A Tool for Modeling JsonLogic based Business Process Rules

Katira Soleymanzadeh
Hermes Iletisim, Pasaport
Izmir, Turkey
katira.soleymanzadeh@hermesiletisim.net

Yiğit Bul
Hermes Iletisim, Pasaport
Izmir, Turkey
yigit.bul@hermesiletisim.net

Sarper Bağcı
Hermes Iletisim, Pasaport
Izmir, Turkey
sarper.bagci@hermesiletisim.net

Geylani Kardas
Ege University
International Computer Institute, 35100, Bornova
Izmir, Turkey
geylani.kardas@ege.edu.tr

***Abstract*—JsonLogic structures, based on JavaScript Object Notation (JSON), are used in software applications in order to create business process rules. However, JsonLogic's textual syntax is different from the general purpose programming languages and it causes difficulties on the formalization of complex business rules. This unfamiliar way of rule creation may also lead to a time-consuming and error-prone development process. In this paper, we introduce a web based visual modeling tool which facilitates the construction of such business rules by following a model-driven engineering methodology. Inside this tool, the developers can visually design business rules with the block programming approach and corresponding JsonLogic codes are automatically generated. Moreover, changes made in these auto-generated codes can be reflected automatically to the related models inside the tool without any human intervention. Hence the synchronization between JsonLogic models and codes is provided. It has also been found that JsonLogic business rules can be created with significantly fewer visual components and hence with simpler models in comparison with the unique editor currently available for the similar purpose. The modeling tool is now used by Hermes Iletisim company during the development of various commercial software products.**

Keywords— JsonLogic, Business Process Rule, Model-driven Software Development, Domain-specific Modeling Language

I. INTRODUCTION

In Business Process Management (BPM) software, the operating rules of business processes and workflows are mostly structured and managed based on logic principles [1]. There is a need for logic constructions to establish rules for components such as determining the status of tasks and resources, assigning them to actors, specifying operational constraints, directing processes and triggering events.

Utilization of both JavaScript Object Notation (JSON) [2] logic and JsonLogic [3] structure, built on JSON, for the logic-based creation of complex business process rules has been recently introduced during the development of professional BPM software (e.g. IBM BPM [4], Camunda [5]). Although the creation of business process rules with JsonLogic allows these rules to be both stored and shared in various databases within BPMs, the unusual textual syntax of these rules differentiating from the general purpose programming languages makes difficult for software developers to write complex business rules in JsonLogic, and hence causes the rule-making process to be longer and more susceptible to errors. In order to minimize these problems encountered during the creation of JsonLogic business process rules, we presented a domain-specific modeling language (DSML) and discussed how JsonLogic rules can be developed by applying a model-driven engineering (MDE) methodology in our previous study [6]. Use of this DSML was provided via a modeling tool which enables graphically design of business process rules and the corresponding JsonLogic structures are automatically generated for the system implementation. However, this DSML could not support the synchronization between the rule models and the generated JsonLogic codes. Moreover, the graphical syntax is needed to be re-engineered to improve language notations used for the creation of rule models.

This paper introduces a web-based graphical modeling tool, which eliminates the abovementioned deficiencies of the DSML and facilitates model-driven development of JsonLogic business rules with also supporting the reverse engineering of rules to automatically synchronize rule models and JsonLogic structures. That new feature makes both software models and the codes for JsonLogic based business rules consistent in all kinds of modifications. The tool enables the automatic generation of JsonLogic structures from visual models and changes made in auto-generated codes can also be reflected directly in the visual business rule models associated with these codes without any human intervention. Inside the tool, it is also possible to automatically generate the business rule models for any existing JsonLogic structures which previously do not have any model.

The remaining of the paper is structured as follows: Section 2 gives brief information on JsonLogic. The modeling tool developed for the MDE of JsonLogic rules is discussed in Section 3. Section 4 demonstrates how the tool can be used within a case study from the industry. Related work is given in Section 5 and Section 6 concludes the paper.

II. JSONLOGIC

JsonLogic is a structure defined on JSON logic and is used to build complex logic rules, serialize them as JSON data, share between front-end and back-end of applications and store them in databases. Each JsonLogic rule is a JSON object and is defined as an operator-data pair. In a JsonLogic structure, the operator is in the key position while one or a number of arguments are in the value position. Each argument itself can be a logic rule, so it is possible to define complex rules with JsonLogic [3]. Although it is not a complete programming language, JsonLogic is a powerful alternative to rule-based creation of business processes, especially as it allows rules to be stored as data and thus dynamically generated by user interaction.

