2009 International Conference on Information and Multimedia Technology

Extracting Software Functional Requirements from Free Text Documents

Yunhe Mu,Yinglin Wang, Jianmei Guo
1. Department of Computer Science and Engineering;
2. MOE-Microsoft Key Laboratory for Intelligent Computing and Intelligent Systems;
Shanghai Jiao Tong University
800 Dong Chuan Road, Shanghai 200240, China
e-mail: {mumaomao, *ylwang*, guojianmei } @sjtu.edu.cn

***Abstract*—The acquisition of requirements assets are important in software product line (SPL) engineering for it help enhancing the effectiveness of reuse. Traditional methods are heavily based on manual effort. This appears to be a barrier for many organizations which tend to launch a SPL. In this paper, we propose an approach to extract functional requirements by analyzing text-based software requirements specifications (SRSs). We analyze the linguistic characterizat-ion of SRSs. According to it we define extended functional requirements framework (EFRF) which consists of 10 semantic cases, then we generate converting rules. We introduce an NLP (Natural Language Process) approach to build EFRFs from documents based on the concept of EFRF and the converting rules. The extracted EFRFs are suitable for expression and modeling of functional requirements variability. We apply our method to an auto-marker software product line. The result shows the approach has high accuracy and efficiency, and the approach is readily scalable and extensible.**

Keywords-information extract; product line; functional requirement ;variability model; rule-based

I. INTRODUCTION

Software product line (SPL) engineering [1][2] is one of the success stories of software reuse whose purpose is to improve quality and productivity. More and more organizations tend to apply SPL engineering after they accumulate development experience in a domain and want to reuse these resources. Requirements assets enhances the effectiveness of reuse as developers can work on the abstractions closer to the SPL's initial concepts.

Mass manual effort will be associated with domain analysis of requirement specifications. It presents a barrier for many organizations that could otherwise benefit. So it urges us to develop an extracting method to help requirement analysis.

Some researchers focused their attentions on how to ease the transition from a single-system mentality to software mass customization.

Liaskos et al. [3] identified variability in goal models via Fillmore's case theory [4]. They took a closer look at the semantic characterization of every goal's OR-decompositions. They identified the background variability by three cases. They demonstrated the importance of OVM (Orthogonal variability model) modeling through different cases.

Nan Niu et al. introduced a semi-automated approach to identifying functional requirements assets by analyzing natural language (NL) documents [5]. They relates assets modeling techniques like definition hierarchies, n-dimensional SPL[9]. They analyzed and constructed a product line variability model which is comprised of FRP (verb + direct object) and 6 semantic cases: agentive, objective, locational, temporal, process and conditional [5]. They offer an approach based on statistical algorithm to extract verb + direct object automatically. The cases of variable models are acquired by manual. However, the method has some drawbacks. They only extracted FRPs (verb-dierctObject relations) from requirement specifications. The cases defined in functional variability model are not extracted and their definitions are not clear enough. The "Objective" case mentioned is actually the modifier of object; the feature of "Agentive" and "Objective" are not found in this method; "Process" case is too general to get more detailed information from the system functional requirement. For the drawbacks of the model, we propose an extended approach to extracting product line functional requirements. Majority of requirements are written in NL. So we choose NL document as the primary data source of extraction.

In this paper, we propose an extended functional requirements framework (EFRF) consists of 10 cases to characterize EFRF's semantics more clearly and completely. We generate converting rules after analyzing the linguistic characterization of SRSs. We present an approach to extract EFRFs from textual documents based on the structure of EFRF. Our extracting approach consists of two processes: natural language process and rule-based converting process. The extracted results show more detailed and integrated information about the variability of functional requirements. We use OVM and the SRS of auto-marker software to evaluate the usefulness and effectiveness of the approach. Our aim is to reduce the manual effort and help increasing the efficiency of generating functional variability model from text-based requirements.

The rest of this paper is organized as follows: Section 2 presents the framework of our approach, including the definition of EFRF, the whole architecture of our approach and the converting rules. Section 3 is the evaluation of the approach. Related work and conclusions are given in Sections 4 and 5, respectively.

978-0-7695-3922-5/09 $26.00 © 2009 IEEE
DOI 10.1109/ICIMT.2009.47

II. FRAMEWORK

In this section, we will illustrate the concept of EFRF and the architecture of our extracting approach. Then we will introduce converting rules and the process of our method in detail.

