LarKC

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

FP7 – 215535

D1.4.2

Evolved Framework for Measuring and Evaluating Heuristic Problem Solving

Gaston Tagni, Zhisheng Huang

Document Identifier:	LarKC/2008/D1.4.2
Class Deliverable:	LarKC EU-IST-2008-215535
Version:	version 1.0.0
Date:	December 30, 2010
State:	final
Distribution:	public

Executive Summary

In this deliverable we refine the framework proposed in our previous Deliverable D1.4.1 [1] by providing a more formal framework for evalauting heuristic problem solving in LarKC. This evolved framework provides an abstract and formal definition of three of the measures proposed in the aforementioned document. More concretely, this document provides a formal definition of the *scalability*, *runtime* and *accuracy* of both plug-ins and workflows. The definitions of these metrics abstract over specific methods and metrics for assessing their value such as precision and recall for defining accuracy.

Document Information

IST Project Number	FP7 – 215535	**Acronym**	LarKC
Full Title	Large Knowledge Collider		
Project URL	`http://www.larkc.eu/`		
Document URL			
EU Project Officer	Stefano Bertolo		

Deliverable	**Number**	1.4.2	**Title**	Evolved Framework for Measuring and Evaluating Heuristic Problem Solving
Work Package	**Number**	1	**Title**	Conceptual Framework

Date of Delivery	**Contractual**	M33	**Actual**	31-December-10
Status	version 1.0.0		final ☒	
Nature	prototype ☐ report ☒ dissemination ☐			
Dissemination Level	public ☒ consortium ☐			

Authors (Partner)	Gaston Tagni (VUA), Zhisheng Huang (VUA)			
Resp. Author	Gaston Tagni		**E-mail**	g.tagni@vu.nl
	Partner	VUA	**Phone**	+31 20 59 87753

Abstract (for dissemination)	In this deliverable we refine the framework proposed in our previous Deliverable D1.4.1 [1] by providing a more formal framework for evalauting heuristic problem solving in LarKC. This evolved framework provides an abstract and formal definition of three of the measures proposed in the aforementioned document. More concretely, this document provides a formal definition of the *scalability*, *runtime* and *accuracy* of both plug-ins and workflows. The definitions of these metrics abstract over specific methods and metrics for assessing their value such as precision and recall for defining accuracy.
Keywords	Heuristic Problem Solving, Evaluation Framework, Measurement, Evaluation, Formative Evaluation

