Metrics Framework Infrastructure Specification

Abstract

This document describes the features and functionality of the Metrics Framework Infrastructure. The main procedures detail in this specification are associated with the registry and monitoring metrics, and the collection of data from metrics executions.

Status of This Document

This is an editor's draft of a document for the Development and Analysis of ELIXIR Metrics Framework. Distribution of this document is unlimited.

Table of Contents

1. Introduction

Measuring the impact of bioinformatics resources is of paramount importance. Metrics encourage attainment of certain standards of quality of resources. We propose the development of a metrics framework to provide an infrastructure capable of collect/register different metrics definitions and their functionalities, process them and provide a user interface to access these results. Data collected from each metric can be used as an indicator of quality that measure the impact of specific bioinformatic resource (software, biological databases, knowledge diffusion, computational performance, etc.) which suggesting a certain standard of quality.

This framework consists of three elements which work together for carrying out the complete registry and monitoring processes. The first element Component, define the metric schema and functionality; the second component Components Registry, register the component metadata into a registry to make available for use; The third element Data and Monitoring Repository, install and execute components and collect data from the component execution. Data generated from each component execution will be able for their visualization

This document provides the Metrics Framework specification, in addition to definition, description and functionality of each element of this infrastructure. Also, this document contains the system analysis and design.

2. Basic Grammar and Conventions

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Metrics Framework Definition

3.1. Metrics Framework Elements

The metrics framework design follows a modular structure, consisting of three autonomous elements: Component, Components Registry and, Data and Monitoring Repository. These elements interact with each other and besides this interaction, a user can also interact independently with each element using an user interface adjust to each setting.

3.1.1. Component

A component is the basic unit defined into the Metrics Framework. Each component are defined through two differentiated parts:

- ***Metric standard definition***, following a common schema. This schema is described in a metadata file that defines a set of descriptors used in deploying a component. Examples of descriptors are: name, dependencies, frequency, resource, output, repository, etc.
- ***Metric functionality***, following a common structure. Code, documentation, guidelines, and examples can define this structure.

Due to the distributed approach of this framework, the components MUST be stored in a Source Code Manager (e.g. GitHub [GitHub]) that allows an easy and quick access to each component.

The basic component structure SHOULD follow the structure shown in Figure 1.

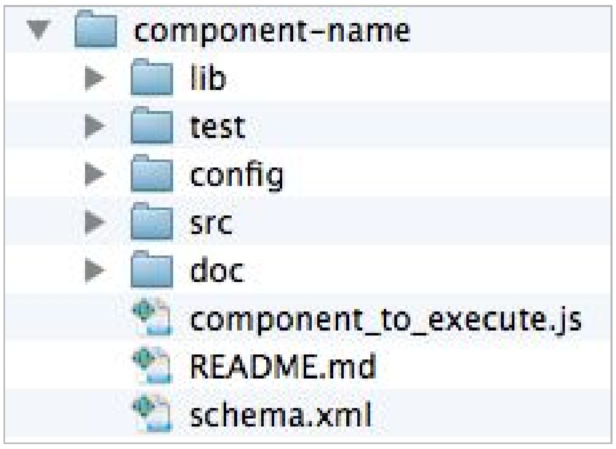

```
component-name
    lib
    test
    config
    src
    doc
    component_to_execute.js
    README.md
    schema.xml
```

Figure 1. Basic component structure. The schema.xml file describes the metadata information or the Metric standard definition. Other directories and files define the metric functionality (Code, documentation, guidelines, and examples)

The file schema.xml describes the component metadata therefore this file MUST be included into the component structure. This metadata contains essential information that will be used to register, install and execute a component, and collect data from this component. The Figure 2 shows the schema architecture, and the meanings of each parameter are given in the Table 1.

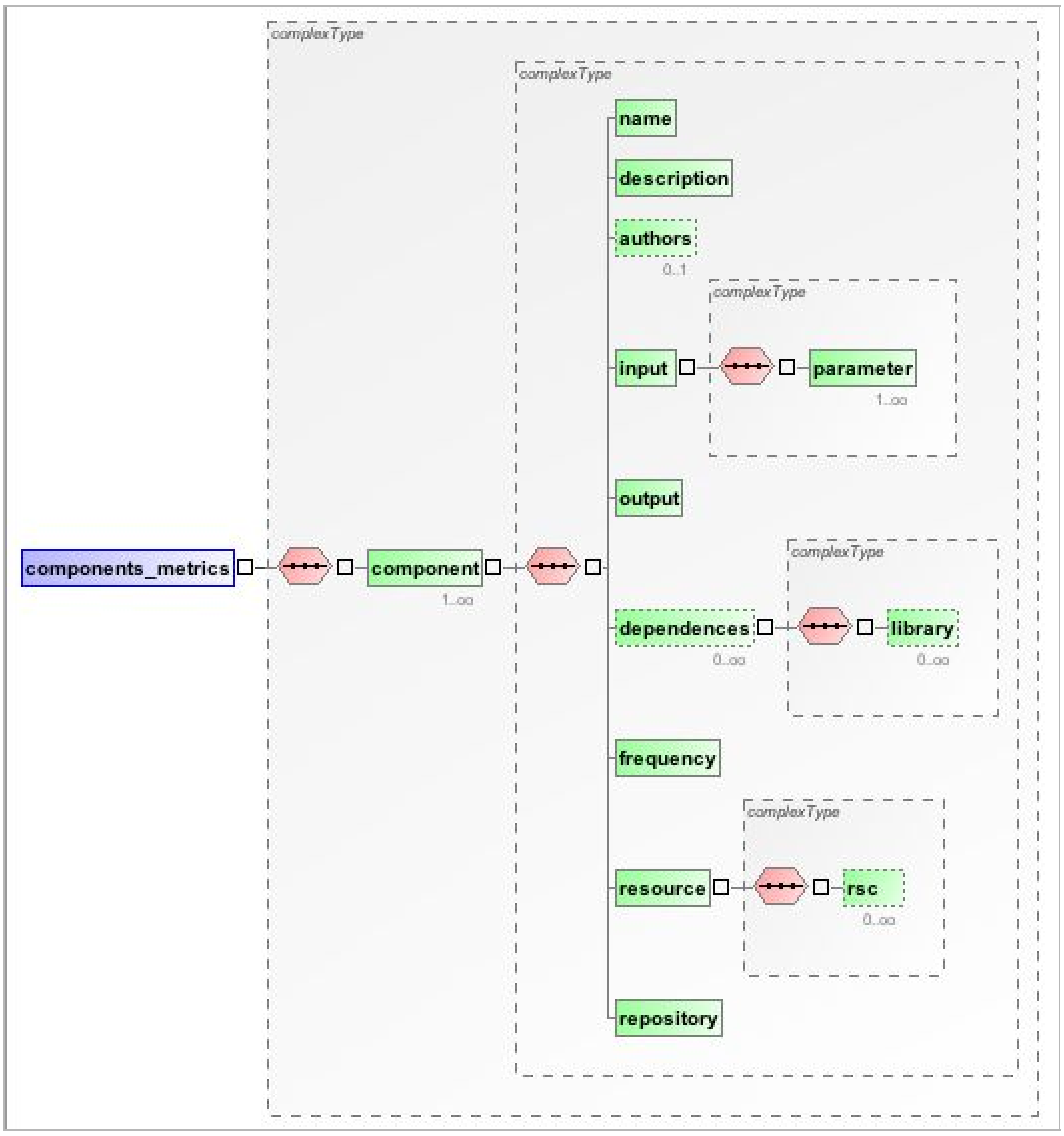


Figure 2. Graphical component schema. This graphic shows the component schema hierarchy.

Parameters	Description	Cardinality	Type
Name	Component name	1	String
Description	Short component explanation	1	String
Authors	Authors names separated by commas	0..1	String
Input	Input parameters list used in the component execution.	1..*	String
Output	Type execution result: number, string or list.	1	String
Dependencies	Libraries list necessaries for this component.	0..*	String List
Frequency	Time interval to execute a component: monthly, weekly, daily.	1	String
Resources	Resource list representation: database, software, training, computational.	1..*	String List
Repository	Source Manger Code URL where the component is stored.	1	String

Table 1. Component schema. This table describes in detail, the parameters which defines the components metadata.

3.1.2. Components Registry

The Components Registry contains metadata information for each component that has been registered. This information MUST be used when a specific component is installed.

The metadata information is described in a common schema consisting of component metadata (described in section 3.1.1) and additional descriptors, which are essentials for registering a component. When a component is registered MUST contain this schema definition in its metadata file. This schema configuration is shown in Figure 3, and theirs parameters definition are described in Table 2.

Parameters	Description	Cardinality	Type
Name	Component name	1	String
Description	Short component explanation	1	String
Authors	Authors names separated by commas	0..1	String
Input	Input parameters list used in the component execution.	1..*	String
Output	Type execution result: number, string or list.	1	String
Dependencies	Libraries list necessaries for this component.	0..*	String List
Frequency	Time interval to execute a component: monthly, weekly, daily.	1	String
Resources	Resource list representation: database, software, training, computational.	1..*	String List
Repository	Source Manger Code URL where the component is stored.	1	String
Id	Unique identifier, this is composed for the component name and one exclusive alphanumeric.	1	String
version	Component version.	1	String
lastupdated	Component last date of upgrade.	1	Date
...			

