A Scenario Model Aggregation Approach for Mobile App Requirements Evolution Based on User Comments

Dong Sun and Rong Peng(✉)

State Key Laboratory of Software Engineering,
Computer School, Wuhan University, Wuhan, China
{rongpeng,sundong}@whu.edu.cn

Abstract. With the increasingly intense competition in mobile applications, more and more attention has been paid to online comments. For the masses, comments have been viewed as reliable references to guide the choice of applications; for providers, they have been regarded as an important channel to learn expectations, demands and complaints of users. Therefore, comments analysis has become a hot topic in both requirements engineering and mobile application development. But analyzers in both areas are always not only suffered from the vast noise in comments, but also troubled by their incompleteness and inaccuracy. Therefore, how to obtain more convincing enlightenments from comments and how to reduce the manpower needed become the research focuses. This paper aims to propose a Scenario Model Aggregation Approach (SMAA) for analyzing and modeling user comments of mobile applications. By selecting appropriate natural language processing technologies and machine learning algorithms, SMAA can help requirements analysts to build aggregated scenario models, which can be used as the source of evolutionary requirements for the decision making of application evolution. The aggregated scenario model is not only easy to read and understand, but also able to reduce the manpower needed greatly. Finally, the feasibility of SMAA is exemplified by a case study.

Keywords: Scenario model aggregation approach · Aggregated scenario model · Mobile application · User comments · Kernel concerns

1 Introduction

With the increasing development of mobile Internet, mobile applications become more and more prevalent. But intense competition, short life cycle and low user adhesion are all obstacles for their success. Being alert to the changes of user expectations and evolving the mobile app accordingly are the only way for them to stand out. Therefore, online comments, as one of the most important channels of expressing user expectations and dissatisfaction, have become a vital source to obtain evolutional requirements from the masses.

The contents described in user comments always have strong relationships with scenarios. But as online comments are spontaneously described by users rather than elaborately elicited by professional requirements engineer, most of them do not contain the essential scenario information clearly, which makes them difficult to understand and results in incomplete and inaccurate understanding on the real intents of users. To help requirements analysts understand authentic user intents, it needs to develop a scenario extraction, modeling and aggregation method to extract and aggregate scenario information hidden inside those comments.

This paper proposes a Scenario Model Aggregation Approach (SMAA) which can support different components to utilize various natural language processing (NLP) technologies, machine learning algorithms and modeling methods to analyze and categorize comments and their attached records, and construct aggregated scenario models (ASMs) based on certain kernel concerns. Each part of SMAA is exemplified by a sample method. Finally, the feasibility of SMAA is shown by a case study.

The remainder of this paper is organized as follows: Section 2 introduces related work. Section 3 introduces the preliminary knowledge. Section 4 elaborates SMAA with sample methods. Section 5 presents the case study. Section 6 presents the comparison with related work. Section 5 draws out the conclusion and further work.

2 Related Work

As the development of mobile application stores, many studies focus on the analysis and utilization of online comments of mobile applications. These studies can be mainly divided into two categories.

One category is concerned on the types [1, 2], characteristics [1] and effects of App comments [3-5]. They point out that some types of comments, such as functional errors and requests for additional features, are good sources to obtain user requirements [1, 2]. But how to extract requirements is not the focus of these articles.

The other category is concerned on how to extract and analyze information from App comments. We summarized the research topics, research methods, and outputs of this category in Fig. 1.

The 1st research topic is **informative comments extraction**. As users post their comments for different purposes, some comments are noises from the perspective of requirements extraction. Therefore, many researchers [6, 7] have focused on how to extract informative comments from the raw comments. Thus, the input of this stage is raw user comments, and the output is informative comments in the perspective of requirements engineers. The informative comments extraction mainly adopts classification models or keyword based extraction methods. E.g., Bayesian classifier is used to extract informative user comments by filtering noisy and irrelevant ones [6]. A prototype system is developed to extract new feature requests by summarizing 237 keyword based grammar rules [7].

The 2nd research topic is **requirements topics extraction**. Extracting requirements topics from thousands to millions of comments manually is time-consuming and laborious. So, many researchers are dedicated on how to extract feedback topics from user comments automatically. The input of this stage is informative comments, and the outputs are representative sentences or words of each requirements topic. Topic models and cluster models are commonly used in this stage. E.g., topic models such as LDA [6-9] and ASUM [10] have been used to extract implicit topics and representative topic words from informative comments. Clustering models, such as K-Means

[11] and GN community discovery model [12], have been used to cluster comments and select the topic sentence for each cluster.

The 3rd research topic is **Topic-based analysis**. After requirements topics extraction, developers try to use the topics for further analysis. The input of this stage is usually requirements topics and their related data, the outputs are various analysis results for decision-making. Statistical analysis methods are commonly used in this stage. For example, granger causality model are used to analyze whether the utility of a topic (system aspect) is useful in forecasting the software sales [11]. Linear regression model are used to identify reasons why users like or dislike a given App [8].

