Flexible Modelling for Requirements Engineering

Athanasios Zolotas[1], Nicholas Matragkas[2],
Dimitrios S. Kolovos[1], and Richard F. Paige[1]

[1] Department of Computer Science, University of York, York, UK
[2] Department of Computer Science, University of Hull, Hull, UK
Email: {amz502, dimitris.kolovos, richard.paige}@york.ac.uk,
n.matragkas@hull.ac.uk

Abstract. Many applications that are developed do not completely fulfil the requirements of their stakeholders. This can be a result of inadequate requirements elicitation and poorly defined requirements. Many solutions, including model-driven inspired ones, have been proposed to improve the elicitation of the requirements, though many of them are not yet widely used in practice as they require training of both the employees and the stakeholders. In this paper we propose the use of flexible modelling for eliciting and capturing the requirements of applications to facilitate the production of correct products that deliver on the *contract* defined between clients and developers. We argue that the use of flexible modelling can lower the entry barrier for use in the industry. The proposed method, called FlexRE, is applied to a scenario to demonstrate its capabilities and ways it can be extended.

1 Introduction

Tendered contracts [1] are very often used for the 'first contact' between the clients and the companies which develop applications. An initial, unstructured set of needs is given in the form of a tendered contract to candidate companies. Developers bid for the project by proposing an estimation on cost and time needed for the realisation of the project. The clients pick the solution that appears to be the most appropriate. After the agreement, business analysts, following different requirements elicitation techniques (e.g. interviews, prototypes, etc.) add more details to the requirements to let the developers have a better understanding on the needs of the clients. Many projects that follow this process fail to match the real client needs. Different studies [2], [3], [4] show that one of the reasons for that problem is the ambiguity of the requirements.

Many techniques have been developed to tackle this problem, including those that are model-driven inspired. The Agile methodologies, were proposed to increase the active participation of the client in the software development process, thus in the requirements elicitation phase. The above research surveys suggest that the problem still exists.

The proposed solutions so far urge the composition of either a very structured artefact which should conform to a rigorously-defined metamodel or a totally unstructured composition of requirements using text. The former are difficult to be used by non-technical stakeholders, as they force them to follow the semantics that are bound to a fixed metamodel. Thus there is low clients' contribution to the requirements specification document and high entry barrier of the methodologies in real world. In contrast, the latter offer no or limited structure, thus one cannot benefit from the use of model management techniques.

This paper presents a novel approach in expressing tendered contracts that promotes the active involvement of the clients in the composition of the document having as primary goal to lower the entry barrier and reduce the cost of using Model-Driven Engineering (MDE) in the requirements elicitation phase. This approach, offers a varying range of structure, positioning itself between the unstructured and rigidly structured requirements methodologies. It is based on the GraphML [5] standard and the flexible modelling technique introduced in [6]. Due to specific interest from our industrial partner in Web applications, we applied and tested the approach in this domain. There were no indications that it could not be applied to other domains, like desktop or mobile applications.

2 Background and Motivation

Clients are usually domain experts; they generally can understand the terminology and the processes that take place in the domain. By contrast, business analysts, modellers and developers are technology experts and know how systems are built. The lack of knowledge of the domain by technology experts and the lack of knowledge of the technology by domain experts is termed as the *symmetry of ignorance*. [7] "Contributory methods" like prototypes and scenarios, were added to the development processes to increase the active client's involvement and cooperation with the system's modellers. [7].

In WebML [8], probably the current standard in modelling web applications and their workflows, non-technical stakeholders should use elements from a palette that they are not familiar with as these elements were defined to represent technical concepts, like edges that represent successful and unsuccessful messages between actions or nodes that add/remove entries in ER Data diagrams.

A second problem of approaches that are based on rigorous and pre-defined metamodels is the fact that they restrict users of expressing requirements that the author of the metamodel did not think about. Thus, the scope of the projects that they can model is reduced (e.g. support for data-driven web applications only) or clients are restricted to use specific concepts that do not represent their envisioned applications. This boils down to the following anecdotal quote:

> "I've been in situations where I found that the modeling tool was simply too structured to let me describe everything I needed to describe."[3]

[3] http://programmers.stackexchange.com/questions/55679/why-arent-we-all-doing-model-driven-development-yet

A survey conducted among 12 Business Analysts working with IBM [9] verifies this argument and summarises the advantages and disadvantages of each of these two types of requirements elicitation approaches.

