ELSEVIER

Information and Software Technology 43 (2001) 121–135

INFORMATION AND SOFTWARE TECHNOLOGY

www.elsevier.nl/locate/infsof

Structuring requirement specifications with goals

Jonathan Lee[a,*], Nien-Lin Xue[a], Jong-Yih Kuo[b]

[a]*Software Engineering Laboratory, Department of Computer Science and Information Engineering, National Central University, Chungli, Taiwan, ROC*
[b]*Department of Computer Science and Information Engineering, Fu-Jen Catholic University, HsinChuang, Taipei Hsien, Taiwan, ROC*

Received 24 April 2000; revised 22 June 2000; accepted 18 July 2000

Abstract

One of the foci of the recent development in requirements engineering has been the study of conflicts and vagueness encountered in requirements. However, there is no systematic way in the existing approaches for handling the interactions among nonfunctional requirements and their impacts on the structuring of requirement specifications. In this paper, a new approach is proposed to formulate the requirement specifications based on the notion of goals along three aspects: (1) to extend use cases with goals to guide the derivation of use cases; (2) to analyze the interactions among nonfunctional requirements; and (3) to structure fuzzy object-oriented models based on the interactions found. The proposed approach is illustrated using the problem domain of a meeting scheduler system.

Keywords: Requirements engineering; Conflicting requirements; Goals; Fuzzy object-oriented model

1. Introduction

In recent years, goal-based requirements analysis methods have attracted increasing attention in the area of requirements engineering, as goals information is valuable in identifying, organizing and justifying software requirements [2,3,9,12,26,27,37,38]. The tenet of goal-based approaches is to focus on why systems are constructed, which provides the motivation and rationale to justify software requirements. Other benefits include: (1) helping to acquire requirements by elaborating what requirements are needed to support the goals; (2) making easy the justification and explanation of the presence of requirements in a progressive way by starting from system-level and organizational objectives from which such lower level descriptions are progressively derived [12]; and (3) providing the information for detecting and resolving conflicts that arise from multiple viewpoints among agents [12,29,37].

Subsequently, a number of researchers have reported progress toward the improvement of goal-based techniques [3,12,26,37]. In particular, Dardenne et al. [12] have advocated a goal-directed approach to models acquisition. Mylopoulos et al. [26] have proposed a framework for representing nonfunctional requirements in terms of goals, which can be evaluated in order to determine the degree to which a nonfunctional requirement is supported by a particular design. Moreover, they advocated that object-oriented modeling approach can then be used to model functional requirements to compensate the goal-oriented approach [27]. Meanwhile, Anton [3] has proposed a goal-based requirement analysis method to identify, elaborate and refine goals for requirements specifications.

However, there are three main problems with these approaches: (1) though their approaches support a systematic way to acquire and analyze requirements, no *de facto* standard modeling language has been adopted, which may impose a negative impact while applying to large scale systems; (2) though interactions among goals are a crucial factor in the structuring of requirement specifications, no attempt has been made to address such issues; and (3) though informality is an inevitable (and ultimately desirable) feature of the specification process [4,5], little effort has been devoted to the formulation of imprecise requirements.

To model imprecise requirements and conflicting requirements, we propose, in this paper, an approach to structure fuzzy object-oriented models [25] by extending our previous work on the goal-driven use cases approach [24] to formulate requirement specifications based on the notion of goals along three aspects (see Fig. 1 for an overview):

- *Extending use cases*[1] *with goals to guide the derivation of*

* Corresponding author. Tel.: +886-3-422-7151; fax: +886-3-422-2681.
E-mail addresses: yjlee@se01.csie.ncu.edu.tw (J. Lee), nien@se01.csie.ncu.edu.tw (N.-L. Xue).

[1] Use cases diagrams are one of the modeling techniques in UML to elicit, analyze and document requirements [11,21,32,33,34].

