Json-GUI - a module for the dynamic generation of form-based web interfaces

Antonella Galizia[a,*], Gabriele Zereik[a], Luca Roverelli[a], Emanuele Danovaro[a], Andrea Clematis[a], Daniele D'Agostino[a]

[a] *CNR-Institute of Appled Mathematics and Information Technologies "E. Magenes", via De Marini 6, 16149 Genova, Italy*
{zereik,roverelli,galizia,danovaro,clematis,dagostino}@ge.imati.cnr.it

Abstract

Json-GUI is an AngularJS front-end module that dynamically generates form-based web interfaces. Starting from a formal JSON configuration object describing a list of inputs, Json-GUI is able to build a form frame interface at runtime, with standard and personalized validation rules, giving the possibility to define constraints between input fields. Validated data are stored as Json objects or text files. Json-GUI has been exploited by scientific communities to effectively reduce the development and maintenance of customized user interfaces in science gateways. Moreover, Json-GUI can also be employed in the development of general-purpose Web forms.

Keywords: AngularJS, web form, science gateways

1. Motivation and significance

Computational science represents a broad field where advanced computing capabilities are exploited to understand and solve complex, interdisciplinary problems. Present technologies and infrastructures represent important enablers because of their support to large-scale sharing of software,

*Corresponding author

data, instruments, computing services, and other domain-specific resources [1]. Science gateways are integrated ecosystems that exploit web technologies to make the sharing easier and to shield users from low-level technological issues. Science gateways are domain oriented and the provided interfaces for workflow configuration are mostly based on end user knowledge elicitation. Most of the available toolkits and frameworks for the design of science gateways decouple front-end and back-end with API-based interfaces. With this approach, the gateway communities can focus their effort on the design of community-specific Graphical User Interfaces (GUI) [2]. However, the development of front-end solutions can be a challenging task for non-IT experts [3].

With this vision in mind, we developed Json-GUI, a front-end library composed by a set of reusable AngularJS[1] directives, that allows the dynamic generation of full-featured form-based web interfaces for AngularJS applications. Starting from a formal JSON[2] configuration object, Json-GUI simplifies and automatizes the design and the implementation of a standard web form; the tool includes added value features as validation, constraints and the straightforward use of geographic maps. Json-GUI improves the interaction with users in the elicitation of new requirements and allows rapdly and incremental implementation of GUI improvements supporting agile methodology [4]. The form produces as output a set of validated data stored as JSON objects or text files. In a science gateway context, the output text files can be customized to be used as configuration files to run models, therefore they can be passed and processed by any back-end technology.

Json-GUI has been employed in several scientific contexts [5, 6, 7]; fur-

[1]AngularJS Official site, https://angularjs.org

[2]http://www.json.org

thermore, due to its flexibility, Json-GUI can be employed in more general contexts, e.g. commercial tools and wherever it is necessary to define a form-based web interface.

The paper is organized as follow: in the next Section the scientific context and similar tools are analyzed; in Section 3 Json-GUI is described from logical, architectural and functionality points of view; in Section 4 we discuss two main experiences of the uses of Json-GUI to develop the form-based web GUIs of science gateways addressing the requirements posed by meteorological and astrophysical communities. Section 5 highlights the benefits and added value features of the tool, while the last Section concludes the paper.

2. Scientific and technological context

Recently, several tools have been developed with different levels of maturity and completeness. In the following we briefly give an overview of different possibilities currently available in this rapidly evolving field. Most of the tools are oriented to support web/business communities; they may provide appealing interfaces to define forms, potentially hide programming aspects, be deeply integrated with third party frameworks, natively implement services typically more oriented to a commercial usage.

json-editor[3] represents a simple but complete editor that starts from a JSON schema to generate a web form and gives back a JSON object with the fields and values filled though the form. No support is provided to define the JSON schema. Alpaca[4] provides a library of out-of-the-box JSON schema to define field types, controls, templates, etc. The library has to be used, through a text editor, to create the HTML file that will generate interactive

[3] http://jeremydorn.com/json-editor
[4] www.alpacajs.org

forms for web and mobile applications. Schema Form[5] is a set of AngularJS directives that, similarly to Alpaca, provides a set of out-of-the-box of JSON schema, but provides user-friendly interfaces to create the initial schema of the forms. JotForm[6] and <form.io>[7] instead allow to completely skip the manual first schema generation and manage this part autonomously through the use of drag-and-drop interfaces and services.

Most of the cited tools support many types of parameters, integrate valuable external services, e.g. Paypal or Braintree payment, and support the possibility to extend the parameters/services natively provided. All tools implement validation rules with different levels of complexity, from basic to customized validation logic, but none of them allows the definition of complete custom constraints cross-checking of a set of values coming from different form fields. Moreover, being designed for general purpose applications, such tools lack the possibility to define markers and geographical areas on a map.

There is no evidence of the adoption and the exploitation of the above mentioned tools by the scientific community that achieved few benefits from the development of these interesting softwares. Json-GUI represents an attempt made to cover this gap and, although somewhere simplifies features with respect to the previous tools, it has proved its effectiveness in several scientific contexts: it has been employed to develop the science gateway of the EXTraS project [5] for the astrophysics community, for the refactoring of a science gateway for hydro-meteorological community [7] and, more generally, to *dress* Airavata, a powerful middleware supporting the development of

[5] http://schemaform.io/

[6] www.jotform.com

[7] form.io

solid science gateways, together with the EasyGateway toolkit [6]. In these projects, Json-GUI was exploited to develop the GUI to configure model runs as well as to generate configuration files that have been used by the specific software available for model execution.

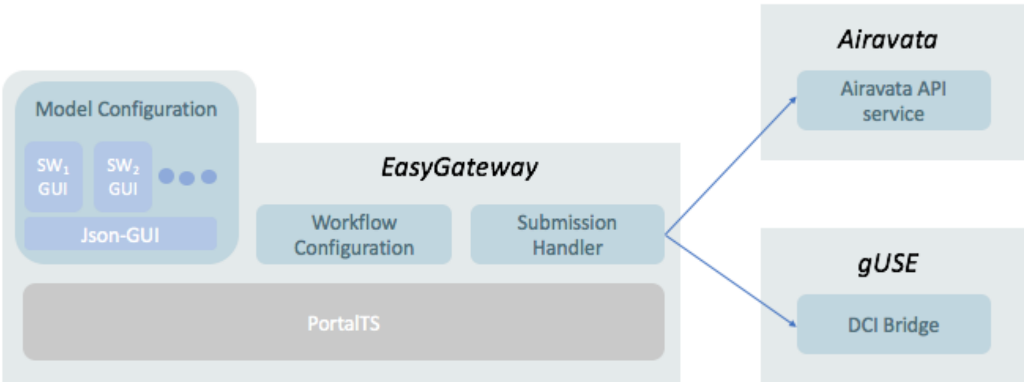


Figure 1: The architectural approach to integrate Json-GUI in a science gateway.

The integration of Json-GUI within a science gateway can be obtained smoothly because only the model configuration component leverages on Json-GUI. The existing submission handler component of the science gateway in fact is provided with data collected through the GUI, i.e. parameters to configure the model run, and it can run the model without modification. This architectural schema is depicted in Figure 1 and it has been discussed in details in [6].

3. Software description

Json-GUI generates at runtime a complete form-based web GUI that a user can exploit to insert heterogeneous values. The fields of the form and related customized rules are defined by manipulating an array of parameters, actually a JSON object. The input data collected through the form are stored as a JSON object that can be converted in a text file with an user-defined format. Completely integrated with Bootstrap[8] and based on responsive technologies, Json-GUI suitably addresses also mobile experiences while implementing a model-view-controller pattern.

