 1).

Then following Figure 2, these parameters are sent, via the ask(...) message, to the peer performing the vendor role, that matches them to its own parameters in its get_wine(...) method, as shown in Figure 1. An implementation of get_wine(), which concludes the interaction as described in Figure 2, is as follows:

```
public boolean get_wine(Argument R, Argument Col, Argument Cost,
                        Argument Year){
// the connection to the DB is defined somewhere else
DBConnection=factory.getDBConnection();
Result result=DBConnection.query("SELECT cost FROM wines WHERE"+
    "Country='"+    R.getValue("/Region/Country[0]")+   "',"+
    "Area='"+       R.getValue("/Region/Area[0]")+      "',"+
    "Colour='"+     Col.getValue("/Colour")+            "',"+
    "Year='"+       Year.getValue("/Year")+"' LIMIT 1");
Record r = result.next(); //single tuple
Cost.setValue("/Cost", r.getInt(0));
}
```

3 Future work

Future work proceeds at least along the following directions: (i) making implementation of the plug-in under consideration robust, for example, while not currently implemented, semantic description of a method should also include a possibility of defining a default value for parameters when there is no corresponding value in the constraint; (ii) smooth integration of the plug-in into the OK system.

References

[1] David Dupplaw, Uladzimir Kharkevich, Spyros Kotoulas, Adrian Perreau, Ronny Siebes, and Chris Walton. *OpenKnowledge Deliverable 2.1: Architecting Open Knowledge*. http://www.cisa.informatics.ed.ac.uk/OK/Deliverables/D2.1a.pdf, 2006.

[2] Sindhu Joseph, Adrian Perreau de Pinninck, Dave Robertson, Carles Sierra, and Chris Walton. *OpenKnowledge Deliverable 1.1: Interaction Model Language Definition*. http://www.cisa.informatics.ed.ac.uk/OK/Deliverables/D1.1.pdf, 2006.

[3] Pavel Shvaiko, Fausto Giunchiglia, Mikalai Yatskevich, Juan Pane, and Paolo Besana. *OpenKnowledge Deliverable 3.6: Implementation of the ontology*

matching component. http://www.cisa.informatics.ed.ac.uk/OK/Deliverables/D3.6.pdf, 2007.