Inf Syst Front (2012) 14:1019–1032
DOI 10.1007/s10796-011-9314-0

Applying ontology-based blog to detect information system post-development change requests conflicts

Chi-Lun Liu · Heng-Li Yang

Published online: 16 July 2011

Abstract Post-development change requests are user requirements for information systems changes after development. Conflicts might occur as contradictive or inconsistent relationships between requests and existing system design. Detecting conflicts in post-developmenet change requests is an important task during requests management processes. To address this topic, this article proposes an ontology-based blog for automatically discovering conflicts in the extended use-case models of requests from users. This study proposed an information system maintenance process. The proposed approach applies ontologies to represent domain knowledge. A set of rules are used to detect conflicts. This study developed a prototype and invited two companies to evaluate it. Usage feedback opinions about ontology-based blog from two companies indicated the usefulness. The ontology-based blog is a relatively new approach which bridge requirements blogs with a formal and machine interpretable representational model. The automatic conflicts detection capability of the ontology-based blog can reduce the labor cost in requirements analysis phase.

C.-L. Liu
Department of Information and Electronic Commerce,
Kainan University,
1, Kainan Road,
Luzhu Shiang, Taoyuan County, 338, Taiwan
e-mail: tonyliu@mail.knu.edu.tw

H.-L. Yang (✉)
Department of Management Information Systems,
National Cheng-Chi University,
64, Section 2, Chihnan Road, Mucha Dist., 116,
Taipei, Taiwan
e-mail: yanh@nccu.edu.tw

Keywords Information system maintenance
Ontology-based blog · Requirements conflict detection · Ontologies

1 Introduction

Managing user requirements can be a complex and difficult task (Robinson and Pawlowski 1999). Understanding what the information system is supposed to do is necessary before not only developing new systems but also maintaining existent systems (Sommerville 2005; Kang and Chiang 2006). User requirements for system maintenance are called post-development change requests. Unlike new system requirements can be collected during a comparative short period, post-development change requests usually come randomly and respectively during a long period after system deployment. Information systems department should review the request content, prioritize its sequence in a queue, and estimate its cost formally (Abran and Nguyenkim 1993; April et al. 2005).

Use-case approach has rapidly emerged as a standard for representing user requirements (Madallli and Suman 2008; Dobing and Parsons 2000; Fatolahi and Shams 2006). A typical large and complicated system would have a lot of use cases. On the other hand, requirements inconsistency is a major problem in requirements engineering phase (Paulo et al. 2007). Conflicting requirements cannot be designed and implemented by information systems departments. Stakeholders should provide their opinions in the conflict resolution process (Gervasi and Zowghi 2005). Hence, detecting use-case conflicts between a post-development change request and the overall existing systems is important in request management process. But conflict detection might be a time-consuming and costly task when use cases are numerous.

Modeling use cases involves domain knowledge. Ontologies are shared conceptualization for expressing explicit knowledge (Gruninger and Lee 2002). Ontologies also include a representation vocabulary that describes concepts and relationships between concepts, e.g., kind relationship and composition relationship (Chandrasekaran et al. 1999). Ontologies can be used to represent software design specification (Eden 2002). Applying ontologies as domain knowledge can be beneficial because semantic processing of user requirements by computer is possible (Kaiya and Saeki 2006).

This study uses extended use cases to handle functional and nonfunctional requirements. For example, a functional requirement about data storage is "bill payment can store credit card number". A nonfunctional requirement example about security is "unregistered user cannot use purchase function".

This study proposes an ontology-based blog to detect the conflicts of extended use-case models between requests and existing system. Blog is a widespread communication tool nowadays. The ontology-based blog includes a set of ontologies and rules. The terms in ontologies can be used to constitute an extended use-case model for representing a post-development change request. The proposed rules can detect contradictions according to the semantic relations between request and existing system concepts. The rules may be triggered when a stakeholder use ontology-based blog to submit a new function suggestion in the post-development change requests management process.

The article is organized as follows. Section 2 discusses related literature about post-development change requests management, use cases, and ontology-based conflict detection. Post-development change requests management process, which is supported by the proposed tool for detecting request conflicts, is presented in Section 3. The proposed use-case approach, ontologies, and rules are shown in Sections 4 and 5. Finally, Section 6 presents the ontology-based blog prototype tool and usage feedback opinions.

2 Literature review

Formulating systematic requirement management process is an imperative topic for researchers (Jiao and Chen 2006). A request management process in systems maintenance should involve a supportive tool, standardized user request form, request's priority, cost, and up-to-date feedback (April et al. 2005). However, so far, there exists few request management process about systems maintenance. Problem Management Process (Kajko-Mattsson 2002) is a request management process for corrective maintenance. This process involves three stages: problem reporting, problem analysis, and problem resolution. Enhancive Maintenance Model (EMM) (Kajko-Mattsson and Bosu 2006) is a request management process for enhancive maintenance based on the study of three organizations. The process has four phases: enhancement submission, enhancement analysis, decision making, and contract signing. Several guidelines are provided in each phases. Supportive tool and conflict resolution do not explicitly addressed in EMM.

Use-case approaches are valuable and popular in representing user requirements. Original use-case approach includes three basic concepts: *actor*, *use case*, and *system* (Conallen 1999; Booch et al. 1998). *Actor* is a role using the system. *Use case* is depicted as an oval and presents a part of the system. *System* indicates the software boundary outside of several ovals of use cases. These three parts are important points in user requirements. However, there are three problems in original use-case approach for representing requirements. Nonfunctional parts in requirements are not systematically classified. Goal-driven use cases (Lee and Xue 1999) indicate that goals as nonfunctional parts should be explicitly presented in extended use cases. Also, in order to detect conflicts inside the oval, *use case* should be cut into several small pieces. Besides, prohibition (e.g., unregistered user cannot use purchase function) and unnecessary design (e.g., member does not need to use pop-up advertisement) are not considered in original use-case approach.

Although computer-aided software engineering (CASE) tools have successfully support modeling and code generation, CASE tools have been less successful in supporting requirements analysis (Robinson et al. 2003). On the other hand, there are a lot of new requirements due to the changing business environment (Sommerville 2005). Hence automatic requirements conflict detection is a critical issue and has been noticed.

There are several studies focusing on conflict detection issue. CARL tool (Gervasi and Zowghi 2005) checks logical contradictions (e.g., a car can drive on bus lane and a car cannot drive on bus lane) in natural language requirements. AGG tool (Hausmann et al. 2002) detects differences between two concurrent versions of the requirement specification including class, activity, and use case diagram. For example, in two class diagram V1 and V2, attribute Y of class X=1000 in version V1. Meanwhile, attribute Y of class X=1040 in version V2. In this case, the two versions are different and result in conflict.

