International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:11, No:3, 2017

JREM: An Approach for Formalising Models in the Requirements Phase with JSON and NoSQL Databases

Aitana Alonso-Nogueira, Helia Estévez-Fernández, Isaías García

***Abstract*—**This paper presents an approach to reduce some of its current flaws in the requirements phase inside the software development process. It takes the software requirements of an application, makes a conceptual modeling about it and formalizes it within JSON documents. This formal model is lodged in a NoSQL database which is document-oriented, that is, MongoDB, because of its advantages in flexibility and efficiency. In addition, this paper underlines the contributions of the detailed approach and shows some applications and benefits for the future work in the field of automatic code generation using model-driven engineering tools.

***Keywords*—**Conceptual modeling, JSON, NoSQL databases, requirements engineering, software development.

I. Introduction

SOFTWARE development has a lot of methodologies, both traditional and agile [1] and the most popular one nowadays is Scrum [2], an agile methodology. However, all of them have the same processes:

1) Requirements analysis resulting in a software requirements specification.
2) Software design. In this step, the system architecture is designed in order to guide its development.
3) Implementation. The code of the application is written.
4) Testing of the implementation in order to make sure that it fits with the specification given.
5) Integration, if there are multiple subsystems.
6) Deployment (or Installation).
7) Maintenance of the system: Fixing errors and developing improvements.

All of these processes have a huge impact on the final product. They are also dependent on each other: A proper software design cannot be done without a concise requirements specification; a good implementation cannot be carried out without the proper software design; and so far and so forth. As a result, we may agree that the first step is the most important one since nothing successful can be built without it.

With regards to Requirements Engineering (RE), the most accurate definition could be the one given by [3] which says: 'Requirements engineering is the branch of software engineering concerned with the real-world goals for, functions of, and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behaviour, and to their evolution over time and across software families'.

RE is usually divided into six activities: Elicitation or discovery, analysis and reconciliation, specification, modeling or representation, verification and validation and management. These activities should be carried out in sequential order and all of them must be taken into account in the software development.

Some of the biggest problems that we face working in RE fields are the ambiguity, inconsistency and incompleteness of the requirements; integration between them and the architecture of a system; communication of requirements; and model interoperability [4].

There are many works in RE, most of which study the elicitation of requirements from natural language [5], [4]. Nevertheless, the research community has made a great effort trying to fix the ambiguity errors that arise when trying to merge the requirements specified by different people. Most solutions to this problem are based on ontologies like [6] or [7]. But deep down there is still the same problem, merging different requirements specifications causes ambiguity problems in each of the applications that are developed. To reduce this problem, we propose to use an Open Data Model (ODM) [8] that could be used as a software design for all applications. ODM is described in Section II *C*.

This paper aims to describe the tool that we have developed. It might help us to create and manage the model. Due to the quick evolution of the RE needs, where documents have become less important, new RE tools should be lightweight and user-friendly [9] to lighten the improvements and changes of the system. This tool has been developed using JavaScript and it stores the data in JavaScript Object Notation (JSON) documents. Those technologies are also briefly described in this paper.

The paper is structured as follows: Section II provides a short introduction to the different subjects we need to know before introducing the approach: modeling methods, JSON, ODM and MongoDB. Section III shows how the approach formalises a model in JSON to be used as an ODM. Section IV illustrates how the approach can be applied to a concrete case study of a library. Finally, Section V contains the conclusions and Section VI provides some future lines.

Aitana Alonso-Nogueira, PhD candidate, is with the Universidad de León, Spain.
Helia Estévez-Fernández, PhD candidate, is with the Universidad de León, Spain (e-mail: helia.estevez@unileon.es).
Isaías García, PhD, is with the Dept. of Electrical and Systems Engineering, Universidad de León, Spain.

II. Background

This section presents different subjects to help the reader understand the rest of the paper better.

