



## **LarKC**

*The Large Knowledge Collider*

*a platform for large scale integrated reasoning and Web-search*

**FP7 – 215535**

---

### **D5.4.2.1**

### **Release 2.5 of the LarKC Platform**

---

**Coordinator: Alexey Cheptsov (HLRS)**

**With contributions from: Matthias Assel (HLRS),  
Christoph Fuchs, Norbert Lanza (STI), Vassil  
Momtchev (ONTO), Spyros Kotoulas (VUA), Luka  
Bradesko, Blaz Fortuna (CYC), Ioan Toma, Mihai Chezan  
(Softgress)**

Quality Assessor: Alexey Cheptsov (HLRS)

Quality Controller: Alexey Cheptsov (HLRS)

<b>Document Identifier:</b>	LarKC/2008/D5.4.2.1
<b>Class Deliverable:</b>	LarKC EU-IST-2008-215535
<b>Version:</b>	1.0
<b>Date:</b>	May 28, 2011
<b>State:</b>	final
<b>Distribution:</b>	public



## EXECUTIVE SUMMARY

This document corresponds to the second release series (v2.5) of the LarKC Platform and constitutes the documentation of that release.

The main objective of this document is twofold:

- To serve as a guide for the LarKC users who want to use the LarKC Platform together with existing plug-ins and applications' workflows.
- To serve as a developer's guide for those users who are interested in developing their own plug-ins as well as composing workflows out of them and/or with existing plug-ins.

The document serves as a complete manual for the LarKC Platform, which may be used as a “getting started guide” by all kinds of potential LarKC early adopters and also as a manual for more experienced LarKC users. Additionally, several tools and features around the LarKC Platform (mailing lists, forums, plug-in marketplace) are described for supporting the different kinds of LarKC users accordingly.

## DOCUMENT INFORMATION

<b>IST Project Number</b>	FP7 – 215535	<b>Acronym</b>	LarKC
<b>Full Title</b>	Large Knowledge Collider		
<b>Project URL</b>	<a href="http://www.larkc.eu/">http://www.larkc.eu/</a>		
<b>Document URL</b>			
<b>EU Project Officer</b>	Stefano Bertolo		

<b>Deliverable</b>	<b>Number</b>	5.4.2.1	<b>Title</b>	Release 2.5 of the LarKC Platform
<b>Work Package</b>	<b>Number</b>	5	<b>Title</b>	The Collider Platform

Date of Delivery	Contractual	M33	Actual	31-Apr-11
Status	1.0		final <input checked="" type="checkbox"/>	
Nature	prototype <input checked="" type="checkbox"/> report <input type="checkbox"/> dissemination <input type="checkbox"/>			
Dissemination Level	public <input checked="" type="checkbox"/> consortium <input type="checkbox"/>			




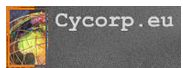










Authors (Partner)	Alexey Cheptsov, Matthias Assel (HLRS), Christoph Fuchs, Norbert Lanza (STI), Vassil Momtchev (ONTO), Spyros Kotoulas (VUA), Luka Bradesko, Blaz Fortuna (CYC), Ioan Toma, Mihai Chezan (Softgress)			
Resp. Author	Alexey Cheptsov		E-mail	cheptsov@hlrs.de
	Partner	HLRS	Phone	+49 (711) 685 60470


<b>Abstract (for dissemination)</b>	<p>This document corresponds to the second release series (v2.5) of the LarKC Platform and constitutes the documentation of that release.</p> <p>The main objective of this document is twofold:</p> <ul style="list-style-type: none"> <li>• To serve as a guide for the LarKC users who want to use the LarKC Platform together with existing plug-ins and applications' workflows.</li> <li>• To serve as a developer's guide for those users who are interested in developing their own plug-ins as well as composing workflows out of them and/or with existing plug-ins.</li> </ul> <p>The document serves as a complete manual for the LarKC Platform, which may be used as a "getting started guide" by all kinds of potential LarKC early adopters and also as a manual for more experienced LarKC users. Additionally, several tools and features around the LarKC Platform (mailing lists, forums, plug-in marketplace) are described for supporting the different kinds of LarKC users accordingly.</p>
<b>Keywords</b>	LarKC Platform, Workflow Designer Guide, Plug-in Developer Guide, Semantic Web



Version Log				
Issue Date		Rev No.	Author	Change
April 2011	26,	0.1	Alexey Cheptsov	First draft

## PROJECT CONSORTIUM INFORMATION

Acronym	Partner	Contact
Semantic Technology Institute Innsbruck <a href="http://www.sti-innsbruck.at">http://www.sti-innsbruck.at</a>		Prof. Dr. Dieter Fensel Semantic Technology Institute (STI) Innsbruck, Austria Email: dieter.fensel@sti-innsbruck.at
AstraZeneca AB <a href="http://www.astrazeneca.com">http://www.astrazeneca.com</a>		Bosse Andersson AstraZeneca Lund, Sweden Email: bo.h.andersson@astrazeneca.com
CEFRIEL SCRL. <a href="http://www.cefriel.it">http://www.cefriel.it</a>		Prof. Dr. Emanuele Della Valle CEFRIEL SCRL. Milano, Italy Email: emanuele.dellavalle@cefriel.it
CYCORP, RAZISKOVANJE IN EKSPERIMENTALNI RAZVOJ D.O.O. <a href="http://cyceurope.com">http://cyceurope.com</a>		Dr. Michael Witbrock CYCORP, RAZISKOVANJE IN EKSPERIMENTALNI RAZVOJ D.O.O., Ljubljana, Slovenia Email: witbrock@cyc.com
Höchstleistungsrechenzentrum, Universität Stuttgart <a href="http://www.hlrs.de">http://www.hlrs.de</a>		Matthias Assel Höchstleistungsrechenzentrum, Universität Stuttgart Stuttgart, Germany Email : assel@hlrs.de
MAX-PLANCK GESELLSCHAFT ZUR FÖRDERUNG DER WISSENSCHAFTEN E.V. <a href="http://www.mpg.de">http://www.mpg.de</a>		Dr. Lael Schooler, Max-Planck-Institut für Bildungsforschung Berlin, Germany Email: schooler@mpib-berlin.mpg.de
Ontotext Lab, Sirma Group Corp. <a href="http://www.ontotext.com">http://www.ontotext.com</a>		Atanas Kiryakov, Ontotext Lab, Sirma Group Corp. Sofia, Bulgaria Email: atanas.kiryakov@sirma.bg
SALT LUX INC. <a href="http://www.saltlux.com/EN/main.asp">http://www.saltlux.com/EN/main.asp</a>		Kono Kim SALT LUX INC Seoul, Korea E-mail: kono@saltlux.com
SIEMENS AKTIENGESELLSCHAFT <a href="http://www.siemens.de">http://www.siemens.de</a>		Dr. Volker Tresp SIEMENS AKTIENGESELLSCHAFT München, Germany Email: volker.tresp@siemens.com
THE UNIVERSITY OF SHEFFIELD <a href="http://www.shef.ac.uk">http://www.shef.ac.uk</a>		Prof. Dr. Hamish Cunningham THE UNIVERSITY OF SHEFFIELD Sheffield, UK Email: h.cunningham@dcs.shef.ac.uk
VRIJE UNIVERSITEIT AMSTERDAM <a href="http://www.vu.nl">http://www.vu.nl</a>		Prof. Dr. Frank van Harmelen VRIJE UNIVERSITEIT AMSTERDAM Amsterdam, Netherlands Email: Frank.van.Harmelen@cs.vu.nl
THE INTERNATIONAL WIC INSTITUTE, BEIJING UNIVERSITY OF TECHNOLOGY <a href="http://www.iwici.org">http://www.iwici.org</a>		Prof. Dr. Ning Zhong THE INTERNATIONAL WIC INSTITUTE Mabeshi, Japan E-mail: zhong@maebashi-it.ac.jp
INTERNATIONAL AGENCY FOR RESEARCH ON CANCER <a href="http://www.iarc.fr">http://www.iarc.fr</a>		Dr. Paul Brennan INTERNATIONAL AGENCY FOR RESEARCH ON CANCER Lyon, France Email: brennan@iarc.fr
INFORMATION RETRIEVAL FACILITY <a href="http://www.ir-facility.org">http://www.ir-facility.org</a>		Dr. John Tait INFORMATION RETRIEVAL FACILITY Vienna, Austria Email: john.tait@ir-facility.org

TECHNICAL UNIVERSITY OF CLUJ-NAPOCA <a href="http://www.utcluj.ro">http://www.utcluj.ro</a>		Prof. Dr. Eng. Sergiu Nedevschi TECHNICAL UNIVERSITY OF CLUJNAPOCA Cluj-Napoca, Romania Email: <a href="mailto:sergiu.nedevschi@cs.utcluj.ro">sergiu.nedevschi@cs.utcluj.ro</a>
SOFTGRESS S.R.L. <a href="http://www.softgress.com">http://www.softgress.com</a>		Dr. Ioan Toma SOFTGRESS S.R.L. Cluj-Napoca, Romania Email: <a href="mailto:ioan.toma@softgress.com">ioan.toma@softgress.com</a>

---

## TABLE OF CONTENTS

1	INTRODUCTION	1
1.1	Who should read this document? . . . . .	1
1.2	Document structure . . . . .	2
2	THE LARKC PLATFORM ARCHITECTURE 2.5	3
3	END-USER QUICK START GUIDE	7
3.1	Getting the Code . . . . .	7
3.2	Software Prerequisites . . . . .	7
3.3	Starting the Platform . . . . .	7
3.4	Submission of a Simple Workflow . . . . .	8
3.4.1	Query submission to the simple workflow . . . . .	8
3.5	Using the Workflow Designer tool . . . . .	9
3.5.1	Installation . . . . .	10
3.5.2	Main operations . . . . .	11
4	WORKFLOW DESIGNER GUIDE	12
4.1	LarKC Workflows . . . . .	12
4.1.1	From pipelines to workflow graphs . . . . .	12
4.1.2	The workflow description ontology . . . . .	13
4.1.3	A more complex example . . . . .	15
5	PLUG-IN DEVELOPER GUIDE	19
5.1	Use of the LarKC's Maven Archetype . . . . .	19
5.1.1	Using a Command Line Tool . . . . .	19
5.1.2	Using a Graphical Eclipse Wizard . . . . .	20
5.2	Importing the Generated Project to Eclipse IDE . . . . .	23
5.3	Building and Compiling the Plug-in with Maven . . . . .	24
5.4	Inside the Plug-in . . . . .	24
5.4.1	Basic Data Layer Functionality . . . . .	26
5.4.2	Caching Techniques . . . . .	27
5.4.3	Writing Tests . . . . .	27
5.5	Integration into the Platform . . . . .	28
5.5.1	Using the Plug-in . . . . .	28
6	USER SUPPORT	31
6.1	Joining LarKC@SourceForge.net . . . . .	31
6.2	User and Developer Support . . . . .	31
6.3	Contact us . . . . .	32
7	THE LARKC PLUG-IN MARKETPLACE	34
8	LICENSING	36
9	OPEN ISSUES AND FUTURE WORK	37
10	CONCLUDING REMARKS	38





---

## LIST OF FIGURES

2.1	The high-level view of the LarKC architecture. . . . .	4
2.2	The LarKC Platform components and their main responsibilities. . .	6
3.1	Management Interface's GUI (in a web browser). . . . .	8
3.2	Query submission through the Management Interface. . . . .	9
3.3	Workflow creation using the LarKC Workflow Designer. . . . .	10
4.1	A partial workflow graph. . . . .	12
4.2	Visual representation of <i>predicates</i> used to model a <i>LarKC workflow</i> . .	13
4.3	A complete workflow graph with an endpoint connected to a path. .	13
4.4	A more complex workflow consisting of multiple plug-in instances. .	16
5.1	Eclipse with LarKC plug-in wizard installed. . . . .	21
5.2	Choose platform location. . . . .	21
5.3	Creating a new LarKC plug-in project. . . . .	22
5.4	Creating a new LarKC plug-in. . . . .	22
5.5	Select your plug-in type. . . . .	23
5.6	The LarKC plug-in template in Eclipse. . . . .	24
5.7	Test your own plug-in with the LarKC Platform. . . . .	25
5.8	OWLIM Cluster configuration. . . . .	27
5.9	Accessing the Build Properties . . . . .	28
5.10	Setting the Build Target Properties . . . . .	29
6.1	LarKC bug-tracker. . . . .	32
6.2	LarKC developers forum. . . . .	33
7.1	Available plug-ins listed on the LarKC Plug-in Marketplace. . . . .	34
7.2	Plug-in submission form. . . . .	35



## LIST OF TABLES

## LIST OF ABBREVIATIONS

<b>API</b>	Application Programming Interface
<b>CPU</b>	Central Processing Unit
<b>GAT</b>	Grid Application Toolkit
<b>GSISSH</b>	Grid Security Infrastructure Secure Shell
<b>GSISCP</b>	Grid Security Infrastructure Secure Copy
<b>GT4</b>	Globus Toolkit 4
<b>GUI</b>	Graphical User Interface
<b>FOAF</b>	Friend Of a Friend
<b>FTP</b>	File Transfer Protocol
<b>HTML</b>	Hypertext Markup Language
<b>HTTP</b>	Hyper Text Transfer Protocol
<b>IDE</b>	Integrated Development Environment
<b>JDK</b>	Java Development Kit
<b>JEE</b>	Java Enterprise Edition
<b>JMX</b>	Java Management Extensions
<b>JVM</b>	Java Virtual Machine
<b>LarKC</b>	Large Knowledge Collider
<b>LTW</b>	Load Time Weaving
<b>ORDI</b>	Ontology Representation and Data Integration
<b>OWL</b>	Web Ontology Language
<b>OWLIM</b>	OWL In Memory
<b>PBS</b>	Portable Batch System
<b>RDF</b>	Resource Description Framework
<b>RDFS</b>	RDF Schema
<b>RTE</b>	Runtime Environment
<b>SIM</b>	Semantic Instrumentation and Monitoring
<b>SoS</b>	SetOfStatements
<b>SPARQL</b>	SPARQL Protocol And RDF Query Language
<b>SSH</b>	Secure Shell
<b>SCP</b>	Secure Copy
<b>SVN</b>	Subversion
<b>TCP</b>	Transmission Control Protocol
<b>URI</b>	Uniform Resource Identifier
<b>URL</b>	Uniform Resource Locator
<b>WID</b>	Workflow Identifier
<b>WSDL</b>	Web Services Description Language
<b>XML</b>	Extensible Markup Language

# 1 INTRODUCTION

Recent advances in the Semantic Web community have yielded a variety of reasoning methods used to process and exploit semantically annotated data. However, most of these have been developed for small, closed, trustworthy, consistent, and static domains. There is still a deep mismatch between the requirements for reasoning on a Web scale and the existing efficient reasoning algorithms over restricted subsets.