0950-5849/01/$ - see front matter
PII: S0950-5849(00)00144-0

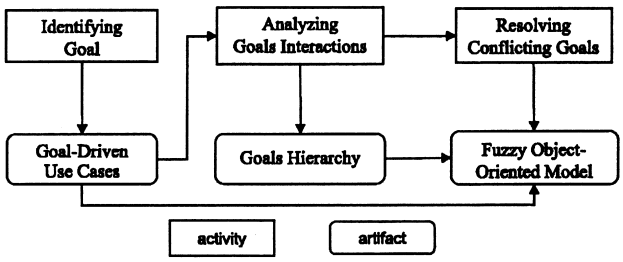


Fig. 1. Overview of our approach.

use cases: a faceted classification scheme is proposed to identify goals from domain descriptions and system requirements; and a use case is viewed as a process that can be associated with a goal to be achieved, optimized, maintained, ceased, or impaired by the use case.

- *Analyzing the interactions among nonfunctional requirements:* four types of interactions among nonfunctional requirements are identified, which could be either conflicting, cooperative, irrelevant, or counterbalanced.
- *Structuring fuzzy object-oriented models based on the interactions analyzed:* goals are organized into several alternatives based on the interactions analyzed to form a goals hierarchy; and a stable kernel is constructed to serve as a basis for further refinement in an incremental fashion. Various techniques are also proposed for resolving conflicts between goals into several alternatives based on the interactions analyzed to form a goals hierarchy; a stable kernel is constructed to serve as a basis for further refinement in an incremental fashion. Various techniques are also proposed for resolving conflicts between goals.

We chose the meeting scheduler problem as an example throughout this paper to illustrate our approach for two main reasons: First, as was pointed out by Potts et al. [28], the research community has adopted the meeting scheduling problem as a benchmark, and the requirements illustrate problems typical of requirements for real systems (see also Ref. [37] for more details). Second, the meeting scheduler problem can help us to address the main challenge for requirements analysis, that is, to turn a *vague* and *contradictory* mission statement into a detailed specification [8,28].

In this sequel, we first outline the related work in Section 2. The way to use the goals information to structure use cases is described in Section 3. The analysis of goals interactions is presented in Section 4. In Section 5, the structuring mechanism for modeling fuzzy object-oriented models requirement specifications based on the goals information and goals interactions is fully discussed. Finally, we summarize the potential benefits of the proposed approach and outline our future research plan in Section 6.

2. Related work

Work in a number of fields has made its mark on our research. Our approach has drawn upon several ideas from goal-based approaches [3,12,26], techniques in handling conflicts (Table 1) [7,15,36,39], and formulations of imprecise requirements [22,42].

2.1. Goal-based requirements engineering approaches

In Ref. [12], Dardenne et al. have proposed a goal-directed approach for model acquisition. Goals are seen as determining the respective roles of agents in the system and providing a basis for defining which agents should best perform which actions. In their approach, a goal is a nonoperational objective while a constraint is an operational one. That is, a goal cannot be achieved directly by the application of actions available to some agents. Instead, it is achieved by satisfying the constraints operationalizing it. To satisfy these constraints, appropriate restrictions may be required in turn on actions and objects.

Mylopoulos et al. have proposed a framework for representing and using nonfunctional requirements during the development process [26]. Goals are used to represent nonfunctional requirements, design decisions, and arguments in support of or against other goals. Goals representing nonfunctional requirements can rarely said to be "accomplished" or "satisfied" in a clear-cut sense. Instead, different design decisions contribute positively or negatively towards a particular goal. A labeling procedure to determining the degree to which a set of nonfunctional requirements is supported by a particular design is also proposed.

In Ref. [3], Anton proposes a goal-based requirement

Table 1
Variations of goal-based requirements engineering approaches

	Dardenne [12]	Mylouplos [26]	Anton [3]	Our approach
Types of goals	System goal, Privacy goal	Nonfunctional reqt. Goal, satisfying goal, argument goal	Achievement goal Maintenance goal	Rigid, soft, actor-specific System-specific, functional, nonfunctional
Roles of goals	Reqt. Acquisition, Reqt. Analysis	Nonfunctional Reqt. Analysis	Reqt. analysis Reqt. evolution	Use case structuring, Reqt. evolution, models structuring
Relationships between goals	Conflicting	Support, against	Dependent	Conflicting, cooperative, irrelevant, counterbalanced

analysis method (called GBRAM) to identify, elaborate and refine goals for requirement specifications. In GBRAM, an agent is responsible for the achievement of goals, which can be identified from process descriptions, transcripts of interviews, or searching action works. Goals are classified into two types: achievement goals and maintenance goals. Achievement goals usually map to actions that occur in a system, while maintenance goals map to nonfunctional requirements. Goals in GBRAM are elaborated and refined through identifying goal obstacles, analyzing scenarios and constraints. Scenarios are also used to help uncover hidden goals.