Aligned with agile methodology and mockups [8, 9], Json-GUI allows a flexible approach to requirements and quick user-feedbacks, and reduces the time to deploy through cycles of interaction with users and incremental refinements of the GUIs. The development phase converges in few iterations of elicitation of domain specific knowledge and integration in user interfaces, i.e. the Web form GUI built through Json-GUI. The logical phases of this process are schematized in Figure 2.

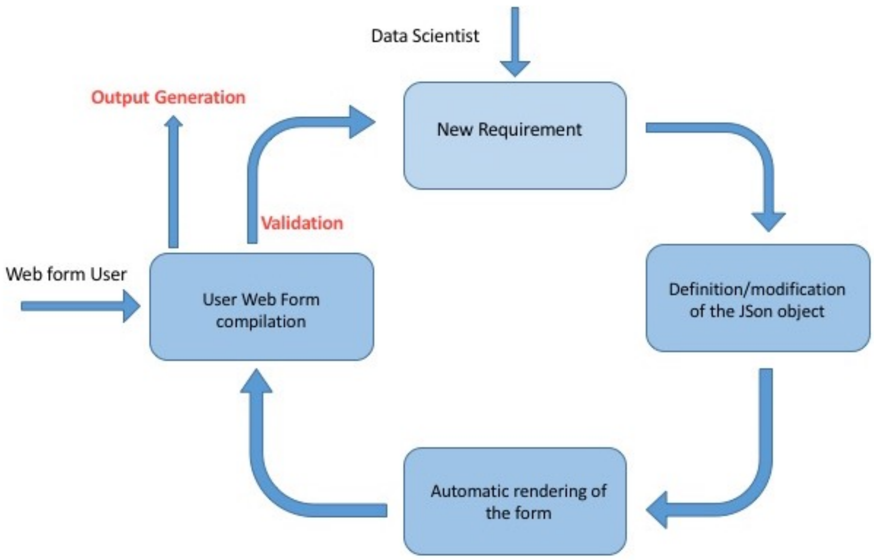


Figure 2: A logical schema of the Json-GUI usage.

Starting from the interaction with scientists, a first round of requirements are elicited and the definition of the JSON object is derived. In this phase, the

[8] https://getbootstrap.com

main actors involved are data scientists and Web form users. At this point, Json-GUI automatically generates the Web form corresponding to the JSON object, and users/scientists can fill in the values corresponding to the defined fields. Once the Web form is compiled, Json-GUI generates the output: a JSON object that can be possibly customized and used for the final aim. The generated output is a generic Json object, and, thus, it is ready to be processed by any middleware, workflow manager or local scheduler.

If the form is in a validation phase, the interaction among scientists and the Json-GUI user can continue to elicit more requirements, modify the JSON object and lead to the correct Web form. Also thanks to model-view-controller pattern at the base of the tool, Json-GUI speeds up this phase enabling a run-time visualization of the Web form, reducing the duration of iterations for the elicitation/integration of derived information and consequently the development time of the final GUIs.

Since the definition of the input for the generation of the web form could become a bit challenging, we developed a graphical tool to build the corresponding JSON object, called Json-GUI-Builder[9]. Through a simple interface, the Builder completely supports developers, i.e. Json-GUI users, in the definition of parameters and related validation, constraint and condition rules. The Builder is provided as separated tool since it could be also used autonomously, i.e. to define any type of JSON object, and no dependencies are actually implemented among the two tools. However, Json-GUI without the Builder comes less interesting and the combination of the two tools represents an added value for both. In Section 4, two examples of Json-GUI-Builder graphical user interface are reported.

[9]https://github.com/portalTS/Json-gui-builder

Form Fields

The core of the input object consists in an array of *parameters*, where each element defines (and renders) a single input field of the form. The possible input forms are: **integer** and **float** respectively generating a field for the specification of an integer and a float number; **datetime** generating fields for the specification of a date, including hours and minutes; **select** generating a combo box to select a value among the available ones; **text** generating a plain text input field; **domains**, generating a geographical map where rectangular domains and single markers can be drawn; **fileupload** defining an input box to upload one or more files.

Json-GUI offers high level features to enrich the form interface by defining:

- **Validation checks** - each parameter type has internal format validation, e.g. float and integer types have a built-in number format verification. Moreover, it is possible to add a custom validation for the specification of a behavior: e.g. a user may define a datetime input valid if it predates a specific date - the 1st January 1970.

- **Constraint rules** - since parameter values may mutually influence their behavior, constraints among different inputs can be implemented: if a time range has to be fixed, it is possible to set the "Start date" parameter value valid only if predates the "End Date" parameter value. This gives Json-GUI the potential to specify all standard constraints of a classic HTML5 form based interface.

- **Conditions** - Json-GUI offers the possibility to specify a condition (constant or depending on the value itself) to activate/deactivate parameters in the input form. This permits to enrich the form interface with a dynamic behavior when managing, for example, `Select` and

`Domain` parameters. A common example for `Select` parameter can be a form for online payment, Json-GUI allows to present different form fields depending on the value of a *payment method* field: if the selected value is “Paypal”, the GUI presents fields for “Paypal” login, with a Credit Card value, the GUI presents fields for credit card configuration (e.g. the credit card number, CVV, name and surname of the owner), and so on. The same level of dynamism is ensured when considering the `Domain` parameter, since the number of geographical domains relies on the user interaction and is unknown a-priori: depending on the number of domains that a user draws, the GUI can display different form fields and information. For example, in Figure 4 three geographical domains are considered, and the related geographical coordinates are displayed for each domain.

As standard behavior implemented by Json-GUI, if one of the rules/checks described above is violated, it will be not possible to submit the form and the output will not be generated. A custom message can be displayed if specified during the definition of the related parameter. Examples are reported in the remaining of the Section and in Figure 8.

Software Architecture

Json-GUI presents a two-level software architecture, schematized in Figure 3. The higher level, named *Form*, is composed by the Web form GUI automatically rendered from the JSON object, equipped with its overall logic and behavior. This includes the validation checks among parameters and the collection of each value to build the overall output, i.e. couples of parameters and corresponding values possibly stored in a text file following an user-defined format. The lower level, named *Fields*, is represented by the

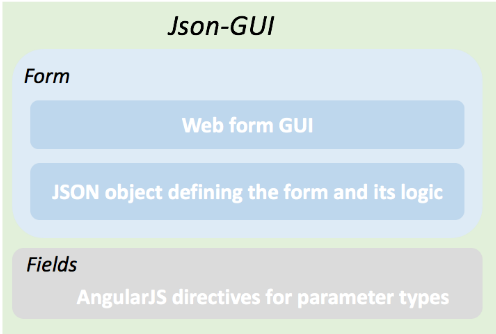


Figure 3: A logical schema of the Json-GUI usage.

AngularJS directives defining each parameter type. This level defines the individual behavior of the form fields, including the internal validation. Each validation rule can be general-purpose or specific.

Json-GUI is designed to be easily installed, extended or customized[10]. In particular, since the tool is open-source and developed with free technologies, a user can modify the css default settings and use the preferred css files thus to change how the form is rendered. Furthermore, a user can render his/her own customized fields by adding the definition of the logic for the new field; similarly, a user can extend the Json-GUI Builder to have the Builder support.

