The Improve platform analysis framework

The Improve platform exposes a REST API by which rehabilitation systems can upload data gathered during sessions with patients. Such data typically consists of information in three different categories:

1. Information that describe background settings, e.g., session date, the type of exercise performed during the session, difficulty level, targets of various kinds, etc.
2. Information that describes session results, e.g., session duration, achieved targets, scores, etc.
3. Sensor data, i.e., actual readings from sensors worn by the patients during the session

As the platform does not dictate exactly which kind of such session information a rehabilitation system should provide (only that is should be encoded as JSON), a challenge is how to support data analysis without becoming dependent on one particular rehabilitation system or one particular way of describing a session result. The solution is an analysis framework that is based on a model of pluggable analysis modules, thus allowing different types of analysis implementations to coexist and the possibility to dynamically develop and install new ones. An analysis handler component in the platform manages all such modules, and handles calls via the REST API requesting their services, see Figure 1.

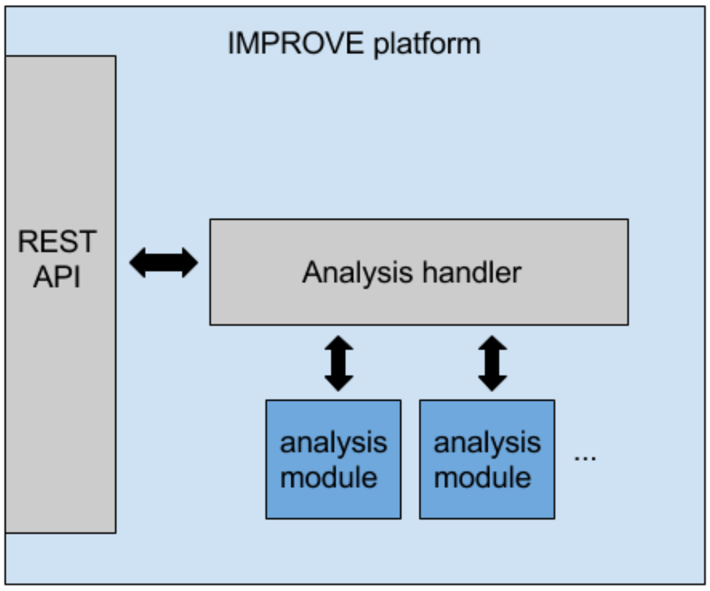

Fig. 1: A high level overview of the Improve platform analysis framework Analysis module specification

Analysis module specification

An analysis module is implemented as a Python module, i.e., a file containing Python definitions and statements. Each analysis module is identified in the platform using a name, e.g., `mymodulename`, and this name should be used in the Python file name, like so:

```
modulename.py
```

Thus, a module with name `mymodulename` should be placed in a file called `mymodulename.py`. The Python module must contain at least one class, which should have the same name as the module name:

```
class modulename(...):
```

This class must derive from the platform base analysis class AnalysisModule that has the following definition:

```
class AnalysisModule(object):
        def description(self):
        def necessary_config_params(self):
        def permission_level(self):
        def analyse(self, username, configParams, dbHandler):
```

The analysis module class must provide implementations of the description, necessary_config_params, permission_level and analyse functions, which are called by the platform analysis handler at different stages during the module's lifetime.

The *description* function is supposed to return a human readable string that explains what kind of analysis the module is able to perform. This description is intended to be presented to a human user, which can then, based on the description, decide whether to use the module or not.

The *necessary_config_params* function should return a JSON expression that describes which configuration parameters the module expects, i.e., parameters that should be supplied when initiating the data analysis procedure provided by the module. The JSON expression should be an array of parameter object declarations, where each such declaration is described and exemplified in Table 1.

Name	Config parameters documentation			
configParam Attributes	Name	Type	Description	Requ ired
	name	Text,	The name of the config	True

	String	parameter.	
description	Text, String	The description of the config parameter.	True
type	Text, String	The type of the config parameter. See available types further down.	True
default	Text, String	The default value of the config parameter. Not always present.	False
range	Text, String	The range of accepted values.	True
required	Bool, bool	true if the config parameter is required for the module, false otherwise.	True
max_amount	Int, int	Maximum number of values to send of this config parameter. Negative value means that this is arbitrary.	True
min_amount	Int, int	Minimum number of values to send of this config parameter. Negative value means that this is arbitrary.	True

Available config parameter types

Name	Primitive type	Description
Text	[string]	
Int	[int]	
Float	[float]	
Bool	[boolean]	True or False
Enum	[string]	All accepted values.
exerciseID	[string]	ID used in the database.
dataID	[string]	ID used in the database.
deviceID	[string]	ID used in the database.
exerciseResultID	[string]	ID used in the database.
rehabilitationSetID	[string]	ID used in the database.
patientInformationID	[string]	ID used in the database.

	patientConditionID	[string]	ID used in the database.
	patientID	[string]	ID used in the database.
	userID	[string]	ID used in the database.
	userGropID	[string]	ID used in the database.
	organizationID	[string]	ID used in the database.
	analysisTaskID	[string]	ID used in the database.

ConfigParam description Example	{
	   "name": "birth year",
	   "description": "Patient birth year",
	   "type": "int",
	   "default": "",
	   "range": "1-100",
	   "required": true,
	   "max_amount": -1,
	   "min_amount": -1
	}		

Table 1: Analysis module configuration parameters specification and example

The *permission_level* function should return a platform permission level number, which describes the minimal permission level necessary to start an analysis using the module.