Project Consortium Information

Acronym	Partner	Contact
Semantic Technology Institute Innsbruck `http://www.sti-innsbruck.at`		Prof. Dr. Dieter Fensel Semantic Technology Institute (STI) Innsbruck, Austria E-mail: dieter.fensel@sti-innsbruck.at
AstraZeneca AB `http://www.astrazeneca.com/`		Bosse Andersson AstraZeneca Lund, Sweden E-mail: bo.h.andersson@astrazeneca.com
CEFRIEL SCRL. `http://www.cefriel.it/`		Emanuele Della Valle CEFRIEL SCRL. Milano, Italy E-mail: emanuele.dellavalle@cefriel.it
CYCORP, RAZISKOVANJE IN EKSPERIMENTALNI RAZVOJ D.O.O. `http://cyceurope.com/`		Dr. Michael Witbrock CYCORP, RAZISKOVANJE IN EKSPERIMENTALNI RAZVOJ D.O.O., Ljubljana, Slovenia E-mail: witbrock@cyc.com
Hchstleistungsrechenzentrum, Universitaet Stuttgart `http://www.hlrs.de/`		Georgina Gallizo Hchstleistungsrechenzentrum, Universitaet Stuttgart Stuttgart, Germany E-mail : gallizo@hlrs.de
Max-Planck-Institut fur Bildungsforschung `http://www.mpib-berlin.mpg.de/index_js.en.htm`		Dr. Lael Schooler, Max-Planck-Institut fr Bildungsforschung Berlin, Germany E-mail: schooler@mpib-berlin.mpg.de
Ontotext Lab, Sirma Group Corp. `http://www.ontotext.com/`		Atanas Kiryakov, Ontotext Lab, Sirma Group Corp. Sofia, Bulgaria E-mail: atanas.kiryakov@sirma.bg
SALTLUX INC. `http://www.saltlux.com/EN/main.asp`		Kono Kim SALTLUX INC Seoul, Korea E-mail: kono@saltlux.com
SIEMENS AKTIENGESELLSCHAFT `http://www.siemens.de/`		Dr. Volker Tresp SIEMENS AKTIENGESELLSCHAFT Muenchen, Germany E-mail: volker.tresp@siemens.com
THE UNIVERSITY OF SHEFFIELD `http://www.shef.ac.uk/`		Prof. Dr. Hamish Cunningham THE UNIVERSITY OF SHEFFIELD Sheffield, UK E-mail: h.cunningham@dcs.shef.ac.uk
VRIJE UNIVERSITEIT AMSTERDAM `http://www.vu.nl/`		Prof. Dr. Frank van Harmelen VRIJE UNIVERSITEIT AMSTERDAM Amsterdam, Netherlands E-mail: Frank.van.Harmelen@cs.vu.nl
THE INTERNATIONAL WIC INSTITUTE, BEIJING UNIVERSITY OF TECHNOLOGY `http://www.iwici.org/`		Prof. Dr. Ning Zhong THE INTERNATIONAL WIC INSTITUTE Mabeshi, Japan E-mail: zhong@maebashi-it.ac.jp
INTERNATIONAL AGENCY FOR RESEARCH ON CANCER `http://www.iarc.fr/`		Dr. Paul Brennan INTERNATIONAL AGENCY FOR RESEARCH ON CANCER Lyon, France E-mail: brennan@iarc.fr

Table of Contents

1 Introduction

The goal of this deliverable is to refine the framework described in our previous Deliverable D1.4.1 [1] by providing a more formal framework for evalauting heuristic problem solving in LarKC that provides an abstract and formal definition of the different metrics outlined in the aforementioned document.

By providing a formal characterization of the metrics defined in the previous deliverable we are able to define precisely what aspects of a system (eg. a plug-in or workflow) we intent to measure. In general, given a property of a system or component we are interested in measuring there may exist several methods or techniques to asses its value. For example, one method to assess the performance of a plug-in is to measure its runtime, ie. the time it requires to complete a given task. Another way of assessing performance is to consider the number of operations the plug-in needs to perform in order to carry out a given task. Both methods allows us to determine a plug-in's performance measure albeit in terms of two different metrics, namely time and number of operations. A desired characteristic of a framework for measuring heuristic problem solving in LarKC is the ability to encompass and capture the plethora of methods and techniques for measuring those properties of a plug-in or workflow that allow us to evaluate heuristic problem solving in LarKC. Therefore, in this deliverable we intent to define some of the relevant metrics for measuring heuristic problem solving in such a way that we are able to abstract from specific metrics and methods and provide a general characterization of the different metrics. By providing metrics that abstract over specific methods and measures allows us to focus our attention on the desired properties of a given metric so that independently of what method is used for assessing its value the results capture the target property we are interested in measuring.

Some of the properties we intent to measure are applicable to different components at different levels of abstraction. For example, processing speed is a property that can be measured both at the plug-in and the workflow level. In this case, the processing speed of a workflow is determined based on the processing speed of its constituent plug-ins and the number of times its plug-ins are executed. In this case, processing speed can be measured per plug-in or per workflow and can be measured for every type of plug-in.

1.1 Evaluating Heuristic Problem Solving in the Context of LarKC

Although the work being done in WP11 ("Instrumentation and Evaluation") is closely related to the work being done in WP1 in the context of task 1.4 ("Evaluation"), which includes the family of deliverables D1.4.x, there are some important differences between the two of them that we will highlight here.

