GUITAR: An Ontology-based Automated Requirements Analysis Tool

Tuong Huan Nguyen, John Grundy, Mohamed Almorsy
Faculty of Science, Engineering and Technology
Swinburne University of Technology
Melbourne, Australia
{huannguyen, jgrundy, malmorsy}@swin.edu.au

***Abstract*—Combining goal-oriented and use case modeling has been proven to be an effective method in requirements elicitation and elaboration. However, current requirements engineering approaches generally lack reliable support for automated analysis of such modeled artifacts. To address this problem, we have developed GUITAR, a tool which delivers automated detection of incorrectness, incompleteness and inconsistency between artifacts. GUITAR is based on our goal-use case integration meta-model and ontologies of domain knowledge and semantics. GUITAR also provides comprehensive explanations for detected problems and can suggest resolution alternatives.**

***Index Terms*—Goal-oriented requirements engineering; Use case; Ontology-based requirements analysis; incorrectness; incompleteness; inconsistency; detection and resolution**

I. Introduction

Goal-use case coupling techniques have been proven to be an effective method in requirements elicitation and elaboration [6]. 3Cs problems (incorrectness, incompleteness and inconsistency) are key challenges of the modeled artifacts' quality. Several techniques have been developed to help in requirements management and analysis. KAOS [7] supports the management of requirements conflicts. However, it does not explicitly detect inconsistency between goals and use cases, incorrectness and incompleteness among the artifacts. Cesar [5] allows the analysis of natural language requirements using domain ontologies. However, its ontological term-mapping technique for 3Cs problem detection may not be sufficient to deal with complicated cases. In addition, it does not support analysis in goal-use case models. RAT/QRA analysis tool [1] depends on adopting linguistic techniques and domain ontologies, which make it limited to linguistic defects (i.e. ambiguity, readability or subjectivity), similarity between pairs of requirements, and unused boilerplates. Thus, there is still a lack of comprehensive automated supports dedicated for goal-use case integration models. For instance, how to verify if a goal or use case is not correctly specified? How to ensure a use case is matched with its associated goal? How to identify whether a required goal/use case has not been elicited? How to identify inconsistent artifact specifications?

To automatically deal with these 3Cs related problems, artifact specifications need to be transformed into a form that is syntactically and semantically understandable by machines. We developed GUITAR (**G**oal-**U**se case **I**ntegration **T**ool for **A**nalysis of **R**equirements) as an extension to our previous

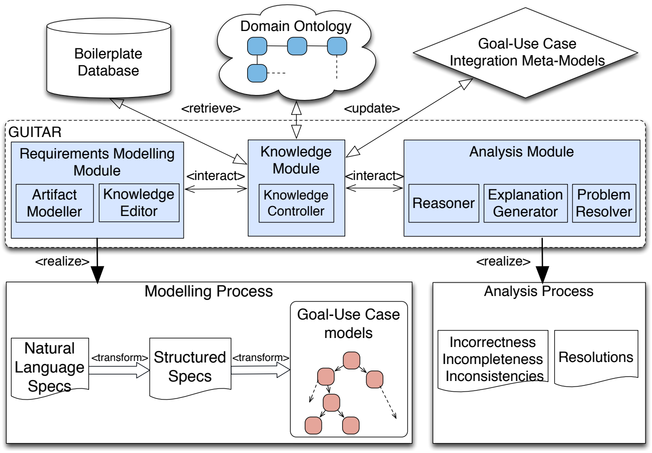


Fig. 1 GUITAR Process

work on inconsistency detection in goal models [3, 4]. GUITAR allows a wide range of natural language artifacts descriptions to be translated into Manchester OWL Syntax and integrated with ontologies of domain knowledge and semantics to automate the analysis process. In addition, our developed meta-model for goal-use case integration enables the detection of syntactical problems. GUITAR also automatically generates explanations and possible resolutions for identified problems.

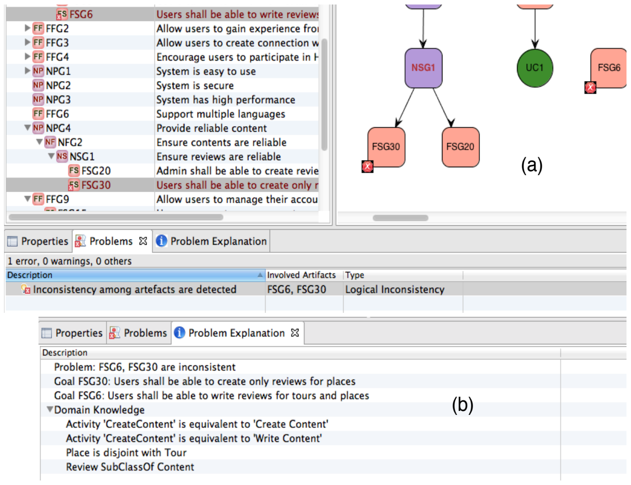
More information about GUITAR can be obtained from http://www.it.swin.edu.au/personal/huannguyen/guitar.html.

II. GUITAR's Process

Fig. 1 shows an overview of GUITAR. The tool consists of the requirements modeling, knowledge and analysis modules which support the modeling and analysis of requirements.

