Domain-oriented specialization tools for knowledge-based systems development platform

A.I. Pavlov [0000-0002-7753-7514] and A.B. Stolbov [0000-0001-6513-7030]

Matrosov Institute for System Dynamics and Control Theory of Siberian Branch of the Russian Academy of Sciences, 134 Lermotov st., Irkutsk, Russia
{asd,stolboff}@icc.ru

Abstract. New details of the implementation of the data representation and editing component of the knowledge-based systems development platform is proposed in the paper. The two modes of considered component is described. The typical component structure along with the core to client component parts communication issues are presented. The main UI control elements are listed and grouped according to the criteria of the number of efforts that applied developers spend in the course of making UI. On the top of controls from the first group considered component provides user interface for data control component and implements unified user interface for any valid component of the platform. The second group is responsible for user interface of the platform as a whole and also for problem-oriented functions of the core components (for example, rule or workflow editors). The third group relies on a predefined set of parameterized typical database manipulations and supports the applied developer in the process of UI customization. The implementation of two techniques is considered on the case of the conceptual model design component extension.

Keywords: Knowledge-Based Systems, User Interface, Component-Based Development.

1 Introduction

Knowledge-based systems (KBSs) are specialized computer-based tools for storing, processing, and visualization of the data and knowledge in order to solve a wide range of problems in the decision support and information management. As stated in [1], KBSs "can aid human cognitive deficiencies by integrating various sources of information, providing intelligent access to relevant knowledge, and aiding the process of structuring decisions". KBSs have been successfully applied in many different domains but in the near future, one can expect significant growth of its applications due to the successful distribution of IT systems in so many areas. So modern KBSs should be consistent with contemporary IT level and continuous work for new KBSs developing and refinement of existing ones is still an actual problem.

In a series of publications [2-5] the authors make a contribution to the discussed problem by proposing the knowledge-based systems development platform (KBSDP).

Previously, the description idea was primarily to show platform usability as a universal tool for making typical products: a set of system, core, auxiliary, and applied components was considered with different levels of detail and attention. Meanwhile, the platform can be discussed from another viewpoint: how applied KBS developers can set up the typical results produced by the platform for some domain-specific needs. The current paper partially covers the mentioned issue focusing on end-user interaction aspects for which domain friendly customization is an important factor.

The data representation and editing component was briefly presented among others in [1, 5] to help with the description of KBSDP architecture elements. The main concern was devoted to the relation of it with the data control component and to the supported at that time set of user interface (UI) control elements. But in the current paper UI related problems are discussed in a broader sense.

In the next sections of the paper the overview of KBSDP functionality related to UI design support and client-server communication features in the context of UI are described. An illustrative example of the conceptual model design component extension is considered. Some implementation issues are also discussed.

2 User interface design and customization features of knowledge-based systems development platform

2.1 Available functionality overview

The design and implementation process of the user interface part of the software systems is a difficult and time-consuming task. Moreover, this process requires continuous modifications and refinements regarding changes as in the problem-solving part of the system and in utilized tools or current fashion of UI design. As confirmed in [6], generally the user-interface development process takes up to 50% of development and 40% maintenance time as well as 50% of the code of the whole software. Up to date a lot of efforts has been devoted to UI methods, libraries, frameworks for different platforms and their versions. At present, the main user interface development activity occurs in web space and a lot of new tools and methodologies are emerging. In this situation, a program from which it is expected to be maintained in the long term has to be developed with innate flexibility in its architecture.

To address this problem KBSDP internally contains the possibility of expansion on different levels: on program interface level [2] to support system and problem-oriented component interoperability and on user interface level to provide domain-oriented customization. Let's express this idea in a more detailed and structured manner. As stated previously [5] typical component S^{typ} has the following structure:

$$S^{typ}=<\{B^{UI}\}, B^{Core}, [\{B^{Ext}\}] >,$$

where B^{UI} – client part with the user interface, B^{Core} – available to the user server part with component's functionality, B^{Ext} – optional different extensions of server part with limited access, which are mainly used for integrating existing software systems, such as the Drools or the Madkit engines. In this paper we continue to consider typi-

cal component architecture in the context of component interaction with end-user and applied KBS developers.