Table 2. Component Registry schema. This table describes in detail, the parameters which defines the metadata of components registered. The orange cells show additionals descriptors as compared to component metadata schema.

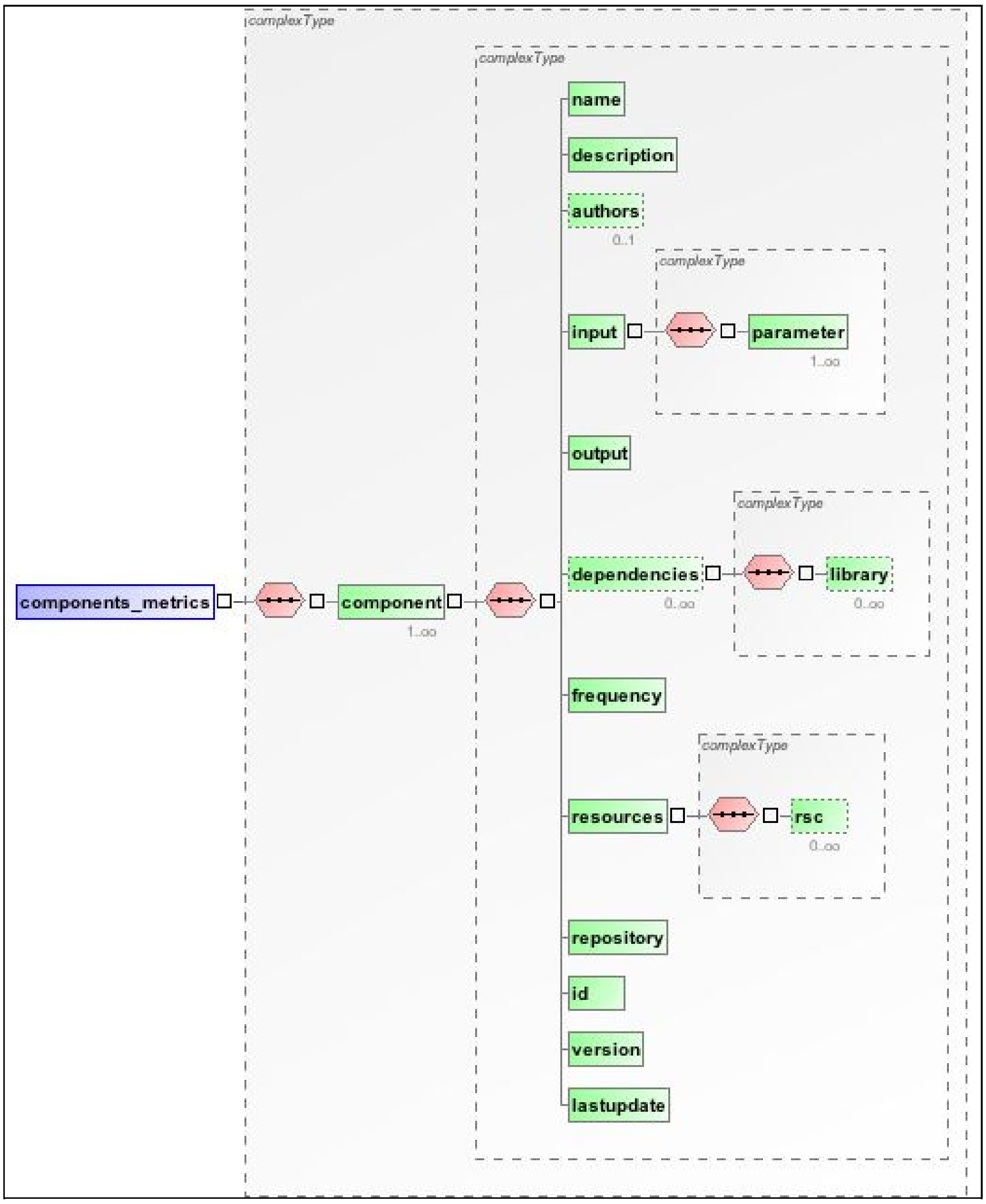


Figure 3. Graphical Component Registry schema. This graphic shows the component schema hierarchy consisting of essential component metadata and additional descriptors: id, version, last update.

3.1.3. Data and Monitoring Repository

This Repository collects data from a set of components registered into the Components Repository; also it makes available these data for different users.

As a part of its configuration, the Data and Monitoring Repository MUST use a schema from where, it extracts necessary information to install and execute a specific component.

This schema configuration is shown in Figure 4, and theirs parameters definition are described in Table 3.

Parameters	**Description**	**Cardinality**	**Type**
Id	Unique identifier, this is composed for the component name and one exclusive number.	1	String
Name	Component name.	1	String
Frequency	Time interval to execute a component: monthly, weekly, daily.	1	String
Resource	Resource representation: database, software, training, computational.	1	String
Enable	Define if the component functionality is available into the source code manager.	1	Boolean
Installed	Define if the component is installed.	1	Boolean
Input	Parameter list for executing a component.	0..*	String
Executable	File name of the executable component.	1	String
Dependencies	Libraries list necessaries for this component.	0..*	String List
Repository	Source Manger Code URL where the component is stored.	1	String

Table 3. Selected Component schema. This table describes in detail, the parameters which defines the metadata of selected components.

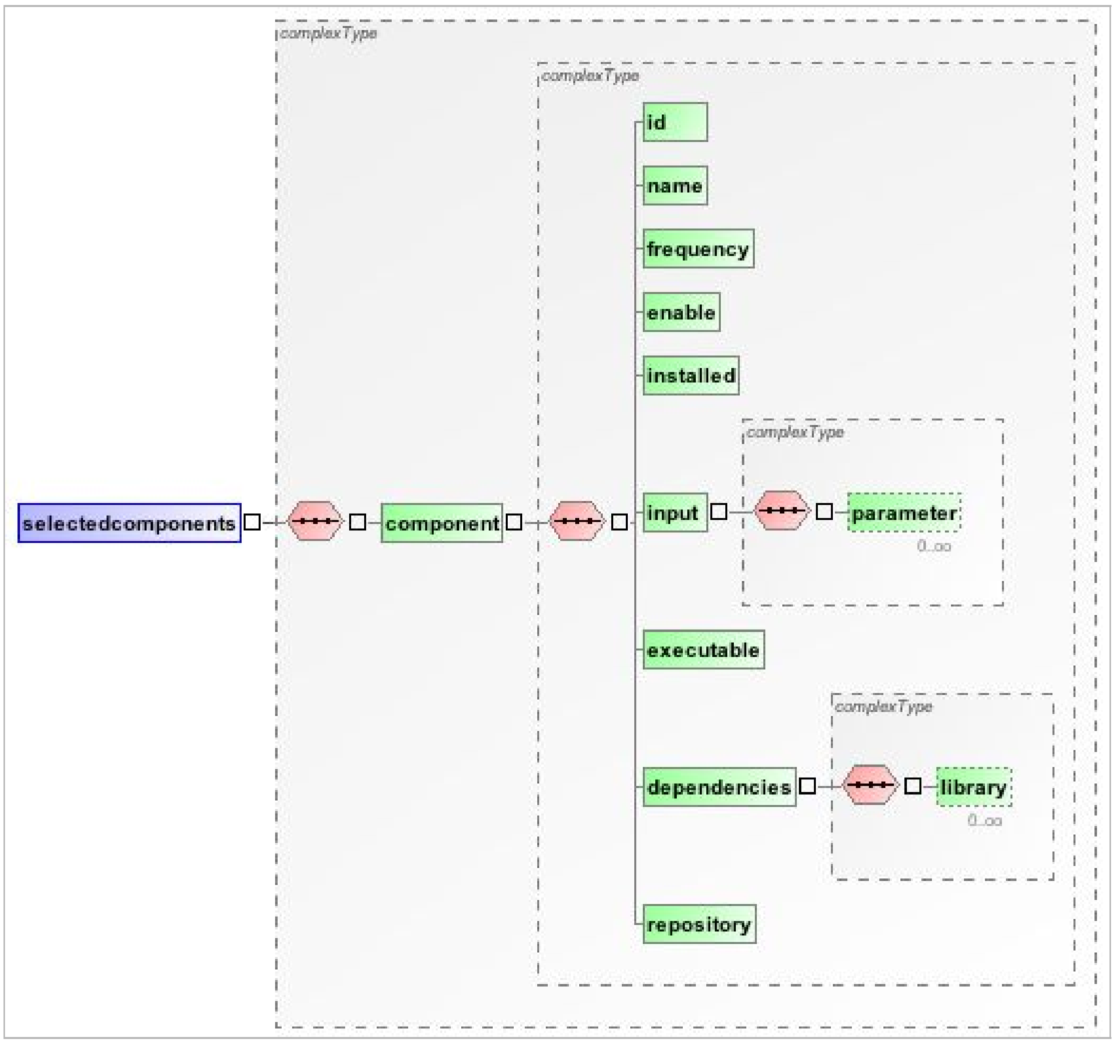
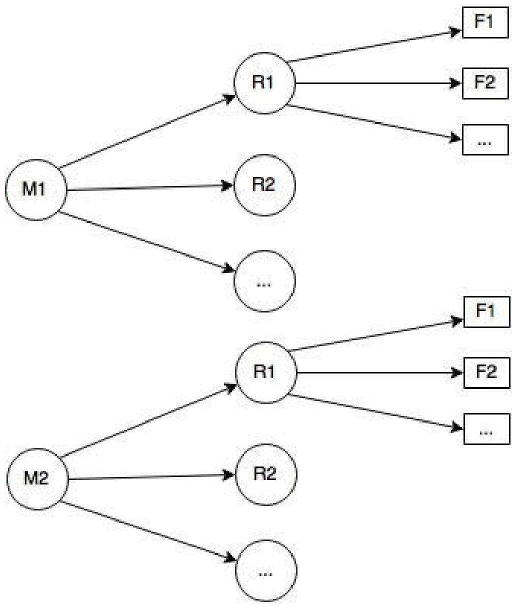