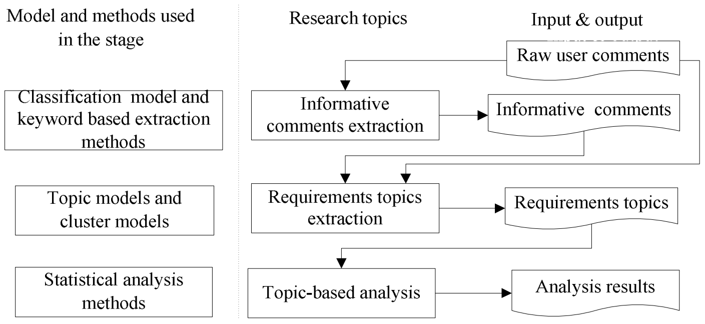

Model and methods used in the stage
Research topics
Input & output
Raw user comments
Classification model and keyword based extraction methods
Informative comments extraction
Informative comments
Topic models and cluster models
Requirements topics extraction
Requirements topics
Statistical analysis methods
Topic-based analysis
Analysis results

Fig. 1. Research topics of information extraction and analysis from App comments

The above studies focus on extracting useful information and topics from online user comments. For the providers, the extracted information can be used to learn users' expectations and complaints. But as pointed in [13], due to the arbitrariness and imperfection of the expression, it would be difficult to spy out the true intent behind the text. Thus, only relying on the surface meanings of the online comments to infer user requirements and make evolution decisions will face significant risks.

This paper focuses on providing a Scenario Model Aggregation Approach (SMAA) which can integrate the above techniques to construct ASMs from user comments and related data sources, which can help analyzers learn the implications more clearly and accurately.

3 Preliminaries

3.1 Definitions

Definition 1: Raw User Comment (RUC)
RUCs refer to the raw comments which are posted by users of some specific mobile application in a certain application market. They usually contain the information such as text, rating, and publishing time. Some comments may present user preferences, error feedbacks and advices which are useful to understand users' demands. And others may only contain useless information from the perspective of requirements engineers such as pure emotion expressions and advertisements.

Definition 2: Informative Comment (IC)
ICs refers to the comments which contain useful information for further improvement of the application, such as new feature requests, defect feedbacks and error reports. ICs are also called as potential evolution requirements, as they can be used to extract user requirements. According to the different intents, ICs can be divided into 2 categories: Improvement Comments (ImCs) and Fault Feedback Comments (FFCs).

Definition 3: Kernel Concern (KC)
KC refers to the core user demand implicated in a specific IC.

Definition 4: Informative Comment with Scenario Information (SIC)
SICs refer to ICs with scenario information. Scenario information refers to the information, such as trigger operations, usage contexts and underlying rationales, which can be used to reconstruct the scenario.

This paper aims to automatically extract scenario information of similar comments and build ASMs.

3.2 Stanford Parser

Stanford parser is a multi-language syntactic parser developed by Stanford natural language processing team [14]. Its output has many formats, such as part-of-speech (POS) tagged text, phrase structure tree (PST), and dependency relations (DRs) [14]. POS, PSTs and DRs are always adopted by various methods to understand the text written in natural language automatically.

In this paper, firstly, we use Stanford Word Segmenter to split each comment written in Chinese into a sequence of words; then, the sequence is imported into Stanford parser to get its PST and DRs for further processing.

3.3 Decision Tree Model

Among various classification models, decision tree model (DTM) is famous for its good understandability and high accuracy. DTM is a tree-like model used to predict or classify. Each non-leaf node represents a "test" on an attribute and each leaf node represents a class label. Each branch represents the outcome of the test and the path from root to a specific leave represents classification rules.

In SMAA, it recommends using DTM to identify SICs. The dependency relations identified by Stanford parser and the category attribute identified by analysts will be used to construct DTM.

4 Scenario Model Aggregation Approach and Its instantiation

4.1 Scenario Model Aggregation Approach

To help requirements engineers to extract the scenario information and aggregate ASMs from RUCs, a kernel concern based scenario model aggregation approach is proposed. The approach includes the following 5 stages, as shown in Fig. 2:

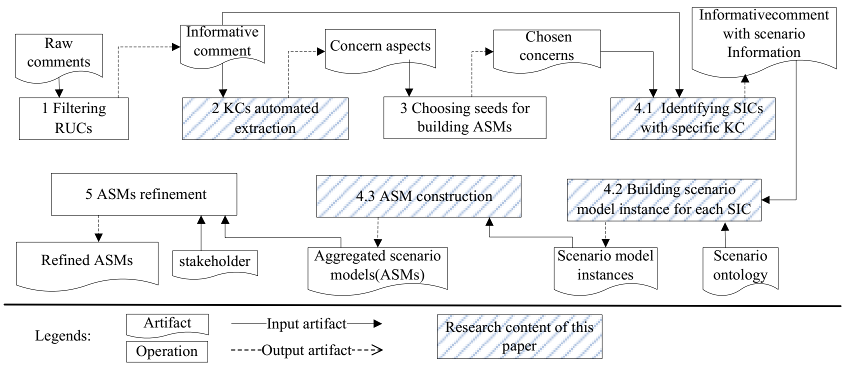

Raw comments
1 Filtering RUCs
Informative comment
2 KCs automated extraction
Concern aspects
3 Choosing seeds for building ASMs
Chosen concerns
Informativecomment with scenario Information
4.1 Identifying SICs with specific KC
5 ASMs refinement
4.3 ASM construction
4.2 Building scenario model instance for each SIC
Refined ASMs
stakeholder
Aggregated scenario models(ASMs)
Scenario model instances
Scenario ontology
Legends:
Artifact
Operation
Input artifact
Output artifact
Research content of this paper

Fig. 2. The main steps of SMAA

1) Filtering RUCs: In this stage, the goal is to obtain ICs from RUCs through de-noising and filtering. Many machine learning algorithms can be used to fulfill the task, such as Naive Bayes [6, 15].

2) KCs automated extraction: This stage aims to mine representative KCs from the collection of ICs. NLP and data mining technologies can be used in this stage, which will be exemplified in Section 4.2.

