Inishell 2.0: Semantically driven automatic GUI generation for scientific models

Mathias Bavay[1], Michael Reisecker[1,3], Thomas Egger[2], and Daniela Korhammer[1]

[1]WSL Institute for Snow and Avalanche Research SLF, Flüelastrasse 11, CH-7260 Davos Dorf, Switzerland
[2]Egger Consulting GmbH, Hohenstaufengasse 7, 1010 Wien, Austria
[3]Alpine Software Michael Reisecker, Schiliftstraße 504, 5753 Saalbach, Austria

Correspondence: M. Bavay (bavay@slf.ch)

Abstract. As numerical model developers, we have experienced first hand how most users struggle with the configuration of the models, leading to numerous support requests. Such issues are usually mitigated by offering a Graphical User Interface (GUI) that flattens the learning curve. This requires however a significant investment for the model developers as well as a specific skill set. Moreover, this does not fit with the daily duties of model developers. As a consequence, when a GUI has been created – usually within a specific project and often relying on an intern – the maintenance either constitutes a major burden or is not performed. This also tends to limit the evolution of the numerical models themselves, since the model developers try to avoid having to change the GUI.

In this paper we describe an approach based on an XML description of the required numerical model configuration elements (that is, the data model of the configuration data) and a C++/Qt tool (Inishell) that populates a GUI based on this description on the fly. This makes the maintenance of the GUI very simple and enables users to easily get an up-to-date GUI for configuring the numerical model. The first version of this tool was written almost ten years ago and showed that the concept works very well for our own surface processes models. A full rewrite offering a more modern interface and extended capabilities is presented in this paper.

1 Introduction

1.1 Context

Numerical models can be defined as computational models designed to simulate and predict the behaviour of real-world or physical systems. As illustrated in Fig. 1, given a set of input data (for example meteorological measurements) and configuration parameters (for example the simulation timestep and spatial resolutions), the numerical model will produce a set of outputs, for example snow cover and hydrological response of a catchment after simulating the physical processes leading to snow cover development and runoff generation. Numerical models are very powerful tools now widely used in diverse fields,

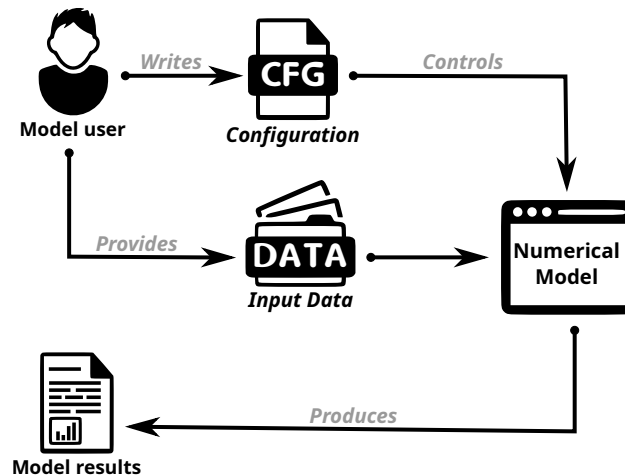



Figure 1. Numerical models from the user point of view

such as medicine, energy and environment, materials, industrial and defense as well as homeland security (Oden et al., 2006). Naturally they also see widespread use in research.

When using numerical models, one of the major issues for new users is the configuration of the model. Often the numerical
25 models are configured by ways of multiple configuration files filled with obscure configuration parameters, making a steep learning curve. Moreover, users tend to overlook even the best written documentation (Mendoza and Novick, 2005; Ceaparu et al., 2004) and resort to copying and tweaking example files. This is not satisfactory as it leads to under-performing simulations as well as large numbers of questions directed to the model developers. The usual solution is to implement a graphical user interface (GUI) for configuring the numerical models that also allows to predominantly show the most common settings while
30 keeping expert settings available at deeper levels. Unfortunately, this task hardly fits the job description of the modelers and is very time consuming because of the potentially large number of configuration parameters: for example the Snowpack model (Lehning et al., 2002) and its pre-processor MeteoIO (Bavay and Egger, 2014) – two of the numerical models Inishell was originally developed for – define more than 350 configuration keys. Developing a traditional GUI for such numerical models, where each input widget is manually laid out, would require a significant investment (Kennard and Leaney, 2010). Although
35 not specifically aimed at recent scientific software, (Myers and Rosson, 1992) found that on average 48% of the code in graphical applications was dedicated to the user interface, representing 50% of the development time in the implementation phase

and 47% of the maintenance time. Moreover as numerical models might evolve quite fast, new configuration options would frequently be added that would also require a rework of the GUI. This is hardly sustainable and leads to either out-of-date GUIs or no GUIs at all.