Figure 4. Selected Components schema. This graphic shows the selected components schema hierarchy.

Data from selected components' execution SHOULD collect and store in a cross-platform document-oriented database as Mongdb [MongDB]. In this document we explain the details of this configuration.

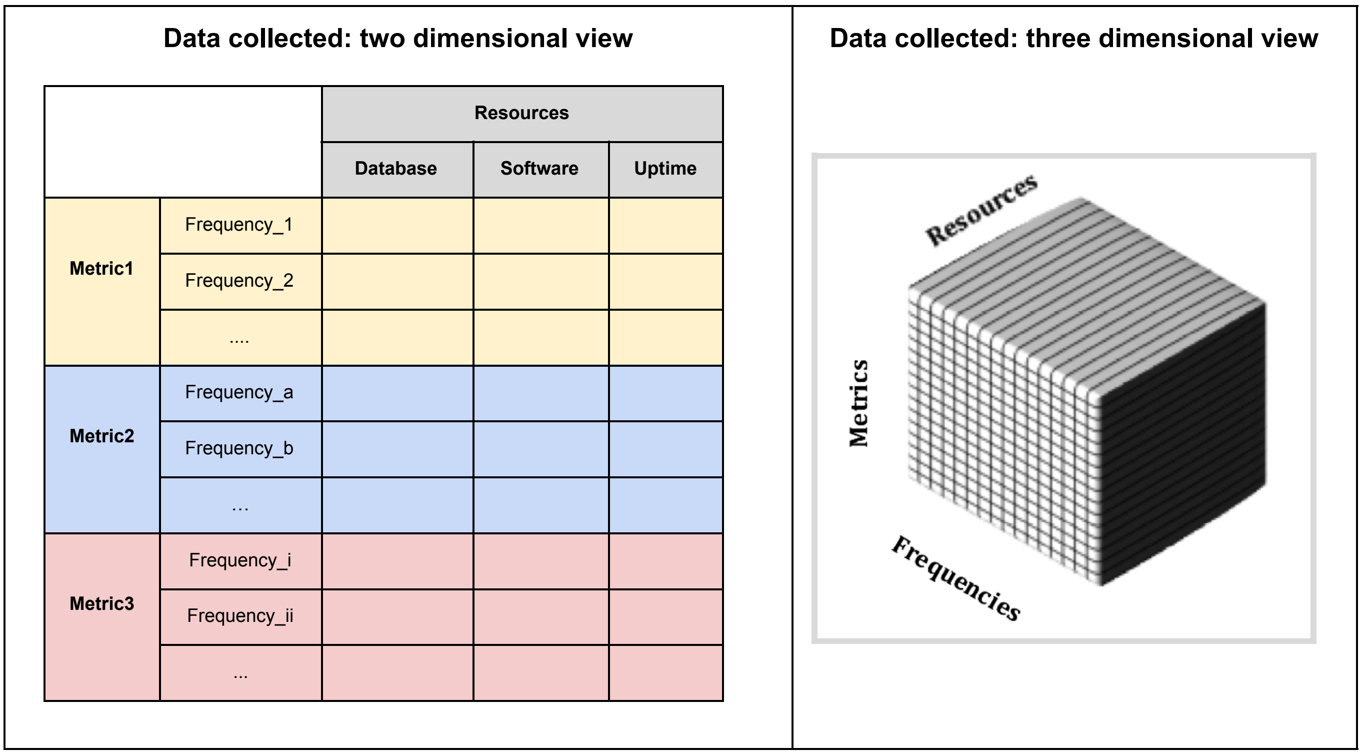
The data storage' structure is shown in the Table 4. The main approach for this configuration is organize the collected data by grouping ***resource-metric-frequencies***:



```
Resource
{
         "_id"           : assigned by Mongodb
        , "name"         : resource name
        ,"type"          : resource type (database, software, training, etc.)
        , "metric"       : metric_id association.
        ,"frequencies"   : array with the collected data in each execution and in an specific
                 frequency.
}
```

Table 4. Mongodb record schema. The collected data from component execution is stored following this structure.

Using this organization it is possible build reports as it is showing in the Figure 5. We manipulate data that we must linkage using a three dimensional format.



Data collected: two dimensional view

		Resources		
		Database	Software	Uptime
Metric1	Frequency_1			
	Frequency_2			
	....			
Metric2	Frequency_a			
	Frequency_b			
	…			
Metric3	Frequency_i			
	Frequency_ii			
	...			

Data collected: three dimensional view

Figure 5. The graphic shows a two dimensional view on the collected data but in fact three dimensions (shown in cube) are included.

Using data collected from the selected components execution, stored in the database, it is possible visualize this information. To achieve this functionality, the Data and Monitoring Repository MUST use a schema to extract necessary information to make available a list specific components which are visualized using a GUI (Graphical User Interface).

This schema for available components is shown in Figure 6, and theirs parameters definition are described in Table 5.

Parameters	Description	Cardinality	Type
Id	Unique identifier, this is composed for the component name and one exclusive number.	1	String
Name	Component name.	1	String
Description	Brief description of this component		
Detail	Describe specific technical information about one component, api, query, output.	1	Element
Frequency	Time interval to execute a component: monthly, weekly, daily.	1	String
Resource	Describe the resource representation: name and type (software, training, computational),	1	Element
Repository	Source Manger Code URL where the component is stored.	1	String

Table 5. Available Component schema. This table describes in detail, the parameters which defines the metadata of available components.

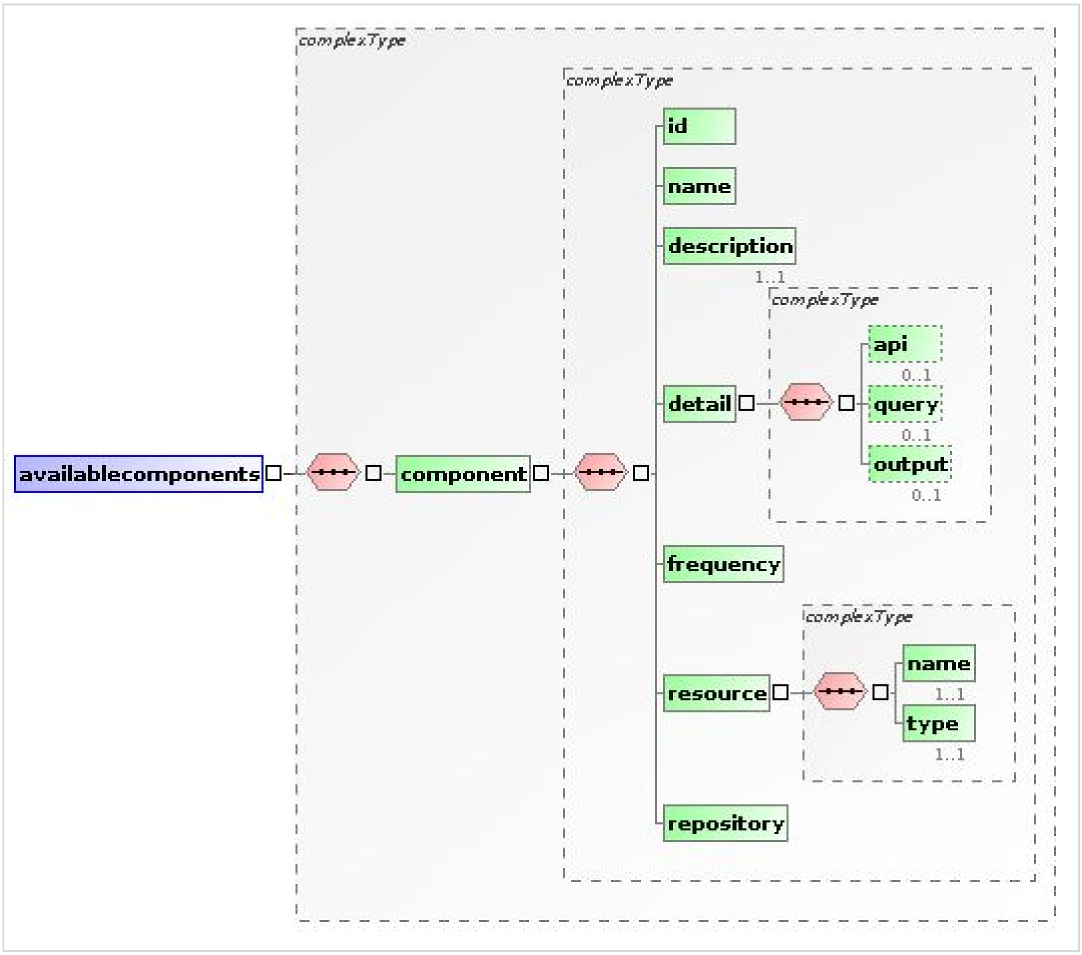


Figure 6. Available Components schema. This graphic shows the available components schema hierarchy.