The EU FP7 Large Knowledge Collider project (LarKC, for short, pronounced lark) focuses, therefore, on supporting large-scale reasoning over billions of structured data in heterogeneous data sets. Specifically, LarKC aims at developing an experimental platform for massive, distributed and incomplete reasoning that will remove the scalability barriers of currently existing reasoning systems for the Semantic Web. The service-oriented platform architecture supports several types of plug-ins, which can be composed in a workflow and combined in an appropriate manner for the execution of a concrete task. This structure enables researchers and practitioners to run their own experiments and applications, and should allow for scaling well beyond what is currently possible.

In this document, we present the Release v2.5 of the LarKC Platform, with particular attention to its use of running Semantic Web applications on top of it. In addition, we explain how developers can create their own plug-ins and workflows using the LarKC Platform development environment.

The main objective of this document is twofold:

- To serve as a guide for LarKC users who want to use the LarKC Platform together with existing plug-ins and applications' workflows.
- To serve as a developer's guide for those users who are interested in developing their own plug-ins as well as composing workflows out of them and/or with existing plug-ins.

The document serves as a complete manual for the LarKC Platform, which may be used as a "getting started guide" by all kinds of potential LarKC early adopters and also as a manual for more experienced LarKC users. Additionally, several tools and features around the LarKC Platform (mailing lists, forums, plug-in marketplace) are described for supporting the different kinds of LarKC users accordingly.

## 1.1 Who should read this document?

This document is written bearing in mind as main readers the three kinds of LarKC users, namely:

- **Workflow designers:** they select existing plug-ins and combine them in the appropriate way (i.e., create a workflow description) to solve a certain task. Therefore, they should focus on Chapter 4. Chapter 2 is recommended in order to get an idea of the different platform components and the overall architecture.

- **Plug-in developers:** they design and implement single plug-ins and deploy them in the platform for future use. They should focus on Chapter 5 and 6. Chapter 2 is also recommended in order to get an idea of the different platform components and the overall architecture.
- **End-users:** their main task within LarKC is to use the services provided by the platform, i.e. a particular workflow description to execute SPARQL queries. The most interesting part for them is Chapter 3.

Chapter 7 and 8 might be of interest for all three user groups, since it may affect the purposes for which the platform is used and the license under which newly developed components should be released. Chapter 9 might be of interest for all kind of users and developers which would like to follow the platform development progress.

Note: All LarKC deliverables, some of which are referenced in this document for those readers interested in getting further details, can be found at <http://www.larkc.eu/deliverables/>.

## 1.2 Document structure

The remainder of this document is structured as follows:

- Chapter 2 gives a high-level overview of the LarKC Platform Architecture
- Chapter 3 provides a quick start guide for the new users as well as early adopters
- Chapter 4 aims at the workflow designer who construct their own application workflows based on LarKC plug-ins
- Chapter 5 aims at the plug-in developers who want to write their own plug-ins
- Chapter 6 gives references to the user support tools provided and maintained by LarKC, such as mailing lists, trouble ticketing systems and forums.
- Chapter 7 introduces the LarKC Plug-in Marketplace and describes its main purpose and benefit
- Chapter 8 provides information about the licensing scheme under which the platform code is released
- Chapter 9 gives some hints on future developments which are planned within the framework of the LarKC Platform until the end of the project
- Chapter 10 concludes this document

## 2 THE LARKC PLATFORM ARCHITECTURE 2.5

This chapter presents a very high level overview of the new LarKC Platform Architecture and how the software is structured according to it. Note that this chapter is not intended to give an exhaustive and detailed description of the LarKC Platform Architecture and its components. Explicit information and detailed descriptions of particular components as well as their main interactions can be found at <http://www.larkc.eu/deliverables/>.

From the system organisation standpoint, LarKC is a software-hardware infrastructure that offers:

- a framework for the development of plug-ins and designing workflows based on those plug-ins;
- a run-time environment for executing workflows;
- an infrastructure for storing the data and running workflows on a dynamic resource pool made of heterogeneous and distributed systems, ranging from web servers hosting the plug-in to parallel and supercomputing systems.

The second edition of the LarKC architecture aims to be even more open, dynamic, service-oriented, and scalable as its predecessor. The architecture design allows overcoming the known limitations and meeting emerging user requests. The techniques underlying the design of the LarKC Platform allow for a trade-off between flexibility and performance, in order to achieve a good balance between generality and usability of plug-ins and workflows, respectively.

The architecture is conceptualised as shown in Figure 2.1.

The three main domains are served by the architecture:

- user domain;
- platform domain;
- infrastructure domain.

The *user domain* aims at three main categories of users, namely plug-in developers, workflow designers, and end-users. The *plug-in developers* enjoy the collaborative development within the LarKC Marketplace<sup>1</sup>, supported by special tools offered by the platform such as Plug-in Wizard (refer to 5.1.2).

The *workflow designers* get access to the Marketplace in order to construct a workflow from the available plug-ins, combined to solve a certain task. The workflows are already standalone applications, which can be submitted to the platform and executed (by means of such tools as the Workflow Designer GUI, described in Section 3.5) However, the workflows can also be used as a basis for building more advanced applications, for example offering a more extended interface to be used by the *end-users*.

The platform domain is a core part of the architecture that consolidates services that facilitate the plug-in development and the workflow design. Moreover,

---

<sup>1</sup><http://www.larkc.eu/plug-in-marketplace/>

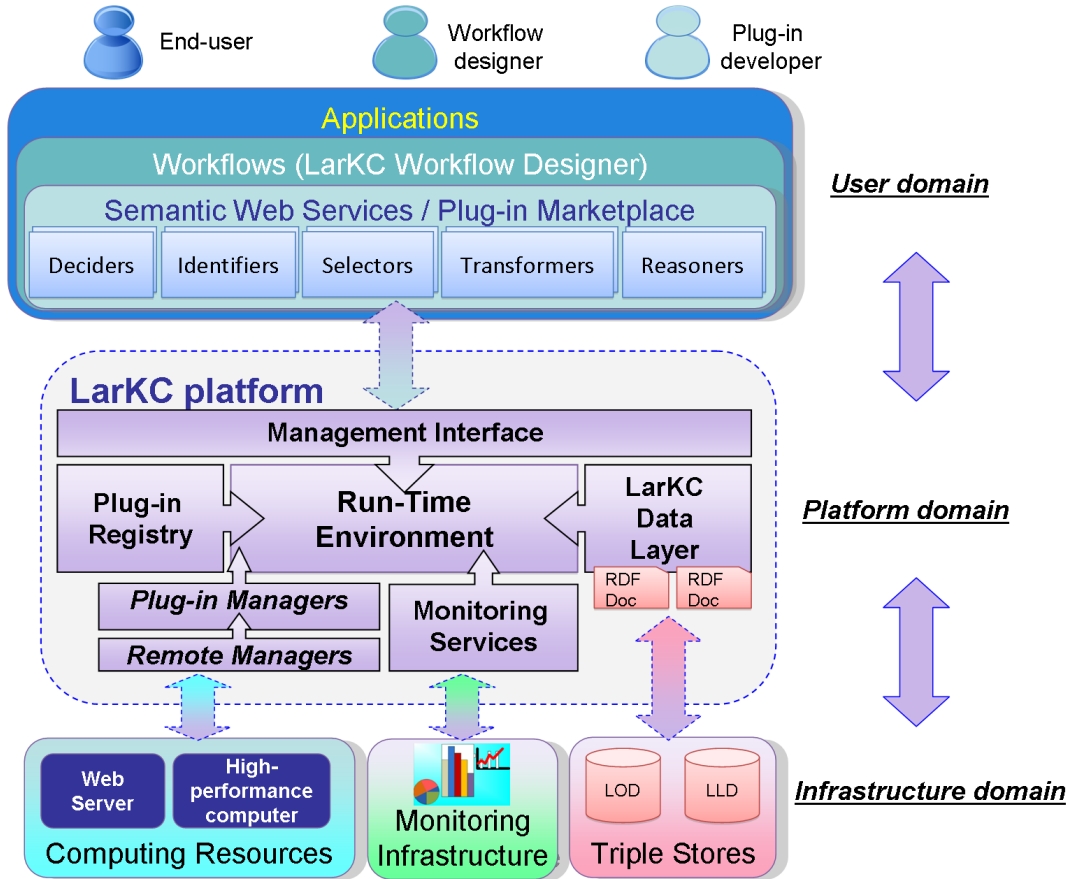


Figure 2.1: The high-level view of the LarKC architecture.

the platform serves a run-time environment for the plug-in and workflow execution, sets up and manages the data layer for the efficient data flow handling, provides means for the performance monitoring and evaluation. The run-time environment is not limited to the host where the platform is deployed, but also enables remote hosts (please refer to the LarKC architecture documentation at <http://www.larkc.eu/deliverables/>) for the plug-in execution.

The efficient workflow execution often requires special infrastructure resources to be accessible by workflows. Those resource facilities constitute the infrastructure domain. The platform offers all the necessary support for easy and transparent access of the external computation, storage, and monitoring facilities.

In the following, we give a brief description of the LarKC Platform components (see Figure 2.2 for details) and their location in the source code repository<sup>2</sup>:

- **The LarKC Runtime Environment (RTE)** is the “control centre” of the LarKC Platform and responsible for the initialisation and invocation of workflows. Its core components are the Executor and the respective End-points (i.e, user interfaces to send queries and retrieve results) on top of it. The executor takes the workflow description, selects the plug-ins from the registry and constructs the corresponding workflow. It takes care of what we call *workflow branching, conditional loops, and splits and merges of the*

<sup>2</sup>Both the latest release and the development trunk can be found at: <https://sourceforge.net/projects/larkc/>

*data flow*. In addition, the Executor interacts with Plug-in Managers to steer their proper execution and regulates the internal processing of events.

- The main code implementing the LarKC Runtime Environment can be found in package *eu.larkc.core.executor* and *eu.larkc.core.endpoint*, respectively.
- **The Management Interface** implements a “bridge” towards users as well as developers allowing them to submit their workflow description - workflows are now specified in RDF.
  - The main code implementing the Management Interface can be found in package *eu.larkc.core.management*
- **The Plug-in Registry** bases mainly on the OpenCyc<sup>3</sup> engine and is responsible for the management of plug-ins. It allows the registration of plug-ins, including their semantic descriptions (plug-in descriptions are given in RDF), and the subsequent retrieval for their use within an execution workflow.
  - The main code implementing the Plug-in Registry can be found in package *eu.larkc.core.pluginregistry*
- **The Plug-in Managers** facilitate the integration of plug-ins within a workflow and the management of their execution. Hence, they allow the deployment of particular plug-ins either locally or remotely/distributed. The later is enabled with the help of the corresponding extensions for remote execution (i.e., adapters to connect to various remote resources, e.g. SSH or GT4 adapter<sup>4</sup>).
  - The main code implementing the Plug-in Managers and its extensions can be found in package *eu.larkc.core.pluginManager*
- **The Data Layer** supports the plug-ins and applications with respect to storage, retrieval (including streaming), and lightweight inference on top of large volumes of RDF data. It therefore relies on OWLIM. The LarKC Data Layer is connected to the rest of the Platform through a well-defined API. The LarKC Data Layer API describes the different types of RDF data providers and their functionality.
  - The main code implementing the LarKC Data Layer API can be found in package *eu.larkc.core.data*

---

<sup>3</sup><http://www.opencyc.org>

<sup>4</sup>see <http://www.cs.vu.nl/ibis/javagat.html> for more details



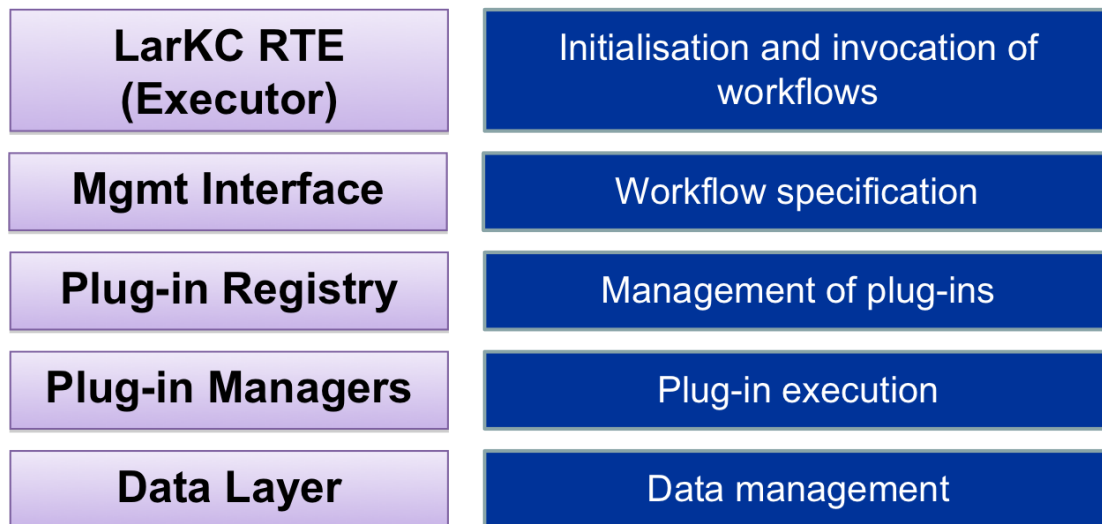


Figure 2.2: The LarKC Platform components and their main responsibilities.

