CERN-BE-2023-015
31/10/2022

SURVEYPAD, A COMMON INTERFACE TO GEODE & DATA PROCESSING TOOLS

P. Lewandowski*, R. Ducceschi, F. Klumb, CERN, Geneva, Switzerland

Abstract

The metrology and alignment of components installed in accelerators is a very complicated process requiring many calculations to ensure the best performance of the CERN complex. Throughout the years, surveyors at CERN have been relying on several pieces of different offline processing software that were meant to function without an internet connection in the tunnels. Most of the programs were operating as clunky command line tools. They required text input files, formatted in a specialized and unique way, and generated non-standardized text file outputs. At the same time, surveyors had to use a survey database, and its web interface named Geode, as the source of theoretical positions and all related measurements and computations. Since a need to simplify this workflow arose, SurveyPad was created.

SurveyPad is a C++ plugin-based software meant to integrate and govern multiple pieces of processing software. It requires a plugin to be created for each one of them. The plugins provide communication with their corresponding software and come with a graphical user interface that can be displayed within SurveyPad providing easy manipulation of the project files. The plugins share some more advanced features. These include elaborate text editing and one token look-ahead, left-to-right grammars (LALR(1)) for a tailor-made parsing of the contents of software files. Lastly, although the pieces of survey software cannot communicate with one another, the plugins can. They cooperate and share some functionalities via SurveyPad.

Current feedback from users indicates that despite ongoing developments, SurveyPad has already proved to be a functional and convenient tool used by surveyors on a daily basis.

INTRODUCTION

CERN complex consists of tens of thousands of elements that need to be measured and aligned. This results in a great quantity of data that has to be stored and analyzed. [1] The measurement themselves are obtained with high-accuracy survey instruments and the output from acquisition software is stored in the survey database that can be accessed using a browser via its web interface named Geode [2]. For further processing and analysis of the data, several pieces of offline software are used. Each of them requires data from Geode and some of them need to upload their results back to the database. This workflow is presented in Fig. 1.

For historical reasons, each piece of processing software has different creators, different use cases, and therefore, different interface and data formats. Each of them uses a different format for input and output files, presents points in different ways, and might use different (or several) reference systems. There is no possibility to communicate between them and due to that surveyors had to manually convert the files to comply with their standards. Since most of them have no user interface, users had to manually modify the files with basic text editors and provide them as arguments to the executable. However, these pieces of software are crucial to surveyors and cannot be easily replaced without a large workload.

OBJECTIVES

SurveyPad is meant to greatly simplify the previous workflow and is meant to call the processing pieces of software in a user-friendly way. Its main principles are:

- One simple single interface to manage all pieces of survey software.
- Facilitation of editing any survey project files.
- Interoperability between any survey software and better, straightforward communication with the database.

Having fulfilled these objectives, the manual repeatable work of handling and converting the files by surveyors would disappear. Moreover, any additional project file editing features will ease the daily work and might help in discovering errors in computations. Lastly, by keeping the interface simple and expandable using plugins, new advanced features could be easily introduced without modifications to the common interface.

ARCHITECTURE

In order to achieve it, a plugin-based architecture was selected. For this application, it means that the main software (SurveyPad) by itself will not provide any survey-specific functionalities and these will be provided by respective survey software plugins. The main reason behind it is to decouple, i.e. separate, all different pieces of processing software. Since none of them is directly relying on any other, this approach avoids unnecessary dependencies and allows keeping the development contained and specialized. This is especially important since any user can use different plugins and is not required to install all of them to ensure the proper operation of their software.