3) Choosing seeds for building ASMs: In this stage, an analyst needs to choose KCs they are interested in, and take these concerns as seeds to construct ASMs.

4) Constructing ASMs for each KC: In this stage, the following operations should be performed on each chosen KC:

4.1) Identifying SICs with a specific KC: Find all ICs which contain the KC and identify those with scenarios information to construct a set of SICs. Various classification approaches, such as decision tree and deep learning, can be used. The sample method is described in Section 4.3.1.

4.2) Building scenario model instance for each SIC: For each SIC, extract the scenario information from the text of the SIC and its attached record, such as device type and OS type; and then create a scenario model instance. Ontology-based and context-aware methods can be used to extract scenario information automatically. An ontology-based scenario model instance construction method is exemplified in Section 4.3.2.

4.3) ASM construction: Aggregate all the scenario elements from the scenario model instances of a specific KC and build a ASM. The way of aggregating scenario model instances should be decided according to the characteristics of the scenario elements. For instance, “AND/OR tree” can be used for aggregating scenario elements of “trigger condition”.

5) ASMs refinement: The kernel concern based ASMs is checked one by one manually to verify whether it misses some key elements or not. If the information of any necessary element is deficient, the analyst should organize the relative stakeholders together to refine the models. The refined model can be used as the reference model, which can help providers and analyzers to understand the user demand correctly.

Due to space limitation, the steps 1, 3 and 5 will not be discussed further as many existing methods can be used directly. In the following sections, only the sample methods of step 3 and 4 will be elaborated. Since the sample data are from Android

Market (http://apk.hiapk.com/) in China, the sample methods are constructed to be suitable to analyzing user comments written in Chinese.

4.2 Kernel Concerns Automated Extraction Based on NLP

Many techniques can be used to extract KCs from ICs, such as LDA and ASUM [6-10]. Here, we'll instantiate it by using NLP parser, which includes the following two steps: 1) Topic comments extraction: Extract topic comments from all ICs; 2) Kernel concerns (KCs) extraction: Extract KCs from the topic comments.

4.2.1 Topic Comments Extraction

As shown in Fig. 3, the topic comments extraction contains two steps: 1) Clustering ICs; and 2) choosing representative topic comments (RTCs) for each cluster.

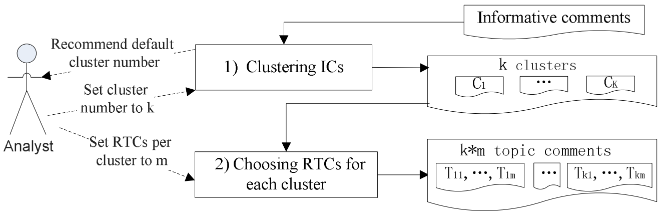


Fig. 3. Topic comments extraction

1) Clustering ICs: Using K-means can classify ICs to several clusters. The recommended cluster number (RCN) can be calculated according to a modified version of Cans' metric [17, 18], in which RCN will be high if individual comments are dissimilar with each other, and low otherwise. If all comments are same, RCN equals to 1. RCN should be equal or lesser than the number of the comments. Analysts can manually adjust RCN according to their own will or reading ability: the bigger the cluster number is, the more the representative topic comments they need to read [18].

2) Choosing RTCs for each cluster: After the analyst determines the number of RTCs for each cluster (here suppose it to m), the comments which is m nearest to the centroid (measured by cosine similarity) will be chosen as the representatives of the cluster.

4.2.2 Kernel Concerns Extraction

Here, feature requests and fault feedbacks are regarded as the default KCs recommended automatically from the perspective of requirements engineers. And analysts can manually modify the KCs according to their preferences.

By analyzing the features of output formats of Stanford parser, the KCs extraction algorithm can be designed as follow:

Algorithm 1: KCs extraction algorithm (KCsEA)
Input: representativeTopicComment
Output: kernelConcern

1) Parse **representativeTopicComment** by Stanford Parser to get its phrase struc-

ture tree **GT** and dependency relations set **DependencySet**;

2) Identify the core word of verb phrase **VP**:

2.1) Locate in the bottom right **VP** in the **GT**;

2.2) Annotate the last verb (**VV** or **VA**) of this **VP** as its core word;

3) Expand the core word: Find the words in **DependencySet** which has one of the following dependency relations with the core word: **advmod** (adverbial modifier), **nsubj** (nominal subject), and **dobj** (direct object); and combine the located modifier with the core word to construct the output **kernelConcern**;

4) return **kernelConcern**.

The returned **kernelConcern** will be regarded as recommended KC for the specific topic comment. It will be displayed together with the topic comment to analysts, which can help them understand its meaning and context. E.g., the KC of RTC "升级后经常死机" (in English "often crashes after upgrade") is shown as Fig. 4.

In Chinese:	"升级后经常死机"
In English:	"**often crash** after upgrade"

Fig. 4. The display style of a RTC and its recommended KC

Recommending KCs automatically can greatly reduce the manual workload of extracting them from a large volume of ICs. At the same time, it allows the analyzer to modify or redesignate the KCs according to his/her preference, which is helpful to construct ideal scenario models.

4.3 KC Based ASM Construction

Many modeling methods can be used to construct ASMs based on KCs. The following method is just a sample method.

4.3.1 Decision Tree Based Automatic Identification of Informative Comments with Scenario Information

As stated in Section 3.3, the paths from root to leaf represent classification rules in decision tree model. These paths are helpful to understand which dependency relation has closely correlation with scenario information. Therefore, decision tree model is chosen as the classification model to distinguish ICs with or without scenario information.