## 3 END-USER QUICK START GUIDE

This section aims at the Early Adopters who would like to get a quick overview about LarKC, get it installed, and run a simple application/workflow with LarKC.

### 3.1 Getting the Code

The latest LarKC distribution can be downloaded from [LarKC@SourceForge.net](http://LarKC@SourceForge.net).

Afterwards, the content needs to be extracted from the zip file (to a new directory). The unpacked archive contains the following main entries:

- **platform** - the core of LarKC, a software infrastructure and run-time environment that enables flexible development and serves a deployment environment for the LarKC plug-ins (see more details below);
- **platform/plugins** - Exemplary collection of the LarKC Plug-Ins, including:
  - *LLDReasoner* - executes a SPARQL query against Linked Life Data repository
  - *SOSToVBtransformer* - is used for transformation of a SetOfStatements to the VariableBinding representation
  - *plugging.TestXX* - dummy plug-ins that just return the input given to them
- **development\_kit/Workflow-Designer** - a tool for constructing workflow
- **workflows** - exemplary workflows, including:
  - *LLD\_Reasoning* - executes a SPARQL query against Linked Life Data
  - *Test\_Local* - a test workflow that consists of empty plug-ins
  - *Test\_Remote* - a test workflow that contains a plug-in (*TestRemoteIdentifier*) running remotely on a Tomcat Server
- **doc** - the user/developer manual.

### 3.2 Software Prerequisites

A Java Development Kit (JDK) v1.5 or more, please refer to [Oracle's download page](#)

### 3.3 Starting the Platform

In order to start the platform, switch to the *platform* directory and execute the *run-larkc* script or batch file (depending on your OS).

If the platform started correctly, the output should be as follows:

```
1 ...
2 INFO: Starting the internal HTTP server on port 8182
3 ...
4 INFO e.l.c.m.ManagementInterfaceMain: Management server started on 8182
```

### 3.4 Submission of a Simple Workflow

If the previous steps have successfully been performed, the LarKC platform's **Management Interface** should be accessible at <http://localhost:8182>, see in Figure 3.1

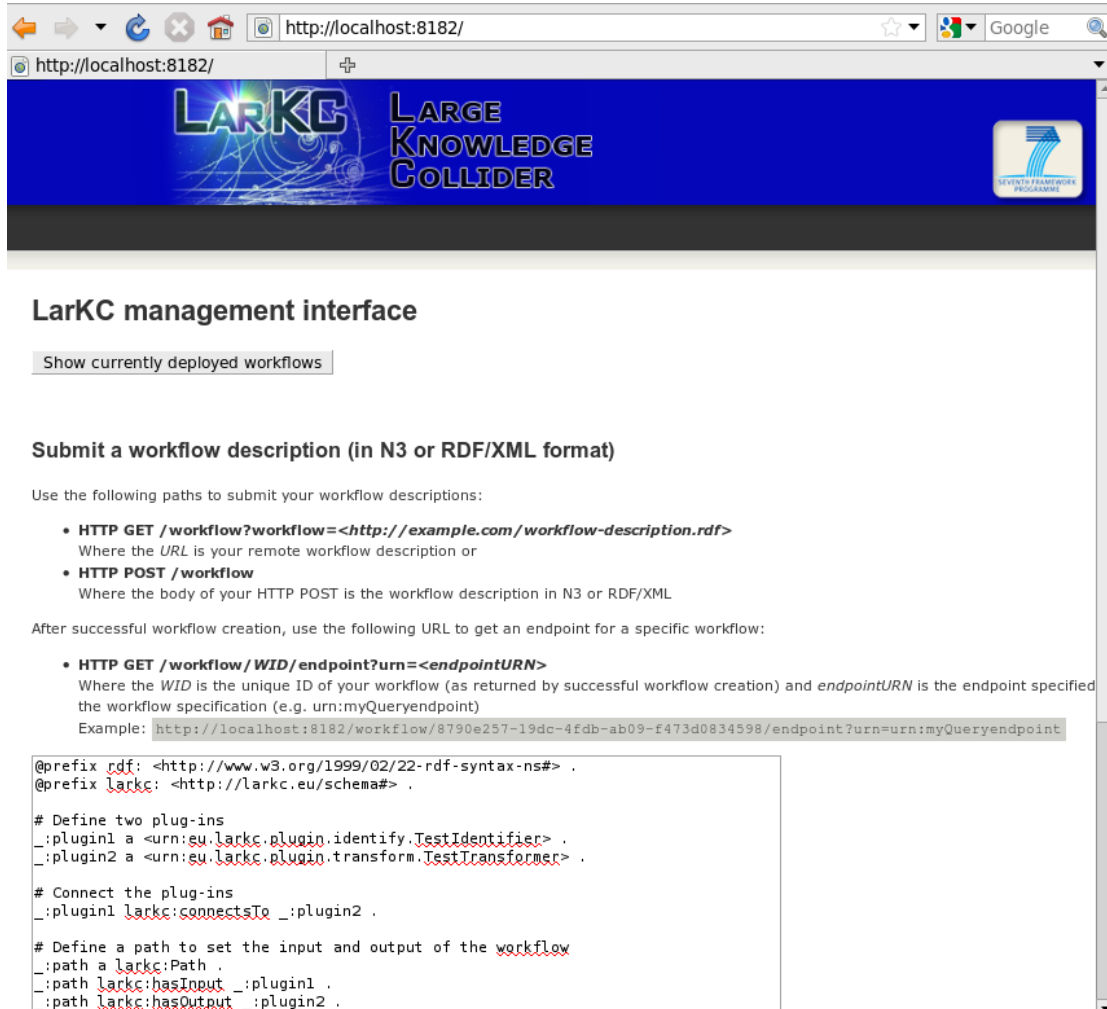


Figure 3.1: Management Interface's GUI (in a web browser).

- In the *Submit a workflow* field, add you workflow description, either in N3 or XML format, or keep the default description. Click on the *Submit* button to submit the workflow to the *Management Interface*. Return back to the main view of the *Management Interface*, i.e. by clicking on the browser's return button. Clicking on *Show workflows* button, you should your workflow id in the list of the submitted workflows.

#### 3.4.1 Query submission to the simple workflow

Once the workflow has been submitted, a query can be sent to the workflow's end-point for execution. To do this, click on the *Show currently deployed workflows*, pick the workflow that the query should be submitted to, copy the workflow end-point's urn to the field next to the *Submit* button, and click on it (see Fig 3.2). To get the answer on the query, click on the *Get result* button.

## LarKC management interface

### Currently deployed workflows

Workflow ID	Endpoints
1c317f5b-8178-4c20-afc3-802bcec53c47	urn:myTestEndpoint (http://127.0.1.1:8183/testendpoint)

### Submit a query to an endpoint

To submit the query in the textarea below to an endpoint you have to specify the endpoint URL in the textarea next to the **Submit** button (all available endpoints are shown in the table above in the last column).