The first difference is in regards to the main goal of both task forces. The work in D1.4.1 and D1.4.2 is about defining a framework for measuring heuristic problem solving in LarKC and hence the object or process we are concerned with measuring and evaluating is heuristic problem solving rather than a single component of subsystem of the platform. For a detailed explanation of the term heuristic problem solving the reader is referred to Deliverable D1.4.1 [1]. As a consequence, we are not generally focused on evaluating the several plug-ins and components that comprise the LarKC platform but with evaluating the performance of different heuristic-based approaches to problem solving in LarKC. Given a task or problem we want to solve in LarKC we intent to evaluate and compare how different heuristic-based strategies perform in solving the given problem, which in LarKC terms is represented by a query posed by the user. In LarKC, such a task or problem is solved by first building a workflow comprised by instances of different plug-in types and then by executing such a workflow. Heuristic problem solving in LarKC can occur at different levels such as at the plug-in level (eg. through heuristic-based plug-ins) or at the workflow level. Therefore, evaluating heuristic problem solving calls for metrics and methods for measuring heuristic problem solving at these different levels.

In contrast to this, WP11 is concerned with instrumentation and evaluation of the entire platform and all its components and thus it may include a much larger set of metrics, eg. metrics for evaluating the performance of the data layer component. In addition to this, the work in WP11 is concerned with monitoring the execution (performance) of the platform and thus many of the metrics may need to be evaluated at runtime. Moreover, these measures can be used to make decisions about how to best control the execution of the platform (eg. adding more compute nodes on demand to deal with large amount of data of complex plug-in executions). Monitoring the performance of the platform is not a task we are concerned with in D1.4.2. In task 1.4 we are interested in evaluating how different heuristic-based approaches solve problems in LarKC.

Although there are some fundamental differences between the tasks being conducted in WP1 and WP11 in regards to the evaluation of LarKC there is a relation between both as well. In particular, the work on WP11 (see Deliverable D11.1.2 [2]) serves as the foundation on which the evaluation framework for heuristic problem solving should be built upon. Any measure to asses how different heuristic methods perform in solving a given problem should be computed based on the use of concrete metrics supported by the platform and defined in WP11 through the ontology of metrics and on the underlying monitoring component. For example, if we plan to measure performance of a plug-in by measuring its execution time then we must be sure the platform provides the means (metrics and methods) to do that.

1.2 Organization of this Document

This document is organized as follows. Chapter 2 introduced a series of formalizations that form the basis of our formal framework. Chapter 3 provides a formal and abstract characterization of three of the metrics proposed in our previous deliverable, namely runtime, scalability and accuracy. Then in Chapter **??** we report on the results of a series of initial experiments aimed at measuring different plug-ins according to the proposed metrics. Chapter 4 concludes the document.

2 Plug-ins and Workflows: Formal Definition

We consider a plug-in to be an algorithm that implements a function that maps a set of inputs (called an input tuple) into a set of outputs (called an output tuple). More formally, a plug-in can be defined as follows:

Definition 1 (Plug-in) *A plug-in p is a (partial) function which maps an input tuple $\langle i_1, \ldots, i_n \rangle$ into an output tuple $\langle o_1, \ldots, o_m \rangle$. In that case, we can write that $p(\langle i_1, \ldots, i_n \rangle) = \langle o_1, \ldots, o_m \rangle$.*

For every plug-in p we say the plug-in's output is undefined for every invalid input t of the plug-in. We denote this by an expression of the form $p(t) = undefined$.

Let $Plugin$ be the set of all plugins and $Workflow$ be the set of all workflows. In the following, we will define several types of workflows based on the control-flow structures used to combine the different plug-in types used to construct the workflow. Naturally, every plug-in can be thought of as a workflow with a single plug-in and no control-flow structure. Formally,

$$p \in Plugin \Rightarrow p \in Workflow \tag{2.1}$$

2.1 Sequential Workflows

The most basic control-flow operator for building workflows in LarKC is the *sequential operator*. This operator allows us to construct a workflow by combining multiple plug-ins sequentially, ie. where the execution of one plug-ins immediately follows the execution of another plug-in. Formally, let p_1 and p_2 be two plug-ins ($p_1 \in Plugin$ and $p_2 \in Plugin$) then a new workflow $f_1 = p_1; p_2$ can be constructed by concatenating p_1 and p_2 using the sequential operator ';'. We call f_1 a sequential workflow. More generally and from definition 2.1 we have the following:

$$f_1 \in Workflow, f_2 \in Workflow \Rightarrow f_3 = f_1; f_2 \in Workflow$$

For an input tuple t_1, the output t_3 of the workflow f which consists of a sequence of two plugins p_1 and p_2, i.e., $f = p_1; p_2$, can be defined as the outpout of the second plugin p_2 when it takes the output of p_1 as its input. Namely,

$$f \equiv p1; p2, p_1(t_1) = t_2, p_2(t_2) = t_3 \Rightarrow \langle t_1, t_3 \rangle \in f.$$

A requirement in this case is that the output of plug-in p_1 must be a legal input of plug-in p_2, otherwise we say that the workow f is undefined for input t_1.

2.2 Parallel Workflows

Furthermore, we extend a workflow so that it can be constructed with the parallel operator $||$. Thus, we have

$$f_1 \in Workflow, f_2 \in Workflow \Rightarrow f_1 || f_2 \in Workflow.$$

We assume that the two parallel processes among the two plugins do not interfere with each other. Thus, we can define the output of a workflow $f = p_1 || p_2$

as the union of the output of the two plug-ins for the same input t_1. Namely, we have the following property:

$$f \equiv p_1 || p_2, p_1(t_1) = t_2, p_2(t_1) = t_3 \Rightarrow \langle t_1, t_2 \rangle \in f$$

Thus, we define a workflow as a relation between input tuples and output tuples, rather than as a function. More generally, we define the relations of workflows as follows:

$$\langle t_1, t_2 \rangle \in f_1, \langle t_2, t_3 \rangle f_2 \Rightarrow \langle t_1, t_3 \rangle \in f_1; f_2.$$

$$\langle t_1, t_2 \rangle \in f_1, \langle t_1, t_3 \rangle f_2 \Rightarrow \langle t_1, t_2 \rangle, \langle t_1, t_3 \rangle \in f_1 || f_2.$$

2.3 Conditional Workflows

A conditional workflow has the form $if\ \phi\ then\ f_1 else\ f_2$ where ϕ is a formule (more exactly a function) which returns a boolean value. Thus, the semantics of the conditional operator in a workflow is defined as follows:

$$f \equiv if\ \phi\ then\ f_1 else\ f_2, \langle t_1, t_2 \rangle \in f_1, \langle t_1, t_3 \rangle \in f_2, \phi \text{ holds} \Rightarrow \langle t_1, t_2 \rangle \in f$$

$$f \equiv if\ \phi\ then\ f_1 else\ f_2, \langle t_1, t_2 \rangle \in f_1, \langle t_1, t_3 \rangle \in f_2, \neg\phi \text{ holds} \Rightarrow \langle t_1, t_3 \rangle \in f$$

We define the operator *skip* as a trivial operator which does nothing. Namely, the input and the output relation of the skip operator is the reflexive relation, i.e, for any tuple t, $\langle t, t \rangle \in skip$.

Now, we can define a simple conditional (i.e one without the else part) in terms of a conditional as follows:

$$if\ \phi\ then\ f =_{df} if\ \phi\ then\ p\ else\ skip.$$

2.4 Iterative Workflows

We construct a loop operator in a workflow as follows:

$$f \in Workflow, \phi \text{ is a formula} \Rightarrow (repeat\ f\ until\ \phi) \in Workflow.$$

The semantics of the loop operator is defined as:

$$\langle t_1, t_2 \rangle \in f, \ldots, \langle t_{i-1}, t_i \rangle \in f, \phi \text{holds for the minimal } i \Rightarrow \langle t_1, t_i \rangle \in (repeat\ f\ until\ \phi).$$

We define the while operator for a workflow in terms of a conditional and a loop as follows:

$$while(\phi) f =_{df} if \phi\ then (repeat\ f\ until\ (\neg\phi)).$$

3 Formal Characterization of Metrics

3.1 Preliminary Definitions

In the following we introduce a series of preliminary definitions that will be used throughout the chapter to formally define runtime, scalability and accuracy of plug-ins and workflows.

