Fast Prototyping of Web Applications Running Business Processes from a Specification Written in Restricted Natural Language

Jean Pierre Alfonso Hoyos, Felipe Restrepo-Calle

***Abstract*—In this paper is shown the process of obtaining automatically an executable web application from a restricted natural language specification. first, an introduction to the problem and its relevance is made, then similar approaches previously made are described. next the proposal of this paper is described, where the starting point are known specification languages; after this explanation an implementation is made and some results are shown.**

***Index Terms*—Requirements, Natural language specification, natural language processing, compilers, software construction automatization, software model automatic construction**

I. Introduction

Software development life cycle (SDLC) activities include requirements analysis, design, development, testing and maintenance. During the requirements analysis, activities such as requirements elicitation, refinement, selection and control are carried out with the purpose of coordinating and organizing efforts to develop the requested software. Later in this cycle, a set of models are built in order to begin the translation of customer needs (or desires) into a computer implementable software [1]. Both of these two steps (requirements and design) have wide impact in the project success, and errors in these early stages can have large impact in the project duration and budget [2].

Requirements elicitation tasks can have errors due to incompleteness or ambiguities in the requirements [2]. The designer's lack of domain knowledge and limited communication with the stakeholders can also introduce errors in software [2], [3]. To overcome these difficulties, diverse approaches have been proposed including restricting the way requirements are written [4], and automate the construction of models from texts [5], [6].

The design stage can be seen as a translation of text describing the software into models like UML (Unified Modeling Language), BPMN (Business Process Model Notation), or E-R (Entity Relation) diagrams. Defining a possible model from a list of system requirements is a task that is usually made manually with the support of some CASE tools such as Rational Rose[1] or Visual Paradigm[2]. Thus, this process requires someone with knowledge of the target system environment and a modeling language (or specification language) [3].

Software development methodologies have been widely used to control the development process in a systematic way. For large projects, methodologies like RUP are used in order to manage the SLDC in large work groups and obtain high quality software [7]. Nevertheless, using traditional methodologies, the SDLC persists being slow and requires to complete a full development cycle to validate the software [7], [8].

To mitigate these problems, various strategies like fast prototyping and agile software development methodologies have been proposed [9]. However, these agile methodologies are more suitable for small teams and small-scale value-oriented software projects, and left behind big scale or mission critical software [8], [10]. Moreover, fast prototyping approaches attempt to make SDLC faster, less sensible to human errors and less sensible to natural language inherent ambiguities and incompleteness. Therefore, they can reduce production costs, time to market delivery and prototyping costs [11]. In many cases, these approaches use a specification written in natural language (restricted or unrestricted) to develop software pieces. This permits to achieve an fast validation of the software and more direct feedback from stakeholders [5], [12].

Three main ways of achieving (semi-)automatic fast prototyping from a specification written in natural language where identified: text mining, using templates to write the requirements, using restricted natural language. The first approach, text mining, uses natural language processing techniques to extract design information from textual specifications [12]. The second one consists of restricting the way each requirement is written to match a template, putting the design information in fixed positions [5], [13]. Third, the specification is written in a specification language, which is proposed restricting the natural language constructs [14].

The first approach makes easy for any person to describe the intended structure of some software, even with some writing errors, information still can be used [6]. In the downside, sometimes a training data set [15] and/or a set of ontologies [16] is needed to achieve acceptable results. In addition, this method is the slowest of the three [15].

In the second approach, using templates to write requirements, although the description of the software is more difficult to build, it is as easy to understand by the stakeholders as in the first approach [17]. The use of templates facilitates the processing tasks, making this approach much faster than the first one. Its main disadvantage is that designers must be very careful writing requirements or some information may be lost

[1]http://www-03.ibm.com/software/products/en/enterprise

[2]https://www.visual-paradigm.com/

[18].

In the case of the third approach, using a restricted natural language, writing requirements is very similar to work with a programming language, avoiding ambiguities and facilitating the processing [19]. Nonetheless, having to learn an artificial language is an inconvenient for the designers and the stakeholders [14].

In this context, this paper presents an approach to achieve a web application prototype running business processes automatically using a restricted natural language specification. To achieve this goal two different specification languages and transformations between natural language are proposed. This work proposes the use of templates and a restricted natural language, this is, a hybrid between the second and third approaches explained above. The main contribution of this paper is a fast prototyping method for web applications running business processes using a restricted natural language specification.

