Integrating Goal-Oriented and Use Case-Based Requirements Engineering: The Missing Link

Tuong Huan Nguyen, John Grundy, and 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 shown as an effective method of requirements engineering. To ensure the quality of such modeled artifacts, a conceptual foundation is needed to govern the process of determining what types of artifacts to be modeled, and how they should be specified and analyzed for 3Cs problems (completeness, consistency and correctness). However, such a foundation is missing in current goal-use case integration approaches. In this paper, we present GUIMeta, a meta-model, to address this problem. GUIMeta consists of three layers. The artifact layer defines the semantics and classification of artifacts and their relationships. The specification layer offers specification rules for each artifact class. The ontology layer allows semantics to be integrated into the entire model. Our promising evaluation shows the suitability of GUIMeta in modeling goals and use cases.**

***Index Terms*—Goal and Use Case, Meta-model, Functional Grammar, Ontology.**

I. Introduction

Requirements Engineering (RE) is an iterative process of eliciting, specifying, analyzing, and managing requirements on a software system [17]. Goal-use case integration modeling (GUIM) [8, 9, 18] has been recognized as a key approach in this process. GUIM can capture the underlying rationale of the system being developed while aligning the business objectives with the functionalities and constraints of system components. The details of system-user interactions (use cases) are also modeled and linked to system goals. Such a combination enables GUIM to provide a comprehensive view of the system [1]. GUIM has been used to enhance requirements elicitation [1, 8], modeling [9, 18], and facilitate architecture design [10].

However, despite of the recognized benefits of GUIM, no work has been done to establish a conceptual foundation on which system goals and use cases should be modeled together. Such foundation is needed to provide guidance as to what types of artifacts should be modeled, how they are specified, classified, and connected to each other. In fact, existing GUIM techniques are generally isolated. Different approaches, with different foci, have different ways to specify, classify and connect artifacts (i.e., goals, use cases). For instance, Cockburn [4] defines the *summary*, *user* and *sub-function* levels of abstraction. The first two levels are for functional goals while the sub-function level is for use cases. Lee et al. [9] classify goals under three facets: rigid vs. soft goals, actor-specific vs. system-specific goals and functional vs. non-functional goals; and use cases can be connected to goals in the intersecting type of functional, actor-specific and rigid. Such isolation makes it difficult if models created in different approaches were to be combined. Also, no existing GUIM approach is comprehensive enough to model both goals and use cases as each lacks support for certain artifact types. Thus, requirements engineers are not well supported in modeling both goals and use cases without a fundamental foundation.

Moreover, the lack of a conceptual foundation prevents goal-use case models to be adequately analyzed. Such models need to be analyzed for defects such as incompleteness, inconsistency, and incorrectness (the 3Cs problems). Although some GUIM approaches have their own ways of analysis, they do not sufficiently address key questions such as: how to verify if an artifact is not properly specified? How to check if artifacts are not correctly connected? How to ensure a use case is matched with its associated goal? How to detect if a required artifact has not been elicited? In fact, a conceptual foundation with clearly defined and classified artifacts, and dependences between them could potentially a solution to these problems.

In this paper, we present a novel **G**oal-**U**se Case **I**ntegration **Meta**-model (GUIMeta) as a conceptual foundation of GUIM. GUIMeta was designed to first, provide a more comprehensive way of modeling and analyzing goals and use cases, and second, unify the existing GUIM approaches. GUIMeta is the underlying component of our **G**oal and **U**se case **I**ntegration **F**ramework (GUI-F) that provides automated support for the extraction of goal-use case models from textual documents and the analysis of such models for syntactic and semantic 3Cs problems. GUIMeta consists of three layers. The *artifact layer* defines the classification of commonly used artifacts in a goal-use case integration model and their relationships. The *specification layer* provides the specification rules of each artifact type based on functional grammar [6]. The *ontology layer* defines the structure of ontologies that can be optionally integrated with artifact specifications to facilitate the semantic understanding of the entire model. We summarize the key contributions of GUIMeta as follows:

(1) A comprehensive meta-model of goals and use case components, and their relationships in GUIM
(2) A set of specification rules for artifacts
(3) A method of using ontology to provide semantics to goal-use case models. As shown later, several benefits are offered by this feature.

(S1)This software system will be a Social Networking System for travellers around the world. (S2)This system will be designed to allow travellers to better plan their travels by providing tools to assist in facilitating the communication between travellers. (S3)By supporting the travellers to share and gain experiences, the system will meet the their needs while remaining easy to understand and use. (S4)More specifically, this system is designed to allow a traveller to quickly write reviews for different places and tours
...
Use case: Create Reviews
Brief Description: A user creates a review
Initial Step-By-Step Description
(S5) Before this use case can be initiated, the traveller has successfully logged into the system.
Step 1. (i) The editor selects to create a review.
Step 2. (i) System prompts the user to select a review category.
Step 3. (i) The user selects review category. (ii)The list of categories includes hotel, attraction, restaurant and tour.
Step 4. (i) System displays the suitable review creation form to user.
Step 5. (i) The user enters the review content. (ii) A review content contains subject, overall rating, individual ratings and comment
Step 6. (i) System validates the review. (ii) The validation should be completed within 1 second.
Step 7. (i) If the review content is valid, the system stores the review into the database; (ii) else the system prompts the user to repeat step 5.

Fig. 1. Goal-Use Case Modeling Scenario

The rest of the paper is organized as follows. In Section II, we discuss the key motivation for this work. Section III provides an overview of our approach. Section IV describes the details of GUIMeta. Section V presents the evaluation results and our applications of GUIMeta. Section VI and section VII provide additional discussions on the meta-model and related research respectively. Section VIII summaries our contribution.

II. Motivation

In this section, we discuss the motivation for this work. We use the term *artifact* to refer to goals, use cases and use case components (i.e., step, condition).

A. Modeling Goals and Use Cases

Consider a scenario in which goals and use cases need to be extracted and modeled from the requirements text in Fig 1.

1. What to model and how to specify them?

Although general understanding of requirements and goals may help identify artifacts from text (i.e., sentence (S1) should not be considered as an artifact since it is an introduction to a system), to ensure the consistency of goal-use case modeling, fundamental guidelines are needed to instruct requirements engineers on what artifacts to be modeled, what their roles are and how they should be specified. For instance, should the sentences Step 3-(ii), Step 5-(ii) and Step 6-(ii) be modeled as artifacts? From our analysis, they are important artifacts (data and quality constraints) within the use case. However they are not supported in many existing approaches (i.e., [9], [4]). In addition, although the artifact *"Facilitate the communication between travelers"* can be modeled in several GUIM techniques, it is classified into different categories by those techniques. For instance, it is categorized as a rigid goal in [9], summary goal in [4] and design goal in [8]. Therefore, we need to unify them in a conceptual foundation. Moreover, artifacts in a model need to be consistently specified. For example, should the entire sentence (S2) be a specification for an artifact or should it be split into two artifacts whose specifications are highlighted in Fig. 1? Unfortunately, existing approaches provide insufficient guidance on how to specify such artifacts.