3.1.1 Input and Output Spaces

In this deliverable we reuse the notions of input and output spaces defined in Deliverable D1.4.1 [1] and generalize them so they can be applied not only to reasoners but also to other types of plug-ins in LarkC. More concretely, we generalize the term *Input Space* to denote the set of all possible tasks a given plug-in or workflow is responsible for carrying out. In other words, the term input space refers to all the possible tasks a given plug-in is given and for which the plug-in must compute answers. For example, given an Identify plug-in the input space contains all the possible Identify tasks that can be given to the plug-in. In this particular case an Identify task consist of a SPARQL query for which the plug-in must find a collection of Information Sets relevant for answering the given query. The same reasoning can be applied to define the input space of the other plug-ins. Formally speaking, we define the input space as a probability space (Ω, P), where Ω is some set (of inputs) and P is a probability measure on Ω. The probability $P(\omega)$ encodes how often a specific input, which depends on the specific type of plug-in, ω occurs in practice, resp. how relevant it is for practical purposes. The importance of formalizing the input space of a plug-in as a probability space is that it will allow us to measure the correctness of the results of a given plug-in relative to the frequency with which plug-in tasks (ie. identification, selection, reasoning, transformation and decision tasks) occur in LarkC. Incorrect answers to an input that occurs less often is more tolerable than if the input occurs very often.

The *Output Space* of a plug-in in LarkC is defined as in [1] and comprises all possible answers to any of the tasks from the input space. In our framework we refer to the output space as the set $OUTPUT$. As in the previous deliverable we assume that the output space of a plug-in or workflow contains a distinguished element $\perp$ which denotes *no output*.

3.1.2 Error Function

In D1.4.1 [1] an *error function* was defined in order to capture the degree to which an (approximate) answer to a reasoning task differs from the expected (correct) answer. Using this error function together with the reasoning time of a reasoner it was possible to formalize and quantify the usefulness of an approximate reasoning algorithm. In this deliverable we borrow the definition of the error function introduced in D1.4.1 and propose to use it to measure the extent to which an answer (ie. the result of a given plug-in or workflow) deviates from the expected output. Formally, the error function is defined as $e : Z \times Z \Rightarrow \Re$ where Z denotes any possible answer/output that can be returned by a component, ie. Z is an object in a plug-in's output space. The following requirements are imposed on this function:

- We assume that $e(X, X) = 0$. See D1.4.1 [1] for more details on the definition of this function in the context of evaluating anytime, approximate reasoning.
- Assuming Y is the expected result set and that X_1 and X_2 are two result sets such that X_1 is closer to Y (the expected answer) than X_2 then $e(X_1, Y) < e(X_2, Y)$.
- $e(X, Y)$ is the number of items by which results X and Y differ. This could be computed based on recall, precision or other means.
- In order to be a metric the error function e must have an integral.

Notice that this definition of the error function introduced above requires knowledge about what is the expected or correct output returned by a plug-in or workflow. In order to capture the notion of a correct or expected output we use the *Correct Output Function* $f_0 : \Omega \rightarrow OUTPUT$ defined in Deliverable D1.4.1. The value of such function is determined by an external specification or formal semantics of the problem to be solved by a given plug-in or workflow.

3.2 Characterizing and Measuring Runtime

In this section we will give a formal characterization of the runtime of a workflow based on the control-flow structures used for building the workflow and on the runtime of its constituent plug-ins. By $f_{runtime} : Workflow \rightarrow [0, +\infty)$ we denote a function that captures the runtime of a plug-in or, more generally, a workflow $f \in Workflow$. We assume the runtime of a plug-in can be measured by some external method and expressed in some standard unit of time, typically milliseconds. In the following we characterize formally the runtime of a workflow based on its structure.