Software Functionalities

The basic element of the JSON object, input of Json-GUI, is a `parameter` that contains the value and all conditions that apply on it. Each parameter of the Json-GUI object has the following structure:

```
parameter: {
  value: {type: "parameterType"},
```

[10]https://github.com/portalTS/json-gui/wiki

```
    displayName: {type: "string"},
    dbName: {type: "string"},
    isValid: {type: "string"},
    parameterType: {type: "enum('float', ..., 'fileupload')"},
    parameterCategory: {type: "integer"},
    computedResult: {type: "string"},
    dependencies: [{type: "string"}, ...],
    required: {type: "boolean"},
    editable: {type: "boolean"},
    description: {type: "string"}  }
```

The *displayName* property defines the name of the parameter to be displayed in the interface, while the *dbName* is a unique identifier used internally. The *parameterType* defines the type to be specified among the ones supported. The *parameterCategory* property allows to logically group parameters in the form, e.g. by appearing in the same tab. The parameter can also be marked as *required*, it is possible to specify if the default *value* can be *editable* or not. The *description* property contains a text shown in an info box, and can be used as hint to the user. The *dependency* property is an array containing the references to the parameters on which the current parameter depends. These are the parameters that shall be used within the *isValid* property. This property is a string containing a Javascript function body to possibly define custom validations. The following is an example, where also a custom message is set for invalid condition:

```
isValid : "if(parameter.value < dependencies['dep_1'].value) {
    isValid.valid= false; isValid.message='custom error message';}"
```

The *computedResult* property defines a Javascript function meant to perform a final computation in order to (possibly) refine the value before the form submission. An example is the following:

```
computedResult: "return parameter.value/1000;"
```

Please note that the *computedResult* property allows a further customization for the value of the single fields; this is extremely useful when a specific format is required, e.g. datetime parameters formatted in a different standard or a specific projection for a geographical domain parameter.

4. Illustrative Examples

A valuable example is presented by the form field `Domain` defined to support the hydro-meteorological community in the configuration of the Weather Research and Forecasting, WRF[11] Model. The possibility to draw a geographical domain by using a graphical map has been actually acknowledged by scientific community [10]; for this reason, the `Domain` input type has been implemented with the integration of the Google Map JavaScript library. Furthermore, meteorological models usually enable the definition of more than one domains, that can be nested or not: nest is a finer-resolution model run, that can be embedded simultaneously within a coarser-resolution (parent) model run, or run independently as a separate model forecast. The first case, depicted in Figure 4, represents nested domains. For this reason, the `Domain` is enhanced with the possibility (for the user) to draw up to three rectangles, each one representing a geographical domain, and a constraint to permit the drawing of nested domains has been defined.

Figure 5 presents a sample code related to the hydro-meteorological science gateway [7] and leading to the configuration depicted in Figure 4: the

[11] http://www.wrf-model.org

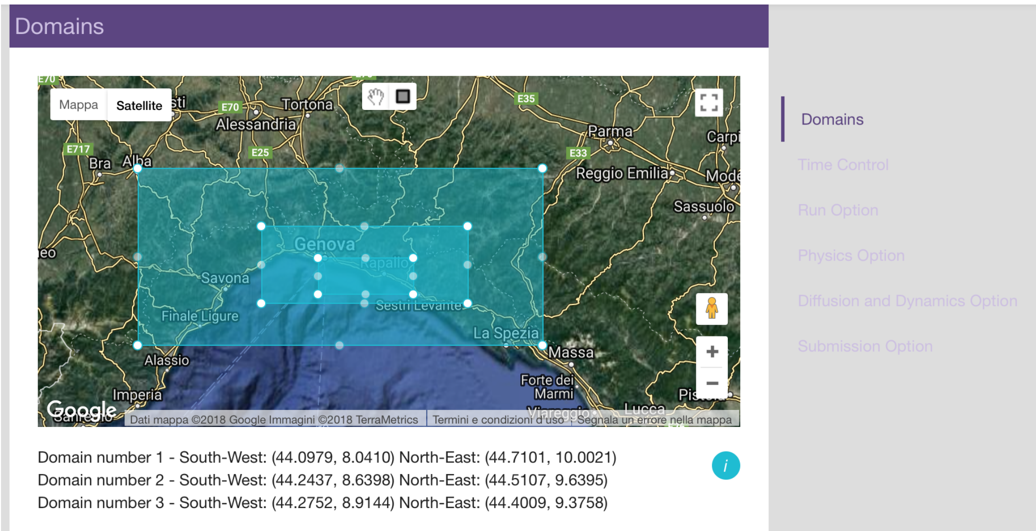


Figure 4: An example of GUI to draw up to three domains exploiting a Google map.

parameter `maxDomains` limits to three the maximum number of drawable domains and the parameter `onlyNested` permits to draw domains only inside a single parent domain. In Figure 6, an example of Json-GUI Builder interface corresponding to the `Domain` parameter is shown.

Another significant example is provided in Figure 7, that shows sample code related to the form field `Time_Interval_Selection`, representing one of the input of the transient analysis tool provided by the EXTraS science gateway. A wide diversity of astrophysical phenomena - from stars to supermassive black holes - are characterized by flux and spectral changes on time scales, ranging from a fraction of a second to several years. Current observing facilities subdivide an observation in a set of images, with a time resolution of the order of 1 sec. or shorter. In particular the transient analysis is based on the use of two alternative subdivision strategies, i.e. the use of fixed time intervals or variable intervals based on the Bayesian blocks algorithm. Therefore the user can select only one method and, consequently, the form field depends on the parameter named `Time_Interval_Selection_Bayesian`, be-

```
▼ Object
  type: "meteo"
  ▶ parametersCategories: Array [2]
  ▼ parameters: Array [3]
    ▼ 0: Object
      description: "Select up to three domains on which run the simulation"
      editable: true
      namelistName: "domains"
      allowMarkersOutDomains: true
      ▶ required: Object
      drawDomains: true
      drawMarkers: false
      maxMarkers: 0
      maxDomains: 3
      onlyNested: true
      mapZoom: 8
      ▶ center: Object
      parameterType: "domains"
      ▶ value: Object
      isValid: ""
      computedResult: "(function(){return true;}())"
      unremovable: true
      ▶ dependencies: Array [0]
      parameterCategory: 0
      dbName: "domain1466505616682"
      displayName: "Domains"
    ▶ 1: Object
    ▶ 2: Object
```


Figure 5: The Json-GUI parameters for automatic building of the Domains form field.

cause one and only one of them must have the "no" value. This is specified with the *dependencies* and *isValid* properties. Figure 7 shows also the error message raised in the GUI when the condition related to the parameters are not verified. In Figure 8, an example of Json-GUI Builder interface corresponding to the `Time_Interval_Selection` parameter is shown.

5. Impact and sustainability

Json-GUI represents a step towards closing the gap between the high level and low level layers of a science gateway, represented respectively by the community-specific GUI and the general-purpose middleware plus the computational infrastructure. Most of the available framework to develop science gateways do not provide a suitable support for the definition of cus-