2. How to specify relationships between artifacts?

The existing GUIM techniques lack support for several important relationships between artifacts at goal levels and use case level. For instance, the *refine* relationship between the goal *"travellers shall be able to quickly write reviews"* (from (S4)) and the use case constraint *"the validation should be completed within 1 second"* (from Step 6-(ii)) (both are about speed) is neglected. Moreover, since different techniques use different artifact categories, different sets of relationships are defined. Thus, we need to study the correlation of these relationships and unify them under a conceptual foundation.

B. Analyzing Goal-Use Case Integration Models

Since different existing GUIM approaches define different sets of modeled artifacts, no single method can be used to analyze a goal-use case model if such a model does not follow a specific approach. Moreover, not all techniques provide analysis support (i.e., [4, 8, 15]). In addition, as each existing approach lacks support for several types of artifacts that may be modeled in a goal-use case integration model, their analysis method, if any, is inadequate for such a model. Therefore, to provide a comprehensive analysis framework for GUIM, there needs to be a foundation that classifies GUIM's commonly used artifacts and relationships, and defines rules as to how each artifact and relationship should be specified to be considered correct, consistent and complete. Below, we provide some examples of common 3Cs problems in GUIM:

Incorrectness: The modeled goals and use cases may be incorrect due to their unsound specifications or ill-formed relationships between them.

Example 1: Consider the functional goal G "*The system being built shall be secure*" and the use case step S "*User enters article subject quickly*". These specifications are malformed for their types. G should describe a system's functionality rather than quality while S should not state how a user achieves a task.

Example 2: Consider a use case UC that has the pre-condition "*user has been logged in*" and the post-condition "*traveller has been signed in*". Such a use case specification is invalid since its post-condition is identical to its pre-condition, given that *user* is equivalent to *traveller* and *logged in* is identical to *signed in* in this domain.

Example 3: The goals "users shall be able to create website contents" and "users shall be able to create travel articles" are linked to each other via a bidirectional *refinement* link. Such a dependency is invalid as *refinement* is a one-way relationship.

Incompleteness: Goal and use case model can be incomplete. (i.e., missing artifacts or relationships).

Example 4: Consider two goals G1 *"Users shall be able to register for a membership"* and G2 *"System shall support communication"*. G1 should have an operationalizing use case while G2 needs to be further refined into more specific goals.

Example 5: Consider goal G "*Users shall be able to create travel articles*" and a use case UC for "a user creates a travel article". Assume they are not connected, then that is incomplete since UC describes the steps to realize G.

Inconsistency: Mismatches between specifications give rise to inconsistencies.

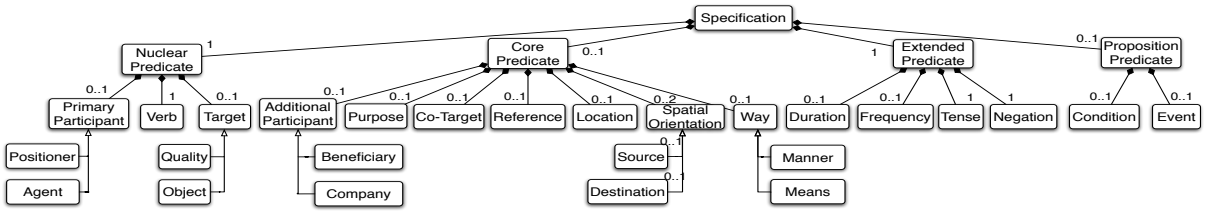

Fig. 2. Structure of a Specification

Example 6: The goal G "*Users shall be able to create travel articles*" is operationalized by a use case UC that has the description of "*A user edits a review*". This is considered an inconsistency because UC needs to describe the steps to achieve the goal that it operationalizes (not right in this case).

Example 7: Consider two goals "*Users shall be able to create reviews for tours*" and "*Users shall be able to write only reviews for places*". They are inconsistent given "*write reviews*" and "*create reviews*" are equivalent activities while *tour* and *place* are disjoint concepts in the domain of interest.

III. Our Approach

We developed a three-layered meta-model (GUIMeta) as a conceptual foundation for GUIM. The *artifact layer* classifies the artifacts and relationships to be modeled. It also specifies the dependencies between relationships. The *specification layer* provides specification rules for the defined artifacts. It provides the guidelines on what should and should not be included in the specifications of each artifact type. The *ontology layer* defines the ontology structure to allow semantics to be integrated into the entire model. GUIMeta also incorporates a categorization of correctness, consistency and completeness problems in a model. Based on that, 3Cs problems can be verified.

To unify the existing concepts in GUIM, we established a correspondence between GUIMeta and the existing GUIM approaches. The formation of our specification rules is inspired by function grammar [6], in which a specification is described by a verb and a number of parameters, each having its own semantic function (e.g., agent, object, location). That provides a consistent way to interpret the semantics of a specification.

The design of GUIMeta was based on over 450 goals and 170 use cases from the literature and industry. These are from many different domains including web applications, embedded systems, process control systems, and information systems. It was done in four steps. First, we focused on the artifact layer. We studied the existing GUIM approaches to identify the overlaps and differences between their defined artifacts. Based on that, we developed the core categories of artifacts. We then examined the exemplar goals and use cases to recognize the goals and use case components that were not classifiable into the core categories. In such cases, we obtained new categories. At the end of this step, the newly identified categories were merged into the core categories. A similar process was done for characterizing relationships between artifacts. The correspondence between our artifact and relationships categories and those in existing GUIM approaches was also an outcome of this step. Second, we analyzed individual goals and use cases. We used functional grammar to parameterize the specifications of goals and use cases, and studied the commonalities among semantic functions usually used for each category of goal or use case component. The outcome of step two is the specification layer. Third, we focused on providing semantics to artifact specifications. The criterion is to allow the meaning tracking for each semantic function in a specification. The result of this step was the ontology structure and how ontology is integrated to models. Lastly, based on the artifact and relationship classifications, we developed a categorization of incompleteness, inconsistency and incorrectness in goal-use case integration models.

IV. Goal-Use Case Integration Meta-model

Due to space limitations, only selected components of GUIMeta are presented in this paper. Interested readers are referred to http://goo.gl/ttqX4Z *for more details*.