This paper is structured as follows. The second section describes the related works. The third section presents our proposal. The fourth section shows the implementation details. Later, the fifth section explores the results of a case study. Finally, the sixth section summarizes some concluding remarks and suggests the future works.

II. Related Works

Three main approaches where identified to get a model or source code from some form of natural language (restricted or unrestricted):

First, to extract a model from an **unrestricted natural language text**, Two papers show the ideas that later were automated (Abbott [12], Saeki et al. [20]). These two works were manual approaches where words matching certain part of speech and phrases in the description will be used to build a model.

To automate this approach, natural language processing techniques are used, in some cases will need a set of predefined knowledge in order to extract relations between terms [21], in addition current tools can be imprecise leading to induce interpretation errors of the specification [15].

Secondly, many have tried to **restrict the way requirements are written**. In this way the information to achieve a design can be easily extracted from requirements lists [5], [22]–[26]. This approach have the downside of having to define each template in order to extend the proposed solution, and requirements not matching any template will be discarded leaving information outside of the model [18]. Table 1 shows some examples of templates used in related works.

Third, restricting the grammar of a natural language such as English, can lead to overcome the ambiguity and imprecision of such languages. Also can be easier to process. Attempto is an executable **specification language** created restricting the grammar of English language. This approach is slower generating code than the second but faster than the first [14].

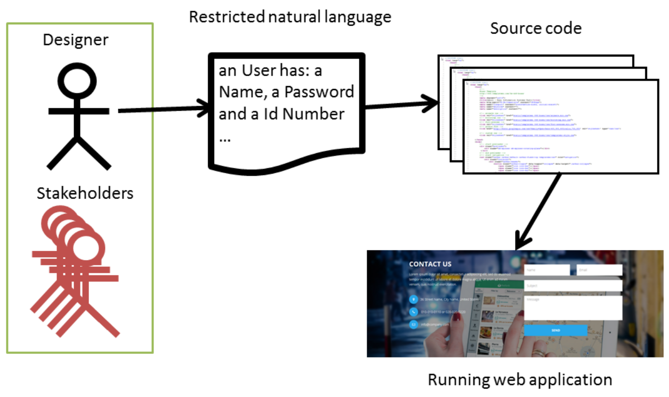
Fig. 1. Example templates for requirements templates approach

Work	Example
Nishida et al. [5]	<procedure sentence>::= <procedure predicate verb><term> [(to|from|with)<term>]* [in<format>] [by<procedure name>] <procedure sentence>, and store the result in <term>
Zapata et al [13].	A <ESTA CONFORMADO POR> B
Zeaaraoui et al. [25]	As a <role>, I want to <action-1> and <action-2> <action-n> <object>, so that <business value>
Videira et al. [27]	<Entity Inheritance Definition> : <Entity> is a <Entity>
Konrad and Cheng [23]	"it is always the case that " (durationCategory | periodicCategory | re- altimeOrderCategory)

III. Proposal

The main idea of this paper is to generate a prototype web application using the model-view pattern, implementing a set of business processes operating over a set of domain classes, using only a restricted natural language specification. A general schema is shown in Fig. 2. where a designer writes the specification, next the source code of the web application is generated and executed.

Fig. 2. Proposal


Restricted natural language
Designer
an User has: a Name, a Password and a Id Number ...
Source code
Stakeholders
CONTACT US
Running web application

A. *Restricted Natural Language*

BPMN is a graphical specification language designed to show the flow of activities, decisions and events that occur in an organization in order to generate some form of business value.

Let b be a well constructed BPMN model, and let $f(b)$ a function that maps b to source code, now let t_1 be a restricted natural language specification such that $g(t_1) = b$ where g is a function transforming the input text t_1 in a BPMN model. An implementation of this function can bee seen in Friedrich et al. [28].

$$f(b) = s_1 \tag{1}$$

$$g(t_1) = b \tag{2}$$

E-R diagrams are graphical specifications that model the relations between information entities within a system and the information contents of each of these entities.

Similarly let e be a well constructed E-R model, $h(e)$ a function that generates source code, t_2 a restricted natural language specification such that $j(t_2) = e$ where j is a function transforming t_2 in a E-R model. An implementation of this function is made by Geetha and Mala in [29].

$$h(e) = s_2 \tag{3}$$

$$j(t_2) = e \tag{4}$$

Now, evaluation of functions $f(g(t_1))$ and $h(j(t_2))$ will return source code from the natural language specification.

