



Project Number 732223

D5.5 Workflow Development Tools - Final Version

**Version 2.0
30 June 2019
Final**

Public Distribution

University of York

Project Partners: Athens University of Economics & Business, Bitergia, Castalia Solutions, Centrum Wiskunde & Informatica, Eclipse Foundation Europe, Edge Hill University, FrontEndART, OW2, SOFTEAM, The Open Group, University of L'Aquila, University of York, Unparallel Innovation

Every effort has been made to ensure that all statements and information contained herein are accurate, however the CROSSMINER Project Partners accept no liability for any error or omission in the same.

© 2019 Copyright in this document remains vested in the CROSSMINER Project Partners.

PROJECT PARTNER CONTACT INFORMATION

Athens University of Economics & Business Diomidis Spinellis Patision 76 104-34 Athens Greece Tel: +30 210 820 3621 E-mail: dds@aub.gr	Bitergia José Manrique Lopez de la Fuente Calle Navarra 5, 4D 28921 Alcorcón Madrid Spain Tel: +34 6 999 279 58 E-mail: jsmanrique@bitergia.com
Castalia Solutions Boris Baldassari 10 Rue de Penthièvre 75008 Paris France Tel: +33 6 48 03 82 89 E-mail: boris.baldassari@castalia.solutions	Centrum Wiskunde & Informatica Jurgen J. Vinju Science Park 123 1098 XG Amsterdam Netherlands Tel: +31 20 592 4102 E-mail: jurgen.vinju@cw.nl
Eclipse Foundation Europe Philippe Krief Annastrasse 46 64673 Zwingenberg Germany Tel: +33 62 101 0681 E-mail: philippe.krief@eclipse.org	Edge Hill University Yannis Korkontzelos St Helens Road Ormskirk L39 4QP United Kingdom Tel: +44 1695 654393 E-mail: yannis.korkontzelos@edgehill.ac.uk
FrontEndART Rudolf Ferenc Zászló u. 3 I./5 H-6722 Szeged Hungary Tel: +36 62 319 372 E-mail: ferenc@frontendart.com	OW2 Consortium Cedric Thomas 114 Boulevard Haussmann 75008 Paris France Tel: +33 6 45 81 62 02 E-mail: cedric.thomas@ow2.org
SOFTEAM Alessandra Bagnato 21 Avenue Victor Hugo 75016 Paris France Tel: +33 1 30 12 16 60 E-mail: alessandra.bagnato@softeam.fr	The Open Group Scott Hansen Rond Point Schuman 6, 5 th Floor 1040 Brussels Belgium Tel: +32 2 675 1136 E-mail: s.hansen@opengroup.org
University of L'Aquila Davide Di Ruscio Piazza Vincenzo Rivera 1 67100 L'Aquila Italy Tel: +39 0862 433735 E-mail: davide.diruscio@univaq.it	University of York Dimitris Kolovos Deramore Lane York YO10 5GH United Kingdom Tel: +44 1904 325167 E-mail: dimitris.kolovos@york.ac.uk
Unparallel Innovation Bruno Almeida Rua das Lendas Algarvias, Lote 123 8500-794 Portimão Portugal Tel: +351 282 485052 E-mail: bruno.almeida@unparallel.pt	

DOCUMENT CONTROL

Version	Status	Date
1.1	Full draft version	14 June 2019
1.2	Further updates and final for internal review	24 June 2019
2.0	Final Version	30 June 2019

TABLE OF CONTENTS

1. Introduction.....	1
1.1 Motivation	1
1.2 Outline.....	2
2. Crossflow Workflow Language.....	3
2.1 Running Example	4
2.2 Code Generation	4
3. Graphical Workflow Modeling.....	7
3.1 Background	7
3.2 Crossflow Graphical Model Editor (EuGENia-based)	7
3.2.1 Editor Updates	8
4. Textual Workflow Modeling.....	11
4.1 Background	11
4.2 Crossflow Textual Model Editor (Xtext-based)	12
4.2.1 Editor Updates	15
4.3 Crossflow XML-based Model Editing (Flexmi).....	16
5. Workflow Deployment, Execution, and Monitoring.....	17
5.1 Crossflow Model Deployment and Execution (Eclipse)	17
5.2 Crossflow Model Deployment, Execution, Monitoring, and Management (Web-based).....	18
5.3 Crossflow Worker Execution (CLI)	26
6. Installation Guides	28
6.1 Crossflow Graphical Editor Installation and Launch	28
6.2 Crossflow Textual Editor Installation and Launch.....	28
6.3 Crossflow Web Application Installation and Launch	29
7. Conclusion and Future Work	31
Table on final status of use-case partner requirements for WP5	32
Table on final status of technology requirements for WP5.....	33
REFERENCES	34
Appendix A: Crossflow Graphical Modeling Language Definition (crossflow.emf)	35
Appendix B: Crossflow Textual Workflow Modeling Language Definition (Crossflow.xtext).....	37

EXECUTIVE SUMMARY

Work package 5 (WP5) aims at creating a framework for supporting the development of high-performance declarative open-source project analysis workflows. This is underpinned by Crossflow—a domain-specific language capable of expressing open-source project analysis workflows in a high level of abstraction. Creation and execution of Crossflow workflows is aided by an Eclipse-based graphical editor and a browser-based interface providing feedback during and after the execution of the workflow, respectively. Workflows are executed through a scalable execution engine, capable of both parallel and distributed execution that is running against code generated using appropriate model transformation languages to executable code.

This document expands upon D5.3 by providing details into the current implementation of Crossflow by means of an Eclipse-based graphical model editor as well as by presenting new components that include a browser-based workflow deployment, monitoring, and execution-management interface. First, it presents an exemplary use case showcasing the creation of a Crossflow workflow model, the generation of base classes from such a model, as well as the specification of fine-grained behaviour through implementations extending generated code. Secondly, it presents packaging and deployment of Crossflow workflows as well as their monitoring and execution-management during runtime.

1. INTRODUCTION

This document will present the development tools of the workflow component of CROSSMINER. Workflows allow for the definition of bespoke analysis algorithms in a high-level domain-specific language and require the creation of appropriate tools to create, manipulate, generate and monitor them and their execution.

Figure 1 depicts the integration of Crossflow in the Crossminer architecture. On one hand, workflow specifications, i.e. Crossflow models created by the employing workflow development tools described in Sections 3-4 of this document, can be executed by metric providers by instantiating the Crossflow Java API. On the other hand, Crossflow workflow tasks can retrieve data from the Crossminer knowledge base REST API (cf. Section 9.4 in D6.5). In more detail, data from the knowledge base can be retrieved within the context of a Crossflow workflow by submission of HTTP GET/POST requests that are coded into workflow tasks and triggered during

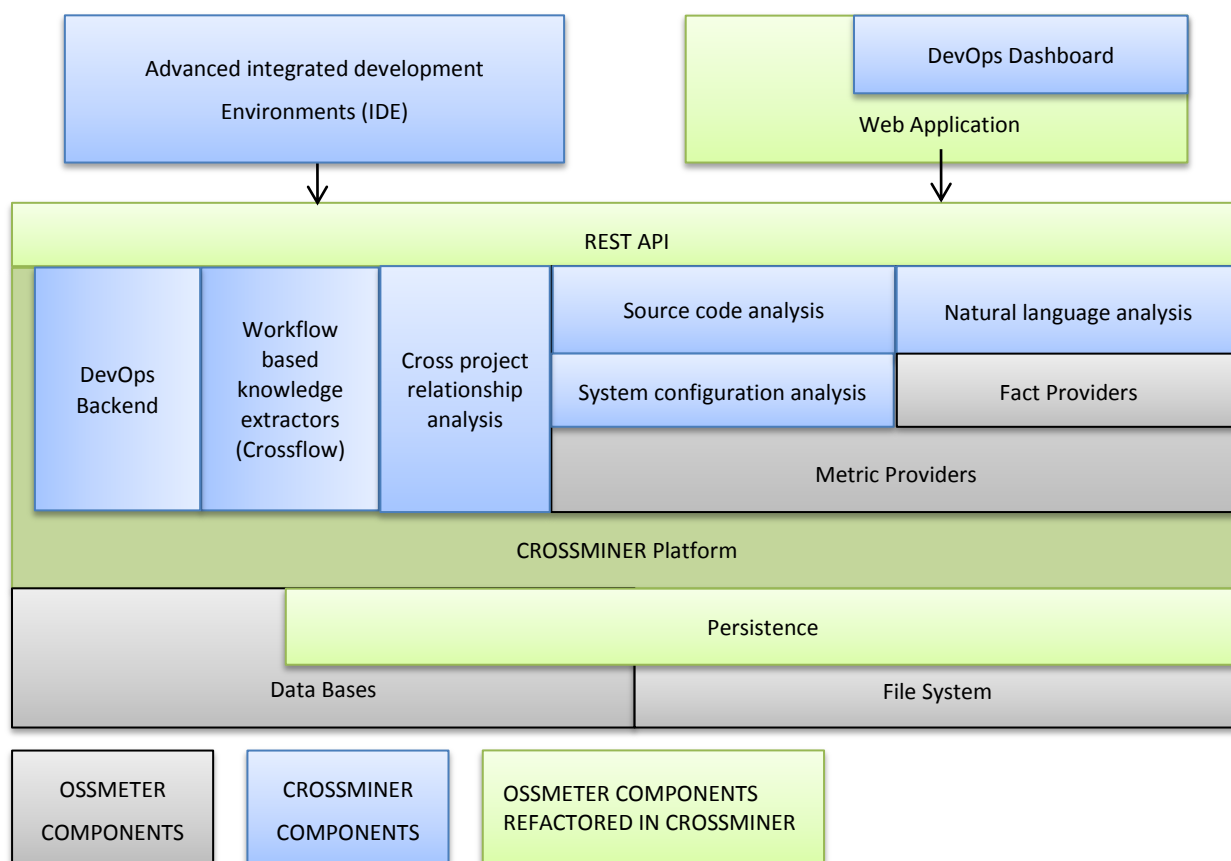


Figure 1: Integration of Crossflow in Crossminer architecture.

their workflow execution (cf. Figure 36 in D6.5).

1.1 MOTIVATION

Two important factors when measuring the effectiveness of a software process or product are the level of abstraction and the level of reusability. Important aspects in this regard include increasing

the developer's productivity, decreasing the cost of software construction while preserving the desired quality and improving the reusability and maintainability of software.

Model Driven Development (MDD), is an approach to software development elevating models to first-class citizens of the process (Mohagheghi, Fernandez, Martell, Fritzsche, & Gilani, 2009). As such, it is focused on the creation of semantically-rich models encapsulating the problem and/or solution domains, while leaving the execution domain to model-based code generators.

Such models can be based on a graphical or textual representation (or a mixture of both) and be supported by graphical or textual design tools. These tools can either be generic and bound to a high-level abstract domain (such as object graphs), hence requiring more effort to describe domain-specific concepts or are bound to the domain and cannot be used for any other purpose. As such, it is both important to choose the appropriate level of abstraction and the type of representation for the domain in question.

1.2 OUTLINE

Modeling language editors are commonly split into three categories: graphical, textual, and hybrid; in D5.3 we presented an overview of those categories and their core differences. In this deliverable we briefly describe the selection of tools and frameworks upon which Crossflow editor implementations are built. Moreover, we present the current state of the Crossflow graphical and textual editor implementation in detail and alongside a running example. Next, we present the Crossflow web application enabling deployment, monitoring, and management of workflow execution in web browser environments.

2. CROSSFLOW WORKFLOW LANGUAGE

This section provides a short overview of the Crossflow language (Kolovos, Neubauer, Barmpis, Matragkas, & Paige, 2019) by describing its concepts and thus updating earlier deliverables such as D5.1. The Crossflow language is defined by the Crossflow Ecore metamodel (cf. simplified version in Figure 2) that is referenced by both graphical and textual language implementation of Crossflow presented in Section 3.2 and 4.2, respectively, as well as used by the Crossflow code generator to produce strongly-typed scaffolding Java code from workflow models.

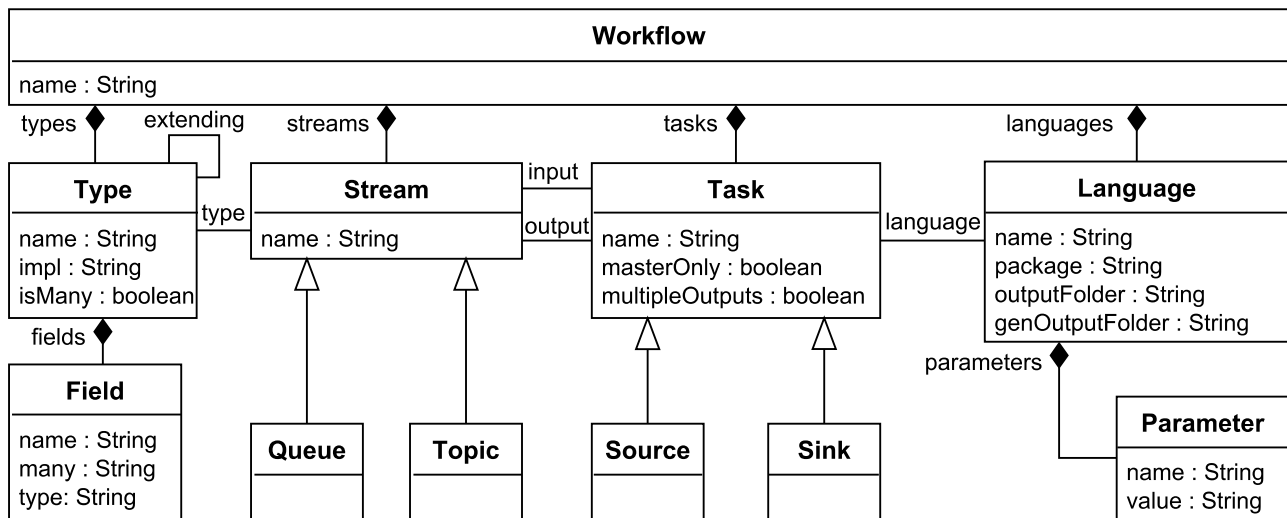


Figure 2: Simplified version of the Crossflow language metamodel (crossflow.ecore).

Workflow. In detail, instances of a Crossflow `Workflow` may contain `types`, `streams`, `tasks`, and `languages`.