A. Functional Grammar

Functional Grammar (FG) is a general theory concerning the grammatical organization of natural languages [6]. In our work, FG is used to parameterize artifact textual specifications into different parameters called semantic functions. Such parameterization provides a standard way to interpret the semantic role of each group of words in a specification and thus offers means to analyze the semantics of specifications.

Fig. 2 shows the key components of an artifact specification in our meta-model (i.e., goals, use case steps/conditions). A specification consists of four predicates, each denoting a number of semantic functions. For instance, *nuclear predicate* contains elements describing which action is conducted (*verb*), by whom (*agent*), or on what target (*object*). *Core predicate* enriches *nuclear predicate* with details about the *beneficiary* or how an activity is performed (*manner*).

Each semantic function is described by a *term*. For instance, *verbal terms* describe activities and thus are used to specify statements; *nominal terms* denote entities and are used to specify the values of most semantic functions. For example, the nominal term *Head(Review) + Quantifier(Quantity(2) + Comparative_Operator(More_Than_Or_Equal)) + Quality(Attribute(High_ quality))* describes the phrase *'2 or more high quality reviews'*. The following examples demonstrate the use of FG in parameterizing artifact specifications.

Example 1: A goal *"System shall notify users when new messages arrive"* is parameterized as *"Agent(system) + Verb(notify) + Object(users) + Event(Positioner(Attribute(new) + Head(messages)) + Verb(arrive) + Tense(present) + Negation(false)) + Tense(present) + Negation(false)"*.

Example 2: A use case step *"If the review has less than 50 characters, the system shall display an error message"* is parameterized as *"Condition(the review has less than 50*

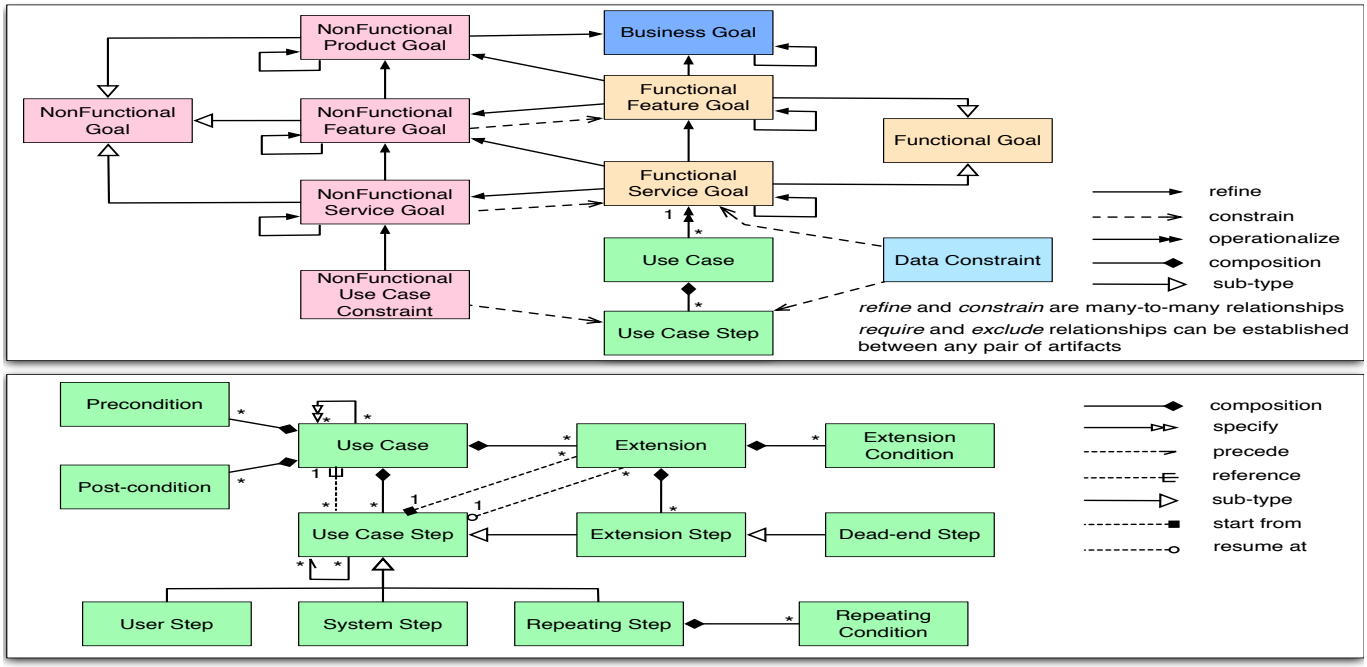

Fig. 3. Artifact Layer

characters) + Agent(system) + Verb(display) + Object(error message) + Tense(present) + Negation(false)". The *condition* is structured as *"Positioner(review) + Verb(has) + Object(Quantifier(Comparative_Operator(less_than) + Quantity(50) + Head(characters)) + Tense(present) + Negation(false)*".

B. Artifact Layer

Fig. 3 illustrates the *artifact layer,* which defines the following components:

Business goals (BG) describe the business objectives of the software system being built. A business goal does not include any information about what the system should do or how it should operate (e.g., *"Improve quality of travel planning"*).

Functional feature goals (FFG) list features a system should support in order to achieve business goals. FFGs should not offer details as to what functions are needed for the system to support a feature; rather a FFG is an abstract description of the feature itself (e.g., *"System shall support communication"*).

Functional service goals (FSG) provide the details of how a feature is achieved. A FSG describes what function a user or the system can perform. The main difference between a FSG and a FFG is that, a FSG is detailed enough to form a testable unit and is operationalized by a use case, whereas a FFG cannot have any connected use cases (e.g., *"Users shall be able to create travel articles"*).

Non-functional product goals (NPG) describe quality constraints on the entire product. A NPG should not contain any information about particular features or services supported by the system (e.g., *"The system being built shall be secure"*).

Non-functional feature goals (NFG) name quality constraints of a particular feature of the system (i.e., which is specified in a FFG). A NFG should not contain detailed information about how such constraint can be met by the system (e.g., *"Users shall be able to share experience easily"*).

Non-functional service goals (NSG) list quality constraints on a particular service. For example, *"The article creation process shall be familiar to typical Internet users"* is a NSG restricting the FSG *"Users shall be able to create articles"*.

Use cases only operationalize functional service goals. Our use case structure is adopted from Cockburn's use case template [5]. A use case includes pre/post conditions, steps, and extensions. Steps describe system-user interactions, whereas extensions handle exceptions. We also define two types of constraints in use cases. A *non-functional use case constraint* (NUUC) describes a quality constraint at use case level (e.g., *"The system validates a user's identity within 2 seconds"*). A *data constraint* (DC) captures a data requirement for a particular entity mentioned in a functional service goal or use case (e.g., *"A review contains a rating, and a comment*)".