Ontology-based conflict detection is an emerging topic. Lightweight semantic process approach (LSPA) (Kaiya and Saeki 2005) is an ontology-based method and can detect inconsistency in requirements. LSPA define inconsistency as a tradeoff between two system features. For example, if a user raises an issue of system quality about time efficiency, the system designer may need to sacrifice the complex function which is required by other users.

The above related works are fundaments for more in-depth study in the conflict detection. The value of CARL

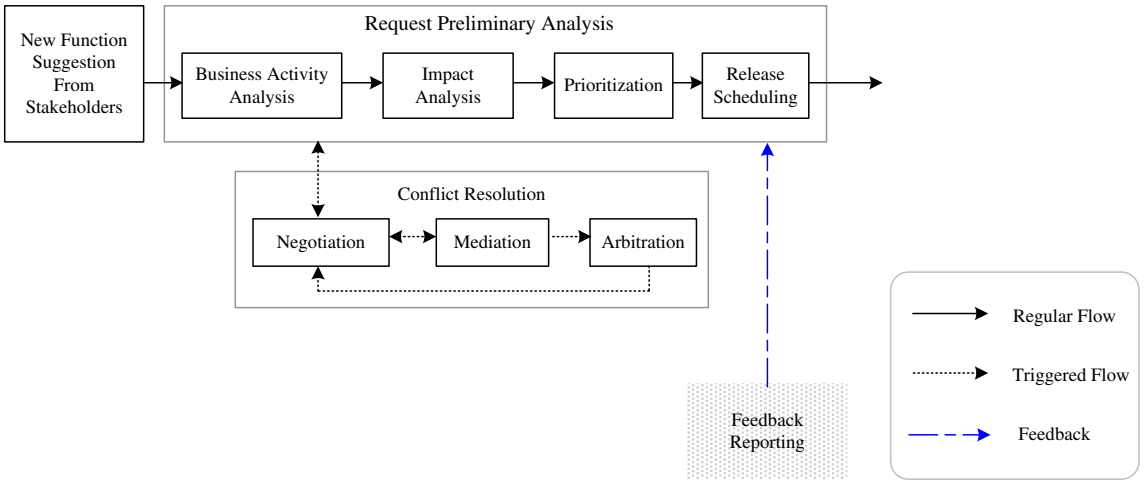

Fig. 1 Process model for managing post-development change requests

tool (Gervasi and Zowghi 2005) is disclosing contradictions between verbs. AGG tool (Hausmann et al. 2002) points out that Unified modeling language (UML) is a good starting point for proposing requirements meta-model. LSPA (Kaiya and Saeki 2005) indicates ontologies as a semantic knowledge representation are a key element for requirements conflicts detection. However, these ideas are separated and do not be integrated. Therefore, it is necessary to integrate the above ideas for enhancing the conflict detection mechanism.

3 Post-development change requests management process

This research proposes a management process for handling post-development change requests from stakeholders at the beginning of system maintenance works. The distinguishing feature of this process model is to avoid that most conflicts are arbitrarily resolved by the programmer during the implementation of the post-development change request. This process model hopes to facilitate performing preliminary and widespread discussions of the conflicts.

To deal with new function suggestions, the process model includes three parts (Fig. 1): *request preliminary analysis*, *conflict resolution*, and *feedback reporting*. *Request preliminary analysis* involves four phase: *business activity analysis*, *impact analysis*, *prioritization* of request implementation, and *release scheduling* of new version software. In *business activity analysis* phase, the system analysis should consider what the request touch on the business level, such as the role who uses the system, the activity which is supported by the system, and the goal and assumption which are considered in system design. *Impact analysis* is a process to evaluate the scope of the change in software, resource needed to accomplish the change, and the benefit accompanies with the change. *Prioritization* adjusts priorities of all requests in a waiting queue. *Release scheduling* concerns with placing the request in a version release schedule and modifying other requests' release dates when necessary.

Conflict resolution, includes *negotiation*, *mediation*, and *arbitration phase*, is a bottom-up approach from basic level to administrative level in enterprises. *Negotiation phase* encourages the disputants to negotiate the conflict issue. If consensus cannot be achieved in negotiation phase, meditation phase is triggered to elect an opinion leader to introduce new professional opinions. If these two prior phases are neither workable, the higher-level manager or decision committee must provide a judgment to break deadlock in *arbitration phase*.

Feedback reporting considers receiving follow-up information during implementation of the post-development change request and usage of new version software. Feedback information can include release schedule delay, resource demand raise, and technical and usage constraints.

Fig. 2 Proposed extended use-case approach

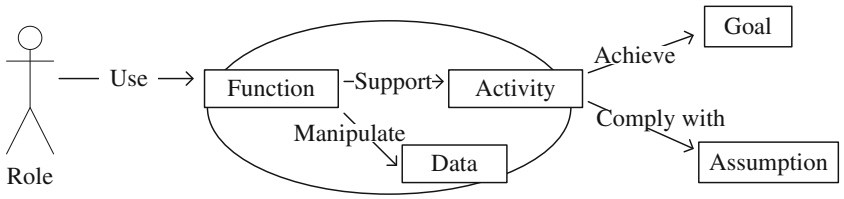

Table 1 Evolvable noun ontologies

Noun ontology type	Definition	Example terms
Role ontology	Different sizes and levels of organization units.	Net-surfer, registered member, unregistered visitor
Function ontology	Parts constituting software system.	Purchase function, shopping cart, bill payment, member join function
Activity ontology	Valuable tasks for stakeholders	Putting commodities into shopping cart, paying the money to the company on-line
Data ontology	Messages which are handled in software system	Commodity item name, amount, receiver name, address, credit card number
Goal ontology	Interests which explain why activities should exist and be necessary	Minimizing bill payment phases, promoting related products, increasing security level, providing acceptable performance
Assumption ontology	Constraints which guide how activity works and what kinds of functions is necessary	Emphasizing global markets, maintenance cost regulation

The result of request preliminary analysis may be modified according to feedback information.

In the context of post-development change requests management process, this study proposes a blog-based prototype tool in supporting *new function suggestion from stakeholders*. The conflict detection mechanism of the tool can show the conflict situation between the submitted request and existing system design. If the tool shows that the submitted request has a conflict, the progress of the process should be toward *conflict resolution*.

4 Extended use-case approach and ontologies

To address the conflict detection problem, this study proposes a meta-model which is derived from use case diagrams. The proposed meta-model includes six noun concepts shown in Fig. 2. *Role* is the actor in use case diagrams. The oval describing a part of the system is split into three concepts: *function*, *activity*, and *data*. *Function* means the system boundary. *Activity* indicates which work is supported in the function. *Data* is the message which are inputted, outputted, and stored by the function. Besides, *goal* and *assumption* concepts are added in the proposed meta-model because the enterprise interests and backgrounds are often presented unavoidably in user requirements.