From the data flows viewpoint communication between B^{Core} and B^{UI} parts can be done in the following ways:

$$C=<C^{Data} \mid C^{DB} \mid C^{DS}>,$$

where C^{Data} is based mainly on the functionality of data control component (S^{data}) [2]; C^{DB} relies on a predefined set of parameterized typical database manipulations; C^{DS} is the most general case (DS means any Data Source) where expandable set of domain-oriented or problem-oriented operation identified by a unique name, implemented by the applied developers directly by themselves and processed on a server by any technique (PHP script, executing a program, redirecting requests to remote resources, etc.).

The C^{Data} type of communication formalizes the data exchange process for a standard set of operations at the level of database entities (tables, persistent classes): retrieving, inputting, modifying, and deleting information for one or more elements of the selected entity (table rows, objects). In this case, entities are identified by their name in the database, entity properties are accessed by their internal name (standard prefix + ordinal number). The conditions for selecting elements for retrieving or data modification operations are defined using *IConditionsOfSearching* program interface, described in [2].

The C^{DS} provides a process of data exchange in cases of the execution of the operations, due to their domain or problem specificity, have an extremely low potential for reuse and are characterized only by a name and a set of parameters.

The S^{UI} component is designed to automate the implementation of B^{UI} blocks of KBSDP components and contains a set of controls of various levels of purpose and complexity, ranging from a simple dialog box with an input field to a table that provides rich information representation with the ability to sort, search, and edit. There are two main modes of using S^{UI}: **library mode**, where the developer creates a user interface by combining calls of controls from S^{UI} with their own code; **constructor mode**, when the developer creates a description of the user interface in a special editor without writing a code.

The current implementation of S^{UI} is done in Javascript using functionality of following free libraries: jQuery, jQueryUI, jQueryGrid, jsPlumb to basic set of controls for representation data in tabular format, performing data editing operation in unified way, organizing user interface for accessing to functions of components in style of oriented graph, etc. The control elements from the S^{UI} component can be divided in 3 types:

$$S^{UI}=< S^{UI\text{-}Std}, S^{UI\text{-}Platform}, S^{UI\text{-}Lib}>,$$

where the $S^{UI\text{-}Platform}$ block contains controls intended for implementing the user interface at the level of KBSDP and its components, for example, it is an GUI for accessing platform functions using a visual model of the oriented graph or the GUI of the rule editor, etc. Each of these controls depends on a specific data model, so to use

them, the application developer needs either to have access to the server part of the corresponding component, or to implement a similar model together with its processing tools. Because the controls are more specialized, to use them it is only needed to specify the container on the html page because the server interaction parameters are determined by the set of named operations from the C^{DS} defined in the control.

In the next section, $S^{UI\text{-}Std}$ and $S^{UI\text{-}Lib}$ are considered in more detail.

2.2 The user interface implementation issues

The $S^{UI\text{-}Std}$ block provides a user interface for accessing the functionality of data control component (S^{Data}). The main controls of this block are: table to browse the contents of a selected database, table to browse the information of a selected database entity, the customizable form for edit data of object, control for browse structural model of database or its selected part like schema. The main feature of these controls is their ability to be configured automatically, when the applied developer only needs to specify the name of this entity and the HTML element where it should be placed. During configuration, the control independently accesses the S^{Data} and, using the entity name, gets its description in the required format, which is used to form the appropriate user interface. For example, to browse the information of selected database entity the control generates related number of columns and configures forms for editing and searching information. For this control, S^{Data} converts the database entity metadata from an internal view in the form of the *IDescriptionOfDatabaseObject* interface [2] to the jQueryGrid library table format. Thus, by explicitly metadata usage, $S^{UI\text{-}Std}$ provides for the S^{Data} component both a unified view of the user interface and detailed consideration of the database entity structure simultaneously. At the same time, the $S^{UI\text{-}Std}$ controls have the highest degree of versatility and reuse.

An important function of $S^{UI\text{-}Std}$ is to provide the KBSDP component with a standard user interface. In this case, it is assumed that the control for viewing information in the form of a table will be used to view and edit the properties of the state and behavior of the component. In this case, the source data for generating the parameters of the jQueryGrid table is information available through the *IComponent* interface [2]. The only difference in this case is additional processing of behavior description elements, in which tables formed based on information from the *IMethodDescription* interface [2] are supplemented with a special button that allows launching this method. As a result of this function execution, the developer is presented with a dialog box with a table, where the first row is responsible for the state of the component, and the rest for the behavior. A nested table is associated with the component state row, where each column is responsible for a property, and each row that is responsible for a component method opens a nested table whose columns are method parameters. Using the proposed component prototyping scheme allows one to reduce the overall time spent on component development by eliminating the user interface creation stage, which is especially important when the component is intended for various types of calculations.