3.2.1 Runtime of a Sequential Workflow

Let $f_x \in Workflow$ be a sequential workflow comprised of n workflows, ie. $f_x \equiv f_1; f_2; \ldots; f_n$. Then the runtime of workflow f_x can be defined as follows:

$$f_{runtime}(f_x) = \sum_{i=1}^{n} f_{runtime}(f_i) \tag{3.1}$$

3.2.2 Runtime of a Parallel Workflow

Let $f_x \in Workflow$ be a parallel workflow comprised of n workflows to be executed in parallel, ie. $f_x \equiv f_1 || \ldots || f_n$. Then the runtime of workflow f_x can be defined as follows (for $i : 1 \ldots n$):

$$f_{runtime}(f_x) = MAX\{f_{runtime}(f_i)\} \tag{3.2}$$

3.2.3 Runtime of a Conditional Workflow

Let $f_x \in Workflow$ be a conditional workflow of the form $f_x \equiv if\ \phi\ then\ f_1 else\ f_2$. Then the runtime of workflow f_x can be defined as follows:

$$f_{runtime}(f_x) = \begin{cases} f_{runtime}(f_1) & \text{if } \phi \text{ is true} \\ \\ f_{runtime}(f_2) & \text{if } \phi \text{ is false} \end{cases}$$

3.2.4 Runtime of an Iterative Workflow

Let $f_x \equiv (repeat\ f\ until\ \phi)$ be an iterative workflow where f is a workflow that is to be executed multiple times until the condition ϕ holds and let n be the number of times workflow f is executed. Then, the runtime of workflow f_x can be defined as follows:

$$f_{runtime}(f_x) = \sum_{i=1}^{n} f_{runtime}(f) \tag{3.3}$$

3.3 Characterizing and Measuring Accuracy

Accuracy is a measure of the correctness or quality of the results returned by a system or component and, in particular in LarKC, by a workflow or plug-in. In this section we intent to characterize and define a general measure of the accuracy of a plug-in or workflow that abstracts over specific measures of accuracy typically used in practice, eg. precision and recall in the Information Retrieval field. To simplify things the terms correctness of results and accuracy of a component are used interchangeably in this deliverable.

The accuracy of a plug-in or workflow is one of the quantitative dimensions that need to be measured in order to evaluate heuristic problem solving in LarKC [1]. Accuracy can be used to evaluate and compare two alternative heuristic-based problem solving methods in LarkC in order to determine which of them provides better solutions with an efficient use of the computational resources available. Moreover, accuracy can be used to compare an heuristic-based method and a non-heuristic method. As it was mentioned on Deliverable D1.4.1 heuristic problem solving methods in LarKC can be implemented by individual plug-ins, eg. in the form of a Reason plug-in that uses stopping rules for determining when to stop the current reasoning process, or by a combination of them, ie. by a workflow that combines for example the usage of stopping rules within a Decide plug-in and the usage of ontology subsetting methods based on cognitively-inspired heuristics for selecting a subset of the relevant datasets.

In general, different approaches (or metrics) for evaluating the accuracy of a component may be used depending on the type of plug-in and the underlying monitoring and evaluation services provided by the LarKC platform. For instance, the correctness of Select or Identify plug-ins can be measured using the notions of precision and recall, both metrics borrowed from the field of Information Retrieval. Precision and recall are two standard measures of correctness defined in terms of the number of retrieved documents and relevant documents. Precision, on the one hand, is a metric borrowed from the Information Retrieval field that measures the correctness of results as the fraction of correctly identified items, ie. the number of correct or relevant items retrieved. The higher the precision the better the system is at identifying or retrieving correct items (ie. computing correct results to a query) and consequently the higher the precision of the component (in our case a plug-in or workflow). Recall, on the other hand, measures the correctness of a component as the fraction of relevant items retrieved or returned by the component. Our goal in this deliverable is to propose a more general measure of correctness that abstracts over specific metrics such as the aforementioned ones. In the case of Select plug-in the documents or items to be retrieved and whose relevance is to be evaluated are RDF triples while in the case of Identify plug-ins documents refer to information sets (ie. RDF graphs). In the same way, the traditional approach to measuring the correctness of a reasoning service is to use the notions of soundness and completeness.