Note that t_1 and t_2 can be unrestricted specifications, and the previous reasoning still applies, but for the proposes of this paper, some extra work is done in order to overcome limitations in the two models.

Suppose that a pair of functions $g^{-1}(b)$ and $j^{-1}(e)$ exist ($e \in E{-}R, b \in BPMN$), meaning that there is a projection of the models in natural language. The creation of these functions can be viewed as a ”design” process starting from the model space to the natural language space.

$$g^{-1}(b) = t_1 \tag{5}$$

$$j^{-1}(e) = t_2 \tag{6}$$

This projections are not unique, but some subset of natural language can be selected in a way that is not ambiguous and can be used to transform both models to a single common representation: restricted natural language.

Having both models in the same space can be useful to mix them in a way that exceeds the code generation capabilities of both models separated, and with no ambiguity restrictions a traditional parser can be used to process the natural language specification.

Also note that the function to map natural language to models (g and j) are not needed because the restricted natural language space will not be ambiguous, thus code can be generated directly from the restricted natural language specifications.

1) From E-R diagrams to Restricted NL ($j^{-1}(e)$): To achieve a textual representation of an E-R class model, the following grammar is defined.

To define a class, a class name is needed, a set of relations and a set of properties of the desired class. Note that this class name is a noun (this fact will later be useful when mixing the representations).

The grammar shown in Fig. 3. can be used to generate a class diagram including grade of relations and aggregations. In this work it is used to generate a domain class model. An example diagram is shown in Fig. 4 and its respective restricted natural language representation its shown in Fig. 5 :

Note that the reference to ”Phones” entity is made in the user as a set of these. The ”a set of” token allows to define many-to-many and one-to-many relations. The one to one relations are made with of the field rule, if the matched word is another domain class then its mapped as a relation.

Fig. 3. Grammar for E-R diagrams

```
id ->  FID (FID)*;
entity -> (’a’|’an’) id ’has’ ’:’ proplist;
proplist -> prop (((’,’) prop)* ’and’ prop)?;
prop -> (’a’|’an’) FID FID? | (’a set of’) FID;
FID -> [A-Z][a-zA-Z0-9]*
```

Fig. 4. Example of E-R diagram

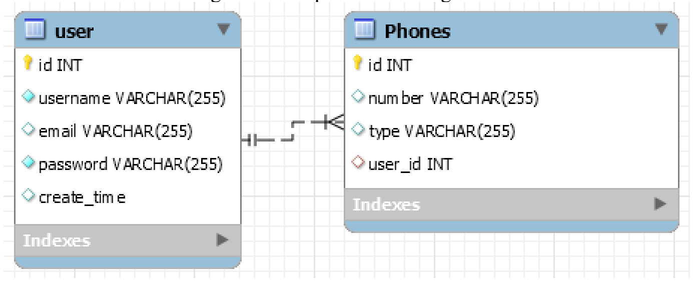


Fig. 5. Example of Restricted specification for E-R diagrams

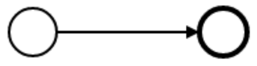
an User has: a Username, a Email, a Password, a Creation Date, and a set of Phones

a Phone has: a Number, a Type Value

2) From BPMN to Restricted Nl ($g^{-1}(b)$): Now, for a textual representation of BPMN with a restriction top only some of the BPMN constructs:

a) Start Event, End Event: the BPMN start and end events have the representation shown in Fig. 6.

Fig. 6. Example of BPMN start and end events



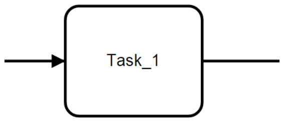
This couple of events define the start and end of the process, no matter where the end event appears, process intermediately finishes. In order to achieve a textual representation the grammar shown in Fig. 7 is defined:

Fig. 7. Grammar fragment of BPMN for start and end events

```
tasklist: task ((’,’ task)* ’and’ task)? ’.’;
process: ’the’ id ’process stars, then’ ’:’  tasklist;
task: ’the process ends’| ...;
```

b) Tasks: task in BPMN are the minimum and indivisible work unity, they represent some action to be performed in the organization environment. In BPMN the tasks have the representation shown in Fig. 8:

Fig. 8. Example of BPMN task



Each task has a value, this value is a verb phrase [30] denoting some action to be realized inside that task, the out-coming flow of the task can land in a gateway or can go back to previous task. To consider this two cases the grammar in Fig. 9. (simplified) is defined.