3.2. Metrics Framework Functionality

3.2.1. Component Description

One component MUST describe its metadata information in a XML file, following the specification explained in the section 3.1.1. (Table 1 and the Figure 2).

The component implementation MUST be developed in NodeJS [NodeJS] technology however, the component implementation MAY define after the component registry. A component MUST include its implementation into the Source Content Manager, before its installation and execution.

3.2.2. Component Registry

The registry component follows the next actions (Figure 7):

1. New component MUST provide the metadata URL where it is stored (XML file URL).
2. The XML schema is extracted from this URL.
3. The new component XML schema is compared with the base schema defined into the Component Registry.
4. If this schema is verified, the unique component ID is generated. The final ID will be composed of the component name and unique alphanumeric string (**name-alphanumeric_string**).
5. The new component metadata definition,its ID and other default descriptors, are saved in the **registered components** XML schema.
6. Finally, the HTML schema is created, adding the new component.

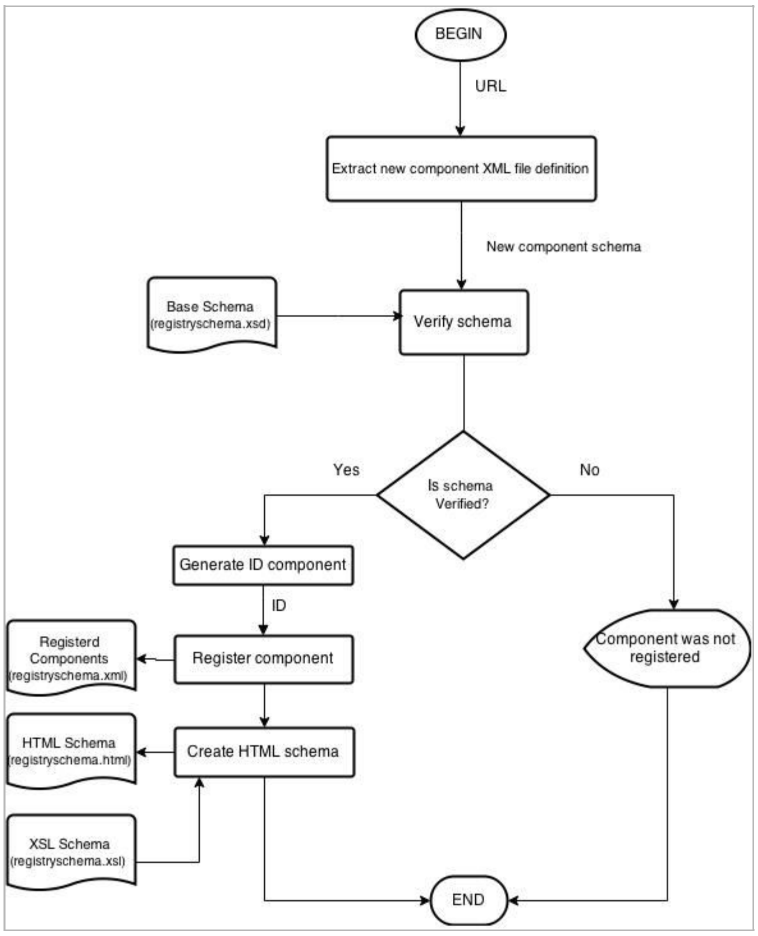

BEGIN
URL
Extract new component XML file definition
New component schema
Base Schema (registryschema.xsd)
Verify schema
Is schema Verified?
Yes
No
Generate ID component
ID
Registerd Components (registryschema.xml)
Register component
HTML Schema (registryschema.html)
Create HTML schema
XSL Schema (registryschema.xsl)
Component was not registered
END

Figure 7. Registry Component Flowchart. This graphic shows the procedure to register a new component.

3.2.3. Monitoring and Collection Data

The Data and Monitoring Repository install and execute a specific group of components. Moreover, the data from these executions are stored in this repository and then, different users can access to this repository visualizing particular information.

These are the main tasks performed in this repository:

1. Install components (Figure 8):
    a. Evaluate the list of Selected Components (XML file) to install a component.
    b. Get the URL from each ***not*** *installed* component with ***enable*** parameter equal ***true***.
    c. Access to Source Code Manager (e.g. GitHub) and bringing the component implementation.
    d. Create a directory component to store the component implementation. This directory will be called as the component ID.
    e. Install component (and its dependencies).
    f. If the installation is successful, update into the Selected Components XML schema, the ***enable*** descriptor with ***true*** value.
    g. If not, undo the changes.

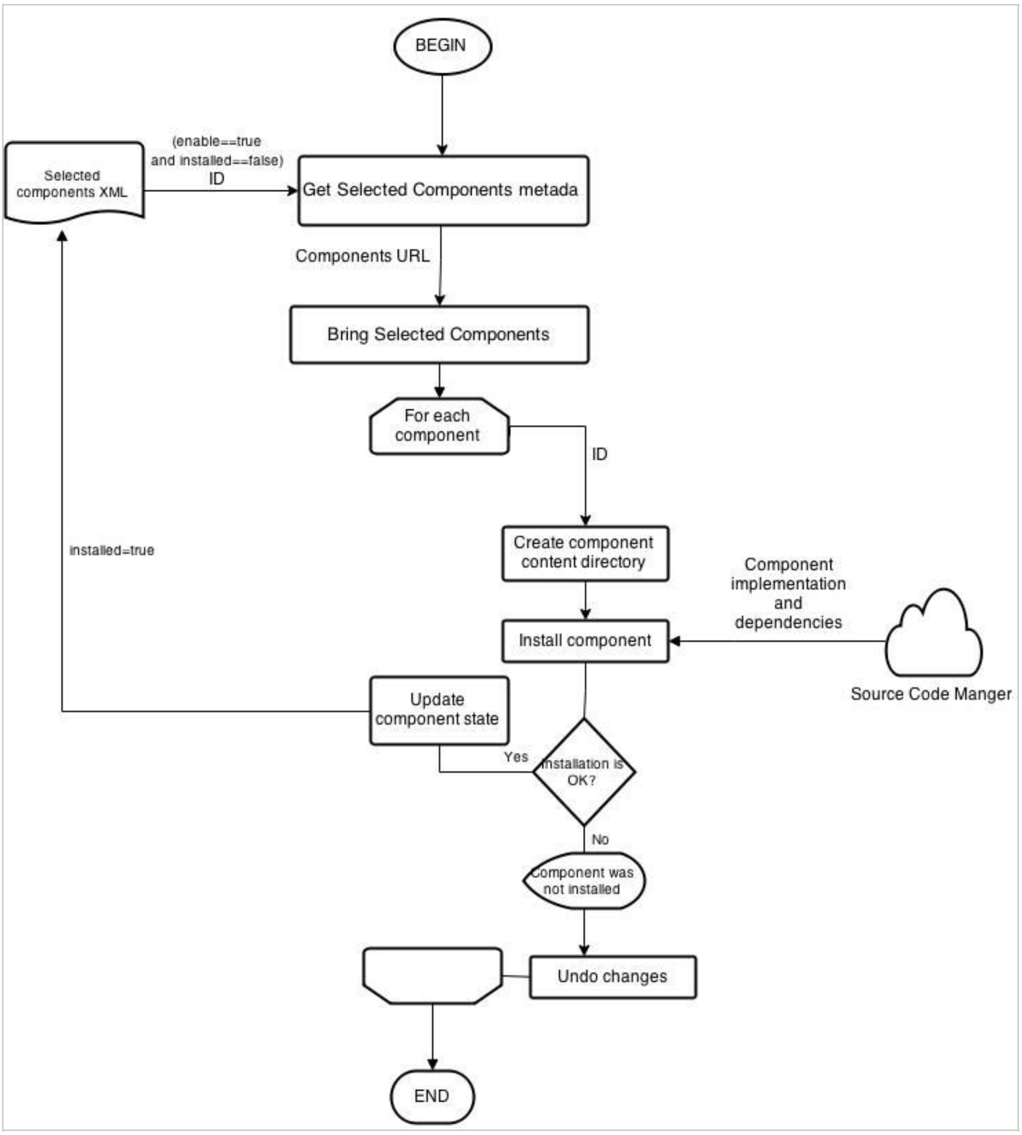

BEGIN
Selected components XML
(enable==true and installed==false)
ID
Get Selected Components metada
Components URL
Bring Selected Components
For each component
ID
Create component content directory
Component implementation and dependencies
Install component
Source Code Manger
installed=true
Update component state
Yes
Installation is OK?
No
Component was not installed
Undo changes
END

Figure 8. Install Component Flowchart. This graphic shows the procedure to install a set of selected component.