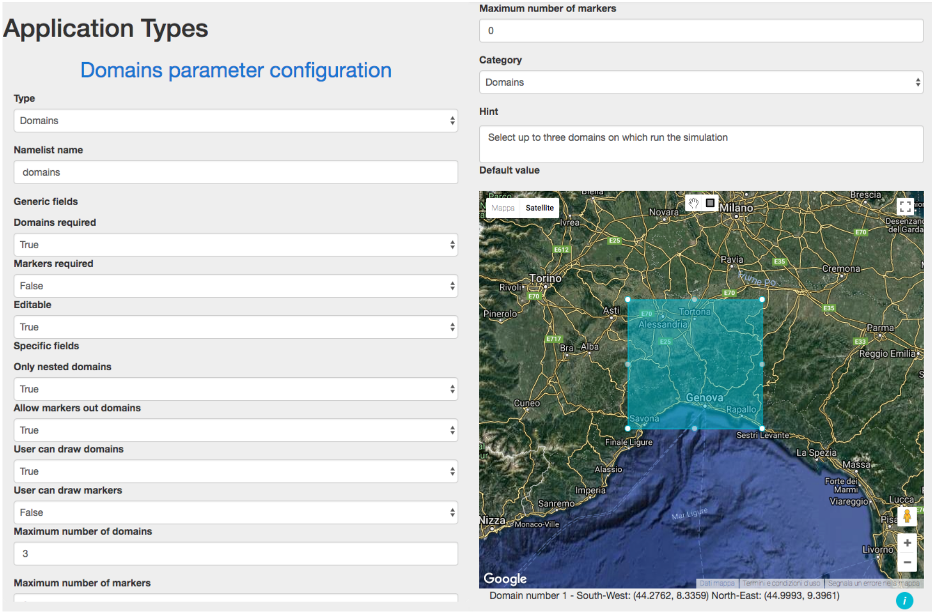
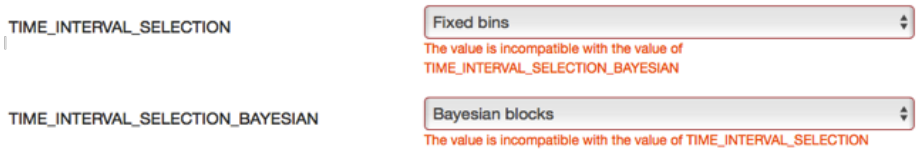


Figure 6: An example of parameter definition with Json-GUI-Builder.




```
{
  "required": true,
  "parameterType": ”select",
  "values": [”Fixed bins", "no"],
  "displayName": "TIME_INTERVAL_SELECTION",
  "dependencies": ["time_interval_selection_bayesian"],
  "isValid": if(dependencies['time_interval_selection_bayesian'].value
      =='bayesian blocks' && parameter.value!='no') {
      isValid.valid= false; isValid.message = 'The value is
      incompatible with Time_Interval_Selection_Bayesian';}"}
...
```

Figure 7: An example of parameter and consistency check definition with Json-GUI.

tomized GUI [7]. This may be challenging for non-IT communities, and a wrong selection of the front-end technology, combined with frequent developer turnover, can represent a major issue for the gateway sustainability [3]. Json-GUI definitely accomplishes this task while adding valuable features.

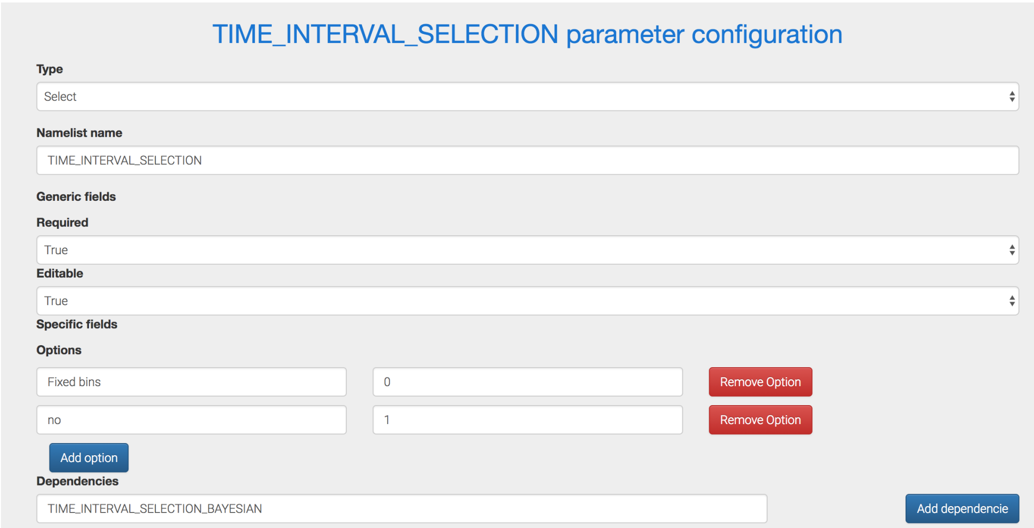


Figure 8: An example of parameter definition with Json-GUI-Builder.

Actually, Json-GUI allows the dynamic generation of web forms without the need to write any line of code. However this does not limit its expressiveness. The possibility to define customized rules on/among parameters in facts gives Json-GUI the potential to specify all standard constraints of HTML5 forms. The possible complexity in the definition of parameters rules are delegated to the Json-GUI-Builder, therefore again, this task does not suppose specific programming expertise. By contrast, more expert users could extend the tool to address their requirements since Json-GUI is open-source, based on widespread technologies and based on modern architectural pattern.

Furthermore, since user interfaces are dynamically generated starting from a JSON Object, it is possible to modify a web form interface on the fly by simply modifying the object without the need to re-deploy or restart any service. The resulting faster development cycle is very relevant in research fields relying on software tools developed (and frequently updated) by the community. Of course, such reduction has an impact also in terms of costs, thus becoming appealing in a general-purpose context.

Focusing on the added value features, the most valuable are constraints and conditions. The consistency check among parameters supports the proper configuration of experiments and, performed before the actual execution of the models, avoids the waste of CPU time due to execution of a misconfigured experiment. Also the possibility to draw geographical domains has been actually appreciated in the scientific community, and a great effort as been dedicated to this point, as outlined in Section 4.

And last but not least, Json-GUI effectively supports the creation of configuration files that can be directly ingested by target applications. Validated data collected through the generated form interfaces in fact can be stored as Json object or text file, e.g. as classical key-value format, but it is possible to define further customization to match the expectations of the models/applications. A user can develop and override any standard behavior of the generation phase: a transformation function can be defined for each field as well as for the final configuration file. This file can be used by the specific tools in charge for application execution; the actual submission can then be performed by the science gateway services, as described in [6].

As for software sustainability, this represents an open problem that may strongly affect the usefulness of new software tools. Json-GUI has the potentiality of satisfying most of the features requested to define software sustainability [11]. User interfaces developed using Json-GUI are: 1) *easy to maintain* because no specific programming expertise are required, without limiting their expressiveness. Furthermore they support a flexible approach to requirements and quick user-feedback and fast refinements; 2) *easy to evolve* because they are based on technologies and an architectural pattern that separate logic and presentation layers. This supports the possibility to simply implement customized solutions; 3) *able to fulfill their aim in a* *dynamic environment* since it is possible to easy adapt them to changing requirements.

6. Conclusions

We presented Json-GUI, an AngularJS front-end module which allows to quickly create form-based web interfaces. The module supports the export of the parameters in structured data files, which are often used for configuring complex experiments. The tool demonstrated its effectiveness a) in supporting users for the configuration of scientific experiments, where it is important to keep consistency among the inserted values, and b) in supporting non-IT experts for the design of such complex interfaces. Due to the successful user experience gained with two communities, we plan further effort to improve the visibility of tool and to engage other scientific communities.

References

[1] G. Andronico, V. Ardizzone, R. Barbera, B. Becker, R. Bruno, A. Calanducci, D. Carvalho, L. Ciuffo, M. Fargetta, E. Giorgio, et al., E-infrastructures for e-science: a global view, Journal of Grid Computing 9 (2) (2011) 155–184.

[2] P. Kacsuk, Science gateways for distributed computing infrastructures, Springer International Publishing. doi 10 (2014) 978–3.

[3] K. A. Lawrence, M. Zentner, N. Wilkins-Diehr, J. A. Wernert, M. Pierce, S. Marru, S. Michael, Science gateways today and tomorrow: positive perspectives of nearly 5000 members of the research community, Concurrency and Computation: Practice and Experience 27 (16) (2015) 4252–4268.