The $S^{UI\text{-}Lib}$ block contains all control elements adopted to make the B^{UI} part of the component. In these sense $S^{UI\text{-}Std}$ and $S^{UI\text{-}Platform}$ are constructed based on $S^{UI\text{-}Lib}$ by the

platform developers. The applied developers can also use $S^{UI\text{-}Lib}$ block control elements but in general they can spend a lot of efforts. To facilitate the $S^{UI\text{-}Lib}$ block usage some controls have versions ($S^{UI\text{-}Lib\text{-}DB}$) with the build-in support for C^{DB} communication type. In this case, it is possible to reduce time and efforts spent on B^{UI} development process as most of the server-client and event handling routines are preconfigured.

For example, the subclasses of *EposTableView* provide the functionality for processing data in tables depends on the specified type of relation database: own data in tables (without foreign keyed data), recursion in one table (to represent, for example, concept hierarchy) one-to- many, many-to-many relationship. to describe desired table elements for display and edit. The *EposTableView* class is parameterized with an instance of *EposRecordInDBDataSource*.

The *EposRecordInDBDataSource* structure describes desired table elements for display and edit. The developer has to specify a table name and a list of columns in the form of *ColumnDataInDB* structure, that includes the name of column, title for displaying, and a specification of GUI control – an instance of *EposUIControl* subclasses, with parameters (for example, true/false flag for "is key, "string" for "data type", "true" for "can edit", "id" properties).

To provide data manipulations, currently, $S^{UI\text{-}Lib\text{-}DB}$ supports the following subclasses of abstract class *EposUIControl*:

- *EposUITextControl* – a KBSDP wrapper for input html control;
- *EposUINumbertControl* – a KBSDP wrapper for input html control with integer or real number validation;
- *EposUIBigTextControl* – a KBSDP wrapper for textarea html control;
- *EposUIEnumControl* – a KBSDP wrapper for input checkbox html control.

These set is intended to support literal datatypes (numeric, text, Boolean), but theoretically there is no restriction on the GUI element to be warped by $S^{UI\text{-}lib}$ or $S^{UI\text{-}Lib\text{-}DB}$.

At last the *EposObjectView* class provides functionality for combining different types of $S^{UI\text{-}Lib\text{-}DB}$ views that from a developer viewpoint must be visually represented for the user at the same layout (for example, information about some object type). The *EposObjectView* is parameterized with the name of html container, a title, and URL for server script and a list of particular views inherited from the base class *EposTableView*.

3 An illustrative example

3.1 Description and motivation of example

As an illustrative example for the ways how applied developers can design and customize components in KBSDP, the extension of one of the core components is considered. Namely, S^{CM} – the conceptual model design component [2] is being refined. The original version of S^{CM} has the following functionality: creating a hierarchy of concepts, defining attributes and assigning them to concepts, establishing relations,

making instances. The basic S^{CM} data model *CM* [2] represents the following information:

CM=<Concept, Attribute, Relation, Concept Instance>,

<Concept>=<Name> <Description> {<Attribute>},

<Attribute>=<Name> <Attribute type> <Default value>,

<Attribute type>= Literal | Concept,

<Relation>=<Concept> <Name> <Relation type><Concept>,

<Relation type>=Literal.

Here, *Concept*, *Attribute*, *Relation*, *Literal* and other related terms can be referred to the definitions commonly known in the computer science and knowledge engineering domains.

The B^{UI} part of S^{CM} is implemented by the platform developers and currently has a rich GUI [2], including a semantic network view to represent, for example, *Relation*, hierarchical table view for a list of concepts, or accordion view for visual layout of actions and properties. All these control elements are configured and combined together by a lot of code and so in previously defined terms it corresponds to $S^{UI\text{-}Platform}$ block of S^{UI} component.

Nowadays, when the problem complex ontology development is considered it is urgent to acquire as much information on the design stage as possible. The sources of information, diversity in definitions and synonyms, creators profiling and other useful data must be identified and formally described. These problems are currently in focus of computer science and knowledge engineering domains [7, 8].