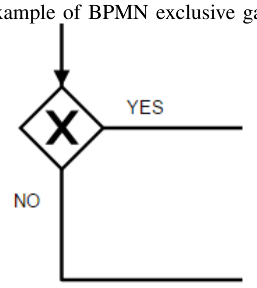
Fig. 9. Grammar fragment of BPMN for tasks

```
task:  ... |id
|'go back to' id ...
```

Note that the task is simply the contents of the original square in this new representation.

c) Exclusive Gateways: This gateways execute only the path that matches certain condition a diagram is shown in Fig. 10.

Fig. 10. Example of BPMN exclusive gateway



Taking into consideration that the outcoming flows can also land on a previously defined task the same "go back to" rule applies. The grammar in Fig. 11. can achieve that result.

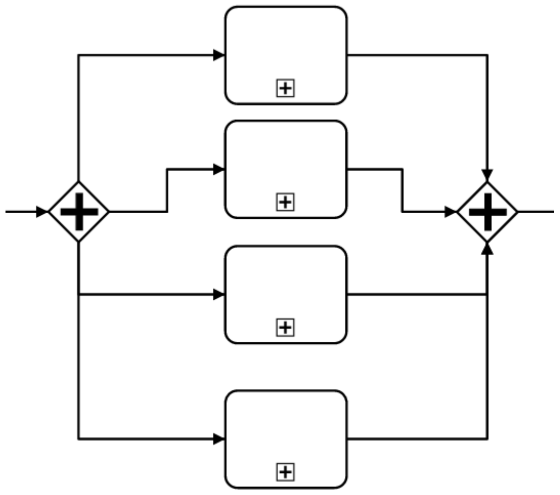
Fig. 11. Grammar fragment of BPMN for exclusive gateways

```
if_task: 'ask' 'if' id
control : 'if the answer to' id 'is' (FID|NUMBER) 'then'
            ':' tasklist
```

Note that only two alternative flows are possible in this diagram mimicking the "if" statement present in common programming languages.

d) Parallel Gateways: This gateways execute all the possible paths at the same time and waits for all to conclude. then executes the out-coming flow of the gateway an example is shown in Fig. 12.

Fig. 12. Example of parallel gateways



The "go back to" rule sill applies to the outcoming flow. The grammar fragment in Fig. 13 can achieve these results.

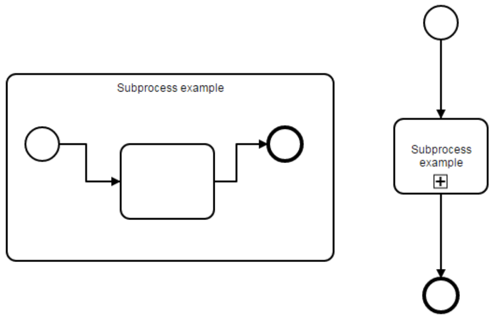
Fig. 13. Grammar fragment of BPMN for parallel gateways

```
task: 'do at the same time the' id 'tasks'
control :'for' id 'tasks' ('also')? 'do'':' tasklist;
```

Note that n flows can start an the divergent gateway. Also this grammar fragments grant the ability of allowing specifications to take more than one paragraph.

e) Subprocess: this type of action does a subprocess call, this subprocess is also a process thus some structures can be reused.

Fig. 14. Example of subprocess



This subprocess have two representations: the collapsed and expanded, the last one shows the detail of activities involved in the subprocess this is shown in Fig. 14.

To achieve the restricted representation, subprocess is defined outside the task list leaving only its name in the main representation. The grammar fragment is shown in Fig. 15.

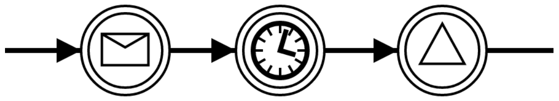
Also, giving the ability of specifying the subprocess in a separated paragraph.

Fig. 15. Grammar fragment of BPMN for subprocesses

```
task: ... |'the' id 'is' 'made' ...
subprocess: 'the' id 'subprocess stars,' 'then'
      ':'  tasklist;
```

f) Events: there are three places where the events can be placed: At the process start, in the process itself, and in the process end. For this work three intermediate catch events are implemented leaving the rest (boundary interrupting and non-interrupting) out of this work. An example is shown in Fig. 16.