As shown in Fig. 5, the identification process of SICs based on decision tree model has two stages: constructing classifier and using classifier.

The process of **constructing classifier** is as follow:

1) Annotating the categories: Select some comments as the training dataset, and ask some analyst(s) to pick out all SICs. Thus, all training data will have a class label. Table 1 showes some sample ICs annotated by an analyst.

2) Parsing IC: Get the dependency relations of each IC in the training dataset by Stanford parser.

3) Vectorizing IC: Use dependency relations and the category information to

construct attribute vectors for ICs: $ca = (d_1, d_2, \ldots, d_n, r)$, where d_i (i=1..n) represents a certain dependency relation, n is the number of frequently used dependency relation in Chinese [14], and r is its corresponding class label.

For example, suppose that the kth IC in the training dataset contains dependency relations 1, 2 and 3, and it is annotated as SIC. Then its attribute vector is: $ca_k = (1,1,1, 0, \ldots,0, 1)$

4) Training the model: Select the C4.5 algorithm in WEKA 3.12 to train the model.

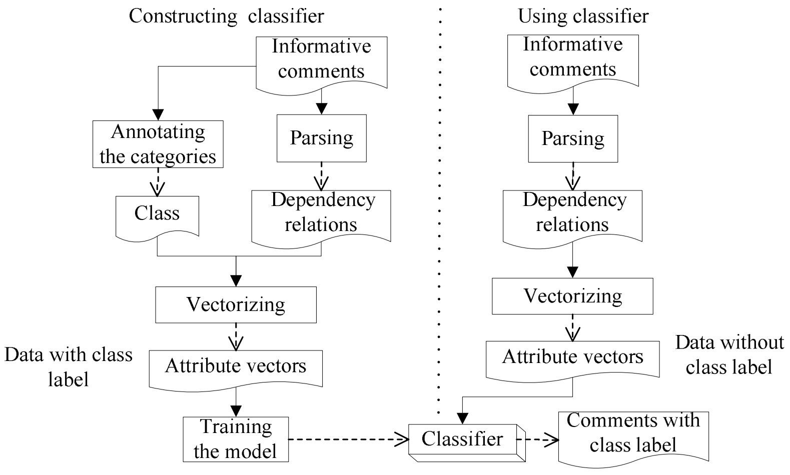


Fig. 5. The identification process of SICs based on decision tree model

The process of **using classifier** is as follows:

First parse all ICs one by one to get the dependency relations of each comment; then, construct the attribute vector $cd_i=(d_{i1},d_{i2},\ldots,d_{in})$ for each IC; finally, use the trained model to test all vectors to determine whether they are SICs or not.

Table 1. ICs with or without annotation

NO	Comments in Chinese (C) and English (E)	SIC?
1	打开定位后就自动退出 (C) Abnormal exit after open the positioning function (E)	☑
2	有时候自动退出，望改进 (C) Sometimes it will exit abnormally, please fix it.(E)	☐
3	老是闪退 (C) Always quit without prompt.(E)	☐
4	5.0版本有时出现闪退 (C) Version 5.0 always exits without prompt.(E)	☑

4.3.2 Scenario Model Instances Construction

As pointed in [16], scenario model can help requirements analysts understand the authentic intentions of users. Thus, building scenario model is crucial for extracting scenario information from comments and their attached records.

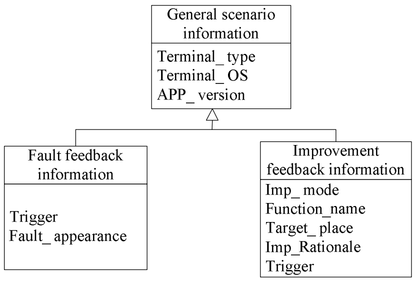


Fig. 6. Scenario metamodel

As shown in Fig. 6, a scenario metamodel is defined according to the available scenario information which could be extracted from the features of user comments in mobile application stores. The scenario metamodel defines that a scenario model should contain the following information:

1) General scenario information: General scenario information contains some basic elements such as terminal type, terminal OS and application version. These kinds of information are recorded by mobile applications as soon as users post their comments. Thus, they can be crawled or directly accessed from the application stores.

2) Categorized information: the information extracted from the text of the comment. As Fault Feedback Comments (FFCs) and Improvement Comments (ImCs) are 2 typical kinds of comments, the information extracted from them should be also classified into 2 categories: improvement feedback (IF) info and fault feedback (FF) info.

The scenario information in FFCs mainly concerns on: the trigger (the operation(s) triggered the fault) and the fault appearance (system appearance when fault occurs).

The scenario information in ImCs mainly focuses on: the improvement mode (added/ modified / deleted / improved / lowered, et al), the specific function name which need to improve, the target place (usually represented by its parent function), the improvement rationale and the trigger. The data models of 2 kinds of scenario model instances are shown in Fig. 7(a) and 7(b).

```
FFScenarioModel {
// General Info
Terminal terminal;
OS os;
AppVersion appVersion;
// FaultFeedback ScenarioInfo
Type type=faultFeedback
KernelConcern kernelConcern;
Trigger trigger;
FaultAppearence faultAppearence;
}
```

(a) Data model of FF Scenario Model

```
IFScenarioModel {
// General Info
Terminal terminal;
OS os;
AppVersion appVersion;
// ImprovementFeedback ScenarioInfo
Type type=improvementFeedback
KernelConcern kernelConcern;
Trigger trigger;
ImpMode impMode;
FuncName funcName;
Target target;
ImpRationale impRationale;
}
```

(b) Data model of IF Scenario Model

Fig. 7. Data models of scenario model

Based on the above scenario metamodel, an ontology based scenario model information extraction method is proposed.