Plugins

The plugins themselves are specific to a single piece of processing survey software, which is distributed in the form of an executable. Plugins by definition are fully optional, i.e. they are not required for an application to run, and are meant to increase the set of functionalities provided by the

* przemyslaw.lewandowski@cern.ch

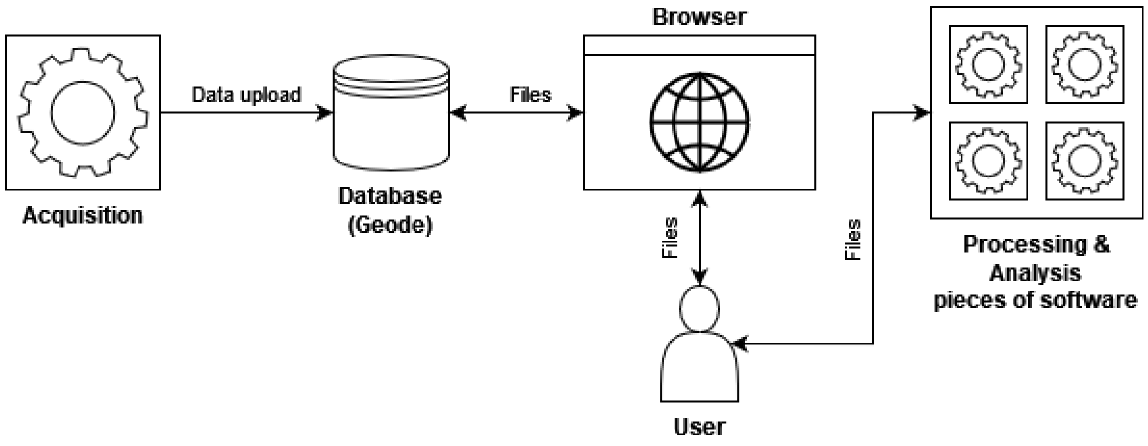


Figure 1: Initial data processing workflow.

software. [3] In SurveyPad they serve as bridges to their respective processing software and they consist of the three main components:

- Graphical interface - user interface meant to modify the project files. It can be in any form: text, table, or custom-made.
- Launcher interface - defines how to launch the application.
- Configuration interface - exposes configurable settings to SurveyPad.

Every plugin needs to implement the *SPluginInterface* that contains all these interfaces. This design is presented in Fig 2. The plugin interface is contained inside of SUrvey Graphical Library (SUGL in short). This library is shared between plugins and SurveyPad, and therefore, it also contains all the plugin-shared functionalities.

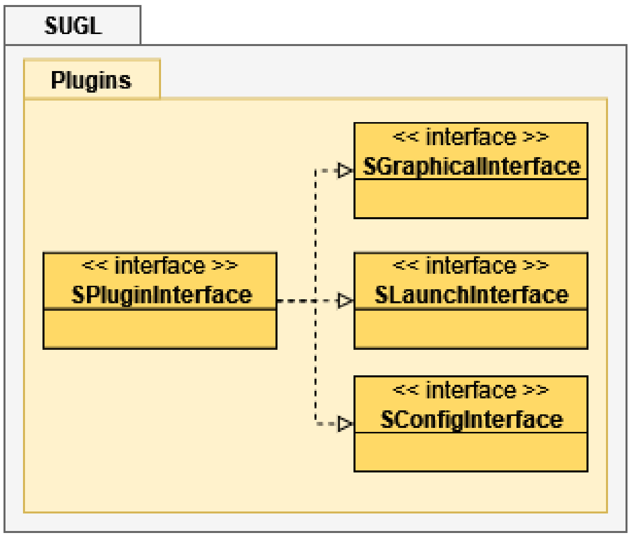


Figure 2: Plugin interface.

In terms of implementation, plugins can differ a lot. Some of them might provide just a text editor displaying the project files, some others table editors, and some others both, or a custom GUI form. There is no restriction in terms of graphical representation. The same naturally applies to settings or launching the application since it may be very specific and unique for each piece of software. This freedom is very important and allows adapting the plugins to the needs of a survey application and its project files.

As for the programming language, C++ was chosen for implementation. This is due to the fact that most of the survey processing software is also C++ based. Moreover, along with C++ *Qt* can be easily used. It is a cross-platform software for creating graphical applications, and although *Windows* is the main operating system used by surveyors, SurveyPad is still meant to run both on *Windows* and on *Linux* operating systems. The last external dependency is *QScintilla*, which is a port to *Qt* of the *Scintilla* programmers editor widget [4] [5]. *Scintilla* is a free, open-source library that provides a text editing component that is extremely important for the implementation of reliable and advanced text editors required for all pieces of survey software.