The artifact layer also defines commonly used relationships between artifacts in goal and use case integrated modeling [3, 5, 19]. The main relationships are described as follows:

Refine relationships are used to model the refinements of goals and constraints. *AND-refine relationships* are used for cases of *minimal refinement*, which means an artifact would only be satisfied if all the sub-artifacts linked to it via AND-refine relationships are satisfied. *OR-refine relationships* are used in cases of *alternative refinement*, which means the artifact being refined can be satisfied by fulfilling any of the sub-artifacts involved in the OR-refine relationships. *Optional-refine relationships* are used in cases of *optional refinement*. This denotes that the sub-artifacts involved in Optional-refine relationships are preferred options but they are not strictly required for the parent-artifact to be fulfilled.

Constrain relationships are used to define non-functional or data constraints on a functional goals or a use case step. For instance, the data constraint DC1 constrains the use case step UC1_Step5.

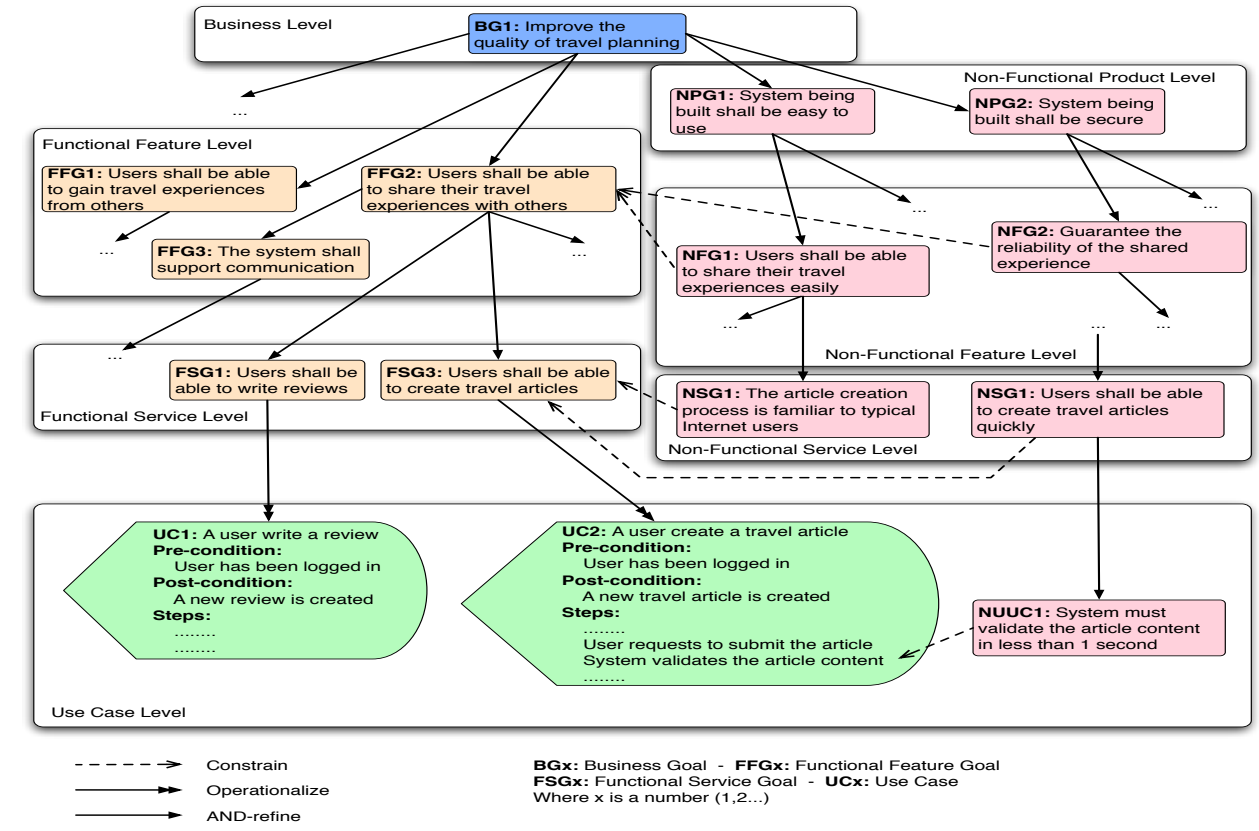

Fig. 4. A Sample Goal-Use Case Integration Model

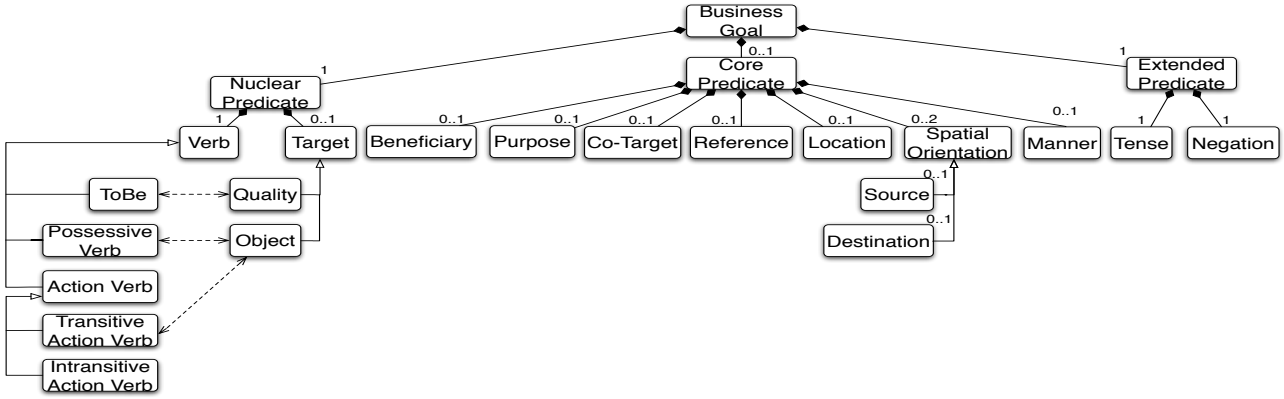

Fig. 5. Specification Rules for Business Goals

B1. <transitive action verb> <object> ((for) <beneficiary>) ([with | to]) <reference> ([in | at | on] <location>) (for <purpose>)

B2. <transitive action verb> <object> ((for) <beneficiary>) ([in | at | on] <location>) (from <source>) (to <destination>) (for <purpose>)

B3. <intransitive action verb> ((for) <beneficiary>) ([with | to | of]) <reference> ([in | at | on] <location>) (for <purpose>)

Note: (…) denotes optional parameters. [x | y | .. | z] denotes alternative parameters

Fig. 6. Specification Rules for Business Goals

Require relationships describe situations in which the satisfaction of an artifact requires the satisfaction of another.