A. Modeling Methods

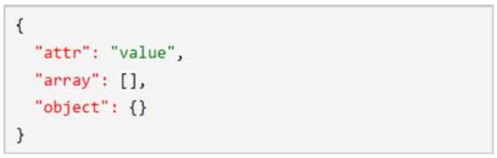
The first activity in RE is to collect the requirements of the software from users, customers and other stakeholders. This process is well known as elicitation and it results in the specifications of the requirements. Then these specifications are modeled using certain tools and modeling languages.

In our context, modeling can be defined as the process that aims to express the requirement specification in a structure defined by a consistent set of rules. This structure is called model and is built using modeling languages. Some of them are: Ecore [10], AADL [11], SPEM [12] or SYSML [13]. However, the most used language to represent object-oriented models is Unified Modeling Language [14] which is a standard. Moreover, it can be aided by Object Constraint Language (OCL) and the meta-models generated with Ecore can be applied to UML models.

There are many solutions to elicit software specifications captured in natural languages (NL); some of them follow the Object-Oriented Analysis and Design (OOAD) [15] to design class models. Examples of these approaches are: UMGAR [16] can generate the Use-case diagram, the analysis class model, the collaboration diagram and the design class model. RAUE [17] makes requirement analysis and UML diagram extraction. SUGAR [18] is a static UML model generator from analysis of requirements. RAPID [19] assists analysis by providing an efficient and fast way to produce the class diagram from their requirements. LOLITA [20] generates object models. LIDA tool [21] helps analysts to identify type elements in the object-oriented model like classes, attributes, roles etc.. NL2UMLviaSBVR [22] generates an SBVR representation and then, a UML diagram. All these frameworks, tools, approaches or methods identify use-cases, actors, classes with its attributes and methods and the relations among classes: Associations, generalizations or aggregations from NL. Some of them use these results to build a model, commonly UML, others only return them.

B. JavaScript Object Notation (JSON)

JSON is a lightweight data-interchange format. It is processable by machines and understandable by humans. JSON is built on two structures: *collection of name-value pairs* and *ordered list of values.* In most languages, the former structure is realised as an object while the latter is realised as an array. An example of a JSON document is depicted in Fig. 1.



```
{
  "attr": "value",
  "array": [],
  "object": {}
}
```

Fig. 1 Different values in a JSON document

JSON has five types of values: String, number, array, object and literal name. String is a sequence of characters; number is a number written in decimal or exponential notation; array is a sequence of values that do not have to be the same type; object is a sequence of properties; and literal name is one of the three special values null, true and false. Properties of an object are a name-value pair. To have more detailed information, the specification is in [23]

The approach presented here takes advantage of certain benefits from [24], such as:

1) JSON is simpler than other formats like XML, so it makes possible for people who are less experienced to understand it.
2) It is the preferred format for exchangeable data because it is faster, which is useful with requirements considering that they must be shared between different stakeholders and applications.
3) There is no need for a parser as the interpreter decodes the data like native JavaScript object, which is good because our model is object oriented.

C. Open Data Model

ODM ensures a standard format for the application’s data structures which makes the data easy to share (Fig. 2). Reference [5] proves that there is a lack of open data mechanisms in the literature and some of the studies use a specific API instead.

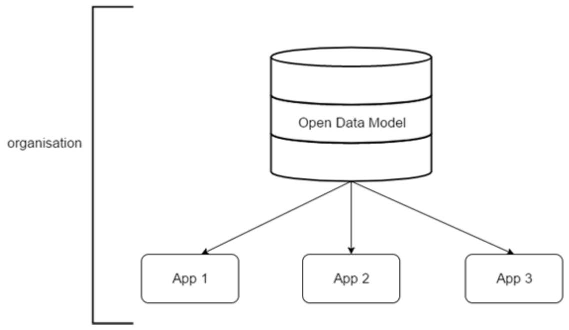


Fig. 2 All of the applications of an organisation can use the ODM

The well-known benefits of using ODM are:

1) The transparency of the data. Being open makes it accessible by all the members of the organization with access to the database.
2) Improving effectiveness of services. It would avoid duplicated fields, naming conflicts, and other semantical ambiguities which will cause problems in the software development.
3) Improving efficiency of services. Due to the fact that everyone uses the same model, concepts that often appear in different systems have to be defined only once. It could improve the efficiency of the process considerably.