Fig. 16. Example of events



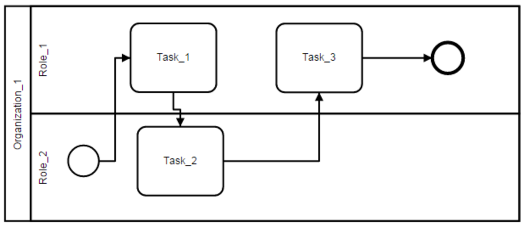
Obtaining as a result the grammar shown in Fig. 17. Note that for each type of event a grammar fragment is defined. Adding more types of events to this representation implies more grammar fragments.

Fig. 17. Grammar fragment for events

```
task: ... |'wait for the' id 'signal'
|'wait for the' id 'message'
|'wait' NUMBER ('week'|'weeks')
```

g) Lanes: objects representing an entity to execute the task (an organization role, another system). An example is shown in Fig. 18.

Fig. 18. Example of Lanes



The grammar to process the restricted representation is shown in Fig. 19. The definition of role for each task is not necessary. By default all task belong to a "default" lane and it changes when specified. All of the following task use the same lane.

3) Mixing resultant representations: This two representations can be mixed using the fact that class name is a noun and BPMN tasks are represented by verb phrases.

Fig. 19. Grammar fragment for Lanes

```
asignedtask: task ('(by ' id ')')?;
```

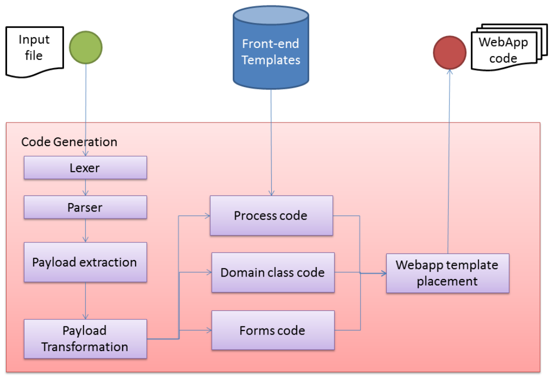
Verb phrases for BPMN task are of the type "action + object". For this paper, object is an instance of some domain class defined in the representation.An "action" is some operation done in this domain class [30].

In this way both representations combine to overcome limitations of each specification language having to code each verb.

B. Generating code

In order to generate the target web application from the resultant specification the following approach is proposed: First a lexical and syntactical analysis is performed. Next a design information extraction is made. Then a alteration over this extracted information. Finally the code is generated in four parts: code for domain classes, code for web forms, code for process flow, and place all the generated codes in a web application template. A diagram is shown in Fig. 20.

Fig. 20. Code generation process



1) Payload extraction: In order to extract the necessary information a parse tree is obtained first, this can be done using a traditional $LL(*)$ parser, because the representations are not ambiguous.

After this, the information belonging to each class is extracted from the parse tree. The class name and attributes for the domain classes definition part and for the processes, tasks names, order of execution, gateways and subprocesses.

Also, for complementary definitions such as "control" and "subprocess" this process is made and left referenced in memory preparing to further steps.

2) Payload Alteration: In this step two parts are identified: altering the attributes of each domain class to match expected relationships, and inject "control" and "subprocess" into the process where they are called.

To match expected relationships we check for some cases:

1) An attribute has a known data type.

2) An attribute references another class in the specification.
3) One class has a ”a set of” attribute.
4) One class has a ”a set of” attribute and the other class have a back reference.

If the case is 1) we simply continue, if the case is 2) we assign the data type of the attribute to a foreign key referencing the class. For case 3) we understand this as a one-to-many relation, thus we alter the referenced class inserting a foreign key attribute pointing to the class that have a set of the other. Finally for 4) case is interpreted as a many-to-many relation, thus we insert an additional class referencing the two involved classes.

After control and subprocesses parsing, an injection of the resultant payloads in its referencing process is performed. The parse tree of the process is walked again for the ”tasklist” non-terminal, when a reference is made to a subprocess o control structure a search is performed to get the information to inject in place. That is the alternative paths and subprocesses. This process is made recursively to allow referencing between paragraphs.

3) Web application code generation: Starting from the altered payload we generate a class definition with its defined parameters and references, including the foreign keys and many-to-many relations. These classes can be used later with an ORM (object relational mapper) to generate database code.

Next using the same scheme a form code is generated in order to process request from the web browser. Taking care of the foreign key references to be rendered as the result of some query involving the referenced class.