Exclude relationships are opposite to *require* relationships. They describe the situations in which two artifacts in the model cannot be both satisfied.

Operationalize relationships are used to connect a use case to a functional service goal to model the situation that the use case describes the system-user interactions to achieve the goal.

Fig. 4 shows a partial goal-use case model where each artifact is classified into our defined artifact types.

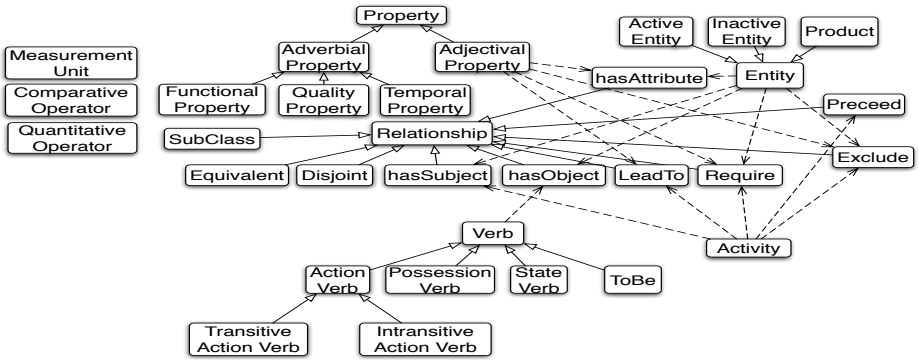

Fig. 7. Ontology Structure

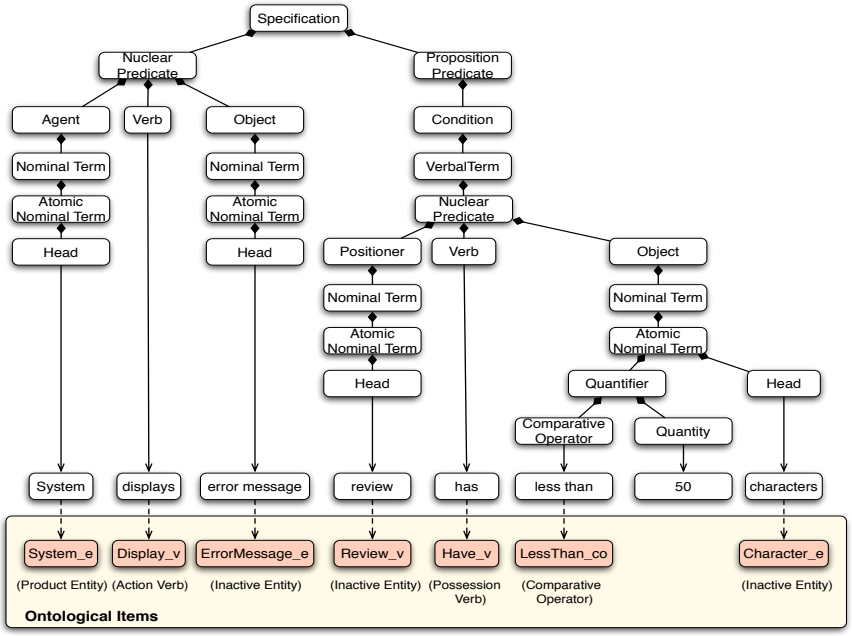

Fig. 8. Example of Integrating Specifications with Ontology

C. Specification Layer

The *specification layer* provides rules for specifying each type of artifacts defined in the artifact layer. The benefits of specification models are twofold. First, they provide guidelines for writing artifacts. For instance, as business goals are usually high-level strategic statements, *condition* or *duration* should not be specified while other parameters (i.e., *beneficiary*, *destination*) are permitted.

Second, they enable the detection of 3Cs problems. Fig. 5 shows the specification model of business goals that indicates the compulsory and optional semantic functions of a business goal specification. For example, the nuclear predicate's possible semantic functions are *verb*, *object*, *location*, *source* and *destination* in which verb and object are compulsory components. Consider a valid business goal *"Assist travellers to plan their travels"*, parameterized as *"Verb(assist) + Object(Agent(traveller) + Verb(plan) + Object (travel))"*. If a condition were specified (i.e. *"If the demand is high"*), the specification would become invalid because a condition semantic function is not permitted here. This specification model is used to define specification rules in the form of boilerplates[1]. Each rule outlines one or more ways of writing specifications of a certain type of artifacts. Fig. 6 presents some specification rules for business goals, which are derived from the specification model in Fig. 5. The specification boilerplates for other artifacts can be found at http://goo.gl/ttqX4Z.

D. Ontology Layer

GUIMeta enables semantics to be added into models by allowing ontologies to be integrated with artifact specifications in case semantics are needed. Fig. 7 depicts GUIMeta's ontology structure that includes the key components as follows:
Verb refers to verbs in the domain that describe *actions* (i.e., *display*, *create*), *possession* (i.e., *have*, *contain*) or *statuses* (i.e., *arrive*, *come*). Action verbs are further classified as transitive and intransitive action verbs.
Entities are core elements in the domain. *Product entities* refer to the system and its components (e.g., *system*, *product*). *Active entities* are the rest of the entities that can perform an action (e.g., *user*, *librarian*). *Inactive entities* cannot perform actions; they are objects of actions (e.g., *book*, *password*).

[1] Boilerplates are templates for writing textual artefact specifications

Table I. Correspondence between key GUIM approaches and GUIMeta

Approach	Existing GUIM Approaches' Concept	GUIMeta's Corresponding Concept(s)
Lee [9]	Rigid goal: goal that must be completely satisfied	FFG, FSG
	Soft goal: goal that can be partially satisfied	BG, NPG, NFG, NSG
	Actor-specific goal: actors' objectives with the system	FFG, NFG, FSG, NSG with agent is an actor
	System-specific goal: requirements on services that system provides	NPG, NFG, NSG with agent is a system or system component
	Functional goal	FFG, FSG
	Non-functional goal	NPG, NFG, NSG
	Original goal: intersection of rigid, actor-specific and function goal	FFG, FSG with agent is an actor
	Cooperative and conflict relationships between goals	Require/refine and exclude relationships respectively
	Satisfied/denied relationships between an original goal and a use case	Satisfied/denied relationships can be represented in GUIMeta through *operationalize*, *refine*, *require* and *exclude* relationships.
Cockburn [4]	Summary goal: system objectives	FFG
	User goal: user's task	FSG
	Sub-function goal: describe user activities	Use case step
Rolland [15], Kim [8]	Business Goal: ultimate purpose of the system	BG
	Design goal: possible manners of fulfilling a business goal	FFG
	Service goal: possible manners of providing services to fulfill design goals	FSG
	Refinement/alternative relationship	Refine/alternative relationship
Gorschek [7]	Product level requirement: product strategies	NPG, FFG
	Feature level requirement: high-level feature that the product supports	FFG
	Functional level requirement: action that users can perform	FSG
	Component level requirement: steps of how each function is performed	DC, NUCC

Measurement unit refers to measurement units in the domain. They can be classified further (e.g., data storage units like *MB*, or time units like *MHz*).