So, the mentioned basic CM model currently is not enough for complex ontology or conceptual model design and a new information must be added for processing in S^{CM} component. Based on our experience in conceptual modeling we apply the *Annotation* data model shown on figure 1.

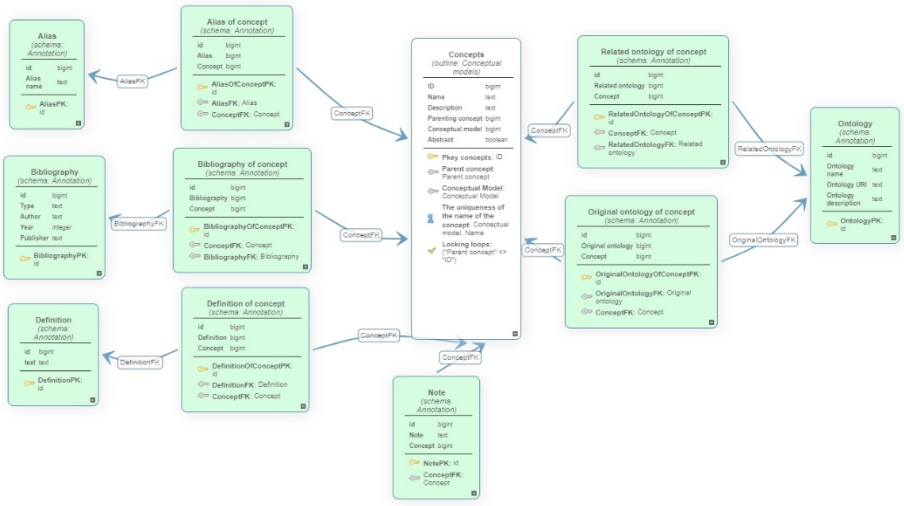

Fig. 1. Annotation data model

Further, let's consider the case when, for some reason, the platform developer does not intend (in the short term) to add new functionality about annotating the conceptual model. Meanwhile, for applied developers the platform provides two possible ways to facilitate the process of design and customization of new S^{CM-A} component. These ways are considered in the next section.

3.2 Designing and customizing component for annotating the conceptual model

The S^{CM-A} was tested for designing two conceptual models each of each is the result of integration and adaptation of a number of ontolgies. One conceptual model is Conceptual Model of Infrastructure Logistics (CMIL) [9] based on [10] and other 5 ontolgies. Another conceptual model is now under development and is based on [11].

The figures 2 and 3 illustrate the result of making GUI for annotating CMIL with the help of different S^{UI} parts. On the figure 2 the S^{UI-Std} related interface is presented. It was generated automatically based on description of entities defined by unified interface implemented by the S^{CM-A} component.