40 The choice of tools to develop GUIs for numerical models is less than satisfying in the long term. One possibility consists of using a Rapid Application Development environment (RAD, Spreitzhofer et al. (2004)). This is easy and can appropriately be assigned to an intern or a short term student. However this is risky in the long term since such RAD implementations are often proprietary and therefore dependent on the goodwill of its editor to maintain compatibility or even to keep the product running, potentially forcing the model developers to perform a full rewrite of the GUI. Another possibility is using standard toolkits

45 and languages to develop such an interface. This requires more investment and expertise from the developer but increases the long term availability of the product. However, maintaining the product also requires some technical knowledge that is usually not found in model developers. At the very least, it adds a considerable workload which may have to be put off until later. This practically means that upgrades (such as introducing new configuration options) will only happen when another intern or student with the proper skill set can be found and funded. Delegating this task to a temporary employee however loses

50 first-hand knowledge about the new options. Then the graphical user interface becomes a hindrance for the model itself since it prevents the fast deployment of new configuration options. Another possibility relies on Declarative User Interface Model (Da Silva, 2000) or Model-Based UI Development (Paterno, 1999), an approach that has been steadily maturing over the last several decades (Meixner et al., 2011). However the downside of this approach is that it can be highly theoretical and hard to understand by designers and developers (Bogdan, 2017).

55 Finally, numerical models are getting more and more modular, including through coupling of existing numerical models. This leads to modules that can be used standalone or within a wider numerical model. As such, there is no centralization of the configuration data that has been provided by the end user and a centralized data model for the configuration data is not possible: the main module does not have any insight into the data model of its sub-modules. Moreover, there is usually no explicit data model for the configuration data, it is only implicitly expressed through the source code as assumptions and

60 enforced requirements. It might also be explicitly laid out in the documentation (that must properly link to the documentation of each sub-module) and in a GUI (that must include the configuration options for all sub-modules) but they must then be kept synchronized with the implementation in the source code in order to be useful to the end user.

1.2 Requirements

Ideally model developers would like to offer a user friendly graphical interface for configuring their numerical model that re-
65 quires very little initial investment and expertise and where new configuration options are quickly deployed. This configuration interface should provide explanations of every configuration parameter, validate the user input (to avoid possible misconfigurations), easily integrate new options and output the complete configuration in a standard configuration file format. This GUI should also be able to transparently integrate the configuration options for each sub-module without requiring any duplication of efforts.

70   Keeping the concept of a configuration file is important since such models are often configured on one system and then sent to run on some clusters to perform the heavy duty computing. This file should be manually editable in order to allow for copy-pasting some of it between similar simulations (keeping in mind that several hundred lines might be copied that represent hours of carefully choosing the options), to be able to modify it with text terminals through remote sessions (as is typically the case when running on a computing cluster) or to generate at least some parts of the configuration with scripts (for example

75   to study the sensitivity of certain configuration parameters). In order to further improve the quality of the numerical modeling work and constrain the problem, numerical model developers are strongly encouraged to rely on a single configuration file for the whole model and all its sub-modules, including as much of the pre- and post- processing as possible. This has the advantage that a copy of the said configuration file is then a reproducible description of the numerical simulation that has been performed (Bavay et al., 2020a). It represents a fixed state and keeping a changelog of the configuration files enables investigations into

80   past simulations.

As a side effect, having an easy to maintain GUI at their disposal allows model developers to explicitly describe the data model of the configuration data, encourages them to document new features and even to avoid hard-coded values since making them a dynamic setting read from a user-editable configuration file is easy and quickly done.

2   Inishell overview

85   The Inishell Open Source software alongside numerical model design considerations is our technical answer to the previously laid out requirements. It is written in C++ with the Qt framework[1] as a way to provide a cross-platform GUI with native look and feel that can be reused for multiple numerical models and that is sustainable over many years. It aims to feel familiar to the end users while considerably lowering the required skill set and time investment for the model developers and also shifting support requests away from IT tasks to work directly concerning the models. It is Open Source under a GPLv3 license and

90   works on Linux, Microsoft Windows and Apple macOS among others.

2.1   Principles

On the numerical model side, in order to allow for high modularity of the numerical models with respect to their sub-modules, the configuration data is centrally read as key/value pairs of strings into a C++ map data structure (similar to a dictionary in Python, for example) but not processed any further. This data structure is then provided to each sub-module to extract and parse

95   its supported configuration keys. Therefore the data model is delegated to the sub-modules which enforce data types, ranges, validation and dependencies between configuration keys, keeping each sub-module independent of the others.