D. MongoDB

MongoDB [25] is the main database of the NoSQL databases. These databases do not store the data in tables as relational databases do. MongoDB stores the data within

JSON documents so it is known as a document-oriented database. In fact, it stores BSON (Binary JSON) which is the binary form for representing simple data structures. BSON makes MongoDB more efficient in storage and scan-speed.

NoSQL database is used instead of relational database because of its flexibility - which allows storing scheme-less data and easy expanding, its low cost, and because reading and writing is quicker than in relational databases. Although it is not concerned about high-performance reading and writing concurrent, it ensures a good query performance [26] that will be useful in multiple usages of the formalised model.

As a brief overview of MongoDB we must know that it stores BSON documents in collections and the collections in databases. These collections are analogous to tables in relational databases.

III. Our Approach

Taking into consideration the problems associated with RE in Software Engineering, we present an approach whose goal is to minimize its problems effects, such as time-consuming difficulty and ambiguity.

We intend to save the requirements extracted in the elicitation phase in a data model which will be common to all applications within an organization. In the elicitation process, existing tools previously presented in Section II *A* are used.

The model created with our JREM tool will be an ODM that will be formalised in JSON due to the advantages mentioned in Section II *B*. To facilitate the access to these documents, they will be lodged in a NoSQL database; in particular a document-oriented database called MongoDB and explained in Section II *D*. Our model will be lodged in MongoDB because is the main NoSQL document-oriented database and many organizations are considering using it instead of relational databases.

As we have already said, our database is structured in a collection of class models (Fig. 3 (a)), so each class model is a document in the database, with the structure shown in Fig. 3 (b).

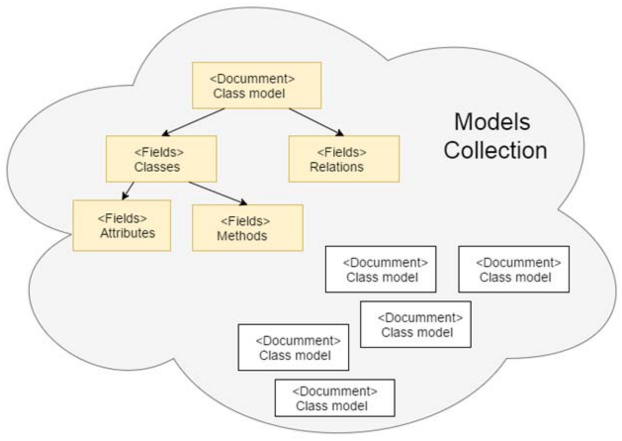


Fig. 3 (a) Structure in MongoDB of stored class models

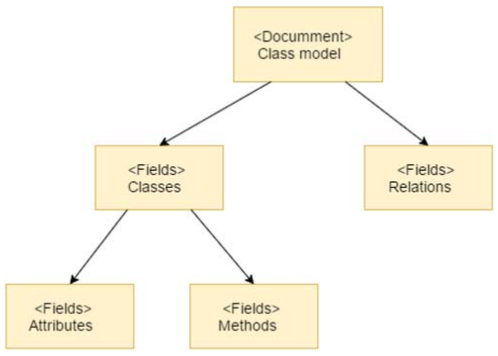


Fig. 3 (b) Structure of each class model

To prove our approach we have developed an online tool which is a middleware, where the user through a user-friendly interface can introduce these resulted elements, obtained using previous elicitation and modeling methods, and generates a formalization class model in JSON following the Class Diagram Guidelines [27].