[4] J. M. Rivero, J. Grigera, G. Rossi, E. R. Luna, F. Montero, M. Gaedke, Mockup-driven development: Providing agile support for model-driven web engineering, Information and Software Technology 56 (6) (2014) 670 – 687. doi:https://doi.org/10.1016/j.infsof.2014.01.011.
URL http://www.sciencedirect.com/science/article/pii/S0950584914000226

[5] D. D'Agostino, L. Roverelli, G. Zereik, G. L. Rocca, A. D. Luca, R. Salvaterra, A. Belfiore, G. Lisini, G. Novara, A. Tiengo, A science gateway for exploring the x-ray transient and variable sky using egi federated cloud, Future Generation Computer Systems-doi:https://doi.org/10.1016/j.future.2017.12.028.
URL http://www.sciencedirect.com/science/article/pii/S0167739X17310051

[6] A. Galizia, L. Roverelli, G. Zereik, E. Danovaro, A. Clematis, D. D'Agostino, Using apache airavata and easygateway for the creation of complex science gateway front-end, Future Generation Computer Systemsdoi:https://doi.org/10.1016/j.future.2017.11.033.
URL http://www.sciencedirect.com/science/article/pii/S0167739X17310671

[7] D. D'Agostino, E. Danovaro, A. Clematis, L. Roverelli, G. Zereik, A. Galizia, From lesson learned to the refactoring of the drihm science gateway for hydro-meteorological research, Journal of Grid Computing 14 (4) (2016) 575–588.

[8] C. D'Souza, V. Deufemia, A. Ginige, G. Polese, Enabling the generation of web applications from mockups, Software: Practice and Experience 48 (4) 945–973. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.2559, doi:10.1002/spe.2559.
URL https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2559

[9] C. Torrecilla-Salinas, J. Sedeño, M. Escalona, M. Mejías, Agile, web engineering and capability maturity model integration: A systematic literature review., Information and Software Technology 71 (2016) 92 – 107. doi:https://doi.org/10.1016/j.infsof.2015.11.002.
URL http://www.sciencedirect.com/science/article/pii/S095058491500186X

[10] E. Danovaro, L. Roverelli, G. Zereik, A. Galizia, D. DAgostino, G. Paschina, A. Quarati, A. Clematis, F. Delogu, E. Fiori, A. Parodi, C. Straube, N. Felde, Q. Harpham, B. Jagers, L. Garrote, L. Dekic, M. Ivkovic, O. Caumont, E. Richard, Setting up an hydro-meteo experiment in minutes: The drihm e-infrastructure for hm research, in: 2014 IEEE 10th International Conference on e-Science, Vol. 1, 2014, pp. 47–54. doi:10.1109/eScience.2014.40.

[11] C. Venters, C. Jay, L. Lau, M. K. Griffiths, V. Holmes, R. Ward, J. Austin, C. E. Dibsdale, J. Xu, Software sustainability: The modern tower of babel, in: Proceedings of the Third International Workshop on Requirements Engineering for Sustainable Systems co-located with 22nd International Conference on Requirements Engineering (RE 2014), Vol. 1216, RWTH Aachen University, 2014.

Required Metadata

Current code version

Nr.	Code metadata description	Please fill in this column
C1	Current code version	1.1.3
C2	Permanent link to code/repository used for this code version	https://github.com/portalTS/json-gui/releases/tag/1.1.3
C3	Legal Code License	Apache License 2.0
C4	Code versioning system used	git
C5	Software code languages, tools, and services used	Javascript, HTML, CSS, AngularJS, Bootstrap
C6	Compilation requirements, operating environments & dependencies	AngularJS, Bootstrap, JQuery
C7	If available Link to developer documentation/manual	https://github.com/portalTS/json-gui/wiki
C8	Support email for questions	gabrielezereik@gmail.com

Table .1: Code metadata (mandatory)

AUTHORS DECLARATION OF INTEREST

We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome.

Signed by all authors as follows:

[LIST AUTHORS AND DATED SIGNATURES ALONGSIDE]

Antonella Galizia

Gabriele Zereik

Luca Roverelli

Emanuele Danovaro

Andrea Clematis

Daniele D'Agostino

Json-GUI - a module for the dynamic generation of form-based web interfaces

Gabriele Zereik[a,*], Luca Roverelli[a], Antonella Galizia[a], Emanuele Danovaro[a], Andrea Clematis[a], Daniele D'Agostino[a]

[a] *CNR-Institute of Appled Mathematics and Information Technologies "E. Magenes", via De Marini 6, 16149 Genova, Italy*
{zereik,roverelli,galizia,danovaro,clematis,dagostino}@ge.imati.cnr.it

Abstract

Json-GUI is an AngularJS front-end module that dynamically generates form-based web interfaces. Starting from a formal JSON configuration object describing a list of inputs, Json-GUI is able to build a form frame interface at runtime, with standard and personalized validation rules, giving the possibility to define constraints between input fields. Validated data are stored as Json objects or text files. Json-GUI has been exploited by scientific communities to effectively reduce the development and maintenance of customized user interfaces in science gateways. Moreover, Json-GUI can also be employed in the development of general-purpose Web forms.

Keywords: AngularJS, web form, science gateways

1. Motivation and significance

Computational science represents a broad field where advanced computing capabilities are exploited to understand and solve complex, interdisciplinary problems. Present technologies and infrastructures represent important enablers because of their support to large-scale sharing of software,

*Corresponding author

data, instruments, computing services, and other domain-specific resources [1]. Science gateways are integrated ecosystems that exploit web technologies to make the sharing easier and to shield users from low-level technological issues. Science gateways are domain oriented and the provided interfaces for workflow configuration are mostly based on end user knowledge elicitation. Most of the available toolkits and frameworks for the design of science gateways decouple front-end and back-end with API-based interfaces. With this approach, the gateway communities can focus their effort on the design of community-specific Graphical User Interfaces (GUI) [2]. However, the development of front-end solutions can be a challenging task for non-IT experts [3].

With this vision in mind, we developed Json-GUI, a front-end library composed by a set of reusable AngularJS[1] directives, that allows the dynamic generation of full-featured form-based web interfaces for AngularJS applications. Starting from a formal JSON[2] configuration object, Json-GUI simplifies and automatizes the design and the implementation of a standard web form. Json-GUI reduces the development time, includes added value features as validation and constraints while supporting an agile methodology and map based user interfaces. The form produces as output a set of validated data stored as JSON objects or text files. In a science gateway context, the output text files can be customized to be used as configuration files to run models, therefore they can be passed and processed by any back-end technology.

Json-GUI proved its effectiveness in several scientific contexts: it has been employed to develop the science gateway of the EXTraS project [4] for the

[1]AngularJS Official site, https://angularjs.org
[2]http://www.json.org

astrophysics community as well as to *dress* Airavata, a powerful middleware supporting the development of solid science gateways [5]. Moreover, Json-GUI has been used for the refactoring of a science gateway for hydro-meteo community [6]. In these projects, Json-GUI was also exploited to generate configuration files that have been used by the specific tools available for model execution. Due to its flexibility, Json-GUI can be employed in more general contexts, e.g. commercial tools and wherever it is necessary to define a form-based web interface. Recently, several tools have been developed with different levels of maturity and completeness: json-editor[3], <form.io>[4], Alpaca[5], JotForm[6]. Most of the cited softwares support many types of parameters, as color-picker, and integrate valuable external services, e.g. Paypal or Braintree payment. Furthermore, all tools implement validation rules with different levels of complexity, from basic to customized validation logic, but none of them allows the definition of complete custom constraints cross-checking of a set of values coming from different form fields. Moreover, being designed for general purpose applications, such tools lack the possibility to define markers and geographical areas on a map.