The following is an example of a simple JsonLogic business rule. In that rule, if a person's e-mail address is not null, and the number of e-mails sent to that person is less

This study is funded by the Scientific and Technological Research Council of Turkey (TUBITAK) Technology and Innovation Funding Programs Directorate (TEYDEB) under grant no: 7180424.

978-1-7281-3992-0/19/$31.00 ©2019 IEEE

than three (that is, the operation result of the rule will be logical true), a new e-mail will be sent to that person.

JsonLogic rule: {"and" : [{ "!=" : [{“var” : “email_address”},null] },{ "<" : [{“var” : “number_of_sent_mail”},3] }] }

JsonLogic rule parsers [3] can transform written rules into statements in e.g. Python, JavaScript, PHP and Ruby and provide the execution of these rules. For instance, Python counterpart of the above JsonLogic rule is achieved as follows:

Python code: ((email_address != null) and (number_of_sent_mail < 3))

III. The Modeling Tool for JsonLogic Rules

MDE of business rules based on JsonLogic is supported with the use of a web based modeling tool which presents a graphical integrated development environment (IDE) for the developers. Fig. 1 shows a screenshot from our IDE which is built upon Google’s Blockly visual programming library [7]. Empowering with the block programming principles, the tool enables the developer to visually model JsonLogic business rules as interlocking blocks.

Graphical notations for JsonLogic components are listed on the left of the tool inside a palette. Modeling elements with similar semantics are categorized inside the palette to facilitate their utilization. This graphical concrete syntax given inside the tool is originated from the metamodel we introduced in [6]. The metamodel defines each *JsonLogic Element* as being an *AccessingData*, *Operation* or *Log*. An instance of an *AccesingData* element processes the data entered by the user and returns the result. Usually each data in object format is defined by a *var* concept with a specific name. Instances of *Operation* meta-entity represent the main elements in a JsonLogic model, and JsonLogic rules are created using the *Operation* variants. These variants are mainly grouped under the following sub entities: *Logic*, *Boolean*, *Numeric*, *Array* and *String*. Finally, *Log* instances are especially used to control complex JsonLogic rules and to display output in debugging. They can produce in logical true / false, number or word types. Further discussion on these metamodel entities and how graphical notations can be derived from this metamodel is found in [6].

A software developer can drag and drop JsonLogic model elements from the palette to the modeling area in the middle region of the IDE to create the desired number of rule instances. While creating JsonLogic business rule models, a number of static semantic controls are also performed automatically by the modeling tool according to a series of language constraints. For example, comparing elements (e.g. ==, !, !!) have two inputs, so the modeling tool allows the developers to use only two-input block notations for these elements. Input type (e.g. var, string, number, array, true and false) checks are also automatically made by the tool during modeling. Another example of static semantics control can be given for the JsonLogic component, called “between”. According to its metamodel definition, the tool allows instances of this component to be included in the model with a maximum of three input blocks. In addition, the inputs of the "between" elements can only be in "number" and "var" types, and only those types of inputs are allowed to be defined for the "between" instances during block designs in the modeling area. Performing these constraint checks automatically during modeling helps developers to achieve accurate JsonLogic rules.

When the model is validated, JsonLogic structures corresponding to modeled business process are automatically generated and shown on the right side of the tool (see Fig. 1). After the business rules are modeled, the model-to-text transformations, which are executed on these models, enable the automatic generation of JsonLogic structures corresponding to the related models. We implemented the transformation engine with using JavaScript language. For each JsonLogic model instance, the transformation engine first determines the model elements and creates JsonLogic codes according to the predefined transformation rules. It is worth indicating that the whole generation process is abstract from the developers and executed behind with using this operational semantics. In other words, it is sufficient for the developer to prepare the visual model of the business rules. He/she does not need to interfere into code generation process.

Finally, the modeling tool supports the reverse engineering in which it is possible to automatically generate graphical models of any business rule given in JsonLogic structures. The benefits of this reverse engineering are twofold: First, the automatic synchronization between business rule models and the corresponding JsonLogic is possible in case one of them is modified e.g. adding a new Operation variant into the model or removing a “var” definition from textual JsonLogic description. Second, it is possible to automatically create the graphical model of business process rules defined in JsonLogic structures from scratch since textual JsonLogic descriptions for a series of business rules may not previously have a model. That also paves the way of integrating pre-written JsonLogic definitions into our modeling environment.