Rather than using scenarios to concretize goals, Rolland et al. [9] have proposed a new approach to discovering goals from scenarios. The basic modeling component is a combination of goal and scenario where the scenario is authored for the goal. Goals discovery and scenario authoring are complementary activities. Once a goal is discovered, scenario authoring can be done, followed by goal discovery. These goal-discovery/scenario-authoring sequence is repeated to incrementally populate a requirement chunks hierarchy.

2.2. Techniques in handling conflicts

A number of researchers have reported progress towards the analysis of conflicting requirements, which can be classified into three different types of conflict: conflicts due to the interactions among requirements advocated by different stakeholders [1,2,15,29,30], conflicts in between various designs [7,39,40], and conflicts resulted from the different structures adopted [18,36].

2.2.1. Interaction conflicts

In this case, requirements are said to be conflicting if the satisfaction of one requirement may impair or cease the satisfaction of another requirement. In a library system, a librarian wishes to minimize loan periods whereas a patron wishes to maximize the loan periods. A conflict arises due to the interactions among the requirements advocated by librarians and patrons. Robinson and Fickas [29] proposed an approach, called Oz, to requirements negotiation. There are three steps involved in Oz: conflict detection, resolution generation and resolution selection. The conflict detector of Oz does a pairwise comparison across all specifications. It does so by matching up design goals from perspectives and by comparing their plans. The specifications and conflicts will be passed to the conflict resolver, which will provide analytic compromise and heuristic compensation and dissolution for each conflict. Compensation is to add similar but conflict free requirements to negotiations, while, dissolution is to replace conflicting items potentially less contentious items. Finally, the resolver will provide guidance for search control by choosing intermediate alternatives and automated negotiation methods. Each method can be applied in any sequence to derive resolutions. The nonconflicting specifications are jointed into a single specification by merger of Oz.

Easterbrook [15] proposes a framework for representing conflicting viewpoints in a domain model. A viewpoint in his framework is a self-consistent description of an area of knowledge representing the context in which a role is performed. In evolving viewpoints, a new viewpoint will need to be split if it causes inconsistency. The new viewpoint and its negation are placed in different descendants of the original viewpoint, so that each remains self-consistent individually. The detection of conflict might be based on detection of logical inconsistencies. Thus, a hierarchy of viewpoints is established as the elicitation proceeds. The inheritance structure implies that the higher an item in the hierarchy, the more widely agreed it is. One of the aims of using viewpoints is to reduce the need for consistency checks. This approach allows many viewpoints to be combined into a single domain model without necessary resolving conflicts between them. Conflicts are treated as an important part of the domain and are to be represented and understood.

In Ref. [2], Lamsweerde et al. introduce a weak form of conflict called divergence. A divergence is defined as a logical inconsistency between goals under a specific boundary condition. Formal techniques and heuristics are proposed for detecting divergences from the specifications (based on real-time temporal logic) of goals. Various divergence patterns are also identified to help divergence detection by selecting a matching generic pattern and by instantiating it accordingly. Divergences can be resolved by creating/modifying/deleting goals assertions or transforming the objects in the specification into a conflict-free version.

2.2.2. Design conflicts

Conflicts may arise if two requirements cannot be both supported by a design architecture or model. For example, portability can be improved via a layered architecture, but usually at some cost in performance. In this case, the requirements portability and performance are conflicting w.r.t. an architecture, rather than conflicts on their definitions.