2. Software description

Json-GUI generates at runtime a complete form-based web GUI that a user can exploit to insert heterogeneous values. The fields of the form and related customized rules are defined by manipulating an array of parameters, actually a JSON object. The input data collected through the form are stored as a JSON object that can be converted in a text file with an user-defined

[3]http://jeremydorn.com/json-editor
[4]form.io
[5]www.alpacajs.org
[6]www.jotform.com

format. Based on agile technologies and mockups [7], the development phase converges in few iterations of elicitation of domain specific knowledge and integration in user interfaces, i.e. the Web form GUI built through Json-GUI. The logical phases of this process are schematised in Figure 1. Starting from

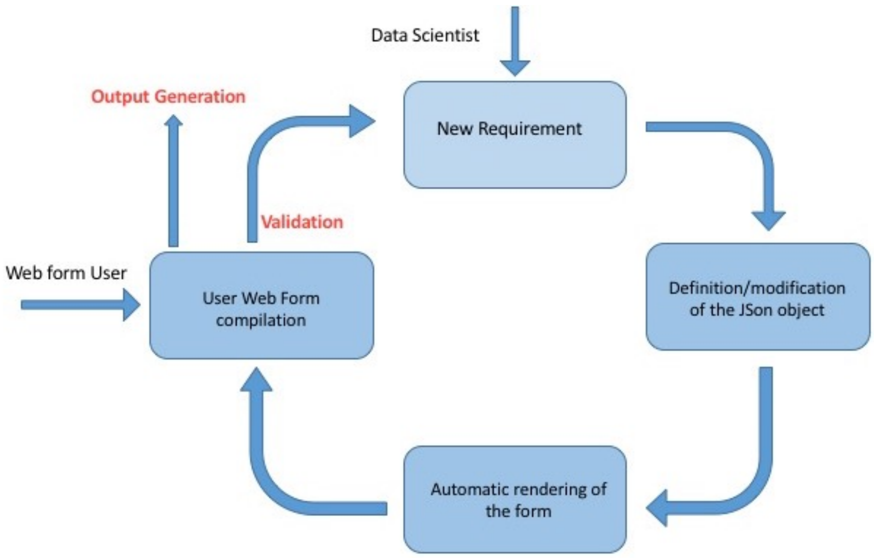


Figure 1: A logical schema of the Json-GUI usage.

the interaction with scientists, a first round of requirements are elicited and the definition of the JSON object is derived. In this phase, the main actors involved are data scientists and Web form users. At this point, Json-GUI automatically generates the Web form corresponding to the JSON object, and users/scientists can fill in the values corresponding to the defined fields. Once the Web form is compiled, Json-GUI generates the output: a JSON object that can be possibly customized and used for the final aim. The generated output is a generic Json object, and, thus, it is ready to be processed by any middleware, workflow manager or local scheduler. If the form is in a validation phase, the interaction among scientists and the Json-GUI user can continue to elicit more requirements, modify the JSON object and lead to the correct Web form. Json-GUI speeds up this phase enabling a run-time visualization of the Web form, reducing the duration of iterations for the elicitation/integration of derived information.

Since the definition of the input for the generation of the web form could become a bit challenging, we provide a tool to build the corresponding JSON object, called Json-GUI-Builder. Through a simple interface, the Builder completely supports developers, i.e. Json-GUI users, in the definition of parameters and related validation, constraint, condition rules. In Section 3, two examples of Json-GUI-Builder graphical user interface are reported.

Form Fields

The core of the input object consists in an array of *parameters*, where each element defines (and renders) a single input field of the form. The possible input forms are: **integer** and **float** respectively generating a field for the specification of an integer and a float number; **datetime** generating fields for the specification of a date, including hours and minutes; **select** generating a combo box to select a value among the available ones; **text** generating a plain text input field; **domains**, generating a geographical map where rectangular domains and single markers can be drawn; **fileupload** defining an input box to upload one or more files.

Json-GUI offers high level features to enrich the form interface by defining:

- **Validation checks** - each parameter type has internal format validation, e.g. float and integer types have a built-in number format verification. Moreover, it is possible to add a custom validation for the specification of a behavior: e.g. a user may define a datetime input valid if it predates a specific date - the 1st January 1970.

- **Constraint rules** - since parameter values may mutually influence their behavior, constraints among different inputs can be implemented: if a time range has to be fixed, it is possible to set the "Start date" parameter value valid only if predates the "End Date" parameter value. This gives Json-GUI the potential to specify all standard constraints of a classic HTML5 form based interface.

- **Conditions** - Json-GUI offers the possibility to specify a condition (constant or depending on the value itself) to activate/deactivate parameters in the input form. This permits enrich the form interface with a dynamic behavior when managing, for example, `Select` and `Domain` parameters. A common example for `Select` parameter can be a form for online payment, Json-GUI allows to present different form fields depending on the value of a *payment method* field: if the selected value is Paypal, the GUI presents fields for Paypal login, with a Credit Card value, the GUI presents fields for credit card configuration (e.g. the credit card number, CVV, name and surname of the owner), and so on. The same level of dynamism is ensured when considering the `Domain` parameter, since the number of geographical domains relies on the user interaction and is unknown a-priori: depending on the number of domains that a user draws, the GUI can display different form fields and information. For example, in Figure 2 three geographical domains are considered, and the related geographical coordinates are displayed for each domain.

As standard behavior implemented by Json-GUI, if one of the rules/checks described above is violated, it will be not possible to submit the form and the output will not be generated. A custom message can be displayed if specified during the definition of the related parameter. Example are reported in the remaining of the Section and in Figure 6.

Software Architecture

Json-GUI presents a two-level software architecture. The higher level is represented by the Web form GUI automatically rendered from the JSON object, equipped with its overall logic and behavior. This includes the validation checks among parameters and the collection of each value to build the overall output, i.e. couples of parameters and corresponding values possibly stored in a text file following an user-defined format. The lower level is represented by the AngularJS directives defining each parameter type. This level defines the individual behavior of the form fields, including the internal validation. Each validation rule can be general-purpose or specific.

Software Functionalities

The basic element of the JSON object, input of Json-GUI, is a `parameter` that contains the value and all conditions that apply on it. Each parameter of the Json-GUI object has the following structure:

```
parameter: {
  value: {type: "parameterType"},
  displayName: {type: "string"},
  dbName: {type: "string"},
  isValid: {type: "string"},
  parameterType: {type: "enum('float', ..., 'fileupload')"},
  parameterCategory: {type: "integer"},
  computedResult: {type: "string"},
  dependencies: [{type: "string"}, ...],
  required: {type: "boolean"},
  editable: {type: "boolean"},
  description: {type: "string"}  }
```

The *displayName* property defines the name of the parameter to be displayed in the interface, while the *dbName* is a unique identifier used internally. The *parameterType* defines the type to be specified among the ones supported. The *parameterCategory* property allows to logically group parameters in the form, e.g. by appearing in the same tab. The parameter can also be marked as *required*, it is possible to specify if the default *value* can be *editable* or not. The *description* property contains a text shown in an info box, and can be used as hint to the user. The *dependency* property is an array containing the references to the parameters on which the current parameter depends. These are the parameters that shall be used within the *isValid* property. This property is a string containing a Javascript function body to possibly define custom validations. The following is an example, where also a custom message is set for invalid condition:

```
isValid : "if(parameter.value < dependencies['dep_1'].value) {
  isValid.valid= false; isValid.message='custom error message';}"
```

The *computedResult* property defines a Javascript function meant to perform a final computation in order to (possibly) refine the value before the form submission. An example is the following:

```
computedResult: "return parameter.value/1000;"
```

Please note that the *computedResult* property allows a further customization for the value of the single fields; this is extremely useful when a specific format is required, e.g. datetime parameters formatted in a different standard or a specific projection for a geographical domain parameter.