SurveyPad

With a plugin-based approach, the software can be logically split into core and its extensions. Having a stable core the application can be extended on demand with plugins and the core itself could be delivered with the minimum functionalities. Hence, the application is fully customizable and can be adapted to the user's needs. [3]

By applying this concept, SurveyPad became a relatively simple application that is meant to load and govern the plugins and their projects during its operation. By itself it will provide the following functionalities:

- Plugin loading and project management.
- Handling general settings and the ones specific to plugins.
- Displaying the graphical interface provided by a plugin.
- Logs handling and display.

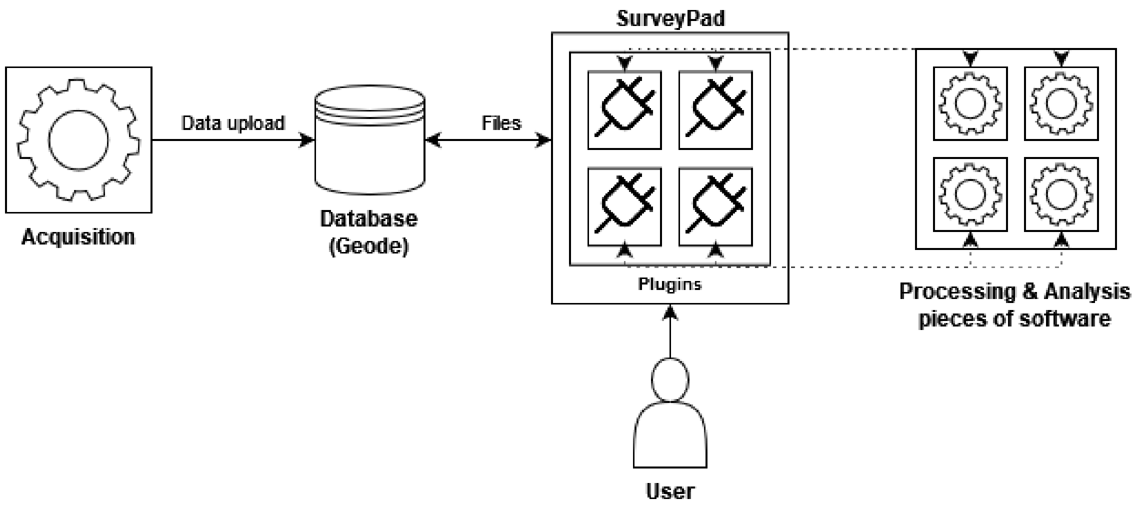


Figure 3: Simplified data processing workflow with SurveyPad.

The rest is meant to be provided by the plugins themselves. Since the plugins extend the possibilities of SurveyPad and some of them can directly interact with the database, the current workflow is simplified and is presented in Fig 3.

MAIN FEATURES

SurveyPad provides many improvements to the workflow and provides some features that were tailor-made for the needs of surveyors.