For the user provided configuration data, as it would not be feasible to support all possible configuration file syntax choices, a reasonable standard is enforced. The INI[2] informal standard has been chosen as it is a text format that is very easy to read and parse with various programming languages as well as for interoperability with existing numerical models. Its syntax is also

---

[1]https://www.qt.io/
[2]https://en.wikipedia.org/wiki/INI_file

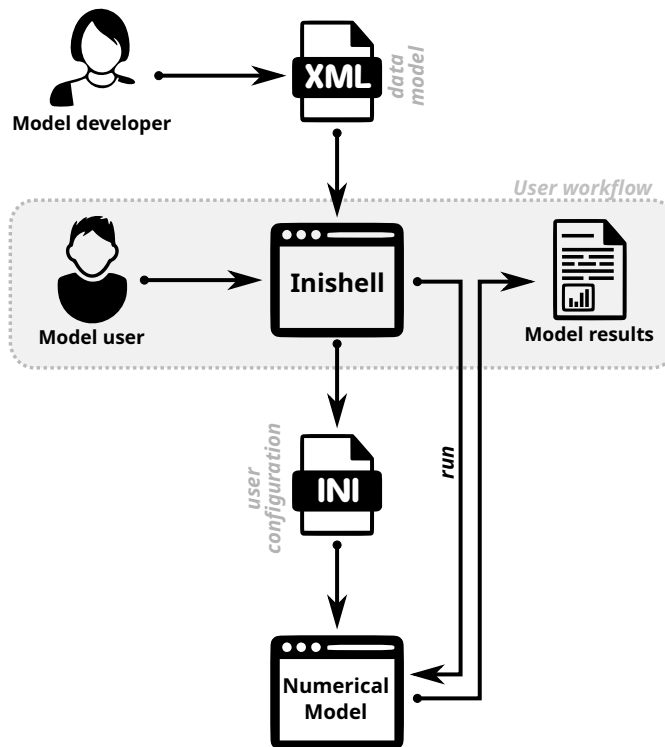



Figure 2. Inishell principle of operations

100 supported by many text editors, making manual edition convenient on multiple platforms. However its simplistic syntax can not contain enough information to define a data model and as it is created by the end user, all inputs coming from an ini file must be checked by the numerical model. It is described in details in section 3.1.

The Inishell software is a simplified derivative of Declarative User Interface Models that focuses on the data that has to be provided by the end user instead of the appearance of the GUI. The data model of the configuration data is explicitly defined in
105 an XML file that supports including other XML files to integrate the data models of various sub-modules together. The XML syntax defines for each configuration key the key name itself, the data type of the value that should be provided by the end user (including units) as well as a help text. Input validation is supported through data types, optional range checking, and by regular expressions. This is similar to the input validation provided by common JavaScript libraries such as Angular[3] or React[4]: data type, min, max, required or not, pattern.

110 The data model of the configuration data provided by the numerical model developer as an XML file is then used by the Inishell software to automatically populate its GUI that is presented to the end user to input the numerical model configuration data. Inishell enforces the user input validations and therefore the writing of a configuration file that the numerical model

---

[3]https://angular.io/api/forms/Validators
[4]https://react-hook-form.com

can rely upon to run (more details in section 3.2). The servicing of existing GUIs and the creation of new GUIs is therefore decoupled from the release cycle of Inishell itself.

115   The general principle of operation is shown in Fig. 2. The model developer does not look into low level interface attributes (exact positioning, complex layout of the GUI) but only provides a high level semantic description of the configuration parameters and delegates the GUI generation to Inishell, similarly to JSONForm[5] or Json-GUI (Galizia et al., 2019). This is then a higher level view of the GUI than in previous efforts such as XUL (XML User Interface Language, Goodger et al. (2001)) that is still focused on low level widgets or even UIML (User Interface Markup Language, Abrams et al. (1999)) that still keeps

120 low level widgets as basic building blocks. It can be best compared to Atomic Design (Frost, 2016) since it is also built around a hierarchical point of view but keeping in mind that here the focus is not the entry widget type but data semantics of the data that has to be retrieved from the user.

2.2   Overview of the general interface

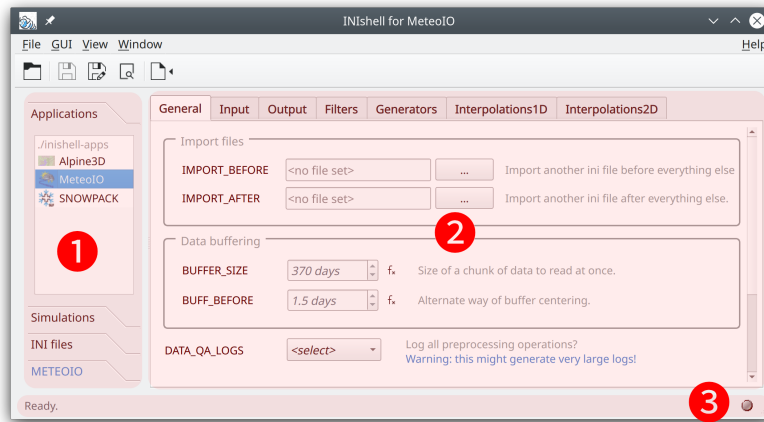



Figure 3. Overview of the Inishell software with the MeteoIO numerical model profile loaded (here on Linux).