3. Illustrative Examples

A valuable example is presented by the form field `Domain` defined to support the hydro-meteorological community in the configuration of the Weather

Research and Forecasting, WRF[7] Model. The possibility to draw a geographical domain by using a graphical map has been actually acknowledged by scientific community; for this reason, the `Domain` input type has been implemented with the integration of the Google Map JavaScript library. Furthermore, meteorological models usually enable the definition of more than one domains, that can be nested or not: nest is a finer-resolution model run, that can be embedded simultaneously within a coarser-resolution (parent) model run, or run independently as a separate model forecast. The first case, depicted in Figure 2, represents nested domains. For this reason, the `Domain` is enhanced with the possibility (for the user) to draw up to three rectangles, each one representing a geographical domain, and a constraint to permit the drawing of nested domains has been defined.

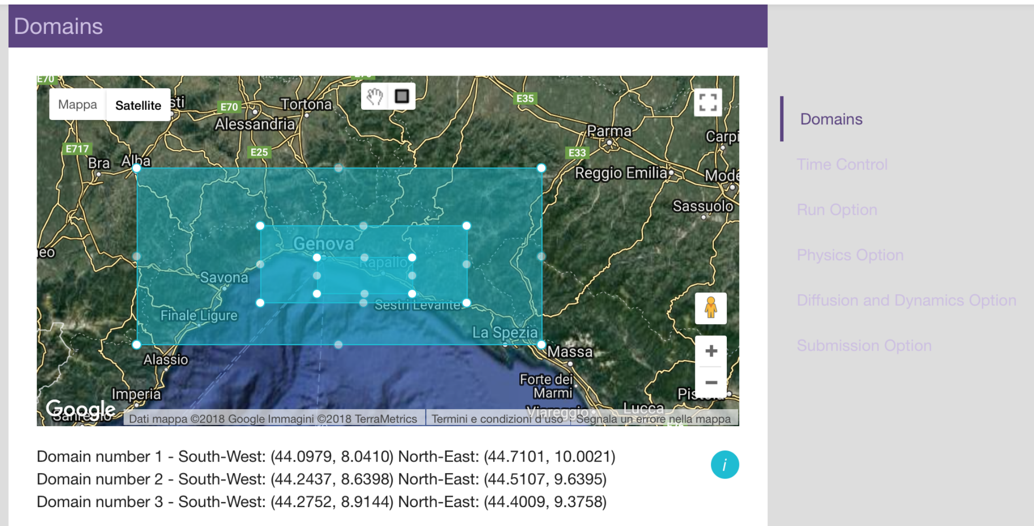


Figure 2: An example of GUI to draw up to three domains exploiting a Google map.

Figure 3 presents a sample code related to the hydro-meteorological science gateway [6] and leading to the configuration depicted in Figure 2: the

[7] http://www.wrf-model.org

parameter `maxDomains` limits to three the maximum number of drawable domains and the parameter `onlyNested` permits to draw domains only inside a single parent domain. In Figure 4, an example of Json-GUI Builder interface corresponding to the `Domain` parameter is shown.


```
▼ Object
  type: "meteo"
  ▶ parametersCategories: Array [2]
  ▼ parameters: Array [3]
    ▼ 0: Object
      description: "Select up to three domains on which run the simulation"
      editable: true
      namelistName: "domains"
      allowMarkersOutDomains: true
      ▶ required: Object
      drawDomains: true
      drawMarkers: false
      maxMarkers: 0
      maxDomains: 3
      onlyNested: true
      mapZoom: 8
      ▶ center: Object
      parameterType: "domains"
      ▶ value: Object
      isValid: ""
      computedResult: "(function(){return true;}())"
      unremovable: true
      ▶ dependencies: Array [0]
      parameterCategory: 0
      dbName: "domain1466505616682"
      displayName: "Domains"
    ▶ 1: Object
    ▶ 2: Object
```


Figure 3: The Json-GUI parameters for automatic building of the Domains form field.

Another significant example is provided in Figure 5, that shows sample code related to the form field `Time_Interval_Selection`, representing one of the input of the transient analysis tool provided by the EXTraS science gateway. A wide diversity of astrophysical phenomena - from stars to supermassive black holes - are characterized by flux and spectral changes on time scales, ranging from a fraction of a second to several years. Current observing facilities subdivide an observation in a set of images, with a time resolution of the order of 1 sec. or shorter. In particular the transient analysis is based

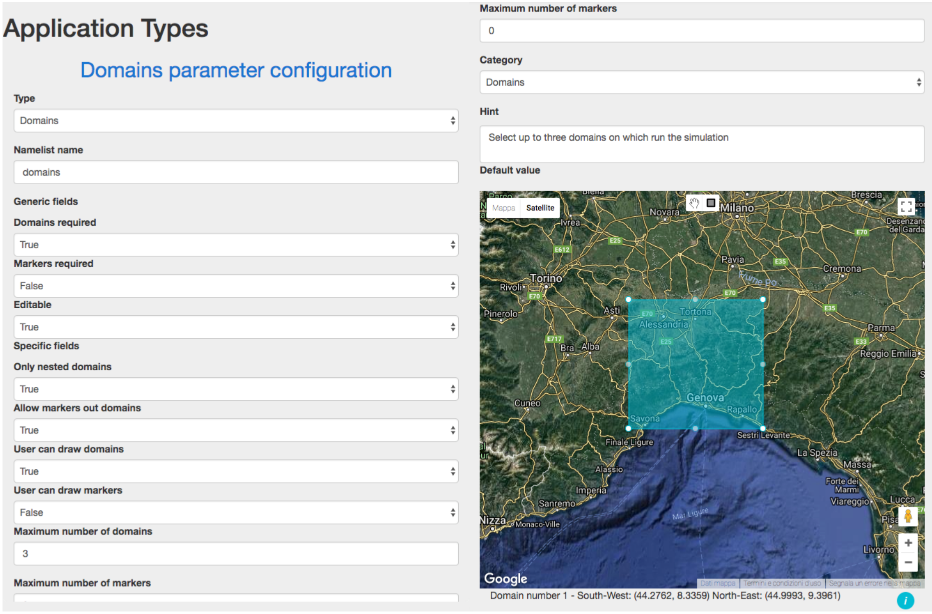


Figure 4: An example of parameter definition with Json-GUI-Builder.

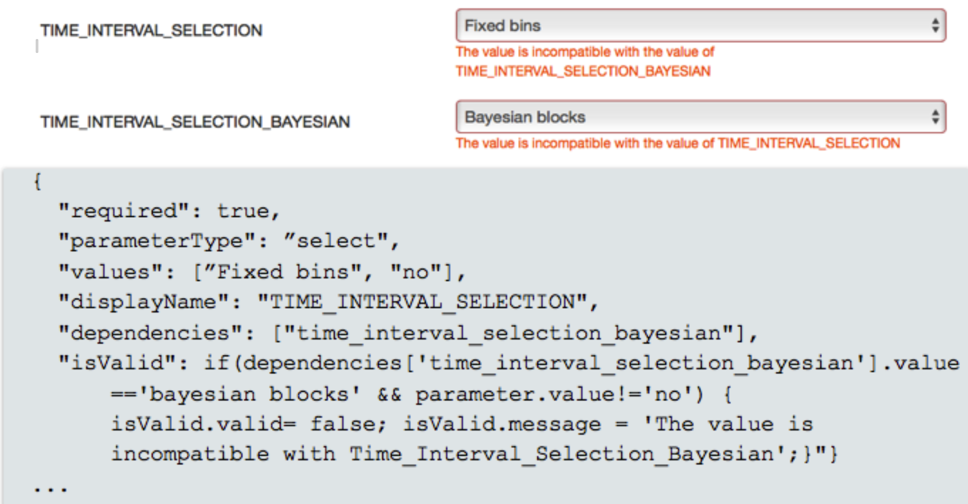


Figure 5: An example of parameter and consistency check definition with Json-GUI.

on the use of two alternative subdivision strategies, i.e. the use of fixed time intervals or variable intervals based on the Bayesian blocks algorithm. Therefore the user can select only one method and, consequently, the form field depends on the parameter named `Time_Interval_Selection_Bayesian`, because one and only one of them must have the "no" value. This is specified with the *dependencies* and *isValid* properties. Figure 5 shows also the error message raised in the GUI when the condition related to the parameters are not verified. In Figure 6, an example of Json-GUI Builder interface corresponding to the `Time_Interval_Selection` parameter is shown.