User interface

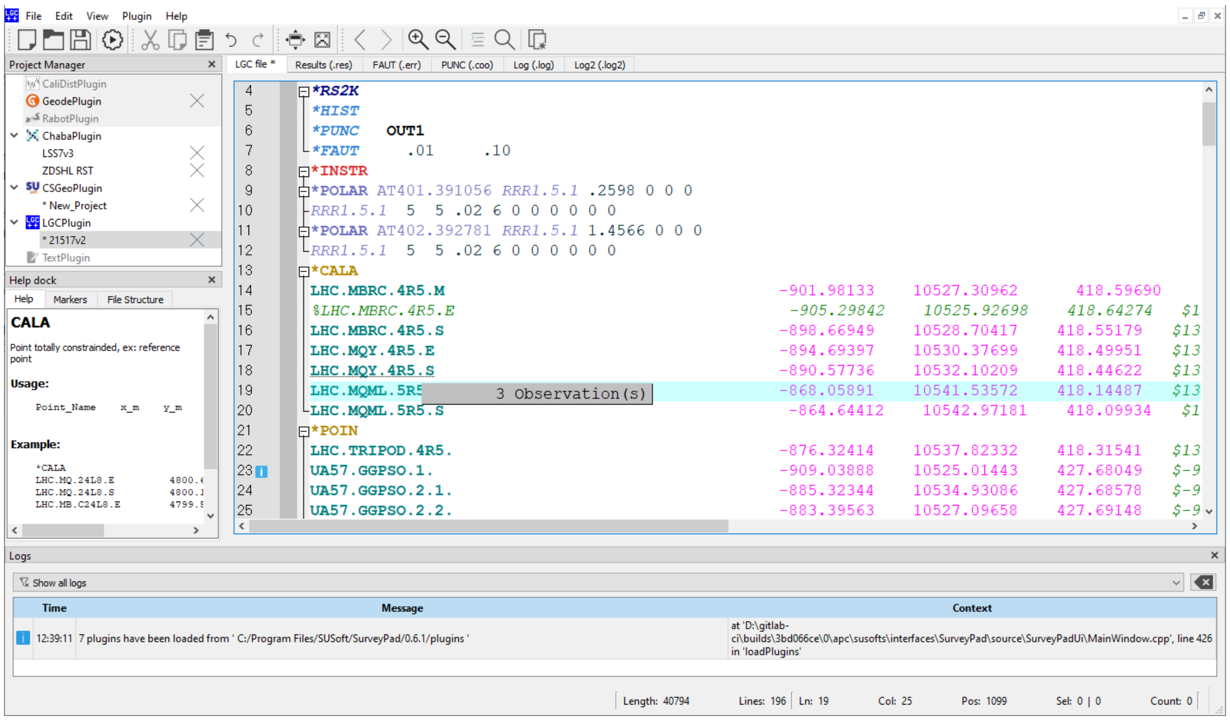
In Fig. 4 the current interface is presented, with a Project Manager and a Help dock on the left, a Log viewer at the bottom, and a dedicated editor for a plugin in the center. The most distinguishing features of this interface are:

- A Project Manager that has typical file management functionalities of creating new files, opening, saving, and closing them. It shows the projects in a clear tree structure to ease the navigation. It also distinguishes between plugins that can have multiple projects and the ones that should be single instance only.
- A highly customizable graphical interface that allows changing positions of all docks and toolbars that will be kept and restored properly when reopened. Toolbars contain often accessible functionalities whereas a full list of them is available in the submenus on top.
- A Help dock displaying currently relevant information. What is displayed there is plugin specific, but most of the time, it consists of three elements. The first one is the Help tab with text information explaining what is expected of the user. The second is the Marker tab that allows to mark positions in a text editor to ease the navigation and also displays the positions of errors. Lastly, there is a File Structure tab displaying a schema of a currently processed file as a nested hierarchy.
- A Logs viewer that keeps track of all that happened during runtime. It supports log filtering based on the severity level.
- All plugins expose their expected file extension and SurveyPad automatically knows which plugin to pass the file to and all these extensions are associated with SurveyPad on *Windows*.

Text editing

Besides the typical text editing features along with markers and help dock mentioned in *User Interface*, text editors also provide advanced text styling. This styling is achieved with *QScintilla* and the logic is realized using lexers or, in more complicated cases, parsers along with some hand-made grammar. The basic principle of operation is quite simple: find tokens (i.e. words), try to identify them, and based on their type or positioning, style them. However, the contents of the input and especially output files of survey software are not very systematized and easily analysable since such a use case was never foreseen. Moreover, lexers and parsers cannot be easily duplicated between the plugins since the formatting of these files is entirely different and nonstandard.

For simple cases, a hand-written lexer with all the relevant logic can be created that will handle the styling. However, for more complicated cases a parser is required. To obtain it, *QLALR* from *Qt* is used. *QLALR* is a parser generator that, based on a file with grammar, produces C++ code that can match input to that grammar. The accepted grammar needs to be one token look-ahead, left-to-right grammar (LALR(1)). This is a type of deterministic context-free grammar, also called Chomsky Type 2. Such grammar allows a text to be matched to a sequence of replacement rules, which can match to other rules, but finally always resolves to terminal symbols [6]. Terminal symbols are the ones that can be seen directly inside of the text files, whereas non-terminal symbols, also called rules, are the ones that define relations between them.