Type. Instances of `Type` represent data types and may specify multiplicity (`isMany` attribute), existing (or intended) implementations through the value of the `impl` attribute as well as instances of `Field` detailed by a `name`, multiplicity (`isMany` attribute), and `type`.

Stream. Instances of `Stream` either represent queues or topics in a workflow model. Streams manifest either as instances of `Queue` or `Topic` and receive input from tasks as well as produce output for tasks. Queues and topics are following the point-to-point and publish-subscribe subscription model, respectively.

Task. Instances of `Task` are defined by `name`, execution restriction to only allow the master node to execute them (`masterOnly` attribute), and output multiplicity (`multipleOutputs` attribute). Moreover, tasks may manifest as instances of `Source` or `Sink`. Sources and sinks produce input and output for their containing workflow, respectively.

Language. Instances of `Language` are defined by `name`, `package`, generated source code output directory (`outputFolder` attribute), and generated base source code output directory (`genOutputFolder` attribute). Moreover, languages may own parameters. Instance of `Parameter` are defined by `name` and `value` to represent additional information required by a particular language. For example, the Python language requires the specification of a parameter named `module` for modularization, i.e. splitting programs into several programs to enable reuse and ease maintenance.

2.1 RUNNING EXAMPLE

This subsection provides a brief introduction of the running example appearing throughout the remaining sections of the document.

In summary, the initial activity in the running example, i.e. `TechnologySource` in Figure 3, entails reading tuples consisting of keyword and file extension from a comma separated file (CSV). Next, the `CodeSearcher` activity involves looking for instances of files from specific modeling technologies on GitHub, i.e. by using their file extensions and a keyword contained in the file as the matching metric. The repositories containing such files are then cloned locally and various analysis steps are performed to calculate the number of repositories, files and authors for each such technology (cf. `RepositorySearcher`). Finally, this analysis data is aggregated and output to console (cf. `RepositoryResultSink`).

More specifically, the first time the example workflow is executed in a distributed setup, different worker nodes will end up with different cloned Git repositories as a result of the execution of their repository search tasks (cf. `RepositorySearchDispatcher`). The next time the workflow is executed (e.g. after a bug fix or after adding more technologies to the input CSV file), repository search jobs, i.e. executing queries to the GitHub API, are routed to nodes that already have clones of relevant repositories from the previous execution (if available). Thus, unnecessary cloning of the same repositories in different nodes as required by subsequent repository analysis jobs, i.e. requiring access to repository clones, is prevented.

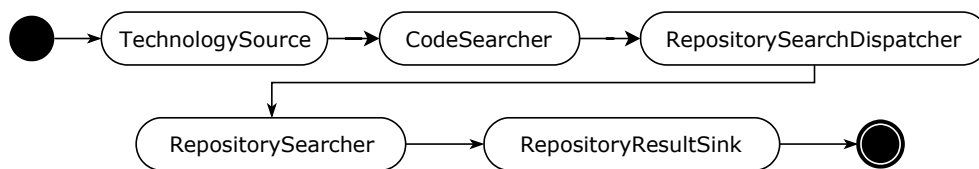


Figure 3: Crossflow Running Example Activity Diagram.

2.2 CODE GENERATION

The capability of generating base classes and executables from models created by either one of the Crossflow editors (cf. 3.2 and 4.2) has been made available to the Eclipse environment through context menu entries. Note that this capability requires the installation of Crossflow as outlined in Section 60. Moreover, make sure that Crossflow executables have been created beforehand

Crossflow models, i.e. files with the extension `crossflow_diagram` and `crossflow_model`, can be employed for base class code generation by right-click and selection of “Crossflow” followed by “Generate CROSSFLOW Base Classes” in the Package Explorer view (cf. Figure 4). In detail, this will generate base classes and skeletons of implementations classes in the location defined by the property “Gen Output Folder” and “Output Folder” of `Language` instances, respectively (cf. Figure 5). The keywords `outputFolder` and `genOutputFolder` of `Language` instances in textual Crossflow models serve the equivalent purpose (cf. Figure 6).

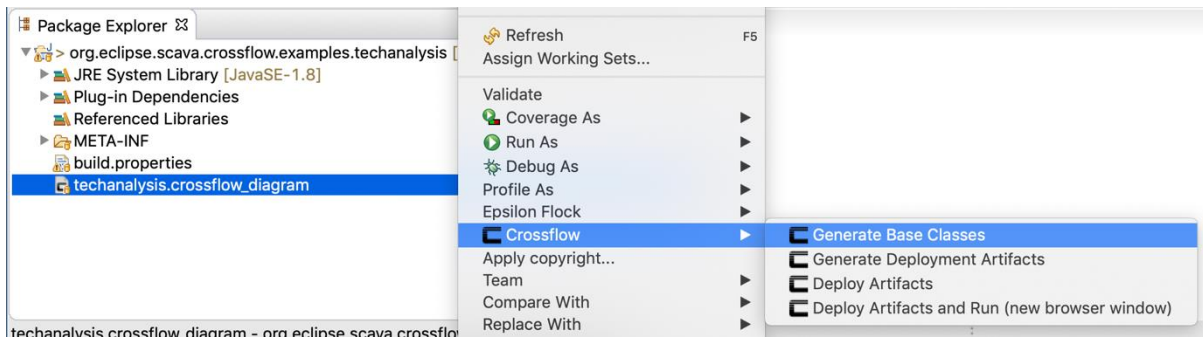


Figure 4: Crossflow context menu in Eclipse Package Explorer view.

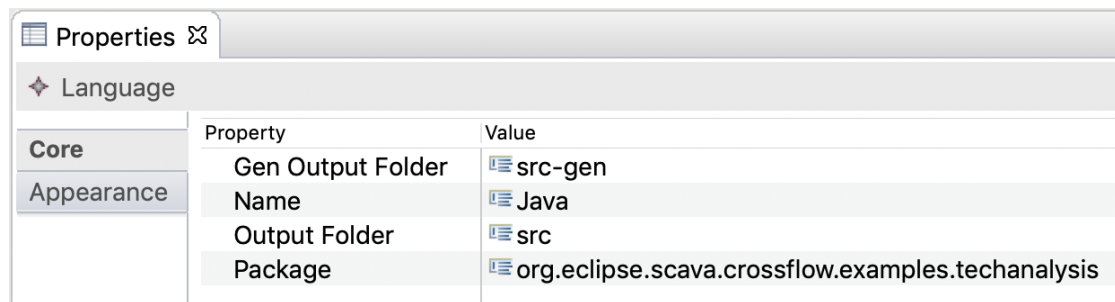


Figure 5: Running example workflow Java Language specification in Properties view.

```
languages {
  Language Java {
    package "org.eclipse.scava.crossflow.examples.technanalysis.xtext"
    outputFolder "src"
    genOutputFolder "src-gen"
  }
} // workflow languages
```

Figure 6: Running example workflow Java Language specification in Crossflow textual model editor.

Additionally, a folder named “experiment” is created containing workflow input and output data directories, a web browser-based representation of the Crossflow model for the web browser (abstract.graph), and a configuration file for the Crossflow model deployment, execution, monitoring, and management web application (experiment.xml).

Finally, the generation of executables of the selected Crossflow model as well as their assembly into a ready-to-deploy ZIP archive is accomplished by right-clicking on the Crossflow model in the Package Explorer view and selecting “Crossflow” followed by “Generate Deployment Artifacts”. Figure 7 shows the running example in the Package Explorer in its ready-to-deploy state.

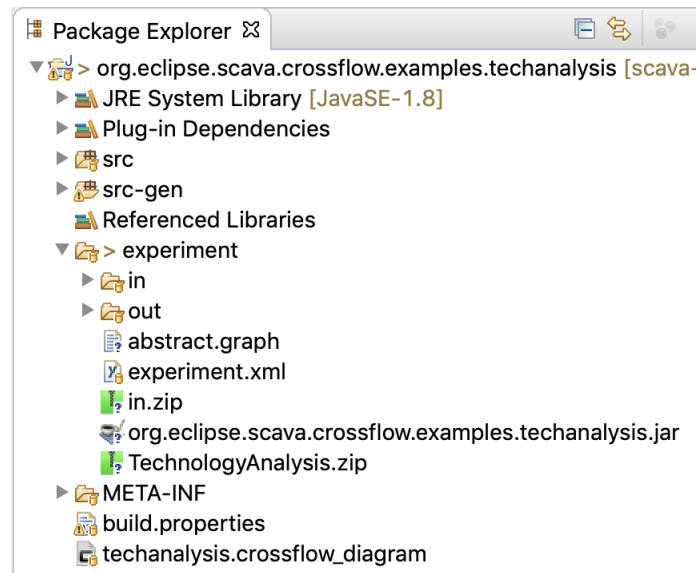


Figure 7: Running example project ready-to-deploy state in Package Explorer view.

3. GRAPHICAL WORKFLOW MODELING

This section provides background on graphical editors and in particular EuGENia¹ (Kolovos, García-Domínguez, Rose, & Paige, 2017) and GMF, i.e. the open-source (modeling) language graphical editor creation tool chosen to implement the Crossflow graphical editor. Crossflow is the domain-specific language created for constructing and editing workflows in CROSSMINER and is described in D5.2, Section 2.

3.1 BACKGROUND

Graphical editors focus on the use of shapes and images to provide a high-level representation of the domain model. Such editors often provide common graphics-related functionalities like inserting, deleting, moving and resizing graphical elements, which are directly linked to the domain model elements themselves; as such, any domain-level changes made in these graphical elements (such as the addition of a reference to another element) will be mirrored in the relevant domain model elements.

This type of editor is well suited for rapid creation/prototyping of models from small- to medium-sized languages with the use of drag-and-drop or copy-paste techniques. When the language starts becoming sufficiently large, graphical editors may end up overtly complex to use, as finding appropriate elements or connections in the UI may start becoming laborious. Similarly, when the model itself is sufficiently large, navigating such a visual space may end up taking substantial amounts of time and effort. Techniques like views and filters can alleviate some of these concerns but the core issue of managing large models in a graphical environment is a complex one, nevertheless.

EuGENia. EuGENia (Kolovos, García-Domínguez, Rose, & Paige, 2017) is a graphical editor generation language as part of the Epsilon² modeling suite. It is described as a front-end to GMF, aimed at speeding up the process of creating GMF-based graphical editors. Simple editors can be generated with the addition of a small number of annotations to the language (metamodel) denoting which elements are to be represented visually in the diagram (and which shape they will be), which references in the language (from one element to another) are to be visible, which element represents the diagram itself, etc. Further enhancements to the editor can be achieved in multiple ways, such as by editing the generated Java code of the editor and adding any custom logic regarding the representation of any diagram elements and their interactions, as well as any additional UI elements that may be useful for the current domain. An alternative is to create a polishing transformation (written in the Epsilon Object Language³) that has access to and can manipulate both the language model and the graphical model that is used to create the generated editor.

3.2 CROSSFLOW GRAPHICAL MODEL EDITOR (EUGENIA-BASED)

In order to use EuGENia to generate a graphical editor for Crossflow, the Crossflow language had to be augmented with relevant annotations denoting the various visual semantics of each element. Details on concepts available in the Crossflow language are described in Section 2. Moreover,

¹ <https://www.eclipse.org/epsilon/doc/EuGENia/>

² <https://www.eclipse.org/epsilon/>

³ <https://www.eclipse.org/epsilon/doc/eol/>

Appendix A: Crossflow Graphical Modeling Language Definition (crossflow.emf) provides a complete copy of the Crossflow graphical workflow modelling language definition, i.e. by means of crossflow.emf, that is current as by the time of the submission of this document. It shows a textual representation of the enhanced language, including the aforementioned annotations.

3.2.1 Editor Updates

In comparison to the previous version of the editor (cf. Figure 8), the current version of the editor (cf. Figure 9) reflects the following adjustments:

- The graphical element of queues has been customized by replacing the circle shape with the shape of a cylinder.
- The graphical element of sources and sinks has been customized by replacing the shape of rounded rectangles with trapezoids.
- The rectangularly-shaped concept of `Language` and fields to select the language for instances of tasks, i.e. including `Source` and `Sink`, i.e. specific forms of `Task`, has been introduced to conform to the latest version of the Crossflow metamodel.

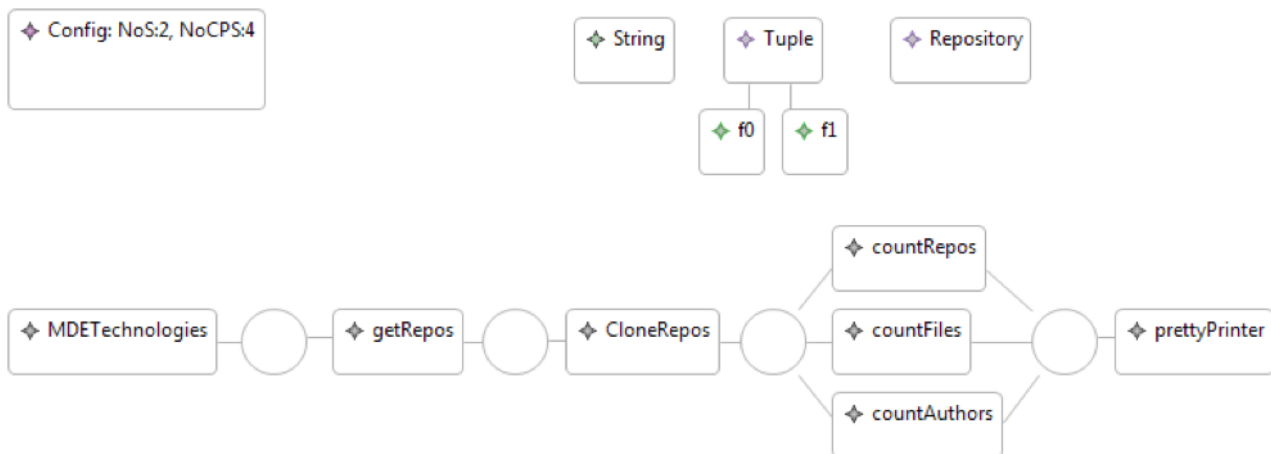


Figure 8: Running example workflow visualized in the previous version of the Crossflow graphical model editor (cf. D5.3).