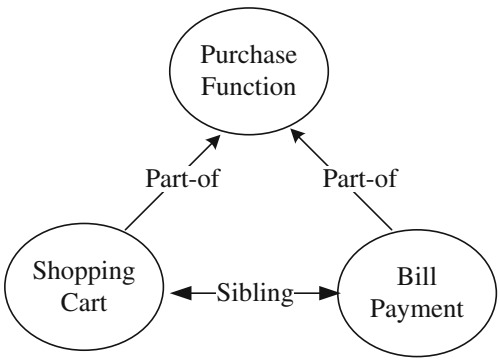

Fig. 3 Example of function ontology

The proposed meta-model also includes five verb concepts to present the relationships between noun concepts in Fig. 2. '*Use'* concept presents that the role uses, can not use, or do not need to use the function. '*Support*' concept reveals that whether the function support in the activity or not. '*Manipulate*' concept describes that operation relationship between the function and the data. The concepts of '*achieve*' and '*comply with*' show which background consideration should be or has been examined in system design.

Stakeholders can use the shared vocabulary in ontologies to fill in the proposed meta-model for the requirement model presentation. In other words, ontology is like dictionary. Meta-model is like a form. Stakeholders can use ontological terms (in dictionary) to fill in a meta-model (form) for articulating a requirement. In this study, noun ontologies are unfixed and can be added and deleted depending on information system's situation in practice. Six noun ontologies are explained in Table 1.

Table 2 Fixed verb ontologies

Verb ontology type	Including verb concepts
'Use' ontology	Use, cannot use, do not need to use
'Support' ontology	Support, cannot support, do not need to support
'Manipulate' ontology	Input, cannot input, do not need to input; store, cannot store, do not need to store; output, cannot output, do not need to output
'Achieve' ontology	Achieve, cannot achieve, do not need to achieve
'Comply with' ontology	Comply with, cannot comply with, do not need to comply with

Table 3 Conflict reasons

Conflict reason	Description	Relevant rule no.
Access control contradiction	Whether a role should use a function to do something or not	1, 8, 10
Data manipulation contradiction	Whether a data should be manipulated by a function or not	2, 11, 14
Support activity contradiction	Whether a activity should be supported by a function or not	3, 9, 12
Goal achievement contradiction	Whether a goal should be achieved by a activity or not	4, 13
Assumption accordance contradiction	Whether a assumption should be complied by a activity or not	5, 15
Goal diversity	Different goals should be achieved by a activity	6, 16
Possible assumption violation	A function supporting in a activity must comply with a regulative assumption	7, 17

An example of function ontology is depicted in Fig. 3. On-line retailing website usually includes *purchase function* supporting in shopping. Purchase function includes two functions: *shopping cart* and *bill payment*. Therefore, there is a part-of relationship between *purchase function* and *shopping cart*. There is also a part-of relationship between *purchase function* and *bill payment*. In addition, a sibling relation exists between *shopping cart* and *bill payment*.

On the other hand, verb ontologies are fixed and can not be modified. There are three kinds of verbs in each verb ontology: V, Cannot V, and Do not need to V. V is a variable to present a verb concept. "Cannot V" can be used in a user requirement which wants to prohibit something. And "Do not need to V" reveals can be used in a user requirement which wants to neglect something. The verb concepts which are included in five verb ontologies are shown in Table 2.

5 Proposed conflicts detection rules

This study proposes 18 rules for detecting conflicts between user post-development change requests and system designs presented by the extended use-case approach. So far, this study has classified seven kinds of reasons that can cause conflicts. These reasons include five types of contradictions, one type of goal diversity, and one type of possible assumption violation. These reasons' explanations and related rule number are shown in Table 3.

The 18 rules are designed to detect two forms of requests. The first form of request models is named *2N-1V request* which includes two noun concepts and one verb concept. The second form of request models is named 3N-2V request which includes three noun concepts and two verb concepts. All these rules can be expressed as "If (condition), then conflict may occur".

The conflict condition of *2N-1V request* includes new request model, existing system model, and their relationships (Fig. 4). *2N-1V request* includes *noun E*, *verb G*, and *noun I*. The system model with corresponding *2N-1V request* includes *noun F*, *verb H*, and *noun J*. The relationship between request model and system model includes *R1*, *R2*, *R3*, and *R4*. The conflicts will occur if concepts *E*, *F*, *G*, *H*, *I*, and *J* conform to the specific conditions and R1, R2, R3, and R4 fit in with specific relationships between the request and system model.

There are seven rules for detecting the conflicts in 2N-1V request (Table 4). Rule 1 to rule 5 are designed to detect contradictions if an antonym verb exists. Rule 1 in Table 4 can be explained as "If E and F are role concepts, G and H are usage concepts, I and J are function concepts, R1 is a equal, composition, or kind relationship between E and F, R2 is antonym relationship between G and H, and R3 is equal, composition, or kind relationship between I and J, then the conflict occurs." For example, the new request model is "Net-surfer use shopping cart". One of the existing system models is "Unregistered visitor cannot use shopping cart". The ontologies indicate kind relationship between net-surfer and unregistered visitor. According to rule 1, the above new request model conflicts with existing system model. Rule 2 to rule 5 have the similar contradiction pattern comparing to rule 1, but they are designed to cope with different concept types, such as data, goal, and assumption concepts. For example, a new request is "Bill payment cannot store credit card number" for preventing hackers from stealing credit card numbers

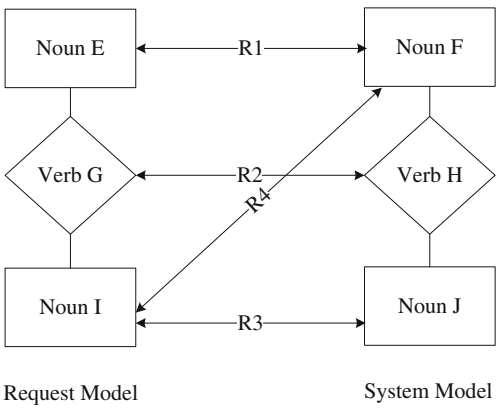

Fig. 4 2N-1V conflict condition meta-model

Table 4 Proposed conflict condition for 2N-1V request

Rule No.	E, F	G, H	I, J	R1	R2	R3	R4
1	C_R, C_R	C_U, C_U	C_F, C_F	ER/CKR	AR	ER/CKR	Null
2	C_F, C_F	C_M, C_M	C_D, C_D				
3		C_S, C_S	C_{Act}, C_{Act}				
4	C_{Act}, C_{Act}	C_{Ach}, C_{Ach}	C_G, C_G				
5		C_C, C_C	C_{As}, C_{As}				
6		Achieve, Achieve	C_G, C_G		ER	OR	
7	C_F, C_{Act}	Support, Comply with	C_{Act}, C_{As}	Null			ER/CKR

C_R Role Concept, C_F Function Concept, C_{Act} Activity Concept, C_D Data Concept, C_G Goal Concept, C_{As} Assumption Concept, C_U Usage Concept, C_S Support Concept, C_M Manipulation Concept, C_{Ach} Achievement Concept, C_C Compliance Concept, *ER* Equal Relationship, *CKR* Composition or Kind Relationship, *OR* Opposing Relationship, *AR* Antonym Relationship, *'/'* Or

in database. But the existing design is "Bill payment store credit card number" so that customers don't need to key in their credit card number every time. Hence rule 2 is triggered.