CALA

Point totally constrainded, ex: reference point

Usage:

```
Point_Name    x_m    y_m
```

Example:

```
*CALA
LHC.MQ.24L8.E          4800.
LHC.MQ.24L8.S          4800.
LHC.MB.C24L8.E         4799.
```

```
*RS2K
*HIST
*PUNC   OUT1
*FAUT     .01     .10
*INSTR
*POLAR AT401.391056 RRR1.5.1 .2598 0 0 0
RRR1.5.1  5  5 .02 6 0 0 0 0 0 0
*POLAR AT402.392781 RRR1.5.1 1.4566 0 0 0
RRR1.5.1  5  5 .02 6 0 0 0 0 0 0
*CALA
LHC.MBRC.4R5.M                             -901.98133   10527.30962    418.59690
%LHC.MBRC.4R5.E                             -905.29842   10525.92698   418.64274   $1
LHC.MBRC.4R5.S                             -898.66949   10528.70417   418.55179   $13
LHC.MQY.4R5.E                              -894.69397   10530.37699   418.49951   $13
LHC.MQY.4R5.S                              -890.57736   10532.10209   418.44622   $13
LHC.MQML.5R5          3 Observation(s)     -868.05891   10541.53572   418.14487   $13
LHC.MQML.5R5.S                              -864.64412   10542.97181   418.09934   $1
*POIN
LHC.TRIPOD.4R5.                            -876.32414   10537.82332   418.31541   $13
UA57.GGPSO.1.                              -909.03888   10525.01443   427.68049   $-9
UA57.GGPSO.2.1.                            -885.32344   10534.93086   427.68578   $-9
UA57.GGPSO.2.2.                            -883.39563   10527.09658   427.69148   $-9
```

Time	Message	Context
12:39:11	7 plugins have been loaded from ' C:/Program Files/SUSoft/SurveyPad/0.6.1/plugins '	at 'D:\gitlab-ci\builds\3bd066ce\0\apc\susofts\interfaces\SurveyPad\source\SurveyPadUi\MainWindow.cpp', line 426 in 'loadPlugins'

Length: 40794   Lines: 196   Ln: 19   Col: 25   Pos: 1099   Sel: 0 | 0   Count: 0

Figure 4: SurveyPad v0.6.1.

```
Number:  '0'  |  '1'  |  '2'  |  '3'  |  '4'
      |  '5'  |  '6'  |  '7'  |  '8'  |  '9';
Letter:  'A'  |  'B'  |  'C'  |  'D'  |  'E'
      |  'F'  |  'G'  |  'H'  |  'I'  |  'J'
      |  'K'  |  'L'  |  'M'  |  'N'  |  'O'
      |  'P'  |  'Q'  |  'R'  |  'S'  |  'T'
      |  'U'  |  'V'  |  'W'  |  'X'  |  'Y'
      |  'Z';
EOL:  '\n';
Letters:  Letter
      |  Letters  Letter;
Name:  Letters;
Point:  Name  Number  Number  Number  EOL;
Points:  Point
      |  Points  Point;
```

Figure 5: Simplified example grammar defining a list of points separated by an end-of-line character.

As seen in Fig. 5, the first three rules consist of terminal symbols only, whereas the rest uses non-terminal rules to define the relations. Grammars of this type can even match recursive "nesting", however, they act locally, i.e. they cannot handle problems regarding the entire document. Nonetheless, using *QLALR* to generate our own parser, one can still supply user-written code to introduce superficial context-sensitivity, which is extensively used in some plugins. The limited context-sensitivity includes storing some names encountered in some grammar rules and accessing them in further rules to enable some specialized behavior. For instance, this could be storing a point name when it is defined and counting its dedicated observations to check if their number is sufficient for a given point type.

With both lexers and parsers, a number of things can be achieved and personalized for given words or lines:

- Colour styling of the text. Different styles for point names, coordinates, sigmas, etc.

- Showing tooltips with additional information when hovering over text.