A parser is written again in JavaScript to analyze textual JsonLogic descriptions and determine the fragments from which corresponding visual elements are derived. Relations of the rules are also parsed to realize the interlocking between the generated visual elements, i.e. blocks. That process is also abstract from the developers and executed behind the scene automatically when the developer clicks the button on the top right of the tool (see Fig. 1). Any JsonLogic description can be written or copied and pasted into the text area located at the right of the IDE and the corresponding visual model is created / updated in the modeling area.

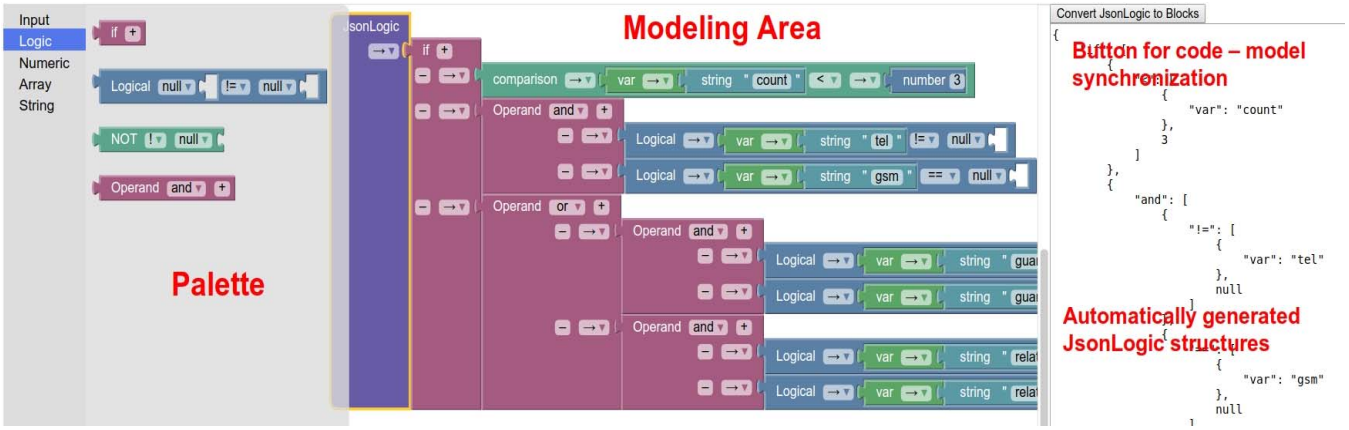

Fig. 1. A screenshot from the modeling tool for JsonLogic based business process rules

IV. Case Study: Model-driven Development of Workflow Models for Debt Collecting Advocacy

In order to give some flavor of using the proposed modeling tool, the model-driven development of business rules composed by a BPM software, called Debt Collecting Advocacy which is one of the commercial products of Hermes Iletisim [8] company, is discussed in this section. That BPM software is currently used by the law offices to enable advocates both creating and managing the complex business process for collecting debts more efficiently.

The main duty of a debt collector advocate is to reach a debtor, collect the debt from this debtor and deliver it to the payees such as individuals or corporations. For this purpose, these advocates can use various channels e.g. phone call, SMS, Voice Message, or national ID SMS to reach a debtor. However, the advocate usually can not reach the debtors by using a single way of communication; in fact, he/she mostly needs reaching the debtor's guarantor, mother, father or other relatives in many different ways. Moreover, these debt collectors should deal with more than ten thousand case files on average which must be handled only in one month. The management of such a business process becomes very complex when the relevant business process rules are not well established.

In the case study, the creation of a rule that should be implemented for the management of the process of reaching a debtor will be given as an example. This process rule is defined by the debt collector advocate. According to this rule and the corresponding job description, it is attempted to reach the debtor's personal phone for three days. If the debtor cannot be reached, this time, the debtor's guarantor or relative's phone will be contacted for three days. The formal JsonLogic definition of the related rule is given below. Software developers, who prepared these business rules in the company, reported that although this rule is one of the simplest rules contained in the relevant BPM software, it is troublesome and time consuming to prepare it accurately with JsonLogic notation.