Rule 6 are designed to detect that activities achieve opposing goals. Rule 6 in Table 4 can be explained as "If E and F are activity concepts, G and H equal 'achieve', I and J are goal concepts, R1 is a equal, composition, or kind relationship, R2 is equal relationship, and R3 is opposing relationship between goal I and goal J, then the conflict occurs.". For example, a request is "Paying the money to the company on-line achieve minimizing bill payment phases". And the relevant original system design principle is "Paying the money to the company on-line achieve promoting related products". In other words, the original design is to add several phases to prompt other commodities when customers pay the bill. The two goals of "minimizing bill payment phases" and adding the phases for "promoting related products" are opposing opinions. Therefore the conflict occurs in this request.

Rule 7 are designed to detect regulative assumptions when a function supports a new activity. For example, the request is "Member join function support paying the money to the company on-line" to give the membership fee. And the original system design is "Paying the money to the company on-line comply with emphasizing global markets" because the target market of this company is overseas customers. When the information systems department implements this request, the department should pay attention that this new function should allow overseas payment conveniently.

The conflict condition of *3N-2V request* also includes new request model, existing system model, and their relationships (Fig. 5) and provides more complicated concepts than *2N-1V request* to describe maintenance requirements. *3N-2V request* includes *noun E*, *verb G*, *noun I*, *verb K*, and *noun M*. The system model with corresponding *3N-2V request* includes *noun F*, *verb H*, *noun J*, *verb L*, *noun N*, *verb O*, and *noun P*. The relationship between request model and system model includes *R1*, *R2*, *R3*, *R4*, and *R5*. The conflicts will occur if the specific conditions exist in Fig. 5.

There are 11 rules for detecting conflict in *3N-2V request*. Rule 8 to rule 15 in Table 5 detect contradictions if an antonym verb exists. For example, a new request is "Net-surfer using shopping cart support putting commodities into shopping cart" and the original system design is "Unregistered visitor using shopping cart cannot support putting commodities into shopping cart". Unregistered visitor is a kind of net-surfer. According to rule 9, the conflict occurs in this request.

Rule 16 in Table 5 is proposed to detect goal divergence when a function supports an activity. Rule 17 in Table 5 detects which assumption regulates a request involving a function support an activity.

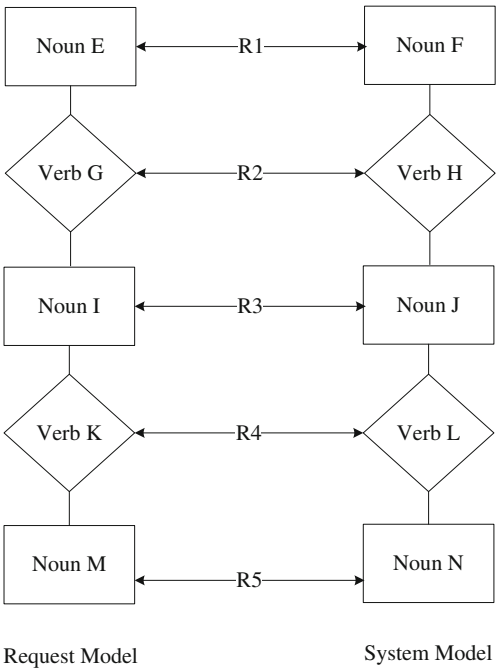

Fig. 5 3N-2V conflict condition meta-model

Table 5 Proposed conflict condition for 3N-2V request

Rule no.	E, F	G, H	I, J	K, L	M, N	R1	R2	R3	R4	R5
8	C_R, C_R	C_U, C_U	C_F, C_F	Support, support	C_{Act}, C_{Act}	ER/CKR	AR	ER/CKR	ER	ER/CKR
9		Use, use		C_S, C_S			ER		AR	
10		C_U, C_U		Input, input/store, store/ output, output	C_D, C_D		AR		ER	
11		Use, use		C_M, C_M			ER		AR	
12	C_F, C_F	C_S, C_S	C_{Act}, C_{Act}	Achieve, achieve	C_G, C_G		AR		ER	
13		Support, support		C_{Ach}, C_{Ach}			ER		AR	
14		C_S, C_S		Comply with, comply with	C_{As}, C_{As}		AR		ER	
15		Support, support		C_C, C_C			ER		AR	
16				Achieve, achieve	C_G, C_G				ER	OR
17				Null, comply with	Null, C_{As}				Null	

C_R Role Concept, C_F Function Concept, C_{Act} Activity Concept, C_D Data Concept, C_G Goal Concept, C_{As} Assumption Concept, C_U Usage Concept, C_S Support Concept, C_M Manipulation Concept, C_{Ach} Achievement Concept, C_C Compliance Concept, *ER* Equal Relationship, *CKR* Composition or Kind Relationship, *OR* Opposing Relationship, *AR* Antonym Relationship, *'/'* Or

Besides the above *2N-1V* and *3N-2V request*, *4N-3V request* can exist. However, few 4N-3V requests appear and the rules for detecting the conflicts in *4N-3V request* are similar to above rules. Rule 1~17 can be used to detect 4N-3V request. For example, in Fig. 6, "Member use checkout function to support paying the money to the company on-line to achieve minimizing bill payment phases" conflicts with "Member use checkout function to support paying the money to the company on-line to achieve promoting related products".. This conflict can be detected by Rule 6 if we only consider two nouns and one verb ("Paying the money to the company on-line achieve minimizing bill payment phases" and "Paying the money to the company on-line achieve promoting related products"). Therefore the conflicts in *4N-3V request* are not described additionally here.