A. Extended Functional Requirements Framework (EFRF)

Here, we propose EFRF which is an extension of FRP proposed in [5]. We focus on SRS requirement document which follows the IEEE-STD-830 standard in a textual form [7]. We anticipate our textual-based technique can work in a wide spectrum of domains.

Following Fillmore's idea of defining a universal set of cases and Nan Niu's variation structure, we introduce a set of extended dimensions for conceptualizing the functional variability structure. An EFRF is composed by 10 different semantic cases. Thus, we consider the following variation dimensions for an EFRF.

- *Agentive* defines the agent whose activities will occur in EFRF's affairs. For example, {student}$_{Agentive}$ "do homework".
- *Action* defines the main action of the activity. For example, "TA" {mark}$_{Action}$.
- *Objective* defines the object affected by the activity. For example, "mark" {homework}$_{Objective}$.
- *Agentmod* defines the feature of agent. For example, {Senior}$_{Agentmod}$ {student}$_{Agentive}$.
- *Objmod* defines the feature of object. For example, {c++}$_{Objmod}$ {homework}$_{Objective}$.
- *Locational* defines the location where the EFRF affairs occur. For example, "do homework" {at home}$_{Locational}$.
- *Temporal* defines the duration or frequency of the EFRF's activity. For example, "mark homework" {every Sunday}$_{Temporal}$.
- *Manner* defines the way or tool by which the EFRF's activity is performed. Some examples are, "access Internet" via {Ethernet, Wireless}$_{Manner}$, "mark assignment" through {Internet}$_{Manner}$.
- *Goal* defines the goal of the EFRF's activity. For example, "do homework carefully" {in order to get an A}$_{Goal}$.
- *Constraint* defines the requirement and constraint that make the activity occur. For example, "do homework on-line" if {access Internet}$_{Constraint}$.

This extended set reflects most grammatical features that are associated with functional requirements description; hence it can serve as a framework of categories that helps analysts understand the variation points, i.e. *what* can vary, of the EFRF. The systematical and clear definition of these variable points can help the effectiveness of SPL's functional variability modeling.

B. Architecture

The whole architecture of the approach is in Fig. 1.

The source data is SPL's SRS. After the processing we will get structured functional requirements EFRF. There are two main processors in the approach: NLP Parser and Extractor. First, we preprocess these documents to get dependence relations. Second, we convert grammatical relations into EFRFs by converting rules. There are three elements necessary for the processors: Dependendency Tree, EFRF Concept and Converting Rules.

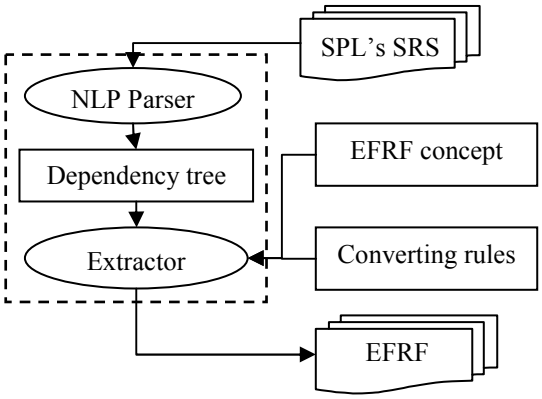

Figure 1. Architecture

Step1: NLP Parser. We choose Stanford Parser [8] as the NLP parser. It can generate dependency trees to express the grammatical relations in sentences. The grammatical relations are arranged in a hierarchy, rooted with the most generic relation named "dependent". Altogether, the hierarchy contains 48 grammatical relations. The whole hierarchy of the grammatical relations is given in [6].

Step2: Extractor. We convert grammatical relations into EFRFs by converting rules which will be illustrated in detailed later. In this step, we merge EFRFs which express the same functional requirements. If two EFRFs describe a same activity, then they will be merged by combining their cases.

C. Converting Rules

We discover converting rules as in Tab. Ⅰ which can be used to automatically convert dependency relations from requirement specifications into EFRF cases. The name of these converting rules is corresponding to the dependent relations of NLP parser. Here we consider two different voices. There are some different converting rules between these two voices.

Here are some explanations of the tags in the converting rules.

- Cases such as *Agentive* represent the result's tag when the rule is matched. The result that the rule gets will be annotated as the corresponding case.
- Cases such as *Action* occur in the rules represent the phases which have been annotated as *Action* by other rules
- Relation name such as *Subj* is corresponding to the dependent relations of NLP parser.
- ? represents any relation names
- * represents any number of words
- × represents results that should be extracted
- & represents conditions should be matched.