The ontology tree model is shown as Fig. 8. The top ontology is the root. It

According to the source of the ontology, the ontology can be divided into **DomainOntology** and **ApplicationOntology**. **DomainOntology** contains the domain general concepts and the relationships among them. The concepts such as "add", "modify" and "delete" are all instances of DomainOntology, as they are general concepts for the whole domain. **ApplicationOntology** represents the ontology related to a specific application. For example, the concepts, such as "search around" and "positioning", may only be used for the map related applications.

According to the nature of the ontology, it can be divided into **ActionOntology**, **EntityOntology** and **ModifierOntology**. **ActionOntology** can be further subdivided into **OperationOntology**, such as "click" and "move", and **ExpectationOntology**, such as "hope" and "suggestion". **EntityOntology** can be subdivided into **FunctionOntology**, such as "navigation" and "positioning", **PeripheralOntology**, such as "camera" and "microphone", and **FaultOntology**, such as "exception" and "crash".

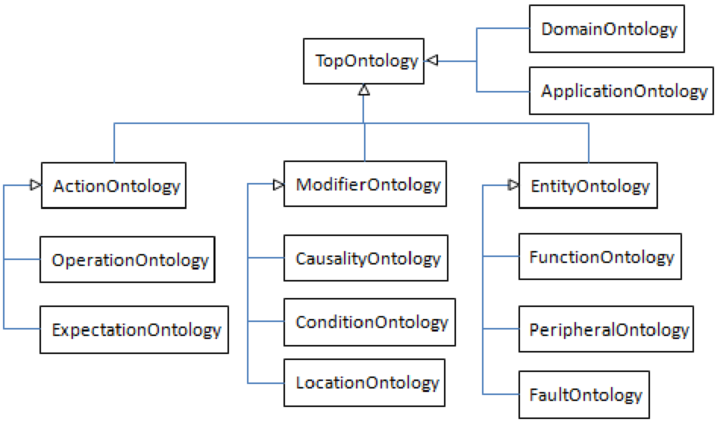


Fig. 8. The ontology tree model for scenario model element extraction

The scenario model instance construction algorithm based on ontology tree is as follow.

Algorithm2: Scenario model instance construction algorithm (SMICA)
Input: ***ontoModel***; // the ontology tree model
sic; //the comment with scenario information
Output: ***sic.scenarioModel*** //the scenario model instance of ***sic***.

1) Parse ***sic*** by Stanford parser to get the phrase structure tree ***sic.struTree*,** and the dependency relations set ***sic.DependencySet***.

2) Scan the leaf nodes in ***sic.struTree*** one by one according to the ***ontoModel***: if a leaf node matches one of the concepts in the **ontoModel**, the node will be marked with the ontology category label; otherwise it is marked with label "**Others**".

3) Construct scenario model instance according to ***sic***:

3.1) When ***sic*** contains a phrase marked with ExpectationOntology, extract scenario elements according to the data model of IF scenario model, and construct the corresponding model instance ***sic*.*ScenarioModel***:

3.1.1) ***sic.scenarioModel.type= improvementFeedback*；**

3.1.2) Find the leaf node which is marked with ExpectionOntology, and up traverse ***sic.struTree*** from this node to find the nearest ***vp*** which contain a phrase marked with OperationOntology, and assign the verb (***v***) to the element **impMode** : ***sic.scenarioModel.impMode*** = ***v***, and assign the verb's nearest direct object ***e*** to the element **funcName**: ***sic.scenarioModel.funcName =e***;

3.1.3) If there exists a leaf node ***fn*** in ***sic.struTree*** which is marked with FunctionOntology, check whether its parent node has a leaf node labeled as LocationOntology. If yes, assign ***fn*** to element "**Target**": ***sic.scenarioModel.target=fn***;

3.1.4) If there exists a leaf node which is marked with ConditionOntology, find the word ***tp*** which has a dependency relation **case**, and assign ***tp*** to the element "**trigger**": ***sic.scenarioModel.trigger=tp***；

3.1.5) If there exists a leaf node which is marked with CausalityOntology, find the word ***tp*** which has a dependency relation **case**, and assign ***tp*** to the element "**impRationale**": ***sic.scenarioModel.impRationale = tp***;

3.2) When ***sic*** contains a phrase marked with FaultOntology, extract scenario elements according to the FF scenario metamodel, and construct its model instance ***sic.scenarioModel*** accordingly as follows:

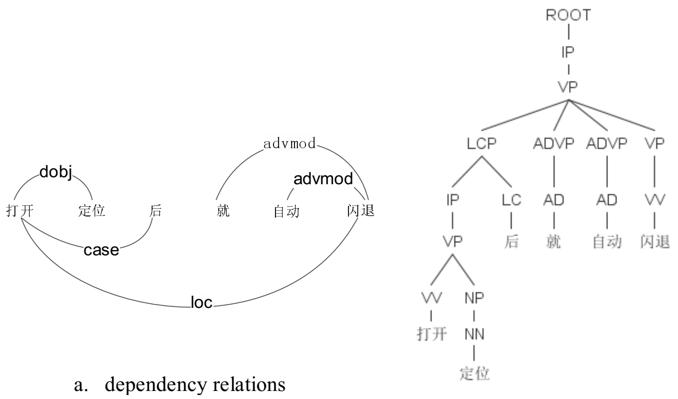
3.2.1) ***sic.scenarioModel.type=faultFeedback;***