The *analyse* function is where the actual data analysis should happen. When this function is called, the module is free to do any kind of computation and access the platform's database storing information about patients and exercise results. The module typically uses the API and the platform's database handler (see [3]) to access the database and retrieve information concerning for instance patients, treatments and exercise results. The call to *analyse* runs in a "private" thread, so the function may take as long as it needs to finish the analysis. Once finished, the function should return analysis result expression, as described in the next section.

Analysis access rights

As defined by the analysis module Python class API, all modules must express which permission level is needed to start the analysis. Since the platform uses authentication and authorization mechanisms to control the access to data, all incoming REST calls will have the same access level as the authenticated user. If the call to start an analysis doesn't have a high enough level for the call to complete (according to the permission level set by the module), the platform returns an access denied error message.

In the case of analysis modules, the permission level set by the module should be the minimum level needed in order for the analysis to succeed. What this means in reality is that the level must be high enough to allow the module to access the data it needs in the database.

When a REST call comes in to start an analysis, the module checks if the access level associated with the call is sufficient for the data access. If not, the module won’t start the analysis and instead return an error message back to the caller. If the level is ok, the analysis is started and the result made available to the caller when the analysis is complete. Note that it’s important to set the correct analysis permission level, if the level is set too low the module will not be able to access the data it needs, and if the level is set to high only the users will be highest access rights will be able to run the analysis.

Analysis results

The result produced by analysis modules should contain both the actual result data of the analysis, as well as information indicating how the result could be presented to a user. The result may also contain a model to be used for later analyses by the same or other modules.

A result is represented by a JSON expression, as described and exemplified in Table 2 below. The expression contains an array of individual result objects, where each object may be of the types plot, text or html.

- `plot` results describe data to be plotted in a 2d chart. Each plot may contain several data series, and it’s possible to specify whether they should be plotted using points, lines, etc.
- `text` results describe data which is encoded as plain text.
- `html` results describe data which should be presented as html.

Name	AnalysisResult documentation			
Result Attributes if “type” = “plot”.	Name	Type	Description	Requir ed
	name	Text, String	The name of this result.	True
	type	Enum, String	In this case this will have the value “plot”.	True
	data	2D-Array	The data to be presented. See format in example.	True
	priority	Number, Float	The priority of this result. Higher priority number means that the frontend possibly presents this result higher up.	True
	legend	Text, String	The name of this dataset. Will be used as a legend in the plot.	True
	subtype	Enum,	Option for how to plot the data. Possible values are lines, bars,	True

	String	columns, points.	
plotID	Text, String	An identifier that could be used when plotting different data in one plot. Different IDs means that the data should be plotted in different plots.	True

Result Attributes if “type” = “text”.

Name	Type	Description	Required
name	Text, String	The name of this result.	True
type	Enum, String	In this case this will have the value “text”.	True
data	String	The data to be presented	True
priority	Number, Float	The priority of this result. Higher priority number means that the frontend possibly presents this result higher up.	True

Result Attributes if “type” = “html”.

Name	Type	Description	Required
name	Text, String	The name of this result.	True
type	Enum, String	In this case this will have the value “html”.	True
data	Html, String	The data to be presented in html-format.	True
priority	Number, Float	The priority of this result. Higher priority number means that the frontend possibly presents this result higher up.	True

Result Example

```
{
    "results":
    [
        {
            "type": "plot",
     "data": [{
           "data": [[1,3],[2,634],[3,33],[5,233]],
           "subtype": “points”,
           "legend": "Legend to use"
         }]
            "priority": 5,
          "title": "a name",
     "x_label": "something",
     "y_label": "something"
    },
         {
              "type": "text",
              "data": "here is a result",
              "priority": 2
         },
         {
              "type": "html",
              "data": "htmlcode...",
              "priority": 6
         }
     ],
  "model": {...}
}
```

Available type types

Name	Primitive type	Description
Plot	string	Indicates that the result should be presented as a plot.
Text	int	Indicates that the result should be presented as text.
Html	float	Indicates that the result should be presented as html.

Available subtype types

Name	Primitive type	Description
div	string	Indicates that the html-code should be put in a div.
body	string	Indicates that the html-code should be put in a body.

Notes	

Table 2: Analysis result specification and example

Analysis results are stored in the platform's database, and are accessible via the REST API.

Apart from a result expression, a analysis module is also able to produce a model. Models are not intended to be presented as results to clients outside of the platform. Instead they are meant to be used "internally" by the same, or another, analysis module in future analyses. A model could for instance contain various statistical values concerning the overall status of all rehabilitation sets, calculated once by a module and then used in all future. Models are represented as JSON and are stored in the platform database for easy access when needed.

Installation of an analysis module

As mentioned above, analysis modules are not part of the core platform, but are rather dynamic components (plugins) that can be added or removed from the platform at will, for instance based on what’s needed in a specific customer platform installation.

In order for the platform to know which analysis modules are available at any given point in time it searches a specific directory in its installation directory tree. All Python module files found in this directory are assumed to represent an analysis module, and are inspected by the platform to determine the module’s name and implementation. Installation of a new module can thus be achieved by simply placing its corresponding Python module file in the correct directory, and the platform will then automatically find it and make it available via the REST API.