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?name where {
  ?person rdf:type foaf:Person.
  ?person foaf:name "Frank van Harmelen".
  ?person2 rdf:type foaf:Person.
  ?person2 foaf:knows ?person.
  ?person2 foaf:name ?name.
}
```

Submit

http://127.0.1.1:8183/testendpoint

Get results

http://127.0.1.1:8183/testendpoint

Figure 3.2: Query submission through the Management Interface.

## 3.5 Using the Workflow Designer tool

To have a more powerful workflow construction interface, there has been a special graphical front-end created, which is also included in the basic LarKC release. Workflow Designer is a LarKC application that can be used to construct workflows out of the available plug-ins in LarKC (see Figure 3.3)

Designer is a middleware on top of the management interface that allows users to construct, configure and execute workflows. The designer is a JavaScript-based web application, offering an intuitive GUI for intelligent workflow design. The application has the following main features:

- Retrieval of the list of all available plug-ins from the plug-in registry, done through the platform's management interface (Figure 3.3a).
- Retrieval of the list of all available remote host templates (i.e., pre-defined configuration files), done through the platform's management interface (Figure 3.3b).
- Creation of workflows by visually dragging and dropping LarKC plug-ins, specification of workflow dependencies (Figure 3.3c).
- Easy and intuitive specification of the deployment host for the plug-in (Figure 3.3d).
- Conversion of the graphical workflow representation into RDF/XML or N3 format, submission of the created workflow description to the management interface, and instantiation of the workflow within the platform.
- Choice of the needed endpoint (Figure 3.3e).

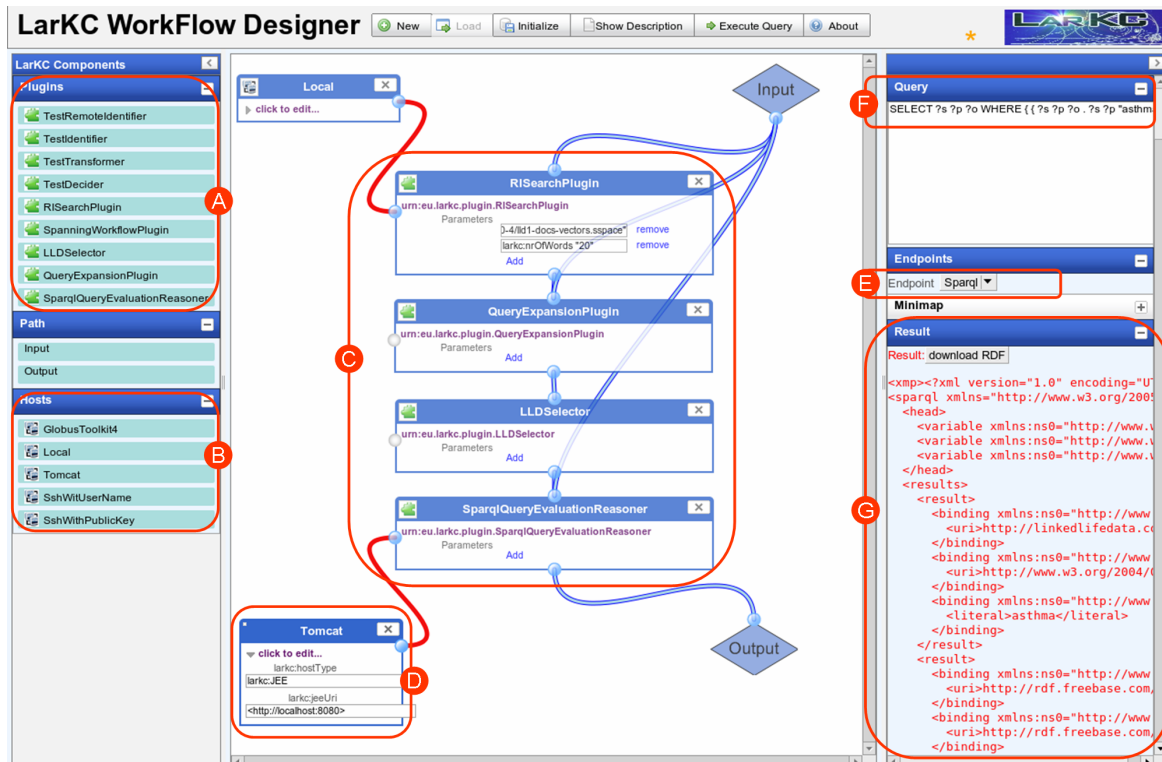


Figure 3.3: Workflow creation using the LarKC Workflow Designer.

- Submission of a query (e.g. SPARQL) to the SPARQL endpoint of the submitted workflow (Figure 3.3f).
- Visualisation the query results in a separate window as an XML document (Figure 3.3g).

### 3.5.1 Installation

Designer a web application that needs an Apache Tomcat web server running on the user's local machine.

- If you don't have Tomcat installed, just switch to the *extra\_tools/workflow\_designer* folder of the redistribution package and execute a run script (or batch file for windows).
- If you already have Tomcat installed and running, you can simply deploy the *war file* of the workflow designer (located at *extra\_tools/workflow\_designer/webapps*) in your Tomcat. In order to do that, open a new web browser window, point it to <http://localhost:8080/manager/html> (refer to the Tomcat documentation for more information or help), go to the field *WAR file to deploy* at the bottom of the management page, click on browse, and point to the Workflow Designer's war file located at *Development\_Kit/Workflow-Designer* entry of your LarKC redistribution.

After the abovementioned steps have been completed, the designer can be started by pointing your browser to <http://localhost:8080/Larkc/>.

### 3.5.2 Main operations

After starting the designer, the available plug-ins will be retrieved from the platform and will be listed on the left side. User can drag and drop the plug-ins to the workflow design area and connect them to build a workflow (middle part of the GUI). If necessary, parameters can also be added for each plug-in separately. Once the workflow is built, users can check the corresponding N3 representation of the workflow by clicking the "Show RDF" button. Afterwards, users can submit the workflow via the management interface to the LarKC Platform by clicking the "Initialize" button.

After the submission of the workflow, the workflow ID and endpoint will be retrieved and displayed in the "Result" panel on the right side. To execute SPARQL queries, the user simply needs to input or select a query from the "Query" field and click the "Execute Query" button. Results are also being shown at the bottom of the right hand side panel.

## 4 WORKFLOW DESIGNER GUIDE

### 4.1 LarKC Workflows

The following section will examine the purpose and usage of *LarKC workflows*. A small example will present the basic principles of a LarKC workflow. More example workflows can be found in the *eu.larkc.shared* package<sup>1</sup> or in Appendix A of this document.

#### 4.1.1 From pipelines to workflow graphs

The workflow concept was first introduced in version 2.0 of the LarKC Platform. Previous platform versions were only able to process a linear sequence of operations (a plug-in “pipeline”), which did not allow for conditional execution paths, conditional loops or simultaneous execution of plug-ins. To overcome this limitation, a more flexible mechanism was developed collaboratively by WP1 and WP5. Since a *LarKC workflow* is a formal description, an ontology was defined in order to express workflows in RDF (details about the ontology will be specified in the forthcoming deliverable D1.3.2B).

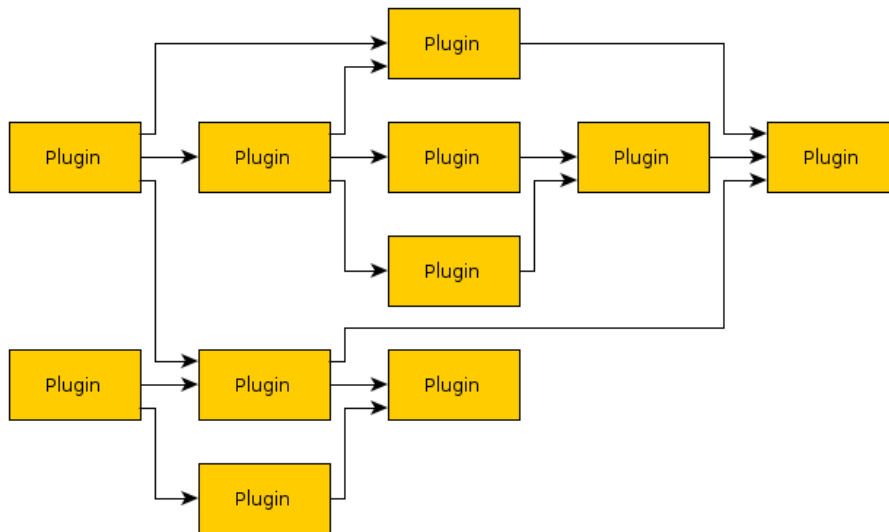


Figure 4.1: A partial workflow graph.

A workflow description is a directed graph where the vertices are plug-ins and endpoints, and the edges represent the connections between those plug-ins and endpoints. Figure 4.1 shows a possible configurations of interconnected plug-ins which form the basis of a workflow. The same configuration of plug-ins is shown in Figure 4.3, where one plug-in is marked as an *input* of a path, and another plug-in is marked as an *output* of a path. A path can be seen as the manifestation of a data flow from a source (input plug-in) to a sink (output plug-in). Each path

<sup>1</sup><http://sourceforge.net/projects/larkc/>

must have at least one input and *exactly one* output. To feed input to a path and retrieve output from a path, an endpoint has to be connected to it. When an endpoint is connected to a path with multiple inputs, every input plug-in instance will get the same input from the endpoint.

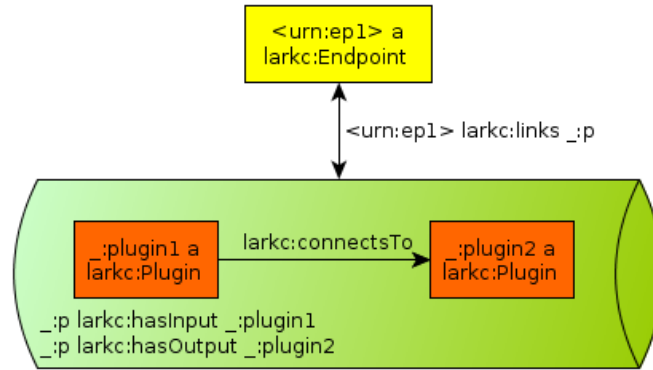


Figure 4.2: Visual representation of *predicates* used to model a *LarKC workflow*.

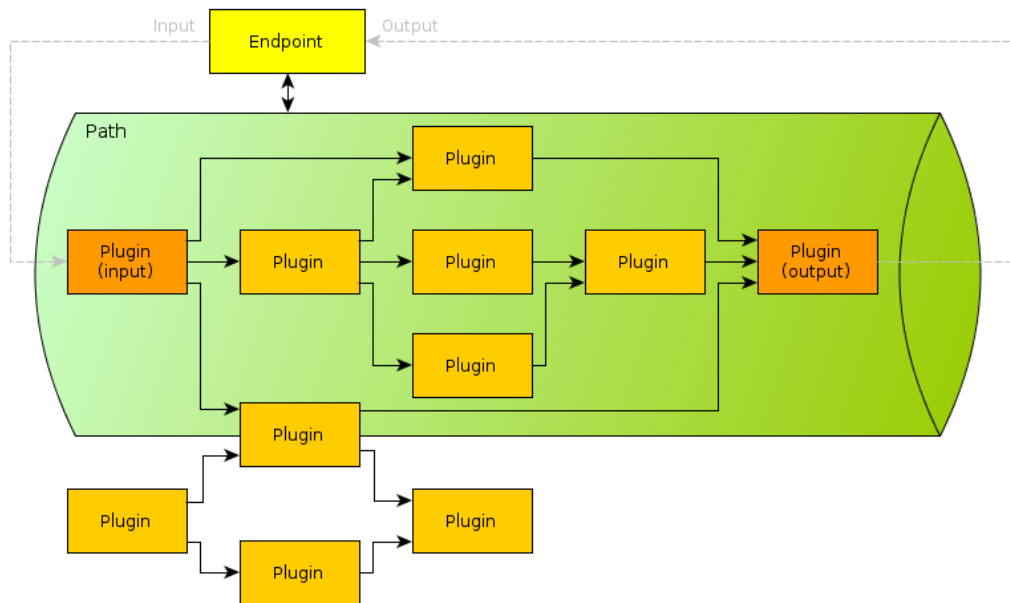


Figure 4.3: A complete workflow graph with an endpoint connected to a path.

#### 4.1.2 The workflow description ontology

As stated before, a *LarKC workflow* is described as a RDF document which represents a graph of interconnected plug-ins and endpoints. Optionally, plug-ins can also have certain properties: an *input-* and *output-behaviour* as well as *input-* and *output-parameters*.



The input- and output-behaviour properties define how the plug-in managers will handle splits and merges of data. Currently there is one kind of output-behaviour supported: cloning. If a plug-in instance has multiple successors, every successor will get the output of this plug-in instance as an input. The input-behaviour property defines how many inputs are required for the plug-in instance to execute. The responsible plug-in manager invokes the plug-in instance as soon as the required inputs are ready.

Input- and output-parameters can be defined by the `larkc:hasParameter` predicate, which offers the possibility to add parameters which are known before the workflow is executed to a plug-in instance. On plug-in initialisation, the Executor will pass all parameters as a `SetOfStatements` to the plug-in instance.

A minimalistic example of a workflow description, which does not utilise any plug-in parameters or behaviours is given in Listing 4.1.

```

1 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
2 @prefix larkc: <http://larkc.eu/schema#> .
3
4 # Define two plug-ins
5 _:plugin1 a <urn:eu.larkc.plugin.identify.TestIdentifier> .
6 _:plugin2 a <urn:eu.larkc.plugin.transform.TestTransformer> .
7
8 # Connect the plug-ins
9 _:plugin1 larkc:connectsTo _:plugin2 .
10
11 # Define a path to set the input and output of the workflow
12 _:path larkc:hasInput _:plugin1 .
13 _:path larkc:hasOutput _:plugin2 .
14
15 # Connect an endpoint to the path
16 <urn:eu.larkc.endpoint.sparql.ep1> a <urn:eu.larkc.endpoint.sparql> .
17 <urn:eu.larkc.endpoint.sparql.ep1> larkc:links _:path .

```

Listing 4.1: Two plug-ins form a path connected to an endpoint.

Figure 4.2 shows a graphical representation of the workflow described in Listing 4.1. The workflow consists of two plug-ins which form a path. A workflow can have multiple paths, each path has an input and an output to mark the beginning and the end of the path. In our example we do not have any intermediate plug-ins nor plug-in parameters - but keep in mind that a more complex workflow may have a big number of intermediate plug-ins as well as plug-in parameters to form a path.

The following table explains all predicates that can be used to create a workflow. Every plug-in instance, parameter instance and path instance (which occurs in the table as domain or range of some predicates) is represented in the workflow description as a blank node (see for example Listing 4.1 line 12). Every endpoint instance is identified by a URN which is unambiguous within the workflow description.

Predicate	Domain	Range	Usage
<rdf:type>	Plug-in or endpoint instance	Plug-in or endpoint type (URN)	Defines the type of a plug-in or an endpoint.
<larkc:connectsTo>	Plug-in instance	Plug-in instance	Connects two plug-ins.
<larkc:hasParameter>	Plug-in instance	Parameter instance	Connects a plug-in instance to its parameters (every parameter uses the blank node as subject).
<larkc:isInputSplittable>	Parameter instance	Boolean value	Indicates if the input for the plug-in is splittable (which enables parallel execution).
<larkc:hasInputBehaviour>	Parameter instance	Integer value	The value states how many inputs are needed for the plug-in to proceed.
<larkc:runsOn>	Plug-in instance	Blank node or URL	Specifies where the plug-in should be executed (defined by a location or other statements with the blank node as subject).
<larkc:hasInput>	Path instance	Plug-in instance	Defines that the plug-in instance is an input for the path.
<larkc:hasOutput>	Path instance	Plug-in instance	Defines that the plug-in instance is the output of the path.
<larkc:links>	Endpoint instance	Path instance	Connects an endpoint to a path.

In the next section this predicates are used and explained in a more complex example.

### 4.1.3 A more complex example

The following example consists of 10 plug-in instances which form two *paths*. Figure 4.4 shows a graphical representation of the workflow description. The full workflow description is presented in Listing A.1 (Appendix A). Throughout this example multiple workflow mechanisms will be demonstrated: splits and merges of data flow, plug-in instances with non-default input behavior (non-blocking plug-in instances), multiple paths per workflow and multiple endpoints per workflow.

Any workflow description needs to have at least one plug-in instance defined. In this particular example, ten plug-in instances are defined (see Listing 4.2).

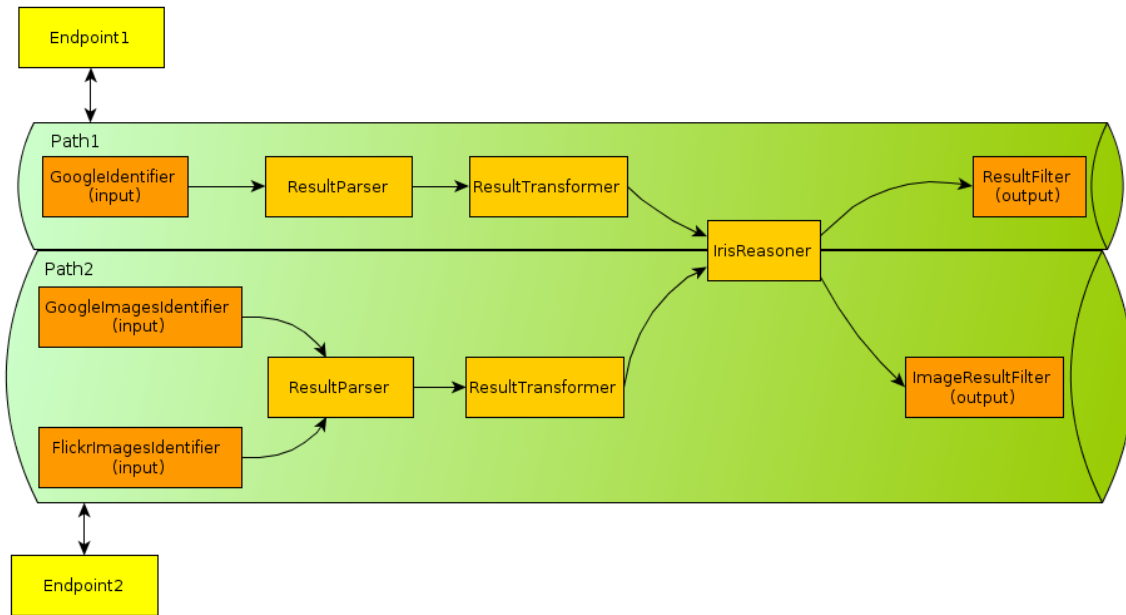


Figure 4.4: A more complex workflow consisting of multiple plug-in instances.

Note that there are only 8 types of plug-ins, since two plug-in types have multiple instances (*ResultParser* and *ResultTransformer*).