    The overall interface (Fig. 3) is made of three areas, as well as a standard top menu bar. Area 1 controls the whole user

125 workflow: In the upper panel (Applications), the user selects which model he or she wants to configure. Below (Simulations), it is possible to open a preconfigured pair of numerical model profile and specific configuration file (i. e. a model with loaded settings ready to run). One drawer lower (INI files), it is possible to open an existing configuration file. Finally, for numerical models that support it, the lowest drawer (METEOIO, shown in blue) enables running the simulation and potentially opening the simulated results.

130   Area 2 in Fig. 3 contains all configuration widgets for the selected numerical model profile. This is where the end user fills a configuration file. Area 3 is a status bar that shows error messages or warnings (such as for missing mandatory configuration keys) or the status of a currently running simulation. Messages are also logged for unattended runs.

---

[5]https://jsonforms.io/

```ini
[General]                          ; entering the "General" section
#meteo data input settings         this whole line is a comment
BUFF_CHUNK_SIZE = 370              ; this is an inline comment
METEO           = SMET             ; value as string
METEOPATH       = ./input/meteo    ; a path is also a string
STATION1        = FLU2             ; providing two station IDs
STATION2        = FIR2

[Filters]                          ; entering the "Filters" section
TA::filter1     = min_max          ; namespace for key "filter1" is "TA"
TA::arg1        = 240  320         ; (here short for Air Temperature)

RH::filter1     = min_max          ; another "filter1" key, but in namespace "RH"
RH::arg1        = 0.01  1.2
```

Figure 4. Syntax of the INI file

In order to further encourage end users to rely on Inishell to configure and run their simulations, a text editor is offered within Inishell under the name *Preview Editor*. It is powered by Inishell's INI format parsing, and as such provides operations specif-
135 ically targeted to INI files in addition to more common text editor functionality and keeping snapshots of the file throughout the editing process. Hence, the Preview Editor incorporates several Inishell features into a text editor like every user will have seen and used while still minimizing classical user errors (such as by marking unrecognized or deprecated keys as unknown with syntax highlighting).

3 Implementation

140 ### 3.1 Supported INI file syntax

Although best practices have emerged that make the INI informal standard reasonably usable as a configuration file syntax, it is too loosely specified to be easily automatically generated and therefore has been defined more strictly for this work as well as extended to better suit the needs of numerical models.

The general format consists of a list of key/value pairs, delimited by an '=' sign. The values can be of type doubles, integers,
145 boolean (*true/false* or 0/1) or strings. It is possible to add comments: all characters following '#' or ';' will be considered to be comments until the end of the line is reached. The keys can be grouped by sections in order to bring more clarity and structure to the configuration file, each section being marked by a section name between square brackets. Spaces and tabs can be used freely between words (either keys or values). Each key must appear only once per section but the same key can appear in several sections: for example a time zone information can appear in an input and an output section.

150   In order to keep the uniqueness of the keys in each section while allowing semantically identical keys to coexist, several extensions have been defined. A first possibility is to simply add a number after the key, making it in effect unique but clearly showing the user that all these keys participate to the same concept. Another possibility is that a key may receive several values by providing the different values space-delimited after the equals sign. Finally, a weak concept of namespaces has been introduced: a key can be prefixed by a namespace so multiple keys belonging to different namespaces can coexist in the same

155 section. This makes it possible for example to declare keys for specific meteorological parameters by using the meteorological parameter abbreviation as namespace.

    A commented example of the syntax described above is given in Fig. 4.

3.2   General architecture

The hierarchical approach to interface design is seen both in the GUI itself and in the underlying architecture (where Inishell

160 mirrors the XML structure) and defines the roles of each contributor to the GUI for any particular numerical model. The atomic elements (*atoms* in Atomic Design) are the widgets provided by the Qt toolkit. These are never exposed to the model developer, instead they are grouped into higher level elements (*molecules* in Atomic Design) by Inishell for each parameter type in the XML file. In effect, by writing a succession of parameters belonging to sections in the XML file, the model developer sets up all parameters necessary for the configuration of a module of his or her model, distributed over one or more tabs in the GUI

165 that act as the next hierarchical level and are mapped to sections in the resulting INI file. These will then be grouped together under an application name that might also receive a workflow (step-by-step instructions to configure and run some model) and an icon. This is the highest hierarchical level as it matches a specific numerical model.