In this paper we argue for an approach that can increase the domain experts' contribution to the specification of the requirements. This approach is based on the use of a simple drawing tool and thus it requires less training given that one can build on familiar idioms and mechanisms reducing the cost of adaptation in the real world. It offers a varying range of structure, positioning itself between the unstructured and rigidly structured requirement methodologies promoting the use of Agile principles and processes.

3 Related Work

The first family of approaches includes those which use natural language as the means of expressing requirements. One approach is the use of spreadsheets or text documents for expressing the requirements. Controlled Natural Language (CNL) techniques aim to tackle the ambiguity of such statements written in natural language. In the same direction, Kaindl et al. [10] proposed the ReDSeeDS requirements language which uses dictionary definitions attached to the words and linking the words that refer to the same entity. These approaches can be used by non-technical stakeholders with no specific training increasing their contribution in the final definition of the system's requirements. However, they cannot benefit from the MDE advantages as they are not structured.

Two other approaches are the Business Process Modelling Language (BPMN)[4] and the Unified Modelling Language (UML)[5]. They allow business analysts to describe workflows of an application. The Web Requirements Engineering (WebRE) [11] is a UML profile specifically built to help with the specification of web applications. These diagrams are not usually used as the only way to describe the system but as supporting to the Software Requirements Specification, artefacts. The Navigational Developemnt Technique (NDT) [12] uses a template to store requirements. The Web Requirements Metamodel (WRM) [13] supports the definition of requirements using either UML Use Case Scenarios or NDT templates. All these approaches offer some structure, like links between the stakeholders and their needs, but not structure that is related with the elements that are used in fulfilling the requirements (e.g. images, buttons, menus, lists, etc.). In addition, training is needed to be used by non-technical stakeholders as they require conformance to specific concrete and abstract syntax.

The Program Design Language (PDL) [14] was the first attempt in the direction of using graphical interfaces to represent requirements, followed by other approaches, like the Structured Analysis and Design Technique (SADT) [15]. These techniques promote a very structured way in storing the requirements and can be seen as a design of the system under development rather than a SRS document thus can only be used by specialists. [16]

[4] http://www.omg.org/spec/BPMN/
[5] http://www.omg.org/spec/UML/

The closest to our approach is the one proposed in [9]. The authors, as mentioned in the previous section, surveyed 12 Business Analysts cooperating with IBM. As an outcome they proposed the use of a flexible modelling tools in the pre-requirements phase highlighting the advantages that such an approach has when domain experts and non-technical stakeholders are involved. In this paper, we propose the use of a flexible modelling approach not only in the pre-requirements but through the whole requirements phase that can also be used as a starting artefact for the other phases (e.g. coding, testing).

4 Flexible Modelling for RE

Our approach is based on the technical concepts of an approach called *Muddles*, proposed in [6], which employs flexible modelling techniques and promotes the use of a general-purpose drawing tool for the creation of programmatically manageable models. Flexible modelling allows the creation of models that are not instances of a specific metamodel. Models are created as an example to help the production of a rigorous metamodel that can then support MDE processes and model management using automated tools. Such a process allows domain experts to be actively involved in the creation of the metamodel by providing example models that describe their envisioned metamodel [6].

4.1 The Muddles Approach

The *Muddles* approach [6] uses a flexible graph definition language, GraphML [5], to allow language engineers to draw models and then annotate them so they can be accessed by model management suites. In the *Muddles* approach, each GraphML *Node* that is created in the drawing is extended with four Data fields attached to it (Fig. 1). The *Type*, is the field where the developer declares the type of each node. The *Properties*, is the field where the developer can add attributes to the node. For example, all the nodes of type "Web Page" have a String property called "Title". The *Default* field defines the variable which will be used by model management suites to access the label of the node. Finally, the *Contents* field defines the descriptor that will be used to get all the nodes contained within a parent node.