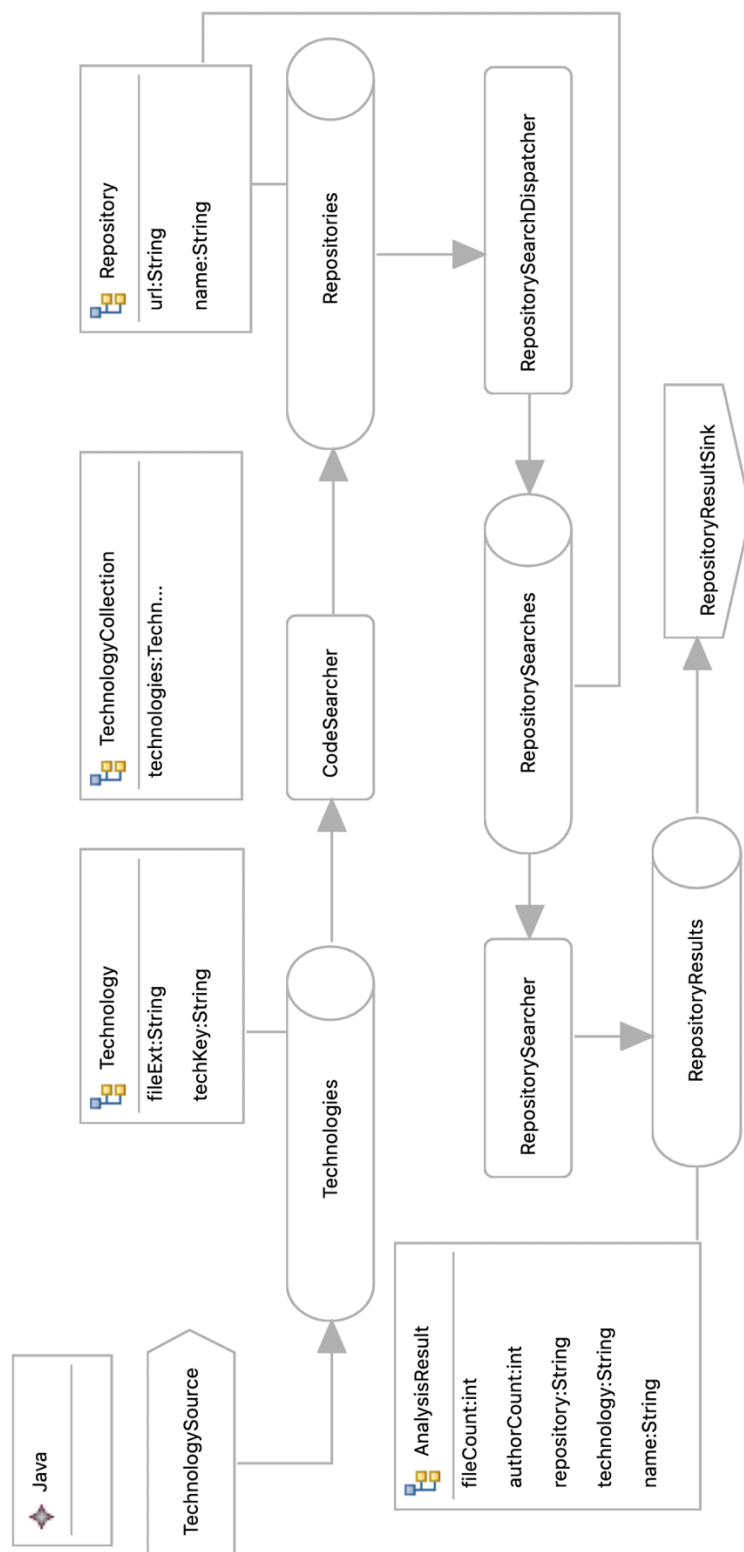


Figure 9: Running example workflow visualized in the current version of the Crossflow graphical model editor.

The Crossflow Diagram Editing editor enables populating Crossflow graphical models, i.e. files with the extension `.crossflow_diagram`, with elements of the Crossflow language by the use of the editor's palette panel and properties view. For example, the task `RepositorySearcher`

of type `CommitmentTask` uses a configuration of type `TechnologyCollection` and produces multiple outputs to the `RepositoryResults` queue (cf. Figure 10).

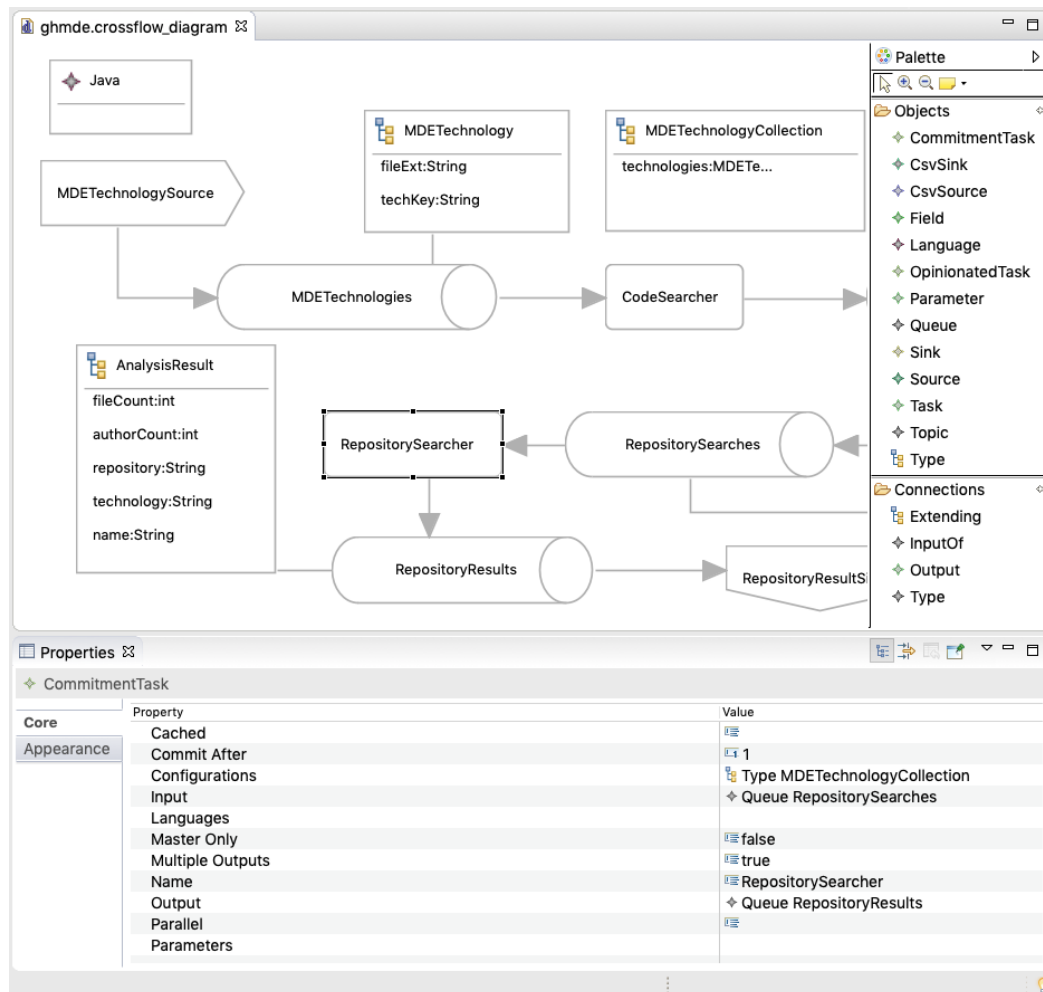


Figure 10: Crossflow Graphical Editor palette panel and properties view usage example.

4. TEXTUAL WORKFLOW MODELING

This section provides background on textual editors and in particular Xtext⁴ (Eysholdt & Behrens, 2010), i.e. the open-source (modeling) language textual editor creation tool chosen to implement the Crossflow textual editor, and Flexmi⁵ (Kolovos, Matragkas, & Garcia-Dominguez, Towards Flexible Parsing of Structured Textual Model Representations, 2016). Crossflow is the domain-specific language created for constructing and editing workflows in CROSSMINER and is described in D5.3, Section 3.

4.1 BACKGROUND

Textual editors focus on using structured text for managing domain models. This text is often a simplified view of the domain elements themselves, often augmented with small textual or graphical queues to aid the understanding of the structure. Such editors often offer a tree-based view of the model, whereby each domain element is displayed as a separate line of text, containing any of its features nested inside it; other views can include table views, commonly used for editing the features of domain elements.

This type of editor can easily represent the containment structure of a model and offers an alternative development approach for those more adept at creating text-based documents. Whilst textual editors can show the structure of arbitrarily large models, managing such models can still remain challenging as one may need to find and manipulate specific elements in the model that they may not be able to uniquely identify without navigating a large portion of the model itself.

Xtext. To support the development of textual Domain-Specific Modeling Languages (DSMLs) (Kelly & Tolvanen, 2008), frameworks, such as Xtext (Eysholdt & Behrens, 2010), emerged, enabling language designers to ease and speed-up language development by leveraging advances in editor technology of mainstream IDEs. Such frameworks automate, for instance, the creation of language specific parsers, serializers, and editors providing basic syntax highlighting, content-assist, folding, jump-to-declaration, and reverse reference lookup across multiple files.

The kind of language created by employing Xtext, which relies on the EMF, can range from small DSMLs to fully-blown General Purpose Languages (GPL). This also includes textual configurations files or human-readable requirement documents. The motivation of Xtext is to automate the generation of basic tooling support for language specifications and thus increase readability, writability, and understandability of documents written in those languages.

Starting with a grammar definition, Xtext generates a parser, serializer and a basic editor implementation for the specified language. Moreover, generated artifacts can be adapted via dependency injection and the use of the Xtext API. Thus, generated artifacts can include customized implementations, such as for validation and linking/scoping.

Flexmi. Flexmi is a flexible algorithm for parsing well-formed XML documents into Ecore metamodel-conforming in-memory models. The parser developed for Flexmi performs a depth-first traversal of the elements in the XML document, expects to find a nsuri processing instruction, i.e. declaring a unique namespace identifier, and employs a stack to keep track of its position during document parsing. The Flexmi implementation also includes a renderer that is accessible as an Eclipse view named “Flexmi Renderer”.

⁴ <http://www.xtext.org>

⁵ <https://www.eclipse.org/epsilon/doc/articles/flexmi/>

4.2 CROSSFLOW TEXTUAL MODEL EDITOR (XTEXT-BASED)

In this section we present the Crossflow textual DSML, i.e. conforming to the structural components of the Crossflow language as defined by the Crossflow metamodel (cf. Figure 2).

Appendix A: Crossflow Graphical Modeling Language Definition (crossflow.emf) provides a complete copy of the language grammar that is current as by the time of the submission of this document.

The Crossflow Editor editor enables populating Crossflow textual models, i.e. files with the extension `.crossflow_model`, with elements of the Crossflow language either by inserting textual tokens or by using of the editor's content assist feature accessible through the keyboard shortcut `CTRL + Space` followed by the selection of an element offered by the displayed list of viable textual tokens to be inserted at the current position of the cursor. For example, selecting `CodeSearcher` as viable value for the `Repositories` queue's `outputOf` attribute (cf. Figure 11).

```
Workflow TechnologyAnalysis {
    // streams for incoming and outgoing tasks
    streams {
        Queue Technologies {
            type Technology
        } // MDETechnologies queue
        Queue Repositories {
            type Repository
        } // Repositories queue
        Queue RepositorySearches {
            type Repository
        } // RepositorySearches queue
        Queue RepositoryResults {
            type AnalysisResult
        } // RepositoryResults queue
    } // workflow streams

    tasks {
        masterOnly CsvSource TechnologySource {
            fileName "input.csv"
            output ( )
            language ↗ Repositories - TechnologyAnalysis.Repositories
            } // MDETech
        multipleOutput TechnologySource {
            input ( Technologies - TechnologyAnalysis.Technologies
            output (
            language ↗ CodeSearcher
            } // CodeSearcher
    }
}
```

Figure 11: Crossflow Textual Editor content assist usage example.

```
Workflow TechnologyAnalysis {
    // streams for incoming and outgoing tasks
    streams {
        Queue Technologies {
            type Technology
        } // MDETechnologies queue
        Queue Repositories {
            type Repository
        } // Repositories queue
        Queue RepositorySearches {
            type Repository
        } // RepositorySearches queue
        Queue RepositoryResults {
            type AnalysisResult
        } // RepositoryResults queue
    } // workflow streams

    tasks {
        masterOnly CsvSource TechnologySource {
            fileName "input.csv"
            output ( TechnologyType )
            language ↗ Change to 'Repositories'
            } // MDETech
        multipleOutput TechnologySource {
            input ( Technologies
            output (
            language ↗ Change to 'Technologies'
            } // CodeSearcher
    }
}
```

Figure 12: Crossflow Textual Editor validation and quick fix usage example.

Moreover, the editor offers validation and quick fix features. The former is triggered as a result of any change performed to the textual model. The latter is accessible by clicking on the light bulb error indicator and selecting an appropriate solution (cf. Figure 12).