3.2.2) Locate the leaf node ***fo*** which is marked with FaultOntology; expand the word ***fo*** with its nearest adjunct word and record the expand phrase as ***efo***, and assign it to the element "**faultAppearence**": ***sic.scenarioModel. faultAppearence = efo***;

3.2.3) If there exists a leaf node which is marked with ConditionOntology, find the word ***tp*** which has a dependency relation **case**, and assign it to the element "**trigger**": ***sic.scenarioModel.trigger = tp***；

3.3) Automatically extract the general scenario elements of ***sic*** and assign them to the corresponding model elements;

4) Return ***sic.scenarioModel***.



a. dependency relations

b. phrase structure tree

Fig. 9. A parsing example

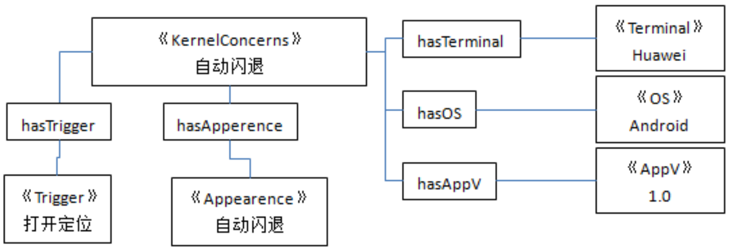


Fig. 10. The corresponding scenario model instance of Fig. 9

For example, the comment "打开定位后就自动闪退" ("Quit without prompt after open the positioning function" in English) can be parsed by Stanford parser to obtain the dependency relations and the phrase structure tree as shown in Fig. 9(a) and 9(b). Referred to the ***ontoModel***, the leaf nodes "打开"(Open)、"后"(After)、 "闪退"(Quit without prompt) are marked with OperationOntology, LocationOntology, and FaultOntology respectively. Therefore, its model instance can be constructed according to the above algorithm shown in Fig.10.

4.3.3 Aggregated Scenario Model Construction

Aggregated scenario models are also classified into 2 categories: the improvement feedback aggregated scenario models (IFASM) and the fault feedback aggregated scenario models (FFASM), as shown in Fig. 11(a) and 11(b). Where,

```
ASM {
// General Info
CTerminalSet terminalSet;
COSSet osSet;
CAppVersionSet appVersionSet;
// fault feedback aggregation
Type type=faultFeedback
CKernelConcern kernelConcern;
CTriggerSet triggerSet;
CFaultAppSet faultAppSet;
}
```

(a) Data model of FFASM

```
ASM {
// General Info
CTerminalSet terminalSet;
COSSet osSet;
CAppVersionSet appVersionSet;
// improvement feedback aggregation
Type type=improvementFeedback
CKernelConcern kernelConcern;
CTriggerSet triggerSet;
CImproveModeSet improveModeSet;
CFuncNameSet funcNameSet
CTarget target
CFaultAppearenceSet faultAppearenceSet;
CRationaleSet rationaleSet
}
```

(b) Data model of IFASM

```
CKernelConcern {
Dimension dim;
DimValue dimValue
Times times;
}
```

(c) Data Model of CKernelConcern

```
CTrigger {
Trigger trigger;
Times times
}
CTriggerSet triggerSet= new Set of CTrigger
```

(d) Data Model of CTriggerSet

Fig. 11. Data models of ASMs

1) Type: Represent the type of the aggregated scenario model;

2) CKernelConcern: As shown in Fig. 11(c), it has 3 attributes: Dimension, DimValue and Times. Dimension represents the designated aggregate dimension which can be any dimension in the scenario model instance; DimValue is a specific value the aggregation focuses on; Times is a counter which record the number of SICs aggregated under the condition of the same Dimension with same DimValue.

3) CTriggerSet: Trigger is an attribute in scenario model instance. It records the trigger of the comment. Accordingly, CTriggerSet records the triggers of all aggregated model instance. For each element CTrigger in CTriggerSet, it consists of the attribute trigger and times, where trigger is the same as the one in the model instance, and times is the count of model instances with a certain same trigger.

4) Other attributes: The data structure of each set attribute is similar with CTriggerSet.

By using the above data structure, the following algorithm can be used to aggregate the model after all model instances are available.

```
Algorithm 3: ASM construction algorithm
Input: insScenarioModelSet; // the scenario model instance set to be aggregated
            insModelType; // the type of model instances to be aggregated
            dim, dimvalue; // the dimension and its value to be aggregated
Output: m; // the aggregated scenario model
1)  New an empty ASM:
      m = new ASM (insModelType);
2) Set the attributes of the Kernel Concern:
      m.kernelConcern.dim=dim;
      m.kernelConcern.dimValue=dimValue;
      m.kernelConcern.times=0;
3) Traverse the insScenarioModelSet:
      For each model instance c in insScenarioModelSet, execute:
      if(c.scenarioModel.dim==dimvalue&&c.scenarioModel.type==m.type) then
      {    m.kernelConcern.times++;
         for any d different from Dimension{dim, type} in c
         { if exist (c.scenarioModel.d, m.d) then
            {e=getelem(m.d, c.scenarioModel.d); e.times++; putelem(m.d, e)}
            else { e=newelem(c.scenarioModel.d, 1); add(m.d, e)}
4) return m.
```

Through the above sample methods, the SMAA can be implemented to support the requirements engineers to analyze the kernel concerns hidden in raw user comments and construct aggregated scenario models.