The tool has been built to be an online application, which makes it accessible by all users connected to the net. Hence, the languages of the web are used, such as JavaScript, HTML and CSS. The principal advantage of using JavaScript with JSON is that JSON is based on features that are inherent in the JavaScript language [24] which makes the whole process faster and more direct.

It has been developed using the MEAN Stack [28]. MEAN stands for MongoDB, Express [29], AngularJS [30] and Node.js [31]. The front-end of the application uses AngularJS. AngularJS is a JavaScript framework that helps building dynamic views in web-applications and eases the control of the data. On the server side, the tool uses Node.js as an interpreter that is designed to build scalable applications. It is used along with Express, a minimal Node.js web application framework which facilitates its management. Finally, the server-side connects the application with a database MongoDB, already described in Section II *D*. In order to use MongoDB within the application, we choose Mongoose [32] as a tool for modeling objects and inserting them into the database.

Speaking of the User Interface (UI), we have used Bootstrap [33], an HTML, CSS and JavaScript framework which allows programmers to develop responsive and mobile projects on the web. The UI of the tool is shown in Figs. 4 and 6.

So far, only the manual part is implemented and the human intervention is necessary. However, it will be possible in the near future to develop a parser tool or plug-in from other formats to JSON. Moreover, due to the fact that JSON is an automatic format that can be processed, it is possible to create a translation from the model in JSON to another language or model. Hence, we could use a method to extract the model elements, and then we use our tool to formalize and transform them to UML or other models.

IV. A CASE STUDY

This paper shows the tool working with a case study presented originally in [34] and then solved in [35], [23], [36]. Its statement is as follows:

> A library issues loan items to customers. Each customer is known as a member and is issued a membership card that shows a unique member number. Along with the membership number other details on a customer must be kept such as a name, address, and date of birth. The library is made up of a number of subject sections. Each section is denoted by a classification mark. A loan item is uniquely identified by a bar code. There are two types of loan items, language tapes, and books. A language tape has a title language (e.g. French), and level (e.g. beginner). A book has a title, and author(s). A customer may borrow up to a maximum of 8 items. An item can be borrowed, reserved or renewed to extend a current loan. When an item is issued the customer's membership number is scanned via a bar code reader or entered manually. If the membership is still valid and the number of items on loan less than 8, the book bar code is read, either via the bar code reader or entered manually. If the item can be issued (e.g. not reserved) the item is stamped and then issued. The library must support the facility for an item to be searched and for a daily update of records.

The different approaches we have mentioned do the object-oriented analysis and give the results shown in Table I. Once we have obtained these results, they are used as an input to our tool. The first step is to create a new model whose name could be *LibraryModel* (Fig. 4).

TABLE I
OBJECT ORIENTED ANALYSIS RESULTS

Type	Names
Classes	Library, Loan_item, Member, Member_number, Customer, Book, Language_tape, Barcode_Reader, Subject_section, Membership_Card
Attributes	name, address, date-of-birth, barcode, classification_mark, title, author, level, membership_number, valid
Methods	issue(), show(), denote(), identify(), extend(), scan(), enter(), read_barcode(), stamp(), search(), update()
Associations	Library issues Loan_items; Membership_Card issued to Member; Library made up of Subject_section; Customer borrow Loan_item; Customer renew Loan_item; Customer reserve Loan_item; Library support facility
Generalizations	Loan_item is type-of Language_tape; Loan_item is type of Books
Aggregations	-
Instances	-