Property includes *adjectival properties* (e.g., *high*, *low*) and *adverbial properties*. An adverbial property is either a *functional property* (e.g., *automatically*, *manually*) or *qualitative property* (e.g., *quickly*, *safely*)

Equivalent, ***Subclass*** and ***Disjoint*** respectively specifies analogous, refinement and non-overlapping relationships between concepts.

Fig. 8 shows an example in which a use case step "if the review has less than 50 characters, the system displays an error message" is parameterized into terms and ontological items (represented as shaded boxes). Extended predicates are omitted for simplicity.

E. GUIMeta's Correspondence with GUIM Approaches

Apart from providing a fundamental foundation for GUIM, GUIMeta was designed to unify existing GUIM approaches. In fact, we provide one-to-one mappings between their concepts and GUIMeta's. This enables the transformation of models from those approaches into our format. Based on that, the combination of models specified in different approaches can be facilitated. Table II presents the correspondence between key GUIM techniques and GUIMeta. It can be seen that they only cover subsets of artifacts and relationships in GUIMeta.

V. EVALUATION

A. Evaluation on Example Requirements Documents

We evaluated GUIMeta by assessing its suitability in goal and use case integration modeling. Specifically, we aimed to address the following questions:

- ***RQ1:*** *How well does GUIMeta handle different types of artifacts in goal and use case integration modeling?*
- ***RQ2:*** *How suitable are GUIMeta's specification rules for goal and use case integration modeling?*

To answer these questions, we employed six industrial case studies. In each case study, we manually identified goals and use cases and used them to evaluate GUIMeta. To address *RQ1*, we focused on the coverage of GUIMeta's artifact layer. Specifically, we aimed to identify which goals and use case components in each case study could and could not be classified by our artifact layer. To address *RQ2*, we evaluated the appropriateness of our specification rules, which are defined in GUIMeta's specification layer, in specifying the classified goals and use case components. To do that, we manually parameterized each artifact and verified if there was one or more of these problems: (1) the artifact specification cannot be sufficiently parameterized using the functional grammar's semantic functions adopted in our work, or (2) the parameterized specification of such artifact contains a semantic functions that is not included in the corresponding specification rule (e.g., a *condition* semantic function exists in the parameterization of a business goal while it is not included in the business goal specification rule).

The first three case studies we used came from the domains of traveller social network (TSN), online publication system (OPS) and split payment system (SPS)[2]. Three further case studies were chosen from the PROMISE (**PR**edict**O**r **M**odels in **S**oftware **E**ngineering) dataset [2], which provided requirements in the domains of Master Scenario Events List Management (MSEL), Real Estate (REs) and Nursing Training Program Administration (NTPA). The number of artifacts in each PROMISE case study is smaller than those in the TSN, OPS and SPS because they do not contain use cases and business goals, due to the focus mainly on functional and non-functional requirements of this dataset. The reasons for choosing these case studies were twofold. First, they provided a wide range of artifacts in different domains. Second, since the

[2] The original requirements for the case studies can be found at http://goo.gl/gCUofM

PROMISE dataset had been used extensively for evaluations in the Requirements Engineering research community, using this dataset in our validation would potentially provide a solid comparison of our work to other works in the field.

As shown in table 3, we successfully categorized all 773 artifacts extracted from all six case studies into our defined artifact classes in GUIMeta's artifact layer, including different types of goals, and use case steps, conditions, data and nonfunctional constraints. Every artifact was classified into one and only one artifact class, which means there was no confusion regarding which class an artifact should belong to. The *C* rows in each case study section (i.e., TSN, OPS) in table III show the number of artifacts classified into each artifact class. Similarly, the *NC* rows present the number of artifacts that were not classifiable (the NC values were zero in all case studies, meaning all artifacts were successfully classified).

In addition, as indicated by the zero values in the *RM* – rule mismatched rows, we identified no mismatch between the artifacts' parameterized specifications and their corresponding specification rules (in table III). Moreover, there was only a small subset of the artifacts (70 out of 773) that were not parameterizable using our adopted set of semantic functions (showed by the *NP* rows). All of these results showed that GUIMeta is suitable for classifying, parameterizing and providing specification rules for goal-use case modeling.

We encountered some problems with parameterizing artifacts with temporal properties. i.e., a goal "*System logs a user off* ***after*** *15 minutes of being inactive*" or a constraint "*A locked account is locked* ***until*** *an admin unlocks it*" was not parameterizable in GUI-F. In addition, specifications with time expressions like *"The product shall be available for use* ***24 hours per day, 365 days per year***" or with "as" like *"The user marks the article* ***as 'favorite'****"* were also not supported. That was due to the lack of sufficient support for such properties in functional grammar.

To sum up, the evaluation results to-date showed that GUIMeta is suitable to use for modeling goal and use case models from requirements documents, which was indicated by the fact that all artifacts identified in each case study was successfully and uniquely classified into a GUIMeta's artifact class. Moreover, the fact that there were only 70 over 773 (9%) indicated that GUIMeta's specification rules are reliable to provide templates for writing and validating textual specifications in goal and use case integrated models. Our full evaluation data can be found at http://goo.gl/gCUofM.

B. GUIMeta Applications

GUIMeta has been used as a fundamental foundation for goal-use case integration modeling in our GUITARiST tool. The tool consists of two integrated components called GUEST [13] (**G**oal-**U**se case **E**xtraction **S**upporting **T**ool) and GUITAR [11, 12] (**G**oal-**U**se case **I**ntegration **T**ool for **A**nalysis of **R**equirements). In this section, we discuss these components and show their use of GUIMeta. GUITARiST and its evaluation results can be obtained from http://goo.gl/gCUofM.