Then the code for process flow is generated, this can be viewed as the conditional redirect between one view and the next. In order to generate the resultant code, for each design element some code is predefined in a template system. As example the code for a exclusive gateway can be a method where some HTML is rendered asking a yes/no question. Then after a POST request to some controller, determine the task to be done next and make an HTTP redirect. Here a template must be defined for each verb in order to render that controller and HTML.

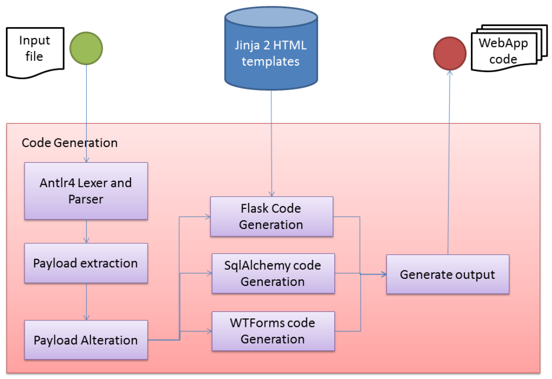
Finally these three codes are placed in a template prepared to execute the web application as its desired.

IV. Implementation

To generate code automatically an $all(*)$ parser is generated using antlr4[1] which is a parser generator. Using this we are capable of obtaining the design information. Finally doing the alteration and translation into our target language and platform: python 2.7 using modules like Flask[2] (a web framework), SqlAlchemy[3] (an ORM system) and WTforms[4] (a HTML forms processor).

Tree different translations are made: a translation to domain models in SqlAlchemy, a translation to WTforms, and finally a translation of the resultant language of BPMN to a Flask application involving the operations over the domain classes defined for each task. a diagram is shown in Fig. 21.

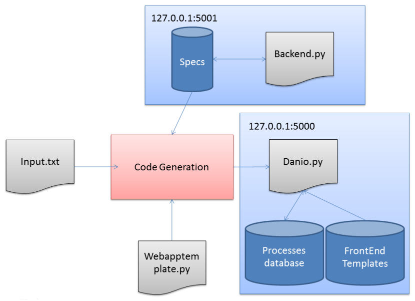
Fig. 21. Implementation



The template system used is Jinja2[5], for storing the templates a small database in sqlite3[6] is used. a small web server is used to edit and create the templates.

The resultant code is scripted to run on port 5000 and uses some templates to generate the front end. a diagram showing the two services and the process is shown in Fig. 22.

Fig. 22. Services and process



V. Case Study

To test this implementation a example process ”Question Cycle” is implemented.In this process a set of question, an asignature, and grade are defined, selected and asked to the same role. The diagram for this process is shown in Fig. 23. and its respective representation in restricted language is shown in Fig. 24 .

The final result is a web application executing processes Fig. 26 shows the set of views in mobile format in order to execute the business process.

Most of the work done for this prototype is in writing two code templates: the front end template and the back end template. This because the code templates mus be defined for the verbs of ”Ask” in this context. Moreover, the same code template is used for all the ”select” views taking into account if the noun is in plural or singular. All the Yes/No

[1]http://www.antlr.org/
[2]http://flask.pocoo.org/
[3]http://www.sqlalchemy.org/
[4]https://wtforms.readthedocs.io/en/latest/
[5]http://jinja.pocoo.org/docs/dev/
[6]https://www.sqlite.org/

Fig. 23. Question Cycle

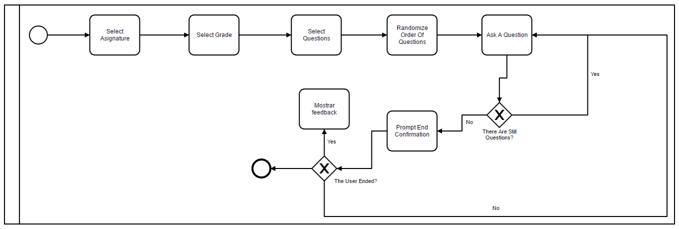


Fig. 24. Question Cycle in restricted Natural language

an Asignature has: a Name, a Code Value and a set of Questions
a Question has: a Dba, a set of Answers, a Grade Number and a Heading Text
an Answer has: a Text Value
a Dba has: a Code Value and a Name Value
a Grade has: a Course Number
the Question Cycle process stars, then: Select Asignature, Select Grade, Select Questions, Randomize Order Of Questions, Ask A Question and, ask if There Are Still Questions.
if the answer to There Are Still Questions is Yes then: go back to Ask A Question. if the answer to There Are Still Questions is No then: Prompt End Confirmation and, ask if The User Ended.
if the answer to The User Ended is Yes then: the process ends. if the answer to The User Ended is No then: go back to Do Question.