3.3.1 Accuracy of Plug-ins and Workflows

In order to define an abstract metric of accuracy for assessing the correctness of a plug-in or workflow we assume that there exist a preference relation among the elements in the output space of a plug-in or workflow. More concretely, we assume the existence of a partial order set $(\mathcal{A}, \prec)$, where $\mathcal{A}$ is the output space of a plug-in or workflow and $\prec$ is the preference relation that allows us to determine which of two possible results is preferred. Notice that we do not impose a total ordering among results as it may be the case that two results are equally (not) preferred.

Using the definition of the error function and correct output function introduced above we can determine the degree of correctness of an answer $x \in OUTPUT$ returned by a given plug-in or workflow with respect to a specific input ω by determining the deviation or error $e(x, f_0(\omega))$; the smaller the value the more correct the answer is. Notice that by defining a measure of correctness in terms of an error function we provide a general, ie. for all plug-in types and workflows, way of assessing the accuracy of a plug-in or workflow that abstracts over specific methods to compute the value of the error function. For example, the error function could be computed by counting the number of correct triples returned by a plug-in.

3.4 Characterizing and Measuring Scalability

Scalability is a property of a system or component that characterize the ability of such system or component to continue to perform at reasonable levels of performance when certain computational parameters increase. Typically, scalability is measured in terms of some criteria such as runtime, memory usage, etc. For instance, we say a component or system scales with respect to the size or complexity of its input if its runtime is below a given threshold as the size or complexity of the input increases. In this deliverable in particular we are interested in measuring the scalability of plug-ins and workflows.

By $f_{scale} : Workflow \rightarrow [0, +\infty)$ we denote a function that captures the extent (level) to which a workflow $f \in Workflow$ is scalable; where $Workflow$ denotes the set of possible workflows. A value in the range $[0, +\infty)$, called the *scalability threshold* is used to define a threshold that determines whether a workflow is scalable or not. A workflow whose f_{scale} value is below the given threshold is considered to be scalable whereas a workflow whose f_{scale} value is above that threshold is considered as a not scalable workflow. The scalability threshold may vary for different workflows. We denote the scalability threshold of a plug-in p or more generally, the scalability threshold of a workflow f with an expression of the form $scalability\_threshold(f)$. We assume the scalability of a plug-in is known and can be expressed as a value in the range $[0, +\infty)$.

Using this scalability measure of a plug-in and the control flow structures of LarKC workflows we can now define the scalability of a workflow. Formally, given a workflow f we define the scalability of f using the function $f^{w}_{scale} : Workflow \rightarrow [0, +\infty)$, where $Workflow$ denotes the set of all possible workflows.

The *scalability threshold* of a given plug-in could be determined empirically, before deployment, by running the plug-in (or workflow in the general case) multiple times and assessing its performance with different settings for the relevant computational resources and input size.

3.4.1 Scalability of a Sequential Workflow

Let $f_x \in Workflow$ be a sequential workflow comprised of n workflows, ie. $f_x \equiv f_1; f_2; \ldots ; f_n$. Assuming a threshold for the scalability of the workflow f_x has been set we then say that workflow f_x is scalable if and only if:

$$(\sum_{i=1}^{n} f_{scale}(f_i)) \leq scalability\_threshold(f_x) \qquad (3.4)$$

3.4.2 Scalability of a Parallel Workflow

Let $f_x \in Workflow$ be a parallel workflow comprised of n workflows that are executed in parallel, ie. $f_x \equiv f_1||f_2|| \ldots ||f_n$. Assuming a threshold for the scalability of the workflow f_x has been set we then say that workflow f_x is scalable if and only if:

$$MAX\{f_{scale}(f_1), ..., f_{scale}(f_n)\} \leq scalability\_threshold(f_x) \qquad (3.5)$$