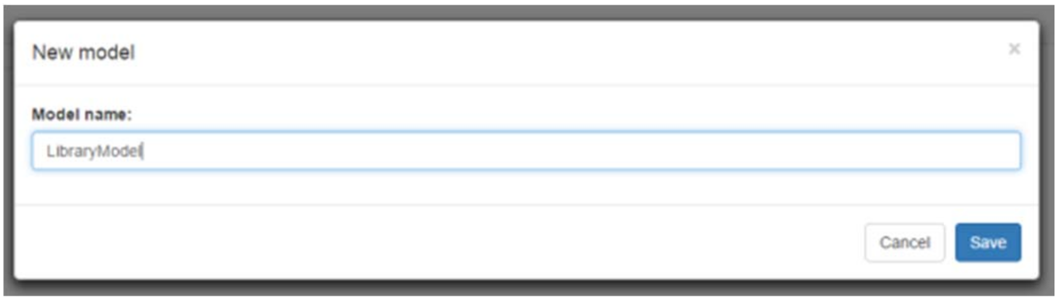
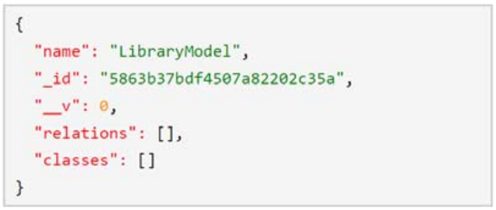


Fig. 4 Creating a new model

The formalization of an initialization model would be similar to Fig. 5. As the figure shows, *name* attribute is a String and *relations* and *classes* attributes are arrays. The *_id* attribute is an object identifier auto generated by MongoDB. It is a hexadecimal number of 12 bytes which can be replaced by the user at any time. The *_v* key contains the internal revision of the document; it is stored and generated by mongoose.



```
{
  "name": "LibraryModel",
  "_id": "5863b37bdf4507a82202c35a",
  "__v": 0,
  "relations": [],
  "classes": []
}
```

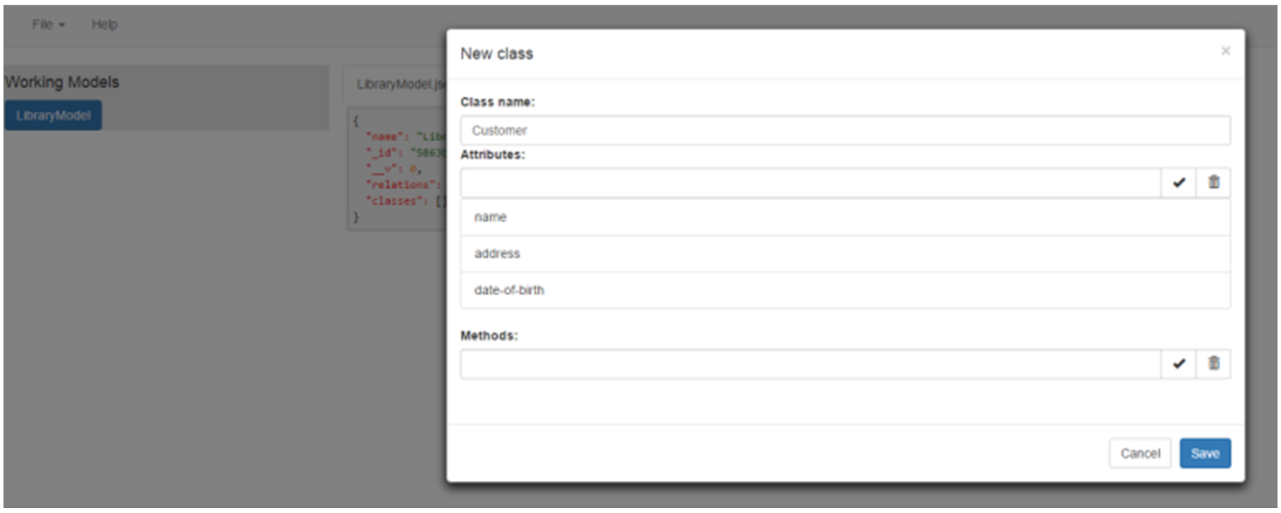
Fig. 5 Model JSON document

In order to build the model, we should add inside *LibraryModel* all the different classes and relationships between classes. The classes will be added with its attributes and methods. For example, we add the class *Costumer* with its attributes: *name*, *address* and *date-of-birth* (Fig. 6). In the same way we will add the class *Loan_item* and the relation between them: *Customer borrrow Loan_item*.