5 Case Study

A mobile application named "Baidu Map" in Android Market (http://apk.hiapk.com/) is selected to exemplify the effect of SMAA. Baidu Map has 2335 user comments in this store by January 1, 2014. Following the steps in SMAA strictly, the results is as follows:

1) Filtering RUCs: Using Naive Bayes and down sampling method described in [15], 610 ICs are obtained.

2) KCs automated extraction:

2.1) Topic comments extraction: According to section 4.2.1, the K-means clustering algorithm in Weka 3.6.10 is adopted to cluster the ICs. The cluster number is set to 11 based on Can's metric [18], and the distance function adopted is Euclidean distance. The nearest comment to the centroid is selected as the RTC for each cluster, which is listed in the 2nd column of Table 2.

2.2)Kernel concerns extraction: According to section 4.2.2, the KCs extraction algorithm based on Stanford parser is adopted to extract and recommend kernel concerns for each topic comment. These concerns are listed in the 3rd column of Table 2 and the final accepted KCs by the authors are listed in the 4th column.

3) Choosing seeds for building ASMs: Choose "闪退" ("quit without prompt" in English) as the seed, namely the selected KC to construct and refine the ASM.

4) ASM construction based on the KC:

4.1) Identifying SICs with the KC: Find all ICs which contain the chosen seed "闪退(quit without prompt)" through retrieval. 15 ICs are retrieved, and 11 of them are labeled as SICs. The decision tree model is trained by C4.5 algorithm in Weka 3.6.10.

4.2) Building scenario model instance for each SIC: According to the algorithm proposed in section 4.3.2, the scenario model instances are built for each comment;

4.3) ASM construction: According to the algorithm in section 4.3.3, the aggregated scenario model with the kernel concern "闪退(quit without prompt)" is built as shown in Fig. 12.

Table 2. Comments Analysis for Baidu Map in Andriod Market

CID	Representative comments of each cluster	Recommended KCs	Accepted KCs
1	4.0不能用 4.0 cannot be used	不能用 Cannot be used	
2	百度地图这弹广告不说,而且不能退出老是自动启动 Not to mention pop ads, Baidu map exit abnormally and often restart.	退出老是自动启动 exit abnormally and often restart	自动启动 Restart
3	新版本4.5，闪退 New version 4.5, quit without prompt	版本闪退 Version quit without prompt	闪退 Quit without prompt
4	度娘又升级了，误差增加到十千米。 Baidu update again, error increased to 10 kms	增加误差 error increase	误差增加 Error increased
5	我是note2，装好后不运行，但电量里看百度地图耗电最高 Note2, not running after installed, but its power consumption is the highest	耗电最高 Highest power consumption	耗电 Power consumption
6	地图不准 Map is not accurate	地图不准确 Map is not accurate	
7	耗电量大，手机发热厉害，都快50℃了 Large power consumption, it become very hot, almost 50℃	50℃了 Almost 50℃	发热 Become hot

Table 3. (*Continued*)

8	定位不准 Positioning is not accurate	定位不准 Positioning is not accurate	
9	地图更新太慢！ Map update too slow!	**地图更新太慢** Map update too slow	
10	定位太差了! Positioning is too bad!	定位太差 Positioning is too bad!	
11	没有农村离线地图。 No rural offline map.	离线地图 No offline map	

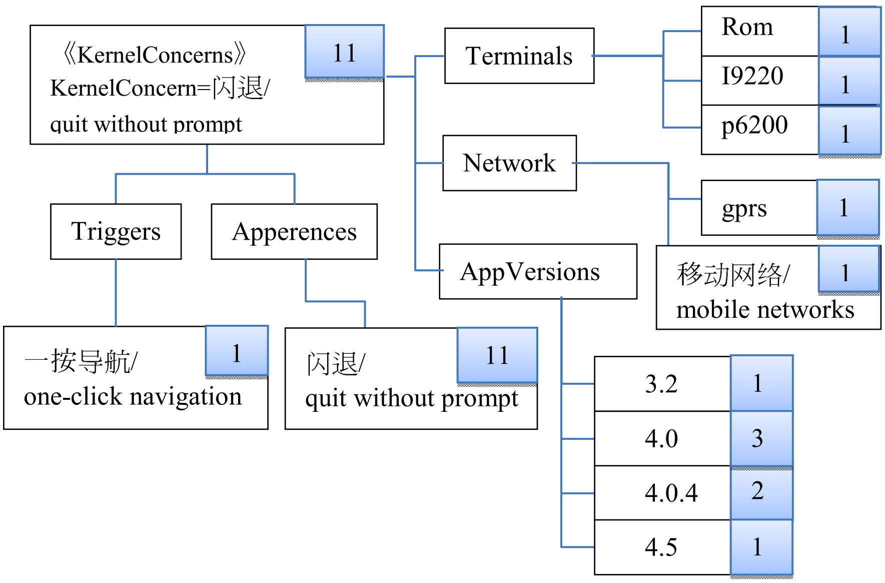


Fig. 12. The ASM in the concern of "闪退(quit without prompt)"

According to the ASM, a requirements analyzer can clearly learn that the version 4.* often "闪退" (quit without prompt), one of its trigger operations is "一按导航" (click navigation), and occurred in GPRS and mobile networks. But the type of the terminal is not clear. If needed, the analyzer can launch a refinement activity according to the step 5 in SMAA.

6 Comparison with Related Work

As summarized in section 2, previous work focused on how to use specific technology to extract and analyze information form App comments. The main outputs of previous work were specific information such as informative comments, topic words, topic sentences, and analysis results based on topic.