To examine the requirements tradeoffs involved in software architecture and process strategies, Boehm et al. [7] proposed an exploratory knowledge-based tool, quality attribute risk and conflict consultant (QARCC), for identifying potential conflicts among quality attributes, flagging them for affected stakeholders, and suggesting options to resolve the conflicts early in the software life cycle. It operates in the context of the WinWin system, a groupware support system for determining software and system requirements as negotiated win conditions. The WinWin system uses Theory W to generate the objectives, constraints, and alternatives to provide win condition. To meet its goal of "making everyone a winner," Theory W involves stakeholders in a process of identifying their

quality-attribute win condition and reconciling conflicts among quality-attribute win conditions.

A number of researchers have put effort into making generic conceptual models available and reusable for benefiting from reusing experience with previous design. One of their challenges is to face the tradeoffs about selecting conceptual model components, as a conceptual design component property may contribute to a particular requirement, but adversely affect another requirement. To model the history of discussion, negotiation and compromise that led to a conceptual design, Vanwelkenhuysen [39,40] proposes a Design Requirement Embedding (DRE) approach. In DRE, by recognizing the interactions between design properties and interactions between design properties and requirements, competing requirements are explored and a design guideline is proposed for the conflict. Design teams can negotiate with users based on proposed design guidelines.

2.2.3. Structural conflicts

A structural conflict arises whenever parts of the same reality are represented in different views using different structural constructs. For instance, an object may be represented as an entity type in one view and as an attribute of an entity type in another view. The generalization concept has been extensively used as a solution to such conflicts (see Ref. [18] for a survey). In particular, Spaccapietra [36] proposes a view integration methodology, designed to be able to automatically resolve structure conflicts among different views without modifying the initial models. To that purpose, knowledge about correspondences that exist among different views should be acquired initially and represented as a formal model. Finally, integration rules are used to build a integrated schema according to the known correspondences and based on the concept of generalization.

2.3. Formulations of imprecise requirements

Our previous work on Requirements Trade-off Analysis technique (RTA) has been on the formulation of vague requirements based on Zadeh's canonical form in test-score semantics [44] and an extension of the notion of soft conditions [22]. The trade-off among vague requirements is analyzed by identifying the relationship between requirements, which could be either conflicting, irrelevant, cooperative, counterbalance, or independent. Parameterized aggregation operations, *fuzzy and/or* [45], are selected to combine individual requirements. An extended hierarchical aggregation structure is proposed to establish a four-level requirements hierarchy to facilitate requirements and criticalities aggregation through the *fuzzy and/or*. A compromise overall requirement can be obtained through the aggregation of individual requirements based on the requirements hierarchy. The proposed approach provides a framework for formally analyzing and modeling conflicts between requirements, and for users to better understand relationships among their requirements.

Yen et al. [42,43] take the notion of conflicting and cooperative degrees as the view of distance-wise between any two individual requirements. They present a formal approach for reasoning about the relative priority by analyzing the customer's trade-off preference among imprecise conflicting requirements. A requirement R_1 is supposed to be more important than another requirement R_2 whenever the domain expert is willing to sacrifice a lot in the satisfaction degree of a requirement R_2 for a small increase in the satisfaction degree of R_2. Moreover, the ratio of R_1's priority to that of R_2 is proportional to the ratio of their changes in the satisfaction degrees.

3. Structuring use cases with goals information

Goal-driven use cases is an approach for requirements engineers to elicit and structure users requirements, and to analyze and evaluate relationships between requirements.[2] There are three steps to construct use cases: (1) identify actors by investigating all possible types of users that interact with the system directly; (2) identify goals based on a faceted classification scheme; and (3) build use case models.

3.1. Identifying actors

An actor is an outside entity that interacts directly with a system, which may be a person or a quasi-autonomous object, such as machines, computer tasks, and other systems. More precisely, an actor is a role played by such an entity. For example, the meeting scheduler system is mainly designed for an initiator to organize a meeting schedule, and therefore, the *initiator* is marked as an actor.

3.2. Identifying goals

A faceted classification scheme is proposed for identifying goals from domain descriptions and system requirements. Each goal can be classified under three facets we have identified: competence, view and content. The facet of competence is related to whether a goal is completely satisfied or only to a degree. A *rigid* goal describes a minimum requirement for a target system, which is required to be satisfied utterly. A *soft* goal describes a desirable property for a target system, and can be satisfied to a degree. For example, if a meeting schedule is convenient for all attendants, the goal *MaxConvenienceSchedule* is defined to be satisfied completely. However, if the schedule is convenient only to some of the attendants, the goal is said to be satisfied to a degree. A soft goal is related to a rigid one in the sense that the existence of the soft goal is dependent on the rigid one.

