LarKC

The Large Knowledge Collider:
a platform for large scale integrated reasoning and Web-search

FP7 – 215535

LarKC Platform Manual – V1.1

Coordinator: Matthias Assel, HLRS
With contributions from: Alexey Cheptsov, Georgina Gallizo, HLRS; Luka Bradesko, CycEur; Vassil Momtchev, Onto; Christoph Fuchs, Norbert Lanzanasto, UIBK

Document Identifier:	LarKC/2008/Platform-Manual/v1.1
Class Deliverable:	LarKC EU-IST-2008-215535
Version:	1.1
Date:	05.07.2010
State:	Final
Distribution:	Public

EXECUTIVE SUMMARY

This document corresponds to the initial release (v1.1) 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 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 constitutes a complete user 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 (mailing lists and fora) have been setup, as part of the LarKC Development Environment, to support the different kinds of LarKC users.

Together with the Platform code, which is released under Apache V2.0 license, various example plug-ins and workflows are released, in order to support developers who wish to create their own components and execute them within the LarKC Platform.

Future releases of the LarKC Platform will include new and improved platform support features and a set of best practices and guidelines for LarKC users.

DOCUMENT INFORMATION

IST Project Number	FP7 – 215535	**Acronym**	LarKC
Full Title	The Large Knowledge Collider: a platform for large scale integrated reasoning and Web-search		
Project URL	http://www.larkc.eu/		
Document URL			
EU Project Officer	Stefano Bertolo		

Date of Delivery	**Actual:** 05.07.2010	
Status	version 1.1	final x
Nature	prototype x report □ dissemination □	
Dissemination level	public x consortium □	

Authors (Partner)	Matthias Assel, Alexey Cheptsov, Georgina Gallizo, HLRS; Luka Bradesko, CycEur; Vassil Momtchev, Onto; Christopn Fuchs, Norbert Lanzanasto, UIBK			
Responsible Author	**Name**	Matthias Assel	**E-mail**	assel@hlrs.de
	Partner	HLRS	**Phone**	

Abstract (for dissemination)	This document corresponds to the initial release (v1.1) 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 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 constitutes a complete user 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 (mailing lists and fora) have been setup, as part of the LarKC Development Environment, to support the different kinds of LarKC users. Together with the Platform code, which is released under Apache V2.0 license, various example plug-ins and workflows are released, in order to support developers who wish to create their own components and execute them within the LarKC Platform. Future releases of the LarKC Platform will include new and improved platform support features and a set of best practices and guidelines for LarKC users.
Keywords	LarKC Platform, manual, user guide, developer guide, plug-in

Version Log			
Issue Date	**Rev. No.**	**Author**	**Change**
28.01.2010	0.1	Georgina Gallizo	Initial skeleton
29.01.2010	0.2	Georgina Gallizo	Incorporated content from the previous "Getting started guide"
01.03.2010	0.3	Georgina Gallizo	Integration of partners' inputs to the different chapters
12.03.2010	0.4	Luka Bradesko	Finalization of "How to develop a plug-in" section
16.03.2010	0.5	Matthias Assel	Finalization of "Running examples" section with the latest status of the release
17.03.2010	0.6	Georgina Gallizo	General sanity check, ready for internal quality assessment
01.04.2010	0.7	Georgina Gallizo	Incorporating comments from quality assessment (Zhisheng Huang and Michael Witbrock) and overall sanity check
21.04.2010	0.8	Georgina Gallizo	Incorporating final feedback. Ready for submission
26.04.2010	1.0	Georgina Gallizo	Incorporate feedback from Scientific Director (Frank van Harmelen)
29.06.2010	1.1	Matthias Assel	Added new sections and updated TOC. Incorporated contributions from Christoph Fuchs, Norbert Lanzanasto, Alexey Cheptsov. Finalized document.

PROJECT CONSORTIUM INFORMATION

Participant's name	**Partner**	**Contact**
Semantic Technology Institute Innsbruck, Universitaet Innsbruck		Dieter Fensel, Semantic Technology Institute (STI), Universitaet Innsbruck, Innsbruck, Austria, E-mail: dieter.fensel@sti-innsbruck.at
AstraZeneca AB		Bosse Andersson, AstraZeneca Lund, Sweden Email: bo.h.andersson@astrazeneca.com
CEFRIEL - SOCIETA CONSORTILE A RESPONSABILITA LIMITATA		Emanuele Della Valle, CEFRIEL - SOCIETA CONSORTILE A RESPONSABILITA LIMITATA, Milano, Italy, Email: emanuele.dellavalle@cefriel.it
CYCORP, RAZISKOVANJE IN EKSPERIMENTALNI RAZVOJ D.O.O.		Michael Witbrock, CYCORP, RAZISKOVANJE IN EKSPERIMENTALNI RAZVOJ D.O.O., Ljubljana, Slovenia, Email: witbrock@cyc.com
Höchstleistungsrechenzentrum, Universitaet Stuttgart		Matthias Assel, Höchstleistungsrechenzentrum, Universitaet Stuttgart, Stuttgart, Germany, Email: gallizo@hlrs.de
MAX-PLANCK GESELLSCHAFT ZUR FOERDERUNG DER WISSENSCHAFTEN E.V.		Lael Schooler, Max-Planck-Institut für Bildungsforschung Berlin, Germany Email: schooler@mpib-berlin.mpg.de
Ontotext AD		Atanas Kiryakov, Ontotext Lab, Sofia, Bulgaria Email: naso@ontotext.com
SALTLUX INC.		Kono Kim, SALTLUX INC, Seoul, Korea, Email: kono@saltlux.com
SIEMENS AKTIENGESELLSCHAFT		Volker Tresp, SIEMENS AKTIENGESELLSCHAFT, Muenchen, Germany, E-mail: volker.tresp@siemens.com
THE UNIVERSITY OF SHEFFIELD		Hamish Cunningham, THE UNIVERSITY OF SHEFFIELD Sheffield, UK, Email: h.cunningham@dcs.shef.ac.uk

VRIJE UNIVERSITEIT AMSTERDAM		Frank van Harmelen, VRIJE UNIVERSITEIT AMSTERDAM, Amsterdam, Netherlands, Email: Frank.van.Harmelen@cs.vu.nl
THE INTERNATIONAL WIC INSTITUTE, BEIJING UNIVERSITY OF TECHNOLOGY		Ning Zhong, THE INTERNATIONAL WIC INSTITUTE, Mabeshi, Japan, Email: zhong@maebashi-it.ac.jp
INTERNATIONAL AGENCY FOR RESEARCH ON CANCER		Paul Brennan, INTERNATIONAL AGENCY FOR RESEARCH ON CANCER, Lyon, France, Email: brennan@iarc.fr
INFORMATION RETRIEVAL FACILITY		John Tait, INFORMATION RETRIEVAL FACILITY Vienna, Austria Email : john.tait@ir-facility.org

TABLE OF CONTENTS

List of Figures

List of Tables

List of Acronyms

Acronym	**Description**
API	Application Programming Interface
GUI	Graphical User Interface
EU	European Union
FP7	Framework Programme 7
IDE	Integrated Development Environment
IRI	Internationalized Resource Identifier
JavaGAT	Java Grid Application Toolkit
JEE	Java Enterprise Edition
JVM	Java Virtual Machine
LarKC	Large Knowledge Collider
LarKC RTE	LarKC Run-Time Environment
OWL	Ontology Web Language
RDF	Resource Description Framework
URI	Unique Resource Identifier
WP	Work package

1. Introduction

The aim of the EU FP 7 Large-Scale Integrating Project LarKC is to develop the Large Knowledge Collider (LarKC, for short, pronounced "lark"), a platform for massive distributed incomplete reasoning that will remove the scalability barriers of currently existing reasoning systems for the Semantic Web.

LarKC aims to support a highly innovative reasoning approach, which combines interdisciplinary problem solving techniques (inductive, deductive, incomplete reasoning, etc.) with methods from diverse fields (information retrieval, machine learning, cognitive and social psychology, etc.).

The LarKC platform architecture allows an effective combination of techniques coming from different disciplines by following a service-oriented computing approach. Following such approach, a complex problem may be split in simpler pieces (the LarKC plug-ins) which can be composed in an execution 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 scaling well beyond what is currently possible. The pieces of data access and reasoning functionality contained in the plug-ins, having self-contained functionality, may be reused for multiple applications.

1.1. *Objectives*

The main objective of this document is twofold:

- Serve as guide for LarKC users who want to use the LarKC Platform together with existing plug-ins and applications' workflows
- Serve as 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 constitutes a complete user manual of the LarKC Platform, which may be used as a "getting started guide" by all kind of potential LarKC early adopters and also as a manual for more experienced LarKC users.

1.2. *Document structure*

The document is structured as follows:

- Chapter 2 gives a brief overview of the LarKC Platform architecture and how the software is mapped to that architecture.
- Chapter 3 is the part corresponding to the end user´s guide. It includes detailed guidelines for the deployment of the Platform and how to run concrete examples.
- Chapter 4 constitutes the developer´s guide. It includes instructions on how to get the Platform code, set up the development environment for different kinds of tools (Ant, Eclipse) and develop plug-ins and workflows to be executed within the LarKC Platform.
- Chapter 5 gives some references to the user support tools provided and maintained by LarKC, such as mailing lists, trouble ticketing systems and fora.
- Chapter 6 gives information about the licensing scheme under which the platform code is delivered.
- Chapter 7 gives some hints on future developments which are planned within the framework of the LarKC Platform in a near to medium term.

1.3. *Who should read this document*

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

- **Plug-in developers**: they design and implement single plug-ins and deploy them in the Platform for future use. They should focus on chapter 4 and 5. Chapter 2 is recommendable, in order to get an idea of the different Platform components and the overall architecture.
- **Workflow designers**: they select existing plug-ins and combine them in the appropriate way to solve a certain task, either by implementing a scripted Decider or an intelligent one. As the

previous category, they should focus on chapter 4 and 5. Chapter 2 is recommendable, 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 configuration to execute SPARQL queries[1]. This includes both interaction with existing application interfaces and application designers who implement applications that use the services provided by the Platform. The most interesting part for them is chapter 3. Chapter 2 is only recommendable in case they are interested to know how the LarKC Platform is designed.

Chapter 6 is of interest for all kind of users, since may affect the purposes for which the Platform is used and the license under which new developed components should be released. Chapter 7 is of interest for all kind of users which would like to follow the LarKC project progress in a short-medium term.

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/.

2. LarKC Architecture Design

This chapter aims to give a brief overview of the LarKC Platform Architecture and how the software is structured according to it. Note that this chapter is not intended to be an exhaustive and detailed description of the LarKC Platform and its components. Explicit information and detailed descriptions of particular components as well as their main interactions are provided in deliverable D5.3.2 [2].

Figure 1 shows an overview of the LarKC Architecture, including the relevant Platform components and other external components, which interact directly with them. Three main domains (where the various components can be grouped) can be distinguished:
- The user domain, which includes the application(s) interface(s) and the Semantic Web components, implemented and provided in the form of plug-ins;
- The platform domain, which bridges the gap between the applications' users, semantic services and diverse hardware infrastructures, and contains features which constitute the actual core of the LarKC Platform; and
- The infrastructure domain, representing various types of computing and storage resources, which may be considered for the deployment and execution of the LarKC components (including components from both User and Platform domains).

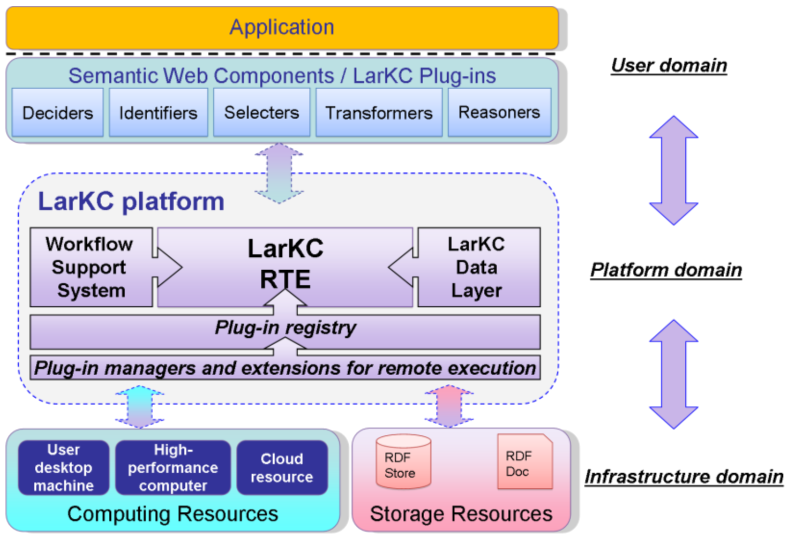


Figure 1: LarKC Architecture Overview

1 http://www.w3.org/TR/rdf-sparql-query/

In the following, a brief description is given of the LarKC Platform components and their location in the source code repository[2]:

- LarKC Run-Time Environment (RTE): based on the OpenCyc [2] engine. Its main goal is to provide a Semantic Web endpoint for remote processing of SPARQL queries (accessible through HTTP). It is the main entry point to the LarKC Platform.
  - The main code implementing the LarKC RTE can be found under: ..\platform\src\eu\larkc\core\Larkc.java
- Plug-in Registry: it allows registration of plug-ins, including their semantic descriptions, and the subsequent retrieval for their use within an execution workflow.
  - The main code implementing the Plug-in Registry can be found under: ..\platform\src\eu\larkc\core\metadata\PluginRegistry.java
- Plug-in Management System: composed by the plug-in managers and their corresponding extensions for remote execution. The plug-in managers enable the integration of plug-ins within a workflow and the management of their execution.
  - The code implementing the Plug-in Management System can be found under: ..\platform\src\eu\larkc\core\pluginManager
- Workflow Support System: it is a collection of Platform tools which supports the workflow execution and the communication between plug-ins (by means of queuing mechanisms).
  - The main code implementing the Workflow Support System can be found under: ..\platform\src\eu\larkc\core\pluginManager\local\queue
- Data Layer: it supports the plug-ins and applications with respect to storage, retrieval, and lightweight inference on top of large volumes of RDF data. It relies on OWLIM [4]. 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 is under: ..\platform\src\larkc\core\data

3. End User's guide

This chapter provides users with the instructions on how to perform the Platform installation as well as some guidelines for running the existing examples provided with the release redistribution.

For the end user's convenience, LarKC is distributed in a single package that contains the Platform and a selection of the plug-ins of different types. However, users can easily extend the plug-in collection by downloading new plug-ins from the plug-in repository (please refer to chapter 4).

3.1. Prerequisites and dependencies

The following software is required to be installed on your machine for working with LarKC:

- Java JRE[3] (tested with version 1.6)

- Apache Ant[4] tool for starting the graphical workflow demonstrator

3.2. Getting the redistributable package

The redistributable package can be downloaded from our collaborative development environment, LarKC@SourceForge[5]. It is available at https://sourceforge.net/projects/larkc/files/.

The current release package is a zip archive containing:

- Source Code

[2] Relative paths are given. Both the latest release and the development trunk can be found at: https://sourceforge.net/projects/larkc/

[3] http://www.java.com/

[4] http://ant.apache.org/

[5] http://larkc.sourceforge.net

- Platform (/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).
- Exemplary collection of the LarKC plug-Ins (/plugins): self contained, loosely coupled and functionally interoperable modules grouped in the following categories (according to the functionality, please refer to chapter 4 for more details), including:
  - DECIDER (/plugins/decider):
    - ScriptedSimpleAnyTimeDecider (/plugins/decider/ScriptedAnytimeDecider): a simple example that executes a workflow iteratively over a set of the exemplary plug-ins, returning intermediate results at the end of each iteration (before returning the final result).
    - Template Decider (/plugins/decider/template_decider): a simple template for creating new Decider plug-ins
  - TRANSFORMER (/plugins/transform):
    - SPARQLToTriplePatternQueryTransformer (/plugins/transform/SPARQLToTriplePatternQueryTransformer): returns a set of triple patterns that are contained in the given input SPARQL query.
    - Template Transformer (/plugins/decider/template_transform): a simple template for creating a new Transform plug-ins.
  - IDENTIFIER (/plugins/identify):
    - SindiceTriplePatternIdentifier (/plugins/identify/SindiceTriplePatternIdentifier): produces a set of RDF Graphs for the given Triple Pattern set.
    - Template Identifier (/plugins/identify/template_identify): a simple template for creating a new Identifier plug-ins.
  - SELECTER (/plugins/select):
    - GrowingDataSetSelecter (/plugins/select/GrowingDataSet): provides a full selection over the given SetOfStatements.
    - Tempate Selecter (/plugins/select/template_selecter): a simple template for creating a new Selecter plug-ins.
  - REASONER (/plugins/reason):
    - SparqlQueryEvaluationReasoner (/plugins/reason/SparqlQueryEvaluationReasoner): wraps the Jena reasoner and enables the execution of SPARQL Select, Construct, Describe and Ask requests to be executed against the given query.
    - Template Reasoner (/plugins/reason/template_reasoner): a simple template for creating a new Reasoner plug-ins.
- Demonstration GUI for composing a simple workflow (/demos/simple_pipeline).
- A front-end graphical interface for submitting queries to the Platform's SPARQL end-point (development_kit/SPARQLClient).

- User and developer manual in pdf (/doc).
- Binaries: The binary release contains the pre-built component from the source code release.

Each of the release components includes:

- an Ant build script for the installation
- an Eclipse project (provided with the source code redistribution only) for the code development

Moreover, the root directory contains a "master" Ant build script, which compiles and deploys the Platform from "/platform" and all the plug-ins from "/plugins" directories (additional tools are not built by the master).

3.3. Deployment of LarKC

The code included in the redistributable package contains the pre-compiled binaries of the source code and ready to use after extraction from the package's archive.

3.3.1. Starting the Platform

For running the Platform Run-Time Environment, you must execute the "run-larkc" script (.bat for Windows compatible OS and .sh for Linux like OS). The environment automatically loads the plug-ins located in the "plugins" folder (files with the ".larkc" extension, each corresponding to a plug-in included in the current redistribution).

Important: Note that the default DECIDER is the last one loaded by the Platform. If you want to change the default DECIDER, you must specify it explicitly in the "conf/plugin.ini" file. Refer to chapter 4 for more details about editing the "conf/plugin.ini" file. The Platform does not support multiple deciders so far. If we have a number of them, the last loaded one is taken.

An alternative to editing the "conf/plugins.ini" file for choosing the desired DECIDER is simply to remove all the other deciders (notably, the corresponding ".larkc" files) from the "plugins" folder.

3.3.2. Submitting a query

For submitting a query to the LarKC Platform's SPARQL end-point, a SPARQL client is provided with the current distribution, which can serve as a user GUI. To start the GUI, the user needs simply to execute "startClient" script (.bat for Windows compatible OS and .sh for Linux like OS), which can be found at ..\development_kit\SPARQLClient. The GUI provides the controls for entering the query, submitting it as well as posting the results.