Another advantage of the approach is that the tool saves live models in a MongoDB database and as a result those models are ready to be shared between different clients.

V. CONCLUSION

Software development is in steady progress and there is a lot of work to do in the field. In the requirements phase, the main problem that we identify is the diversity of methods and tools used and the ambiguity that is generated when we try to mix them. We believe that by using an ODM, the problem is solved. Besides, the time that is spent processing the same requirements for different applications or for different organization could be avoided using a common database. Moreover, we have developed a tool that, given certain requirements of an application, formalises them in JSON and stores them in a MongoDB. The tool is online, so it could be used by anyone with authorization to do it, and the generated data are securely stored and shared. The tool has an easy user interface to facilitate its use and it has been developed with the latest technologies which allow it to be responsive and adaptive.



```
{
  "name": "Lib
  "_id": "5863
  "__v": 0,
  "relations":
  "classes": []
}
```

New class

Class name:

Customer

Attributes:

name

address

date-of-birth

Methods:

Fig. 6 Inserting *Costumer* class with the tool

VI. Future Lines

The approach presented in this paper is still improving. We aim to develop a completely independent and automatic tool that helps developers and engineers in their work.

In the near future we hope to join this approach with the Model Driven Architecture. Therefore, having stored the formalized requirements in an ODM, we will be able to generate the code automatically. This code could be written in any language: Java, C, and so on.

Acknowledgment

The authors of this paper thank the "Instituto de Automática y Fabricación" of the Universidad de León, and Prof. Ángel Álonso Álvarez for the essential support given.

References

[1] Jovanovich, D., & Dogsa, T. "Software Development: Agile vs. Traditional". *In Proceedings of the 7th International Conference on* pp. 11-13, 2003.

[2] Schwaber, K. "Scrum development process". *In Business Object Design and Implementation. Springer London*. pp. 117-134, 1997.

[3] Zave, P. "Classification of research efforts in requirements engineering", *ACM Computing Surveys*, pp. 315-321, 1997.

[4] Dermeval, D., Vilela, J., Bittencourt, I. I., Castro, J., Isotani, S., Brito, P., & Silva, A. "Applications of ontologies in requirements engineering: a systematic review of the literature." *Requirements Engineering*, pp. 1-33, 2015.

[5] De Gea, J. M. C., Nicolás, J., Alemán, J. L. F., Toval, A., Ebert, C., & Vizcaíno, A. "Requirements engineering tools: Capabilities, survey and assessment." *Information and Software Technology*, 54(10), 1142-1157, 2012.

[6] Boukhari, I., Bellatreche, L., & Jean, S. "An ontological pivot model to interoperate heterogeneous user requirements." *In International Symposium on Leveraging Applications of Formal Methods, Verification and Validation Springer Berlin Heidelberg*. pp. 344-358, 2012.

[7] Cardei, I., Fonoage, M., & Shankar, R. "Model based requirements specification and validation for component architectures." *In Systems Conference, 2nd Annual IEEE* (pp. 1-8). IEEE, 2008.

[8] Myers, B. A. "The case for an open data model" *(No. CMU-CS-98-153). Carnegie-Mellon Univ Pittsburgh PA School of Computer Science*, 1998.

[9] de Gea, J. M. C., Nicolás, J., Alemán, J. L. F., Toval, A., Ebert, C., & Vizcaíno, A. *Requirements engineering tools. IEEE software*, 28(4), 86-91, 2011.

[10] Zhang, J., Brand, M. V. D., Şutîi, A. M., & Hamilton, M. "Pattern specification and application in meta-models in Ecore". *In Proceedings of the 1st Industry Track on Software Language Engineering (pp. 3-12). ACM.* 2006.

[11] Feiler, P. H., Gluch, D. P., & Hudak, J. J. "The architecture analysis & design language (AADL): An introduction (No. CMU/SEI-2006-TN-011)". *Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst.* 2006.