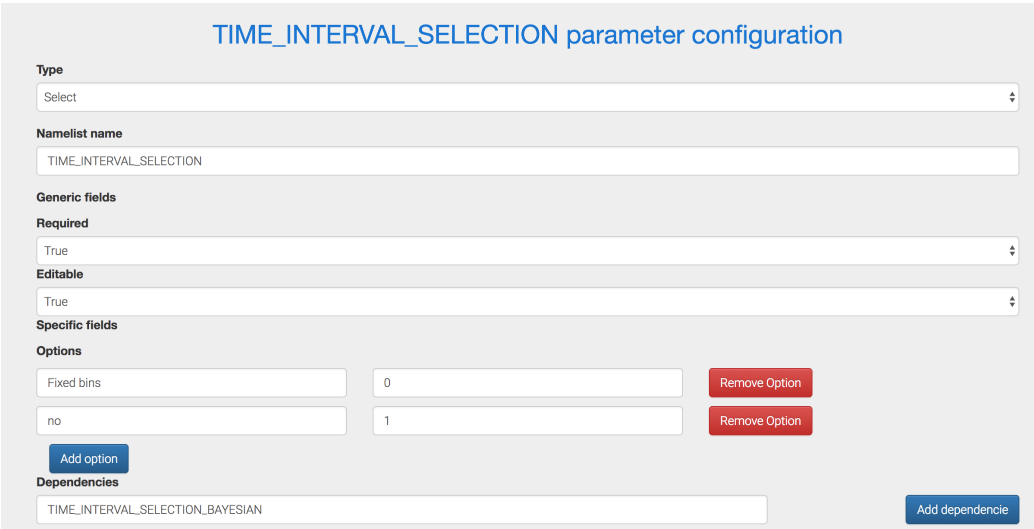


Figure 6: An example of parameter definition with Json-GUI-Builder.

4. Impact

Json-GUI represents a step towards closing the gap between the high level and low level layers of a science gateway, represented respectively by the community-specific GUI and the general-purpose middleware plus the computational infrastructure. Most of the available framework to develop science gateways do not provide a suitable support for the definition of customized GUI [6]. This may be challenging for non-IT communities, and a wrong selection of the front-end technology, combined with frequent developer turnover, can represent a major issue for the gateway sustainability [3]. Json-GUI definitely accomplishes this task while adding valuable features.

Actually, Json-GUI allows the dynamic generation of web forms without the need to write any line of code. However this does not limit its expressiveness. The possibility to define customized rules on/among parameters in facts gives Json-GUI the potential to specify all standard constraints of HTML5 forms. The possible complexity in the definition of parameters rules are delegated to the Json-GUI-Builder, therefore again, this task does not suppose specific programming expertise.

Furthermore, since user interfaces are dynamically generated starting from a JSON Object, it is possible to modify a web form interface on the fly by simply modifying the object without the need to re-deploy or restart any service. The resulting faster development cycle is very relevant in research fields relying on software tools developed (and frequently updated) by the community. Of course, such reduction has an impact also in terms of costs, thus becoming appealing in a general-purpose context.

Focusing on the added value features, the most valuable are constraints and conditions. The consistency check among parameters supports the proper configuration of experiments and, performed before the actual execution of the models, avoids the waste of CPU time due to execution of a misconfigured experiment. Also the possibility to draw geographical domains has been actually appreciated in the scientific community, and a great effort as been dedicated to this point, as outlined in Section 3.

And last but not least, Json-GUI effectively supports the creation of configuration files that can be directly ingested by target applications. Validated data collected through the generated form interfaces in fact can be stored as Json object or text file, e.g. as classical key-value format, but it is possible to define further customizations to match the expectations of the models/applications. A user can develop and override any standard behavior of the generation phase: a transformation function can be defined for each field as well as for the final configuration file. This file can be used by the specific tools in charge for application execution; the actual submission can then be performed by the science gateway services, as described in [5].

5. Conclusions

We presented Json-GUI, an AngularJS front-end module which allows to quickly create form-based web interfaces. The module supports the export of the parameters in structured data files, which are often used for configuring complex experiments. The tool demonstrated its effectiveness a) in supporting users for the configuration of scientific experiments, where it is important to keep consistency among the inserted values, and b) in supporting non-IT experts for the design of such complex interfaces.

References

[1] G. Andronico, V. Ardizzone, R. Barbera, B. Becker, R. Bruno, A. Calanducci, D. Carvalho, L. Ciuffo, M. Fargetta, E. Giorgio, et al., E-infrastructures for e-science: a global view, Journal of Grid Computing 9 (2) (2011) 155–184.

[2] P. Kacsuk, Science gateways for distributed computing infrastructures, Springer International Publishing. doi 10 (2014) 978–3.

[3] K. A. Lawrence, M. Zentner, N. Wilkins-Diehr, J. A. Wernert, M. Pierce, S. Marru, S. Michael, Science gateways today and tomorrow: positive perspectives of nearly 5000 members of the research community, Concurrency and Computation: Practice and Experience 27 (16) (2015) 4252–4268.

[4] D. D'Agostino, L. Roverelli, G. Zereik, G. La Rocca, A. De Luca, R. Salvaterra, A. Belfiore, G. Lisini, G. Novara, A. Tiengo, A science gateway for exploring the x-ray transient and variable sky using egi federated cloud, Future Generation Computer Systems.

[5] A. Galizia, L. Roverelli, G. Zereik, E. Danovaro, A. Clematis, D. D'Agostino, Using apache airavata and easygateway for the creation of complex science gateway front-end, Future Generation Computer Systems.

[6] D. D'Agostino, E. Danovaro, A. Clematis, L. Roverelli, G. Zereik, A. Galizia, From lesson learned to the refactoring of the drihm science gateway for hydro-meteorological research, Journal of Grid Computing 14 (4) (2016) 575–588.

[7] C. D'Souza, V. Deufemia, A. Ginige, G. Polese, Enabling the generation of web applications from mockups, Software: Practice and Experience.

Required Metadata

Current code version

Nr.	Code metadata description	Please fill in this column
C1	Current code version	1.1.3
C2	Permanent link to code/repository used for this code version	https://github.com/portalTS/json-gui/releases/tag/1.1.3
C3	Legal Code License	Apache License 2.0
C4	Code versioning system used	git
C5	Software code languages, tools, and services used	Javascript, HTML, CSS, AngularJS, Bootstrap
C6	Compilation requirements, operating environments & dependencies	AngularJS, Bootstrap, JQuery
C7	If available Link to developer documentation/manual	https://github.com/portalTS/json-gui/wiki
C8	Support email for questions	gabrielezereik@gmail.com

Table .1: Code metadata (mandatory)