Rule for reaching the payer: {"if" : [{"<" : [{"var" : "count"},3]},{"and" : [{"!=" : [{"var" : "tel"},null]},{"==" : [{"var" : "gsm"},null]}]},{"or" : [{"and" : [{"!=" : [{"var" : "guarantorTel"},null]},{"==" : [{"var" : "guarantorGsm"},null]}]},{"and" : [{"!=" : [{"var" : "relativeTel"},null]},{"==" : [{"var" : "relativeGsm"},null]}]}]}]}

Fig. 2 shows the model designed by the developer for the above business rule. The developer used the graphical concrete syntax provided by our tool for modeling JsonLogic and created the model given in the middle of the IDE. Following the semantic checks on the model, the tool automatically generated the required JsonLogic structures as can be seen in the right of the IDE. Indentation format of the generated structures is disarranged here to fit into the figure.

As previously discussed, it is possible to automatically synchronize the models and the textual JsonLogic structures in case of any modification. When any structure is modified inside the coding area of the IDE, firstly the syntax of these codes are checked and formatted with proper indentation and then the graphical model of the business rules is refreshed in order to reflect the changes made in the structures. For instance, in the coding area of the IDE, let us suppose that the developer modifies above textual JsonLogic structure as reaching the debtor's personal phone for two days instead of three. Furthermore, he/she also removes the JsonLogic codes for the conditions to access both the debtor's relative and the guarantor at this time. Upon applying these changes into the textual structure, the related graphical business rule model is updated automatically by the tool as can be seen in Fig. 3.

Conducted study also provided the usability evaluation of the proposed modeling tool by comparing it with the JSON editor [9] which solely allows JSON modeling in Blockly. To the best of our knowledge, it is the only editor intended to visually prepare JSON elements and hence we chose this editor for the comparison purpose. The same case study was applied and the above business rule was tried to be modeled in this editor. However, the block model of the rule was approximately three times bigger than the one designed inside our tool by means of the "operation" instances since the editor in [9] does not include the ready-to-use JsonLogic model elements. For instance, each "==" JsonLogic element covered inside the model given in Fig. 2 needed the construction of at least two additional blocks in the JSON Editor.

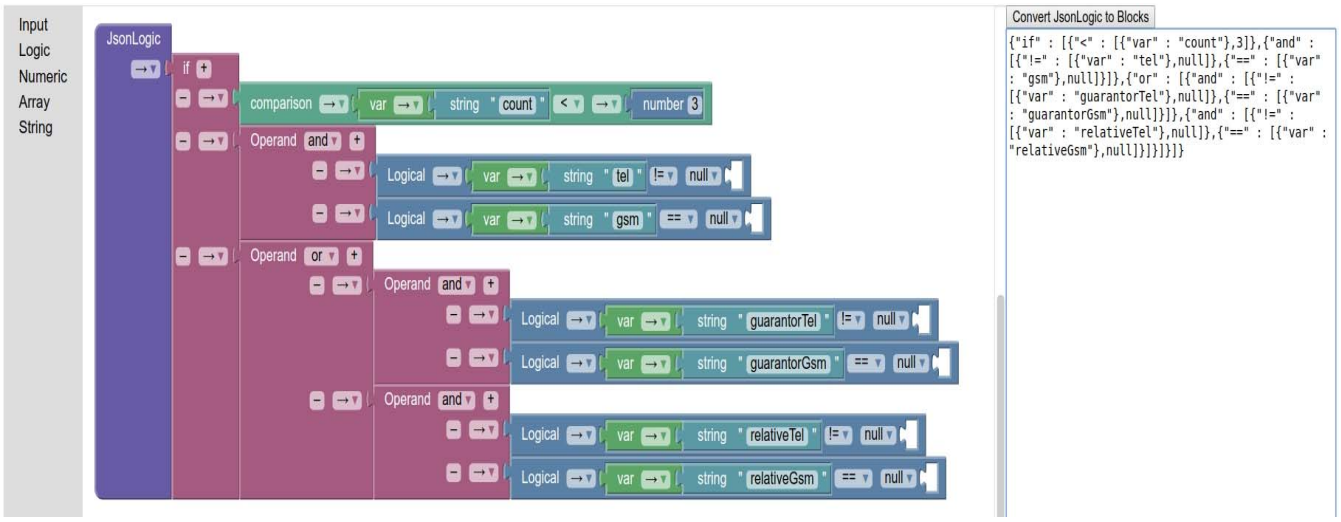

Fig. 2. A screenshot from the modeling tool showing the model-driven development of JsonLogic structures corresponding to a business rule for the management of the process of reaching a debtor

Fig. 3. Automatic update in the graphical models in case of any modification made inside JsonLogic textual structures

V. Related Work

Related work in the literature can be discussed in two categories covering 1) model-driven development of rule-based business processes and 2) JSON / JsonLogic data and logic sharing formats.