3.4.3 Scalability of a Conditional Workflow

Let $f_x \in Workflow$ be a conditional workflow of the form $f_x \equiv if\ \phi\ then\ f_1 else\ f_2$. Assuming a threshold for the scalability of the workflow f_x has been set we then say that workflow f_x is scalable if and only if:

$$\begin{cases} f_{scale}(f_1) \leq scalability\_threshold(f_x) & \text{if } \phi \text{ is true} \\ f_{scale}(f_2) \leq scalability\_threshold(f_x) & \text{if } \phi \text{ is false} \end{cases}$$

3.4.4 Scalability of an Iterative Workflow

Let $f_x \equiv (repeat\ f\ until\ \phi)$ be an iterative workflow where f is a workflow that is to be executed multiple times until the condition ϕ holds. Assuming a threshold for the scalability of workflow f_x has been set we then say that workflow f_x is scalable if and only if the scalability of f is below the threshold after executing f multiple times.

4 Conclusion and Future Work

This document provided an evolved version of the framework for evaluating and measuring heuristic problem solving in LarKC. The main contribution of the document is a formal characterization of three metrics used for measuring heuristic problem solving, namely runtime, accuracy and scalability of plug-ins and workflows. In addition to this the document provides a formal characterization of plug-ins and workflows which form the basis of the formal framework.

The runtime of a workflow in LarKC is formally defined in terms of the runtime of its constituent plug-ins and the four basic control-flow structures supported by LarKC to build workflows, namely sequential, iterative, conditional and parallel control-flow primitives. Runtime is defined abstractly without any reference to specific methods for measuring the runtime of individual plug-ins.

Accuracy is formally defined based on the use of an error function that captures and quantifies the degree of deviation or error of actual results from the expected (correct) ones. As with runtime this metric is defined in abstract terms without making any references to the specific methods used for computing the error between result sets.

The third metric formally defined in this document is scalability. Scalability is defined based on a *scalability threshold*, which defines a reasonable limit for judging the scalability of a plug-in or workflow and, based on the four basic control-flow structures to determine the scalability of workflows based on the execution patterns implied by such structures.

4.1 Future Work

Being this document the second in a series of three deliverables aimed at defining a framework for evaluating and measuring heuristic problem solving in LarKC there are still some aspects that need to be addressed by the final deliverable. One such aspect is in regards to the formalization of other metrics first proposed in Deliverable D1.4.1, namely:

- *Robustness* is the ability of a process to yield meaningful results despite potential breakdowns of components and uncertain, incomplete or noisy data.
- *Quality* is an overall measures of the "goodness" of a thing, process or project, ie. its degree of success and excellence. For practical purposes, quality can be dened in terms of the other measures, ie., as having high values on the dimensions of speed, accuracy, robustness and scalability, as well as having a large community of satised adopters and users.

Another aspect we plan to work on in the next deliverable is the relation between the work of these deliverables and the work being carried out in work package 11 (WP11), which is concerned with the instrumentation and monitoring of the LarKC platform. More specifically we want to make sure that the platform provides the necessary means (atomic and compound metrics) for implementing evaluation methods that allow us to compute the metrics required for measuring heuristic problem solving in LarKC. In addition to this we would like to define a reference binding between the formal metrics defined in this document and the atomic and compound metrics defined in Deliverable D11.1.2 [2].

Finally, we would like to run a series of experiments, and report the corresponding experimental results, aimed at evaluating different plug-ins using the the metrics defined in this deliverable and the next one.

References

[1] Zhisheng Huang, Annette ten Teije, Frank van Harmelen, Gaston Tagni, Hansjorg Neth, Lael Schooler, Sebastian Rudolph, Pascal Hitzler, Tuvshintur Tserendorj, Yi Huang, Danica Damljanovic, and Angus Roberts. D1.4.1 - Initial Framework for Measuring and Evaluating Heuristic Problem Solving, September 2009. Available at http://www.larkc.eu/deliverables/.

[2] Ioan Toma, Raluca Brehar, Silviu Bota, Mihai Negru, Andrei Vatavu, and Mihai Chezan. D11.1.2 - LarKC Metrics Ontologies, November 2010. Available at http://www.larkc.eu/deliverables/.