Each GraphML *Edge* is also extended by the Type, Properties and Default fields. In addition, fields to hold the descriptors of the "Source", "Target", the "Role in source", the "Role in target" and their multiplicities are defined. For example, an edge that represents linking between two nodes, has an outgoing relation named "linksTo" and an incoming relation called "linkedBy" with a multiplicity of 1. The "sourcePage" and "targetPage" define the keywords for the source and the taget page of the link, respectively. These keywords can be used by a model management suite to have access to the source or target element.

After the diagrams are annotated they are automatically transformed to an intermediate model which is instance of the *Muddle* metamodel. A discussion on the steps of the transformation are beyond the scope of this paper. The Epsilon

Data	
Type	Image
Properties	String src="..."
Default	name
Contents	children

(a) The Node Properties

Data			
Type	Link	Properties	String URL = ".."
Source	source	Role in sourcePage	linksTo1
Target	target	Role in targetPage	linkedBy1

(b) The Edge Properties

Fig. 1. The element properties

platform [17] they are using offers a driver that can consume *muddle* models and allow the execution of model management programs on them.

4.2 FlexRE

In this work we propose the use of a drawing tool that implements the GraphML standard to let domain experts in collaboration with business analysts, express the requirements of applications. By using a drawing tool and flexible modelling throughout the requirements engineering phase, we argue that the drawbacks (see Section 2) of current MDE techniques could be tackled. In addition, the drawings that include the requirements are structured and can be consumed by model management tools. We demonstrate this approach through a running example. In the example, we use the yEd[6] editor, the Epsilon platform [17] for model management and the technical concepts of the Muddles approach proposed in [6]. An overview of FlexRE is presented in Figure 2.

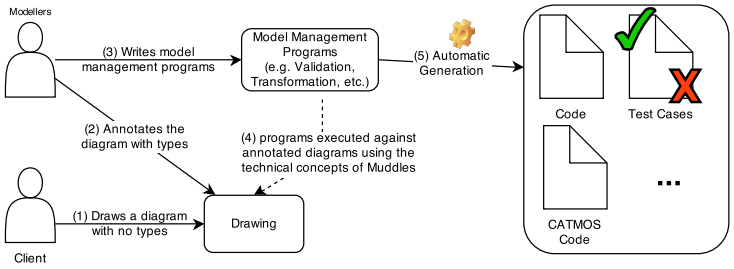


Fig. 2. An overview of FlexRE

In this case study, the client is interested in having a web application that will be used to present their Hotel company. It includes static and dynamic pages to present the rooms, the restaurant and help customers check for the rooms availability. A comprehensive set of the requirements are summarised in Table 1.

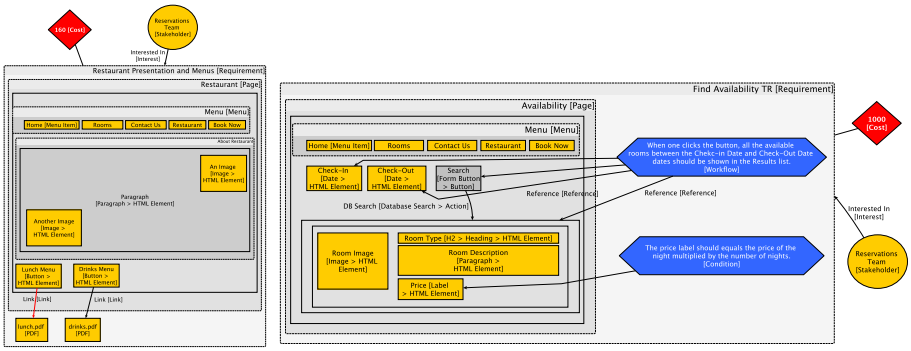
The *Manager*, who is the client in this scenario, starts drawing using the yEd editor, concepts that they are familiar with like check-in dates, check-out dates, etc. (step 1). He does not need to follow any specific rules and can use any shape or image from the palette of the drawing tool that represents their understanding of the required functionality. Business analysts can in parallel annotate the drawings with types that can be useful for the requirement methodologies they follow, using the fields presented in Fig. 1 (step 2). As an example, assume