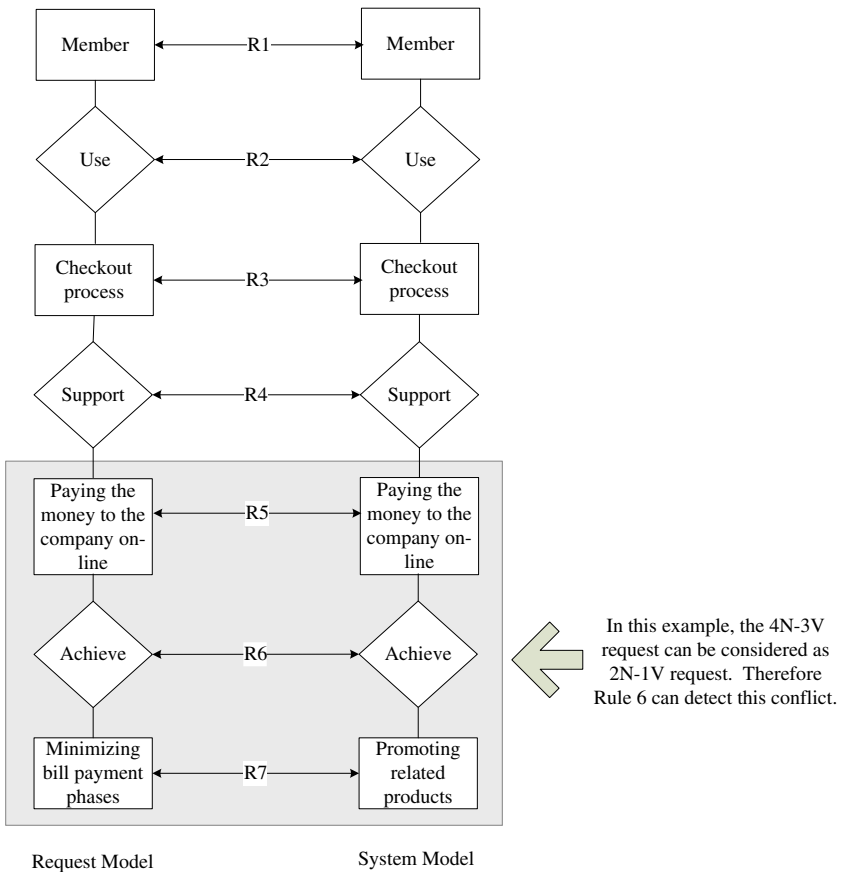

Fig. 6 4N-3V conflict detection example

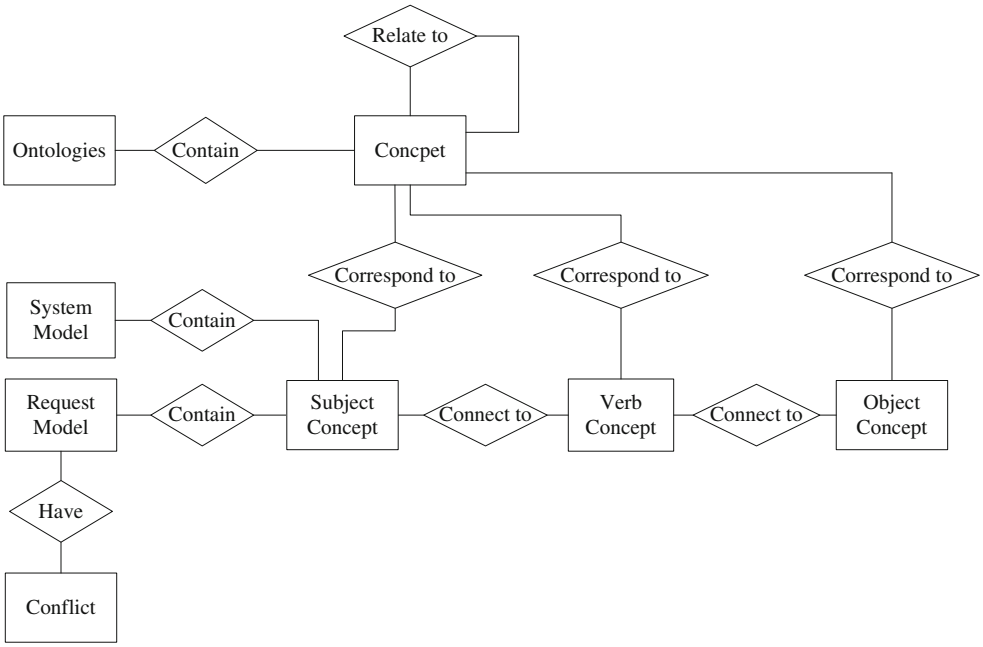

Fig. 7 Entity-relationship model of the database in the tool

6 Ontology-based blog prototype tool

A prototype tool is implemented in this study. The open-source blog package software called Community Server, which is developed by Telligent Cooperation, is chosen as the base to be modified in this study. This prototype tool is developed in C# language and Structured Query Language (SQL). The database administration system of this prototype is Microsoft SQL Server 2005. The original Community Server only provides the English user interface. This study added another version of user interface in Traditional Chinese.

In the database of the blog, several new entities and relationships are added to store ontologies, system model, request models, and conflicts (Fig. 7). Ontologies contain concepts, such as role, function, activity, and data. Relationships between concepts, such as is-a, kind-of, and antonym, are also included in ontologies. Both system model and request model contain subject, verb, and object concepts. These concepts in system model and request model can be found in ontologies or can be added into ontologies. The automatic detection results of the request conflicts can be stored in the database.

This study has added four modules into the original blog package software. These modules are *ontologies manager*, *system model manager*, *request model maker*, and *conflict detector*. *Ontologies manager* and *system model manager* are added in the back-end of the blog-based tool for modeling domain knowledge and existing system design. *Request model maker* and *conflict detector* are added in the front-end of the blog-based tool for making request models and detecting conflicts. The component, sequence, activity, and collaboration diagrams of the four modules are shown in Figs. 8, 9, 10, 11.

Information systems department can use *ontologies manager* to add, modify, or delete the concepts or the relationships of domain knowledge. The concepts include roles, functions, data, activities, goals, and assumptions. The relationships include *is-a, part-of*, and *anonym*. For example, a part of ontologies can be "Unregistered visitor is a kind of Net-Surfer". Besides, this module can be used to