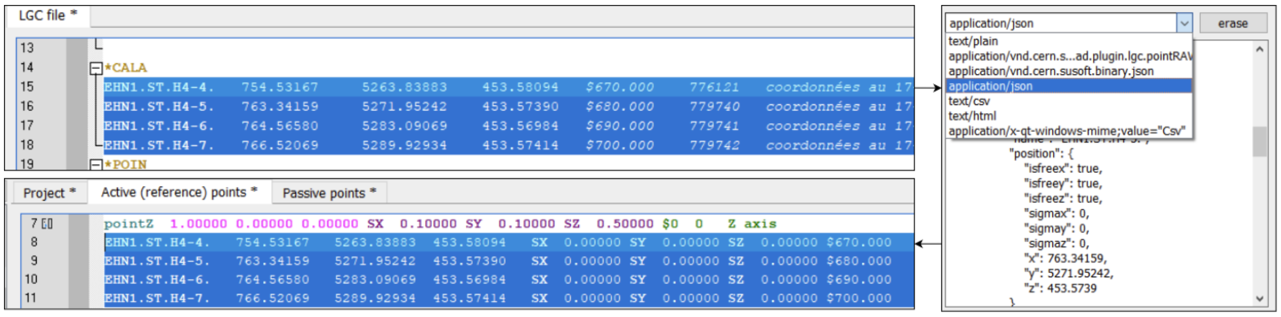
- Input and output data validation to discover some irregularities that can be marked with both markers or colour.

- Linking some words together, even between different files. That means that by clicking on some word, the user can be directed to other parts of the text or file. As an example, by clicking on the point name, the user will always be directed to its declaration.

This enables dynamic text editing, with error detection while working on the file, and enhanced clarity when analysing the results.

Software Interactions

Since all the processing pieces of software are dealing with different input and output text formats, their interactions were limited in the past. However, using SurveyPad some plugin-plugin and web interactions are feasible.


LGC file *
Project *
Active (reference) points *
Passive points *
application/json
text/plain
application/vnd.cern.s...ad.plugin.lgc.pointRAV
application/vnd.cern.susoft.binary.json
application/json
text/csv
text/html
application/x-qt-windows-mime;value="Csv"
erase

```
*CALA
EHN1.ST.H4-4.    754.53167    5263.83883    453.58094    $670.000    776121    coordonnées au 17
EHN1.ST.H4-5.    763.34159    5271.95242    453.57390    $680.000    779740    coordonnées au 17
EHN1.ST.H4-6.    764.56580    5283.09069    453.56984    $690.000    779741    coordonnées au 17
EHN1.ST.H4-7.    766.52069    5289.92934    453.57414    $700.000    779742    coordonnées au 17
*POIN
```

```
pointZ  1.00000 0.00000 0.00000 SX  0.10000 SY  0.10000 SZ  0.50000 $0  0   Z axis
EHN1.ST.H4-4.   754.53167   5263.83883  453.58094   SX  0.00000 SY  0.00000 SZ  0.00000 $670.000
EHN1.ST.H4-5.   763.34159   5271.95242  453.57390   SX  0.00000 SY  0.00000 SZ  0.00000 $680.000
EHN1.ST.H4-6.   764.56580   5283.09069  453.56984   SX  0.00000 SY  0.00000 SZ  0.00000 $690.000
EHN1.ST.H4-7.   766.52069   5289.92934  453.57414   SX  0.00000 SY  0.00000 SZ  0.00000 $700.000
```

```
        "position": {
            "isfreex": true,
            "isfreey": true,
            "isfreez": true,
            "sigmax": 0,
            "sigmay": 0,
            "sigmaz": 0,
            "x": 763.34159,
            "y": 5271.95242,
            "z": 453.5739
        }
```

Figure 6: Copy content serialization from LGC plugin to Chaba plugin, with multiple serialization representations on the right.

Plugins First of all, copy contents can be serialized if needed. This allows serializing some data structures, for example, point definitions, in one software (plugin), and pasting this content into some other one automatically with a different format and the same data, without the user's effort. An example is presented in Fig. 6 where the contents from one plugin are serialized on copy to multiple different formats and on paste action, the most fitting format is selected for deserialization or used as is.