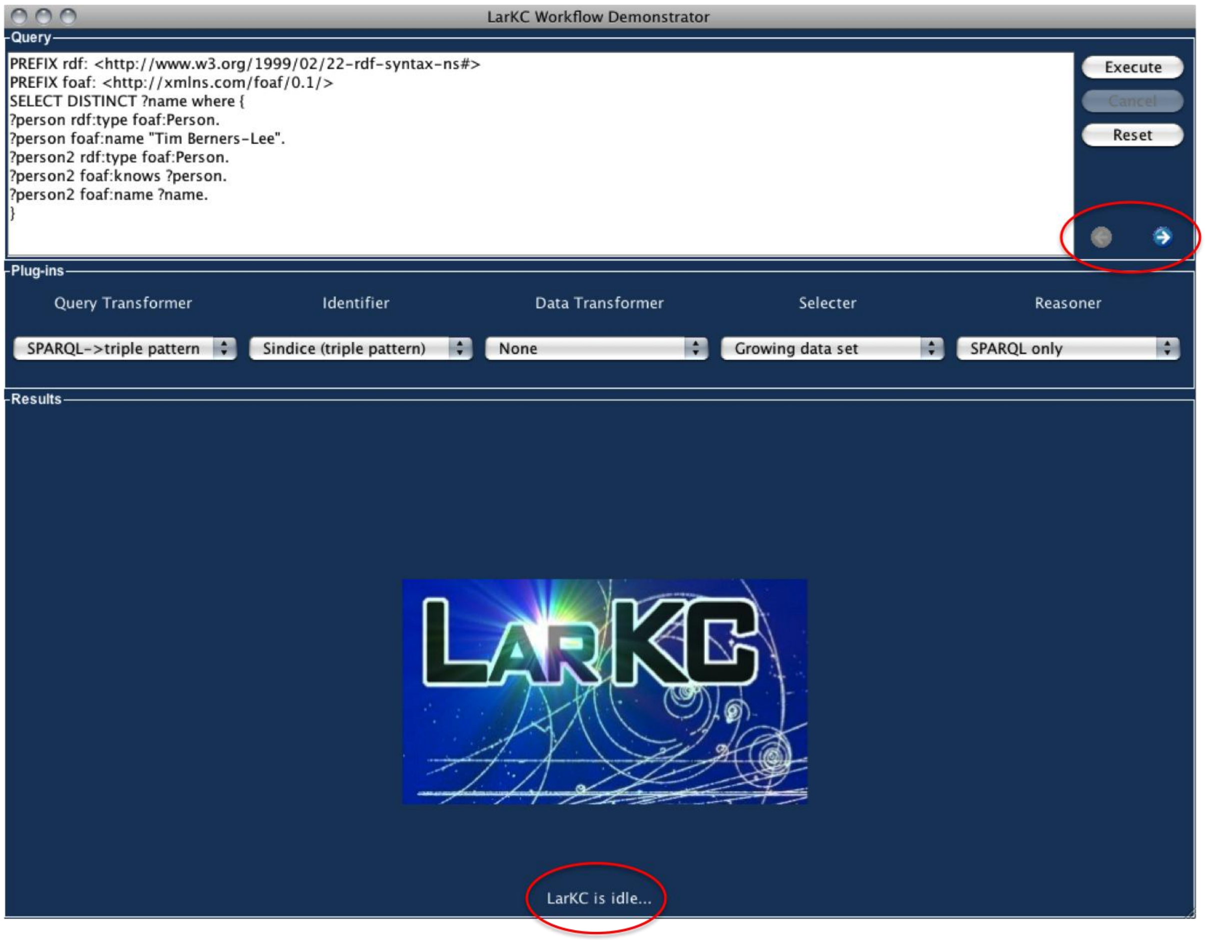
However, the user can use any other end-point for submitting the queries.

3.4. Running examples

In addition to the core Platform and plug-in binaries, the redistributable package also contains a graphical user interface, the so-called "LarKC Workflow Demonstrator" to test simple workflows and execute some predefined queries. Nevertheless, the client could also be used to try out own workflows – users can combine the available plug-ins according to own preferences – and submit self-created queries. Starting the demonstrator is as simple as starting the previously mentioned query client. Users just need to execute the "startDemo" script (.bat for Windows compatible OS and .sh for Linux / Mac like OS), which can be found at ..\demos\simple_pipeline.

Having started the demonstrator, users will see the following application appearing on their desktops (see Figure 2).

The layout is principally divided into three different sections, namely query, plug-ins, and results. The topmost part – "query" – consists hereby of a big input field where users can type in their own query statements or modify one of the predefined queries. To choose between any existing queries, one should press the right arrow to display the next in the row. Reviewing the previous query can be achieved by clicking on the left arrow button, which becomes enabled as soon as the second query has been selected. All in all, four predefined queries are currently available. To start the corresponding query, one can simply click on the execute button and everything is done totally automatic and transparent. To keep users informed about the status of the LarKC Platform, the actual behaviour is always displayed at the bottom of the GUI. "LarKC is idle…" means that queries can be submitted and "LarKC is processing the query…" means that the query has been executed and is currently processed by the LarKC Platform. Nevertheless, one can also stop the execution by pressing the cancel button. Reset foremost removes the previous results listed at the bottom.



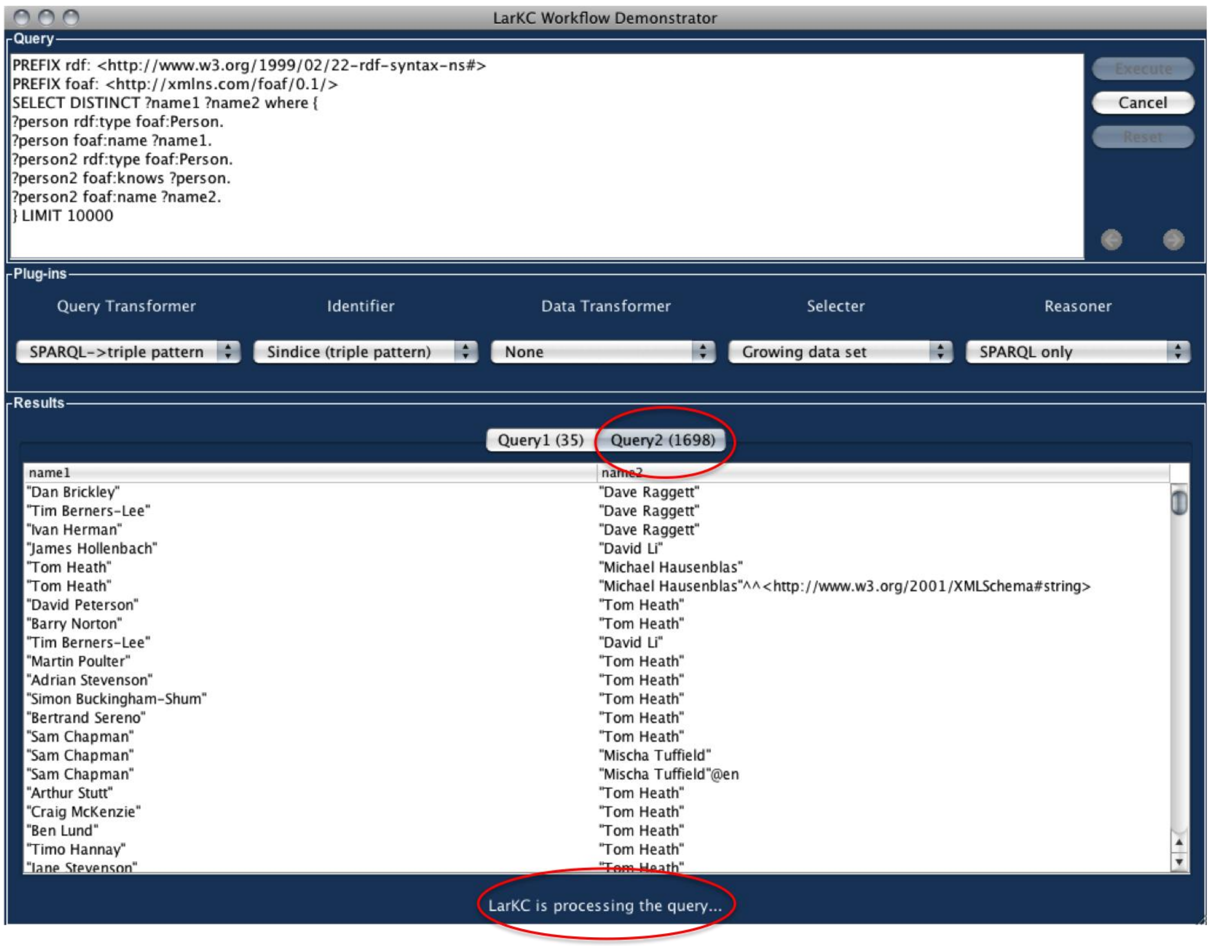
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?name where {
?person rdf:type foaf:Person.
?person foaf:name "Tim Berners-Lee".
?person2 rdf:type foaf:Person.
?person2 foaf:knows ?person.
?person2 foaf:name ?name.
}
```

Query Transformer: SPARQL->triple pattern
Identifier: Sindice (triple pattern)
Data Transformer: None
Selecter: Growing data set
Reasoner: SPARQL only

Figure 2: The startscreen of the LarKC Workflow Demonstrator GUI

As soon as first results arrive, the GUI immediately displays them in the results area. While the query is still running any new arriving results will be automatically added to the current view. This demonstrates very well the any time behaviour of the LarKC Platform – users do not have to wait for the final results but retrieve intermediate results as soon as they arrive (see Figure 3). This functionality is very important since some of the queries can run for a very long time. The number shown in the name of the tab refers to the total number of results retrieved by the currently processed query so far. It is also immediately updated as soon as new results have been found and added to the list of results.

In the middle area of the application, all registered plug-ins are listed according to their capabilities. Five different forms of plug-ins are currently supported: query transformation, identification of sources and data sets, data set transformation, selection of data sets, and finally, reasoning over selected data. If users choose one of the predefined queries, the corresponding workflow (fixed sequence of plug-ins) is automatically created and displayed. If users are willing to create their own workflows, they should carefully select relevant plug-ins and combine them in the correct order. A description of the available plug-ins will soon be available at http://www.larkc.eu/plug-in-marketplace/.



```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?name1 ?name2 where {
?person rdf:type foaf:Person.
?person foaf:name ?name1.
?person2 rdf:type foaf:Person.
?person2 foaf:knows ?person.
?person2 foaf:name ?name2.
} LIMIT 10000
```

Query Transformer	Identifier	Data Transformer	Selecter	Reasoner
SPARQL->triple pattern	Sindice (triple pattern)	None	Growing data set	SPARQL only

name1	name2
"Dan Brickley"	"Dave Raggett"
"Tim Berners-Lee"	"Dave Raggett"
"Ivan Herman"	"Dave Raggett"
"James Hollenbach"	"David Li"
"Tom Heath"	"Michael Hausenblas"
"Tom Heath"	"Michael Hausenblas"^^<http://www.w3.org/2001/XMLSchema#string>
"David Peterson"	"Tom Heath"
"Barry Norton"	"Tom Heath"
"Tim Berners-Lee"	"David Li"
"Martin Poulter"	"Tom Heath"
"Adrian Stevenson"	"Tom Heath"
"Simon Buckingham-Shum"	"Tom Heath"
"Bertrand Sereno"	"Tom Heath"
"Sam Chapman"	"Tom Heath"
"Sam Chapman"	"Mischa Tuffield"
"Sam Chapman"	"Mischa Tuffield"@en
"Arthur Stutt"	"Tom Heath"
"Craig McKenzie"	"Tom Heath"
"Ben Lund"	"Tom Heath"
"Timo Hannay"	"Tom Heath"
"Jane Stevenson"	"Tom Heath"

LarKC is processing the query...

Figure 3: The GUI during an execution of a query

4. Developer’s guide

This chapter is intended for developers who want to make use of the LarKC Platform, both for plug-in development and for workflow design. It includes instructions on getting the Platform code, on setting up the development environment for different kinds of tools (Ant, Eclipse) and on developing plug-ins and workflows to be executed within the LarKC Platform.

4.1. Installation Prerequisites

The following software is required to be installed on your machine in order to start working with LarKC:

- Java JRE (tested with version 1.6)
- One of the Java Development Environments. Due to the fact that the LarKC Platform is written in Java, any of the standard Java IDEs can be used to develop and contribute:
  - Eclipse IDE (inside the consortium, Eclipse is the most widely used IDE for LarKC)
  - Netbeans IDE
- Java development kit[6] (tested with version 1.6)
- One of the SVN clients for getting the code from the LarKC source code repository (if necessary):

[6] http://www.java.com/en/download/index.jsp

  - TortoiseSVN[7] (Windows)
  - Command line SVN client [8] (Linux)
  - Apache Ant[9]
  - Built-in subversion client of Netbeans[10]
- Tool for the deployment one of the Java IDEs (like Eclipse[11], Netbeans[12] etc.), for the development

4.2. Getting the Source Code

The source code of LarKC is available for downloading at our collaborative development environment, LarKC@SourceForge[13]. Through LarKC@SourceForge, the LarKC users may:

- either download the latest release (via web browser, at http://larkc.sourceforge.net) as a package containing the source code, available at: https://sourceforge.net/projects/larkc/files/
- or check the code out from the subversion code repository (requires special clients listed in Section 4.1). The main development version (trunk) of the Platform and plug-ins can be checked-out from https://larkc.svn.sourceforge.net/svnroot/larkc/trunk/platform and https://larkc.svn.sourceforge.net/svnroot/larkc/trunk/plugins.

For the latter, please refer to Section 4.2.1.

4.2.1. Getting the Source Code from the Subversion Repository

The latest source code for the Platform and plug-ins can be checked-out from the LarKC's Subversion repository using any SVN client.

Using command line SVN client (mainly for Linux users)

The following will download the Platform and all the example plug-ins developed by the LarKC consortium into two separate directories. It is also possible to check-out Platform and plug-ins separately and also to check-out each plug-in separately from the plug-in directory tree.

```
me@myComp:~/$ svn checkout --username yourname
https://larkc.svn.sourceforge.net/svnroot/larkc/trunk/ LarKC
```

Using TortoiseSVN (for Windows users)

To check-out the code using the TortoiseSVN tool, create the directory where you want to have the Platform and the plug-ins, for example ./LarKC, and then in windows explorer right click and pick SVN Checkout option (see Figure 4).

[7] http://tortoisesvn.tigris.org/
[8] http://subversion.tigris.org/
[9] http://ant.apache.org/
[10] http://www.netbeans.org/downloads/
[11] http://www.eclipse.org/
[12] http://www.netbeans.org/downloads/
[13] http://larkc.sourceforge.net

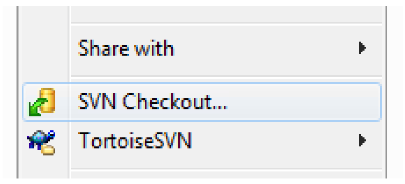
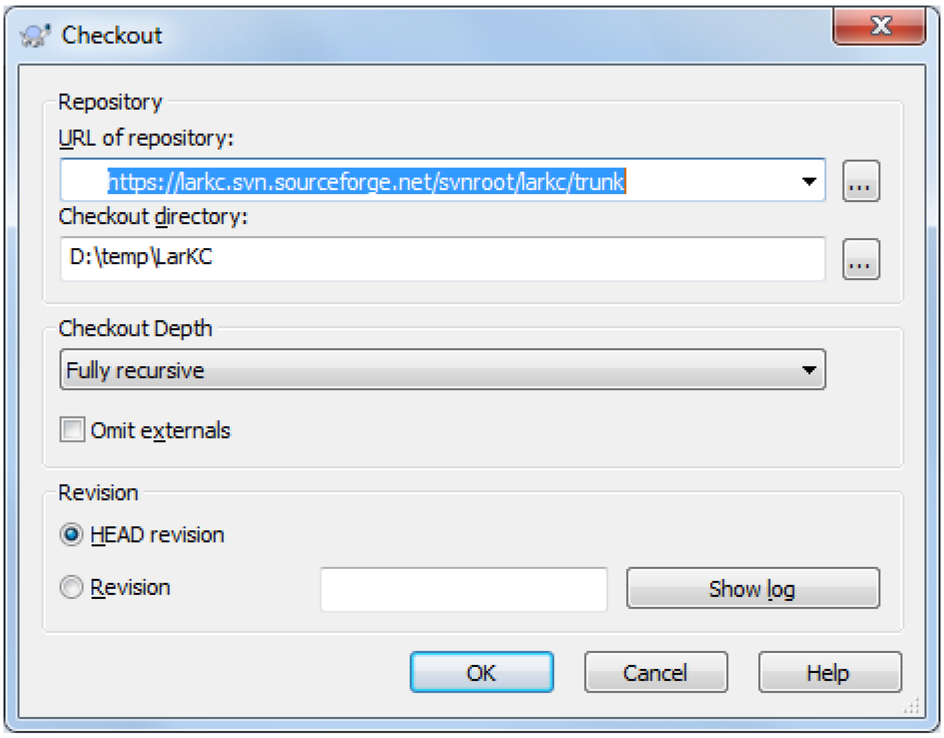


Figure 4: Check-out using TortoiseSVN

In the Checkout window, introduce the URL of the LarKC repository and the local directory[14] where you want to download the code (see Figure 5).


Checkout

Repository

URL of repository: https://larkc.svn.sourceforge.net/svnroot/larkc/trunk

Checkout directory: D:\temp\LarKC

Checkout Depth: Fully recursive

☐ Omit externals

Revision

● HEAD revision

○ Revision

Figure 5: Check-out window from TortoiseSVN

4.2.2. Source Code Structure

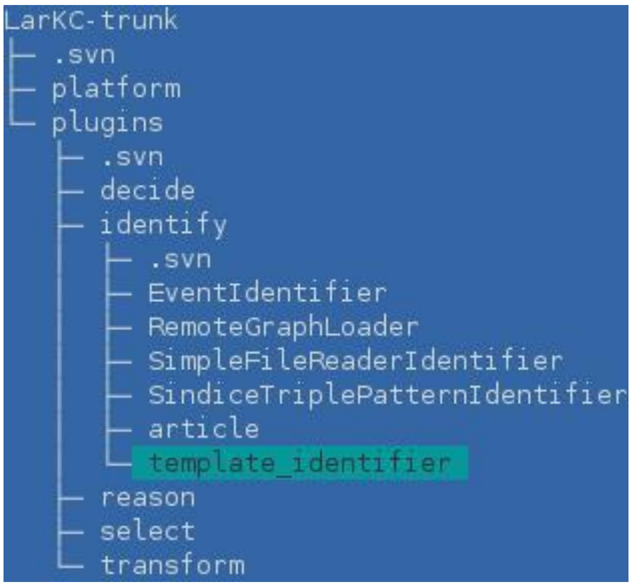
The Source Code of LarKC consists of two essential parts: the Platform and the plug-in collection.

The Platform enables the plug-in registration and execution, construction of a workflow and running experiments.

The plug-ins are grouped in five main categories: Decider, Identifier, Reasoner, Selecter and Transformer (Figure 6).

The plug-in collection contains template for each plug-in type ("template_type"), which might be beneficial for constructing new plug-ins of the corresponding type.

[14] Some users have reported difficulties in checking out LarKC into deeply nested directories on Windows. If you get an error message during checkout, you might try using a high level directory, such as the one in the example

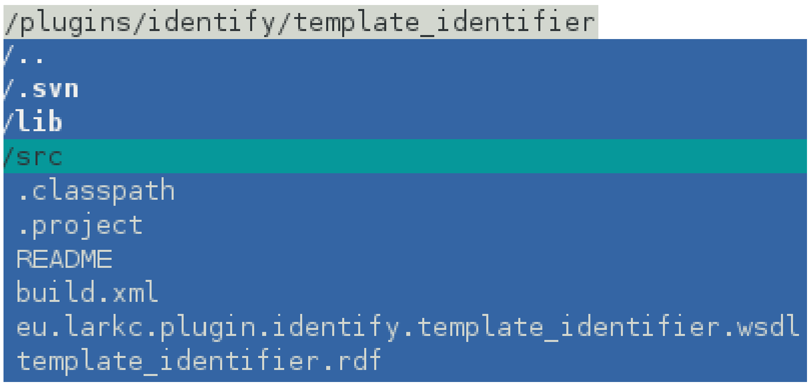


```
LarKC-trunk
├─ .svn
├─ platform
└─ plugins
   ├─ .svn
   ├─ decide
   ├─ identify
   │  ├─ .svn
   │  ├─ EventIdentifier
   │  ├─ RemoteGraphLoader
   │  ├─ SimpleFileReaderIdentifier
   │  ├─ SindiceTriplePatternIdentifier
   │  ├─ article
   │  └─ template_identifier
   ├─ reason
   ├─ select
   └─ transform
```

Figure 6: Source code structure

In the current design, the Platform (or a single plug-in) are separate software projects which are compiled separately. The only prerequisite for compiling plug-ins is the availability of the compiled Platform code.

The content of a typical LarKC plug-in folder (in this case identify/template_identifier) is as shown in Figure 7.