Atomic design	Qt	Inishell	Model developer
atom	widgets (textfield, combobox...)		
molecule		basic building block	
module	layout manager	create and populate the tabs	declare the parameters
application		multiple tabs, workflow & INI editing / writing	gather all parameters, describe the workflow

Table 1. Role distribution for Inishell

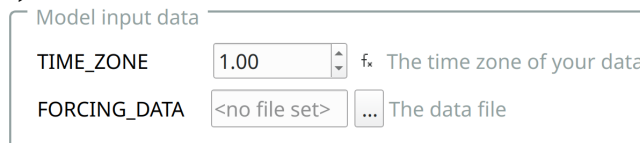
    This hierarchical approach is simplified by relying on two modularity constructs: parameter groups and includes. Parameter groups allow giving an internal name to any group of parameters. This internal name can then be referred to later on to call this

170 group one or multiple times. This is even more meaningful when used with the built-in inclusion system: an arbitrary number of files can be included and from them it is possible to only select the needed subset of parameters thanks to parameter groups.

Several applications sharing most of the same configuration keys for any subset of their configuration can then include one file that defines all possibilities and only call the parameter groups that are relevant. In fact it is recommended to heavily rely on this system for increased modularity and decreased verbosity. In the same way models that rely on other models (e. g. in the form of libraries) can simply include this lower level model and freely extend upon it.

3.3 Basic building blocks

Figure 5. Basic building block: integer entry

Inishell supports the following data types: strings, dates and times, paths to files and paths, decimal numbers, integral numbers and booleans, usually with several display options. Strings are less strictly defined as this type can accommodate free text entry or a selection among a preset list of choices (that can potentially be extended by the end user). Geographic coordinates are matched within strings through a regular expression that triggers the generation of an additional button that shows the provided coordinates on an online map. Strings can be validated by means of a regular expression, as well as through an expression parser to make them suitable for mathematical formulas.

For each data type, Inishell generates a low level entry widget prefixed with a label that shows the matching INI configuration key (or another, better suited label chosen by the model developer) and followed by a help text (that may also contain hyperlinks to a more exhaustive online documentation). Hence, Inishell manages several abstraction layers for the programmer and adequately adding and describing a model setting in the right place is now as easy as adding an XML text node without the need to recompile any software. Several properties for each INI configuration key can be declared. Among those, the XML property *optional* when set to *false* visually emphasizes the widget and displays a warning message when saving the file without setting it. In such a case, all the mandatory keys that have not been set by the end user will be highlighted, listed in a message, and the user can cancel saving. Manual styling of all of the used fonts is possible. Colors can be chosen freely with an RGB hexadecimal representation, but Inishell also offers a set of predefined colors with semantic names (such as *warning*, *info* ...) which have been designed to keep good visibility if the end user changes the GUI theme, for example when using the dark theme or system wide accessibility settings.

3.4 Grouping elements

a) XML declaration

```xml
<frame caption="Model input data">
    <section name="Input"/>
    
        <help>The time zone of your data</help>
    
    
        <help>The data file</help>
    
</frame>
```

b) Inishell GUI

c) INI entry

```ini
[Input]
TIME_ZONE = 1
```

Figure 6. Visually grouping elements together

The first grouping element is matched to an INI structure: sections. It is either expressly declared in the XML or indirectly as the basic building blocks can declare which section they belong to. In the GUI, this is represented by a tab, so all INI keys belonging to a given section will have their matching widgets appear in the same tab. The end user has an overview of all the sections with the list of tabs on the top of Inishell (Fig. 3, on top of area 2).

Another grouping element is available that does not match any INI structure: frames. A frame is used to graphically group basic elements that belong together, for example a set of configuration parameters all related to the same concept in the numerical model. A frame can have its own help text which can be convenient to describe in details the feature that is configured by the keys within the frame.

3.5 Templates

Some fragments of the INI configuration file might have to be repeated multiple times, for example to iterate over multiple input files or over meteorological parameters. In this case, a base key is defined (for example "STATION") and multiple versions derived from this base key will be generated on-demand as requested by the user (for example by clicking on a "+" button to generate "STATION1", "STATION2" ...). This lets the end user provide as many variants as necessary without having to hard-code the configuration keys for each variant. In the XML file, it is handled with a system of templates where the iterators are defined first (for example, as integral numbers or as a fixed list of strings) followed by the group of configuration keys containing a wildcard character. Inishell will then dynamically generate as many entry widgets (or groups) as asked by the end user and write all resulting INI keys in the output file.

a) XML declaration

```

    <help>File name for station number #</help>

```

b) Inishell GUI

STATION#	+	-	File name for station number #

No 1:	WFJ.smet	...

No 2:	DAV.smet	...

c) INI entry

```
[Input]
STATION1 = WFJ.smet
STATION2 = DAV.smet
```

Figure 7. Simple example of templates

3.6 Nested widgets

Some dedicated widgets offer the possibility to include more configuration options that will be shown only when a certain choice is selected by the user (Fig. 8). This allows offering more configuration options related to a given sub-module if the said sub-module has been enabled (for example, ticking a checkbox could show further options of the same INI section and so could the selection of specific list entries). This is a recursive process and allows for indefinite nesting.