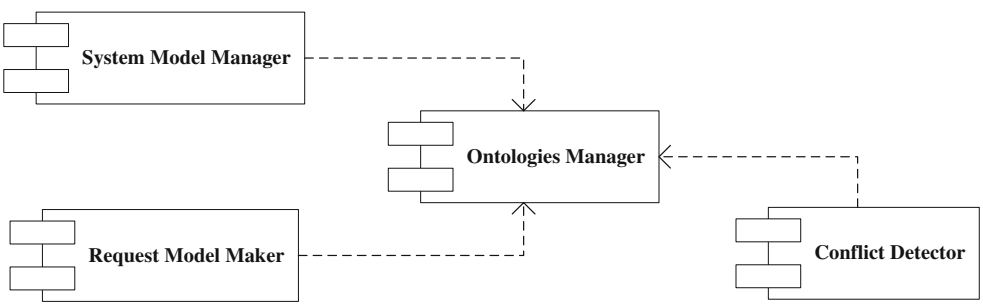

Fig. 8 Component diagram of the prototype

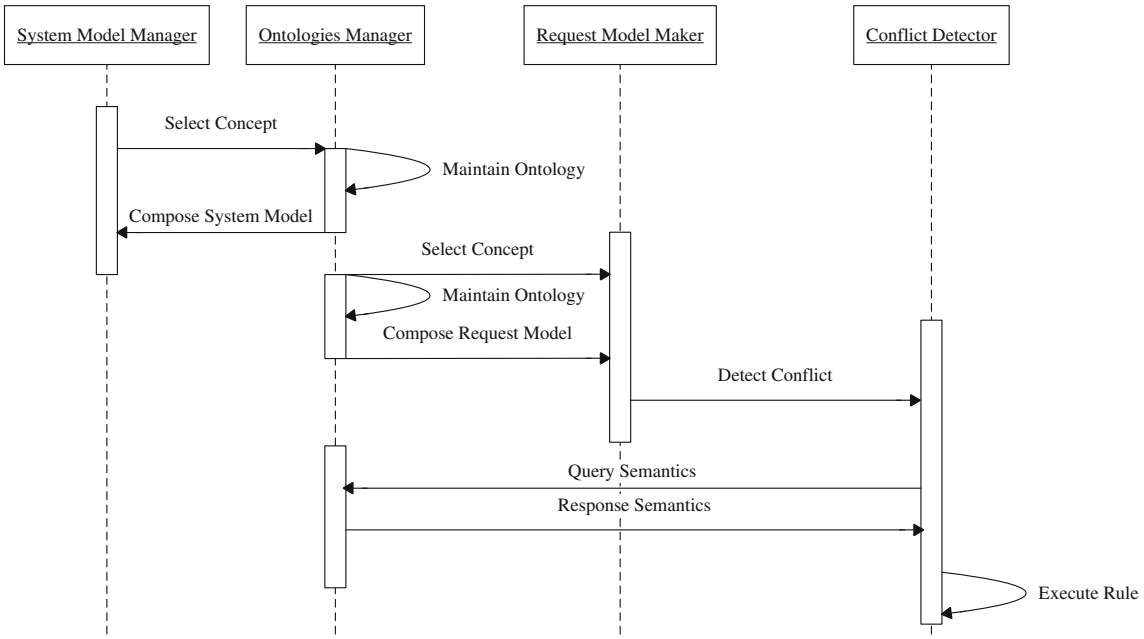

Fig. 9 Sequence diagram of the prototype

put new terms, which are proposed by users, into existing ontologies.

Information systems department can use *system model manager* to select concepts from ontologies for composing model of existing systems design. System models are made according to the proposed extended use-case approach in Fig. 2. For example, a system model can be "Unregistered visitor cannot use shopping cart". If necessary, s*ystem model manager* can be also used to modify and delete system models.

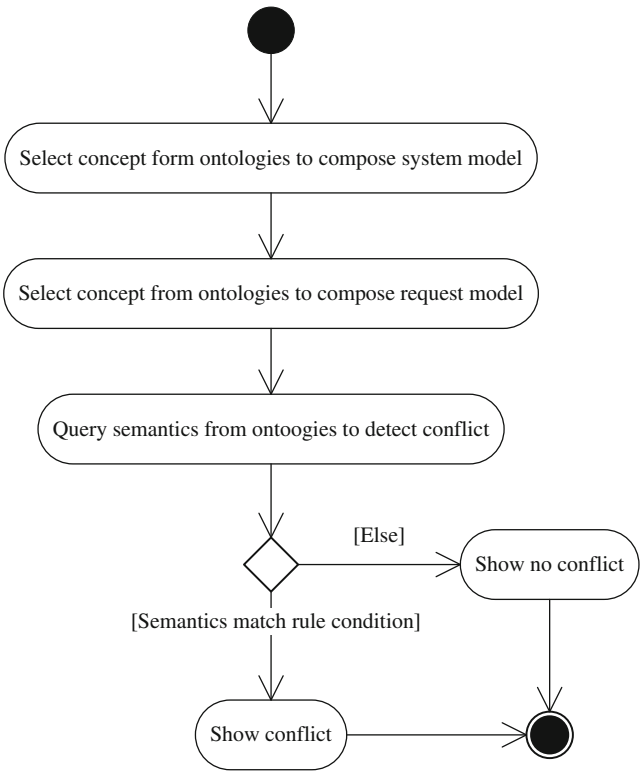

Fig. 10 Activity diagram of the prototype

Stakeholders can use *request model maker* and blog comment function to post a post-development change request. A screenshot of blog comment function are presented in Fig. 12. Stakeholders can join as members, log in the blog, enter personal website Uniform Resource Locator (URL) optionally, click the hyperlink of request model maker to make a request model, enter footnote to explain the request model, and finally submit this request. The meaning of the Chinese sentence in the request footnote of Fig. 12 is as follows "I think the persons, who do not want to enter burdensome and complete personal information, also can use purchase function if they enter few necessary information".

A screenshot of *request model maker* is shown in Fig. 13. The interface in Fig. 13 has three areas: *selected items*, *request model*, and *templates area*. There are five types of templates in *templates area*. Stakeholders can use an appropriate template to choose two nouns of the template into selected items area and then choose a verb of the template to compose a request model which is shown in *request model area*. For example, a user wants to express the request: "Unregistered visitors also can use purchase function in the shopping website". Firstly, he considers template 1 is appropriate for expressing his request. Hence he click role button and browse the role items list which shows role concepts in ontologies. He selects "Unregistered

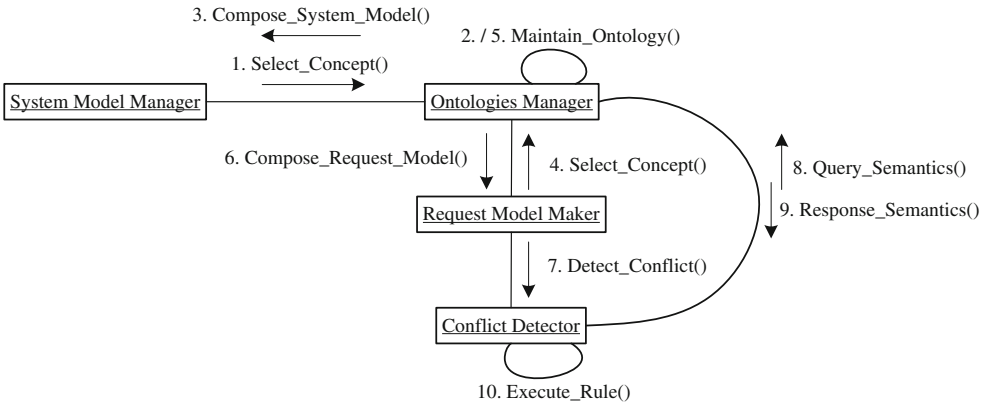

Fig. 11 Collaboration diagram of the prototype

Visitor" item, which is a child of "Net-Surfer", and this item is shown in *selected items area* in Fig. 13. Then he selects "Purchase Function" item and this item is shown in *selected items area*. Finally, he select "use" verb item and the request model (Unregistered visitor use purchase function) is presented in *request model area*.