The complete runtime example's Crossflow textual model equivalent to the Crossflow graphical model (cf. Figure 9) is depicted in Figure 13 (`types` and `languages`), Figure 14 (`tasks`), and Figure 15 (`streams`).

```

60  types {
61
62  Type Technology {
63    fields {
64      Field fileExt {
65        type String
66      }
67      Field techKey {
68        type String
69      }
70    }
71  } // MDETechnology type
72
73  isMany Type Repository {
74    extending ( Technology )
75    fields {
76      Field url {
77        type String
78      }
79      Field name {
80        type String
81      }
82    }
83  } // Repository type
84
85  isMany Type AnalysisResult {
86    fields {
87      Field fileCount {
88        type int
89      }
90      Field authorCount {
91        type int
92      }
93      Field repository {
94        type String
95      }
96      Field technology {
97        type String
98      }
99      Field name {
100        type String
101      }
102    }
103  } // AnalysisResult type
104
105  Type TechnologyCollection {
106    fields {
107      many Field technologies {
108        type Technology
109      }
110    }
111  }
112
113 } // workflow data types
114
115 languages {
116  Language Java {
117    package "org.eclipse.scava.crossflow.examples.techanalysis.xtext"
118    outputFolder "src"
119    genOutputFolder "src-gen"
120  }
121 } // workflow languages
122
123 } // workflow definition

```

Figure 13: Running example workflow types and languages visualized in the Crossflow textual model editor.

```

24  tasks {
25
26  masterOnly CsvSource TechnologySource {
27      fileName "input.csv"
28      output ( Technologies )
29      languages(Java)
30  }// MDETechnologySource
31
32  multipleOutputs Task CodeSearcher {
33      input ( Technologies )
34      output ( Repositories )
35      languages(Java)
36  }// CodeSearcher task
37
38  multipleOutputs Task RepositorySearchDispatcher {
39      input ( Repositories )
40      output ( RepositorySearches )
41      languages(Java)
42  }// RepositorySearchDispatcher task
43
44  multipleOutputs CommitmentTask RepositorySearcher {
45      commitAfter 1
46      input ( RepositorySearches )
47      output ( RepositoryResults )
48      languages(Java)
49      configurations ( TechnologyCollection )
50  }// RepositorySearcher task
51
52  masterOnly CsvSink RepositoryResultSink {
53      fileName "output.csv"
54      input ( RepositoryResults )
55      languages(Java)
56  }// RepositoryResultSink task
57
58  }// workflow tasks
59
1  Workflow TechnologyAnalysis {
2
3  // streams for incoming and outgoing tasks
4  streams {
5
6  Queue Technologies {
7      type Technology
8  }// MDETechnologies queue
9
10 Queue Repositories {
11     type Repository
12 }// Repositories queue
13
14 Queue RepositorySearches {
15     type Repository
16 }// RepositorySearches queue
17
18 Queue RepositoryResults {
19     type AnalysisResult
20 }// RepositoryResults queue
21
22 }// workflow streams
23

```

Figure 15: Running example workflow streams visualized in the Crossflow textual model editor.

Figure 14: Running example workflow tasks visualized in the Crossflow textual model editor.

4.2.1 Editor Updates

Similarly to the updates made in the graphical editor (cf. Section 3.2.1), the textual editor has been adapted to conform to the latest version of the Crossflow metamodel (cf. Figure 2). For example, grammar rules for representing the concept of `Language` and `Field` have been introduced.

4.3 CROSSFLOW XML-BASED MODEL EDITING (FLEXMI)

In this section we present the result of the running example modelled using the XML-based Flexmi editor (cf. Figure 16) as well as its rendered version in the Flexmi Renderer (cf. Figure 17).

```

1 <?nsuri org.eclipse.scava.crossflow?>
2 <?render-graphviz-dot graphviz.egl?>
3 <_>
4 <workflow name="TechnologyAnalysis">
5
6 <!-- TASKS -->
7 <csvsource name="TechnologySource" masterOnly="true" output="Technologies" fileName="input.csv" languages="Java"/>
8 <task name="CodeSearcher" multipleOutputs="true" input="Technologies" output="Repositories" languages="Java"/>
9 <task name="RepositorySearchDispatcher" multipleOutputs="true" input="Repositories" output="RepositorySearches" languages="Java"/>
10 <commitmenttask name="RepositorySearcher" multipleOutputs="true" input="RepositorySearches" output="RepositoryResults" languages="Java"
11   commitAfter="1" configurations="TechnologyCollection"/>
12 <csvsink name="RepositoryResultSink" masterOnly="true" fileName="output.csv" input="RepositoryResults" languages="Java"/>
13
14 <!-- QUEUES -->
15 <queue name="Technologies" type="Technology"/>
16 <queue name="Repositories" type="Repository"/>
17 <queue name="RepositorySearches" type="Repository"/>
18 <queue name="RepositoryResults" type="AnalysisResult"/>
19
20 <!-- LANGUAGES (Java is set by default, can be neglected, and appears here for integrity purposes only) -->
21 <language name="Java" package="org.eclipse.scava.crossflow.examples.technanalysis.flexmi" outputFolder="src" genOutputFolder="src-gen"/>
22
23 <!-- TYPES -->
24 <type name="Technology">
25   <field name="fileExt" type="String"/>
26   <field name="techKey" type="String"/>
27 </type>
28 <type name="Repository" isMany="true" extending="Technology">
29   <field name="url" type="String"/>
30   <field name="name" type="String"/>
31 </type>
32 <type name="AnalysisResult" isMany="true" extending="Technology">
33   <field name="fileCount" type="int"/>
34   <field name="authorCount" type="int"/>
35   <field name="repository" type="String"/>
36   <field name="technology" type="String"/>
37   <field name="name" type="String"/>
38 </type>
39 <type name="TechnologyCollection">
40   <field name="technologies" type="Technology" many="true"/>
41 </type>
42
43 </workflow>
44 </_>

```

Figure 16: Running example workflow visualized in the Flexmi Editor.

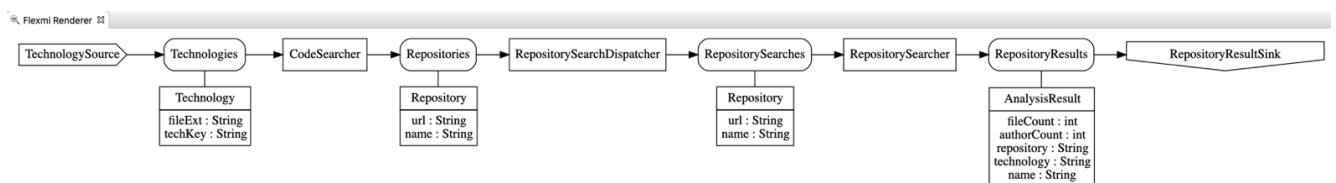


Figure 17: Running example workflow visualized in the Flexmi Renderer.

5. WORKFLOW DEPLOYMENT, EXECUTION, AND MONITORING

This section presents the implementation of for the deployment, execution, and monitoring of workflow models created by the Crossflow Graphical Model Editor and Crossflow Textual Model Editor presented in 3.2 and 4.2, respectively. Finally, we present the Crossflow Command Line Interface (CLI) in X.

5.1 CROSSFLOW MODEL DEPLOYMENT AND EXECUTION (ECLIPSE)

The capability of deploying and executing models that have been created by either one of the Crossflow editors (cf. Sections 3.2 and 4.2) has been made available to the Eclipse environment through context menu entries. Note that this capability requires the installation of Crossflow as outlined in Section 6. Moreover, make sure that Crossflow executables have been created beforehand. Crossflow models, i.e. files with the extension `crossflow_diagram` and `crossflow_model`, can be deployed by right-click and selection of “Crossflow” followed by “Deploy Artifacts” in the Package Explorer view (cf. Figure 18).

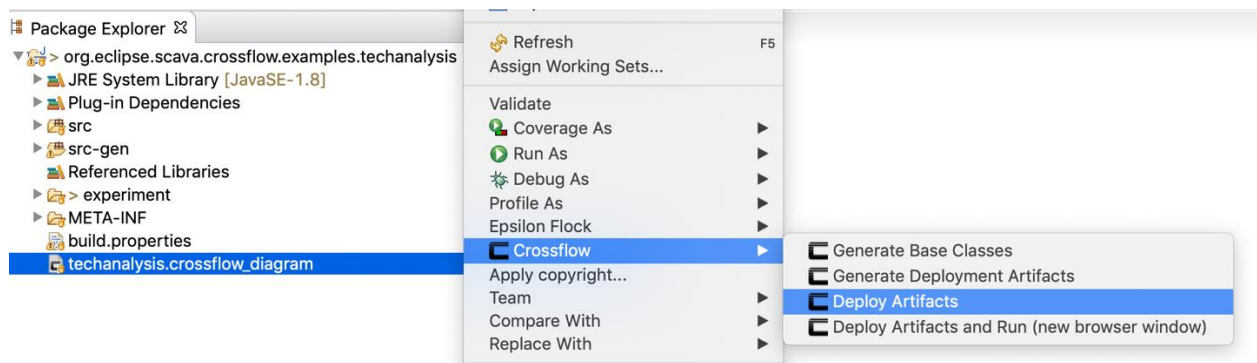


Figure 18: Running example workflow upload in Crossflow context menu.

The result of this procedure is visible by pointing a web browser to the start page of the Crossflow web application (cf. Figure 19).

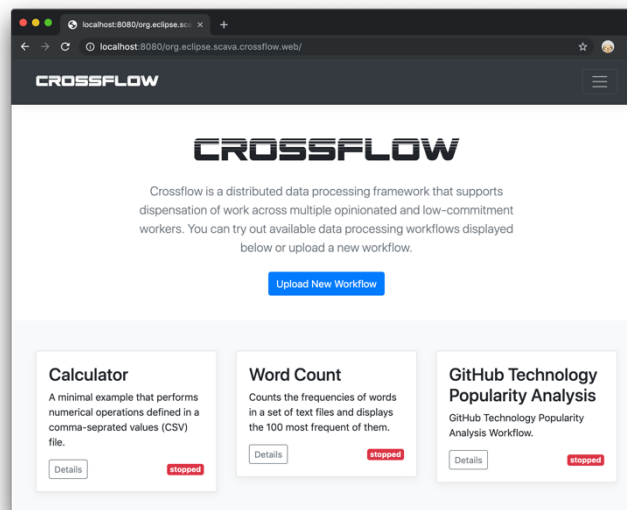


Figure 19: Running example workflow upload completed in Crossflow web application.

Further, to trigger the execution of the selected Crossflow model right after it has been deployed, right-click the Crossflow model in the Package Explorer view and select “Crossflow” followed by “Deploy Artifacts and Run (new browser window)”. Consequently, a new instance of the operating system’s default browser will open and navigate to Crossflow web application start page and display the uploaded workflow as “running”, i.e. instead of “stopped”, after a delay of approximately three seconds, i.e. allowing the web server runtime to load uploaded resources.

5.2 CROSSFLOW MODEL DEPLOYMENT, EXECUTION, MONITORING, AND MANAGEMENT (WEB-BASED)

This subsection describes steps involved during deployment, execution, monitoring, and management of Crossflow models by the use of the Crossflow web application.

Configuration. Workflows that are deployed within the Crossflow web application are configured by means of an XML-based experiment configuration file. Figure 20 depicts the generated experiment configuration of the running example adapted in line two, seven, and thirteen by replacing the auto-generated title, summary, and description stub of the experiment, respectively. Line three specifies the root class of the running example, i.e. extending `org.eclipse.scava.crossflow.runtime.Workflow`, owning the workflow’s streams and tasks as well as behaviour to launch master and worker instances. Line four defines the name and path to the running example workflow packaged as a Java ARchive (JAR). Lines five and six specify the experiment’s input and output directory, respectively. Lines ten and eleven specify the experiment’s input and output files for workflow `Source` and `Sink` and appear as “Input” and “Output” tab on the experiment’s dedicated page in the Crossflow web application, respectively. Line eight defines the target web server running an instance of the Crossflow web application by means of its IP address or DNS name followed by colon and port number. Currently, the value of the `webServer` attribute is being picked up by the Eclipse Package Explorer Crossflow context menu during workflow deployment and execution (cf. Section 5.1) and may require adaptation, e.g. `localhost:80` (or: `localhost`), for deployment on the Crossflow Docker image web server instance (cf. Section 6.3).

```

1 <experiment
2   title="GitHub Technology Popularity Analysis"
3   class="org.eclipse.scava.crossflow.examples.techanalysis.TechnologyAnalysis"
4   jar="org.eclipse.scava.crossflow.examples.techanalysis.jar"
5   input="in"
6   output="out"
7   summary="GitHub Technology Popularity Analysis Workflow."
8   webServer="localhost:8080"
9 >
10 <input path="input.csv" title="Input"/>
11 <output path="output.csv" title="Output"/>
12 <description>
13   Popularity analysis workflow querying GitHub for technologies defined by file extension and keyword.
14 </description>
15 </experiment>

```

Figure 20: Running example workflow XML-based experiment configuration.

Deployment. Workflows can be deployed within the Crossflow web application by browsing to the “Upload New Workflow” page accessible through a button located on the main page. The

upload page displays a form requiring the input of an experiment name, an assembled workflow archive, and (optionally) the selection of a checkbox for immediate execution of the specified workflow (cf. Figure 21). More specifically, the experiment name acts as unique internal identifier within the context of the Crossflow master node and any worker nodes contributing to the same master node. The assembled workflow archive is represented by the ZIP archive file generated by the use of the Crossflow context menu and in particular the selection of “Generate Deployment Artifacts” (cf. Section 2.2). Clicking the “Upload” button on the upload page causes the specified workflow to be deployed and (optionally) executed—producing the equivalent result as achieved by selecting “Deploy Artifacts” or “Deploy Artifacts and Run (new browser window)” from the Crossflow context menu in the Eclipse Package Explorer view in case the “Launch experiment immediately” checkbox is checked. Further, the same artifacts generated in the “Generate Deployment Artifacts” step can be applied as deployment artifacts for worker nodes. Alternatively, to decrease storage, e.g. by excluding workflow input files only required for master nodes, a customized JAR may be assembled by right-click on the project name and selecting “Export...”, then “JAR file” and following the “JAR Export” wizard.

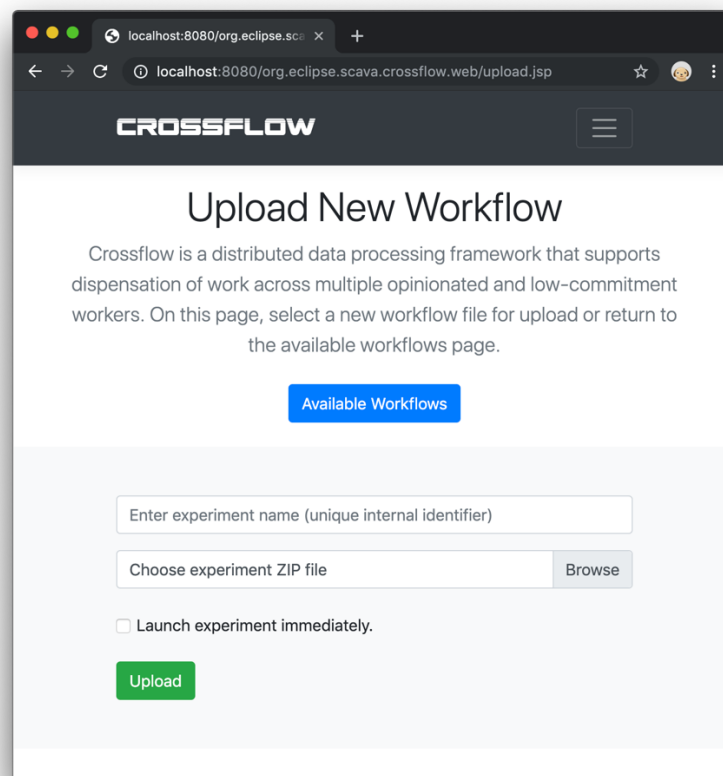
The image shows a web browser window displaying the 'Upload New Workflow' page of the Crossflow web application. The browser's address bar shows the URL 'localhost:8080/org.eclipse.scava.crossflow.web/upload.jsp'. The page has a dark header with the 'CROSSFLOW' logo and a hamburger menu icon. The main content area is white and features the title 'Upload New Workflow'. Below the title is a descriptive paragraph about Crossflow as a distributed data processing framework. A blue button labeled 'Available Workflows' is positioned below the text. The form includes a text input field for 'Enter experiment name (unique internal identifier)', a file selection area with a 'Choose experiment ZIP file' label and a 'Browse' button, and a checkbox labeled 'Launch experiment immediately'. At the bottom of the form is a green 'Upload' button.