3.7 Workflows

In order to allow the end user to run the numerical model from within Inishell, it is possible to declare the necessary workflow in the XML file. This includes command line programs as well as their command line options (based on the data types that are provided by the end user), directory views (for example to open the model results directory) or opening URLs (for example to open an online viewer). The terminal outputs of the applications started by Inishell are captured and shown in Inishell's main window with some basic syntax highlighting in order to highlight error messages or warnings.

3.8 Applications

Since multiple numerical models can be loaded into Inishell by opening their respective XML files, it is necessary to visually show which choices of models are available and easily change between them. This is achieved by providing some meta data of the applications' properties in the XML files that define the previously described XML elements for the configuration widgets (and potentially a workflow). An application will therefore consist of a name and an icon in the applications panel, several tabs with configuration options in the main panel and often a workflow to run the application.

a) XML declaration

```xml

    <section name="Input"/>
    <option value="SMET">
        
        
            <option value="TRUE"/>
            <option value="FALSE"/>
        
        
        
            <option value="TRUE"/>
            <option value="FALSE"/>
        
    </option>

```

b) Inishell GUI before combobox selection

c) Inishell GUI after combobox selection

Figure 8. Example of nested widgets (for clarity, the help texts have been removed). Please note that the METEO key uses an alternate label and is defined as mandatory. Once it is selected as SMET, more widgets appear including the METEOPATH that is then also mandatory.

Combining the features listed here (choosing an application from a list, optionally auto-loading an INI file and a coupled workflow), developers can set up a list of all their models' workflows and simulations. Inishell can then handle everything from configuring the model, running it and performing maintenance work (by executing user defined system commands) – all with the click of a button within one uniform GUI and without any programming necessary.

4 Discussion

Although fully automatic GUI generation by Declarative User Interface Models requires complex modeling (Machado et al., 2017; Meixner et al., 2011), Inishell fully automatically generates a GUI based on a simple, high level description of the data model. This has been made possible by restricting Inishell to the narrow and simple use case of configuring scientific numerical models, contrary to more generic approaches such as (Díaz et al., 2020). In this use case, the numerical models run from a static configuration file without feedback to the GUI other than textual information (such as progression indicators, warnings and errors) and no interactive coupling – the user can not change the configuration data while the numerical model is running and the numerical model can not change its configuration data (as is typically the case when simulating over a domain defined in time and space). This has several important consequences that lead to a great reduction in complexity. First, the palette of interaction patterns is reduced (Machado et al., 2017) to Create, Read, Update and Delete (CRUD, Martin (1983)) operations.

a) XML declaration
```
<workflow>
    <section caption="DEMO">
        <element type="label" caption="Start date:"/>
        <element id="start_date" type="datetime"/>
        <element type="label" caption="INI file:"/>
        <element id="ini" type="text" default="${inifile}"/>
        <element caption="Run DEMO" type="button">
            <command>my_demo -c %ini -b %start_date</command>
        </element>
        <element type="label" caption="Visualize results:"/>
        <element id="visualize" caption="Open niViz" type="button">
            <command>setpath(%outpath, ${key:OUTPUT::PATH})</command>
            <command>openurl(https://run.niviz.org)</command>
        </element>
        <element type="label" caption="Then drag your desired
                                      output file into niViz from below:"/>
        <element type="path" id="smetpath"/>
    </section>
</workflow>
```

b) Inishell Workflow

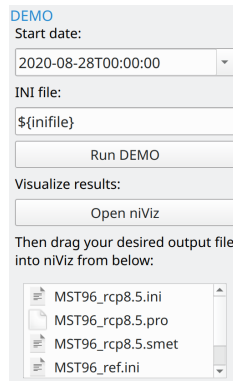



Figure 9. Example of a workflow: a few command line parameters must be provided by the user who can then run the numerical model and open a visualization application.

Then the reduction in application domain to a narrow scope allows reducing the required descriptive capabilities of the data model (Schaefer et al., 2006; Meixner et al., 2011) and data model description file and syntax (Galizia et al., 2019). This in turn makes the data model map to a very limited palette of input widgets (as shown in section 3.2) and the event model is even simpler (feedback stemming from input validation, hiding/showing elements based on another element's value). The focus is also not on the visual appearance of the GUI but on the data that has to be provided by the end user. Finally, the selection of supported platforms is restricted to traditional desktop computers, removing the need for one layer of abstraction (Paterno' et al., 2009). Inishell is therefore a practice-driven simplification of Declarative User Interface Models or Model Based UI Development to make this approach usable by non specialists similarly to other efforts such as (Galizia et al., 2019; Fardoun et al., 2018) or JsonForms for web forms.