```
/plugins/identify/template_identifier
/..
/.svn
/lib
/src
 .classpath
 .project
 README
 build.xml
 eu.larkc.plugin.identify.template_identifier.wsdl
 template_identifier.rdf
```

Figure 7: Plug-in content

The plug-in source code resides in the “src” directory, following the package notation adopted by Java. The libraries used by the plug-in are stored in the “lib/” directory (each plug-in has its own “lib/” directory in order to avoid possible conflicts with the libraries used by other plug-ins and the Platform). Plug-in annotation is stored in the corresponding “.rdf” and “.wsdl” files following the specifications[15] [16].
Creation of the new plug-ins is facilitated by means of the special templates included in addition to the released plug-ins (see section 4.5 for more details on creation of the new plug-ins).

4.3. Deployment and development using the Ant Tool

The downloaded components are equipped with the Ant build scripts facilitating the building process of both the Platform and the plug-ins. The master Ant build script, resides in the main directory. It initiates the building process firstly for the Platform, and afterwards for each of the plug-ins. This is done running the “ant” command in the current directory from the OS command line (please make sure that the Ant tool is installed and works properly).

[15] http://www.w3.org/standards/techs/rdf
[16] http://www.w3.org/TR/wsdl20/

```
me@myComp:~/larkc$ ant
Buildfile: build.xml

…

BUILD SUCCESSFUL

Total time: 23 seconds
```

Tipp: the “ant” command can be also run separately for the Platform, plug-in collection or even each of the plug-ins (simply switching to the location of the subdirectories which are needed to be built).

```
me@myComp:~/larkc/platform$ cd ../plugins/

me@myComp:~/larkc/plugins$ ant

...

BUILD SUCCESSFUL
```

The built output is placed in the “platform/dist” directory:

```
me@myComp:~/larkc/$ cd platform/dist

me@myComp:~/larkc/platform$ ls -la

bin/

conf/

ext/

plugins/

run-larkc.bat

run-larkc.sh
```

The outcome of the building process for the plug-ins is a set of the “.larkc” files/containers produced in the “platform/dist/plugins” directory. Every “.larkc” container includes the binaries as well as libraries of the corresponding plug-in. The Platform handles with the “.larkc” files during the startup.

The following table summarizes the Ant build script parameters implemented for the LarKC source code.

Table 1: Available Ant options

Parameter	Applied for:			
	Platform and plug-ins	Platform	Plug-ins	Concrete plug-in
path	*pwd*	*pwd/platform*	*pwd/plugins*	*pwd/plugins/[plugin]*
dist [default]	performs actions (a) and (b)	a) builds the Platform, stores the binaries at	performs action (b) for all the plug-ins	b) builds the plug-in, creates the plugin.larkc container, puts the

		pwd/dist/bin, prepares startup and other service files		binaries and libraries to the container, stores the container at pwd/platform/dist/plugins
clean		a) cleans up the pwd/dist directory		b) deletes the plugin.larkc container from pwd/platform/dist/plugins
doc		a) generates JavaDoc for the Platform		b) generates JavaDoc for the plug-in
run	starts up the Platform RTE		- not available -	

4.4. Deployment and development using Eclipse IDE

First step is to check out the code, as described in section 4.2.

After you have the code on your disk, open Eclipse, and when it asks for the workspace directory, please make sure that the directory where you checked out the Platform and plug-ins is not in the direct subtree of the workspace directory.

After it starts in the File menu, pick the Import sub menu (*File>Import*). In the import window, pick the *"Existing Projects into Workspace"* option (see Figure 8):

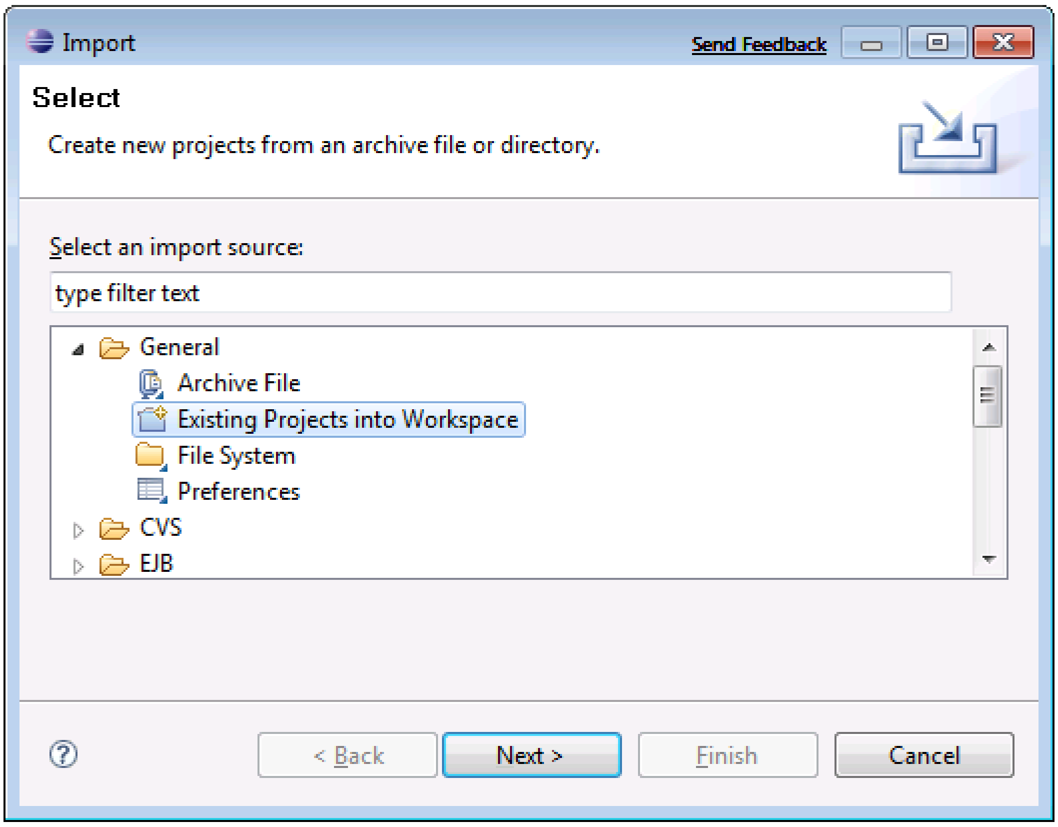
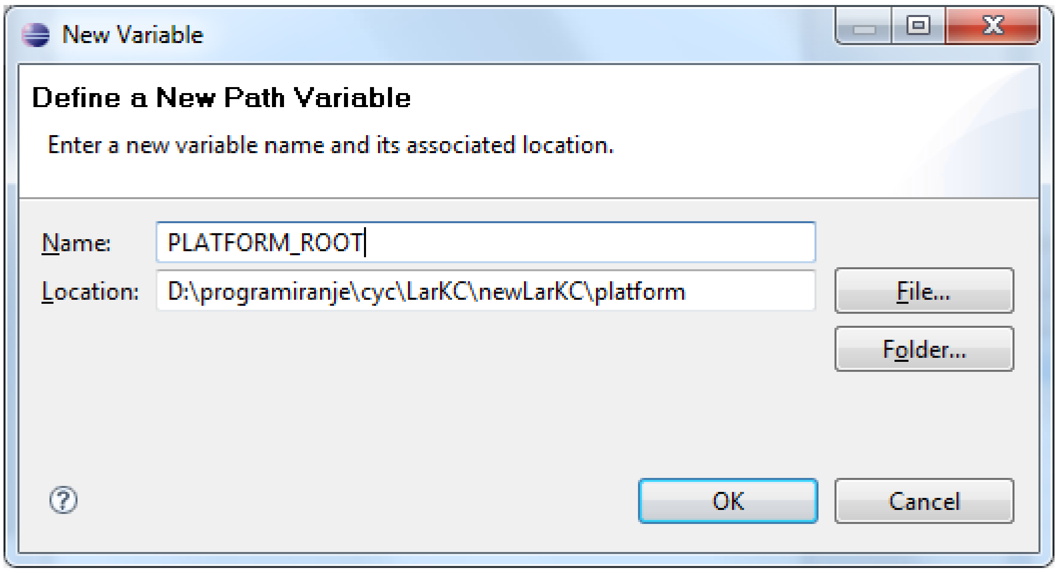

Import
Select
Create new projects from an archive file or directory.
Select an import source:
type filter text

Figure 8: Importing project to Eclipse

In the next window, click on the first *Browse* button, and pick the directory where you have checked out the Platform and plug-ins. After this, just click finish and the Platform and plug-in projects will be imported.

Due to the fact that the Platform needs to have plug-ins registered, all the plug-in projects built the output directory linked to the PLATFORM_ROOT\plugins\PluginName directory. This makes easier to develop and test plug-ins together with the Platform. To set up the PLATFORM_ROOT, go to menu Window->Preferences and, in the side menu, go to General->Workspace->Linked Resources and click New and then Folder (see Figure 9).


New Variable

Define a New Path Variable

Enter a new variable name and its associated location.

Name: PLATFORM_ROOT

Location: D:\programiranje\cyc\LarKC\newLarKC\platform

Figure 9: Add PLATFORM_ROOT variable

In order to run/debug the Platform inside the Eclipse, you have to set up the Eclipse's *run configuration*. For that, click on the arrow next to the green "play" button and then the *Run Configurations...* option (Figure 10):

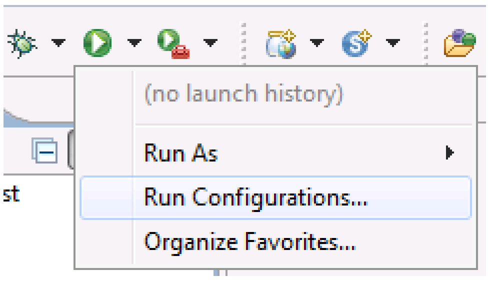
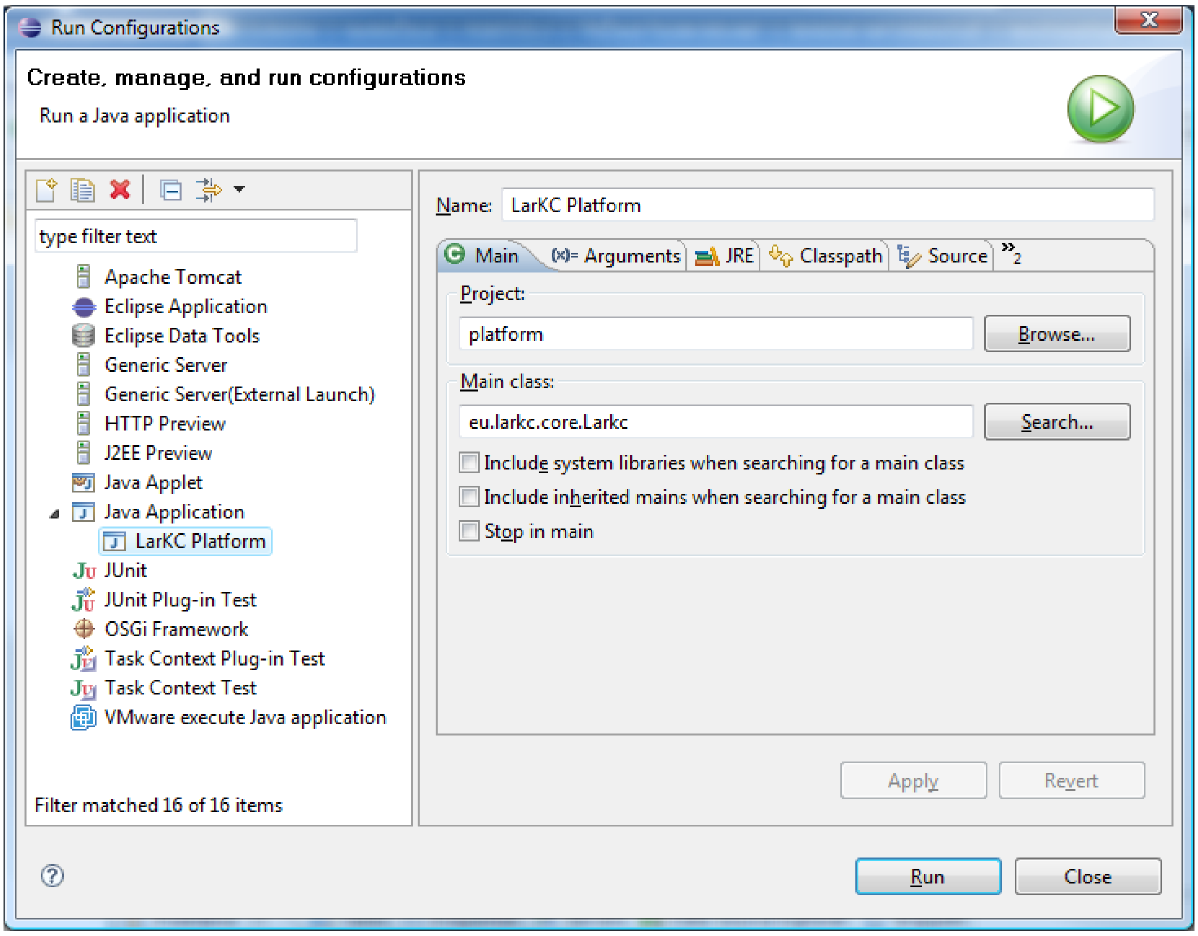


Figure 10: Run Configurations option

On the *Run Configurations* window which opens, create new *Java Application* (left side). Then, under name, enter the configuration name you want to have, for example *platform*. Under the Project, pick the project into which you imported the Platform, and under Main class, write *eu.larkc.core.Larkc* (see Figure 11):


Run Configurations

Create, manage, and run configurations

Run a Java application

type filter text

- Apache Tomcat
- Eclipse Application
- Eclipse Data Tools
- Generic Server
- Generic Server(External Launch)
- HTTP Preview
- J2EE Preview
- Java Applet
- Java Application
  - LarKC Platform
- JUnit
- JUnit Plug-in Test
- OSGi Framework
- Task Context Plug-in Test
- Task Context Test
- VMware execute Java application

Filter matched 16 of 16 items

Name: LarKC Platform

Main | Arguments | JRE | Classpath | Source | »2

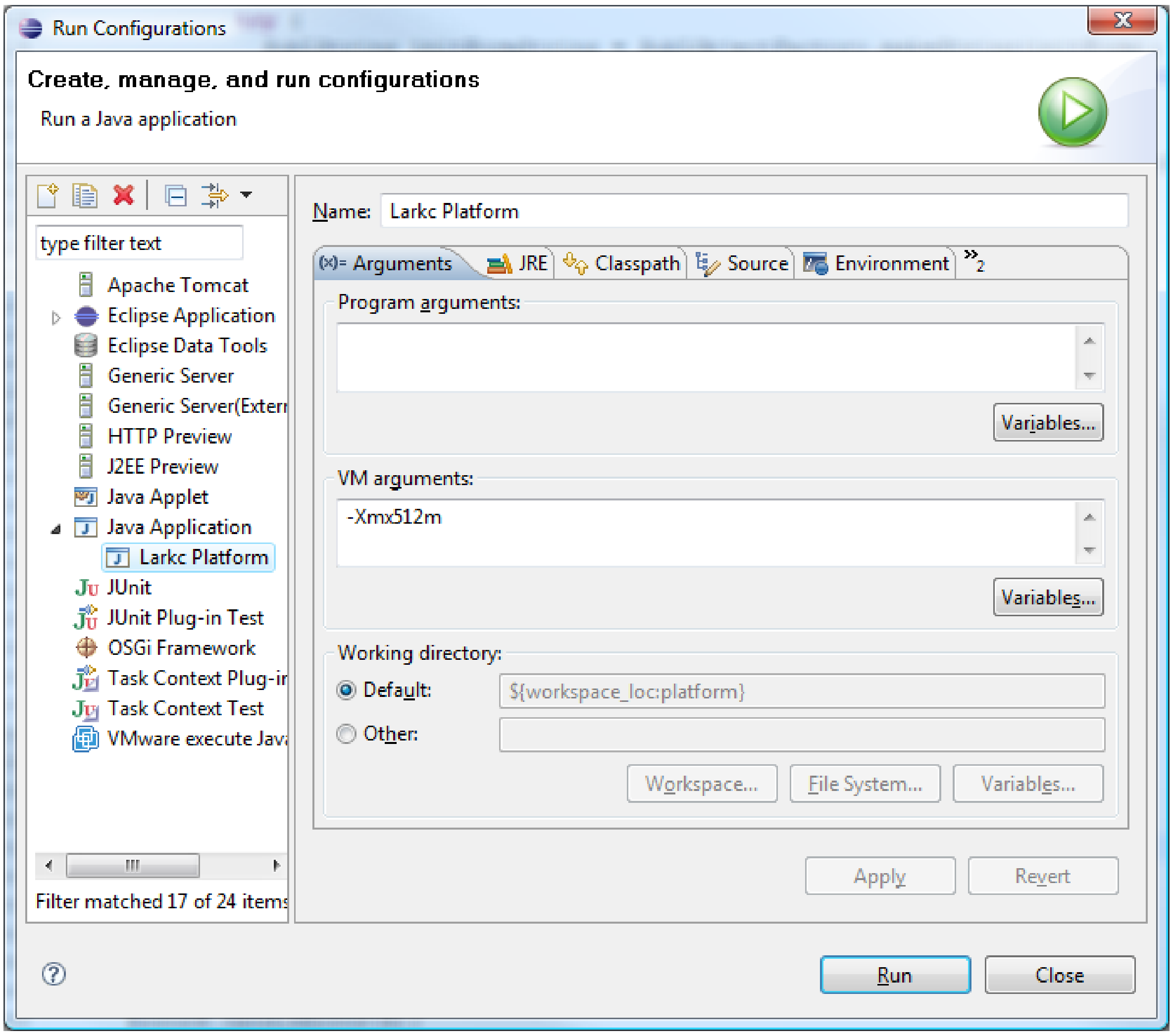
Project: platform

Main class: eu.larkc.core.Larkc

- [ ] Include system libraries when searching for a main class