TABLE I. CONVERTING RULES

Cases	Active voice	Passive voice
Agentive	Subj(*, ×)	Agent(*, ×)
Action	Subj(×, *)	Agent(×, *)
Objective	obj(Action,×) & con_j(objective, ×)	subjpass(*,×)& con_j(objective,×)
Agentmod	mod(Agentive,×)	mod(Agentive,×)
Objmod	mod(Objective,×)	mod(Objective,×)
Locational	prep_at(Action,×)& ×is noun; prep_in(Action,×)& ×is noun	prep_at(Action,×)& ×is noun; prep_in(Action,×)& ×is noun
Temporal	prep_for(Action, ×); tmod(Action,×) & ? (tmodTemporal,×)	tmod(Action,×) & ? (tmodTemporal,×)
Manner	prep_through(Action, ×)	prep_through(Action, ×)
Goal	purpcl(Action, ×)& ?(Goal,×)	purpcl(Action, ×)& ?(Goal,×)
Constraint	prep_if(Action, ×)& mod(Constraint, ×)	prep_if(Action, ×) & mod(Constraint, ×)

We take the converting rules of *Goal*: "purpcl (Action, ×) & ?(Goal,×)" as example to explain the notations used in rules. "purpcl(word1, word2)" means the "purpcl" relations of word1 and word2 annotated by NLP parser. When word1 is Action, then word2 is extracted as *Goal.* "&" means after the matching of "purpcl(Action, ×)", we continue to consider rule: "?(Goal,×)". Notation "?" means every dependent relation. When word3 in "relation (word3, word4)"matches *Goal* that we extracted before, we extract word4 as *Goal.*

D. Functional Requirements Extraction Process

Based on the concepts introduced in the previous section, we present a functional requirements extraction process. The input of this process is NL document of functional requirement and converting rules. The process consists of three main steps. The output is structured variable EFRF. Detailed descriptions and examples for each step will be given in the following sections.

1) Step 1: NLP parsing

In this step, the Stanford parser parses every sentence of the document. The result is the relations between words and the index position of each word: Rela1(A1-indexA1,B1-indexB1),Rela2(A2-indexA2,B2-indexB2), Rela3(A3-indexA3, B3-indexB3), ..., RelaN(AN-indexAN, BN-indexBN). Then we interpret these relations, put them into arrays: Array1 [5] = {Rela1, A1, indexA1, B1, indexB1},Array2 [5] = {Rela2, A2, indexA2, B2, indexB2},…,ArrayN [5] = {RelaN, AN, indexAN, BN, indexBN}. For example, the following sentence1 and sentence2 are excerpts from an SRS text of Auto-Marker System (AMS).

…
TA mark homework in order to evaluate performance. *Sentence1*
…
On-time homework and late homework should be marked by TA every week. *Sentence2*
…

After the parsing process, we can get the following results:

Array1[nsubj, mark, 2, TA, 1];
Array2[dobj, mark, 2, homework, 3];
Array3[mark, evaluate, 7, in, 4]
Array4[dep, evaluate, 7, order, 5]
Array5[aux, evaluate, 7, to, 6]
Array6[purpcl, mark, 2, evaluate, 7]
Array7[dobj, evaluate, 7, performance, 8]
Sentence1

Array1[amod, homework, 2, On-time, 1];
Array2[nsubjpass, marked, 8, homework, 2];
Array3[amod, homework, 5, late, 4]
Array4[conj_and, homework, 2, homework, 5]
Array5[aux, marked, 8, should, 6]
Array6[auxpass, marked, 8, be, 7]
Array7[agent, marked, 8, TA, 10]
Array8[det, week, 12, every, 11]
Array9[tmod, marked, 8, week, 12]
Sentence2

2) Step 2： EFRF Conversion

As mentioned in previous section, we use converting rules to convert dependency relations into EFRF cases.

For every sentence, we classify its voice by its dependency relations. When a passive-voice verb occurs in a sentence, the parser will return a dependency relation named *nsubjpass*. However, the passive-voice verb may occur in either predicate verb or relative clause. So we have to distinguish these two different situations by another dependency relation named *rcmod*. If there is a relation named *nsubjpass* appears in sentence's relevant dependency relations, we can judge it as passive voice when there is no dependency relations named *rcmod* related to the passive-voice verb. Otherwise we consider it as active voice.

After that, we can use corresponding rules to match and transform the relations into EFRF cases. Here, we traverse arrays of each sentence which includes relations between words. When one array arr matches one rule, extract the word and its index, and delete arr from the set of array relations.