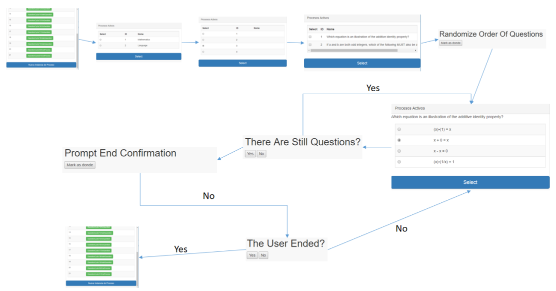
question can be replaced by some source code in the rendered controller. Some limitations of this prototype must be shown: first, the questions mus be filtered by grade and asignature, this is not done. Also some of the yes/no views can be replaced by automatic actions.

VI. Conclusions

The approach implemented in this work allows to make a restricted natural language using as starting point some specification languages, and finally generate code from this mix of restricted natural language representations. this transformation based on previous specification languages gains that the pertinence of the original languages persist in the new representation. in this work is shown a synergy between two languages after a transformation generating an executable web service but with the downside of having to relate each verb to a source codes set.

To further works: a sub grammar for defining tasks, Changing the languages to those that conform UML, or find patterns in task to generate more generic types of task with a mechanism to heritage from those generic tasks.

Fig. 25. Process generated views



References

[1] A. M. Davis, *Software requirements: analysis and specification*. Englewood Cliffs, N.J: Prentice Hall, 1990.

[2] G. S. Walia and J. C. Carver, “A systematic literature review to identify and classify software requirement errors,” pp. 1087–1109, 2009.

[3] R. E. Fairley, *Managing and leading software projects*. Los Alamitos, CA : Hoboken, N.J: IEEE Computer Society ; Wiley, 2009.

[4] J. Bhatia, R. Sharma, K. K. Biswas, and S. Ghaisas, “Using Grammatical knowledge patterns for structuring requirements specifications,” *2013 3rd International Workshop on Requirements Patterns, RePa 2013 - Proceedings*, pp. 31–34, 2013.

[5] F. Nishida, S. Takamatsu, Y. Fujita, and T. Tani, “Semi-automatic program construction from specifications using library modules,” *IEEE Transactions on Software Engineering*, vol. 17, no. 9, pp. 853–871, 1991.

[6] M. Ibrahim and R. Ahmad, “Class diagram extraction from textual requirements using natural language processing (NLP) techniques,” *2nd International Conference on Computer Research and Development, ICCRD 2010*, pp. 200–204, 2010.

[7] P. Kroll and P. Kruchten, *Rational Unified Process Made Easy: A Practitioner’s Guide to the RUP*, 2003.

[8] M. Fowler and J. Highsmith, “The agile manifesto,” *Software Development*, vol. 9, pp. 28–35, 2001. [Online]. Available: http://www.pmp-projects.org/Agile-Manifesto.pdf\nhttp://andrey.hristov.com/fht-stuttgart/The_Agile_Manifesto_SDMagazine.pdf\nhttp://www.pmp-projects.org/Agile-Manifesto.pdf

[9] S. Augustine and R. C. Martin, *Managing agile projects*, ser. Robert {C}. {Martin} series. Upper Saddle River, NJ: Prentice Hall Professional Technical Reference, 2005.

[10] G. Githens, “Managing Agile Projects by Sanjiv Augustine.” *Journal of Product Innovation Management*, vol. 23, no. 5, pp. 469–470, 2006. [Online]. Available: 10.1111/j.1540-5885.2006.00217_3.x\nhttp://search.ebscohost.com/login.aspx?direct=true&db=bth&A
live

[11] I. G. Harris, “Extracting design information from natural language specifications,” *Proceedings of the 49th Annual Design Automation Conference on - DAC ’12*, p. 1256, 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2228360.2228591

[12] R. J. Abbott, “Program design by informal English descriptions,” *Communications of the ACM*, vol. 26, no. 11, pp. 882–894, 1983.

[13] C. M. Z. J, “UN Lencep : Obtención Automática de Diagramas UML a partir de un Lenguaje Controlado,” *Memorias del VII Encuentro Nacional de Computación ENC’06*, pp. 254–259, 2006.