- [ ] Include inherited mains when searching for a main class
- [ ] Stop in main

Figure 11: Run Configutations window

If you use 32bit Java, click on the tab "*Arguments*" and under *"VM arguments"* add "-Xmx512m" as seen on Figure 12. The *"VM arguments"* parameter is only needed if you are running a 32bit version of Java, or running out of memory because of your plug-ins:


Run Configurations

Create, manage, and run configurations
Run a Java application

type filter text
Apache Tomcat
Eclipse Application
Eclipse Data Tools
Generic Server
Generic Server(Extern
HTTP Preview
J2EE Preview
Java Applet
Java Application
Larkc Platform
JUnit
JUnit Plug-in Test
OSGi Framework
Task Context Plug-in
Task Context Test
VMware execute Java
Filter matched 17 of 24 items

Name: Larkc Platform

Arguments | JRE | Classpath | Source | Environment | »2

Program arguments:

VM arguments:
-Xmx512m

Working directory:
Default: ${workspace_loc:platform}
Other:

Figure 12: Run Configurations window command line arguments

The LarKC Platform is now ready to run (without the plug-ins).

4.4.1. Plug-ins

First of all you have to check out the code, as described in section 4.2 (...trunk/plugins).

After you have the *plugins* project and source code on your disk, please repeat the importing project step as described in 4.4 for the Platform, but this time for the plug-ins.

4.4.2. Putting Plug-ins and Platform together

In order to be able to run the plug-ins from the *plugins* project inside the *platform* project, both projects (and at least the plug-ins you would like to use) must be built successfully.

Default settings of the *plug-ins* project, after check-out, are set up to build into a /bin subdirectory. This is also the default location where the Platform will look for the plug-ins (please check the plugins.ini file).

If everything builds, and the plug-in does not need any special libraries, the Platform should be ready to run.
After the Platform server is up and running, together with a Decider and other plug-ins, it can be queried using any SPARQL client (e.g. the one provided with the Platform package, located at development_kit/SPARQLClient).

When debugging the plug-ins for the first time (i.e. set a breakpoint somewhere in your plug-in), the Eclipse may report: "Source not found". In this case, just click the "Edit Source Lookup Path", then "Add" -> "Java Project", check the *plugins* project and click the "OK" button.

4.5. How to develop a plug-in

4.5.1. Using the Plug-in Wizard

Starting Eclipse with the LarKC Plugin Wizard

To use the LarKC Plugin Wizard within Eclipse, the JAR file of the LarKC Plugin Wizard[17] has to be placed in the “plugins” directory of the Eclipse directory. The plug-in is immediately available after restarting Eclipse.

Choose the LarKC platform location

In order to create a new plug-in for LarKC with the Plugin Wizard, it is necessary to specify the location of the LarKC platform (in its binary form), where it can be found on your local disk. If Eclipse is running with the Plugin Wizard, there are several ways to do this.
The Plugin Wizard adds a new menu item to Eclipse, named LarKC (see Figure 13, point 1). The first subitem in this menu is “Select LarKC platform location”. Clicking on this menu entry will open a new window (see Figure 14) where you can see a textfield with the current path to the LarKC platform (empty if no path is defined) and a button “Browse...”. To change the location of the LarKC platform you can either type in the path directly or click the button “Browse...” and navigate in the dialog to the location of the LarKC platform on your disk. If a valid location is selected, the button “OK” will be enabled, otherwise it is disabled and in the information area you can see an error message, which tells you what went wrong.

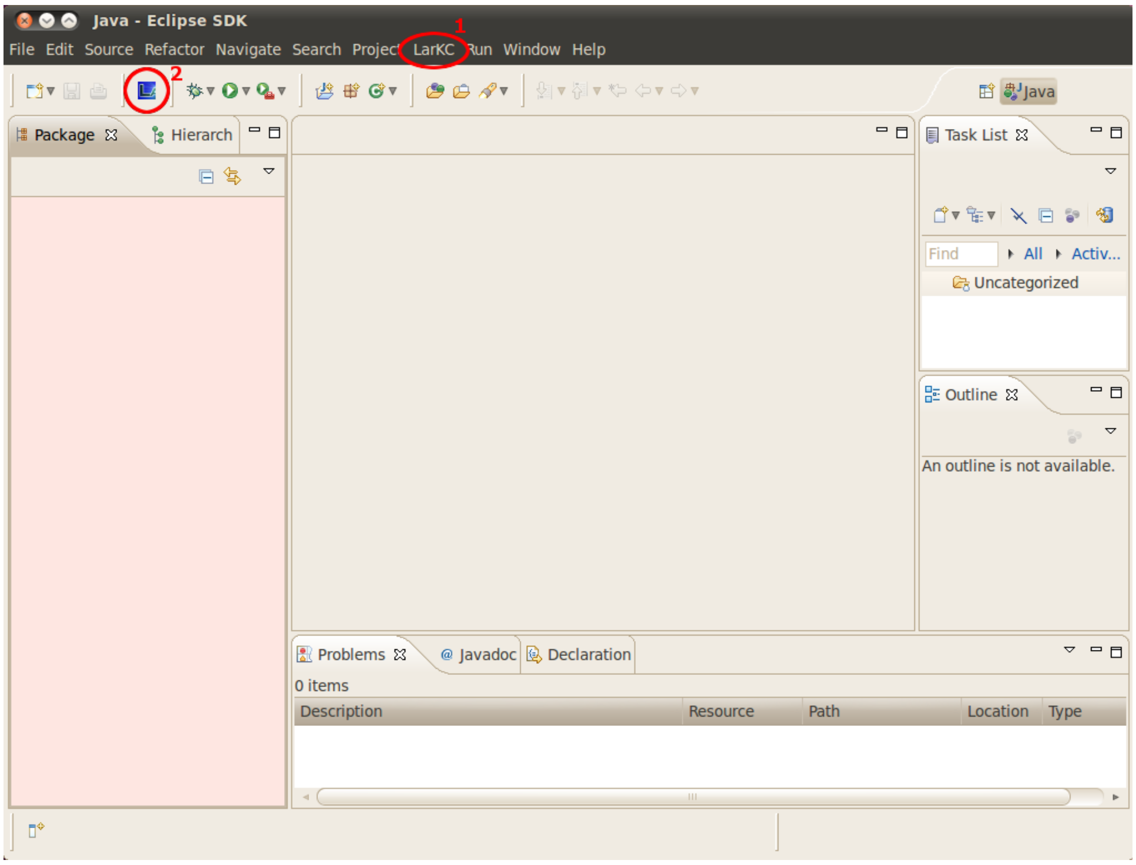
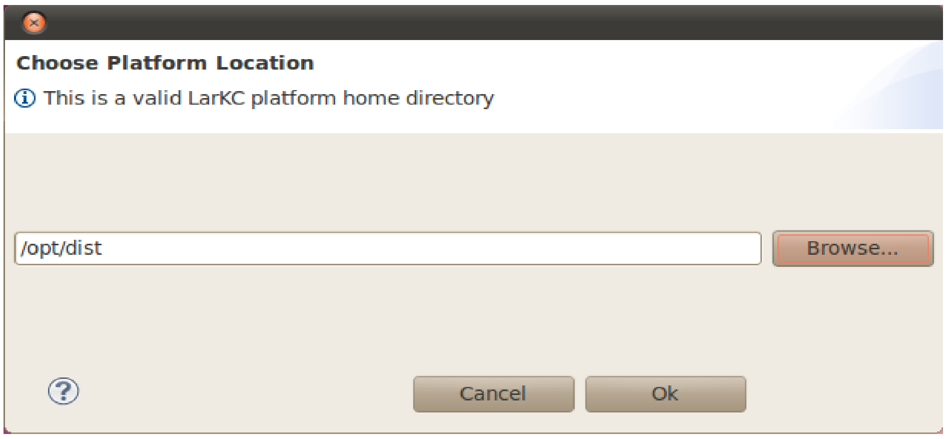


Figure 13: Additional elements on the Eclipse GUI for LarKC

[17] Listed under development_kit/LarKCWizard/dist/LarKCWizard_1.0.0.201006301329.jar

Also a new toolbar entry is added to Eclipse (see Figure 13, point 2). By clicking on this button the same dialog will open where you can choose the location of the LarKC platform.



Choose Platform Location

This is a valid LarKC platform home directory

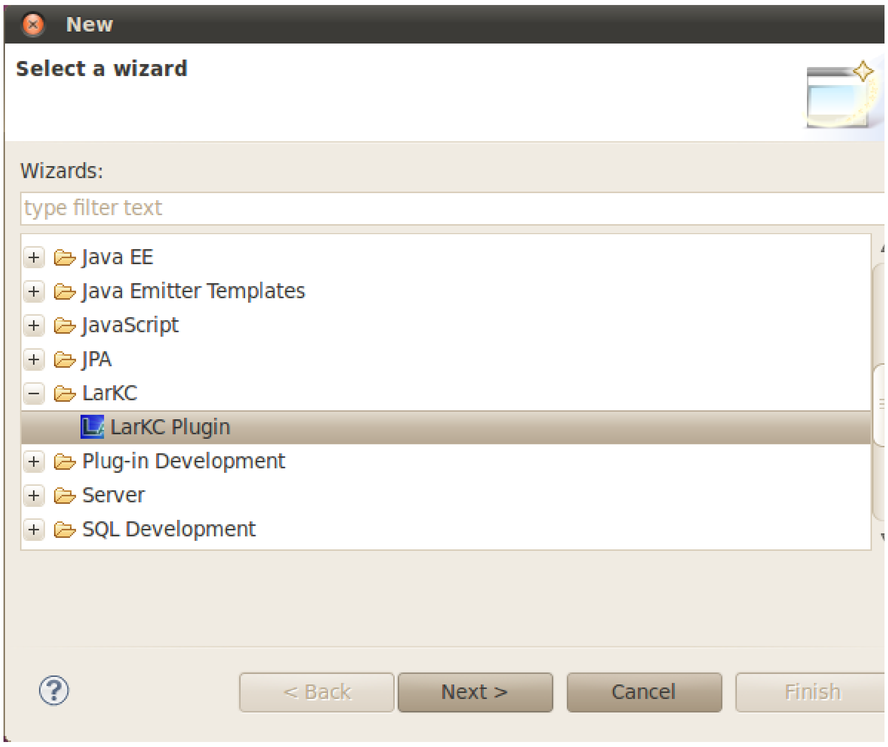
/opt/dist

Figure 14: Dialog to specify the path to the LarKC platform

As a third possibility the location of the platform can also be specified directly when creating a new LarKC plug-in project, which is explained in the next section.

Creating a new LarKC Plug-in project

To create a new LarKC plug-in project click on the menu entry "File → New → Other..." or press CTRL + N. In the dialog a directory named "LarKC" is added. There you can find the entry "LarKC.



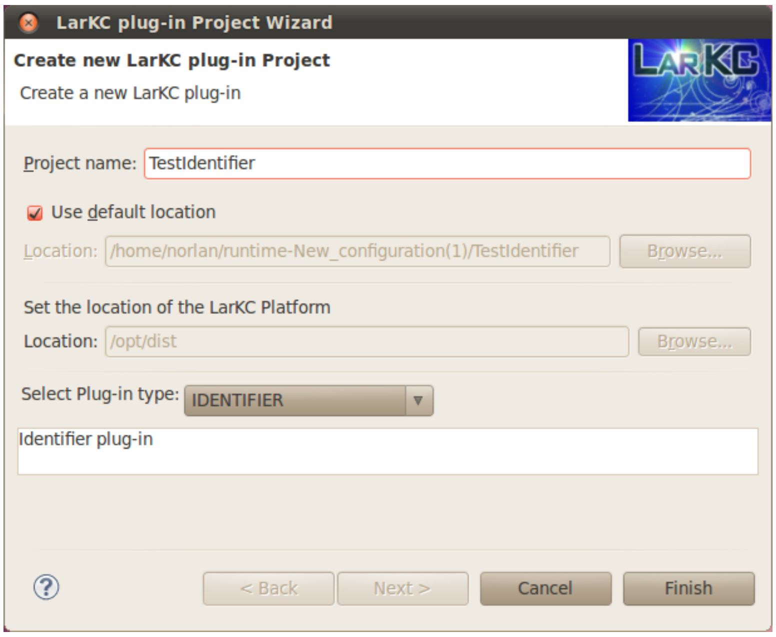
New

Select a wizard

Wizards:

Figure 15: Dialog to create a new project

By selecting this entry and clicking on the button "Next >" you come to a new dialog, where you can specify the LarKC plug-in, which you want to create (see Figure 16).


LarKC plug-in Project Wizard
Create new LarKC plug-in Project
Create a new LarKC plug-in
< Back
Next >
Cancel
Finish

Project name: TestIdentifier

Use default location [✓]

Location: /home/norlan/runtime-New_configuration(1)/TestIdentifier

Set the location of the LarKC Platform

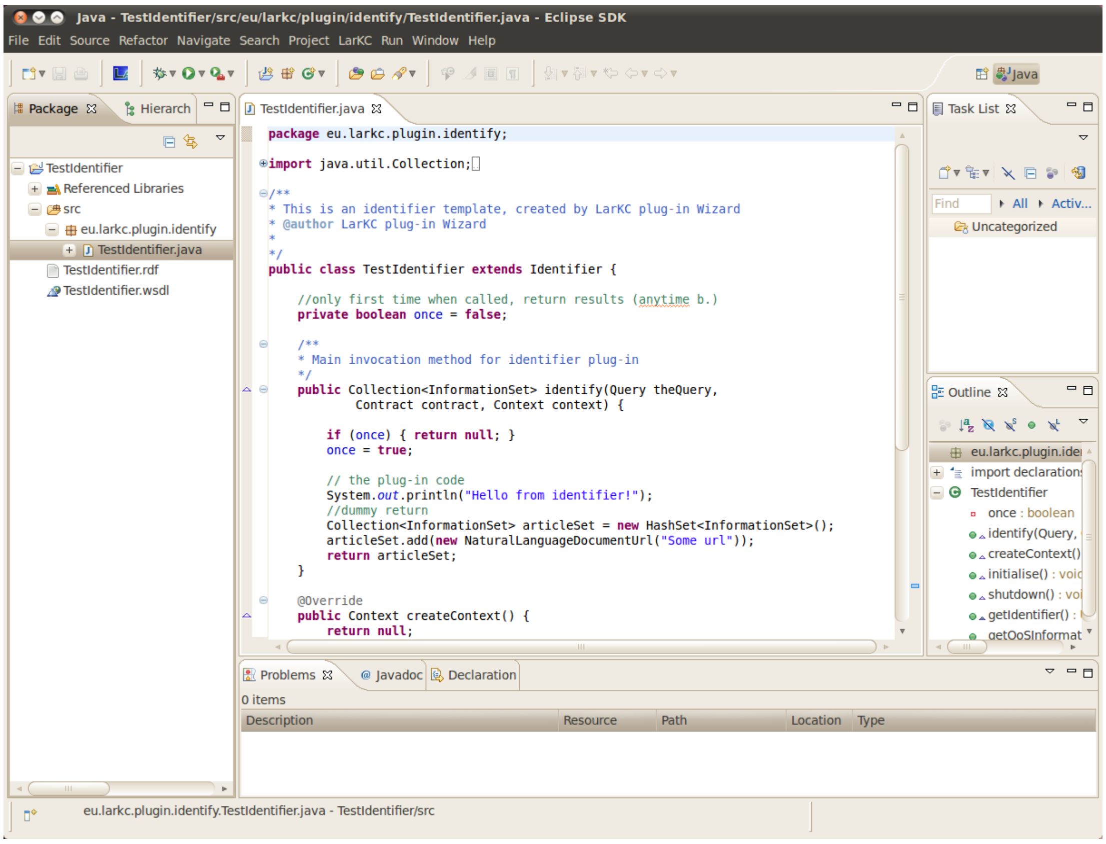
Location: /opt/dist

Select Plug-in type: IDENTIFIER

Identifier plug-in

Figure 16: LarKC plug-in Project Wizard

In the first textfield the name of the plug-in can be typed in. By selecting the checkbox “Use default location”, the project is placed in the actual workspace of Eclipse. Otherwise a different location can be chosen. In this case, the location of the LarKC platform is already defined. If no path is defined you can either type it in the textfield or browse the file system by clicking on the button “Browse...”. In the combobox below the type of plug-in can be chosen. The possibilities are IDENTIFIER, SELECTER, TRANSFORMER, INFORSET_TRANSFORMER, REASONER and DECIDER. In the lowermost textbox, a brief description of the selected plug-in type is given.