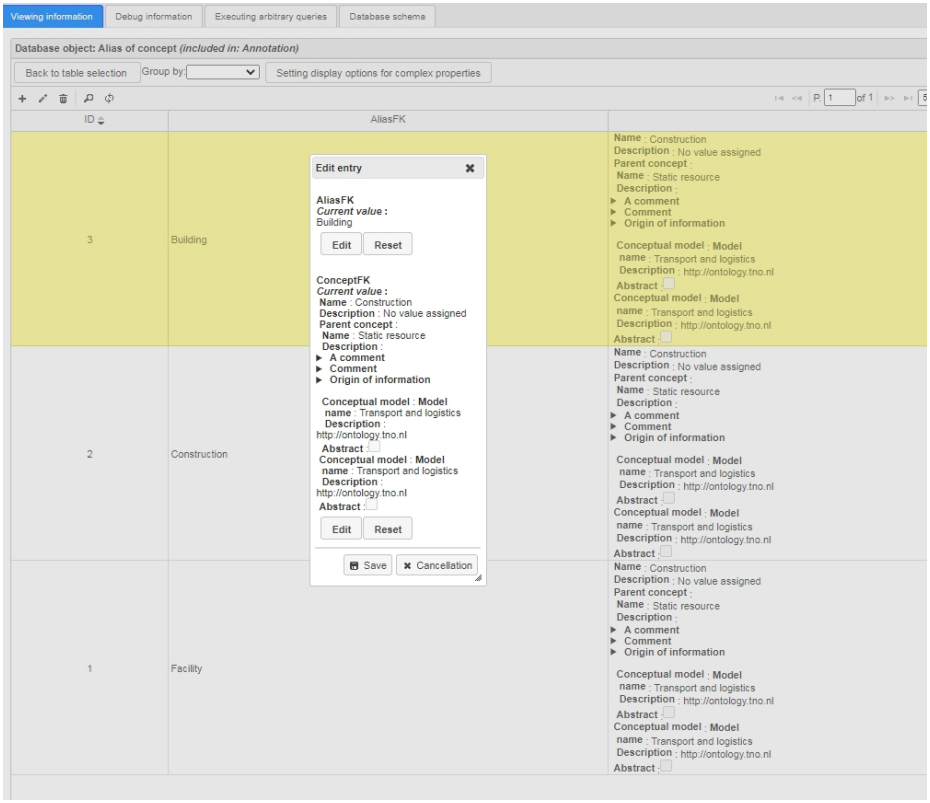

Fig. 2. $S^{UI\text{-}Std}$ based user interface of $S^{CM\text{-}A}$ component

On the figure 3, B^{UI} part of $S^{CM\text{-}A}$ component is made with the help of $S^{UI\text{-}Lib\text{-}DB}$.

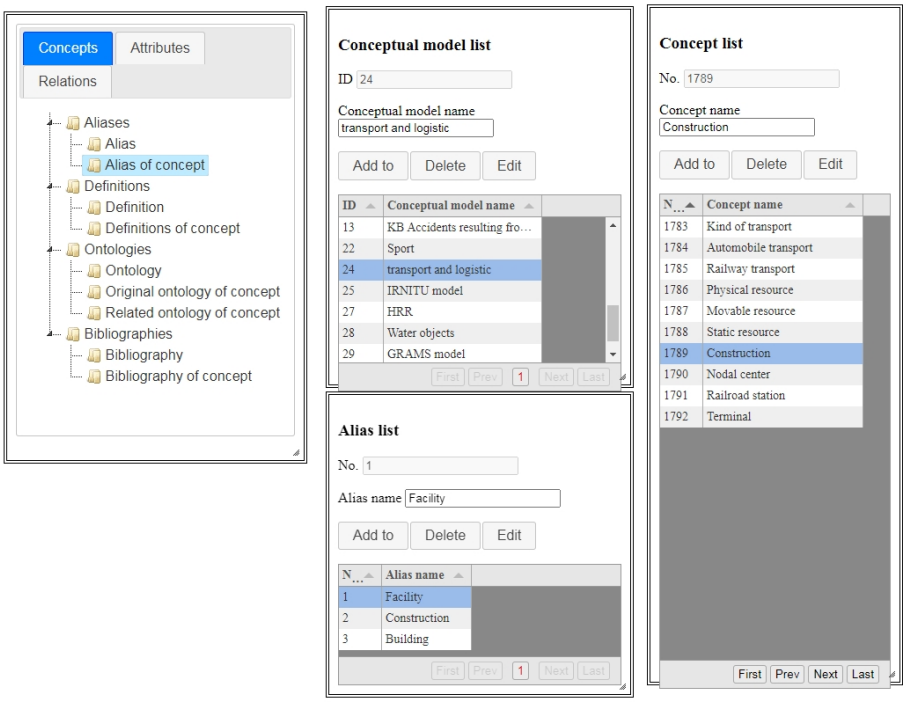

Fig. 3. $S^{UI\text{-}Lib\text{-}DB}$ based user interface of $S^{CM\text{-}A}$ component

To build this interface an applied developer in the code of web page for each annotation element has to create:

- an instance of *EposDataSource* (particularly its subclass – *EposRecordInDBDataSource*) to describe desired table elements for display and edit;
- a view container (instance of *EposObjectView*);
- a set of required view types (subclasses of EposTableView).

Then he adds the main control element based on the tree GUI element from $S^{UI\text{-}Lib}$. And for each node of the tree he defines association with a certain view (*EposObjectView*). All other stuff, including client-server communication, appropriate data updating when elements changed or selected is done automatically based on $S^{UI\text{-}Lib\text{-}DB}$ functionality and C^{DB} communication facilities.

The former variant of obtained interfaces supports few opportunities for customization but also requires few efforts from the applied developer. Meanwhile in the last variant applied developer can choose what he wants to show to the end-user and combine selection and editing dialogs in arbitrary number with flexible layout.