As the Web is becoming the platform of choice for more and more complex tools, a web version of Inishell which could be integrated within a system processing the generated configuration files directly in the cloud certainly would have its benefits. However, a major negative feedback from some of the scientific numerical models that we develop was that the users had to open a terminal, go to the proper directory and run the numerical model from the command line. This has created many

support requests and frustration from the users. Due to sandboxing and obvious security reasons, running a local executable is not permissible from a web application, which is the main reason the implementation as a fat client was the preferred choice. Moreover as web technologies evolve very fast, the long term maintenance and evolution of web applications is a hurdle for research groups that must rely on external contractors for their development including trivial bug fixes. Furthermore, the interaction of a local web application with files in arbitrary locations on the system remains cumbersome.

The choice of file formats has been the result of compromises between ease of use and robustness for manual editing by the end users on one hand and expressiveness and compatibility with existing standards on the other hand. The legacy numerical models and tools have also weighted in for an easy transition to this new system and for a less risky migration. A similar project starting from scratch without any retro-compatibility issue would most probably rely on more modern and better defined standard file formats. Future developments of Inishell could benefit from supporting several output formats in order to generate configuration files for a wider range of scientific numerical models.

Since the first version of Inishell was written in the Java programming language in 2011, it is possible to draw some conclusions related to its real life impact. Overall it has worked well, allowing multiple numerical models to evolve freely without worrying about tedious redesigns of the GUI. Most of the additionally introduced configuration keys could be declared in Inishell in a matter of minutes and the possibilities offered by the XML elements recognized by Inishell have been mostly adequate. Support requests by end users of numerical models have dramatically dropped for users of Inishell since it was launched. However, some issues have been identified and addressed in the current version. First, as the Java environment is often not installed by default on personal computers anymore, it has started to cause more support requests related to the installation of Java as well as its configuration; the move to C++ is a response to this issue. Moreover, the original version of Inishell missed the possibility to run the numerical models directly from within its own interface and this has been identified as a major hindrance towards having more users rely on Inishell for their day to day simulations. This new capability has been brought through the expansion of the descriptive capabilities of the XML elements so Inishell now offers a fully self sufficient environment for configuring and running the numerical models that rely on it. This means that end users don't need to work through a combination of tools that tended to encourage them to manually tweak the configuration files (and therefore introduce errors) but find everything they need in one integrated package. This has also significantly improved the uptake of new numerical models features as end users now visually see new options in the GUI instead of having to read through many pages of documentation or detailed changelogs.

The new version has since been used in complex operational simulation toolchains with completely different numerical models than it was originally developed for. Merely by adhering to the INI syntax it was possible to adequately set up the models' parameters through Inishell, document them and offer an easy to use and familiar GUI to the people running the models.

5 Conclusions

Scientific numerical models require a large number of configuration parameters to operate that are generally quite complex to set up. Providing a Graphical User Interface (GUI) to set up such configuration parameters improves the control that the end users have over the numerical models beyond what a standard documentation would do. However, standard GUIs are very time consuming to program for these large numbers of configuration parameters and often require a skill set that is not found in such numerical models developers. By relying on an approach derived of Declarative User Interface Models and restricting itself to the narrow use case of scientific numerical models configuration (a low complexity use case), Inishell allows model developers to quickly define in an XML file the configuration parameters that must be provided by the users along with a few properties and then generate on the fly a GUI based on these definitions. The maintenance of the GUI solely consists of editing this XML file, for example to add new configuration parameters. Ten years after the first version of Inishell has been deployed in the field, this concept has globally worked well and has been efficient both from the end users point of view and from the numerical models developers point of view. Enforcing a well defined syntax and a single configuration file has also brought added benefits such as improved reproduciblility.

Code availability. The current version of Inishell is available from the project forge https://models.slf.ch/p/inishell-ng/ under the GNU General Public License v3.0 (GPL v3) licence. The exact version of Inishell presented in this paper is archived on envidat.ch (Bavay et al., 2020b).

Author contributions. M. Bavay lead the project from the beginning, contributed maintenance and development on all versions. He also wrote the bulk of the paper. M. Reisecker provided the bulk of the development of the new Inishell as well as maintenance since then and also contributed to the paper. T. Egger assisted with the implementation of the first version, helped maintain it over many years and contributed to the paper. D. Korhammer co-designed and implemented most of the original Inishell.

Competing interests. The authors declare that they have no conflict of interest

Acknowledgements. The authors are very thankful for the continued support of Charles Fierz and Michael Lehning who trusted us with our vision.

References

Abrams, M., Phanouriou, C., Batongbacal, A. L., Williams, S. M., and Shuster, J. E.: UIML: an appliance-independent XML user interface language, Computer networks, 31, 1695–1708, 1999.

Bavay, M. and Egger, T.: MeteoIO 2.4. 2: a preprocessing library for meteorological data, Geoscientific Model Development, 7, 3135–3151, 2014.