[6] `http://www.yworks.com/en/products_yed_about.html`

Table 1. List of the requirements

ID	Stakeholder(s)	Description
FR1	Manager	"A page to present the rooms of the Hotel"
FR2	Manager	"A home page that will have a short introduction of the hotel, the rooms and the staff."
FR3	Manager, Customer Services	"A contact us page."
FR4	Manager, Reservations Team	"A page that will present all the available rooms for a specific range of dates."
FR5	Reservations Team	"A page for presenting the restaurant."

that, for the "FR5" requirement, the client ends up with the drawing shown in Fig. 3(a). The business analyst starts annotating these concepts. In the interest of a more clear presentation, the business analyst annotations for each element are shown in square brackets. In reality, the types are given using the Properties window of each element (Fig. 1). In this example, all the distinct HTML elements that the client uses are annotated as subclasses of the "HTML Element" class that this company uses as to identify them in the development process. This is done by using the ">" symbol which is used to denote the extension relationship. For instance, the Type property of the drawing that the client referred to as an image is set to "Image > HTML Element". These names given to the types are conventions, used to let the elements be accessed by a model management suite. It could be anything that fits the model-driven processes used in each company.



(a) The FR5 requirement drawing

(b) The FR4 requirement drawing

Fig. 3. Two requirements

Non-functional requirements, business behaviour and workflows that can't be described using drawings can be typed in natural language. The business analysts can then either translate this into a drawing based on their experience or can simply attach bits of these requirements written in natural language to parts of the drawing that it is related to. For example, assume that the client needs to express the following requirement: "When the *button* is clicked, all the available rooms between the *check-in* and *check-out* dates should be shown in the *results list*.". This is part of the FR4 requirement shown in Fig. 3(b). The

workflow that cannot be drawn is expressed using plain text inside the hexagons (see Fig. 3(b)) and can then be linked with the button that the client talks about, the check-in/out input fields and the results list that the client refers to.

The client continues drawing his understanding of the desired web application. As soon as the client has finished developers can then prepare model management programs or re-use those they have written in the past (step 3). These programs can be executed against the annotated drawings (step 4) using the technical facilities of the *Muddles* approach, to produce different artefacts that are interested in like pieces of code, test cases, textual contracts, and many others (step 5). Such examples are presented in the Section 5.

The same process could, in principle, be supported with a more traditional editor based on a fixed metamodel (e.g., derived using EMF/GMF). An advantage of using a flexible model is that it doesn't restrict clients in expressing concepts that the developers hadn't considered about when creating the metamodel. In other words, it allows clients to directly formulate the language of discourse without being restricted by a predefined set of constructs produced by non-experts. In addition, the drawing editor is a tool that even non-technical stakeholder arguably are able to use with no extensive training.

A first disadvantage of using flexible modelling is that it is error-prone. For instance, typos in the definition of the type of an element will create a new type. Secondly, there is a possibility of leaving untyped nodes which will left out from the model management processes. The above risks can be tackled by using validation rules. For instance, one could write a re-usable post-processor that calculates distance/similarity metrics for the Types. If the value for two Types is above a specific threshold it alerts the modellers for possible errors. Work in this direction is being carried out.

5 Application Scenarios

As discussed in the previous section, the annotated drawing can be consumed by programs expressed in model management languages of the Epsilon platform [6]. In this section we present some examples on how the drawings can be used to support different phases and aspects of the software development lifecycle.