```

1 # Define the plug-ins used in this workflow
2
3 _:gi a <urn:eu.larkc.plugin.identify.google.GoogleSearchIdentifier> .
4 _:gii a <urn:eu.larkc.plugin.identify.google.images.GoogleImageIdentifier> .
5 _:fii a <urn:eu.larkc.plugin.identify.flickr.images.FlickrImagesIdentifier> .
6
7 _:rp1 a <urn:eu.larkc.plugin.parser.ResultParser> .
8 _:rt1 a <urn:eu.larkc.plugin.transformer.ResultTransformer> .
9 _:rp2 a <urn:eu.larkc.plugin.parser.ResultParser> .
10 _:rt2 a <urn:eu.larkc.plugin.transformer.ResultTransformer> .
11
12 _:ir a <urn:eu.larkc.plugin.reasoner.iris.IrisReasoner> .
13
14 _:rf a <urn:eu.larkc.plugin.filter.ResultFilter> .
15 _:irf a <urn:eu.larkc.plugin.filter.image.ImageResultFilter> .

```

Listing 4.2: Plug-in declaration part.

After the plug-in instances have been declared, they can be connected to form a workflow. In this example two paths are defined. The statements which define the connections of the first path are listed in Listing 4.3. By formulating statements in the form of `_:PluginInstanceA larkc:connectsTo _:PluginInstanceB`, the following connections are drawn: *GoogleIdentifier* connects to *ResultParser1*, which connects to *ResultTransformer1*, which connects to *IrisReasoner*, which connects to *ResultFilter*.

```

1 # Path 1
2
3 # Connect the plug-ins
4 _:gi larkc:connectsTo _:rp1 .
5 _:rp1 larkc:connectsTo _:rt1 .
6 _:rt1 larkc:connectsTo _:ir .
7 _:ir larkc:connectsTo _:rf .
8
9 # Define a path and set the input and output of the path

```

```

10 _:path1 larkc:hasInput _:gi .
11 _:path1 larkc:hasOutput _:rf .

```

Listing 4.3: Connections of plug-in instances which form the first path.

Path 2 consists of six plug-in instances which need to be connected as well. Listing 4.4 shows the `larkc:connectsTo` statements for the second path. Those statements can again be translated as follows: both *GoogleImagesIdentifier* and *FlickrImagesIdentifier* connect to *ResultParser2*, which connects to *ResultTransformer2*, which connects to *IrisReasoner*, which connects to *ImageResultFilter*.

```

1 # Path 2
2
3 # Connect the plug-ins
4 _:gii larkc:connectsTo _:rp2 .
5 _:fii larkc:connectsTo _:rp2 .
6 _:rp2 larkc:connectsTo _:rt2 .
7 _:rt2 larkc:connectsTo _:ir .
8 _:ir larkc:connectsTo _:irf .
9
10 # Define a path and set the input and output of the path
11 _:path2 larkc:hasInput _:gii .
12 _:path2 larkc:hasOutput _:irf .

```

Listing 4.4: Connections of plug-in instances which form the second path.

As shown in Figure 4.4, the reasoner plug-in instance is shared by both paths. This simply means that on invocation this particular plug-in instance may be blocked by the execution of the other path. To communicate with paths each path needs to connect to an *endpoint*, which is responsible for providing the *input plug-in* with queries and retrieving results from the *output plug-in*. Listing 4.5 defines two endpoint instances and connects one endpoint to each path.

```

1 # Connect an endpoint to the paths
2
3 <urn:eu.larkc.endpoint.test.ep1> a <urn:eu.larkc.endpoint.test> .
4 <urn:eu.larkc.endpoint.test.ep1> larkc:links _:path1 .
5
6 <urn:eu.larkc.endpoint.test.ep2> a <urn:eu.larkc.endpoint.test> .
7 <urn:eu.larkc.endpoint.test.ep2> larkc:links _:path2 .

```

Listing 4.5: Declaration of endpoints and connections of paths and endpoints.

It might be necessary to provide some plug-in instances with configuration or other specific data. This can be specified in the workflow by utilising the `larkc:hasParameter` predicate. In Listing 4.6 an example of such a parameter definition can be seen. The *GoogleImagesIdentifier* has a blank node as a parameter, which in turn is defined by two triples. Here `rdfs:label` and `rdf:value` is used to indicate that the parameter name is `minImageSize` and has a value of 1024.

To assure that the *IrisReasoner* plug-in does not block either of the paths, its input behavior has to be set accordingly. If no input behavior is set, a plug-in will wait for all predecessor plug-ins to finish before it starts to execute. By defining `larkc:hasInputBehaviour` this default behavior is overridden. Line 7 of Listing 4.6 shows how the input behavior can be defined.

```

1 # Define plug-in specific parameters
2 _:gii larkc:hasParameter _:param1 .
3 _:param1 rdfs:label "minImageSize" .
4 _:param1 rdf:value "1024" .
5
6 # The IrisReasoner can fire as soon as 1 input is ready

```

```
7 _:ir larkc:hasInputBehaviour 1 .
```

Listing 4.6: Declaration of plug-in instance specific parameters.

## 5 PLUG-IN DEVELOPER GUIDE

It is very important for LarKC that everybody can create plug-ins for the platform. To ease the process of plug-in development, a maven archetype was created. The next section explains how to use this archetype followed by a detailed description how a plug-in is built and how to integrate a created plug-in into the LarKC platform.

### 5.1 Use of the LarKC's Maven Archetype

#### 5.1.1 Using a Command Line Tool

To generate a LarKC plug-in the generate goal of archetype is called. To use the LarKC plug-in archetype we need to specify the archetypeCatalog parameter. Change to your workspace directory and issue the following command (please be careful if doing copy-paste, that you copy full command):

```
mvn archetype:generate -DarchetypeCatalog=http://larkc.svn.sourceforge.net/  
svnroot/larkc/trunk/larkc-plugin-archetype/
```

This may take a while since maven needs to download the necessary plug-ins and resolve all dependencies. After this is done you will be prompted to choose which archetype to use. The output should look as in Listing 5.1.

```
1  $ mvn archetype:generate -DarchetypeCatalog=http://larkc.svn.sourceforge.net/  
    svnroot/larkc/trunk/larkc-plugin-archetype/  
2  [INFO] Scanning for projects...  
3  [INFO] Searching repository for plugin with prefix: 'archetype'.  
4  [INFO]  
    -----  
5  [INFO] Building Maven Default Project  
6  [INFO]    task-segment: [archetype:generate] (aggregator-style)  
7  [INFO]  
    -----  
8  [INFO] Preparing archetype:generate  
9  [INFO] No goals needed for project - skipping  
10 [INFO] [archetype:generate {execution: default-cli}]  
11 [INFO] Generating project in Interactive mode  
12 [INFO] No archetype defined. Using maven-archetype-quickstart (org.apache.  
    maven.archetypes:maven-archetype-quickstart:1.0)  
13 Choose archetype:  
14 1: http://larkc.svn.sourceforge.net/svnroot/larkc/trunk/larkc-plugin-archetype  
    / -> larkc-plugin-archetype (Creates an empty LarKC plug-in; useful if you  
    want to  
15                                     develop your own LarKC plug-ins.)  
16 Choose a number: :
```

Listing 5.1: Select the maven archetype.

Only one archetype should be present: larkc-plugin-archetype. Choose larkc-plugin-archetype by entering 1 and confirm by pressing **return**. If there are multiple versions of the archetype present, select the newest one. The output is shown in Listing 5.2.

```
1  Choose a number: : 1  
2  [INFO] Using property: groupId = eu.larkc  
3  [INFO] Using property: artifactId = plugin.myplugin  
4  [INFO] Using property: version = 0.0.1-SNAPSHOT  
5  [INFO] Using property: package = eu.larkc.plugin.myplugin  
6  [INFO] Using property: larkcPluginName = MyPlugin
```

```

7 Confirm properties configuration:
8 groupId: eu.larkc
9 artifactId: plugin.myplugin
10 version: 0.0.1-SNAPSHOT
11 package: eu.larkc.plugin.myplugin
12 larkcPluginName: MyPlugin
13 Y:

```

Listing 5.2: Create plug-in with the maven archetype.

Press **return** again to create a plug-in using the default properties. By doing so a plug-in skeleton named *MyPlugin* will be created.

In most cases you don't want to use the default properties. To set your own properties deny the default values by entering **n**. Confirm with **return**. You will be asked to set all required properties. To use the default value for a property simply press **return**. Listing 5.3 shows an example of custom properties to create an *IrisReasoner* plug-in.

```

1 Y: n
2 Define value for property 'groupId': eu.larkc:
3 Define value for property 'artifactId': plugin.myplugin: plugin.reasoner.iris
4 Define value for property 'version': 0.0.1-SNAPSHOT:
5 Define value for property 'package': eu.larkc.plugin.myplugin: eu.larkc.plugin
  .reasoner.iris
6 Define value for property 'larkcPluginName': MyPlugin: IrisReasoner

```

Listing 5.3: Create an IrisReasoner with the maven archetype.

Confirm the properties configuration you have entered to create a plug-in skeleton using these specific properties. If all went well you will see an output like in Listing 5.4.