This focuses of the proposed approach SMAA are that 1) automated extract scenario information from comments, and 2) organize the information in appropriate forms for good visibility and understandability. The outputs of SMAA are ASMs which are suitable for evolution requirements analysis and refinement.

7 Conclusion and Future Work

This paper proposes a scenario model aggregation approach SMAA with the following functionality: 1) It can support raw user comments filtering and kernel concerns extraction; 2) it can help requirements engineers to construct aggregated scenario models based on kernel concerns they are interested in. These models are helpful for understanding the authentic user needs and judging how critical they are. Therefore, they can be used for making evolution requirements decision. According to the elaborated sample methods, a case study is carried out, which exemplifies its effects.

This approach has strong scalability and flexibility, as it can adopts various existing natural language processing technologies and machine learning algorithms to filter raw user comments, classify informative comments, extract kernel concerns, and construct aggregate scenario models.

Further studies should be conducted in the following directions: 1) quantitative assessments are needed to verify the effect of different machine learning algorithms and tools adopted in the approach; 2) the effectiveness and efficiency of the sample methods and algorithms should be evaluated; 3) more cases and empirical experiments should be carried out to verify its effectiveness; and 4) an integrated analysis environment should be developed to improve the usability of the approach.

Acknowledgements. The research is supported by the National Natural Science Foundation of China under Grant No. 61170026.

References

1. Pagano, D., Maalej, W.: User feedback in the appstore: an empirical study. In: 2013 21st IEEE International Requirements Engineering Conference (RE), pp. 125–134 (2013)
2. Khalid, H.: On identifying user complaints of iOS apps. In: 2013 35th International Conference on Software Engineering (ICSE), pp. 1474–1476 (2013)
3. Harman, M., Jia, Y., Zhang, Y.: App store mining and analysis: MSR for app stores. In: Proceedings of the 9th IEEE Working Conference on Mining Software Repositories, pp. 108–111 (2012)
4. Kim, H.-W., Lee, H.L., Son, J.E.: An exploratory study on the determinants of smartphone app purchase. In: The 11th International DSI and the 16th APDSI Joint Meeting, Taipei, Taiwan (2011)
5. Chia, P.H., Yamamoto, Y., Asokan, N.: Is this app safe? a large scale study on application permissions and risk signals. In: Proceedings of the 21st International Conference on World Wide Web, pp. 311–320 (2012)

6. Chen, N., Lin, J., Hoi, S.C.H., Xiao, X., Zhang, B.: AR-Miner: mining informative reviews for developers from mobile app marketplace. In: Proceedings of the 36th International Conference on Software Engineering, pp. 767–778 (2014)
7. Iacob, C., Harrison, R.: Retrieving and analyzing mobile apps feature requests from online reviews. In: 2013 10th IEEE Working Conference on Mining Software Repositories (MSR), pp. 41–44 (2013)
8. Fu, B., Lin, J., Li, L., Faloutsos, C., Hong, J., Sadeh, N.: Why people hate your app: making sense of user feedback in a mobile app store. In: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1276–1284 (2013)
9. Oh, J., Kim, D., Lee, U., Lee, J.-G., Song, J.: Facilitating developer-user interactions with mobile app review digests. In: CHI 2013 Extended Abstracts on Human Factors in Computing Systems, pp. 1809–1814 (2013)
10. Galvis Carreño, L.V., Winbladh, K.: Analysis of user comments: an approach for software requirements evolution. In: Proceedings of the 2013 International Conference on Software Engineering, pp. 582–591 (2013)
11. Jiang, W., Ruan, H., Zhang, L.: Analysis of economic impact of online reviews: an approach for market-driven requirements evolution. In: Zowghi, D., Jin, Z. (eds.) APRES 2014. CCIS, vol. 432, pp. 45–59. Springer, Heidelberg (2014a)
12. Jiang, W., Ruan, H., Zhang, L., Lew, P., Jiang, J.: For user-driven software evolution: requirements elicitation derived from mining online reviews. In: Tseng, V.S., Ho, T.B., Zhou, Z.-H., Chen, A.L., Kao, Hung-Yu. (eds.) PAKDD 2014, Part II. LNCS, vol. 8444, pp. 584–595. Springer, Heidelberg (2014b)
13. Kompan, M., Bieliková, M.: Context-based satisfaction modelling for personalized recommendations. In: 2013 8th International Workshop on Semantic and Social Media Adaptation and Personalization (SMAP), pp. 33–38 (2013)
14. Chang, P.-C., Tseng, H., Jurafsky, D., Manning, C.D.: Discriminative reordering with Chinese grammatical relations features. In: Proceedings of the Third Workshop on Syntax and Structure in Statistical Translation, pp. 51–59 (2009)
15. Ni, Y., Peng, R., Sun, D., Lai, H.: Potential evolution requirements detect method based on user comments. Wuhan Univ. (Nat. Sci. Ed.) **61**, 347–355 (2015)
16. Sutcliffe, A.: Scenario-based requirements engineering. In: Proceedings of 11th IEEE International Requirements Engineering Conference, 2003, pp. 320–329 (2003)
17. Davril, J.-M., Delfosse, E., Hariri, N., Acher, M., Cleland-Huang, J., Heymans, P.: Feature model extraction from large collections of informal product descriptions. In: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, pp. 290–300 (2013)
18. Can, F., Ozkarahan, E.A.: Concepts and effectiveness of the cover-coefficient-based clustering methodology for text databases. ACM Trans. Database Syst. **15**, 483–517 (1990)