```java
package eu.larkc.plugin.identify;

import java.util.Collection;

/**
* This is an identifier template, created by LarKC plug-in Wizard
* @author LarKC plug-in Wizard
*
*/
public class TestIdentifier extends Identifier {

    //only first time when called, return results (anytime b.)
    private boolean once = false;

    /**
    * Main invocation method for identifier plug-in
    */
    public Collection<InformationSet> identify(Query theQuery,
            Contract contract, Context context) {

        if (once) { return null; }
        once = true;

        // the plug-in code
        System.out.println("Hello from identifier!");
        //dummy return
        Collection<InformationSet> articleSet = new HashSet<InformationSet>();
        articleSet.add(new NaturalLanguageDocumentUrl("Some url"));
        return articleSet;
    }

    @Override
    public Context createContext() {
        return null;
```

Figure 17: Eclipse with an Identifier plug-in in the workspace

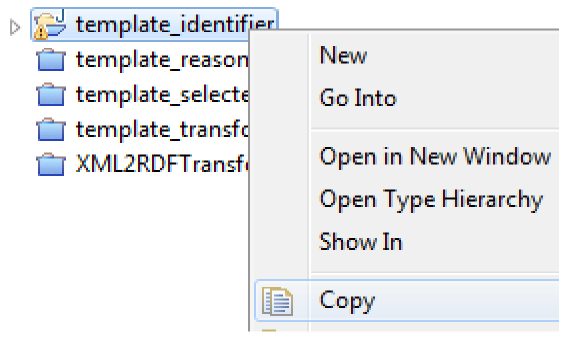
After defining all properties of the plug-in, the source will be created by clicking on the button "Finish". All necessary files are created, together with an implemented structure of the plug-in in the file "<project name>.java" (see Figure 17). Users can now add their own code (i.e., specific algorithms) in order to make the plug-in do what it is supposed to do.

4.5.2. By hand

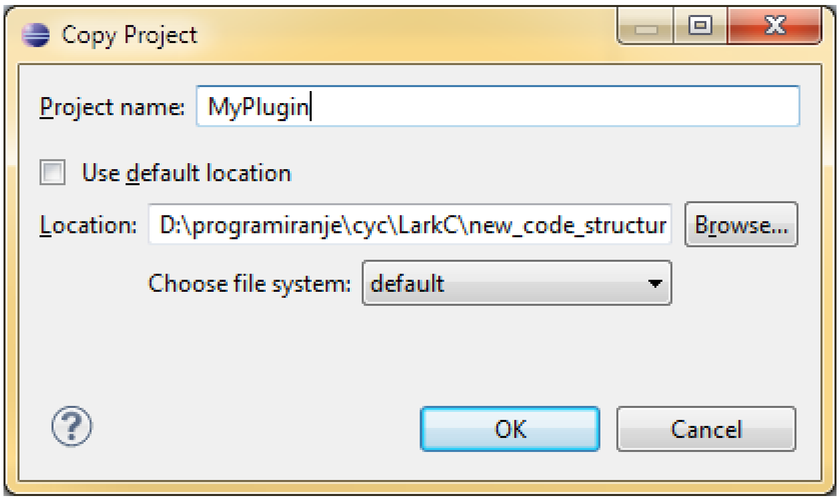
The prerequisite to develop a new LarKC plug-in is that first you are able to compile and run the Platform with example plug-ins (see section 3).

The easiest way to create a new plug-in is to copy one of the template plug-ins provided within the redistributable package (or dowloadable through the LarKC source code repository). You can recognize them by the "template_" prefix (see Figure 6).

To create a new plug-in project, inside Package Explorer in Eclipse, right click the appropriate template project and click Copy and then Paste.



In the dialog box that opens, specify your plug-in name, ie. "MyPlugin", and the location where it should be allocated. Please note that "Use default location" is not always the best option, especially when you want to position your plug-in into the existing directory structure. Also make sure that the plug-in will be located in its own directory.


Copy Project

Project name: MyPlugin

Use default location

Location: D:\programiranje\cyc\LarkC\new_code_structur

Choose file system: default

At this point you should have a copy of template_### plug-in project, with at least four (4) files in it:

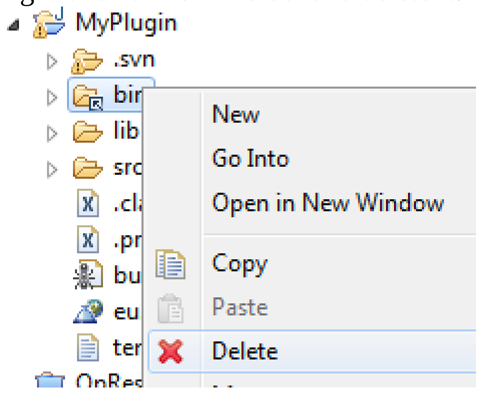
- template_###.java,
- eu.larkc.plugin.###.template_###.wsdl,
- template_###.rdf
- build.xml

To make easier to test and run your plug-in, you have to tell your IDE, how to deploy it to the Platform.

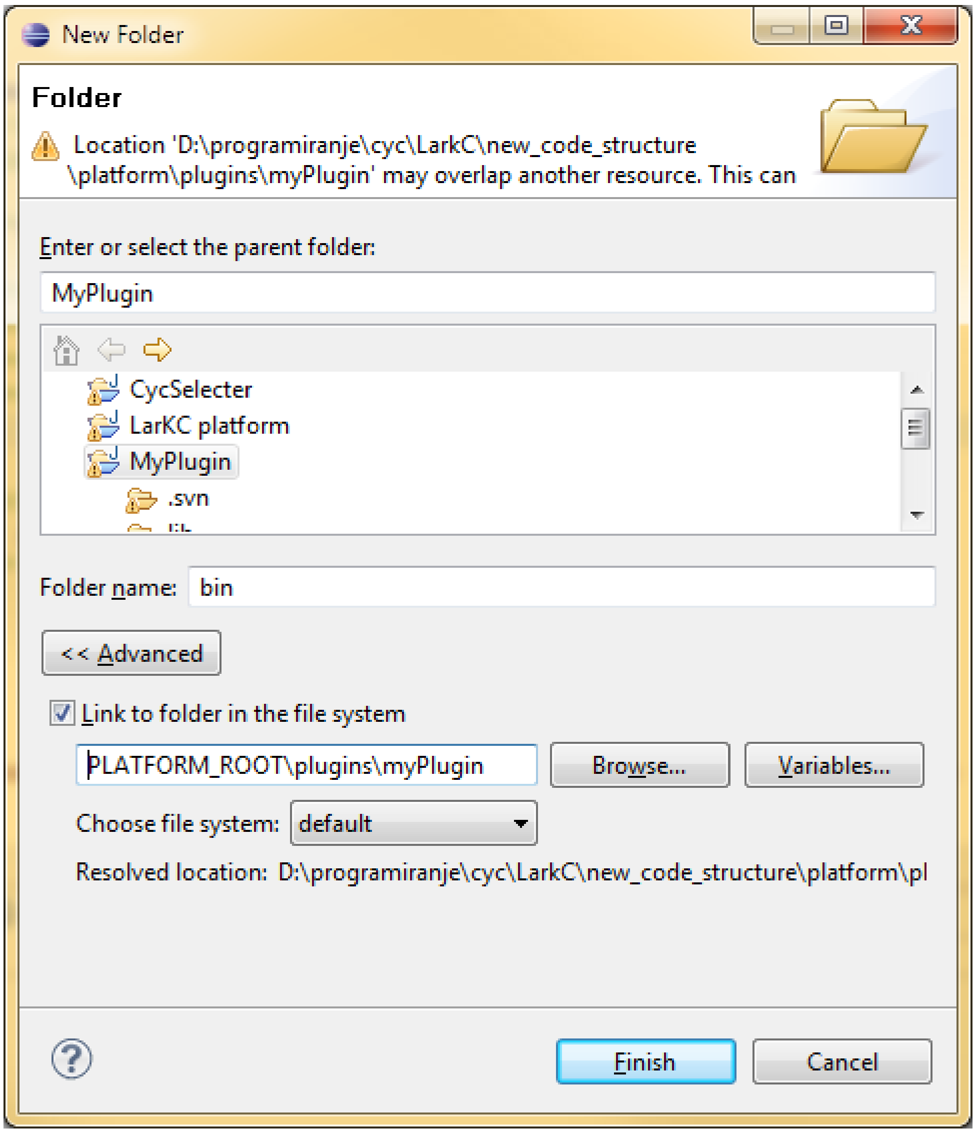
If you use Ant to compile and deploy the plug-in, you have to make sure that you placed the code correctly into the LarKC directory structure, and then just run "ant dist"

For Eclipse you have to specify the build output directory, so the plug-in will be deployed into Platform's plug-ins directory. After you copied the template_.. project, it is pointing to PLATFORM_ROOT/Plugins/template_### directory.

Unless you want it to point there, first step is to delete the old link. Go to Navigator view, right click on "bin" folder and delete it:



Next, you have to re-create the link you just deleted, but with correct plug-in name. To do that, right click the MyPlugin project -> new -> Folder. In the window that opens, please write "bin" as a Folder name, click Advanced and then check the "Link to folder in the file system" check box and add this to the empty text box: "PLATFORM_ROOT\plugins\myPlugin", where myPlugin is the name of your plug-in:


New Folder

Folder

Location 'D:\programiranje\cyc\LarkC\new_code_structure \platform\plugins\myPlugin' may overlap another resource. This can

Enter or select the parent folder: MyPlugin

Folder name: bin

Link to folder in the file system [✓]

PLATFORM_ROOT\plugins\myPlugin

Choose file system: default

Resolved location: D:\programiranje\cyc\LarkC\new_code_structure\platform\pl

After the steps described above, you should have your own plug-in project. When you build the project, the IDE should deploy it automatically within the Platform, which is running in the IDE.

Before running the plug-in in the Platform you should rename the files from template_### to the final plug-in name, and also update the information about the plug-in name and endpoint inside the files corresponding to the plug-in project.

In case your plug-in uses external libraries, they must be copied within a folder named “lib”, within “myPlugin” folder. This structure has the purpose to avoid conflicts between plug-ins using different versions of the same library.

WSDL file

This file is similar for all plug-ins. Usually the only update needed in the template is the plug-in name. The file looks like this:

```
<wsdl:description>
  <!-- COMMON TO ALL PLUGIN-TYPES -->
  <wsdl:interface name="plugin-type" sawsdl:modelReference="http://larkc.eu/plugin#Plugin-type">
  </wsdl:interface>

  <wsdl:binding name="larkcbinding" type="http://larkc.eu/wsdl-binding" />
  <!-- SPECIFIC TO THIS PLUGIN -->
  <wsdl:service name="urn:eu.larkc.plugin.identify.myPlugin" interface="plugin-type"
     sawsdl:modelReference="http://larkc.eu/plugin#MyPlugin" >
       <wsdl:endpoint location="java:eu.larkc.plugin.identify.MyPlugin" />
   </wsdl:service>
</wsdl:description>
```

Where "plugin-type" is one of the five (5) LarKC plug-in types (identify, select, transform, reason, or decide). If you copied the file from the appropriate template, this field should be already filled in.

The important thing to notice are sub-tags under <wsdl:service> tag:

- *name* is the URI of the plug-in, which must be unique
- *modelReference* is the location of the RDF description of the plug-in. If the Platform cannot find the RDF file, it will try to go online to find it on this location
- *location* tells the location of the executable. If it is a Java class, it starts with "java:" prefix, if it is a web service it starts with "http:"

RDF file

For the RDF file please make sure that the line where you define the plug-in corresponds to the plug-in URI. Furthermore, the input and output data types must be specified correctly, according to the types the plug-in itself is accepting and returning. The .RDF file looks like follows:

```
@prefix larkc: <http://larkc.eu/plugin#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

larkc:MyPlugin
rdf:type                  rdfs:Class ;
rdfs:subClassOf          larkc:plugin-type ;
larkc:hasInputType    larkc:SPARQLQuery ;
larkc:hasOutputType larkc:NaturalLanguageDocument.
```

The plug-in description ontology we use here is currently not widely used by existing plug-ins. In future, it will be convenient to add further non-functional descriptions to the plug-in, in order to allow the workflow designer a better selection among the available plug-ins. For more details, see the plug-in description ontology, at /platform/conf/larkc.rdf, for more details.

Java file

This is the file where the plug-in code is located. Each plug-in type has its own default method, which is provided within the corresponding template project.

Note that, currently, the getIdentifier and QoSInformation methods must be implemented due to the interface definition. In future releases this will be replaced completely by the Plug-in Registry features:

```
public URI getIdentifier() {
    return new URIImpl("urn:" + this.getClass().getName());
}

public QoSInformation getQoSInformation() {
    // QOS code here
    return null;
}
```

4.6. How to compose a local Workflow

Currently to construct your local workflow, you need to create your own DECIDER plug-in (see the chapter 4.5 to learn how to write a plug-in). This will be, for simple pre-defined workflows, replaced by a scripted decider, which is currently still under development. But, in some cases it will be necessary to create more complicated workflows, or some other processing while manipulating the plug-ins.

The java interface for the DECIDER plug-in is slightly different than for other plug-ins. It has four (4) entry point methods, one for each type of SPARQL query:

```
publicVariableBinding sparqlSelect(SPARQLQuery theQuery, QoSParameters
        theQoSParameters);

public SetOfStatements sparqlConstruct(SPARQLQuery theQuery, QoSParameters
        theQoSParameters);

public SetOfStatements sparqlDescribe(SPARQLQuery theQuery, QoSParameters
        theQoSParameters);
public BooleanInformationSet sparqlAsk(SPARQLQuery theQuery, QoSParameters
        theQoSParameters);
```

One of these methods is called each time the Platform is queried. The easiest way to construct a workflow, independently of the entry method, is to use the Workflow class, as follows:

```
System.out.println("Composing workflow ...");
Workflow workflow= new Workflow ();
workflow.addPlugIn(new URIImpl("urn:eu.larkc.plugin.identify.template_identifier"));
workflow.addPlugIn(new URIImpl("urn:eu.larkc.plugin.transform.template_transformer"));
workflow.addPlugIn(new URIImpl("urn:eu.larkc.plugin.select.template_selecter"));
workflow.addPlugIn(new URIImpl("urn:eu.larkc.plugin.reason.template_reasoner"));

System.out.println("Starting workflow ...");
try {
        workflow.start(theQuery);
} catch (Exception e) {
        logger.severe(e.getMessage());
        return null;
}
return (VariableBinding) workflow.take();
```

In this release of the Platform, the Workflow class can only construct linear workflows. Future releases will support branching of the plug-in data flow.

Development of error handling mechanisms is currently up to the workflow developer. Future releases will provide a centralized mechanism (and developer's best practices guidelines) to deal with errors, which may occur at runtime.

4.6.1. The LarKC Data Layer

The LarKC Data Layer is an RDF middleware that enables the workflow designer to pass efficiently data between different plug-ins. A more detailed presentation of the Data Layer features and how to use them is given in section 4.8. This section covers the decisions to be taken, with regards to the Data Layer, at workflow level. The typical workflow execution always requires at some point a Plugin A to pass information encoded into RDF to Plugin B. The Data Layer supports three different strategies of doing it (see Figure 18):

a) Pass the RDF data by value: this is the classical workflow approach where the complete RDF graph is transferred from the execution context of Plugin A to Plugin B; if Plugin B resides in a different virtual machine this would requires data marshalling. The Data Layer transparently automates the processes of serialization and deserialization. However, it must be considered that they could take significant amount of resources and become easily a scalability bottleneck.

b) Pass the RDF data by reference using the Platform repository: this is a process where the RDF graph is first persisted and indexed in the local repository and then a pointer to the information is passed. Note that the mechanism is something more than a simple data transfer, because by default the information will remain stored in the repository after the completion of the execution of Plugin B. It is up to the workflow designer to choose what will be the most appropriate mechanism to clean up the data.

c) Pass the RDF data by reference using third party SPARQL endpoint or linked data: this is a similar process to option b), which uses a third party SPARQL endpoint or RDF retrievable data using HTTP. In the case of a SPARQL endpoint, Plugin A may select a dataset (group of named graph) or CONSTRUCT query to generate reference to the data. For linked data you can select only the full information exposed for a given URI.

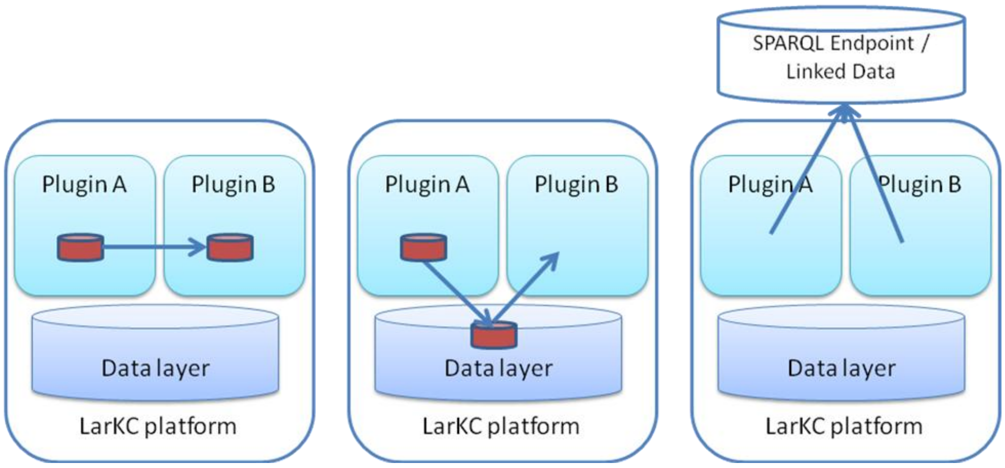


Figure 18: Approaches to pass data with LarKC data layer a) by value, b) by reference to the Platform repository and c) by reference to remote SPARQL endpoint or linked data

The LarKC Data Layer offers the infrastructure to realize many different use cases. Still, the infrastructure is not capable and not aimed to do the global data flow optimizations. Every workflow designer has to choose what is it the best mechanism to control the data flow. In Table 2 a high-level comparison between the different mechanisms of passing data is made. The purpose of the UML diagrams in Figure 25 and Figure 26 is to describe the behaviour of each concrete class. In section 4.8.2 there is also a code snippet example is shown, "Manipulate data from the Platform repository".

Table 2: Comparison of the approaches to pass data between plug-ins with the LarKC data layer

Passing Mechanism	**Pros**	**Cons**
By Value	• Easy to understand and manage • Guarantees data isolation	• Not scalable (limited to use case with small amounts of data[18])
By reference using the Platform repository	• Full control over the data flow (everything is in one place) • Only theoretically limited scalability • Fits very well scenarios	• Difficult to manage (data initialization and cleanup operation; no building isolation) • May be inefficient for very dynamic data (every

[18] 350MB memory is indicative for how much memory is required to pass 10^6 statements by value. There is no physical limitation except the memory that could be allocated by the JVM.

	in dealing relatively static data	time the data is also indexed; many read queries are required to compensate it)
By reference using third party SPARQL endpoint or linked data	• Very simple to setup and use • Simple way to integrate third party data services	• May be slow because of the remote data • Extremely inefficient queries which combine local with remote data

For the design of very data intensive workflows or operation with large scale information models it is possible to combine multiple data flow strategies. For example, a LarKC based application, which has to process large and relatively static information (e.g. the reads are many times more than the writes) and small highly dynamic data. In this case, the static data could be passed by reference to the Platform repository to all subsequent plug-ins and the cost to index and store the information will be minimal compared to the many read operations, which extract a small amount out of it. Thus, a lot of unnecessary information retrieval would be skipped. The highly dynamic data could be retrieve each time from external data service using SPARQL endpoint or linked data compatible REST service and process by value.

4.7. How to compose a distributed workflow

4.7.1. Introduction

Distributed execution is a core functionality of the LarKC's decentralized architecture, which allows the specified parts of a workflow to be executed on one or more remote resources (Figure 19).

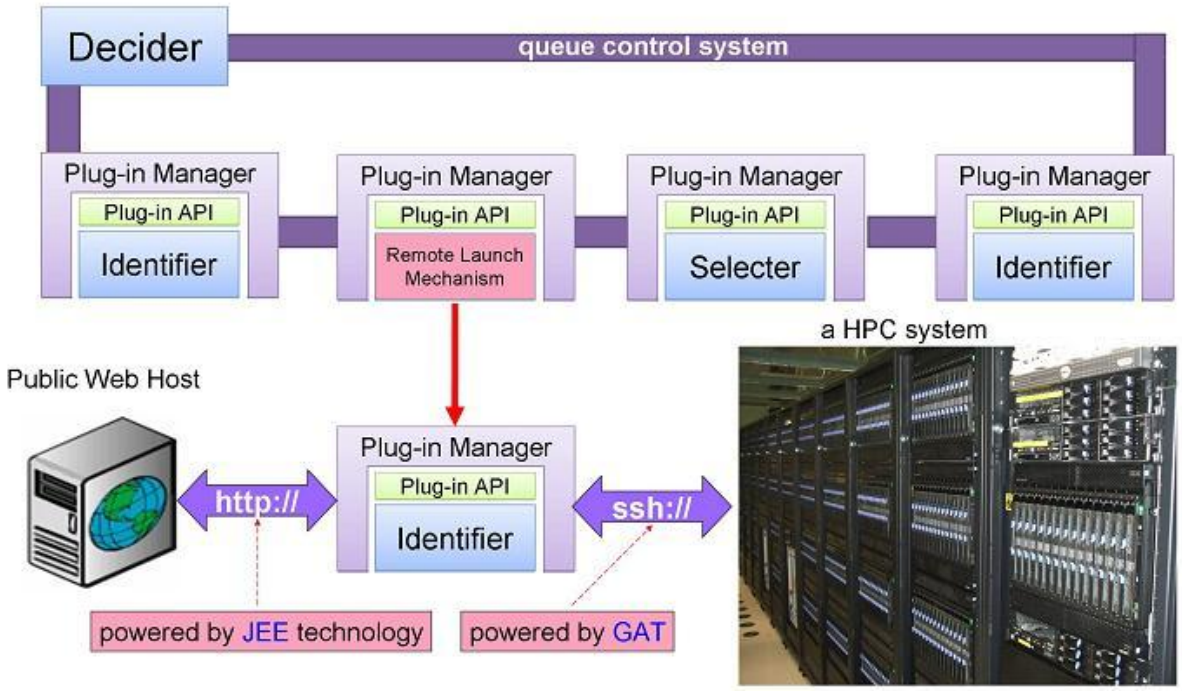


Figure 19: Remote Plug-In invocation schema

The motivation for adoption of distribution techniques by LarKC is twofold. On the one hand, distribution facilitates deployment of the workflow's components (plug-ins, workflow branches or even the entire workflow) on the architecture, which allows obtaining the best performance and scalability of the execution. For example, computation-intensive plug-ins implemented by means of the different parallelization approaches will greatly benefit from running on a high-performance computing cluster, whereas the plug-ins with intensive data exchange will perform better when being executed on the resource located close to the operated data (avoiding the data accessing overhead). On the other hand, considering distributed computing as a form of parallel computing, the distribution techniques will enable to increase the resource pool facilities available for the workflow execution (improving efficiency of the task-level parallelism). Figure 20 illustrates the application of distribution in order to increase the resource pool for task-level parallelism.

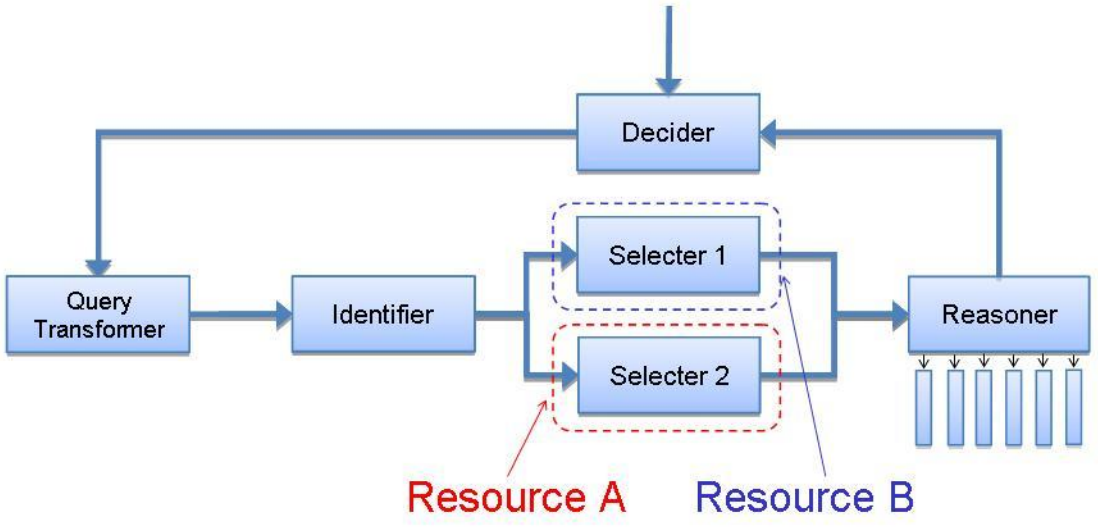


Figure 20: Distributed computing as parallelisation mechanism

The LarKC Platform supports the following computing architectures, which are of potential interest for deployment of the LarKC workflows:

- **any remote computer** capable of executing a LarKC-workflow;
- a **public web server** (which provides the service wrapping of LarKC component);
- a **high-performance computer** accessible via a dedicated front-end node (for complex and computation-intensive components);
- a site of **the grid/cloud infrastructure** available through a resource broker.

To enable the listed deployment resource types, LarKC has adopted two deployment strategies:

- **GAT (Grid Access Toolkit) container**,
- **Web Servlet container** (i.e., based on Tomcat).

In the following subsections, these technologies are described in more detail, focusing on usage in LarKC workflows.

4.7.2. Constructing a workflow with JavaGAT support

The Grid Application Toolkit and its Java implementation (JavaGAT) offer a set of unified, coordinated, generic API for remote accessing of resources of different types. The flexibility of GAT is achieved by means of Adaptors (see Figure 21) – generic mechanism which allows the user to perform all the operations on the remote resource in a unified and transparent way while adaptation to specific of the concrete resource is done with the aid of the corresponding Adaptor.

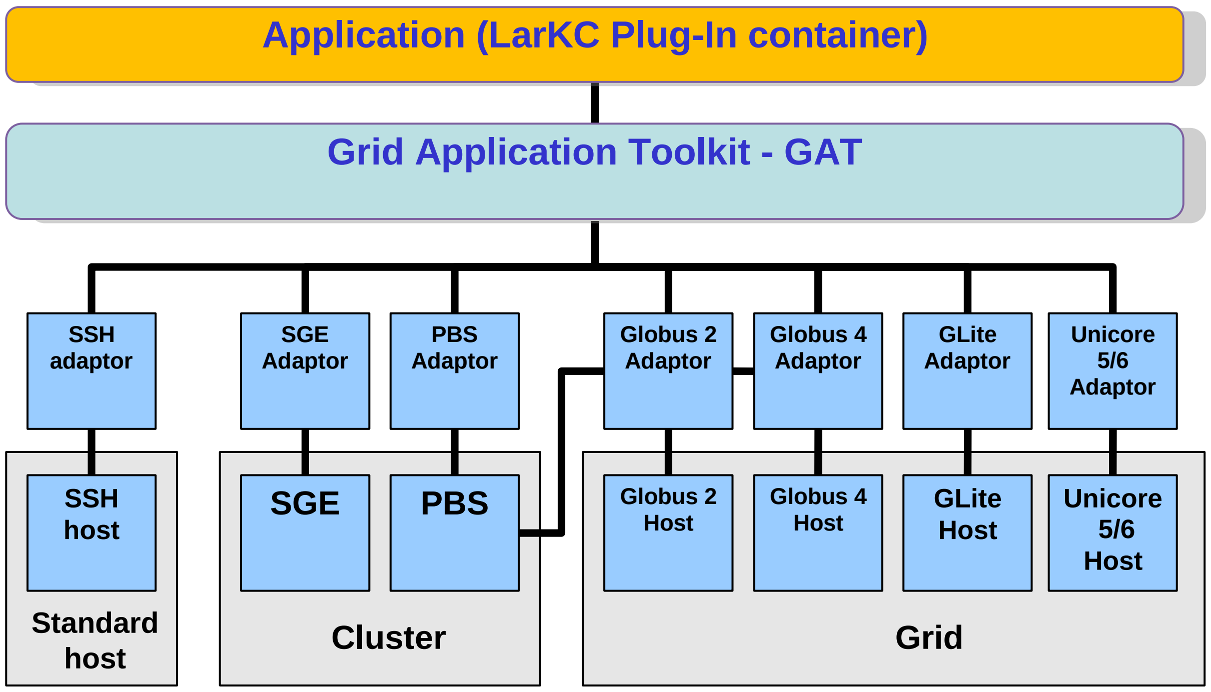


Figure 21: GAT Adaptors Architecture

In LarKC, use of JavaGAT is facilitated by means of the Remote Plug-in Managers which append the Local Plug-in Manager's functionality (see Section 2) to execute the managed Plug-in on the given remote resource.

Important: all the functionality for the remote plug-in execution is provided by the Remote Plug-in Managers and does not require a recoding of the particular plug-in.

The standard mechanism of the plug-in execution on the remote host includes the following main three steps:

1) Input data pre-staging – the data which are used by the Plug-in as input (*and which are obviously not available on the remote host)* are to be serialized on the local host and uploaded to the remote host (as a file containing the serialized input).
2) Plug-in execution – the serialized input data are read from the input file and passed to the plug-in which performs the given operation and saves the produced output data (if any) in the output file (in the serialized form).
3) Output data post-staging – the data produced by the Plug-in of the remote host are downloaded to the local host, deserialized and passed to the next plug-in (plug-ins) in the workflow.

Considering that for some use cases the above mentioned standard execution mechanism can involve not all of the listed steps (for example, if the Plug-in's output data are stored in an external data base and there is thus no need to post-stage the output data to the local host after the plug-in execution, i.e., the step 3 can be omitted), the GAT Remote Plug-in Manager Architecture provides three (despite of only one for the case of the local execution) Manager types, corresponding to the above mentioned remote plug-in execution phases:

1) Pre-stage Plug-in Manager
2) Execution Plug-in Manager
3) Post-stage Plug-in Manager

Let's consider the example of the following Plug-in which might be executed remotely (from the AlphaUrbanLarKC workflow), see Figure 22.