A. Modelling Process

GUITAR allows the modeling of goals and use cases (referred to as *artifacts* in this paper). These two concepts are integrated based on a meta-model that provides the categorization and valid relationships between them. For each artifact, the tool requires both a natural language specification and a structured specification that is used for automated analysis. In our work, functional grammar [2] is used as the underlining model for structured specifications due to its integration capability with domain ontologies. In this structured specification, each term is mapped to an ontological concept to allow semantic analysis. To help ensure artifact specification's



FSG6	Users shall be able to write reviews
FFG2	Allow users to gain experience fro
FFG3	Allow users to create connection w
FFG4	Encourage users to participate in H
NPG1	System is easy to use
NPG2	System is secure
NPG3	System has high performance
FFG6	Support multiple languages
NPG4	Provide reliable content
NFG2	Ensure contents are reliable
NSG1	Ensure reviews are reliable
FSG20	Admin shall be able to create revie
FSG30	Users shall be able to create only r
FFG9	Allow users to manage their accou

1 error, 0 warnings, 0 others

Description	Involved Artifacts	Type
Inconsistency among artefacts are detected	FSG6, FSG30	Logical Inconsistency

Description
Problem: FSG6, FSG30 are inconsistent
Goal FSG30: Users shall be able to create only reviews for places
Goal FSG6: Users shall be able to write reviews for tours and places
Domain Knowledge
Activity 'CreateContent' is equivalent to 'Create Content'
Activity 'CreateContent' is equivalent to 'Write Content'
Place is disjoint with Tour
Review SubClassOf Content

Fig. 2 An Inconsistency Example

quality (i.e. unambiguousness), GUITAR incorporates a database of boilerplates to help writing natural language specifications. For example, if we have a goal *"System shall send update email to users every two weeks"*, a requirements analyst can choose a boilerplate (i.e. *<agent> shall <action> <object> <beneficiary> <frequency>*) and fill in the parameters (i.e., *<agent>*). Based on the specified parameters, GUITAR automatically transforms the natural language specification into a structured specification (i.e., *Agent(System) + Verb(Send) + Object(UpdateEmail) + Beneficiary(User) + Frequency (Quantity(2) + MeasurementUnit(Week)),* the terms within parenthesis are ontological concepts). Domain ontologies and boilerplates can be edited (via *knowledge editor and knowledge controller* components) and reused in different projects.

B. Analysis Process

GUITAR supports the detection and resolution of 3Cs problems in modeled goals and use cases.

1) Problem Detection - Syntactical problem detection is supported by a 2-layer goal-use case integration meta-model. The artifact layer defines different types of artifacts across levels of abstraction and their relationships while the specification layer provides guidance on the composition of artifacts. The meta-model helps, for instance, with the detection of a missing goal type, an incorrect artifact's specification, or inconsistent links between artifacts.

GUITAR relies on ontologies of domain knowledge and semantics for semantic requirements analysis. At the core of our technique is the representation of activities. Each activity contains an action and an object (i.e. *create reviews*). Since various relationships between activities may exist in a domain (i.e., "*create content*" requires "*edit content*"), capturing such relations into ontologies is useful to identify mismatches between artifacts (inconsistencies) or missing artifacts (incompleteness). In addition, relationships between concepts or activities (i.e. *user* and *traveller* are equivalent, in a *traveller network* domain, so are '*write review*' and '*create review*' even *write* and *create* are not) provide the semantics of terms used in structured artifact specifications. GUITAR's *reasoner* allows structured specifications to be automatically transformed into Manchester OWL Syntax for automated reasoning. Fig. 2 shows an example of a detected inconsistency. The involved artifacts are highlighted (a) and the explanation is provided (b).

2) *Problem explanations and resolution suggestions* - For detected problems, GUITAR automatically generates resolution alternatives together with detailed problem explanations based on domain ontologies. Explanations are helpful for requirements analysts to get insight into the issues and select appropriate repairing alternatives.

III. Evaluation

We have evaluated GUITAR with two industrial case studies, one for a traveller social network system and another for an online publishing system. We used precision and recall metrics to assess GUITAR's soundness and completeness in detecting 3Cs problems. Given domain ontologies of high quality, we have obtained 95% and 90% on average for soundness and completeness respectively.

IV. Conclusion and Future Work

In this paper, we have presented GUITAR, a tool providing automated analysis of goals and use cases for incompleteness, incorrectness and inconsistencies. GUITAR also enables the generation of problem explanations and resolution alternatives. Our planned future works include: (1) support automated transformation of natural language requirements into goal-use case models. (2) Improve the ontology editor with visualization editing support. (3) Conduct a formal user evaluation on industry practitioners to assess the usability of the tool.

Acknowledgement

This research is supported by the Victorian Government under the Victorian International Research Scholarships scheme and the Australian Research Council under Linkage Project LP130100201.

References

[1] Requirements Quality Suite. Available from: http://www.reusecompany.com/requirements-quality-suite.

[2] S.C. Dik, "The theory of functional grammar," Walter de Gruyter, 1989.

[3] T.H. Nguyen, B.Vo, M. Lumpe, J.C. Grundy, "KBRE: a framework for knowledge-based requirements engineering," Software Quality Journal, vol. 22, p. 1-33, March 2014.

[4] T.H. Nguyen, B.Vo, M. Lumpe, J.C. Grundy, "REInDetector: a framework for knowledge-based requirements engineering," in Proceedings of the 27th International Conference on Automated Software Engineering, 2012.

[5] A. Rajan and T. Wahl, "CESAR: Cost-efficient Methods and Processes for Safety-relevant Embedded Systems," Springer, 2013.

[6] C. Rolland, C. Souveyet, and C.B. Achour, "Guiding goal modeling using scenarios," IEEE Transactions on Software Engineering, vol. 24, 1998, p. 1055-1071.

[7] A. V. Lamsweerde, "Goal-oriented requirements engineering: A guided tour," in Proceedings of Fifth IEEE International Symposium on Requirements Engineering, 2001.