Table II. GUIMeta Evaluation Results

		BG	FFG	FSG	NPG	NFG	NSG	NUCC	DC	UCS	UCC	**Tot.**
TSN	C	4	11	23	4	7	14	19	9	117	47	**255**
	NC	0										**0**
	RM	0										**0**
	NP	0	0	0	0	1	4	7	0	8	0	**21**
OPS	C	3	19	21	4	8	8	9	5	90	35	**200**
	NC	0										**0**
	RM	0										**0**
	NP	0	0	0	0	4	1	3	0	9	0	**17**
SPS	C	2	16	21	8	11	9	9	11	73	3	**163**
	NC	0										**0**
	RM	0										**0**
	NP	0	0	0	0	2	2	4	0	4	0	**12**
MSEL	C	0	0	15	6	2	5	5	0	0	0	**33**
	NC	0										**0**
	RM	0										**0**
	NP	0	0	0	1	0	5	1	0	0	0	**7**
REs	C	0	3	15	11	6	4	2	0	0	0	**41**
	NC	0										**0**
	RM	0										**0**
	NP	0	1	1	2	1	0	2	0	0	0	**7**
NTPA	C	0	8	39	10	9	5	0	10	0	0	**81**
	NC	0										**0**
	RM	0										**0**
	NP	0	1	0	3	1	1	0	0	0	0	**6**

C: classified **NC**: not classified **RM:** rule mismatched **NP**: not parameterizable **BG**: Business Goal **FFG**: Functional Feature Goal **FSG**: Functional Service Goal **NPG**: Non-functional Product Goal **NFG**: Non-functional Feature Goal **NSG**: Non-functional Service Goal **NUCC**: Non-functional Use case Constraint **DC**: Data Constraint **UCS**: Use Case Step **UCC:** Use Case Condition

1) GUEST: GUEST was developed to improve the goal and use case modeling process by providing automated support for the extraction of goals and use cases from textual requirements documents. Based on linguistic techniques and an extendable set of extraction rules, GUEST is able to automatically identify goals and use cases and their relationships from text, classify them and ensure they are properly specified (ensure grammatical correctness, rewrite goal such as *"users is capable of creating reviews"* to a recommended format of *"users shall be able to create reviews").* In this work, GUIMeta provides a conceptual foundation to determine what artifacts to be extracted, what categories they should be classified into and how they should be specified.

2) GUITAR: GUITAR provides an automated support for the detection and resolution of inconsistency, incompleteness and incorrectness in goal-use case integration models. GUIMeta plays a central role in GUITAR. First, from the definitions and classifications of artifacts and relationships in GUIMeta, we identify the classification of 3Cs problems to deal with. Second, it offers a way in which textual artifact specifications can be parameterized. Third, it offers a technique of understanding semantics of artifacts using ontology (i.e., by the ontology layer). GUITAR identifies syntactic problems based on the artifact and relationship rules defined in GUIMeta. For semantic problems, we developed an ontology-based technique to employ domain ontologies of knowledge and semantics to reason about goals and use cases.

In this technique, the parameterization of artifact is used to transform artifact specifications into Manchester OWL Syntax statements that then facilitate the automated detection of problems based on the Pellet reasoner [16].

C. Threat to validity

Some threats to validity from this evaluation exist. A threat to external validity was the representativeness of the selected case studies. To reduce this threat, we diversified the data by selecting case studies from different domains while obtaining a large number of requirements. Another external threat is that our validation was done using case studies only and thus insufficient for evaluating the use of GUIMeta in practical use. We therefore plan to conduct another evaluation with our industrial partners' projects in future work.

A threat to internal validity was the human factors in classifying (in RQ1), parameterizing artifacts and determining the matching of the parameterized specifications and specification rules (in RQ2). To alleviate this, we did the tasks carefully and reviewed them twice after they had been done. In addition, to further minimize the risk of manual artifact parameterization, we used our FGParam tool to generate the initial parameterizations, and then manually verified the results and made necessary corrections. The threat can be further reduced if two or more people with relevant knowledge and experiences were involved in the validation.

VI. Discussion

Apart from the use in our GUI-F framework, GUIMeta can also offer various benefits in goal and use case modeling.

Specification guidelines: developed based on the commonality of a large number of exemplar requirements, GUIMeta's specification rules (described in the form of boilerplates) can be used as guidelines on writing goal and use case specifications. This helps to ensure the properness of artifact specification right in the stages of elicitation and modeling to save analysis effort later on.

Improve the understanding of artifacts: GUIMeta offers a way to add semantics to textual artifact specifications by the use of functional grammar-based parameterization and ontology. For each specification, it is known what activity it is about, who takes such activity, what object is affected, by what means, in what manner and condition the activity is conducted and so on. Moreover, each word in those semantic functions can be mapped to an ontological concept whose semantic is known. This offers a way to understand artifacts in a model. Below we discuss the tasks that can be beneficial by such understanding. Importantly, our FGParam library can be used to facilitate the automation of these tasks.

- ***Redundancy and Overlap detection:*** artifacts' meaning can be compared to identify redundancies and overlaps.
- ***Detection of other defects:*** based on the captured semantics of artifacts, if detection rules for defects such as ambiguity, unverifiability are defined, artifacts can be analyzed to detect such problems. Our parameterization can offer a more powerful technique than keyword-based detection methods as it offers deeper semantic analysis.
- ***Generation of UML models:*** Some UML models such as sequence diagram and class diagram can be automatically generated based on the captured semantics of use case components (i.e., step, condition, extension).

The key limitation of GUIMeta is that it currently does not support the specifications of temporal properties such as *"until"*, *"unless"*, *"after x seconds"*. Specifications using "as" in the form like *"The DBMS may be located on the **same** machine **as the product**"* or *"The user marks the article **as 'favorite'**"* are also not parameterizable with GUIMeta. That is because these properties are not supported by functional grammar. In functional grammar, the discussed temporal properties are all captured as a general *"time"* semantic function, which does not sufficiently indicate the semantic difference between these properties. Moreover, functional grammar lacks support to handle the phrases such as *"same machine as product"* and *"as favorite"* in the examples. We plan to extend functional grammar with additional semantic functions to accommodate these cases. For instance, each temporal property would be associated to a unique semantic function so that their semantic roles can be fully differentiated.

VII. Related Work

Several approaches have been proposed to integrate goal and use case modeling, or use such integration to improve the requirements engineering process. Cockburn [4] suggests the use of goals to structure scenarios by connecting every action in a scenario to a goal assigned to an actor. He defines three abstraction levels of artifacts: *summary goals* are for system-level objectives, *user goals* specify user tasks in the system, and sub-functions refine user goals and can be steps in the use cases that operationalize user goals. However, non-functional goals are kept out of scope. Also, although writing guidelines are provided for use cases, they remain at a high level and do not specify how these artifacts are formed and validated.