```
InformationSetTransformer x2r =
    (InformationSetTransformer) Larkc.pluginRegistry.getNewPluginInstance
        (new URIImpl("urn:eu.larkc.plugin.transform.urbancomputing.ubl.
                XML2RDFTransformer"));

transformer = new LocalCollectionInformationSetTransformManager(
                        x2r, identifyOutputQueue, transformerOutputQueue);

transformer.setPrevious(identify);
transformer.start();
```

Figure 22: Example of a local plug-in manager use

The very first step which is needed to execute the presented Plug-in remotely is the description of the remote resource, including for example the resource URI, access policy etc. In the current edition of the LarKC platform, the information about the resource is represented in the simple class GATResource. Table 3 illustrates the basic fields of the GATResource class.

Field name	Type	Description
ResourceURI	String	URI of the remote resource
FileAdaptor	String	The GAT File Adaptor used for input/output data pre-/post-staging. The following GAT File Adapters have been tested and supported so far: ▪ “local”: used for local host in the emulation mode (used mainly for tests) ▪ “CommandlineSSH”: used for the host supporting data up/downloading per SCP protocol ▪ “gt4gridftp”: used for any grid site supporting Globus Toolkit 4.0 middleware
Broker	String	The GAT Resource Adaptor used for remote plug-in launching. The following GAT Resource Adapters have been tested and supported so far: ▪ “local”: ”: used for local host in the emulation mode (used mainly for tests) ▪ “CommandlineSSH”: used for the host accessible per SSH protocol ▪ “Wsgt4new”: used for any grid site supporting Globus Toolkit 4.0 middleware
LarKC_Location	String	The path on the file system of the remote host where the LarKC platform is installed
WorkDir	String	The path on the file system of the remote host accessible for the user read/write operations (needed only for storing input and output files)
InputDataID	String	In case if the plug-in takes input data: the file identificator with the input data
OutputDataID	String	In case if the plug-in produces output data: the file identificator with the output data
JavaDir	String	The Java location on the file system of the remote host

Table 3: The GATResource class fields

Below some examples of the Resource description are collected (Figure 23a-c).

```
String ResourceURI="localhost";
String FileAdaptor="local";
String Broker="local";
String LarKC_Location="/home/user/LarKC-trunk”;
String WorkDir="/home/user/LarKC-temp";
String InputDataID="input1";
String OutputDataID="output1";
String JavaDir="/usr/bin";
```

a) The user’s local machine used to emulate the remote machine

```
String ResourceURI="gt4.frontend.dgrid.de";
String FileAdaptor="gt4gridftp";
String Broker="wsgt4new";
String LarKC_Location="/home/user/LarKC-trunk”;
String WorkDir="/home/user/LarKC-temp";
String InputDataID="input2";
String OutputDataID="output2";
String JavaDir="/usr/bin";
```

b) a grid site supporting Globus Toolkit 4 grid middleware

```
String ResourceURI="somemachine.dgrid.de";
String FileAdaptor="CommandlineSSH";
String Broker="CommandlineSSH";
String LarKC_Location="/home/user/LarKC-trunk”;
String WorkDir="/home/user/LarKC-temp";
String InputDataID="input3";
String OutputDataID="output3";
String JavaDir="/usr/bin";
```

c) a machine available through the SSH/SCP access protocols

```
GATResource resource = new GATResource
            (ResourceURI, FileAdaptor, Broker,
             LarKC_Location, WorkDir, JavaDir);
```

d) GATResource class instantiation

Figure 23: Examples of the diverse resource descriptions

After the remote resource is described and the corresponding GATResource class is instantiated (Figure 23d), a sequence of the Remote Plug-in Managers can be created. The syntax of the Remote Plug-in Manager instantiation is similar to the one used for the Local Managers, despite of the extra parameter needed to handle the GAT Resource Description. The typical Remote Plug-in Manager chain corresponding to example in Figure 22 is depicted in Figure 24.

```
//Step1. Data prestaging
prestage_transformer = new RemoteCollectionInformationSetTransformManagerPrestage(
    x2r, identifyOutputQueue, prestage_transformerOutputQueue, resource, InputDataID);

prestage_transformer.setPrevious(identify);
prestage_transformer.start();

//Step2. Plug-in run
execute_transformer = new RemoteCollectionInformationSetTransformManagerExecute(
    x2r, prestage_transformerOutputQueue, execute_transformerOutputQueue, resource,
    OutputDataID);

execute_transformer.setPrevious(prestage_transformer);
execute_transformer.start();

//Step3. Data poststaging
transformer = new RemoteCollectionInformationSetTransformManagerPoststage(
    x2r, execute_transformerOutputQueue, transformerOutputQueue, resource, OutputDataID);

transformer.setPrevious(execute_transformer);
transformer.start();
```

Figure 24: Example of the Remote Plug-in Manager use

As mentioned above, for some particular workflows the steps 1 and 3 can be omitted, which should be considered in the workflow description, accordingly.

Details of the current implementation

- The Remote Plug-in managers can be found in the package eu.larkc.core.pluginManager.distributed.GAT
- To have a quick try of the GAT technology, please look at the code located in eu.larkc.plugin.decider.urbancomputing.ubl.AlphaUrbanLarkcDecider and enable the remote execution by switching on the flag DISTRUBUTED_EXECUTION_ENABLE=true. We recommend you to try out first the local emulation mode specifying the corresponding resource description.

Requirements to the remote host

- Pre-deployed LarKC platform with the plug-in, which is to be executed on this host.
- Valid user certificates/proxies/credentials for accessing the remote host.

Limitations of the current realization

- GAT supports only job-based plug-in invocation. This means the user cannot influence the job execution, i.e. change the input data or execution parameters at the job's run-time. However, this limitation pertains not to GAT, but mainly architectures supported by GAT, i.e. cluster and grid environments.
- The current implementation is a pilot prototype done only for CollectionInformationSetTransformManager. Taking into account the oncoming Plug-in API redesign, which should appear in the next public release of the LarKC, the implementation for other Plug-in Manager types has not been performed yet. However, this process is straightforward and can be performed with the minimum of development efforts.
- Only certificate-based access to the remote host is supported now. This means, the user cannot use authorization based for example on his login and password. This feature will be provided in the next release of the LarKC.

- The Resource Registry will be introduced as the new Platform Service for handling the GAT Resource Description (currently performed manually using the GATResource class as described above).

4.7.3. Constructing a workflow by means of Java servlets

Composing a workflow using the servlet approach is very similar to creating a local workflow. The JEE extensions provide proxy classes for all plug-in types which behave just like local plug-ins. To set up a workflow, which uses remotely deployed plug-ins the plug-ins have to be deployed and the location of the deployed plug-ins has to be known.

The following example will illustrate how to create a simple workflow using a local decider and four deployed plug-ins:

- Query transformer (SPARQLToTriplePatternQueryTransformer)
- Identifier (SindiceTriplePatternIdentifier)
- Selecter (GrowingDataSetSelecter)
- Reasoner (SparqlQueryEvaluationReasoner)

For demonstration purposes those four plug-ins are deployed on a Tomcat server running locally.

```
SimpleAnytimeDecider decider = new SimpleAnytimeDecider(

 new QueryTransformerJeeProxy(

   "http://localhost:8080/SPARQLToTriplePatternQueryTransformer/query_transformer" ),

 new IdentifierJeeProxy(

   "http://localhost:8080/SindiceTriplePatternIdentifier/identifier" ),

 new SelecterJeeProxy(

    "http://localhost:8080/GrowingDataSetSelecter/selecter" ),

 new ReasonerJeeProxy(

    "http://localhost:8080/SparqlQueryEvaluationReasoner/reasoner" ) );
```

The plug-in proxy classes take one argument as a parameter, namely the URL where the plug-in is deployed. The URL consists of the deployment location followed by the plug-in[19] type.

If you want to deploy standard LarKC plug-ins on a servlet container the plug-ins have to be wrapped as web archive files and deployed on a servlet container (e.g. Apache Tomcat[20]). The following section will guide you through those preliminary steps.

Wrapping LarKC plug-ins inside a web application archive

This section should guide you through the process of wrapping a LarKC plug-in inside a .war (web application archive) file, which then can be deployed on a servlet container. In the following example the servlet container used is Apache Tomcat.

Please note that some commands used throughout this section, like `cp` and `mv`, are operating system specific and may differ on other operating systems than Linux. The Windows counterparts of `cp` and `mv` would be `copy` and `move` respectively. Additionally the Linux file separator is a forward slash (`example/example.jar`) as opposed to a backslash in Windows (`example\example.jar`). Please change the commands and file separators accordingly.

[19] Can be one of the following: data_transformer, query_transformer, identifier, selecter, reasoner
[20] http://tomcat.apache.org/

The first step is converting the `.larkc` plug-ins, which are to be deployed, to `.war` files using the `larkc_jee_converter` of the JEE extensions.

```
Usage:

java -jar larkc_jee_converter.jar

 {plugin.larkc}

 {plugin.war}

 {full.java.class.name}

 {query_transformer|identifier |data_transformer|selecter|reasoner}
```

The following example runs inside the LarKC root directory[21] and builds the SindiceTriplePatternIdentifier plug-in.

```
java -jar jee_extensions/build/jar/larkc_jee_converter.jar

    platform/dist/plugins/SindiceTriplePatternIdentifier.larkc

    SindiceTriplePatternIdentifier.war

    eu.larkc.plugin.identify.sindice.SindiceTriplePatternIdentifier

    identifier
```

Running the command provided above should produce some output which indicates the process of the conversion as well as information about which files are copied/moved. The produced output should look as follows:

```
===============================================================

INPUT:  platform/dist/plugins/SindiceTriplePatternIdentifier.larkc
(identifier)

OUTPUT: SindiceTriplePatternIdentifier.war
(eu.larkc.plugin.identify.sindice.SindiceTriplePatternIdentifier)

---------------------------------------------------------------

Copying: META-INF/MANIFEST.MF

Copying: SindiceTriplePatternIdentifier.rdf

Copying:
eu.larkc.plugin.identify.sindice.SindiceTriplePatternIdentifier.wsdl

Moving: eu/larkc/plugin/identify/sindice/AbstractSindiceIdentifier$1.class
=> WEB-

...

... (some output omitted)
```

[21] The LarKC root directory refers to the directory which holds the platform, plugins as well as the jee_extensions directory, among others (demos, development_kit)

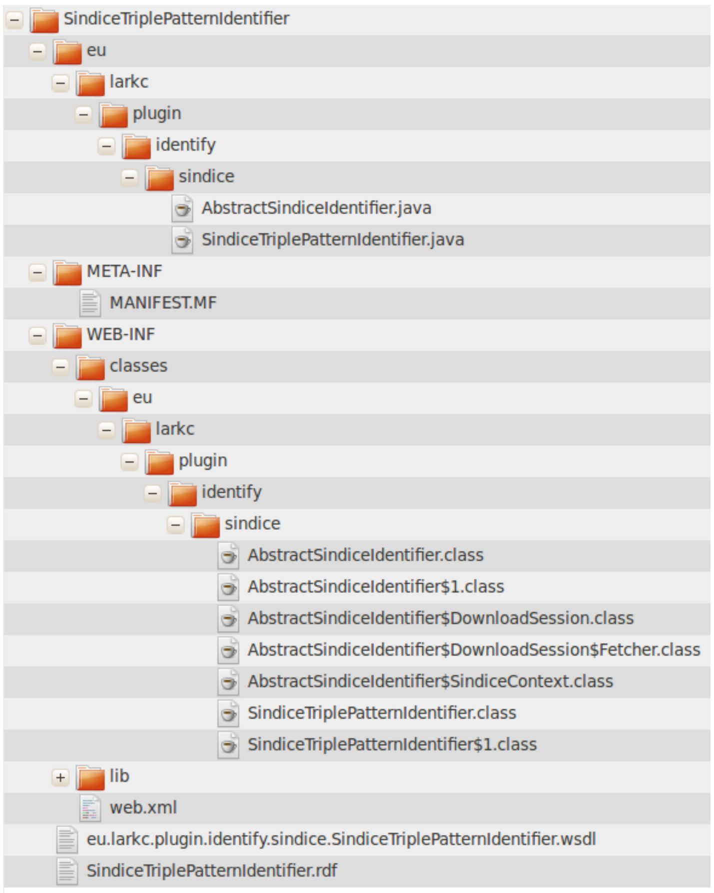
```
...