[14] R. Schwitter, “Attempto-from specifications in controlled natural language towards executable specifications,” *Arxiv preprint cmp-lg/9603004*, 1996. [Online]. Available: http://arxiv.org/abs/cmp-lg/9603004

[15] J. R. Bellegarda and C. Monz, “State of the art in statistical methods for language and speech processing,” *Computer Speech & Language*, vol. 35, pp. 163–184, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0885230815000662

[16] N. Zhou, X; Zhou, “Auto-generation of Class Diagram from Free-text Functional Specifications and Domain Ontology,” no. 2, pp. 1–20, 2008. [Online]. Available: http://www.daviszhou.net/research/INFO626Prj.pdf

[17] M. Selway, G. Grossmann, W. Mayer, and M. Stumptner, “Formalising natural language specifications using a cognitive linguistic/configuration

based approach," *Information Systems*, vol. 54, pp. 191–208, 2015. [Online]. Available: http://dx.doi.org/10.1016/j.is.2015.04.003

[18] J. J. Granacki and a. C. Parker, "PHRAN-SPAN: a natural language interface for system specifications," *24th ACM/IEEE conference proceedings on Design automation conference - DAC '87*, pp. 416–422, 1987. [Online]. Available: http://portal.acm.org/citation.cfm?doid=37888.37950

[19] B. R. Bryant and B. S. Lee, "Two-level grammar as an object-oriented requirements specification language," *Proceedings of the Annual Hawaii International Conference on System Sciences*, vol. 2002-Janua, no. c, pp. 3627–3636, 2002.

[20] M. Saeki, H. Horai, and H. Enomoto, "Software Development Process from Natural Language Specification," *Proceedings of the 11th International Conference on Software Engineering*, pp. 64—-73, 1989. [Online]. Available: http://dl.acm.org/citation.cfm?id=74594

[21] D. Popescu, S. Rugaber, N. Medvidovic, and D. M. Berry, "Reducing ambiguities in requirements specifications via automatically created object-oriented models," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5320 LNCS, pp. 103–124, 2008.

[22] R. Smith, G. Avrunin, and L. Clarke, "From natural language requirements to rigorous property specifications," *Workshop on Software Engineering for Embedded Systems (SEES 2003) From Requirements to Implementation*, pp. 40–46, 2003. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.64.500&rep=rep1&type=pdf

[23] S. Konrad and B. H. C. Cheng, "Facilitating the construction of specification pattern-based properties," *13th IEEE International Conference on Requirements Engineering RE05*, no. August, pp. 329–338, 2005. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1531053

[24] D. Ilić, "Deriving formal specifications from informal requirements," *Proceedings - International Computer Software and Applications Conference*, vol. 1, no. Compsac, pp. 145–152, 2007.

[25] A. Zeaaraoui, Z. Bougroun, M. G. Belkasmi, and T. Bouchentouf, "User stories template for object-oriented applications," *2013 3rd International Conference on Innovative Computing Technology, INTECH 2013*, pp. 407–410, 2013.

[26] W. Dahhane, A. Zeaaraoui, E. H. Ettifouri, and T. Bouchentouf, "An automated object-based approach to transforming requirements to class diagrams," *2014 2nd World Conference on Complex Systems, WCCS 2014*, pp. 158–163, 2015.

[27] C. Videira, D. Ferreira, and A. R. Da Silva, "A linguistic patterns approach for requirements specification," *Proceedings - 32nd Euromicro Conference on Software Engineering and Advanced Applications, SEAA*, vol. 2004, pp. 302–309, 2006.

[28] F. Friedrich, J. Mendling, and F. Puhlmann, "Process Model Generation from Natural Language Text," *Lecture Notes in Computer Science*, vol. 6741, pp. 482–496, 2011.

[29] S. Geetha and G. Mala, "Extraction of key attributes from natural language requirements specification text," in *IET Chennai Fourth International Conference on Sustainable Energy and Intelligent Systems (SEISCON 2013)*. Institution of Engineering and Technology, 2013, pp. 374–379. [Online]. Available: http://digital-library.theiet.org/content/conferences/10.1049/ic.2013.0341

[30] M. Chinosi and A. Trombetta, "BPMN: An introduction to the standard," *Computer Standards and Interfaces*, vol. 34, no. 1, pp. 124–134, 2012.