5.1 Validation

It would be of interest to check if important rules/constraints of the domain are being met. For example, in the web domain, all the "Link" edges must have exactly one source and one target element. Conformance to such rules can be achieved by applying model validation to the drawing. In our scenario we use the Epsilon Validation Language (EVL), to validate that the above constraint (see Listing 1.1). The "source" and "target" keywords used in the code are the names the modellers provided in the *Source* and *Target* elements of the "Link" properties (see Fig. 1). Similarly, the keyword "linksTo" is the identifier of the *"Role in source"* property (see Fig. 1). The EVL statements, can be run against

the drawings, are written once and can be re-used for all the projects.

```
context Link {
  constraint SourceAndTargetExist {
    guard : self.isTypeOf(Link)
    check : (self.linksTo.sourcePage.all.size == 1) and (self.linksTo.targetPage.all.
        size >= 1)
    message : "Link edges should have exactly one source node and at least one
        target node"
  }
}
```

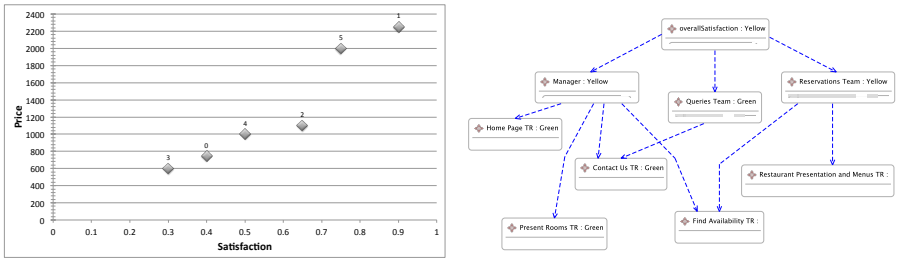
Listing 1.1. Example of EVL rule for validating the Link edges.

5.2 Next Release Problem

The Next Release Problem (NRP) is a multi-objective optimisation problem. The best set of requirements that should be implemented into the next software release, staying within the budget, is calculated. Burton et al. [18] offer a tool, as part of the Capability Acquisition Technique with Multi-Objective Search (CATMOS) methodology, that uses genetic algorithms and Pareto fronts to identify the optimal solutions calculating the possible trade-offs between the requirements of different stakeholders. The optimal solutions are visualised based on the total cost and total satisfaction they offer. Details on how the CATMOS suite works to find the optimal solutions are beyond the scope of this paper.

In this section we demonstrate the automatic transformation of the stored requirements into code that can be consumed by the CATMOS capability acquisition suite in [18] to solve the Next Release Problem.

Based on the scenario presented above, the developers just need to add and annotate two new types on the drawing. The first one is of type "Cost" (see Fig. 3) and defines the estimated cost to implement each component. The second is a "Dependency" edge and shows the dependency between two or more requirements. The drawing is now ready to be consumed by a model-to-text transformation expressed in the Epsilon Generation Language (EGL) and automatically generate the code needed to run the CATMOS tool. The EGL script is written once and can be then re-used.



(a) The optimal solutions (b) The solution #2

Fig. 4. The results of the CATMOS suite

The results of running the CATMOS are shown in Fig. 4. Fig. 4(a) shows all the possible optimal solutions for different budgets. The horizontal axis presents the total satisfaction that each solution offers, while the vertical represents, the cost of implementing each solution. Based on the budget, clients can pick the most appropriate solution knowing that this set of requirements is the optimal

for the amount of money they want to spent. An example is given in Fig. 4(b). In this solution, the client can have the FR2 (Home Page), FR3 (Contact Us) and FR5 (Restaurant Presentation) requirements developed, which is the optimal set of requirements for a budget of 1100 units of money.

5.3 Code and Test Cases Generation

Code and test cases can be generated from the drawings. In this scenario, we use EGL to generate code for the structural and navigational parts of the described application. Code that implements the behavioural requirements (workflows) is not generated, yet. Plans for this process are presented in Section 6. In addition to that, we generate Selenium-Webdriver[7] test cases directly from the specification for the structural and navigational aspect of the application. The test cases can be used as primary artefacts for a Test-Driven Development approach where the developers create the structure and navigation from scratch.

6 Conclusions and Future Work

We presented a method that can be used to elicit and store requirements of applications. The method allows non-technical stakeholders to 'draw' the requirements with no previous training offering a low entry barrier for industrial use. It offers a flexible structure level to the requirements document (from completely un-structured to highly structured) without restricting the types of requirements that can be expressed. Finally, it can be used as a starting point for a number of MDE methodologies like code and test cases generation suites. It is highlighted though, that this approach could be beneficial in domains where the graphical representation of requirements adds value. In contrast, in domains where textual representations are more appropriate, the drawing overhead of the approach should be taken into account.