Lee et al. [9] developed an approach called goal-driven use cases to structure use cases by goals, handle nonfunctional goals and analyze interactions between goals and use cases. They defined a number of artifacts and relationships. For instance, goals are classified in 3 facets: rigid vs. soft, system-specific vs. actor-specific and functional vs. nonfunctional. A use case that *achieves* an *original goal* (defined as the intersection of rigid, actor-specific and functional goals) is called an ori*ginal use case* while *extension use cases* are those *maintain* or *optimize* soft, system-specific or nonfunctional goals that *constrain* original goals. However, this work does not provide the internal structure of use cases and thus it is not clear how extension use cases and original use cases are different. Moreover, except for constrain relationships, other interactions between goals are not defined. This work also lacks guidelines on how each artifact should be specified.

Supakkul and Chung [18] proposed an approach to better integrate functional and nonfunctional requirements (NFR). In this work, use cases, which capture functional requirements, are associated to nonfunctional goals specified using the NFR framework [3]. To facilitate the integration, a set of NFR association points was proposed. For instance, an *actor*

association point is used to link an actor of a use case to his/her desired nonfunctional properties of the system while a *use case association point* is used to link a use case with its required nonfunctional constraints. The approach also provides a set of propagation rules to enable the traceability across a goal-use case model. The limitation of this work is its lack of support for functional goals and thus the interactions between functional goals, nonfunctional goals and their related use cases cannot be captured. Moreover, no specification rules/guidelines, and 3Cs problem analysis support are provided.

Rolland et al. [15] focus on guiding the elicitation of goals using scenarios based on the concept of requirement chunks. Each chunk contains a goal and a scenario that operationalizes the goal. There are three abstraction types of requirement chunks: *contextual*, *system interaction*, and *system internal level* and four abstraction types of goals: *business*, *design*, *service*, and *internal* goals. At each level of chunks, a goal is operationalized into a scenario and thus a new chunk is formed. A suitable scenario step is then selected as a goal and refined further. The authors also provide writing guidelines for goals and scenarios based on functional grammar. However, they are still very abstract and do not specifically define how each artifact is specified and verified. Also, no dedicated support for non-functional goals and 3Cs problem analysis are included. Kim et al. [8] extended [15] to provide guidelines for transferring goals and scenarios into use case models.

In the area of requirements abstraction, Gorschek and Wohlin [7] proposed a requirements abstraction model in which requirements are classified into 4 abstraction levels: product, feature, functional and component. However, nonfunctional requirements are left out of scope and no specification guidelines are provided for the classified artifacts.

A number of linguistic techniques have been proposed to help with the specifications of goals and use case (i.e., [5, 14]). However, they remain at a high level and are not specific to different types of goals and use case components. They thus do not offer the ability to reason about the 3Cs problems.

VIII. SUMMARY

In this paper, we introduced GUIMeta, a novel meta-model to improve the current practice of goal and use case integration modeling. GUIMeta supports more types of artifacts and relationships than existing goal-use case integrated approaches. GUIMeta consists of three layers. The artifact layer defines the classification of artifacts in a goal-use case model and their relationships. The specification layer provides specification rules of artifacts based on functional grammar. The ontology layer defines the ontology structure to allow semantics to be integrated into the entire model. We have established the correspondence between GUIMeta and existing goal-use case integration approaches to allow models in such approaches to be transformed into the format of GUIMeta. GUIMeta has been used in our GUI-F framework to provide automated extraction of goal-use case models from text and analysis of such models for inconsistency, incompleteness and incorrectness. Our promising evaluation results showed that GUIMeta meets all goal and use case integrated modeling requirements.

ACKNOWLEDGEMENT

The authors gratefully acknowledge support from the Victorian Government under the Victorian International Research Scholarships scheme, Swinburne University of Technology, and the Australian Research Council under Linkage Project LP130100201.

REFERENCES

[1] A.I. Anton, R.A. Carter, A. Dagnino, J.H. Dempster, and D.F. Siege, "Deriving goals from a use-case based requirements specification," Requirements Engineering, 6(1): p. 63-73. 2001.

[2] G. Boetticher, T. Menzies, and T. Ostrand, "The PROMISE Repository of Empirical Software Engineering Data," 2007.

[3] L. Chung, B. Nixon, E. Yu, and J. Mylopoulos, "Non-functional Requirements," Software Engineering. 2000.

[4] A. Cockburn, "Structuring Use Cases with Goals," 1997.

[5] A. Cockburn, "Basic use case template," Humans and Technology, Technical Report, 96. 1998.

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

[7] T. Gorschek and C. Wohlin, "Requirements abstraction model," Requirements Engineering, 11(1): p. 79-101. 2006.

[8] J. Kim, S. Park, and V. Sugumaran, "Improving use case driven analysis using goal and scenario authoring: A linguistics-based approach," Data & Knowledge Engineering, 58(1): p. 21-46. 2006.

[9] J. Lee, N.-L. Xue, and J.-Y. Kuo, "Structuring requirement specifications with goals," Information and Software Technology, 43(2): p. 121-135. 2001.

[10] L. Liu and E. Yu, "Designing information systems in social context: a goal and scenario modeling approach," Information systems, 29(2): p. 187-203. 2004.

[11] T.H. Nguyen, J. Grundy, and M. Almorsy. "GUITAR: An ontology-based automated requirements analysis tool," Requirements Engineering Conference (RE), 2014.

[12] T.H. Nguyen, J. Grundy, and M. Almorsy, "Ontology-based automated support for goal–use case model analysis," Software Quality Journal, 23(3). 2015.

[13] T.H. Nguyen, J. Grundy, and M. Almorsy, "Rule-Based Extraction of Goal-Use Case Models from Text," 10th Joint Meeting of the European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2015.

[14] N. Prat, "Goal formalization and classification for requirements engineering, fifteen years later," Research Challenges in Information Science (RCIS), 2013.

[15] C. Rolland, C. Souveyet, and C.B. Achour, "Guiding goal modeling using scenarios," Software Engineering, IEEE Transactions on, 24(12): p. 1055-1071. 1998.

[16] E. Sirin, B. Parsia, B.C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical owl-dl reasoner," Web Semantics: science, services and agents on the WWW, 5(2): p. 51-53. 2007.

[17] I. Sommerville and G. Kotonya, "Requirements engineering: processes and techniques," John Wiley & Sons, Inc. 1998.

[18] S. Supakkul and L. Chung, "Integrating FRs and NFRs: A use case and goal driven approach," framework, 6: p. 7. 2005.

[19] A. Van Lamsweerde, "Goal-oriented requirements engineering: A guided tour," Requirements Engineering, 2001.