4 Conclusions

The paper deals with the knowledge-based systems development platform in the aspect of user interface design and customization. The available options for building user interface are considered. First option is a standard and unified user interface for virtually any valid knowledge-based systems development platform component. It is best suited when fast prototyping of component is a major factor. This way is also may be recommended for the applied developers as it requires fewer efforts compare with other techniques. The main disadvantage is the lack of customization due to universality. On the opposite side from the effort and customization viewpoint is development of the special user interface solutions that express problem-oriented or domain-oriented specificity of the considered task on the top of the user interface library of the platform. In this case, the applied developers have a high customization possibility because they manually coding the desired user interface from the set of platform control elements. And in the middle of these two user interface design techniques there is another one based on templating the server-client and event handling routines of the platform. As an illustrative example the problem of the conceptual model design component extension is considered. The implementation of the first and the third techniques is described.

5 Acknowledgments

The research was supported by the Program of the Fundamental Research of the Siberian Branch of the Russian Academy of Sciences, project no. IV.38.1.2 (reg. no. AAAA-A17-117032210079-1), project no. IV.38.1.3 (reg. no. AAAA-A17-117032210077-7). Results are achieved using the Centre of collective usage «Integrated information network of Irkutsk scientific educational complex».

References

1. Alor-Hernndez, G., Valencia-Garca, R. (eds.): Current Trends on Knowledge-Based Systems. Intelligent Systems Reference Library series. Springer, Cham Switzerland (2017).
2. Nikolaychuk O.A., Pavlov A.I., Stolbov A.B.: The software platform architecture for the component-oriented development of knowledge-based systems. In: Proceedings of the 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pp. 1234–1239. IEEE (2018).
3. Pavlov, A.I., Stolbov, A.B., Dorofeev, A.S.: The workflow component of the knowledge-based systems development platform. In: Bychkov, I.V., Karastoanov D. (eds.) Proceedings for 2nd Scientific-practical Workshop Information Technologies: Algorithms, Models, Systems (ITAMS), CEUR Workshop Proceedings, vol. 2463, pp. 47–58. CEUR-WS.org, Irkutsk Russia (2019).
4. Pavlov, A.I., Stolbov, A.B.: The Application of the Knowledge-Based Systems Development Platform for Creating Scenario Analysis Support Tools. Advances in Intelligent Systems Research, 169, 43–48 (2019). https://doi.org/10.2991/itids-19.2019.36

5. Pavlov, A.I., Stolbov, A.B., Dorofeev, A.S.: The architecture of the software tool for the agent-based simulation models specification development. In: Bychkov, I.V., Karastoanov D. (eds.) Proceedings for 2nd Scientific-practical Workshop Information Technologies: Algorithms, Models, Systems (ITAMS), CEUR Workshop Proceedings, vol. 2463, pp. 112–121. CEUR-WS.org, Irkutsk Russia (2019).
6. Meixner, G., Paterno, F., Vanderdonckt, J.: Past, Present, and Future of Model-Based User Interface Development. i-com 10, 2–11 (2011). 10.1524/icom.2011.0026.
7. Arakaki, F.A., Vesu, A., Rachel, C., Amorim da Costa, S., Placida, L.V.: Dublin Core: State of Art (1995 to 2015). Informacao & Sociedade-estudos 28(2): 7–20 (2018). http://hdl.handle.net/11449/166289
8. Kotis, K., Vouros, G., Spiliotopoulos, D.: Ontology engineering methodologies for the evolution of living and reused ontologies: status, trends, findings and recommendations. The Knowledge Engineering Review 35, 1–34 (2020). 10.1017/S0269888920000065.
9. Lempert, A.A., Stolbov, A.B.: An approach to knowledge bases development for support of complex research in infrastructure logistics. Information and Mathematical Technologies in Science and Management 3(11), 45–54 (2018). 10.25729/2413-0133-2018-3-05.
10. Anand, N., Yang, M., van Duin, J.H.R., Tavasszy, L.: GenCLOn: An ontology for city logistics. Expert Systems with Applications 39, 11944–11960 (2012).
11. Elag M., Goodall, J.L.: An ontology for component-based models of water resource systems. Water Resour. Res. 49, 5077–5091 (2013).