Moreover, plugins can share some functionalities with one another using *Entry Points* [7]. These are some exposed and agreed-on names that provide specific behavior or classes without sharing any source code between them. This is realized using *Qt's Meta-Object System* [8]. Each plugin registers its own entry points with some name when loaded and any other plugin can search for any exposed functionalities just by knowing the name of the functionality it needs. The extent of shared functionalities is not limited in any way and currently, it includes modifying some paths for other plugins, providing some methods to go to specific places in editors, or exposing its own serialized to others.

Web As communication with the survey database and its interface named Geode is of great importance, a dedicated Geode plugin was created. Instead of a typical editor, this plugin features its own *Chromium* based browser. The objective of this plugin is not only to render Geode's contents but also to cooperate with other survey software. Currently, whenever a file is downloaded, it is directly opened with the right plugin, some additional project files may be directly created on download, and using entry-points, output data can be easier transferred back to the database.

Customization

SurveyPad is highly customizable and can be adapted to users' needs. It provides preferences editable by users that define global tweaks to the entire user interface and all plugins. These include the option to auto-save the projects, editors' visibility, logs, text editor styles, and many others.

Moreover, all the plugins have their own dedicated settings that greatly influence their behavior. All these settings can be modified directly via SurveyPad when plugins are loaded.

FUTURE PROSPECTS

Besides already implemented functionalities, SurveyPad and its plugins are still upgraded and expanded with additional features.

In order to improve the plugin interoperability, the work to serialize/structurize the data coming from some survey software is undergoing. That means that besides the output text files some data could be structured and packed to a common format, for example JSON, that can be interpreted without any tailor-made parsers. This would simplify the development of some features and the analysis of the entire calculations, with less effort spent on developing more and more complex grammars.

Having easily accessible and well-formatted data, plugins meant for 2D/3D data visualization could be created. Since *Qt* provides such graphical modules, no additional external dependency would be required. This could be an additional tool for inspecting networks of points and for detecting potential errors.

In terms of web capabilities, ongoing developments test the possibility to communicate with Geode directly using REST API. This will open new possibilities that will not have to rely only on Geode's web interface. Moreover, this would allow progressive migration of some of the existing processing tools already implemented in Geode that should not be there per database design.

CONCLUSION

To solve the issue of simplifying the surveyor's data analysis workflow SurveyPad was created. It was designed with maintainability, interoperability, and ease of expansion in mind. This was achieved with its plugin-based architecture. Thanks to that, SurveyPad can govern all of the survey pieces of processing software via plugins with the standardized interface but without any restrictions in terms of their capabilities.

Moreover, a number of specialized features, not often present in other software, were created to further enhance the user experience. This covers advanced text editing, communication between plugins, interoperability with web interface Geode, and many other improvements. With the current architecture new integration possibilities appeared that will be investigated.

Overall, current positive feedback from users indicates that SurveyPad has already demonstrated that it is a functional and convenient tool used by surveyors on a daily basis. Despite its ongoing developments it has proven to be a stable application with evolving features and promising prospects in the future.

REFERENCES

[1] CERN. LHC Guide. https://cds.cern.ch/record/2255762.

[2] A.-V. Naegely, I. Iliev, and A. Bensahla Talet. Geode: a new database with apex. IWAA'16, ESRF, Grenoble, France, October 2016.

[3] Y.L. Theng and H.B.L. Duh. *Ubiquitous Computing: Design, Implementation and Usability: Design, Implementation and Usability*. Premier reference source. Information Science Reference, 2008.

[4] Riverbank Computing. Qscintilla, September 2022. https://riverbankcomputing.com/software/qscintilla.

[5] Scintilla. Scintilla and scite, September 2022. https://www.scintilla.org/.

[6] Noam Chomsky. Three models for the description of language. *IRE Transactions on information theory*, 2(3):113–124, 1956.

[7] Remi Ducceschi. Entry points python style for qt plugins, September 2022. https://github.com/remileduc/qt_plugin_entrypoint.

[8] The Qt Company Ltd. The meta-object system, September 2022. https://doc.qt.io/qt-6/metaobjects.html.