[12] OMG, S., & Notation, O.M.G. "Software & Systems Process Engineering Meta-Model Specification". *OMG Std., Rev, 2.* 2008.

[13] Friedenthal, S., Moore, A., & Steiner, R. "A practical guide to SysML: the systems modeling language". *Morgan Kaufmann.* 2014.

[14] "UML" (Online: http://www.uml.org). Accessed on 31/01/2017.

[15] Bennett, S., McRobb, S., & Farmer, R. "Object-oriented systems analysis and design using UML". *McGraw Hill Higher Education*. 2005.

[16] Deeptimahanti, D. K., & Babar, M. A. "An automated tool for generating UML models from natural language requirements". *IEEE/ACM International Conference on Automated Software Engineering*, pp. 680-682, 2009

[17] Joshi, S. D., & Deshpande, D. "Textual requirement analysis for UML diagram extraction by using NLP". *International Journal of Computer Applications*, 50(8), 2012.

[18] Kumar, D. D., & Sanyal, R. "Static UML model generator from analysis of requirements (SUGAR)". In Advanced Software Engineering and Its Applications, pp. 77-84, 2008.

[19] More, P., & Phalnikar, R. "Generating UML Diagrams from Natural Language Specifications". *International Journal of Applied Information Systems, Foundation of Computer Science*, 1(8), 2012.

[20] Mich, L., & Garigliano, R. "A linguistic approach to the development of object oriented systems using the NL system LOLITA". *In Object-Oriented Methodologies and Systems. Springer Berlin Heidelberg*, pp. 371-386, 1994.

[21] Overmyer, S. P., Lavoie, B., & Rambow, O. "Conceptual modeling through linguistic analysis using LIDA". *In Proceedings of the 23rd international conference on Software engineering. IEEE Computer Society*, pp. 401-410, 2001.

[22] Bajwa, I. S., & Choudhary, M. A. "From natural language software specifications to UML class models". *In International Conference on Enterprise Information Systems. Springer Berlin Heidelberg*, pp. 224-237, 2011.

[23] Bray, T. (2014). "The JavaScript Object Notation (JSON)" *Data Interchange Format* (No. RFC 7158).

[24] Benson, T., & Grieve, G. "Principles of health interoperability: SNOMED CT, HL7 and FHIR." *Springer,* pp. 74–81, 2016.

[25] "MongoDB". (Online: https://www.mongodb.com). Accessed on 31/01/2017.

[26] Han, J., Haihong, E., Le, G., & Du, J. "Survey on NoSQL database." *Pervasive computing and applications (ICPCA), 6th international conference on. IEEE,* pp. 363-366, 2011.

[27] Amber S.W.: "UML 2 Class diagram Guidelines", (Online: http://www.agilemodeling.com/style/classDiagram.htm), 2016. Accessed on 20/12/2016

[28] "Mean Stack", (Online: http://www.mean.io). Accessed on 31/01/2017.

[29] "Express", (Online: http://www.expressjs.com). Accessed on 31/01/2017.

[30] "AngularJS", (Online: https://angularjs.org). Accessed on 31/01/2017.

[31] "Node.js", (Online: http://nodejs.org). Accessed on 31/01/2017.

[32] "Mongoose", (Online: http://mongoosejs.com). Accessed on 31/01/2017.

[33] "Bootstrap", (Online: https://getbootstrap.com). Accessed on 31/01/2017.

[34] Callan. R.E. "Building Object-Oriented Systems: An introduction from concepts to implementation in C++." *In Computational Mechanics Publications*, 1994.

[35] Harmain, H. M., Gaizauskas R. "CM-Builder: A Natural Language-Based CASE Tool for Object- Oriented Analysis." *Automated Software Engineering*. 10(2):157-181, 2003.

[36] Sharma, R., Srivastava, P. K., & Biswas, K. K. "From natural language requirements to UML class diagrams". *Second International Workshop on Artificial Intelligence for Requirements Engineering (AIRE). IEEE*. pp. 1-8, 2015.