In the automatic creation and management of business processes, model-driven development and domain-specific language (DSL) production are becoming increasingly popular, as well as the use of notation and structuring of well-known languages such as BPMN [10] and YAWL [11]. For example, Brambilla and Pietro [12] propose a language called IFML, which enables software applications to be designed specifically for user interaction and to model flows. They illustrate the use of this language in some business areas. Model-driven development of form-based or data-based business applications that interact with back-end systems is described in [13]. Similarly, how the application-specific functions of business applications can be defined and generated automatically on a platform-independent level with a DSL and toolkit, called IIS * CFuncLang, is described in [14] with a case study dealing with the development of an educational information system. A method for mapping BPMN models to Business Process Execution Language (BPEL) models and reflecting updates on BPMN models to BPEL models is presented in [15].

MDD4CCA language [16] supports the model-driven development of composite content applications with using viewpoints such as form, navigation, workflow and content. Similarly, Bicevska et al. [17] enable business process modeling through the use of a DSL, and the definitions of executing business processes from models can be generated to include event-based architecture in their study. Recently, Ferry et al. [18] provide a model-driven approach both to create cloud applications for the Infrastructure as a Service (IaaS) and to manage these applications independent from the cloud service providers. Another DSL, MAML [19], enables both the model-driven development of business processes for different mobile applications and the automatic generation of local source codes for these applications. Finally, Benaben et al. [20] introduce an approach that manages business processes for service-oriented architectures and allows the development of software designs for this architecture based on various levels of abstraction (e.g., logical or technical) with MDE fundamentals.

Although above mentioned studies provide various noteworthy approaches for the model-driven business rule development, the business logic and constraints necessary for the choreography and orchestration of the workflows are mostly derived only for the specific needs of the applications covered in the same studies and much work is required to generalize them. Moreover, none of the above studies

supports the use of JSON data interchange format and/or modeling business logic based on JsonLogic.

Taking into consideration the research on JSON, related studies mostly investigate the ways of how this data interchange format can be used for retrieving data from resources such as databases. For instance, Bourhis et al. [21] propose a structural model for JSON documents and define a query language which can parse JSON documents by using this structural model. Due to the lack of metadata definition, a schema for JSON is defined and the use of this schema to validate JSON documents is described in [22]. In order to query JSON data more efficiently, a query processor is developed [23]. This study shows that, even with very large data sets, data loading costs are reduced by the rewrite rules contained in the proposed processor. However, both visual modeling of JSON structures and automatic generation of these models are not considered in these studies. Similar to our study, there is an editor [9] for modeling JSON structures with the Blockly [7] infrastructure as discussed in the previously section. However, this editor lacks supporting JsonLogic. Differentiating from these existing studies, the modeling tool, we introduce in this paper, enables both model-driven development of JsonLogic structures and the synchronization between JsonLogic models and the textual business rules written in JsonLogic.

VI. Conclusion

A web based modeling tool has been introduced to facilitate the preparation of complex business process rules defined in JsonLogic. It has been found that the relevant business rules can be created with significantly fewer visual components and hence with simpler models in comparison with the unique editor widely used in modeling JSON elements.

The proposed tool can automatically generate JsonLogic structures from graphical models of business process logic rules, as well as the recovery of visual models from existing JsonLogic structures. Thus, it is possible to reflect the changes in JsonLogic codes to rule models and to ensure synchronization between visual JsonLogic models and codes. The modeling tool is currently used by Hermes Iletisim company during the development of various commercial software products.

In order to improve the usefulness of the tool's graphical syntax, our future work consists of evaluating notations currently provided inside the tool in terms of their expression power and adoption by users. To make this improvement, an evaluation of our modeling environment according to Moody's well-known “Physics of Notations” principles [24] is planned.

References

[1] M. Wang and H. Wang, “From process logic to business logic—A cognitive approach to business process management”, Information & Management, vol. 43(2), pp. 179-193, 2006.

[2] JSON, https://www.json.org/, accessed: October 1, 2019.

[3] JsonLogic, http://jsonlogic.com/, accessed: October 1, 2019.

[4] IBM Business Process Manager, https://www.ibm.com/us-en/marketplace/business-process-manager, accessed: October 1, 2019.

[5] Camunda BPM: Workflow and Decision Automation Platform, https://camunda.com/, accessed: October 1, 2019.