Figure 21: Workflow upload form in Crossflow web application.

Dependencies. Currently, referenced sources and dependencies in Crossflow workflow implementations are made available to the web server running the Crossflow web application by packaging and supplying dependencies as JAR files to the web server’s `lib` folder. In detail, Eclipse workspace projects depicted in the Eclipse Package Explorer view can be packaged into JAR files by right-click on the project name and selecting “Export...”, then “JAR file” and

following the “JAR Export” wizard. In particular, in the running example, the following three projects have been packaged and exported as JAR files into the Apache Tomcat library directory:

- `org.eclipse.scava.crossflow.restmule.core`
- `org.eclipse.scava.crossflow.restmule.dependencies`
- `org.eclipse.scava.crossflow.restmule.client.github`

Advanced Tab. The experiment page of a workflow provides three build-in tabs. The “Advanced” tab (cf. Figure 22) shows an overview of the experiment as well as a checkbox to enable/disable automatic refresh of information displayed throughout several tabs on the experiment page. Moreover, the current status of the broker, i.e. either “stopped” or “running”, to which the Crossflow master node is connected to is displayed. Further, the occurrence of a previous execution as well as cached data of the experiment is displayed. Also, the path in the file system of the master node where experiments and their data is located is displayed by “Serving from”.

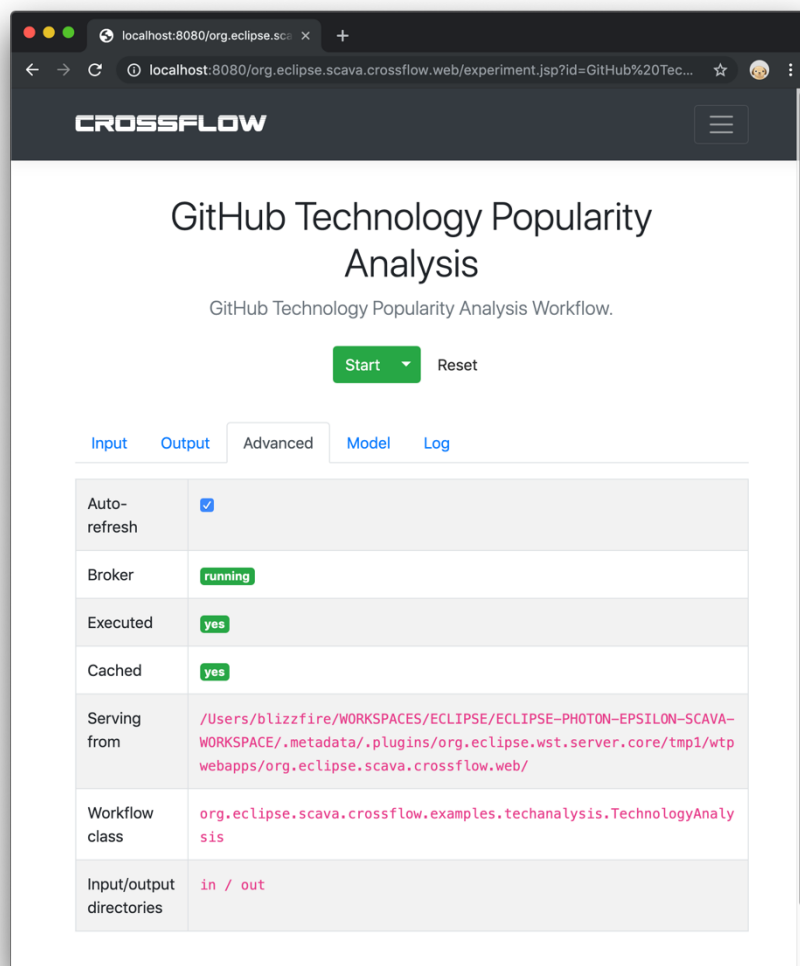


Figure 22: Running example workflow experiment page “Advanced” tab in Crossflow web application.

Execution. Workflows available to the Crossflow web application can be launched either automatically or manually. In the former case, either “Deploy Artifacts and Run (new browser window)” is selected from the Crossflow context menu or the “Launch experiment immediately” checkbox is checked during the deployment phase, respectively. In the latter case, the workflow is launched by browsing to the workflow’s dedicated experiment page within the Crossflow web application followed by a click on the “Start” button.

Log Tab. The log tab on the experiment page (cf. Figure 23) displays log messages produced during the execution of a workflow and in particular instances of `CrossflowLogger` that is accessible to classes extending `org.eclipse.scava.crossflow.runtime.Workflow`, such as `TechnologyAnalysis` in the running example, by passing severity level, i.e. info, warning, or error, and message to the `log` method.

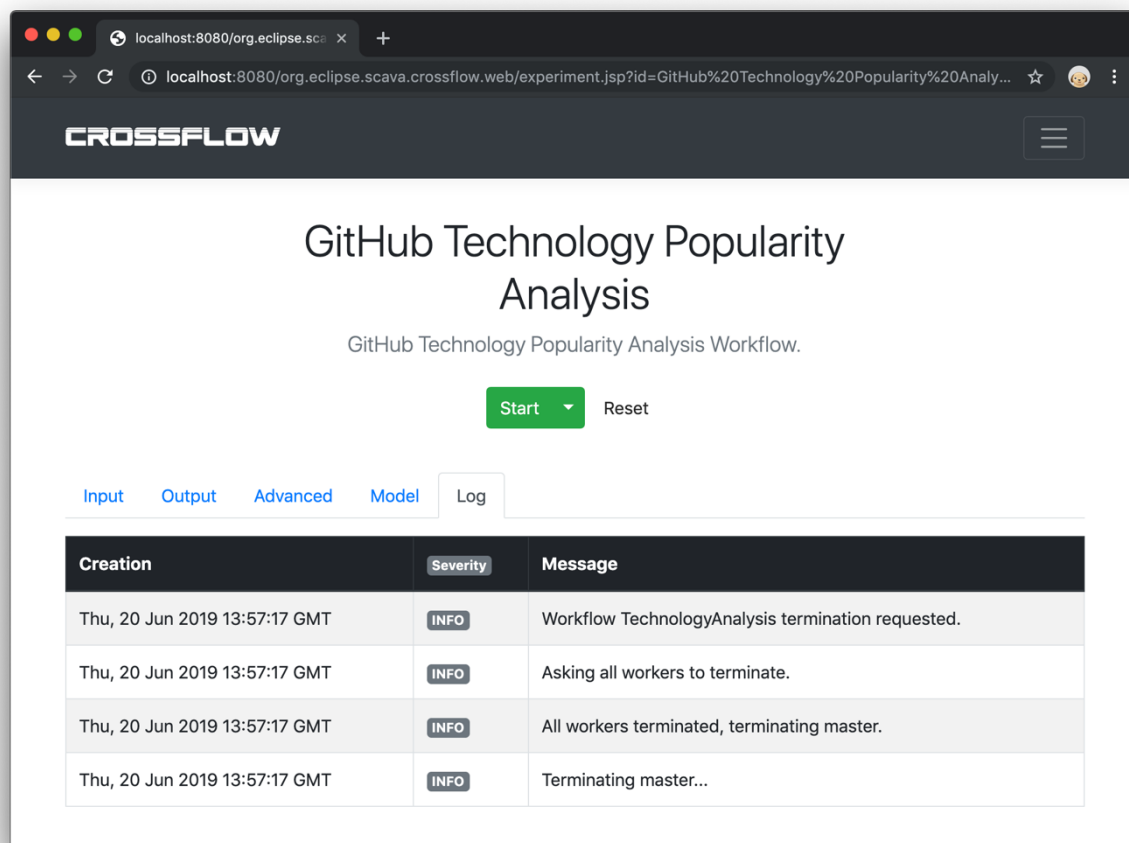


Figure 23: Running example workflow experiment page “Log” tab in Crossflow web application.

Monitoring. The model tab on the experiment page (cf. Figure 24) visualizes the runtime model representation that has been auto-generated from its runtime workflow model during code generation (cf. Section 2.2). Web browser instances pointing to the experiment page of a workflow subscribe to a set of ActiveMQ topics that are maintained by the Crossflow runtime and broadcast the state of experiment execution through messages parsed into visual updates to the runtime model.

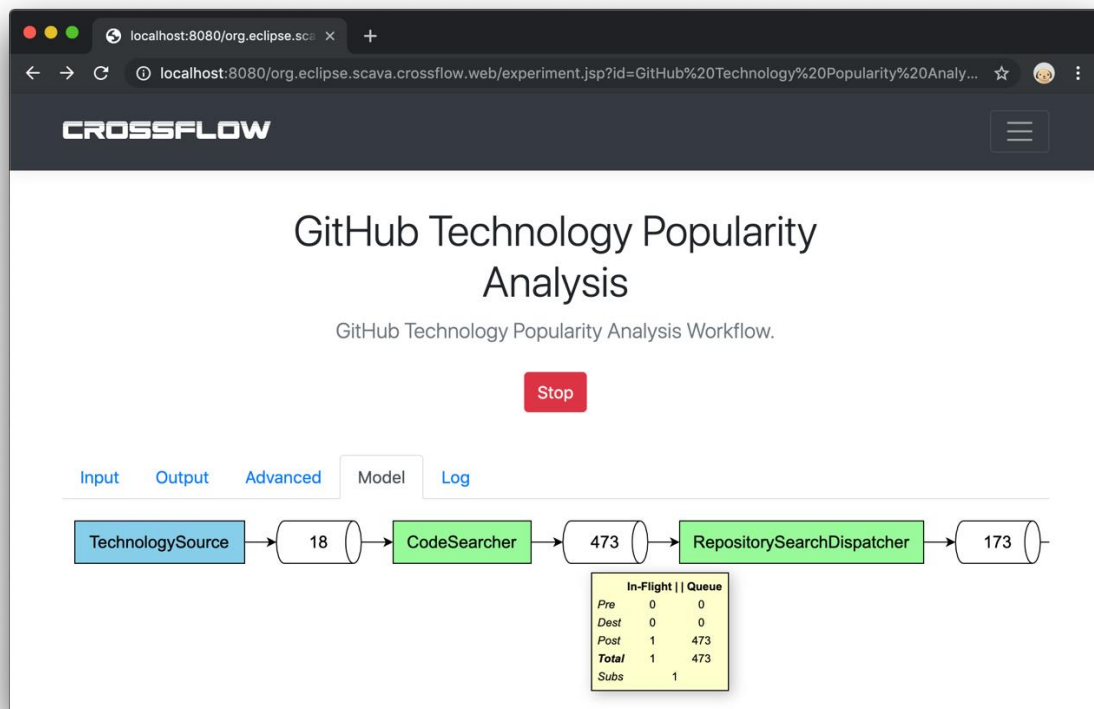


Figure 24: Running example workflow experiment page "Model" tab in Crossflow web application.

The status of a task is reported by its color. The color lightcyan, skyblue, palegreen, salmon, and slategray indicate that a task has been started, is waiting, in progress, blocked, or finished, respectively (cf. Table 1). Further, if a task is colored white, no status has been reported.

Color	Semantics
Lightcyan	Started
Skyblue	Waiting
Palegreen	In Progress
Salmon	Blocked
Slategray	Finished
White	N/A

Table 1: Task status color semantics as reported on workflow experiment page "Model" tab in Crossflow web application.

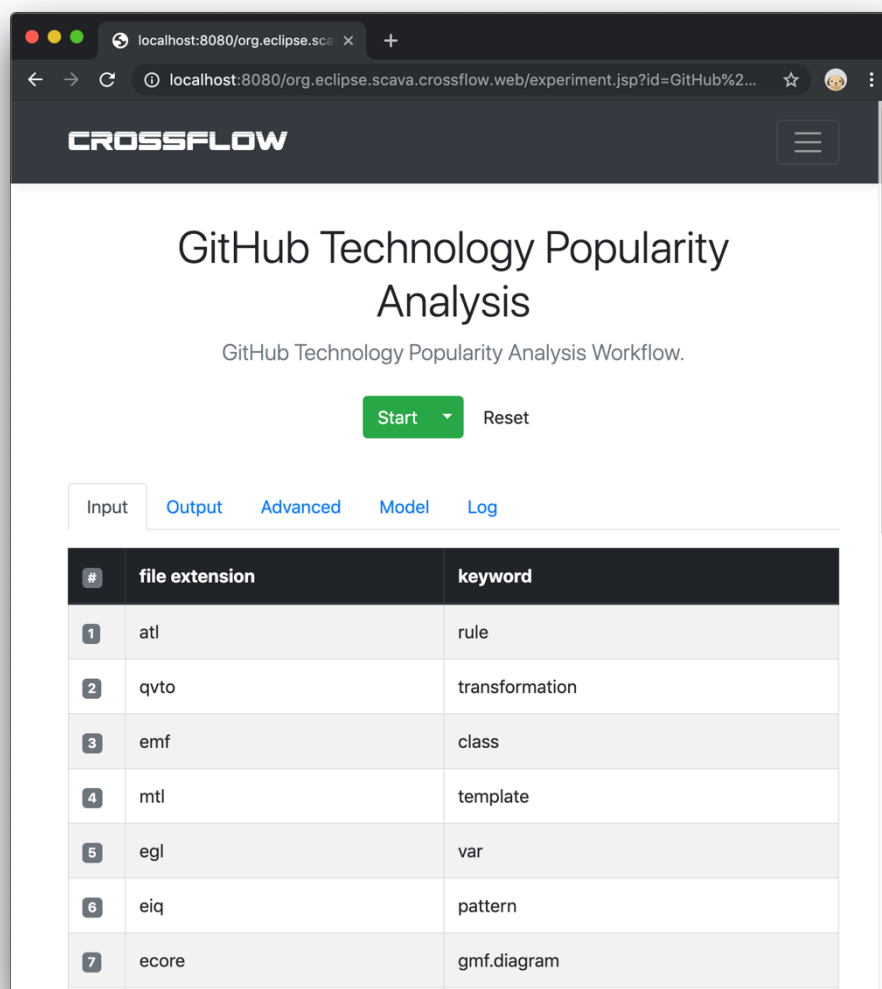
In-Flight Queue		
Pre	0	0
Dest	0	0
Post	1	473
Total	1	473
Subs	1	

Figure 25: Running example workflow experiment page "Model" tab tooltip of RepositorySearches queue in Crossflow web application.