2. Run components (Figure 9):
    a. Evaluate the list of Selected Components (XML file) to run a component.
    b. Select each component with ***installed*** descriptor equal **true**.
    c. Get frequency descriptor format: supported format **cron-parser** (Figure 10)
    d. Run each component according its execution time interval (frequency):
        ○ Execution format: ***node executable [param1 param2 … paramN]***, where:
            - **executable**: is the filename of the executable component (see Table 3).
            - **[param1 param2 … paramN]**: params of execution (see Table 3).
    e. Collect the data from every component execution.

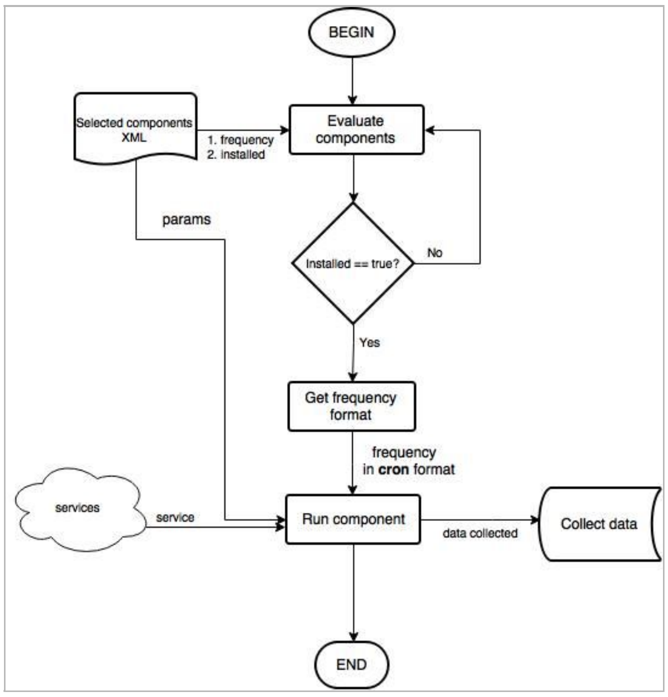
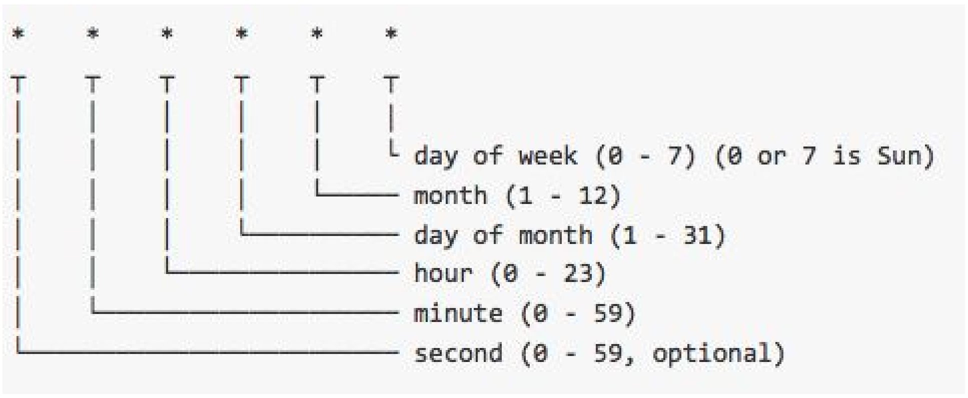


Figure 9. Run Component Flowchart. This graphic shows the procedure to execute a set of selected component.



```
*    *    *    *    *    *
┬    ┬    ┬    ┬    ┬    ┬
│    │    │    │    │    │
│    │    │    │    │    └ day of week (0 - 7) (0 or 7 is Sun)
│    │    │    │    └───── month (1 - 12)
│    │    │    └────────── day of month (1 - 31)
│    │    └─────────────── hour (0 - 23)
│    └──────────────────── minute (0 - 59)
└───────────────────────── second (0 - 59, optional)
```

Figure 10. Cron-parse format. This format is used to get the execution time interval of one component .

3. Collects data from components execution (Figure 11):
    a. Verify if the resource (rosource_name, resource_type and metric_id), from component execution, exists in the database.
    b. If this resource **is not** into the database, create a new record following the database structure (see Table 4).
    c. If this resource is into the database, bring it and update this resource with the new data collected from this execution.

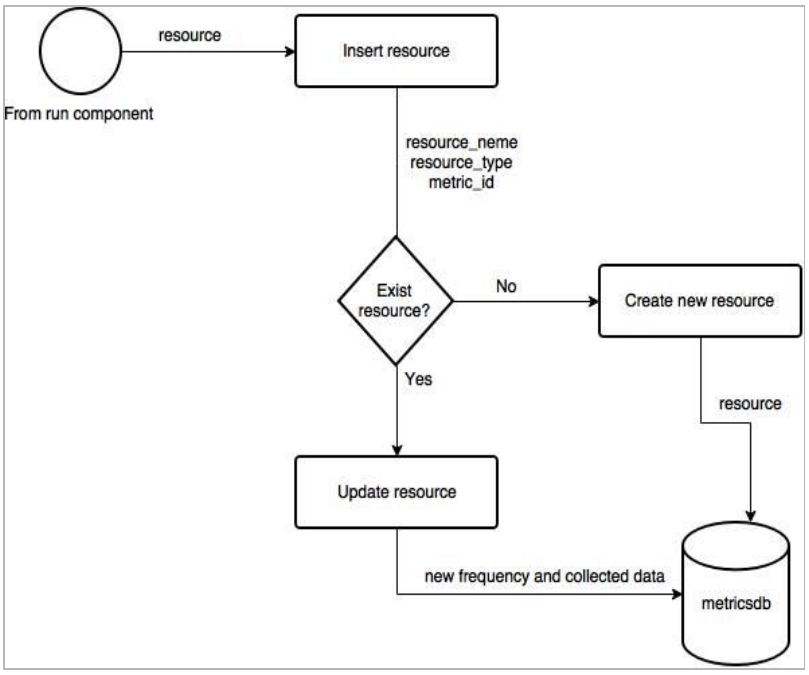


Figure 11. Run Component Flowchart. This graphic shows the procedure to execute a set of selected component.

3.2. Metrics Framework Analysis and Design

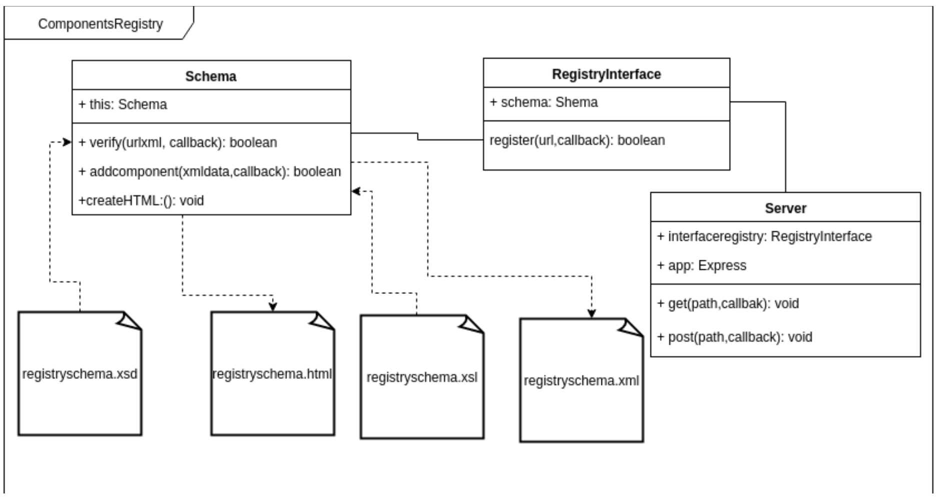

ComponentsRegistry
Schema
+ this: Schema
+ verify(urlxml, callback): boolean
+ addcomponent(xmldata,callback): boolean
+createHTML:(): void
RegistryInterface
+ schema: Shema
register(url,callback): boolean
Server
+ interfaceregistry: RegistryInterface
+ app: Express
+ get(path,callbak): void
+ post(path,callback): void
registryschema.xsd
registryschema.html
registryschema.xsl
registryschema.xml

Figure 12. ComponentsRegistry Module. Classes and methods responsible for the component registry compose this module