```

1 [INFO]
  -----
2 [INFO] BUILD SUCCESSFUL
3 [INFO]
  -----
4 [INFO] Total time: 3 seconds
5 [INFO] Finished at: Mon Mar 21 13:43:26 CET 2011
6 [INFO] Final Memory: 14M/164M
7 [INFO]
  -----

```

Listing 5.4: Successfull creation of the IrisReasoner.

The generated plug-in can now be edited to your likings. Every plug-in skeleton is a valid maven project and can thus be build using the standard maven procedure.

### 5.1.2 Using a Graphical Eclipse Wizard

Developing LarKC plug-ins is the same as developing any Java application, with a few special rules. This section presents the easiest way to start developing LarKC plug-ins, which is using the wizard plug-in for Eclipse. The plug-in can be downloaded from SourceForge<sup>1</sup>.

Once you have downloaded the Eclipse plug-in, it has to be copied into the */plugins/* subdirectory of your Eclipse installation. After restarting Eclipse, a LarKC icon will be displayed in the toolbar (see Figure 5.1).

<sup>1</sup><http://sourceforge.net/projects/larkc/files/>

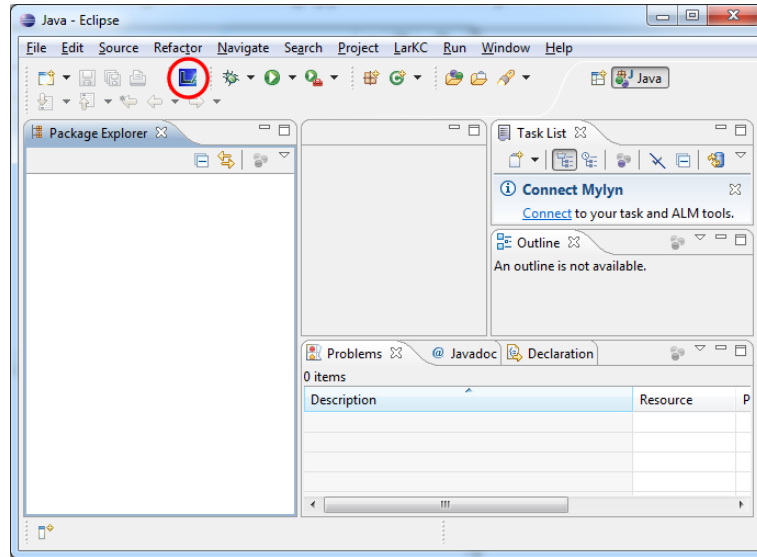


Figure 5.1: Eclipse with LarKC plug-in wizard installed.

To create a new LarKC plug-in using the wizard, you first have to select your LarKC platform directory by clicking on the *LarKC icon* in the toolbar (Figure 5.2).

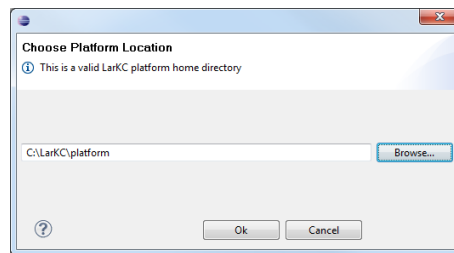


Figure 5.2: Choose platform location.

Then select the *Other* option from the *File - New - Other* menu (Figure 5.3)



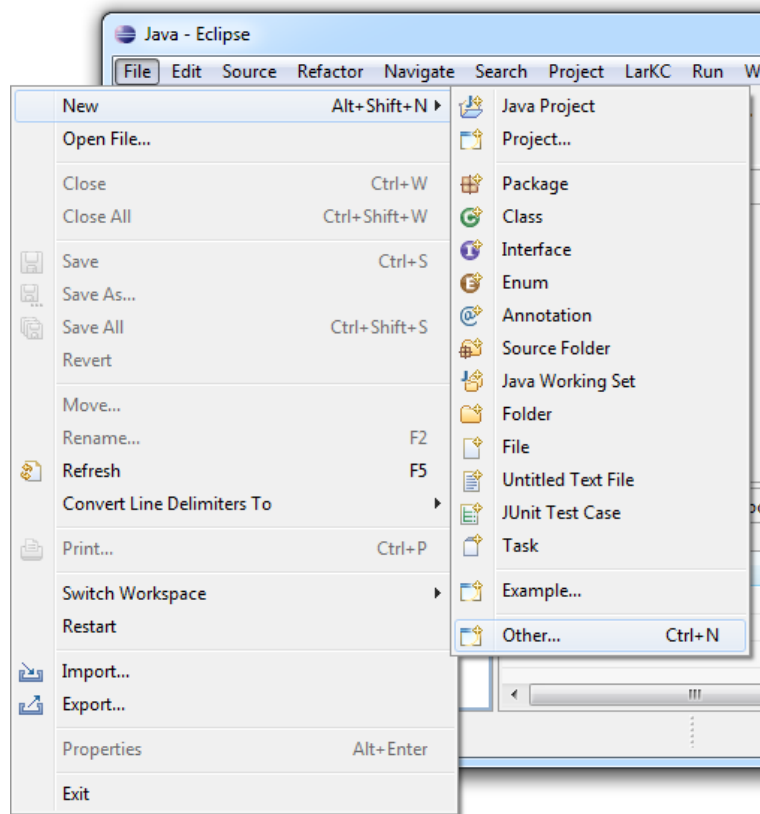


Figure 5.3: Creating a new LarKC plug-in project.

Then select the *LarKC Plugin* option from the LarKC directory (Figure 5.4).

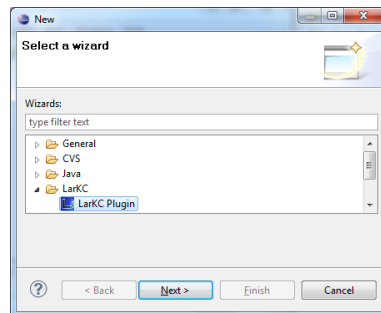


Figure 5.4: Creating a new LarKC plug-in.

In the dialog that follows, write the name of your plug-in and select the plug-in type (Figure 5.5).

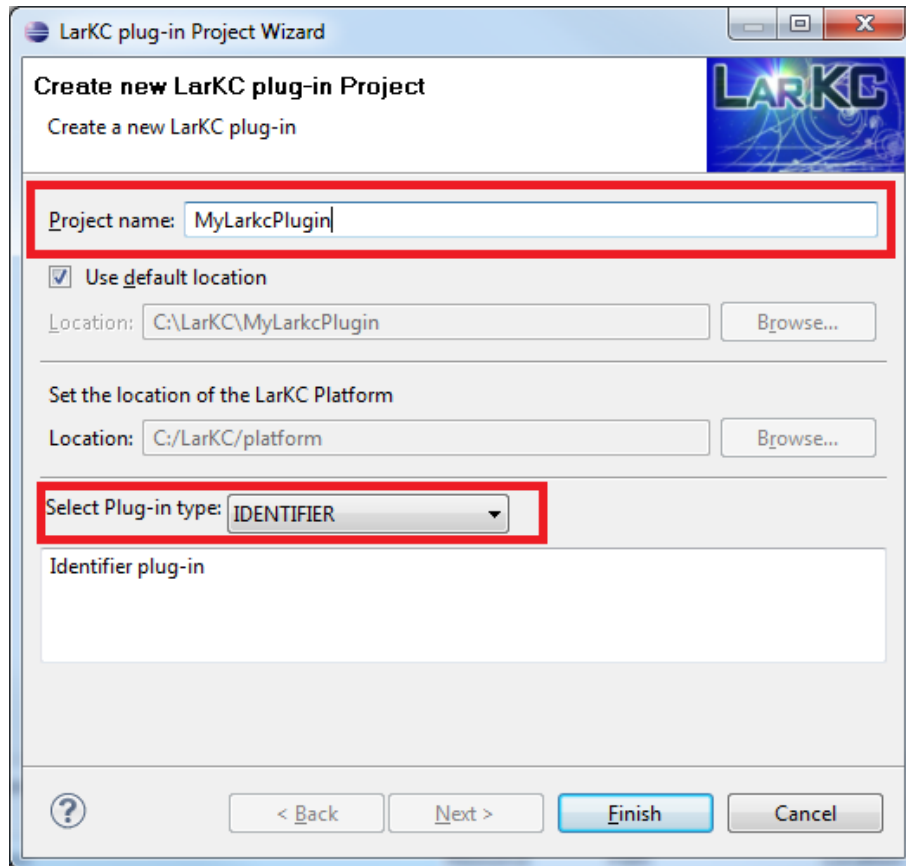


Figure 5.5: Select your plug-in type.

After clicking *Finish*, the wizard should generate the plug-in template and you can start adding some code (Figure 5.6).

When you have finished developing your plug-in and if you would like to test it, right-click on your plug-in project and select (*Run As - Run Configurations...*) Then select *LarkC Plugin* and click *Run*. This command will start your LarkC Platform together with your plug-in (Figure 5.7).

## 5.2 Importing the Generated Project to Eclipse IDE

A very convenient way to develop plug-ins is to import the generated plug-in skeleton in eclipse. You can do that as usual via the *File->Import* dialog. Choose Existing Maven Projects or Existing Projects into Workspace and select the previously generated plug-in skeleton, which is in the workspace directory where the mvn command was run. Make sure that you also have the LarkC platform as a project in your workspace, since every LarkC plug-in is dependent on the platform.

When using the Graphical Eclipse Wizard, this step is not needed since the import is done automatically.

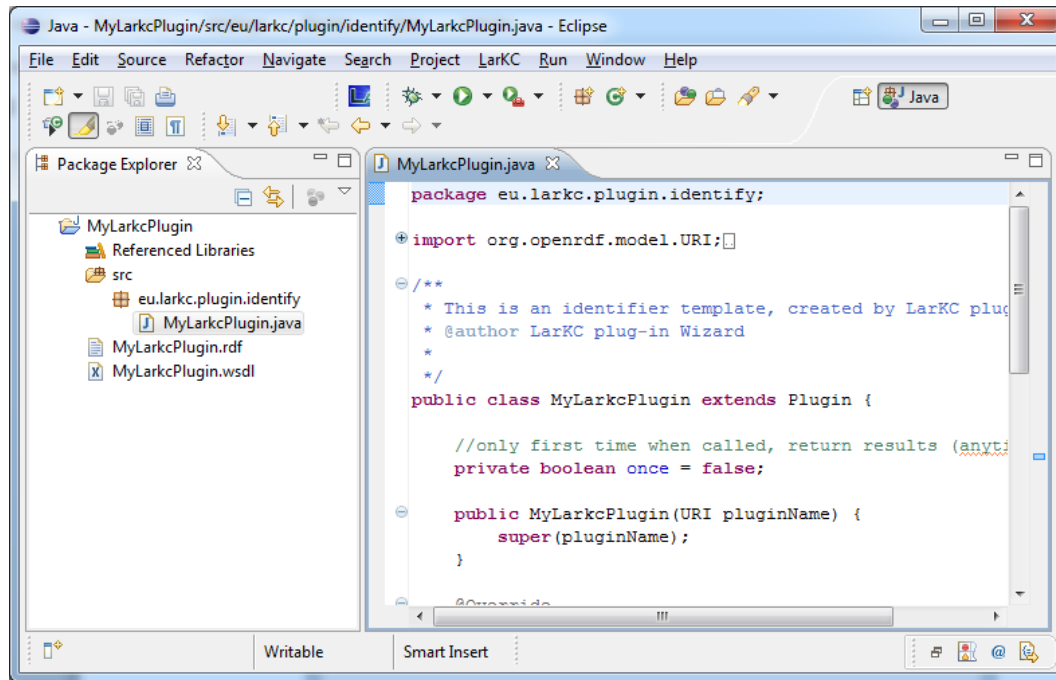


Figure 5.6: The LarKC plug-in template in Eclipse.

### 5.3 Building and Compiling the Plug-in with Maven

Since plug-ins are maven projects, they can be build via `mvn install`. In a terminal simply run `mvn install` in the plug-in directory. Alternatively, in eclipse (with m2eclipse installed) there is a predefined run configuration: *Run As -> Maven install*

If the build is successful a jar file will be created in the projects target folder and automatically copied to `../platform/plugins`.

If your plug-in does depend on any special plug-ins which are not deployed in a maven repository you can use the maven assembly goal to include them in your plug-in file. Place any libraries you wish to include with your plug-in in the `lib/` folder of your plug-in and run `mvn assembly:assembly` (eclipse: *Run As -> Maven assembly:assembly*).

### 5.4 Inside the Plug-in

As in the previous version of LarKC (v2.0 - D5.4.2), in the new version all five types of plug-ins extend the same plug-in interface (`eu.larkc.plugin.Plugin`). The only difference between the Identifier, Transformer, Selector, Reasoner and Decider is conceptual and in the plug-in description files (`.wsdl` and `.rdf`).

Internally, the new LarKC plug-ins are almost identical to the previous version, except that the method `onMessage()` is no longer needed. This means the plug-ins have to override the *constructor with URI parameter*, `initialiseInternal()`, `invokeInternal()` and `shutdownInternal()` methods. The skeleton of a new plug-in looks like this:

```
1 package eu.larkc.plugin.select;
2
3 import org.opencyc.cycobject.CycConstant;
```

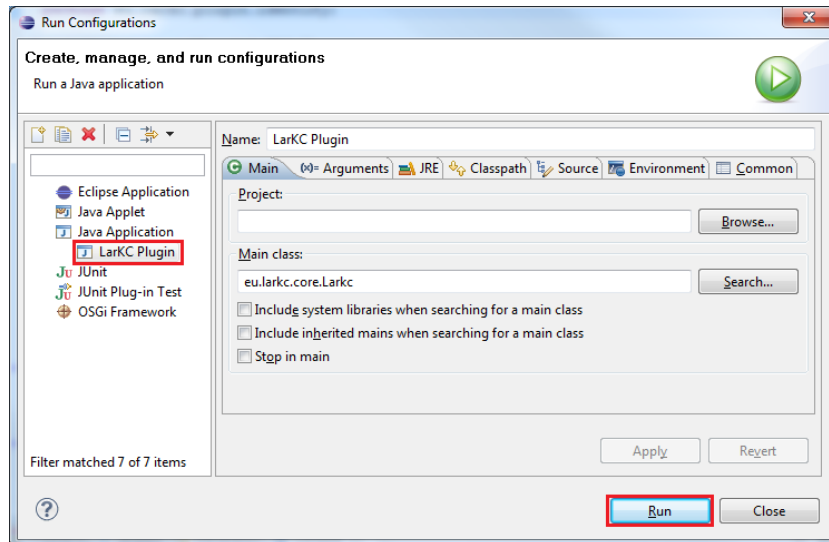


Figure 5.7: Test your own plug-in with the LarKC Platform.