In the future, we aim to connect using M2M transformations, the requirements drawn using FlexRE with other MDE suites like WebRatio, which can perform full code generation.

Acknowledgments

This work was carried out in cooperation with Digital Lightspeed Solutions Ltd, and was supported by the EPSRC through the LSCITS initiative and part supported by the EU, through the MONDO FP7 STREP project (#611125).

References

1. Domberger, S., Hall, C., Li, E.A.L.: The determinants of price and quality in competitively tendered contracts. The Economic Journal (1995) 1454–1470

[7] `http://docs.seleniumhq.org/projects/webdriver/`

2. Abbott, B.: Requirements set the mark. Infoworld (2001) 45–46
3. Epner, M.: Poor project management number-one problem of outsourced e-projects. Research Briefs, Cutter Consortium **7** (2000)
4. Lowe, D.: Web system requirements: an overview. Requirements Engineering **8**(2) (2003) 102–113
5. Brandes, U., Eiglsperger, M., Herman, I., Himsolt, M., Marshall, M.S.: Graphml progress report structural layer proposal. In: Graph Drawing, Springer (2002) 501–512
6. Kolovos, D.S., Matragkas, N., Rodríguez, H.H., Paige, R.F.: Programmatic muddle management. XM 2013–Extreme Modeling Workshop (2013) 2
7. Fernandes, K.J.: Interactive situation modelling in knowledge-intensive domains. International Journal of Business Information Systems **4**(1) (2009) 25–46
8. Ceri, S., Fraternali, P., Bongio, A.: Web Modeling Language (WebML): a modeling language for designing web sites. Computer Networks **33**(1) (2000) 137–157
9. Ossher, H., Bellamy, R., Simmonds, I., Amid, D., Anaby-Tavor, A., Callery, M., Desmond, M., de Vries, J., Fisher, A., Krasikov, S.: Flexible modeling tools for pre-requirements analysis: conceptual architecture and research challenges. ACM Sigplan Notices **45**(10) (2010) 848–864
10. Kaindl, H., Smialek, M., Svetinovic, D., Ambroziewicz, A., Bojarski, J., Nowakowski, W., Straszak, T., Schwarz, H., Bildhauer, D., Brogan, J., et al.: Requirements Specification Language Definition: Defining the ReDSeeDS languages. Institute of Computer Technology, Vienna University of Technology (2007)
11. Escalona, M., Koch, N.: Metamodeling the requirements of web systems. Web Information Systems and Technologies (2007) 267–280
12. Jose Escalona, M., Aragon, G.: NDT. a model-driven approach for web requirements. Software Engineering, IEEE Transactions on **34**(3) (may-june 2008) 377 –390
13. Molina, F., Pardillo, J., Toval, A.: Modelling web-based systems requirements using WRM. In Hartmann, S., Zhou, X., Kirchberg, M., eds.: Web Information Systems Engineering WISE 2008 Workshops. Volume 5176 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2008) 122–131
14. Caine, S.H., Gordon, E.K.: PDL: a tool for software design. In: Proceedings of the May 19-22, 1975, national computer conference and exposition. AFIPS '75, New York, NY, USA, ACM (1975) 271–276
15. Ross, D., Schoman, K.E., J.: Structured analysis for requirements definition. Software Engineering, IEEE Transactions on **SE-3**(1) (jan. 1977) 6 – 15
16. Sommerville, I.: Software Engineering. 6th Edition. Addison Wesley Publishing Company Inc, Essex, England (2001)
17. Paige, R.F., Kolovos, D.S., Rose, L.M., Drivalos, N., Polack, F.A.: The design of a conceptual framework and technical infrastructure for model management language engineering. In: Engineering of Complex Computer Systems, 2009 14th IEEE International Conference on, IEEE (2009) 162–171
18. Burton, F.R., Paige, R.F., Rose, L.M., Kolovos, D.S., Poulding, S., Smith, S.: Solving acquisition problems using model-driven engineering. In: Modelling Foundations and Applications. Springer (2012) 428–443