The total number of messages residing in a queue is reported by the number displayed inside the visible shape of a queue. Hovering over a particular queue shape displays a tooltip with detailed information about the status of a queue (cf. Figure 25). Each queue is made up of three instances

of ActiveMQ queues that are referred to as *Pre*, *Dest*, and *Post* in Crossflow. Tasks connect to *Post* instances of queues. *Subs* indicates the number of subscribers. More details are described in D5.6.

Workflow-Specific Tabs. The running example owns two workflow-specific tabs—Input and Output. These tabs are the result of a `Source` and `Sink` instance defined in the workflow model, respectively. Figure 26 depicts the manifestation of the running example’s input in the Crossflow web application (cf. “Input” tab). More specifically, the `TechnologySource` task submits instances of `Technology`, i.e. consisting of file extension and MDE technology-specific keyword, to the `Technologies` queue. The “Output” tab (cf. Figure 27) represents the running example’s output in the Crossflow web application before experiment launch.



#	file extension	keyword
1	atl	rule
2	qvto	transformation
3	emf	class
4	mtl	template
5	egl	var
6	eiq	pattern
7	ecore	gmf.diagram

Figure 26: Running example workflow experiment page “Input” tab in Crossflow web application.

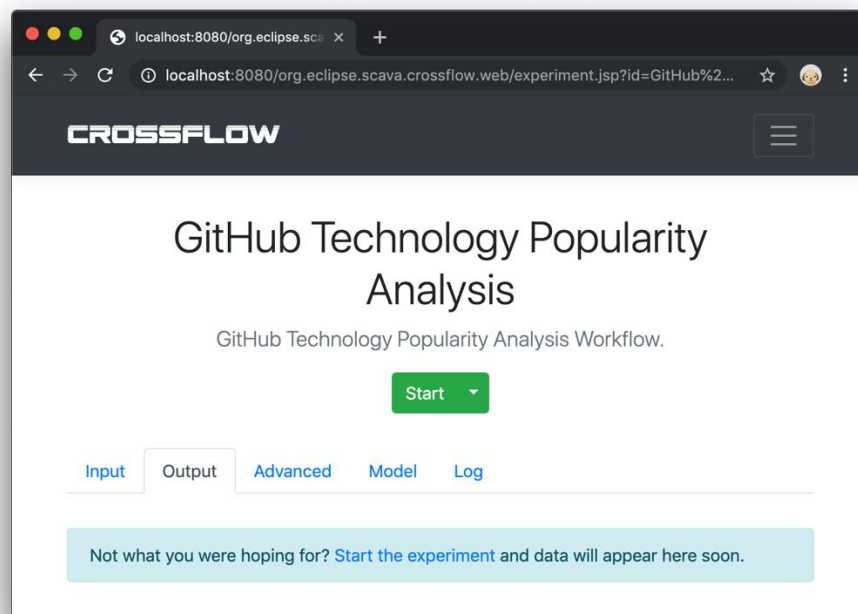


Figure 27: Running example workflow experiment page “Output” tab in Crossflow web application.

Management. The Crossflow web application provides several paths to manage deployed workflows from their experiment page. Clicking the “Stop” button (cf. Figure 28) triggers the submission of termination requests to worker nodes followed by a waiting period and termination of the master node. In case a workflow has been executed beforehand and is currently not being executed, a “Reset” button appears on the experiment page (cf. Figure 29). Resetting an experiment entails the removal of cached data as well as re-establishing its initial state, i.e. the state of initial deployment after completed workflow upload.

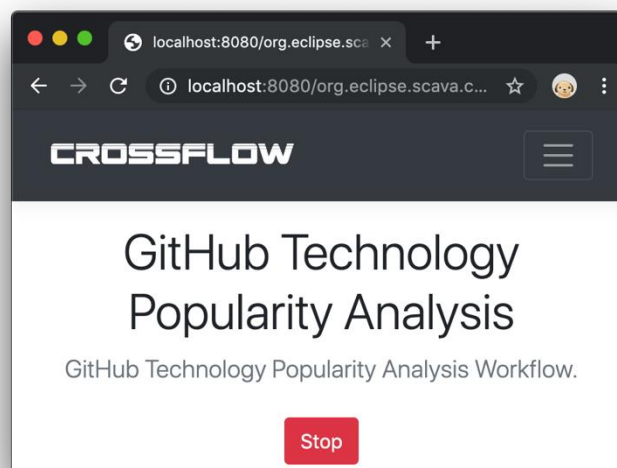


Figure 28: Running example workflow experiment page in Crossflow web application during workflow execution.

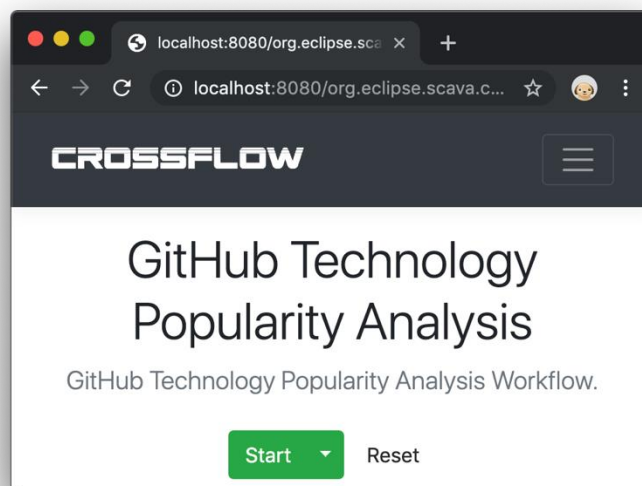


Figure 29: Running example workflow experiment page in Crossflow web application after halted workflow execution.

Moreover, the “Model” tab of a workflow’s experiment page offers the ability to clear the content of a particular queue and all queues of a workflow, respectively. In the former case, a particular queue is cleared by right-clicking the selected queue (cf. Figure 30) followed by left-clicking the option displayed by the appearing context menu and confirming the operation in the exposed pop-up window (cf. Figure 31). In the latter case, all queues are cleared by right-clicking on an empty spot in the runtime model visualization (cf. Figure 32) followed by left-clicking the option displayed by the appearing context menu and confirming the operation in the exposed pop-up window.

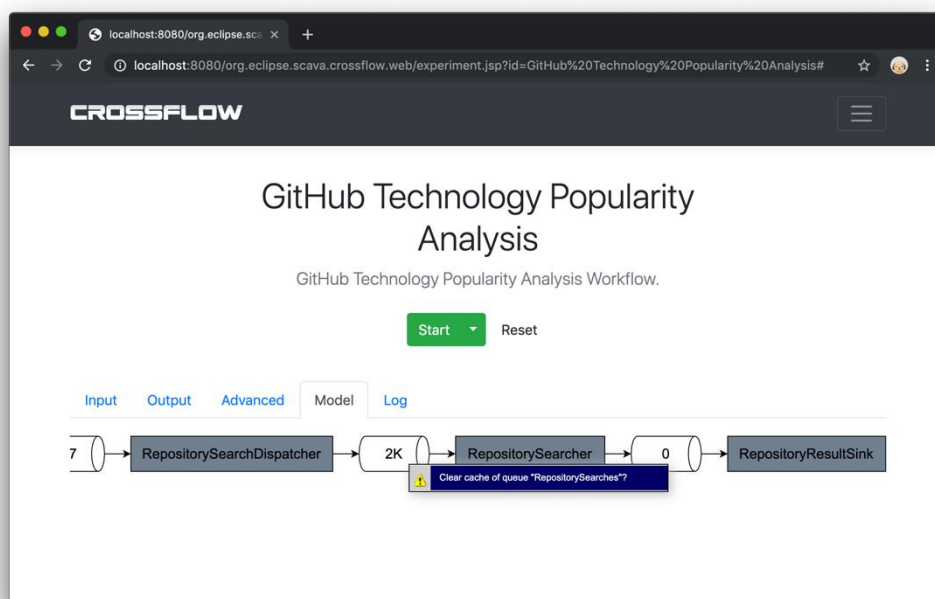


Figure 30: Running example workflow experiment page in Crossflow web application after halted workflow execution and before clearing individual cache.

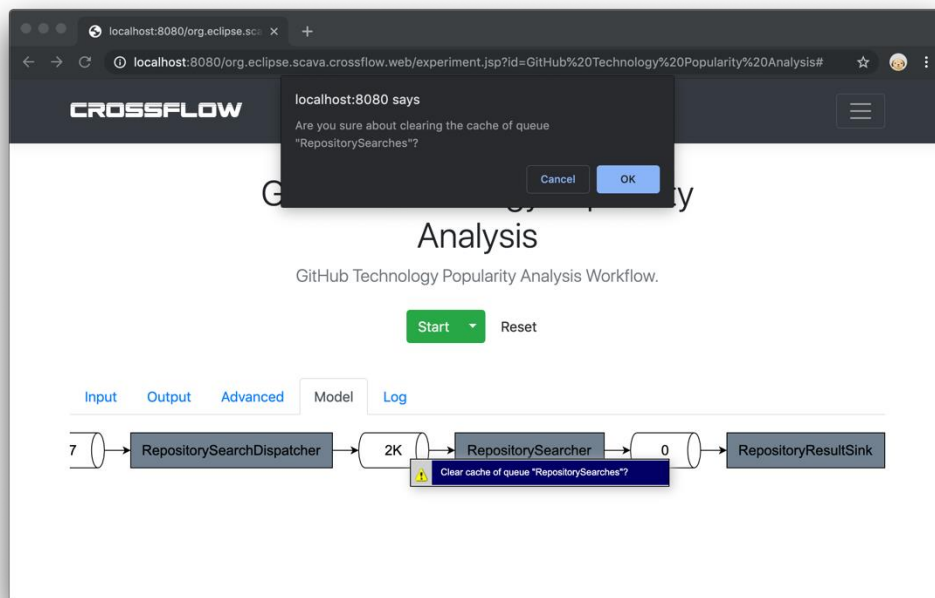


Figure 31: Running example workflow experiment page in Crossflow web application after halted workflow execution and before clearing individual cache confirmation.

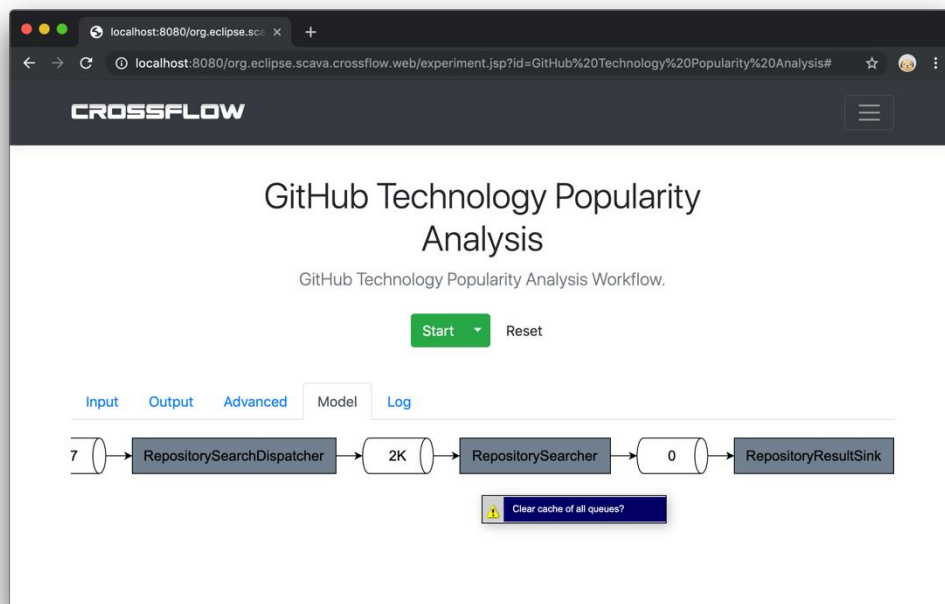


Figure 32: Running example workflow experiment page in Crossflow web application after halted workflow execution and before clearing entire cache.

5.3 CROSSFLOW WORKER EXECUTION (CLI)

This subsection describes steps involved during worker execution by the use of the Crossflow Command Line Interface (CLI).

Execution. Workers can connect to Crossflow workflows by the use of the Crossflow Command Line Interface (CLI) and in particular by specifying command line parameters value such as the name of the workflow to which they want to contribute to (required). A list of possible parameter specifications, their semantics, and default values are provided in Table 2. A minimal example of executing a worker connecting to the running example is listed below:

```
$ java -jar org.eclipse.scava.crossflow.examples.techanalysis.jar -name
TechAnalysisWorkflow -instance TechAnalysisWorkflow
```

Parameter	Semantics	Default
name	The name of the workflow	
master	IP of the master	localhost
port	Port of the master	61616
stomp	Port to use for STOMP based messages	61613
ws	Port to use for WS based messages	61614
activeMqConfig	Location of ActiveMQ configuration file	
instance	The instance of the master (to contribute to)	
mode	Must be master_bare, master or worker	Master
createBroker	Whether this workflow creates a broker or not	true
Parallelization	The parallelization of the workflow (for non-singleton tasks), defaults to 1	1
cacheEnabled	Whether this workflow caches intermediary results of not	False
deleteCache	Before starting this workflow, delete the contents of the cache by queue name (use empty string to delete entire cache)	false
inputDirectory	The input directory of the workflow	in ⁶
outputDirectory	The output directory of the workflow	out ⁷
disableTermination	Flag to disable termination when queues are empty	True

Table 2: Crossflow CLI parameters, semantics, and default values.