We take the two sentences mentioned in previous section as example. Take Array6 and Array7 of sentence 1 as example. "evaluate" is extracted as "Goal" by rule "purpcl(Action, ×)" and "performance" is extracted as "Goal" by rule "?(Goal,×)". Then we array them according to their index. "evaluate" is 7 and "performance" is 8. So the result of "Goal" will be "evaluate performance".

The extracted EFRFs of sentence1 and sentence2 are shown in Tab. Ⅱ.

TABLE II. EXTRACTED EFRFS OF SENTENCE1 AND SENTENCE2

	EFRF1	**EFRF2**
Agentive	TA	TA
Action	mark	mark
Objective	homework	homework
Agentmod	Null	Null
Objmod	Null	on-time; late
Locational	Null	Null
Temporal	Null	every week
Manner	Null	Null
Goal	evaluate performance	Null
Constraints	Null	Null

3) Step3：Merging

After the EFRF Conversion step, we have many original EFRFs, some of which expresses the same functional requirements. In this step, we first check if there are two EFRFs describe a same activity. If two EFRFs describe a same activity, then they will be merged by combining their cases. Two EFRFs will be determined describing the same activity if they satisfy these conditions: 1) the Agentives (the Actions, and Objectives) of two EFRFs are the same; 2) other corresponding cases of these two EFRFs have no conflicts, which mean they are the same or one of them of is Null.

As an example, the two EFRFs we obtained in step 2, can be merged as follows:

TABLE III. MERGED EFRF OF EFRF1 AND EFRF2

	EFRF
Agentive	TA
Action	mark
Objective	homework
Agentmod	Null
Objmod	on-time; late
Locational	Null
Temporal	every week
Manner	Null
Goal	evaluate performance
Constraints	Null

III. EVALUATION

We implemented the EFRF-extraction procedure. We use Stanford Parser for dependency relation annotating. State-of-the-art parsers like Stanford Parser have high accuracy of dependency relation annotation in simple sentences. De Marneffe et al. provided provide a comparison of Stanford dependency parser with Minipar and the Link parser. They obtained a per-dependency accuracy of 80.3% [10]. That makes the dependency parser unlikely to become an extra error source.

Test documents are SRSs of auto-marker system which follows the IEEE-STD-830 standard in a textual form. 249 sentences of the test document include 5,669 words, the number of words without punctuation marks is 4,989. Performance of our process is evaluated by the following three types of measures. Precision, Recall and F-measure applied to 4,989 words excluding punctuation marks. There are sentences of both active voice and passive voice.

We measure case-specific performance in Tab. Ⅳ. We define, A as the total number of correct cases; B as the total number of extracted cases; C as the number of correct extracted cases.

$$Recall = C / A \quad (1)$$

$$Precision = C / B \quad (2)$$

$$F\text{-}measure = 2 * Precision * Recall / (Precision + Recall) \quad (3)$$

TABLE IV. EXPERIMENTAL RESULT OF EFRF EXTRACTION IN PERCENTAGE (%)

	Recall	**Precision**	**F-measure**
Agentive	98.5	95.1	96.8
Action	96.1	93.8	94.9
Objective	81.1	80.4	80.7
Agentmod	86.4	90.1	88.2
Objmod	80.1	84.0	82.0
Locational	35.3	39.3	37.2
Temporal	40.2	91.7	55.9
Manner	18.6	60.5	28.5
Goal	47.4	78.1	59.0
Constraints	78.2	89.7	83.5

Tab. Ⅳ shows that the precision is relatively higher than recall, which is the typical feature of rule-based information extraction mission. Agentive, Action, Objective, Agentmod and Objmod had high performance in both Precision and Recall, thus get a high F-measure. It is because that the subject, predicate, object, and the attribute of subject and object are relatively obvious in a sentence. Other cases of EFRF are binding more semantic meanings, which is relatively hard to classify and extract, so the recall is relatively low using rule-based method. However, the result is acceptable.

The EFRFs can be easily transformed to OVM which shows SPL's variability more clearly. Fig. 2 illustrates the extended OVM of EFRF "mark assignment".