Copying: webapp_lib/slf4j-log4j12-1.5.3.jar ==>> WEB-INF/lib/slf4j-log4j12-
1.5.3.jar

Creating: WEB-INF/web.xml

FINISHED.
```

The last line saying `FINISHED` indicates that the conversion process was successful. If something went wrong the conversion tool will respond with an appropriate error such as "error in opening zip file" if the LarKC plug-in cannot be read or is corrupt.

The generated .war file should contain the following folders and files:

- META-INF folder containing the manifest file
- WEB-INF folder containing the plug-in classes and libraries used
- A `.wsdl` and a `.rdf` file describing the plug-in

The directory structure of the example provided above can be seen in figure 19. If the plug-in was successfully converted to a `.war` file it can be deployed on the Tomcat server.



```
− SindiceTriplePatternIdentifier
  − eu
    − larkc
      − plugin
        − identify
          − sindice
              AbstractSindiceIdentifier.java
              SindiceTriplePatternIdentifier.java
  − META-INF
      MANIFEST.MF
  − WEB-INF
    − classes
      − eu
        − larkc
          − plugin
            − identify
              − sindice
                  AbstractSindiceIdentifier.class
                  AbstractSindiceIdentifier$1.class
                  AbstractSindiceIdentifier$DownloadSession.class
                  AbstractSindiceIdentifier$DownloadSession$Fetcher.class
                  AbstractSindiceIdentifier$SindiceContext.class
                  SindiceTriplePatternIdentifier.class
                  SindiceTriplePatternIdentifier$1.class
    + lib
      web.xml
    eu.larkc.plugin.identify.sindice.SindiceTriplePatternIdentifier.wsdl
    SindiceTriplePatternIdentifier.rdf
```

Figure 19: File structure of the SindiceTriplePatternIdentifier plug-in which was wrapped inside a .war file.

Deploying the .war file

There are basically two options of deploying a .war file on a Tomcat server. If "auto-deployment" is enabled (enabled by default when using Tomcat 5 or newer versions) it is sufficient to simply copy the `.war` file to the `webapps` folder of the installation directory. The second option is to use the integrated *Tomcat Web Application Manager*.

1) Copy or move the generated `.war` file to the Tomcat `webapps` folder.

If “auto-deployment” is enabled the .war file will be extracted and deployed automatically as soon as it is placed in the `webapps` folder. Using the Linux `mv` command the file operation may be initiated like this:

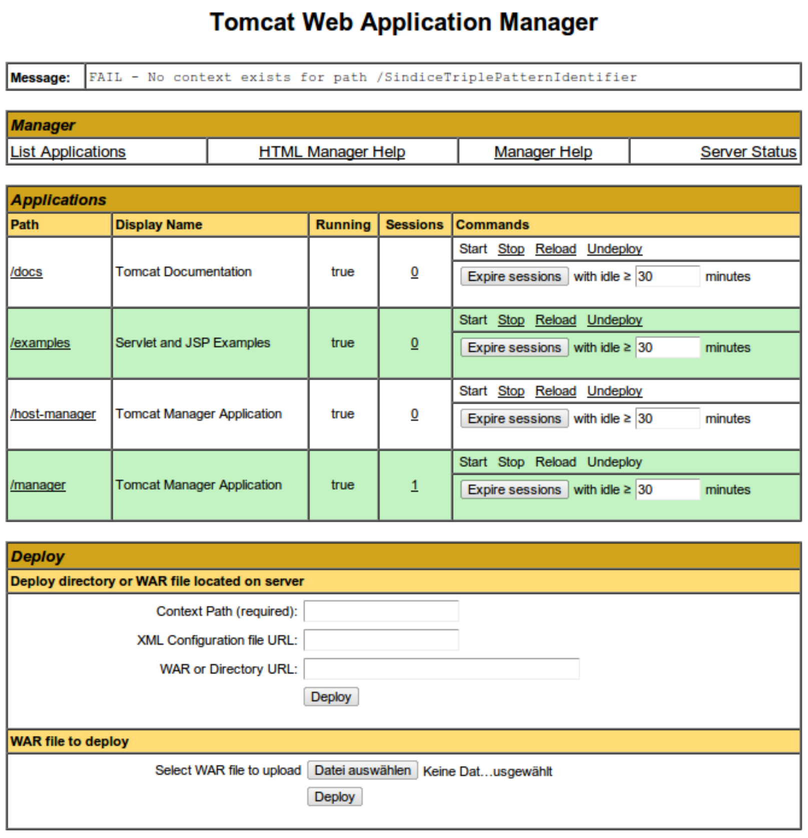
```
mv SindiceTriplePatternIdentifier.war /var/lib/tomcat6/webapps/
```

Please note that `/var/lib/tomcat6/webapps/` is the default web application path of Tomcat6 on many Linux distributions including Ubuntu 9.10 as well as Ubuntu 10.04. Please change the path accordingly if your Tomcat server is installed in a different directory.

Using the integrated Tomcat Web Application Manager

Tomcat comes with a integrated *Web Application Manager* interface which enables you to view deployed web applications and deploy `.war` files.

To access the *Web Application Manager* interface point the browser of your choice to your tomcat server followed by /manager/html/ (e.g. http://localhost:8080/manager/html/). An interface similar to the one shown in Figure 20 should be seen.


Tomcat Web Application Manager

Message:	FAIL - No context exists for path /SindiceTriplePatternIdentifier

Manager			
List Applications	HTML Manager Help	Manager Help	Server Status

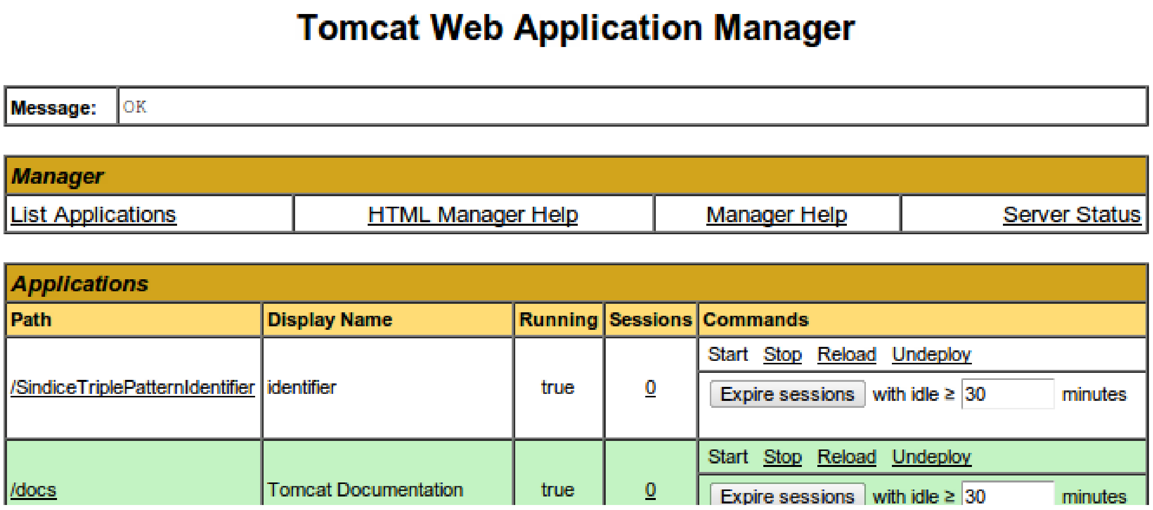
Path	Display Name	Running	Sessions	Commands
/docs	Tomcat Documentation	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/examples	Servlet and JSP Examples	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/host-manager	Tomcat Manager Application	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/manager	Tomcat Manager Application	true	1	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

Deploy
Deploy directory or WAR file located on server
Context Path (required):
XML Configuration file URL:
WAR or Directory URL:
Deploy

WAR file to deploy
Select WAR file to upload Datei auswählen Keine Dat…usgewählt
Deploy

Figure 20: Tomcat Web Application Manager

To deploy a `.war` file on the server click on the “Select file” button (or phrased similarly, since this caption depends on what browser and operating system you are using) and select the .war file, which you want to deploy. Clicking on “Deploy” will deploy the web application archive on the server. The deployed web application will show up in the manager shortly after deployment (see Figure 21).



Tomcat Web Application Manager

Message:	OK

Manager			
List Applications	HTML Manager Help	Manager Help	Server Status

Applications				
Path	Display Name	Running	Sessions	Commands
/SindiceTriplePatternIdentifier	identifier	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/docs	Tomcat Documentation	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

Figure 21: Tomcat Web Application Manager

You can check if the plug-in was deployed correctly regardless of the method of deployment. The plug-in will also show up in the *Tomcat Web Application Manager* if you simply copied/moved the `.war` file to the webapps folder.

4.8. How to interact with the Data Layer

This section describes the interaction with the Data Layer on a plug-in level. In this section we will cover the Java types, how to pass data by value or reference between the plug-in instances and how to operate with remote data. The Data Layer uses the following libraries, each of which gives a different abstraction level and purpose:

a) OpenRDF.org[22]: an open-source community site that supports multiple projects, which are aimed to automate RDF data management. It supports basic RDF types and RDF repository framework.
b) OWLIM[23]: a highly efficient implementation of the abstract storage and inference layer defined by OpenRDF.org. It also features parts of plug-in functionality that require very data intensive operation (rule based reasoning or selection). It is an RDF repository implementation, supporting inference, extended SPARQL features, etc.
c) ORDI[24]: an extension of the RDF model, which introduces new modelling primitives and supports passing by reference functionality and meta-data association.

All libraries share a common underlying implementation, BigTRREE. Every plug-in developer can choose any third party library and use it. The LarKC Data Layer is designed to support efficient selection mechanisms (i.e. query specific subgraphs).

The process of efficient passing of arbitrary RDF types requires internal extension of the data model. ORDI introduces a new modelling primitive named “labelled group of statements” or sometimes just referred to as tripleset, which denotes a logical group to associate meta-data on a per-statement level. The labelled group of statements is defined as follows:

Definition: (Labelled group of statements) $< S, P, O, NG, TS1, \ldots, TSn >$

where NG is an Internationalized Resource Identifier (IRI) to contextualize the statement (i.e. named graph). It has not formal semantics but can be used to specify the provenance of the data {TS1, . . . ,TS*n*} is a set of IRIs to associate the statement to additional logical groups called labelled group of

[22] http://www.openRDF.org
[23] http://www.ontotext.com/owlim/
[24] http://www.ontotext.com/ordi/

statements, which can be regarded as view on a set of triples. Thus, each statement could be member of multiple groups.

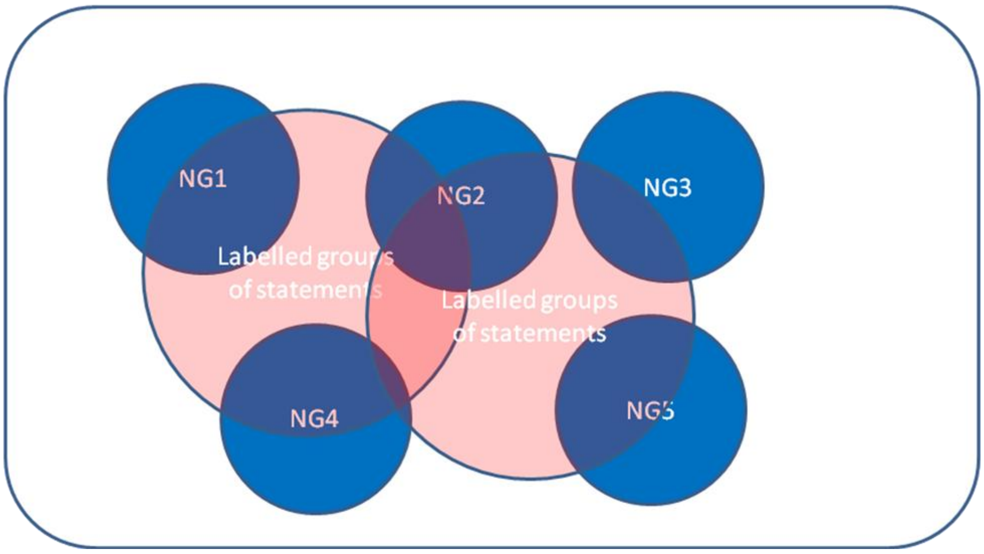


Figure 22: The relation between named graph (NG) and labelled group of statements

The packages eu.larkc.core.data.* and eu.larkc.core.query.* contain all Java types that are part of the Data Layer. They could be classified in multiple groups:

- RDF data exchanged by the plug-ins: hides the complexity in passing RDF statement between plug-ins
- RDF providers: persistently store information
- RDF queries and results: retrieve information from the RDF providers
- Utility methods: implement complex operation like querying remote data

Please note that a Java type could be appearing into multiple categories. For instance, `eu.larc.core.data.RdfGraph` is a result of SPARQL DESCRIBE or CONSTRUCT query and plug-in data exchange structure.

4.8.1. RDF data exchanged by the plug-ins

The super type interface for all RDF passing types is a generic eu.larkc.core.data.SetOfStatements interface, which supports only simple statement iteration. Please note that this type and all its subtypes are used to exchange information between plug-ins but not to interact with them because their interfaces are too generic and low level. There is no single way how to exchange RDF data if we consider all application specific requirements. The graph information could be in memory, stored in a rich-RDF model that supports statement grouping or exposed by a SPARQL compliant endpoint. Table 4 presents different mechanisms to exchange RDF data, based on the information contained in the object and the size of its data marshalling footprint.

Table 4: RDF data types exchanged by the plug-ins

RDF data types	**Mechanism to identify statements**	**Example**	**Passing mechanism**	**Serialization size of 10^6 RDF statements**
Set of statement	All statement parameters	s1, p1, o1, ng1 s2, p2, o2 s3, p3, o3, ng3, {group1}	By value	350 MB

RDF graph	Forth parameter (i.e. context or named graph)	s1, p1, o1, ng1, {group1}
		s2, p2, o2, ng1, {group2}
		s3, p3, o3, ng1
			By value /
			By reference to repository
				350 MB or few KBs

Dataset (SPARQL dataset)	List of allowed values (i.e. contexts or named graph)	s1, p1, o1, ng1 s2, p2, o2, ng2 s3, p3, o3, ng3	By reference to repository	Few KBs

Labelled group of statements	Group associated with	s1, p1, o1, ng1, {group1}
		s2, p2, o2, ng2, {group1}
		s3, p3, o3, ng3, {group1}
			By reference to repository	Few KBs

On Figure 25 a detailed description is given of all classes, which realize the different types of RDF data types exchanged by plug-ins.

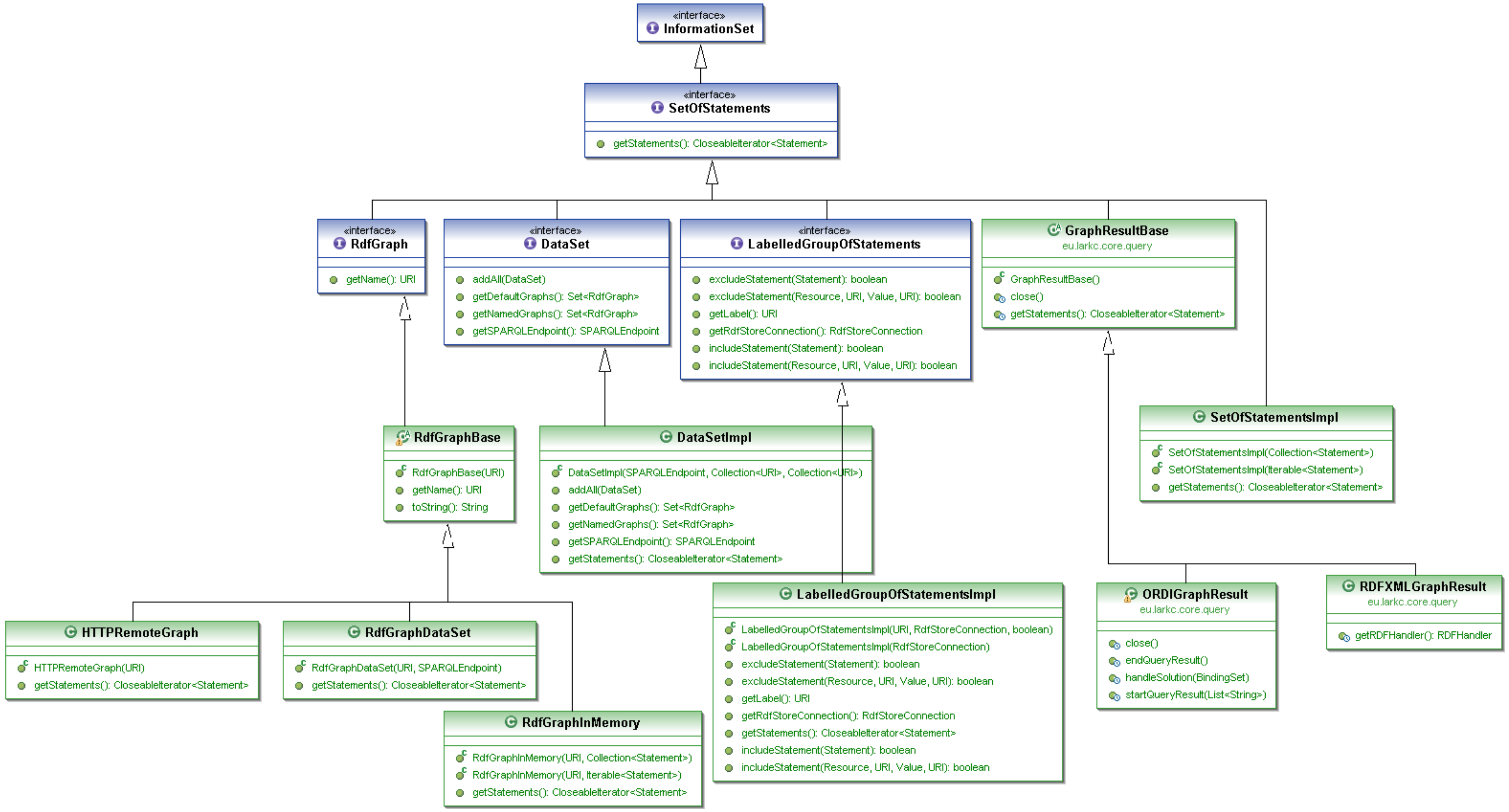

«interface» InformationSet
«interface» SetOfStatements
getStatements(): CloseableIterator<Statement>
«interface» RdfGraph
getName(): URI
«interface» DataSet
addAll(DataSet)
getDefaultGraphs(): Set<RdfGraph>
getNamedGraphs(): Set<RdfGraph>
getSPARQLEndpoint(): SPARQLEndpoint
«interface» LabelledGroupOfStatements
excludeStatement(Statement): boolean
excludeStatement(Resource, URI, Value, URI): boolean
getLabel(): URI
getRdfStoreConnection(): RdfStoreConnection
includeStatement(Statement): boolean
includeStatement(Resource, URI, Value, URI): boolean
GraphResultBase
eu.larkc.core.query
GraphResultBase()
close()
getStatements(): CloseableIterator<Statement>
SetOfStatementsImpl
SetOfStatementsImpl(Collection<Statement>)
SetOfStatementsImpl(Iterable<Statement>)
getStatements(): CloseableIterator<Statement>
RdfGraphBase
RdfGraphBase(URI)
getName(): URI
toString(): String
DataSetImpl
DataSetImpl(SPARQLEndpoint, Collection<URI>, Collection<URI>)
addAll(DataSet)
getDefaultGraphs(): Set<RdfGraph>
getNamedGraphs(): Set<RdfGraph>
getSPARQLEndpoint(): SPARQLEndpoint
getStatements(): CloseableIterator<Statement>
HTTPRemoteGraph
HTTPRemoteGraph(URI)
getStatements(): CloseableIterator<Statement>
RdfGraphDataSet
RdfGraphDataSet(URI, SPARQLEndpoint)
getStatements(): CloseableIterator<Statement>
RdfGraphInMemory
RdfGraphInMemory(URI, Collection<Statement>)
RdfGraphInMemory(URI, Iterable<Statement>)
getStatements(): CloseableIterator<Statement>
LabelledGroupOfStatementsImpl
LabelledGroupOfStatementsImpl(URI, RdfStoreConnection, boolean)
LabelledGroupOfStatementsImpl(RdfStoreConnection)
excludeStatement(Statement): boolean
excludeStatement(Resource, URI, Value, URI): boolean
getLabel(): URI
getRdfStoreConnection(): RdfStoreConnection
getStatements(): CloseableIterator<Statement>
includeStatement(Statement): boolean
includeStatement(Resource, URI, Value, URI): boolean
ORDIGraphResult
eu.larkc.core.query
close()
endQueryResult()
handleSolution(BindingSet)
startQueryResult(List<String>)
RDFXMLGraphResult
eu.larkc.core.query
getRDFHandler(): RDFHandler

Figure 25: Classes corresponding to the different RDF data types

Here are some practical examples how you can utilize the RDF exchange data types. The first example shows how to read a local file that contains data serialized in the format of RDFXML and create a structure that will be passed by value.