⁶ Relative to experiment/in from the root of the user workflow project.

⁷ Relative to experiment/out from the root of the user workflow project.

6. INSTALLATION GUIDES

This section will provide installation guides for both the graphical and textual workflow editors as well as the deployment, monitoring, and execution-management web-application presented in this document. The guides assume that the user has installed the latest version of Eclipse (Photon) Modeling Tools distribution⁸ and has Java 8 installed on their machine.

6.1 CROSSFLOW GRAPHICAL EDITOR INSTALLATION AND LAUNCH

This subsection describes the installation and launch of the Crossflow graphical editor. The use of said editor for the creation of graphical workflow specifications is described in Section 3.2.

Installation. In order to setup the Crossflow graphical editor, the following software needs to be added to the Eclipse installation, using the “Install New Software” option in the “Help” menu:

- GMF Tooling (repository: <http://download.eclipse.org/modeling/gmp/gmf-tooling/updates/releases/>)

The next step is to import the Crossflow graphical editor project from the SCAVA Crossminer repository⁹. After cloning this Git repository to the local machine, the projects can be imported into the Eclipse workspace by using the “File > Import > Git > Projects from Git > Existing local repository”. After selecting the local clone, use “Import existing Eclipse projects” and then in the next page select the following projects to be imported, before clicking “Finish”:

- org.eclipse.scava.crossflow.language(.*)

Launch. Once these tools and projects are installed, running a new Eclipse instance from the “Run” menu, selecting “Run Configurations” and creating a new “Eclipse Application” will enable the editor.

In this new Eclipse instance, first create a new Plug-in Project and then a new graphical Crossflow model by right-clicking on the created project and selecting “File > New > Other > Crossflow Diagram”. This will create a new file with the extension `.crossflow_diagram` in the Crossflow Diagram Editing editor ready to be populated with elements of the Crossflow language.

6.2 CROSSFLOW TEXTUAL EDITOR INSTALLATION AND LAUNCH

This subsection describes the installation and launch of the Crossflow textual editor. The use of said editor for the creation of textual workflow specifications is described in Section 4.2.

Installation. The Crossflow textual editor is installed as follows. First, make sure required Xtext Eclipse plugins are installed; to install Xtext in Eclipse, (i) select “Help > Install New Software”, (ii) add the Xtext update site¹⁰, (iii) select the Eclipse plugin named “Xtext Redistributable”, (iv) click the “Next >” button, and (v) follow the instructions to finalize the plugin installation.

⁸ <https://www.eclipse.org/downloads/eclipse-packages/>

⁹ <https://github.com/crossminer/scava/tree/crossflow/crossflow>

¹⁰ Xtext Eclipse update site: <http://download.eclipse.org/modeling/tmf/xtext/updates/composite/releases/>

Secondly, the Github repository <https://github.com/crossminer/scava/tree/crossflow/crossflow> is cloned to a particular location on a user's local machine. Third, the following set of projects are imported using the "Existing Projects into Workspace" wizard having selected "Search for nested projects":

- org.eclipse.scava.crossflow.language(.*)
- org.eclipse.scava.crossflow.language.xtext(.*)

Launch. Once these tools and projects are installed, running a new Eclipse instance from the "Run" menu, selecting "Run Configurations" and creating a new "Eclipse Application" will enable the editor.

In this new Eclipse instance, first create a new Plug-in Project and then a new textual Crossflow model by right-clicking on the created project and selecting "File > New > Other > Crossflow Model" and confirming the conversion of the selected parent project to an Xtext if requested. This will create a new file with the extension `.crossflow_model` and display it in the `Crossflow Editor` ready to be populated with elements of the Crossflow language.

6.3 CROSSFLOW WEB APPLICATION INSTALLATION AND LAUNCH

Installation. The Crossflow web application is available as a Docker container image hosted on Docker Hub¹¹. To deploy the said image, make sure the target machine has a running Docker installation and provides access to the Docker CLI either by an application such as Kitematic or the target machine operating system (native) console allowing to issue Docker commands. To install Docker on the target machine, follow the instructions provided by the official Docker documentation¹².

Launch. The Docker Crossflow web application container can be deployed from a Docker CLI-capable console by issuing the following command exposing several parameters, such as `-p` for port mappings between the target machine and the Docker container, allowing individual configurations¹³:

```
$ docker run -it --rm -d --name crossflow \  
  -p 80:8080 \  
  -p 61616:61616 \  
  -p 61614:61614 \  
  -p 5672:5672 \  
  -p 61613:61613 \  
  -p 1883:1883 \  
  -p 8161:8161 \  
  -p 1099:1099 \  
  crossminer/crossflow:latest
```

Running the above command in a Docker CLI-capable console will first retrieve the Docker Crossflow web application, i.e. similarly to issuing the command `docker pull`

¹¹ The official Crossflow Docker Hub repository is publicly available at <https://hub.docker.com/r/crossminer/crossflow>.

¹² The official Docker documentation provides details on how to install Docker on several different platforms and can be found online at <https://docs.docker.com>.

¹³ The semantics of Docker run parameters are described by the output produced by issuing the console command `docker run -help`

crossminer/Crossflow:latest, and then launch the obtained image with respect to the specified configuration parameters. In more detail, it follows the steps defined in the respective Dockerfile¹⁴ that includes (i) obtaining copies of Apache Tomcat and Apache ActiveMQ, (ii) configuring both Apache Tomcat and Apache ActiveMQ by copying predefined XML-based configuration files to their respective application configuration directories, (iii) copying the Crossflow web application archive file to the Apache Tomcat web application deployment directory, and (iv) launching both Apache ActiveMQ and Apache Tomcat. Finally, the Crossflow web application (cf. Figure 33) can be accessed by pointing a web browser instance to specified port of the target machine followed by `/org.eclipse.scava.crossflow.web/` or, in case no changes have been made to the above Docker run command, to <http://localhost/org.eclipse.scava.crossflow.web/>.

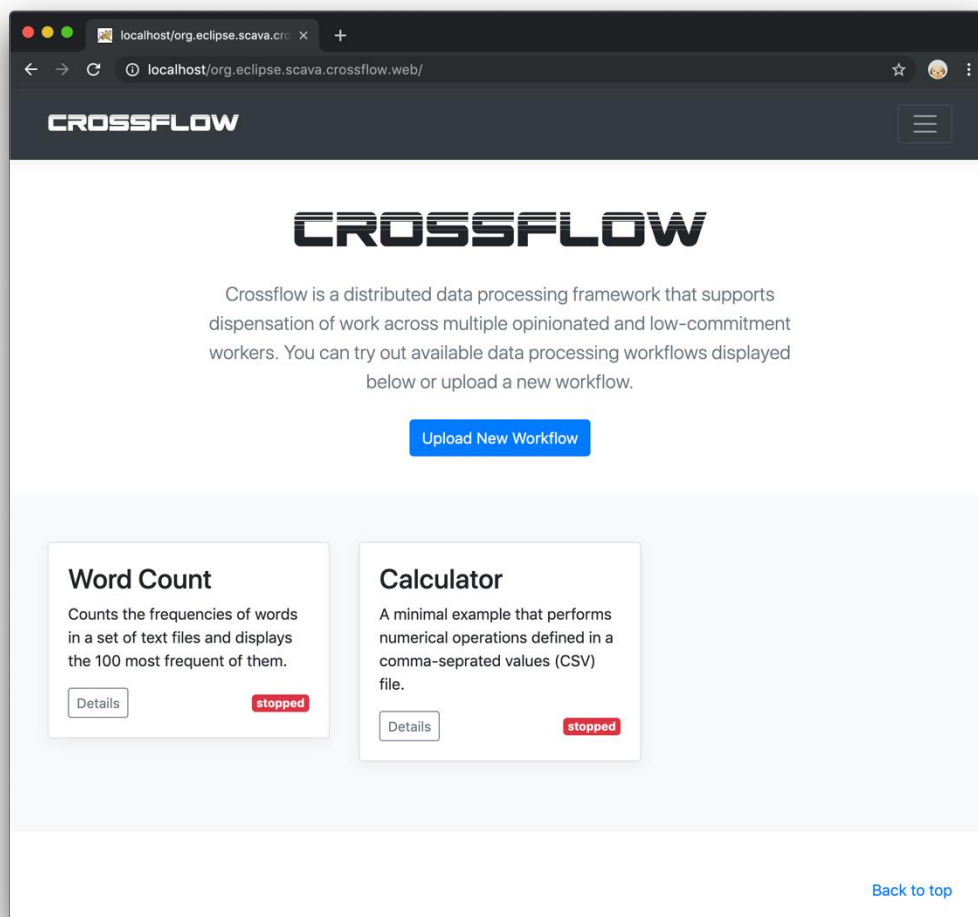


Figure 33: Docker Crossflow web application start page.

¹⁴ The Crossflow web application Dockerfile is part of the `org.eclipse.scava.crossflow.web.docker` project and publicly available at <https://raw.githubusercontent.com/crossminer/scava/crossflow/crossflow/org.eclipse.scava.crossflow.web.docker/Dockerfile>

7. CONCLUSION AND FUTURE WORK

This document presented the progress within the Crossminer task associated with the implementation of tools supporting the workflow modeling process. Workflows allow for the definition of bespoke analysis algorithms in a high-level domain-specific language and require the creation of appropriate tools to create, manipulate, generate, monitor, and manage them and their execution. The reported progress in this regard includes the implementation of a graphical and textual editor supporting the modeling of Crossflow workflows as well as the implementation of a web browser-based application for the deployment, monitoring, and management of workflows.

Future work in this regard includes the adaptation of the Crossflow metamodel, and thus also both graphical as well as textual language, for the definition of additional concepts that may arise from unforeseen workflow specifications. Moreover, the development of transformations between the graphical language as well as the textual language may be provided to enable seamless editing of both textual as well as graphical models. Further, the packaging-step performed during the generation of executables for the deployment of Crossflow workflows will be extended to automate the packaging of referenced projects (cf. “Dependencies” paragraph in Section 5.1).

Tables on the final status of the user and technology requirements related to WP5 of the Crossminer project can be found below, containing the requirement name and description, its overall priority as well as its final status.

TABLE ON FINAL STATUS OF USE-CASE PARTNER REQUIREMENTS FOR WP5

ID	Description	Overall Priority	Status
U122	Provides a mechanism to define a retrieval, cleaning and analysis process based on reusable components	SHOULD	Supported
U123	Provides a mechanism to play individual parts of the retrieval and analysis process	SHOULD	Supported
U124	Provides a mechanism to easily analyse new data sources and define new measures	SHALL	Supported
U125	Able to use data from all existing data collectors	SHOULD	Supported
U126	Provides a mechanism to analyse and visualise data in an external tool (e.g. R, Tableau, .. Excel)	SHOULD	Supported
U127	Provides a library of reusable components	SHOULD	Supported
U128	Provides a list of available data sets when building a new workflow	SHOULD	Unsupported
U129	Provides R as a computing engine for analyses	SHOULD	Supported
U130	Able to identify if the developer is not using the most recent version of a library and provide notification	SHALL	Supported
U131	Provides a means to execute a workflow across data sets	SHOULD	Supported
U132	Provides a means to execute a workflow across forges	SHOULD	Supported
U133	Able to support different execution priorities	MAY	Unsupported
U134	Allows compositions of Crossminer and external results to support decisions	SHALL	Supported
U135	Able to define specific formatting for the results	SHALL	Supported
U136	Able to process data sets from GitHub and StackOverflow	SHALL	Supported

TABLE ON FINAL STATUS OF TECHNOLOGY REQUIREMENTS FOR WP5

ID	Description	Overall Priority	Status
D47	The framework shall provide built-in support for network/API error recovery	SHALL	Supported
D48	The framework shall provide built-in support for data caching	SHALL	Supported
D49	The framework shall provide support for graphical editors for specifying knowledge extraction workflows	SHALL	Supported
D50	The framework shall provide a Java API for specifying knowledge extraction workflows	SHALL	Supported
D51	The graphical workflow editors should provide support for auto-completion, navigation and refactoring	SHOULD	Supported
D52	The framework shall provide parallel workflow execution capabilities	SHALL	Supported
D53	The framework shall provide distributed workflow execution capabilities	SHALL	Supported
D54	The framework shall provide debugging facilities	SHALL	Supported
D55	The framework shall provide workflow execution monitoring facilities	SHALL	Supported
D56	Workflow execution facilities shall be architecturally consistent with the platform so that workflows can be executed as metric providers	SHALL	Supported
D57	Connectors shall be implemented for the APIs of GitHub, StackOverflow and Bugzilla	SHALL	Supported
D58	Connectors should be implemented for GHTorrent, GitHub Archive and JIRA	SHOULD	Unsupported
D59	The platform shall expose a REST API that workflow components can consume	SHALL	Supported
D60	The API of the platform should be formally specified	SHOULD	Supported
D61	Mining tools developed in WPs 2-4 and 6 should be embeddable as components in knowledge extraction workflows	SHOULD	Supported
D62	The platform shall provide facilities for running custom workflows and displaying the results in appropriate dashboards	SHALL	Supported