Bavay, M., Fiddes, J., and Østein Godøy: Automatic Data Standardization for the Global Cryosphere Watch Data Portal, Data Science Journal, 19, https://doi.org/10.5334/dsj-2020-006, 2020a.

Bavay, M., Reisecker, M., Egger, T., and Korhammer, D.: Inishell-2.0.4, https://doi.org/http://dx.doi.org/10.16904/envidat.194, https://www.envidat.ch/dataset/inishell-2-0-4, 2020b.

Bogdan, C.: Declarative interaction towards evolutionary user interface prototyping, in: IFIP Conference on Human-Computer Interaction, pp. 83–90, Springer, 2017.

Ceaparu, I., Lazar, J., Bessiere, K., Robinson, J., and Shneiderman, B.: Determining causes and severity of end-user frustration, International journal of human-computer interaction, 17, 333–356, 2004.

Da Silva, P. P.: User interface declarative models and development environments: A survey, in: International Workshop on Design, Specification, and Verification of Interactive Systems, pp. 207–226, Springer, 2000.

Díaz, E., Panach, J. I., Rueda, S., and Vanderdonckt, J.: An empirical study of rules for mapping BPMN models to graphical user interfaces, Multimedia Tools and Applications, pp. 1–36, 2020.

Fardoun, H. M., Tesoriero, R., Sebastian, G., and Safa, N.: A Simplified MbUID Process to Generate Web Form-based UIs., in: Proceedings of the 13th International Conference on Software Technologies (ICSOFT 2018), pp. 835–842, Science and Technology Publications, Lda, https://doi.org/10.5220/0006943908010808, 2018.

Frost, B.: Atomic design, Brad Frost Pittsburgh, https://atomicdesign.bradfrost.com/, 2016.

Galizia, A., Zereik, G., Roverelli, L., Danovaro, E., Clematis, A., and D'Agostino, D.: Json-GUI—A module for the dynamic generation of form-based web interfaces, SoftwareX, 9, 28–34, 2019.

Goodger, B., Hickson, I., Hyatt, D., and Waterson, C.: XML User Interface Language (XUL) 1.0, Tech. rep., Mozilla.org, https://www-archive.mozilla.org/projects/xul/xul, 2001.

Kennard, R. and Leaney, J.: Towards a general purpose architecture for UI generation, Journal of Systems and Software, 83, 1896–1906, https://doi.org/https://doi.org/10.1016/j.jss.2010.05.079, https://www.sciencedirect.com/science/article/pii/S0164121210001597, 2010.

Lehning, M., Bartelt, P., Brown, B., Fierz, C., and Satyawali, P.: A physical SNOWPACK model for the Swiss avalanche warning: Part II. Snow microstructure, Cold regions science and technology, 35, 147–167, 2002.

Machado, M., Couto, R., and Campos, J. C.: MODUS: Model-Based User Interfaces Prototyping, in: Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS '17, p. 111–116, Association for Computing Machinery, New York, NY, USA, https://doi.org/10.1145/3102113.3102146, https://doi.org/10.1145/3102113.3102146, 2017.

Martin, J.: Managing the data base environment, Prentice Hall PTR, 1983.

Meixner, G., Paternò, F., and Vanderdonckt, J.: Past, Present, and Future of Model-Based User Interface Development, i-com, 10, 2–11, https://doi.org/10.1524/icom.2011.0026, 2011.

Mendoza, V. and Novick, D. G.: Usability over time, in: Proceedings of the 23rd annual international conference on Design of communication: documenting & designing for pervasive information, pp. 151–158, 2005.

Myers, B. A. and Rosson, M. B.: Survey on user interface programming, in: Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 195–202, ACM, 1992.

Oden, J. T., Belytschko, T., Fish, J., Hugues, T. J., Johnson, C., Keyes, D., Laub, A., Petzold, L., Srolovitz, D., Yip, S., and Bass, J.: Simulation-based engineering sciences, Tech. rep., National Science Foundation, https://www.nsf.gov/pubs/reports/sbes_final_report.pdf, 2006.

Paterno, F.: Model-based design and evaluation of interactive applications, Springer Science & Business Media, 1999.

Paterno', F., Santoro, C., and Spano, L. D.: MARIA: A Universal, Declarative, Multiple Abstraction-Level Language for Service-Oriented Applications in Ubiquitous Environments, ACM Trans. Comput.-Hum. Interact., 16, https://doi.org/10.1145/1614390.1614394, https://doi.org/10.1145/1614390.1614394, 2009.

Schaefer, R., Bleul, S., and Mueller, W.: Dialog modeling for multiple devices and multiple interaction modalities, in: International Workshop on Task Models and Diagrams for User Interface Design, pp. 39–53, Springer, 2006.

Spreitzhofer, G., Fierz, C., and Lehning, M.: SN_GUI: a graphical user interface for snowpack modeling, Computers & geosciences, 30, 809–816, 2004.