[6] K. Soleymanzadeh, Y. Bul, S. Kulduk, S., Bagci, and G. Kardas, “A Modeling Environment for the Development of Business Process Rules with JsonLogic”, In proc. 7th Turkish Software Architecture Conference (UYMK 2018), Istanbul, Turkey, CEUR Workshop Proceedings, vol. 2291, pp. 133-148, 2018.

[7] N. Fraser, “Ten things we've learned from Blockly”. In proc. 2015 IEEE Blocks and Beyond Workshop, Atlanta, GA, USA, IEEE, pp. 49-50. 2015.

[8] HERMES Iletisim, https://www.hermesiletisim.net/, accessed: October 1, 2019.

[9] An editor for JSON structures, http://ens-lg4.github.io/MenulyJSON/, accessed: October 1, 2019.

[10] M. Chinosi and A. Trombetta, “BPMN: An introduction to the standard”, Computer Standards & Interfaces, vol. 34(1), pp. 124-134, 2012.

[11] W. M. P. van der Aalst and A. H. M. ter Hofstede, “YAWL: yet another workflow language”, Information Systems, vol. 30(4), pp. 245-275, 2005.

[12] M. Brambilla and P. Fraternali, “Interaction Flow Modeling Language: Model-Driven UI Engineering of Web and Mobile Apps with IFML”, Morgan Kaufmann, Waltham, MA, USA, 2015.

[13] T. A. Majchrzak, I. Ernsting and H. Kuchen, “Achieving business practicability of model-driven cross-platform apps”, Open Journal of Information Systems, vol. 2(2), pp. 3-14, 2015.

[14] A. Popovic, I. Lukovic, V. Dimitrieski and V. Djukic, “A DSL for modeling application-specific functionalities of business applications”, Computer Languages, Systems & Structures, vol. 43, pp. 69-95, 2015.

[15] G. Radhakrishnan, L. Liu, A. Aggarwal and V. Saxena, “Roundtrip merge of bpel processes and bpmn models”, US patent no: US20100057482A1, 2010.

[16] M. Challenger, F. Erata, M. Onat, H. Gezgen and G. Kardas, “A Model-Driven Engineering Technique for Developing Composite Content Applications”, In proc. 2016 Symposium on Languages, Applications and Technologies (SLATE 2016), Maribor, Slovenia, pp. 11:1-11:10, 2016.

[17] Z. Bicevska, J. Bicevskis and G. Karnitis, G. “Models of Event Driven Systems”, Communications in Computer and Information Science, vol. 615, pp. 83-98, 2016.

[18] N. Ferry, M. Almeida and A. Solberg, “The MODAClouds Model-DrivenDevelopment”, In: Di Nitto E., Matthews P., Petcu D., Solberg A. (eds): Model-Driven Development and Operation of Multi-Cloud Applications. pp. 23-33, SpringerBriefs in Applied Sciences and Technology. Springer, Cham, 2017.

[19] C. Rieger and H. Kuchen, “A process-oriented modeling approach for graphical development of mobile business apps”, Computer Languages, Systems & Structures, vol. 53, pp. 43-58, 2018.

[20] F. Benaben, S. Truptil, W. Mu, H. Pingaud, J. Touzi and V. Rajsiri, J. P. Lorre, “Model-driven engineering of mediation information system for enterprise interoperability”, International Journal of Computer Integrated Manufacturing, vol. 31(1), pp. 27-48, 2018.

[21] P. Bourhis, J. L. Reutter, F. Suarez and D. Vrgoc, “JSON: Data model, Query languages and Schema specification”, In proc. 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Chicago, IL, USA, ACM, pp. 123-135, 2017.

[22] F. Pezoa, J. L. Reutter, F. Suarez, M. Ugarte and D. Vrgoc. “Foundations of JSON Schema”, In proc. 25th International Conference on World Wide Web, Montreal, Quebec, Canada, ACM, pp. 263-273, 2016.

[23] C. Pavlopoulou, E. Preston Carman, Jr, T. Westmann, M. J. Carey and V. J. Tsotras, “A Parallel and Scalable Processor for JSON Data”, In proc. 21st International Conference on Extending Database Technology, Vienna, Austria, pp. 576-587, 2018.

[24] D. Moody, “The “physics" of notations: toward a scientific basis for constructing visual notations in software engineering”, IEEE Transactions on Software Engineering, vol. 35(6), pp. 756-779, 2009.