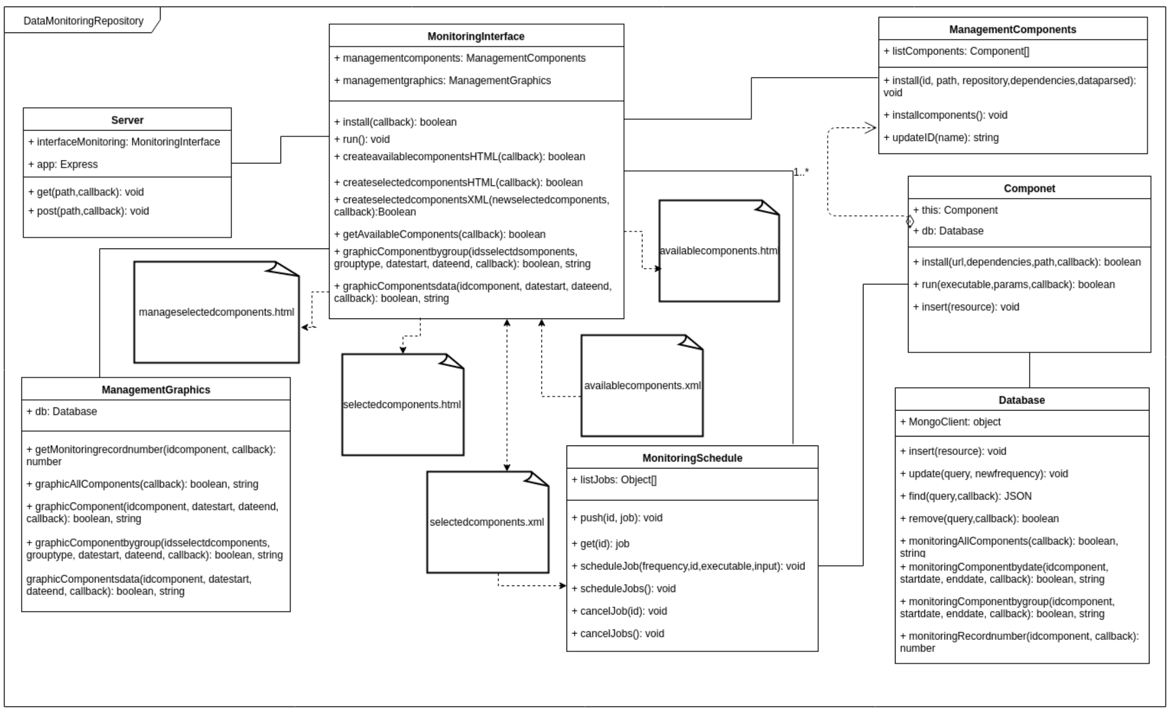

DataMonitoringRepository
MonitoringInterface
+ managementcomponents: ManagementComponents
+ managementgraphics: ManagementGraphics
+ install(callback): boolean
+ run(): void
+ createavailablecomponentsHTML(callback): boolean
+ createselectedcomponentsHTML(callback): boolean
+ createselectedcomponentsXML(newselectedcomponents, callback):Boolean
+ getAvailableComponents(callback): boolean
+ graphicComponentbygroup(idsselectdsomponents, grouptype, datestart, dateend, callback): boolean, string
+ graphicComponentsdata(idcomponent, datestart, dateend, callback): boolean, string
ManagementComponents
+ listComponents: Component[]
+ install(id, path, repository,dependencies,dataparsed): void
+ installcomponents(): void
+ updateID(name): string
Server
+ interfaceMonitoring: MonitoringInterface
+ app: Express
+ get(path,callback): void
+ post(path,callback): void
1..*
Componet
+ this: Component
+ db: Database
+ install(url,dependencies,path,callback): boolean
+ run(executable,params,callback): boolean
+ insert(resource): void
availablecomponents.html
manageselectedcomponents.html
selectedcomponents.html
availablecomponents.xml
selectedcomponents.xml
ManagementGraphics
+ db: Database
+ getMonitoringrecordnumber(idcomponent, callback): number
+ graphicAllComponents(callback): boolean, string
+ graphicComponent(idcomponent, datestart, dateend, callback): boolean, string
+ graphicComponentbygroup(idsselectdcomponents, grouptype, datestart, dateend, callback): boolean, string
graphicComponentsdata(idcomponent, datestart, dateend, callback): boolean, string
MonitoringSchedule
+ listJobs: Object[]
+ push(id, job): void
+ get(id): job
+ scheduleJob(frequency,id,executable,input): void
+ scheduleJobs(): void
+ cancelJob(id): void
+ cancelJobs(): void
Database
+ MongoClient: object
+ insert(resource): void
+ update(query, newfrequency): void
+ find(query,callback): JSON
+ remove(query,callback): boolean
+ monitoringAllComponents(callback): boolean, string
+ monitoringComponentbydate(idcomponent, startdate, enddate, callback): boolean, string
+ monitoringComponentbygroup(idcomponent, startdate, enddate, callback): boolean, string
+ monitoringRecordnumber(idcomponent, callback): number

Figure 13. DataMonitoringRepository Module. Classes and methods responsible for the component installation and execution compose this module.