```
/**
 *  Parse RDF data and create a structure to be passed by value to another plug-in.
*/

// Initialize the Sesame RDF parser and read the file
final Set<Statement> rdf = new HashSet<Statement>();
RDFXMLParser parser = new RDFXMLParser();
parser.setRDFHandler(new RDFHandlerBase() {
      public void handleStatement(Statement st) {
                  if (st != null) {
                        rdf.add(st);
            }
      }
});

try {
      parser.parse(new FileReader("RDFXMLFileToParse.rdf"), "http://default-namespaces);
} catch (Exception e) {
      throw new RuntimeException("Exception while processing the RDF!", e);

}

SetOfStatements rdfData = new SetOfStatementsImpl(rdf);
```

An RDF graph may be passed by reference if you know the URI, which points to RDFXML file or a location that supports HTTP 303 content negation. The remote graph implementation uses a request header "Accept: application/rdf+xml, application/xhtml+xml;q=0.3, text/xml;q=0.2,application/xml;q=0.2, text/html;q=0.3, text/plain;q=0.1".

```
/**

 * Pass by reference remote data exposed as a linked data

 */



String data = "http://linkedlifedata.com/resource/umls/id/C0024117";

RdfGraph graph = DataFactory.INSTANCE.createRemoteRdfGraph(new URIImpl(data));
```

The next example demonstrates how to read RDFXML file and add all its content to a named graph in the Platform repository. The Platform repository connection is compliant with the SPARQL endpoint specification, thus we can pass the graph name as a SPARQL dataset to another component.

```
/**

  * Store data into the data layer from a local RDFXML file and pass it by reference using named graph
  */


// Create a connection to the local data layer

final RdfStoreConnection con DataFactory.INSTANCE.createRdfStoreConnection();

final URI ng = new URIImpl("http://example.named.graph/");


// Initialize Sesame parser

RDFXMLParser parser = new RDFXMLParser();

parser.setRDFHandler(new RDFHandlerBase() {


      public void handleStatement(Statement st) {

            con.addStatement(st.getSubject(), st.getPredicate(), st.getObject(), ng);

      }

});


try {

      parser.parse(new FileReader("RDFXMLFileToParse.rdf"), "http://default.ns");

} catch (Exception e) {

      throw new RuntimeException("Exception while processing the RDF!", e);
}



Set<URI> graphs = new HashSet<URI>();

graphs.add(ng);

DataSet ds = DataFactory.INSTANCE.createRdfStoreConnection().createDataSet(graphs, null);
```

4.8.2. RDF providers

The RDF providers are types that abstract semantic databases (see Figure 26). In the current implementation there are two supported types, `SPARQLEndpoint` and `RdfStoreConnection`, which is a specialized endpoint with the support of data modification and grouping of statements in labelled groups. The SPARQL endpoint could be used to integrate virtually any service capable to feed data into a component. This is one way connection that strictly implements the read only SPARQL communication protocol. The `RdfStoreConnection` type implements the ORDI model. Thus, it supports the selection of arbitrary parts of the RDF graph with the association of individual statements to labelled groups.

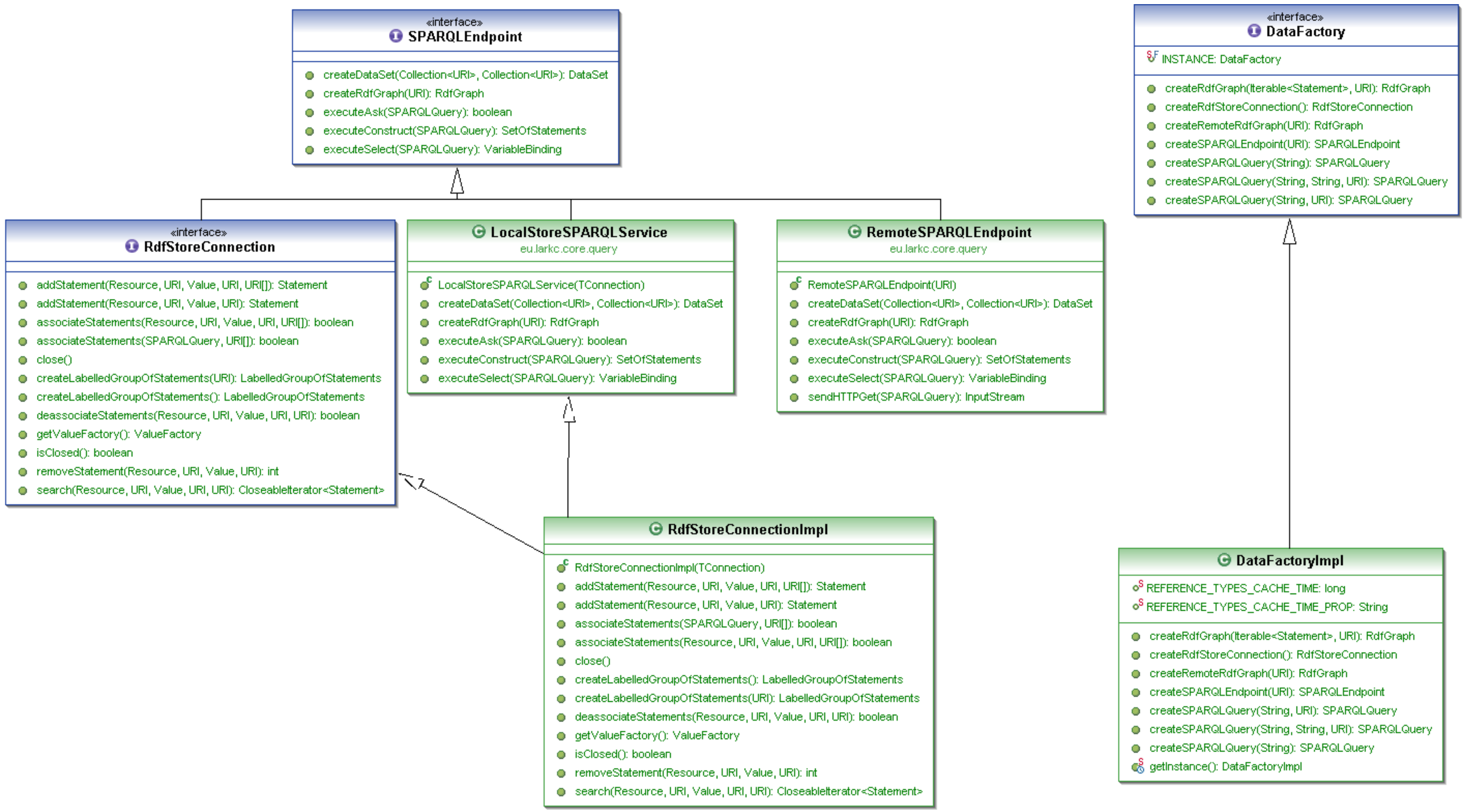

«interface»
SPARQLEndpoint
createDataSet(Collection<URI>, Collection<URI>): DataSet
createRdfGraph(URI): RdfGraph
executeAsk(SPARQLQuery): boolean
executeConstruct(SPARQLQuery): SetOfStatements
executeSelect(SPARQLQuery): VariableBinding
«interface»
DataFactory
INSTANCE: DataFactory
createRdfGraph(Iterable<Statement>, URI): RdfGraph
createRdfStoreConnection(): RdfStoreConnection
createRemoteRdfGraph(URI): RdfGraph
createSPARQLEndpoint(URI): SPARQLEndpoint
createSPARQLQuery(String): SPARQLQuery
createSPARQLQuery(String, String, URI): SPARQLQuery
createSPARQLQuery(String, URI): SPARQLQuery
«interface»
RdfStoreConnection
addStatement(Resource, URI, Value, URI, URI[]): Statement
addStatement(Resource, URI, Value, URI): Statement
associateStatements(Resource, URI, Value, URI, URI[]): boolean
associateStatements(SPARQLQuery, URI[]): boolean
close()
createLabelledGroupOfStatements(URI): LabelledGroupOfStatements
createLabelledGroupOfStatements(): LabelledGroupOfStatements
deassociateStatements(Resource, URI, Value, URI, URI): boolean
getValueFactory(): ValueFactory
isClosed(): boolean
removeStatement(Resource, URI, Value, URI): int
search(Resource, URI, Value, URI, URI): CloseableIterator<Statement>
LocalStoreSPARQLService
eu.larkc.core.query
LocalStoreSPARQLService(TConnection)
createDataSet(Collection<URI>, Collection<URI>): DataSet
createRdfGraph(URI): RdfGraph
executeAsk(SPARQLQuery): boolean
executeConstruct(SPARQLQuery): SetOfStatements
executeSelect(SPARQLQuery): VariableBinding
RemoteSPARQLEndpoint
eu.larkc.core.query
RemoteSPARQLEndpoint(URI)
createDataSet(Collection<URI>, Collection<URI>): DataSet
createRdfGraph(URI): RdfGraph
executeAsk(SPARQLQuery): boolean
executeConstruct(SPARQLQuery): SetOfStatements
executeSelect(SPARQLQuery): VariableBinding
sendHTTPGet(SPARQLQuery): InputStream
RdfStoreConnectionImpl
RdfStoreConnectionImpl(TConnection)
addStatement(Resource, URI, Value, URI, URI[]): Statement
addStatement(Resource, URI, Value, URI): Statement
associateStatements(SPARQLQuery, URI[]): boolean
associateStatements(Resource, URI, Value, URI, URI[]): boolean
close()
createLabelledGroupOfStatements(): LabelledGroupOfStatements
createLabelledGroupOfStatements(URI): LabelledGroupOfStatements
deassociateStatements(Resource, URI, Value, URI, URI): boolean
getValueFactory(): ValueFactory
isClosed(): boolean
removeStatement(Resource, URI, Value, URI): int
search(Resource, URI, Value, URI, URI): CloseableIterator<Statement>
DataFactoryImpl
REFERENCE_TYPES_CACHE_TIME: long
REFERENCE_TYPES_CACHE_TIME_PROP: String
createRdfGraph(Iterable<Statement>, URI): RdfGraph
createRdfStoreConnection(): RdfStoreConnection
createRemoteRdfGraph(URI): RdfGraph
createSPARQLEndpoint(URI): SPARQLEndpoint
createSPARQLQuery(String, URI): SPARQLQuery
createSPARQLQuery(String, String, URI): SPARQLQuery
createSPARQLQuery(String): SPARQLQuery
getInstance(): DataFactoryImpl

Figure 26: RDF Providers classes

In the example below we demonstrate how we can modify and group the statements in the Platform repository. `LabelledGroupOfStatements` may be passed as a RDF data, which will be passed by reference from the Platform repository.

```
/**

 * Manipulate data from the platform repository

 */

RdfStoreConnection con = DataFactory.INSTANCE.createRdfStoreConnection();


URI uri = new URIImpl("http://example.uri");

URI ng = new URIImpl("http://example.named.graph");

URI label = new URIImpl("http://labelled.groupofstatements");


// Adds a statement to the platform repository in a specific named graph

con.addStatement(uri, uri, uri, ng);


// Adds or associate a statement to a labelled group in the repository

con.addStatement(uri, uri, uri, ng, label);


// Retrieve all statements that has a subject http://example.uri

CloseableIterator<Statement> iter = con.search(uri, null, null, null, null);

while (iter.hasNext()) {

      iter.next();
}



// Removes all statements part of http://example.uri graph if existing

con.removeStatement(null, null, null, uri);



// Get a reference to a new group of statements

LabelledGroupOfStatements lgs = con.createLabelledGroupOfStatements();



// Associates if exists all statements from http://example.uri to the labelled group

lgs.includeStatement(uri, null, null, ng);
```

4.8.3. RDF queries and results

The types are used to compose queries and retrieve result from RDF providers. All classes are located in the `eu.larkc.core.query` package and implement asynchronous buffered reading of the streaming results. They should be capable to process virtually unlimited amounts of data.

```


/**

  * Execution of SPARQL query against the platform repository

  */



// Create a connection to the platform data layer
```

```
RdfStoreConnection con = DataFactory.INSTANCE.createRdfStoreConnection();


// Create SPARQL SELECT query

SPARQLQuery query = DataFactory.INSTANCE.createSPARQLQuery("SELECT * WHERE {?s ?p ?o}");


// Execute SPARQL Select query

VariableBinding binding = con.executeSelect(query);


// Iterate all the results

CloseableIterator<Binding> iter = binding.iterator();


// Print all variable names

String[] columns = binding.getVariables().toArray(new String[]{});

for (String col : columns) {

      System.out.println(col);

      System.out.println("\t");

}


// Iteratre all variable bindings
```

```
while (iter.hasNext()) {

      Binding b = iter.next();

      for (Value v : b.getValues()) {

            System.out.println(v.stringValue());

            System.out.println("\t");

            }

      System.out.println("\n");

}
```

5. User support

A number of tools has been setup, as part of the LarKC Development Environment, to support the different kinds of LarKC users.

5.1. How to join LarKC@SourceForge.net

The only prerequisite for joining the LarKC@SourceForge.net[25] is to have a valid SourceForge.net account that can be obtained by registering at http://sourceforge.net/account/registration/.

For getting access to all the LarKC features and services described above, please send your account information to the project administrator, larkc-contact@lists.sourceforge.net.

5.2. End User 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. The user feedback is very relevant for the LarKC development, as an important source for improvement. In order to encourage a community of users to get involved in the LarKC developments, collect user's improvement proposals, solve appearing issues rapidly, promote the discussion within the user community as well as between users and developers, the following services are provided by the LarKC@SourceForge.net:

- **The User Forum:** here 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 the LarKC@SoureForge.net main page[26] or through https://sourceforge.net/projects/larkc/forums/.
- **The User Support Mailing List:** larkc-user-support@lists.sourceforge.net is used in addition to the User Forum with automatic message redirecting from/to the User Forum. The main LarKC developers are subscribed to the Mailing List and provide a prompt feedback to the user support requests. In order to join the mailing list, a subscription at https://lists.sourceforge.net/lists/listinfo/larkc-user-support is required.

5.3. Developer Support

- **The Developer Forum:** here discussions should take place about implementing new LarKC components (including plug-ins, workflows and other possible components) or improving/modifying existing ones. The Developer Forum is accessible through the "Develop/Forums/LarKC-Developers" Tab on the LarKC@SoureForge.net's main page or through https://sourceforge.net/projects/larkc/forums/
- **The Developer Support Mailing List:** larkc-developer-support@lists.sourceforge.net is used in addition to the Developer Forum with automatic message redirecting from/to the Developer Forum. In order to join the mailing list, a subscription at https://lists.sourceforge.net/lists/listinfo/larkc-developer-support is required.

5.4. Contact us

For getting access to the LarKC@SourceFourge.net or in case of problems with using its services, please contact the project administrator at larkc-contact@lists.sourceforge.net.

[25] http://larkc.sourceforge.net
[26] https://sourceforge.net/projects/larkc/develop

6. Licensing

This statement summarizes the decisions taken by the consortium and documented in the LarKC deliverable D9.4 [5]:

The LarKC Platform and its source code will be released under the Apache License, Version 2.0[27].

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 will reiterate the point that 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 can't use to build the Platform. A wider variety of licences (including LGPL but NOT GPL) may be used for plug-ins and other *non-platform* code. Note that this doesn't 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.

7. Open Issues and Future Tasks

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

- Definition, documentation and support of design patterns, providing best practices and guidelines for LarKC users (mainly workflow designers and plug-in developers) to take the maximum advantage of the LarKC Platform features, achieving the needed level of performance for their applications. This task is carried out in close collaboration with WP1.
- Implementation of further Platform support features, such as:
    - Handling and management of splitting and merging of different workflow branches, including the particular input and output data sets. These functionalities allow for building more complex workflows, i.e. workflows will not be necessarily sequentially executed in the future but can be branched and also constructed in loops or conditional execution paths, so as to enable more efficient parallel processing of particular plug-ins.
    - Enhanced event handling features based on publish/subscribe mechanisms in order to facilitate the communication of the Platform with workflows and plug-ins (including error handling mechanisms).
    - Instrumentation and monitoring functionalities embedded in the Platform to enable immediate assessments of running workflows and plug-ins with respect to performance and scalability.
    - A Resource registry containing a set of resource descriptions (i.e., parameter about a resource such as location, authentication method, SLAs, costs, etc.) to identify and make use of possible remote resources.
    - Support features for caching, in order to achieve an improved degree of Platform performance.

Future Platform releases will try to ensure backwards compatibility, whenever possible. However, due to the fact that the Platform is still in a development phase and subject to continuous improvements, we cannot guarantee it. In the case that new releases require any adaptation of developed plug-ins or

[27] http://www.apache.org/licenses/LICENSE-2.0.html

workflows, instructions will be given on how to perform it in the corresponding update of the Platform documentation.

8. References

[1] http://www.larkc.eu
[2] Gallizo et al., D5.3.2 Overall LarKC architecture and design v1, September 2009
[3] http://www.opencyc.org
[4] Kiryakov, A., Ognyanov, D. and Manov, D.: OWLIM - a pragmatic semantic repository for owl. In: Dean, M., Guo, Y., Jun, W., Kaschek, R., Krishnaswamy, S., Pan, Z., Sheng, Q.Z. (eds.) Proceedings of Int. Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2005), LNCS, vol. 3807, pp. 182--192, Springer (2005)
[5] Beffani et al., D9.4 First draft of Exploitation and IPR Plan V1.1, May 2009