REFERENCES

- Eysholdt, M., & Behrens, H. (2010). Xtext: Implement your Language Faster than the Quick and Dirty Way. *Companion Proc. of OOPSLA*, (pp. 307-309).
- Kelly, S., & Tolvanen, J.-P. (2008). *Domain-specific modeling: enabling full code generation*. John Wiley & Sons.
- Kolovos, D. S., García-Domínguez, A., Rose, L. M., & Paige, R. F. (2017, 2 01). Eugenia: towards disciplined and automated development of GMF-based graphical model editors. *Software & Systems Modeling*, 16, 229-255. doi:10.1007/s10270-015-0455-3
- Kolovos, D. S., Matragkas, N., & Garcia-Dominguez, A. (2016). Towards Flexible Parsing of Structured Textual Model Representations. *Proceedings of the 2nd Workshop on Flexible Model Driven Engineering co-located with {ACM/IEEE} 19th International Conference on Model Driven Engineering Languages {\&} Systems (MoDELS 2016)*. Saint-Malo, France: ACM/IEEE.
- Kolovos, D., Neubauer, P., Barmpis, K., Matragkas, N., & Paige, R. (2019). Crossflow: A Framework for Distributed Mining of Software Repositories. *Proceedings of the 16th International Conference on Mining Software Repositories (MSR)*. Montreal, QC, Canada: IEEE/ACM.
- Mohagheghi, P., Fernandez, M., Martell, J., Fritzsche, M., & Gilani, W. (2009). MDE Adoption in Industry: Challenges and Success Criteria. *Lecture Notes in Computer Science*, 5421, pp. 54-59. doi:10.1007/978-3-642-01648-6_6
- Rumbaugh, J., Jacobson, I., & Booch, G. (2004). *Unified Modeling Language Reference Manual*. Pearson Higher Education.
- Steinberg, D., Budinsky, F., Paternostro, M., & Merks, E. (2009). *EMF: Eclipse Modeling Framework 2.0* (2nd ed.). Addison-Wesley Professional.
- Viyović, V., Maksimović, M., & Perisić, B. (2014). Sirius: A rapid development of DSM graphical editor. *Proceedings of INES*, (pp. 233-238).
- Zolotas, A., & et. al. (June 2018). Towards Automatic Generation of UML Profile Graphical Editors for Papyrus. *Proc. 14th European Conference on Modeling Foundations and Applications, ECMFA 2018*. Toulouse, France.
- Zschaler, S., Kolovos, D. S., Drivalos, N., Paige, R. F., & Rashid, A. (2009). Domain-Specific Metamodelling Languages for Software Language Engineering. *Software Language Engineering, Second International Conference, SLE 2009, Denver, CO, USA, October 5-6, 2009, Revised Selected Papers*, (pp. 334-353). doi:10.1007/978-3-642-12107-4_23

APPENDIX A: CROSSFLOW GRAPHICAL MODELING LANGUAGE DEFINITION (CROSSFLOW.EMF)

crossflow.emf

```

1 @gmf
2 @namespace(uri="org.eclipse.scava.crossflow", prefix="cf")
3 package crossflow;
4
5 @gmf.diagram(onefile="true")
6 class Workflow {
7   attr String name;
8   attr String ~package;
9   val Stream[*] streams;
10  val Task[*] tasks;
11  val Type[*] types;
12  val Field[*] parameters;
13  val Language[*] languages;
14 }
15
16 @gmf.node(figure="ellipse", label="name", label.icon="false")
17 abstract class Stream {
18   attr String name;
19 }
20 @gmf.link
21 ref Type type;
22
23 @gmf.link(target.decoration="filledclosedarrow")
24 ref Task[*]#input inputOf;
25
26 ref Task[*]#output outputOf;
27 }
28
29 class Topic extends Stream {
30 }
31
32 class Queue extends Stream {
33 }
34
35 @gmf.node(label="name", label.icon="false")
36 class Task {
37   attr String name;
38
39   ref Stream[*]#inputOf input;
40
41   @gmf.link(target.decoration="filledclosedarrow")
42   ref Stream[*]#outputOf output;
43
44   attr Boolean masterOnly = "false";
45   attr Boolean parallel;
46   attr Boolean cached;
47   attr Boolean multipleOutputs = "false";
48
49   ref Field[*] parameters;
50
51   ref Language[*] languages;
52
53   ref Type[*] configurations;
54 }
55
56 @gmf.node(label="name", label.icon="false", figure="polygon", polygon.x="0 10 11 10 0", polygon.y="0 0 2 4 4")
57 class Source extends Task {
58 }
59
60 class CsvSource extends Source {
61   attr String fileName;
62 }
63
64 @gmf.node(label="name", label.icon="false", figure="polygon", polygon.x="0 10 10 5 0", polygon.y="0 0 4 6 4")

```

Page 1

crossflow.emf

```

65 class Sink extends Task {
66 }
67
68 class CsvSink extends Sink {
69   attr String fileName;
70 }
71
72 @gmf.node(label="name")
73 class CommitmentTask extends Task {
74   attr int commitAfter = 1;
75 }
76
77 @gmf.node(label="name")
78 class OpinionatedTask extends Task {
79 }
80
81 @gmf.node(label="name", figure="rectangle")
82 class Type {
83   attr String name;
84   attr String impl;
85   attr boolean[1] isMany;
86
87   @gmf.link
88   ref Type[*] extending;
89
90   @gmf.compartment(layout="list", collapsible="false")
91   val Field[*] fields;
92 }
93
94 @gmf.node(label="name,type", figure="rectangle", label.pattern="{0}:{1}", label.icon="false")
95 class Field {
96   attr String name;
97   attr String type = "String";
98   attr boolean many = false;
99 }
100
101 @gmf.node(label="name", figure="rectangle")
102 class Language {
103   attr String name;
104   attr String ~package;
105   attr String outputFolder;
106   attr String genOutputFolder;
107
108   @gmf.compartment(layout="list", collapsible="false")
109   val Parameter[*] parameters;
110 }
111
112 @gmf.node(label="name,value", figure="rectangle", label.pattern="{0}:{1}", label.icon="false")
113 class Parameter {
114   attr String name;
115   attr String value;
116 }

```

APPENDIX B: CROSSFLOW TEXTUAL WORKFLOW MODELING LANGUAGE DEFINITION (CROSSFLOW.XTEXT)

Crossflow.xtext

```

1 /*****
2 * Copyright (c) 2019 The University of York.
3 * This program and the accompanying materials
4 * are made available under the terms of the Eclipse Public License 2.0
5 * which is available at https://www.eclipse.org/legal/epl-2.0/
6 *
7 * Contributor(s):
8 *   Patrick Neubauer - initial API and implementation
9 *****/
10 grammar org.eclipse.scava.crossflow.language.xtext.Crossflow with org.eclipse.xtext.common.Terminals
11
12 import "org.eclipse.scava.crossflow"
13 import "http://www.eclipse.org/emf/2002/Ecore" as ecore
14
15 Workflow returns Workflow:
16   {Workflow}
17   'Workflow'
18   name=EString
19   '{
20     ('package' package=EString)?
21     ('streams' '{ streams+=Stream (streams+=Stream)* '})?
22     ('tasks' '{ tasks+=Task (tasks+=Task)* '})?
23     ('types' '{ types+=Type (types+=Type)* '})?
24     ('parameters' '{ parameters+=Field (parameters+=Field)* '})?
25     ('languages' '{ languages+=Language (languages+=Language)* '})?
26   }';
27
28 Stream returns Stream:
29   Topic | Queue;
30
31 Task returns Task:
32   Task_Impl | Source_Impl | CsvSource | Sink_Impl | CsvSink | CommitmentTask | OpinionatedTask;
33
34 EString returns ecore::EString:
35   STRING | ID;
36
37 Task_Impl returns Task:
38   {Task}
39   (masterOnly?='masterOnly')?
40   (parallel?='parallel')?
41   (cached?='cached')?
42   (multipleOutputs?='multipleOutputs')?
43   'Task'
44   name=EString
45   '{
46     ('input' '{ input+=[Stream|EString] (input+=[Stream|EString])* '})?
47     ('output' '{ output+=[Stream|EString] (output+=[Stream|EString])* '})?
48     ('parameters' '{ parameters+=[Field|EString] (parameters+=[Field|EString])* '})?
49     ('languages' '{ languages+=[Language|EString] (languages+=[Language|EString])* '})?
50     ('configurations' '{ configurations+=[Type|EString] (configurations+=[Type|EString])* '})?
51   }';
52
53 Type returns Type:
54   (isMany?='isMany')?
55   'Type'
56   name=EString
57   '{
58     ('impl' impl=EString)?
59     ('extending' '{ extending+=[Type|EString] (extending+=[Type|EString])* '})?
60     ('fields' '{ fields+=Field (fields+=Field)* '})?
61   }';
62
63 Field returns Field:
64   {Field}

```

Page 1

Crossflow.xtext

```

65 (many?='many')?
66 'Field'
67 name=EString
68 '{'
69 ('type' type=EString)?
70 '}'
71
72 Language returns Language:
73 {Language}
74 'Language'
75 name=EString
76 '{'
77 ('package' package=EString)?
78 ('outputFolder' outputFolder=EString)?
79 ('genOutputFolder' genOutputFolder=EString)?
80 ('parameters' '{' parameters+=Parameter (parameters+=Parameter)* '}' )?
81 '}'
82
83 Topic returns Topic:
84 {Topic}
85 'Topic'
86 name=EString
87 '{'
88 ('type' type=[Type|EString])?
89 ('inputOf' '(' inputOf+=[Task|EString] (inputOf+=[Task|EString])* ')' )?
90 ('outputOf' '(' outputOf+=[Task|EString] (outputOf+=[Task|EString])* ')' )?
91 '}'
92
93 Queue returns Queue:
94 {Queue}
95 'Queue'
96 name=EString
97 '{'
98 ('type' type=[Type|EString])?
99 ('inputOf' '(' inputOf+=[Task|EString] (inputOf+=[Task|EString])* ')' )?
100 ('outputOf' '(' outputOf+=[Task|EString] (outputOf+=[Task|EString])* ')' )?
101 '}'
102
103 EBooleanObject returns ecore::EBooleanObject:
104 'true' | 'false';
105
106 Source_Impl returns Source:
107 {Source}
108 (masterOnly?='masterOnly')?
109 (parallel?='parallel')?
110 (cached?='cached')?
111 (multipleOutputs?='multipleOutputs')?
112 'Source'
113 name=EString
114 '{'
115 ('input' '(' input+=[Stream|EString] (input+=[Stream|EString])* ')' )?
116 ('output' '(' output+=[Stream|EString] (output+=[Stream|EString])* ')' )?
117 ('parameters' '(' parameters+=[Field|EString] (parameters+=[Field|EString])* ')' )?
118 ('languages' '(' languages+=[Language|EString] (languages+=[Language|EString])* ')' )?
119 ('configurations' '(' configurations+=[Type|EString] (configurations+=[Type|EString])* ')' )?
120 '}'
121
122 CsvSource returns CsvSource:
123 {CsvSource}
124 (masterOnly?='masterOnly')?
125 (parallel?='parallel')?
126 (cached?='cached')?
127 (multipleOutputs?='multipleOutputs')?
128 'CsvSource'

```

Page 2

Crossflow.xtext

```

129 name=EString
130 '{
131   ('fileName' fileName=EString)?
132   ('input' '(' input+=[Stream|EString] (input+=[Stream|EString])* ')' )?
133   ('output' '(' output+=[Stream|EString] (output+=[Stream|EString])* ')' )?
134   ('parameters' '(' parameters+=[Field|EString] (parameters+=[Field|EString])* ')' )?
135   ('languages' '(' languages+=[Language|EString] (languages+=[Language|EString])* ')' )?
136   ('configurations' '(' configurations+=[Type|EString] (configurations+=[Type|EString])* ')' )?
137   '}';
138
139 Sink_Impl returns Sink:
140 {Sink}
141 (masterOnly?='masterOnly')?
142 (parallel?='parallel')?
143 (cached?='cached')?
144 (multipleOutputs?='multipleOutputs')?
145 'Sink'
146 name=EString
147 '{
148   ('input' '(' input+=[Stream|EString] (input+=[Stream|EString])* ')' )?
149   ('output' '(' output+=[Stream|EString] (output+=[Stream|EString])* ')' )?
150   ('parameters' '(' parameters+=[Field|EString] (parameters+=[Field|EString])* ')' )?
151   ('languages' '(' languages+=[Language|EString] (languages+=[Language|EString])* ')' )?
152   ('configurations' '(' configurations+=[Type|EString] (configurations+=[Type|EString])* ')' )?
153   '}';
154
155 CsvSink returns CsvSink:
156 {CsvSink}
157 (masterOnly?='masterOnly')?
158 (parallel?='parallel')?
159 (cached?='cached')?
160 (multipleOutputs?='multipleOutputs')?
161 'CsvSink'
162 name=EString
163 '{
164   ('fileName' fileName=EString)?
165   ('input' '(' input+=[Stream|EString] (input+=[Stream|EString])* ')' )?
166   ('output' '(' output+=[Stream|EString] (output+=[Stream|EString])* ')' )?
167   ('parameters' '(' parameters+=[Field|EString] (parameters+=[Field|EString])* ')' )?
168   ('languages' '(' languages+=[Language|EString] (languages+=[Language|EString])* ')' )?
169   ('configurations' '(' configurations+=[Type|EString] (configurations+=[Type|EString])* ')' )?
170   '}';
171
172 CommitmentTask returns CommitmentTask:
173 {CommitmentTask}
174 (masterOnly?='masterOnly')?
175 (parallel?='parallel')?
176 (cached?='cached')?
177 (multipleOutputs?='multipleOutputs')?
178 'CommitmentTask'
179 name=EString
180 '{
181   ('commitAfter' commitAfter=EInt)?
182   ('input' '(' input+=[Stream|EString] (input+=[Stream|EString])* ')' )?
183   ('output' '(' output+=[Stream|EString] (output+=[Stream|EString])* ')' )?
184   ('parameters' '(' parameters+=[Field|EString] (parameters+=[Field|EString])* ')' )?
185   ('languages' '(' languages+=[Language|EString] (languages+=[Language|EString])* ')' )?
186   ('configurations' '(' configurations+=[Type|EString] (configurations+=[Type|EString])* ')' )?
187   '}';
188
189 OpinionatedTask returns OpinionatedTask:
190 {OpinionatedTask}
191 (masterOnly?='masterOnly')?
192 (parallel?='parallel')?

```

Crossflow.xtext

```

193 (cached?='cached')?
194 (multipleOutputs?='multipleOutputs')?
195 'OpinionatedTask'
196 name=EString
197 '{
198   ('input' '(' input+=[Stream|EString] (input+=[Stream|EString])* ')' )?
199   ('output' '(' output+=[Stream|EString] (output+=[Stream|EString])* ')' )?
200   ('parameters' '(' parameters+=[Field|EString] (parameters+=[Field|EString])* ')' )?
201   ('languages' '(' languages+=[Language|EString] (languages+=[Language|EString])* ')' )?
202   ('configurations' '(' configurations+=[Type|EString] (configurations+=[Type|EString])* ')' )?
203   '}';
204
205 EInt returns ecore::EInt:
206   '-'? INT;
207
208 EBoolean returns ecore::EBoolean:
209   'true' | 'false';
210
211 Parameter returns Parameter:
212   {Parameter}
213   'Parameter'
214   name=EString
215   '{
216     ('value' value=EString)?
217     '}';
218

```