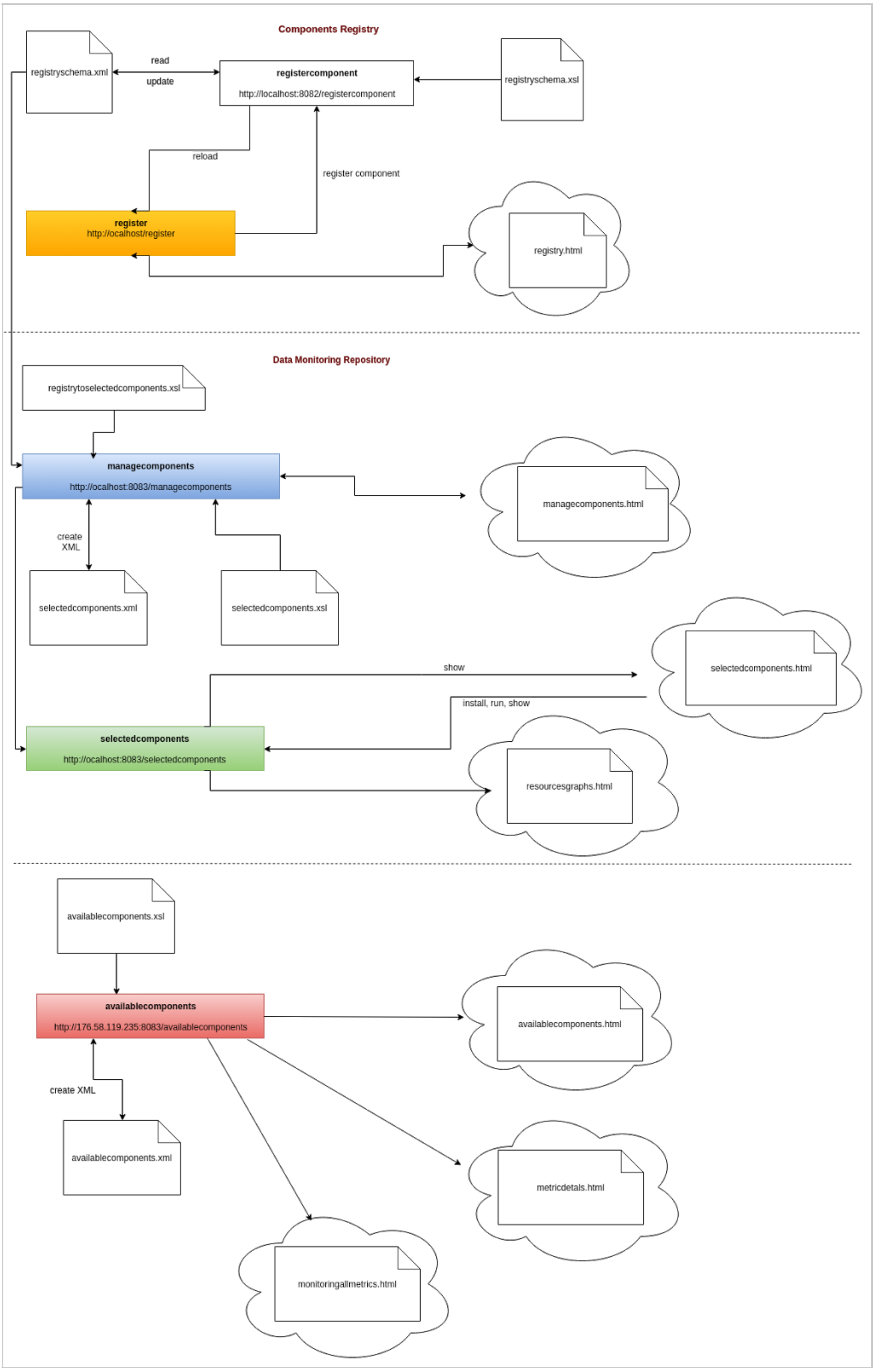


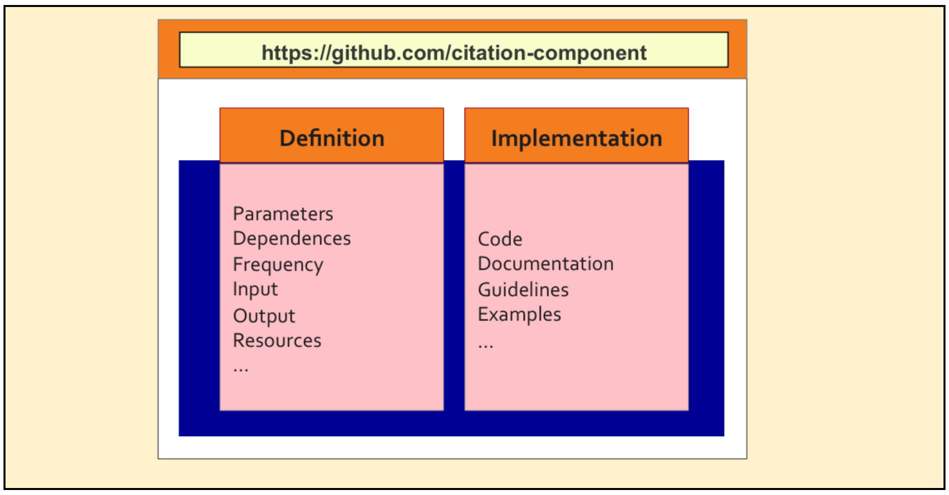
Figure 14. Interface Design. Server interactions with different schemas and HTMLs content related to GUI.

4. Example

This section gives an example concerning the definition, registry and monitoring for a Citation Metric-Component.

Citation Metric: defines a resource search in papers published in Pubmed Central repository. It includes all words and numbers in the title, abstract and article body, as well as in table and figure captions and in the article reference section.This metric can be used to provide evidence of the impact of the corresponding resource (e.g. database).

Citation Component: describes the Citation Metric definition (metadata) in a XML schema, and the Citation Metric implementation. This implementation (code, documentation, examples, etc.) must be stored into Source Code Manager (e.g. Github).

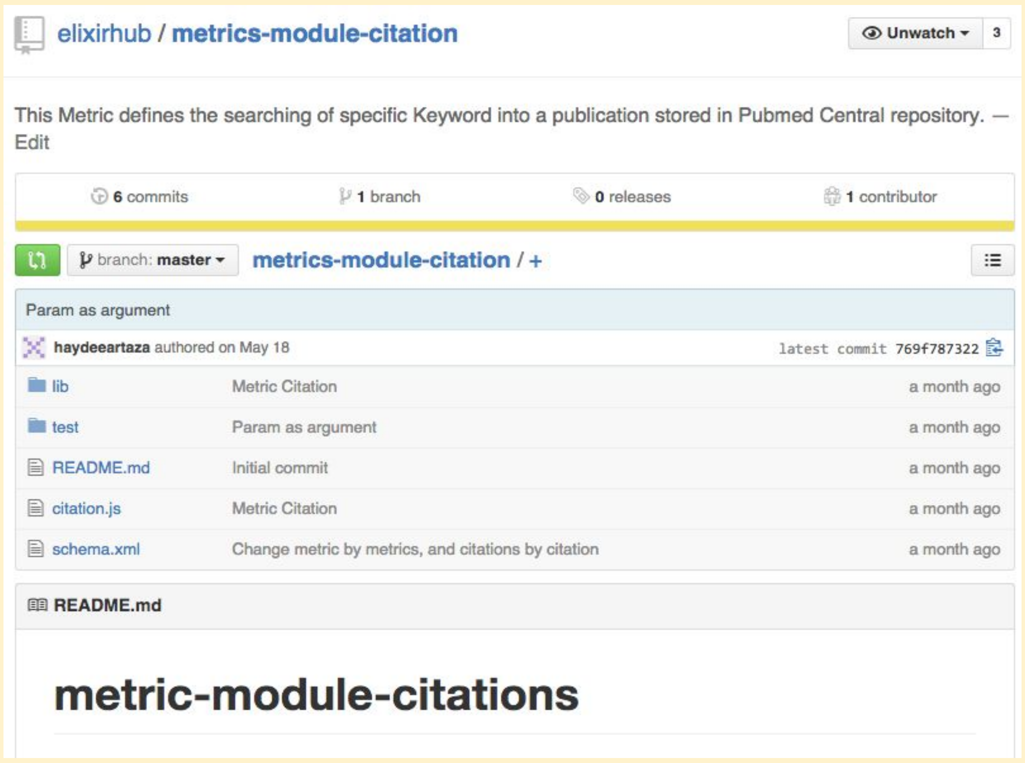


Component Schema: This metadata is used for registering the component. The main information for future execution are defined by:

- Name: used to generate the ID component.
- Input: resource search, in this example a database Uniprot.
- Dependences: libraries necessary to run this component.
- Resource: kind of resource, in this example a database.
- Repository: URL where the component implementation is stored.

```
<?xml version="1.0" encoding="utf-8"?>
<components_metrics>
   <component>
    <name>metric-module-citations</name>
    <description>Citations in publications</description>
    <authors>names of authors</authors>
    <input>
            <parameter>swissprot,swiss prot,swiss-prot,UniProtKB,uniprot</parameter>
            </input>
    <output>number</output>
    <dependences>
             <library>xml2js</library>
    </dependences>
    <frequency>monthly</frequency>
    <resource>
             <rsc>database</rsc>
    </resource>
    <repository>https://github.com/BioPisCO/metrics-module-citation.git</repository>
  </component>
</components_metrics>
```

Component Implementation: composed of code, documentation, examples, guidelines, and other content, which defined the component functionality. For this example, the Component Citation is stored into Github source code manager. The code implementation are distributed in **library** and **test** folder, and in **citation.js** file. The metadata schema is described in **schema.xml** file.



elixirhub / metrics-module-citation

This Metric defines the searching of specific Keyword into a publication stored in Pubmed Central repository. — Edit

6 commits | 1 branch | 0 releases | 1 contributor

branch: master | metrics-module-citation / +

Param as argument

haydeeartaza authored on May 18 — latest commit 769f787322

lib	Metric Citation	a month ago
test	Param as argument	a month ago
README.md	Initial commit	a month ago
citation.js	Metric Citation	a month ago
schema.xml	Change metric by metrics, and citations by citation	a month ago

README.md

metric-module-citations

Component Registry Schema: metadata from each registered component plus extra descriptors which manage the components registry. Additional information are defined in:

- Id: component ID (name+unique_alphanumeric).
- Version: alpha, beta, final, etc.
- Lastupdated: date of last upgrade.

This schema is adapted to increase the descriptors according to the model needs.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<components_metrics>
  <component>
        <name>metrics-module-citation</name>
        <description>Citations in publications</description>
        <authors>names of authors</authors>
        <input>
        <parameter>swissprot,swiss prot,swiss-prot,UniProtKB,uniprot</parameter>
        </input>
        <output>number</output>
```

```
            <dependencies>
            <library>xml2js</library>
            </dependencies>
            <frequency>monthly</frequency>
            <resources>
            <rsc>database</rsc>
            </resources>
            <repository>https://github.com/BioPisCO/metrics-module-citation.git</repository>
            <id>metrics-module-citation-4ywN_j5H</id>
            <version>final</version>
            <lastupdate>12/5/2015</lastupdate>
  </component>
</components_metrics>
```

Data and Monitoring Selected Components Schema: metadata of the selected components with extra descriptors to manage the monitoring and data collecting. Additional information are defined in:

- Enable: the implementation code is available into source code manager (true or false).
- Installed: if one component and its dependencies have been installed. If the value is true the component is ready to run.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<selectedcomponents>
  <component>
            <id>metrics-module-citation-4ywN_j5H</id>
            <name>metrics-module-citation</name>
            <frequency>minute</frequency>
            <enable>true</enable>
            <installed>false</installed>
            <input>
            <parameter>swissprot,swiss prot,swiss-prot,UniProtKB,uniprot</parameter>
            </input>
            <executable>test/testcitations.js</executable>
            <dependencies>
            <library>xml2js</library>
            </dependencies>
            <repository>https://github.com/elixirhub/metrics-module-citation.git</repository>
  </component>
</selectedcomponents>
```

Data and Monitoring Available Components Schema: metadata of the available components to manage the visualization of components in the GUI. This schema is under development.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<availablecomponents>
  <component>
            <id>metrics-module-citation-4ywN_j5H</id>
            <name>Citation</name>
            <description>Provides evidence of the resource impact in publications saved in Pubmed
            Central (PMC) repository</description>
            <detail>
                        <api>NCBI API (E-utilities)</api>
                        <query>Keyword separated by comma</query>
                        <output>Total publication that includes the keyword</output>
            </detail>
```

```
          <frequency>daily</frequency>
          <resource>
                    <name>swissprot,swiss prot,swiss-prot,UniProtKB,uniprot</name>
                    <type>database</type>
          </resource>
          <repository>https://github.com/BioPisCO/metrics-module-citation.git</repository>
  </component>
</availablecomponents>
```

Database schema: example of a database record. The field **_id** is assigned automatically by the database when the resource is created. Also, when a new record is created, the fields **name**, **type**, **metric** and the result of the component execution, are stored into the database. In the next iteration, when the component is executed again (in a specific frequency), the data collected is updated into the frequencies data array.