```

4 import org.opencyc.cycobject.Guid;
5 import org.openrdf.model.URI;
6
7 import javax.jms.Message;
8
9 import eu.larkc.core.data.SetOfStatements;
10 import eu.larkc.plugin.Plugin;
11
12 public class ExamplePlugin extends Plugin {
13
14     public ExamplePlugin(URI pluginName) {
15         super(pluginName);
16     }
17
18     @Override
19     protected void initialiseInternal(SetOfStatements workflowDescription) {
20     }
21
22     public SetOfStatements invokeInternal(SetOfStatements input) {
23         return null;
24     }
25
26     @Override
27     public void shutdownInternal() {
28     }
29 }

```

### Method *onMessage*

This method is called when the plug-in gets a control message from a Decider plug-in or the platform (i.e., the Executor can send out and retrieve messages in order to observe and control the behaviour during runtime).

### Method *initialiseInternal*

This method is called at the beginning of the workflow creation, when the plug-in is instantiated. It should be used to handle the initialisation of the plug-in before it is actually called from the platform.

### Method *invokeInternal*

This is the main method of the plug-in, which is called each time that some other plug-in or the platform requests some data from it.

### Method *shutdown*

The shutdown method is called when the workflow is destroyed by the user and should be used to clean-up some data, if necessary.

LarKC plug-ins accept arguments and return results as sets of RDF statements. These statements contain all meta-data required to invoke the plug-in. By default, LarKC plug-ins will cache their results. This means that calling a plug-in with the same set of RDF statements will not invoke the result, but will return the result calculated on a previous invocation. For some plug-ins, this caching mechanism is not appropriate, namely for plug-ins that return results that depend on external events. In these cases, the caching mechanism can be extended or deactivated by overriding the relevant methods from the Plug-in class.

## 5.4.1 Basic Data Layer Functionality

The Data Layer is a core platform component that supports plug-ins and applications with respect to storage, retrieval, and light-weight inference on top of large volumes of data. The Data Layer has following features:

- Persistence storage to RDF data;
- Reference implementation of ORDI data model;
- Passing data by value or reference;
- Resolvable RDF data identifier;
- Retrieval data exposed via standard SPARQL endpoints;
- Configurable forward-chaining reasoning OWL2-RL;
- Interfaces that allow streaming data processing;
- Utility methods to query remote data or multiple datasets;
- Cluster configuration that improves resilience and provide scalable query answering.

In a cluster configuration, there are two types of nodes: masters and workers. Masters act as the gateway to the cluster and all read/write requests go through these nodes. A cluster can have more than one master node, but only one is allowed to operate in read/write mode. The other master nodes operate in read-only mode, otherwise known as hot-standby. They can be used for marshalling read requests and can take over handling updates if the current read/write master fails (Figure 5.8).

During normal operation, a master node will keep track of the size of each worker's read request queue, such that each read request is sent to the worker with the shortest read queue. Update requests are handled differently. First of all, the update is tested against a single worker node. If the update is successful and subsequent consistency checks pass then the update request is considered safe and is passed to the rest of the worker nodes. Master nodes take additional care

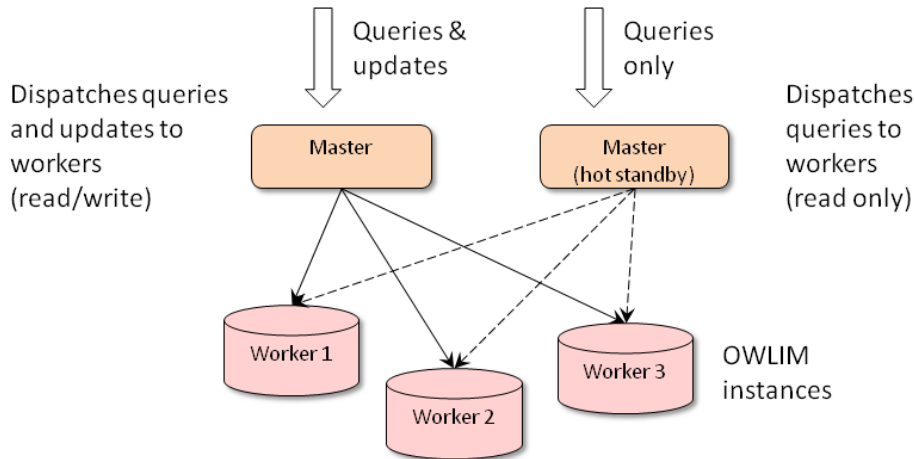


Figure 5.8: OWLIM Cluster configuration.

to ensure that the states of all worker nodes are properly synchronised and if an anomaly is detected, the problem worker node is released from the cluster. The monitor and control JMX interface can be used to return worker nodes to the cluster and initiate their synchronisation. In the event of a failure of a worker node, the performance degradation is graceful with respect to the number of healthy workers. The cluster can remain operational with just a single worker node.

### 5.4.2 Caching Techniques

By default, the LarKC platform caches the results of plugin invocations. The general contract is that if a plugin instance is called with the same parameters (input RDF), it will produce the same results. In this regard, the LarKC workflow can be seen as a data dependency graph, where plugins are only invoked if the output of their parent plugins has changed.

The default caching behavior of LarKC can be overridden, or disabled altogether to suit the needs of specific plugins. Such plugins may depend on external data, which is out of the control of LarKC.

To disable caching altogether for a plugin override the *Plugin.cacheLookup()* and *Plugin.cacheInsert()* methods and make them empty.

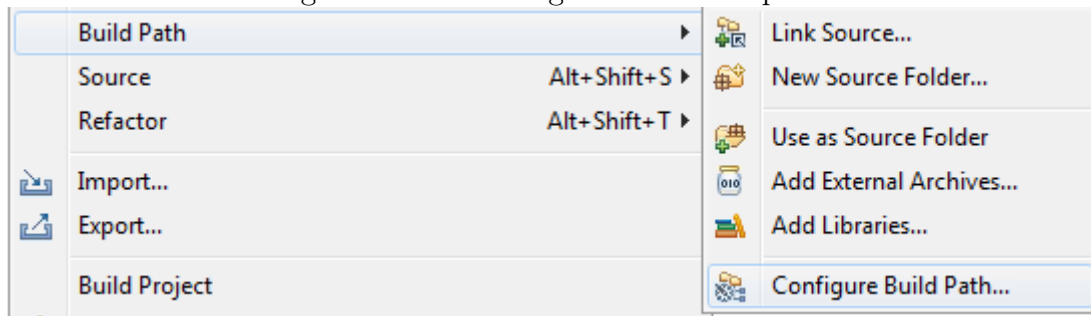
Sometimes, not the entire input needs to be considered for caching, or some other custom behavior is required. The method *Plugin.getInvocationKey()* is used to determine which part of the input is used for caching. The plugin developer may override the default functionality to suit a particular plugin.

If any more specific functionality is required (for example, adding expirations), it can be implemented by overriding the *Plugin.cacheLookup()* and *Plugin.cacheInsert()* methods or by extending the object in the *Plugin.cache* field.

### 5.4.3 Writing Tests

During the generation of the plug-in with LarKC's maven archetype the folder *src/test/java* is created. It contains a package with the name defined during the creation of the plug-in (in the previous example it is *eu.larkc.plugin.reasoner.iris*). There you can find one class (in our example *IrisReasonerTest.java*) where all

Figure 5.9: Accessing the Build Properties



JUnit tests can be placed. It already implements the body of one test to ease the use.

## 5.5 Integration into the Platform

In order to register and test the plug-in on the platform it must somehow accessible to the platform. This can be done either by making it available under the *PLATFORM\_ROOT/plugins* folder or the LarkKC platform should be run with the *-pluginindir=pluginClassesPath*.

Usually the first option is better for the development since it allows registering the plug-in being developed together with other available plug-ins. If using Eclipse, this can be done by pointing the *Default output folder* option to the *PLATFORM\_ROOT/plugins/MyPlugin* folder.

To do this, right click on the newly created plug-in project and select *Build Path-Configure Build Path* option as can be seen on Figure 5.9 and in the dialog that opens click *Browse* next to *Default output folder:* option and point it to the location of the Platform (Figure 5.10).

### 5.5.1 Using the Plug-in

After successful integration of the plug-in in the platform it should be tested. The first step is to create a workflow using the newly created plug-in. If we assume the type of our new plug-in is `<urn:eu.larkc.plugin.myplugin.MyPlugin>` an example workflow is shown in Listing 5.5.

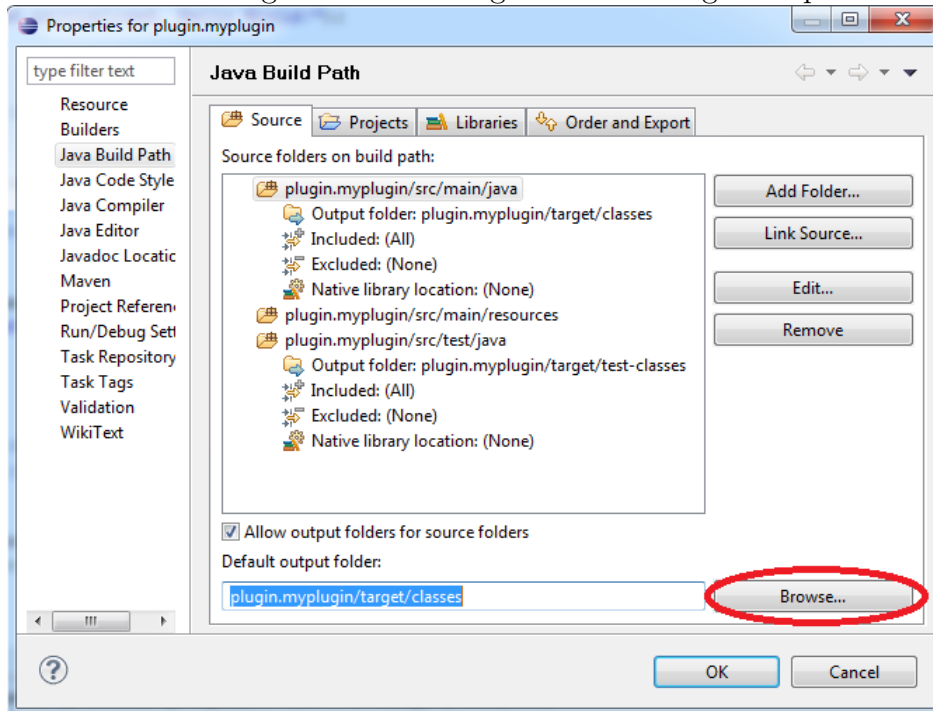
```

1  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
2  @prefix larkc: <http://larkc.eu/schema#> .
3
4  # Define two plug-ins
5  _:plugin1 a <urn:eu.larkc.plugin.myplugin.MyPlugin> .
6  _:plugin2 a <urn:eu.larkc.plugin.myplugin.MyPlugin> .
7
8  # Connect the plug-ins
9  _:plugin1 larkc:connectsTo _:plugin2 .
10
11 # Define a path to set the input and output of the workflow
12 _:path a larkc:Path .
13 _:path larkc:hasInput _:plugin1 .
14 _:path larkc:hasOutput _:plugin2 .
15
16 # Connect an endpoint to the path

```



Figure 5.10: Setting the Build Target Properties



```
17 <urn:myTestEndpoint> a <urn:eu.larkc.endpoint.test> .
18 <urn:myTestEndpoint> larkc:links _:path .
```

Listing 5.5: An example workflow using the new plug-in.

You can initialize this workflow via the management interface (see Section 3.4). After starting LarKC point your browser to <http://localhost:8182/> and submit your workflow there. If the workflow creation was successful, the ID of the workflow will be returned (e.g. `e39d48e0-03c9-4d81-9e0a-7bc4fda7095b`). Use this ID to retrieve the endpoint of your workflow.

The plug-in's constructor and *initialiseInternal* method will be called on workflow creation.

After successful workflow creation, use the following URL to get an endpoint for the specific workflow:

*HTTP GET* `/workflow/WID/endpoint?urn=urn`

Where the *WID* is the unique ID of your workflow (as returned by successful workflow creation) and *urn* is the endpoint urn (e.g. `<urn:myTestEndpoint>`). The complete command is as follows:

*HTTP GET* `http://localhost:8182/workflow/135cc37f-b6e5-490c-8f5b-8271e8d20e17/endpoint?urn=urn:myTestEndpoint`

The URL of your endpoint will be returned (e.g. `http://localhost:8183/testendpoint`). Start your workflow by sending a HTTP POST request to the endpoint of your workflow. The following command is using *curl* to send a POST request to the endpoint:



```
curl -d "query=SELECT * WHERE ?s ?p ?o" http://127.0.1.1:8183/testendpoint
```

This will start your workflow. As soon as your plug-in receives its first input the *invokeInternal* method will be called.

## 6 USER SUPPORT

A number of tools have been setup as part of the LarKC development environment to support the different kinds of LarKC users.

### 6.1 Joining LarKC@SourceForge.net

The only prerequisite for joining [LarKC@SourceForge.net](http://sourceforge.net/projects/larkc)<sup>1</sup> is to have a valid SourceForge.net account which can be obtained on the SourceForge [registration page](#). For getting access to all the LarKC features and services, please send your account information to the project [administrator](#)<sup>2</sup>.

### 6.2 User and Developer Support

The goals of the LarKC project include, among others, the release of a number of open-source software components, under Apache 2.0 license (see section 8 for more details). The user feedback is significant for the LarKC development, as it is an important source for improvements. In order to encourage a community of users to get involved in the LarKC development, collect user improvement proposals, solve appearing issues rapidly and to promote the discussion within the user community as well as between users and developers, the following services are provided by [LarKC@SourceForge.net](http://sourceforge.net/projects/larkc)

**The User Forum** where discussions should take place about using LarKC with implemented use cases (and corresponding components), requesting new use cases to be implemented by LarKC, etc. The User Forum is accessible through the “Develop/Forums/LarKC-Users” Tab on LarKC@sf.net [main page](#)<sup>3</sup> or through the [forums page](#)<sup>4</sup>.

**The User Support Mailing List**<sup>5</sup> is used in addition to the User Forum with automatic message redirection from/to the User Forum. The main LarKC developers are subscribed to the Mailing List and provide a prompt feedback to user support requests. In order to join the mailing list, a [subscription](#)<sup>6</sup> is required.

Besides the forums the LarKC@SourceForge.net website offers the possibility of tracking bugs, feature requests and support requests. These “tracker pages” allow for example the addition of a feature request or the submission of a bug report. See figure 6.1 for a screenshot of the bug-tracker page.

For developers the following support options are available:

**The Developers Forum** here discussions should take place about implementing new LarKC components (including plug-ins, workflows and other possible

---

<sup>1</sup><http://sourceforge.net/projects/larkc>

<sup>2</sup><mailto:larkc-contact@lists.sourceforge.net>

<sup>3</sup><https://sourceforge.net/projects/larkc/develop>

<sup>4</sup><https://sourceforge.net/projects/larkc/forums>

<sup>6</sup><https://lists.sourceforge.net/lists/listinfo/larkc-user-support>

SourceForge.net > Projects > Large Knowledge Collider > Tracker > Bugs > Browse Tracker Items

Large Knowledge Collider Share

Summary | Files | Support | Develop | Hosted Apps | **Tracker** | Mailing Lists | Forums | Code

[Add new](#) | [Browse](#)

Tracker: Bugs

Search:   [Advanced](#) Options [RSS](#)

Page: 1 1 - 16 of 16 Results - Display 25

ID	Summary	Status	Opened	Assignee	Submitter	Resolution	Priority
3117444	Problems loading external libraries	Closed	2010-11-24	<a href="#">bradeskojest</a>	<a href="#">hpcassel</a>	Works For Me	7
3117391	Fix junit test case testSparqlQueryHandler in core.endpoint	Open	2010-11-24	<a href="#">bradeskojest</a>	<a href="#">ch_fuchs</a>	None	5
3116692	Utility methods for serialization/deserialization of DL type	Open	2010-11-23	<a href="#">bradeskojest</a>	<a href="#">vassil_momtchev</a>	None	5
3116689	Ensure proper shutdown of the platform	Open	2010-11-23	<a href="#">vassil_momtchev</a>	<a href="#">vassil_momtchev</a>	None	5
3101582	Delete "active-mq" entry after the execution	Closed	2010-11-02	<a href="#">hpcassel</a>	<a href="#">hpcochep</a>	Fixed	1
3029859	Can not invoke LarKC using openrdfjena	Open	2010-07-15	<a href="#">vassil_momtchev</a>	<a href="#">dellaglio</a>	None	5
3016534	Wrong SVN URL in Release Notes	Deleted	2010-06-15	nobody	<a href="#">markagreenwood</a>	None	5
3016501	Wrong SVN URL in Release Notes	Closed	2010-06-15	nobody	<a href="#">markagreenwood</a>	None	5
2985403	When wsdl file or rdf file is not correct it should report	Open	2010-04-11	<a href="#">bradeskojest</a>	<a href="#">bradeskojest</a>	None	5
2985284	Distributed version throws exception on default sparql query	Open	2010-04-11	<a href="#">hpcochep</a>	<a href="#">mwilbrock</a>	None	5

Assignee:  Status:  Category:  Group:  Submitter:  Keyword:  Artifact:

ID:

Figure 6.1: LarKC bug-tracker.

components) or improving/modifying existing ones. The Developer Forum is accessible through the “Develop/Forums/LarKC-Developers” Tab on the LarKC@sf.net’s [main page](#) or through the [forums page](#). See figure 6.2 for a screenshot of the Developer Forum.


**The Developers Support Mailing List<sup>7</sup>** is used in addition to the Developers Forum with automatic message redirection from/to the Developers Forum. In order to join the mailing list, a [subscription<sup>8</sup>](#) is required.

## 6.3 Contact us

For getting access to the [LarKC@SourceForge.net<sup>9</sup>](#) or in case of problems with using its services, please contact the project [administrator](#).

<sup>8</sup><https://lists.sourceforge.net/lists/listinfo/larkc-dev-support>

<sup>9</sup><http://sourceforge.net/projects/larkc>


FIND AND DEVELOP OPEN SOURCE SOFTWARE

[Register](#)
[Log In](#)

[Find Software](#)
[Develop](#)
[Create Project](#)
[Blog](#)
[Site Support](#)
[About](#)

[SourceForge.net](#) > [Projects](#) > [Large Knowledge Collider](#) > [Forums](#) > [LarkC-Developers](#)

[Share](#)






## Large Knowledge Collider

[Summary](#)
[Files](#)
[Support](#)
[Develop](#)
[Hosted Apps](#)
[Tracker](#)
[Mailing Lists](#)
[Forums](#)
[Code](#)

[LarkC-Developers](#)
[Monitor](#)

The developer support forum

[LarkC-Developers](#)
[LarkC-Users](#)
[Statistics for Forums](#)
[Search Forums](#)

Topic	Replies	Started By	Last Action	Options
<a href="#">Closeable iterator losing variable bindings?</a>	1	tonganqn	2010-06-03 11:12:27 UTC	<a href="#">Monitor</a> 
<a href="#">TREE exception in simple pipeline</a>	2	tonganqn	2010-06-03 08:27:59 UTC	<a href="#">Monitor</a> 
<a href="#">Runtime error accessing conf dir</a>	0	mwitbrock	2010-04-05 02:48:28 UTC	<a href="#">Monitor</a> 
<a href="#">Checkout for Platform fails in Netbeans</a>	2	mwitbrock	2010-04-04 19:50:07 UTC	<a href="#">Monitor</a> 
<a href="#">Support for IDEs</a>	0	mwitbrock	2009-10-18 03:24:16 UTC	<a href="#">Monitor</a> 

[< Previous](#) | 1 | [Next >](#)

Jump To:

### Add a Topic

This forum does not allow anonymous participation.

[Log in](#) to add a topic. Not registered? [Create an account](#) to participate and receive email updates when replies are posted to this topic.

Figure 6.2: LarkC developers forum.

## 7 THE LARKC PLUG-IN MARKETPLACE

The recently introduced LarKC Plug-in Marketplace<sup>1</sup> is the first address for developers and users of LarKC workflows when searching for a particular plug-in type, respectively, looking for a plug-in to solve a specific task. Furthermore, the marketplace helps to increase the number of plug-ins developed by people from outside the LarKC consortium as they may easily make their work publicly available and known to the LarKC community.

As can be seen from Figure 7.1 all previously developed LarKC Plug-ins have already been published on the marketplace. Hence, interested people can immediately gain information about the type and purpose of the plug-in (a short description of the plug-in’s main functionality is provided by moving the mouse over the “i” symbol), download the code for reusing the plug-in within own workflows, or even contact the respective developer.
























































































Plug-In Marketplace						
Plug-ins						
Name ▾	Platform Version	Type	Description	www	Download	Contact person
Base-Line Full-Text Search Selector	1.0	select				
CRION reasoner	Alpha release	reason				
DIGReasoner	1.0	reason				
EventIdentifier	1.0,1.1	identify				
GWADecider	1.1	decide				
GWASIdentifier	1.1	identify				
GWASQueryTransformer	1.1	transform				
Information Retrieval Selector	1.0	select				
Interest-Based Reasoner	Alpha Release	reason				
Interest-Based Selector	1.0	select				
Keyphrase Selector	1.0	select				
OWLAPI Reasoner	1.0	reason				
PION Reasoner	1.0	reason				
RDF2MatrixTransformer	1.0	transform				
Random Indexing Reasoner	1.0	reason				
RandomIndexingDecider	1.1	decide				
RandomIndexingIdentifier	1.1	identify				
RandomIndexingTransformer	1.1	transform				
SPARQL Query Evaluation Reasoner	1.0	reason				
SPARQLDL Reasoner	1.0	reason				
SparqlToCityQueryTransformer	1.0,1.1	transform				
Spreading Activation Selector	1.0	select				

Figure 7.1: Available plug-ins listed on the LarKC Plug-in Marketplace.

In order to commit a plug-in to the marketplace, people should use the respective form, which can be found below the actual plug-in’s overview (see Figure 7.2),

<sup>1</sup><http://www.larkc.eu/plugin-marketplace/>

and enter all relevant information into the appropriate fields. Once the data has been submitted, technical people from the LarKC consortium will carefully check the plug-in's code and all its relevant descriptions (i.e., readme and license files, wsdl and rdf description files). If it turns out that everything is properly described and compatible with the indicated version, the submission will be approved and the plug-in finally becomes visible on the marketplace. If this is not the case, LarKC will contact the respective developer again to explain him/her the reason for rejection so that he/she can readapt the plug-in “pack” again.

### Submit a Plug-in

“ Please, make sure that your plug-in “pack” contains all necessary elements (within the Download link specified below):

- **plug-in description files:** WSDL, annotation (RDF-file), readme, license
- **test suite**, including input data and expected output data, and basic tests
- **compatibility with any valid platform version** (specified in the Platform Version field below),  
i.e. plug-in API, naming, structure (plug-in name, lib – for storing the libs, src – the source code, ...)
- **external libraries** (if required)

**Plug-Ins first need approval before they are shown on the website! Thanks for your understanding.**

Platform Version (use CTRL to select multiple items)	<div>Alpha Release</div> <div>1.0</div> <div>1.1</div> <div>2.0</div>	(required)
Plug-in name	<input type="text"/>	(required)
Plug-in type	<div>identify</div> <div></div>	(required)
Plug-in description	<div></div>	(required)
Plug-in website	<input type="text"/>	
Download link	<input type="text"/>	(required)
Contact person's name	<input type="text"/>	(required)
Contact person's email address	<input type="text"/>	(valid email required)

Submit

Figure 7.2: Plug-in submission form.

## 8 LICENSING

This statement summarises the decisions taken by the consortium and documented in the LarKC deliverable D9.4 [?]:

The LarKC Platform and its source code will be released under the **Apache License, Version 2.0**.

*LarKC plugins, and other code produced by the LarKC consortium may be released under any license which does not have the effect of imposing any copyleft requirement whatsoever on code to which it is linked, connected, or otherwise associated. For the purposes of clarity, "copyleft requirement" here signifies a requirement to license such associated code under a license identical or similar to the license asserting the copyleft requirement. Licenses that are acceptable for release of code produced for the LarKC project include, but are not limited to: the Apache License Version 2.0, the BSD License, the MIT License, the LGPL and the Creative Commons Attribution License without the Share Alike feature. Licenses that are unacceptable for release of code produced for the LarKC project include, but are not limited to: any version of the GNU General Public License (GPL).*

Reading D9.4 carefully, the only acceptable license for code that is required for the operation of the Platform is Apache 2.0. This choice was carefully made, and was designed to minimise the likelihood that organisational policies would prevent coders from working with the Platform code. The cost of this, which we have chosen to pay, is that there are some potentially useful packages that we cannot use to build the Platform. A wider variety of licenses (including LGPL but NOT GPL) may be used for plug-ins and other non-platform code. Note that this does not prevent third parties from producing plug-ins using e.g. GPL, but it does mean that the LarKC project will never distribute such plug-ins.

## 9 OPEN ISSUES AND FUTURE WORK

During the next project period, work will continue, among others, on the following topics:

- Integrate the Plug-in Registry with LarKC Plug-in Marketplace aiming at simplifying integration of plug-ins into workflows.
- Parallelize plug-in support (for both MapReduce and MPI realizations).
- Modularize the LarKC core architecture, e.g. making workflow end-points pluggable.
- Enable more advanced features for monitoring LarKC workflows and plug-ins (i.e., agents that connect with other platform components in order to exchange relevant logging information).
- Provide better community support materials, advanced documentation and extend the number of support plug-in in the marketplace.



## 10 CONCLUDING REMARKS

The implemented LarKC architecture evolved from a simple linear pipeline, at the beginning of the project, towards a flexible workflow concept, with the corresponding support components for composing and managing it. The current LarKC Platform has been designed as a flexible pluggable architecture, which enables design and testing of Semantic Web applications. The platform aims to be open, flexible and scalable but, at the same time, to fulfill the requirements imposed by the use cases, which will potentially make use of it (e.g. Life Science, Cancer Research and so forth). Therefore, it must allow a trade off between flexibility and performance, in order to achieve a good balance between generality and applicability. For this purpose, the platform enables the integration of loosely coupled pieces (plug-ins, the data layer and other platform support components), which are well integrated and interconnected through well-defined interfaces.

## A APPENDIX

```

1 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
2 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
3 @prefix larkc: <http://larkc.eu/schema#> .
4
5 # -----
6 # Define the plug-ins used in this workflow
7
8 _:gi a <urn:eu.larkc.plugin.identify.google.GoogleSearchIdentifier> .
9 _:gii a <urn:eu.larkc.plugin.identify.google.images.
    GoogleImageIdentifier> .
10 _:fii a <urn:eu.larkc.plugin.identify.flickr.images.
    FlickrImagesIdentifier> .
11
12 _:rp1 a <urn:eu.larkc.plugin.parser.ResultParser> .
13 _:rt1 a <urn:eu.larkc.plugin.transformer.ResultTransformer> .
14 _:rp2 a <urn:eu.larkc.plugin.parser.ResultParser> .
15 _:rt2 a <urn:eu.larkc.plugin.transformer.ResultTransformer> .
16
17 _:ir a <urn:eu.larkc.plugin.reasoner.iris.IrisReasoner> .
18
19 _:rf a <urn:eu.larkc.plugin.filter.ResultFilter> .
20 _:irf a <urn:eu.larkc.plugin.filter.image.ImageResultFilter> .
21
22 # -----
23 # Path 1
24
25 # Connect the plug-ins
26 _:gi larkc:connectsTo _:rp1 .
27 _:rp1 larkc:connectsTo _:rt1 .
28 _:rt1 larkc:connectsTo _:ir .
29 _:ir larkc:connectsTo _:rf .
30
31 # Define a path and set the input and output of the path
32 _:path1 larkc:hasInput _:gi .
33 _:path1 larkc:hasOutput _:rf .
34
35 # Connect an endpoint to the path
36 <urn:eu.larkc.endpoint.test.ep1> a <urn:eu.larkc.endpoint.test> .
37 <urn:eu.larkc.endpoint.test.ep1> larkc:links _:path1 .
38
39 # -----
40 # Path 2
41
42 # Connect the plug-ins
43 _:gii larkc:connectsTo _:rp2 .
44 _:fii larkc:connectsTo _:rp2 .
45 _:rp2 larkc:connectsTo _:rt2 .
46 _:rt2 larkc:connectsTo _:ir .
47 _:ir larkc:connectsTo _:irf .
48
49 # Define plug-in specific parameters
50 _:gii larkc:hasParameter _:param1 .
51 _:param1 rdfs:label "minImageSize" .
52 _:param1 rdf:value "1024" .
53

```

```
54 # The IrisReasoner can fire as soon as 1 input is ready
55 _:ir larkc:hasInputBehaviour 1 .
56
57 # Define a path and set the input and output of the path
58 _:path2 larkc:hasInput _:gii .
59 _:path2 larkc:hasOutput _:irf .
60
61 # Connect an endpoint to the path
62 <urn:eu.larkc.endpoint.test.ep2> a <urn:eu.larkc.endpoint.test> .
63 <urn:eu.larkc.endpoint.test.ep2> larkc:links _:path2 .
```

Listing A.1: A more complex workflow consisting of ten plug-in instances with two paths and two endpoints