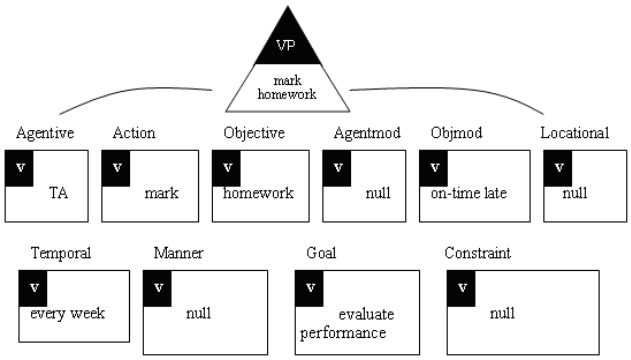

Figure 2. Orthogonal variability model of EFRF

In this OVM, a "VP" triangle represents a variation point (what can vary), which consists of ten "V" box which represents a variant (how the "VP" varies), such as the action and the objective of the EFRFs. Liaskos et al. [3] demonstrated the importance of OVM (Orthogonal variability model) modeling through different cases. The result indicates: the EFRF-extraction process can extract EFRFs to help generating the functional variability models and save manual effort.

IV. Related Work

Our work is related to several different efforts. Liaskos et al. identified variability in goal models via Fillmore's case theory [3]. Their work illuminated the importance for distinguishing between intentional variability and background variability. And their experience showed that background variability could be effective identified by agentive, objective and locational cases.

Nan Niu et al took Fillmore's case theory to analyze functional requirements variability model as well. They proposed a semi-automated extractive approach to building a product line's requirements assets [5]. They proposed a functional variability model of SPL which includes FRP and 6 semantic cases. They offer an approach based on statistical algorithm to extract verb + direct object automatically, but the cases of variable models are acquired by manual.

Our work is an extension of Nan Niu's work. First, we extend the variable model, propose an extended functional requirements framework called EFRF which has more cases and hold more detailed information than FRP. By their nature, EFRFs are functional so they are effective at modeling the SPL's actions. We have used the auto-maker SRS's to evaluate the effectiveness of our approach. The result shows the approach can aid the requirement analysis efficiently.

As mentioned at previous sections, the data source we used in the experiment is the SRSs of the auto-marker system. They follow the IEEE-STD-830 standard in a textual form, the majority of which is consisted of simple sentences. For compound sentences the method did not get good results because of the lack of more effective complicated converting rules. The method we proposed in this paper use different rule sets to get more accurate results according to different voices of sentences, which is a definite advantage that statistical algorithms cannot achieve.

V. Conclusions

In this paper, we contributed an approach to extract functional requirements from free text documents for supporting domain analysis. We proposed an extended variability model – EFRF, which includes 10 detailed semantic cases to help the detailed and integrated expression of product line functional requirements. And then we construct converting rules to automatically convert dependency relations of words in a sentence into cases to support the modeling of the variability of product line functional requirements.

Our work has a number of limitations that we plan to conquer in future. To achieve higher recall in cases extraction and to adopt the extraction of compound sentences, we will use statistical and machine learning method to complement rule-based method to make it more flexible.

Acknowledgment

The paper was sponsored by the National Hightech Research and Development Project of China(863) under the grant No.2009AA04Z106 and the National Science Foundation of China (NSFC) under the grant No.60773088.

References

[1] P. Clements and L. Northrop. Software Product Lines: Practices and Patterns. Addison-Wesley, 2001.

[2] K. Pohl, G. B¨ockle, and F. van der Linden. Software Product Line Engineering: Foundations, Principles, and Techniques. Springer, 2005.

[3] S. Liaskos et al. On goal-based variability acquisition and analysis. In Int'l Reqs Eng Conf, pages 76–85, Minneapolis, USA, September 2006.

[4] C. Fillmore. The case for case. In E. Bach and R. Harms, editors, Universals in Linguistic Theory, pages 1–88. New York: Holt, Rinehart and Winston, 1968.

[5] Nan Niu, Steve Easterbrook, "Extracting and Modeling Product Line Functional Requirements," re, pp.155-164, 2008 16th IEEE International Requirements Engineering Conference, 2008

[6] Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. 2006. Generating Typed Dependency Parses from Phrase Structure Parses. In Proceedings of LREC-06. [pdf, bib]

[7] IEEE Standards Board. IEEE recommended practice for software requirements specifications. 1998.

[8] The Stanford Parser: A Statistical Parser [EB/OL]. [2009-01-10]. http://nlp.stanford.edu/software/dependencies_manual.pdf.

[9] J. M. Thompson and M. P. Heimdahl. Extending the product family approach to support n-dimensional and hierarchical product lines. In Int'l Symp on Reqs Eng, pages 56–64, Toronto, Canada, August 2001.

[10] Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. 2006. Generating Typed Dependency Parses from Phrase Structure Parses. 5th International Conference on Language Resources and Evaluation (LREC 2006), pp. 449-454