```
{
"_id" : ObjectId("558c0837fdfb938e74b7d4e9"),
"name" : "swissprot,swiss prot,swiss-prot,UniProtKB,uniprot",
"type" : "database",
"metric" : "metrics-module-citation-4ywN_j5H",
"frequencies" : [ { "period" : "minute", "date" : "2015-5-18-15-25-2", "value" : "30411" },
                  { "period" : "minute", "date" : "2015-5-18-15-26-2", "value" : "30411" },
                  { "period" : "minute", "date" : "2015-5-18-15-27-1", "value" : "30411" },
                  { "period" : "minute", "date" : "2015-5-18-15-28-1", "value" : "30411" },
                  { "period" : "minute", "date" : "2015-5-18-15-29-1", "value" : "30411" },
                  { "period" : "minute", "date" : "2015-5-25-15-1-2", "value" : "30469" },
                  { "period" : "minute", "date" : "2015-5-26-10-7-2", "value" : "30501" },
                  { "period" : "minute", "date" : "2015-5-26-10-8-1", "value" : "30501" },
                  { "period" : "minute", "date" : "2015-5-26-10-9-1", "value" : "30501" } ]
}
```

The database has been created using the from the mongo shell:

```
[16:11:57][n61806][admin]~$ mongo
MongoDB shell version: 3.0.4
connecting to: test
> use metricsdb
switched to db metricsdb
> db.createCollection("resources",{})
{"ok" : 1}
> show dbs
local        0.078GB
metricsdb  0.078GB
> show collections
resources
system.indexes
```

Components registry: example of console log for the component registry action. The Registry must access to component metadata from an URL. For this example (component citation), the URL used was:
https://raw.githubusercontent.com/BioPisCO/metrics-module-citation/master/schema.xml

```
[14:30:04][admin]~/ComponentsRegistry$ node registrycomponents.js
schema verified: true
Components have been added to ../schema/registryschema.xml
Updated: ../schema/registryschema.html
```

Components installation: example of console log for the component installation.The parameters of execution are defined in XML schema:

- Component ID: metrics-module-citation-4ywN_j5H.
- Enable: true (implementation is available).
- Dependencies: xml2js (libraries necessaries for this component).
- Repository: https://github.com/elixirhub/metrics-module-citation.git (URL from where extracts the component implementation).

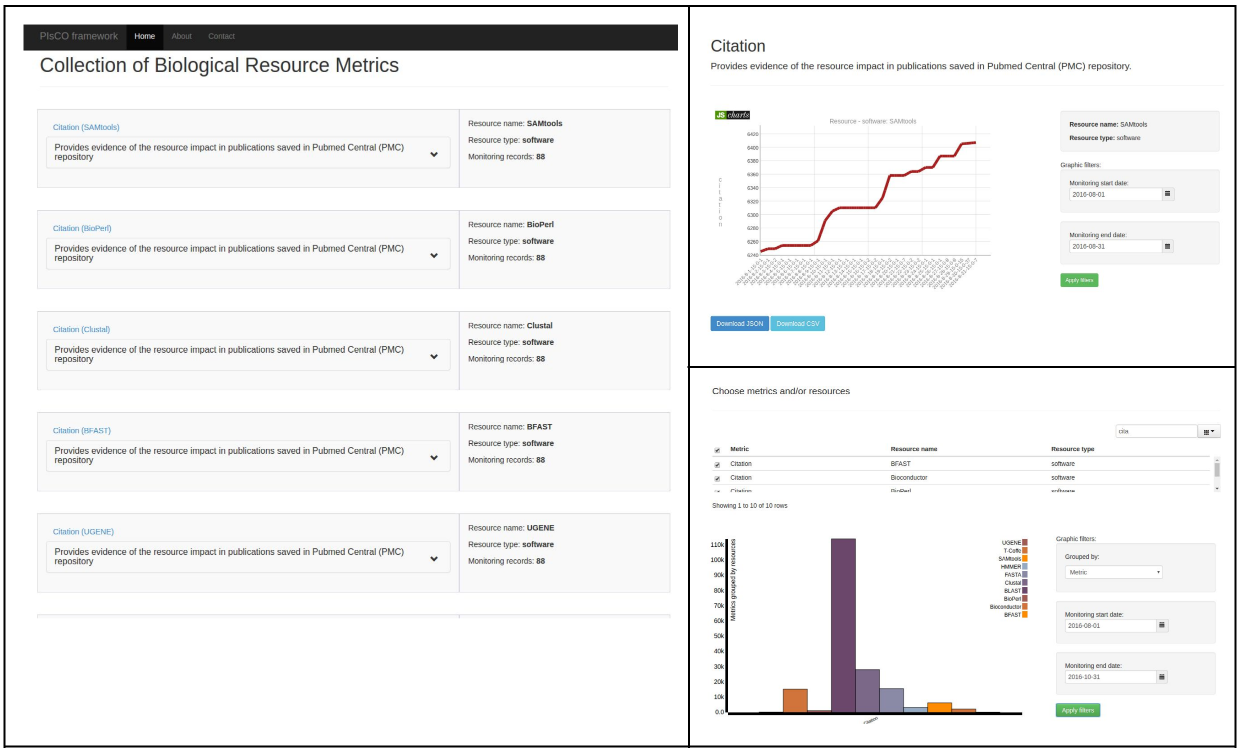
```
[15:06:23][admin]~/DataMonitoringRepository$ node installcomponents.js
https://nodeload.github.com/elixirhub/metrics-module-citation/zip/master
https://github.com/elixirhub/metrics-module-citation.git download into
components/metrics-module-citation-4ywN_j5H completed
installing library xml2js
xml2js@0.4.9 ../node_modules/xml2js
├── sax@0.6.1
└── xmlbuilder@2.6.4 (lodash@3.9.3)
Library xml2js installed
Component metrics-module-citation-4ywN_j5H has been updated in
schema/selectedcomponents.xml
```

Components monitoring: example of console log for the component monitoring. The parameters of execution are defined in XML schema:

- Component ID: metrics-module-citation-4ywN_j5H
- Frequency: minute
- Installed: true (upgrade in the installation execution).
- Parameter: swissprot,swiss prot,swiss-prot,UniProtKB,uniprot
- Executable: test/testcitations.js

```
[15:24:33][admin]~/DataMonitoringRepository$ node runcomponents.js
components/metrics-module-citation-4ywN_j5H/test/testcitations.js
Frequency: minute (cron format 0 * * * * *)
Resource: "swissprot,swiss prot,swiss-prot,UniProtKB,uniprot"
30411 citations for swissprot,swiss prot,swiss-prot,UniProtKB,uniprot
data save into > citation.txt
component metrics-module-citation-4ywN_j5H was executed
component metrics-module-citation-4ywN_j5H was executed
Connection established to mongodb://localhost:27017/metricsdb
Updated data into the "resources" collection:
{"ok":1,"nModified":1,"n":1}
```

User Interface GUI: example of the graphical interface showing the available components. These components are available to make different graphs from the data stored in the database.



Collection of Biological Resource Metrics

Citation (SAMtools)
Provides evidence of the resource impact in publications saved in Pubmed Central (PMC) repository
Resource name: **SAMtools**
Resource type: **software**
Monitoring records: **88**

Citation (BioPerl)
Provides evidence of the resource impact in publications saved in Pubmed Central (PMC) repository
Resource name: **BioPerl**
Resource type: **software**
Monitoring records: **88**

Citation (Clustal)
Provides evidence of the resource impact in publications saved in Pubmed Central (PMC) repository
Resource name: **Clustal**
Resource type: **software**
Monitoring records: **88**

Citation (BFAST)
Provides evidence of the resource impact in publications saved in Pubmed Central (PMC) repository
Resource name: **BFAST**
Resource type: **software**
Monitoring records: **88**

Citation (UGENE)
Provides evidence of the resource impact in publications saved in Pubmed Central (PMC) repository
Resource name: **UGENE**
Resource type: **software**
Monitoring records: **88**

Citation
Provides evidence of the resource impact in publications saved in Pubmed Central (PMC) repository.

Resource name: SAMtools
Resource type: software

Graphic filters:
Monitoring start date: 2016-08-01
Monitoring end date: 2016-08-31

Choose metrics and/or resources

cita

✓	Metric	Resource name	Resource type
✓	Citation	BFAST	software
✓	Citation	Bioconductor	software
✓	Citation	BioPerl	software

Showing 1 to 10 of 10 rows

Graphic filters:
Grouped by: Metric
Monitoring start date: 2016-08-01
Monitoring end date: 2016-10-31

5. References

[GitHub]
https://**github**.com/

[MongoDB]
https://www.mongodb.org/

[NodeJS]
https://nodejs.org/

[RFC2119]
Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.