After a stakeholder submits a post-development change request, the *conflict detector* module is triggered for detect conflicts between request model and system model. The above 18 rules were implemented by C# programming code in this module. Consequently, the request model and the conflict detection result are shown in the blog-based tool. For example, the request model in Fig. 14 can be translated as "Unregistered visitor use purchase function". An existing system model is "Unregistered visitor cannot use purchase function". According to rule 1, the conflict occurs because two verbs are antonyms. Therefore, the result of conflict situation in Fig. 14 can be translated as "This request has conflict because unregistered visitor cannot use purchase function in current system design".

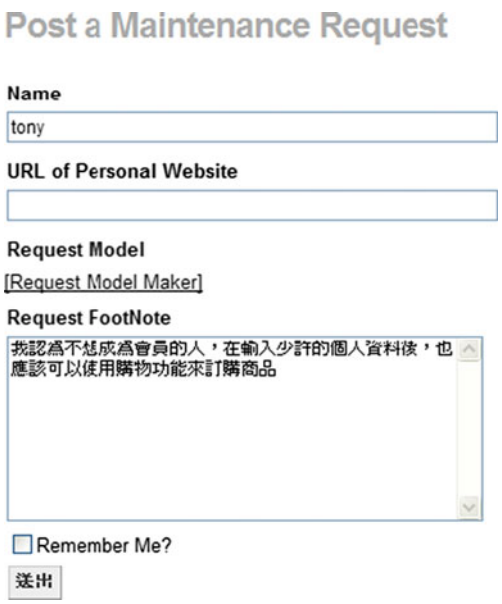

Fig. 12 Posting a request in the tool

7 Usage feedback opinions

Two companies in Taiwan adopted the post-development change requests management process and tried the blog-based prototype tool. The online newspaper company has 1,600 employees and the television station company has 700 employees. The two companies put the hyperlink of the blog-based prototype tool on their newspaper and shopping website for promoting post-development change requests collection. After 2 weeks, several requests were posted in the blog-based tool. The amounts of requirements and conflicts are showed in Table 6. The two companies tried to apply the post-development change requests management process to handle these requests. Their website directors and information systems department directors were then interviewed.

In general, two companies confirmed that the benefits of the post-development change requests management process and the blog-based prototype tool. The benefit of the process is offering disciplined steps for guiding requirement analysis. The benefits of the tool are two folds. One of the benefits is offering an effective and direct channel for understanding requirements from website users. The second benefit of the tool is reducing the labor cost of conflict analysis when companies face over hundred requirements.

The website directors and the information systems department directors also provided several recommendations for improving the blog-based tool. These recommendations are summarized in Table 6.

The above recommendations are detailed and discussed as follows. (1) Indicating problems: Two interviewees believed that stakeholders are often unable to propose request models when a new topic is emerging. Stakeholders may be only capable to indicate where the problems are. Therefore, the blog-based tool should allow stakeholders to select the noun concepts for pointing out which parts of the information system should be improved. (2) Rating a request: Two interviewees suggested how many customers who support or oppose the requests should be surveyed. The blog-based tool should provide rating function for

Fig. 13 Using templates for making request model in the tool

evaluating satisfactions of the requests by customers. (3) Showing implementation status: An information system department director deemed that the company should declare whether the request is implemented or not publicly. Therefore, the blog-based tool should shows the status of the request, such as pending, decided to implement, rejected, implementing, and released. (4) Covering content request: For newspaper website, content is king. The blog-based tool can be modified and applied to collect user requirements about website content. (5) Revealing activity sequences: The proposed extended use-case approach does not cover activity process. However, process, which contains a sequence of activity, is usually an important concept. For example, online payment process and join member process are common in e-commerce website. Therefore, the request model of the blog-based tool should cover the process model. (6) Allowing multiple ontologies: Different departments and groups always have different domain knowledge and jargons. Therefore the blog-based tool should allow the companies to build multiple ontologies and their interconnections between similar concepts in different ontologies. (7) Explaining each function concept concretely: Using URL, text description, or the screenshot can help user to recognize what a function concept is. However, making many screenshots is costly. The blog-based tool should offer these ways for explaining function concepts. (8) Recording member background: Understanding who propose a request is important for deciding to whether to implement the request or not. Therefore the blog-based tool should assist the companies to collect member background information. (9) Offering the wizard of the request model maker: The next version of the blog-based tool should provide an easier way for making a request model. Providing a step-by-step wizard can guild users to make request models through a sequence of questions and answers. (10) Including other models: Use cases diagram is the key model in UML. However, use cases diagram is not enough for modeling more fine grain requirements, such as the workflow sequence of the online payment process. Therefore other models, such as activity

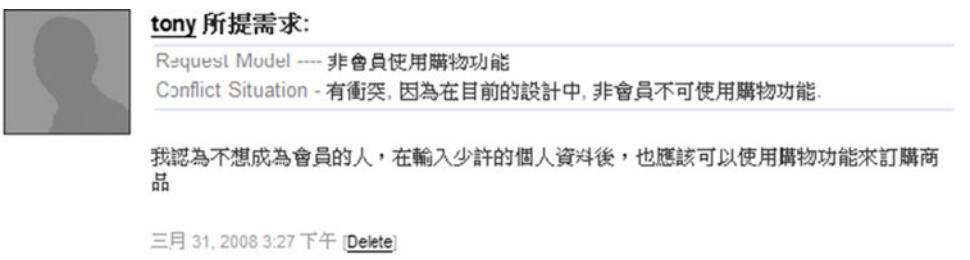

Fig. 14 Conflict detection results in the tool

Table 6 Amounts of requirements and conflicts

	Existing requirements	New requirements	Detected conflicts	Conflict reasons
Newspaper website	88	4	3	Access control contradiction; possible assumption violation
Shopping website	67	3	2	Access control contradiction; goal diversity

Table 7 Improvement recommendations about the prototype tool

	Improvable item	Recommendation
1	Indicating problems	Users can use this tool to easily indicate a problematic concept than propose a request model.
2	Rating a request	The tool should be used to collect how many users support or oppose the request.
3	Showing implementation status	Information system department should use this tool to declare whether the request is implemented or not.
4	Covering content request	This tool can be modified for collecting requirements about website content.
5	Revealing activity sequences	The tool should express the sequence of phases in an activity process.
6	Allowing multiple ontologies	The tool should allow departments or groups to use their own jargon to define their own ontologies.
7	Explaining each function concept concretely	The tool should provide function's description, such as its screenshot or URL.
8	Recording member background	The tool should record who proposes the request.
9	Offering the wizard of the request model maker	The tool should guide users to make a request model step by step.
10	Including other models	The tool should include other models beyond use cases.
11	Applying the tool for requirements phase of a new system	The tool should considered to support new system requirements analysis

Table 8 Comparison among the methodologies

	Detected conflict types	Tool support	Field experiment
AGG	Value diversity	Yes	No revealed in literature
LSPA	Nonfunctional contradiction	No	No revealed in literature
CARL	Logical contradiction	Yes	Partial
This study	Seven kinds of conflicts	Yes	Yes

diagram and sequence diagram, should be included in the next version of the blog-based tool. (11) Applying the tool for requirements phase of a new system: How to apply the tool for analyzing new system requirements is an interesting topic. However, the ontologies are difficult to be predefined before new requirements collection phase because the domain knowledge about an innovative and original new system is still unclear in this time. Therefore the tool should be redesigned to cope with this difficulty in supporting requirements phase of developing new systems.

8 Conclusion

From the theoretical perspective, this study proposes the extended use-case approach for presenting post-development change requests. This study also proposes a set of rules for detecting conflicts between new requests and existing system design base on ontologies. The extended use-case approach and the rules were implemented as the ontology-based blog prototype tool. The comparison among this study and the other methodologies are shown in Table 8.

From the practical perspective, the ontology-based blog can help the companies to collect requests and detect conflicts between request and existing system models. Two companies tried to use this ontology-based blog and confirmed that the tool is one of good alternatives for collecting post-development change requests and detecting conflicts. Several recommendations for improving the ontology-based blog tool are detailed and discussed. The impact of the conflict detection results can guide stakeholders to discuss the inconsistent requirements further. Without the conflict detection results, these inconsistent requirements may be implemented wrongly and cause disaster.

Although this study has obtained a preliminary fruit, more tool usage experiments and tool improvement are necessary in the further studies. Although this study proposes 18 rules, which are richer than the related works, some kinds of conflicts going beyond the best of our current knowledge may not be detected. More rules may be proposed in the future research.

References

Abran, A., & Nguyenkim, H. (1993). Measurement of the maintenance process from a demand-based perspective. *Journal of Software Maintenance Research and Practice, 5*(2), 63–90.

April, A., Hayes, J. H., Abran, A., & Dumke, R. (2005). Software maintenance maturity model (SM^{mm}): the software maintenance process model. *Journal of Software Maintenance and Evolution, 17*(3), 197–223.

Booch, G., Rumbaugh, J., & Jacobson, I. (1998). *The unified modeling language user guide*. Upper Saddle River: Addison-Wesley.

Chandrasekaran, B., Josephson, J. R., & Benjamins, V. R. (1999). What are ontologies, and why do we need then? *IEEE Intelligent Systems, 14*(1), 20–26.

Conallen, J. (1999). *Building web applications with UML*. Upper Saddle River: Addison-Wesley.

Dobing, B., & Parsons, J. (2000). Understanding the role of use cases in UML: A review and research agenda. *Journal of Database Management, 11*(4), 28–36.

Eden, A. H. (2002). A theory of object-oriented design. *Information Systems Frontiers, 4*(4), 379–391.

Fatolahi, A., & Shams, F. (2006). An investigation into applying UML to the Zachman framework. *Information Systems Frontiers, 8* (12), 133–143.

Gervasi, V., & Zowghi, D. (2005). Reasoning about inconsistencies in Natural language requirements. *ACM Transactions on Software Engineering and Methodology, 14*(3), 277–330.

Gruninger, M., & Lee, J. (2002). Ontology: Applications and design. *Communications of the ACM, 45*(2), 39–65.

Hausmann, J. H., Heckel, R., & Taentzer, G. (2002). Detection of conflicting functional requirements in a use case-driven approach. In: *Proceedings of the 24th international conference on software engineering* (pp. 105–115). Long Beach CA: IEEE Computer Society.

Jiao, J., & Chen, C. (2006). Customer requirement management in product developmenet: A review of research issues. *Concurrent Engineering: Research and applications, 14*(3), 173–185.

Kaiya, H., & Saeki, M. (2005). Ontology based requirements analysis: lightweight semantic processing approach. In: *Proceedings of the Fifth International Conference on Quality Software* (pp. 223–230). Long Beach CA: IEEE Computer Society.

Kaiya, H., & Saeki, M. (2006). Using domain ontology as domain knowledge for requirements elicitation. In: *Proceedings of 14th IEEE International Requirements Engineering Conference* (pp. 189–198). Long Beach CA: IEEE Computer Society.

Kang, D., & Chiang, R. (2006). A systematic approach in managing post-deployment system changes. *Communications of the ACM, 49*(6), 91–95.

Kajko-Mattsson, M. (2002). Problem management maturity within corrective maintenance. *Journal of Software Maintenance and Evolution, 14*(3), 197–227.

Kajko-Mattsson, M., & Bosu, M. (2006). Eliciting an enhancive maintenance model in three organisations in Ghana. In: *Proceedings of 5th IEEE/ACIS International Conference on Computer and Information Science and 1st IEEE/ACIS International Workshop on Component-Based Software Engineering*, *Software Architecture and Reuse* (pp. 244–251). Long Beach CA: IEEE Computer Society.

Lee, J., & Xue, N. (1999). Analyzing user requirements by use cases: A goal-driven approach. *IEEE Software, 16*(4), 92–101.

Madallli, D. P., & Suman, A. (2008). UML for the conceptual web. *Online Information Review, 32*(4), 511–515.

Paulo, J., Almeida, A., Iacob, M. E., & van Eck, P. (2007). Requirements traceability in model-driven development: Applying model and transformation conformance. *Information Systems Frontiers, 9*(4), 327–342.

Robinson, W. N., & Pawlowski, S. D. (1999). Managing requirements inconsistency with development goal monitor. *IEEE Transactions on Software Engineering, 25*(6), 816–835.

Robinson, W. N., Pawlowski, S. D., & Volkov, V. (2003). Requirements interaction management. *ACM Computing Surveys, 35*(2), 132–190.

Sommerville, I. (2005). Integrated requirements engineering: A tutorial. *IEEE Software, 22*(1), 16–23.

Chi-Lun Liu is an assistant professor in the Department of Information and Electronic Commerce at Kainan University in Taiwan. He holds his doctorate in Management Information Systems from ChengChi University, Taipei, Taiwan. His current research interests include social media, software engineering, information security, and internet marketing. Contact him at tonyliu@mail.knu.edu.tw or tonyliu@ms4.hinet.net.

Heng-Li Yang is a professor in the Department of Management Information Systems, National Chengchi University in Taiwan. He was the chairman of the department. His research interests include data & knowledge engineering, software engineering, knowledge management, information management in organizations, technology impacts on organizations, and empirical studies in MIS. His papers appeared on Computers in Human Behavior, Behaviour & Information Technology, Computers & Education, Information & Management, Information Processing and Management, Data and Knowledge Engineering, Online Information Review, Industrial Management & Data Systems, etc. Contact him at yanh@nccu.edu.tw.