[2] In order to map fuzzy object-oriented models to code level, we will need to have a program language with fuzzy features, that is, to define membership function of a fuzzy set, to construct fuzzy rules and fuzzy inference, etc.

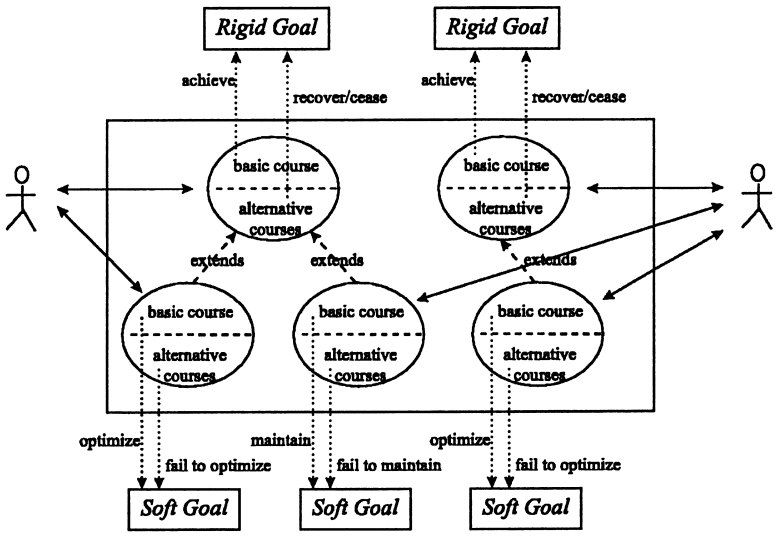


Fig. 2. Extends relationships between use cases.

The facet of view concerns whether a goal is actor-specific or system-specific. Actor-specific goals are objectives of an actor in using a system; meanwhile, system-specific goals are requirements on services that the system provides. For example, through examining the system description, we have found out that the initiator has three objectives in using the meeting scheduler system: (1) to create a meeting; (2) to make the meeting schedule as convenient as possible for the participants; and (3) to maximize the number of participants for the meeting. Therefore, three actor-specific goals can be identified: *MeetingRequestSatisfied*, *MaxNumberOfParticipants* and *MaxConvenienceSchedule*. On the other hand, a system-specific goal takes into consideration "what kinds of properties the system should have in supporting services to *all* users?", or "what are the requirements on the services for the system to provide?". In our example, to construct a meeting is an objective of an initiator, but to accommodate a more important meeting is a requirement of the system. Therefore, a system-specific goal — *SupportFlexibility*, is identified.

Usually, requirements can be classified into functional and nonfunctional requirements based on their content [22]. The construction of functional requirements involves modeling the relevant internal states and behavior of both the component and its environment. Nonfunctional requirements usually define the constraints that the product needs to satisfy. Therefore, a goal can be further distinguished based on its content, that is, a goal can be either related to the functional aspects of a system or associated with the nonfunctional aspect of the system.[3] A functional goal can be achieved by performing a sequence of operations. A nonfunctional goal is defined as constraints to qualify its related functional goal. In our example, creating a meeting is accomplished by a sequence of operations, therefore the goal *MeetingRequestSatisfied* is defined as a functional goal. On the other hand, goals like *MaxNumberOfParticipants* and *MaxConvenienceSchedule* can be viewed as constraints for a schedule to satisfy, which are identified as nonfunctional goals.

3.3. Building use case models

To structure a use case and its extensions, we extend the work by Cockburn [10] by considering several different types of goal. Essentially, each use case is viewed as a process that can be associated with a goal to be achieved, optimized, or maintained by the use case (Fig. 2). To start with, we first consider original use cases to guarantee that the target system will be at least adapted to the minimum requirements. Each original use case in our approach is associated with an actor to describe the process to achieve an *original goal* which is rigid, actor-specific and functional (Fig. 2). Building original use cases by investigating all original goals will make the use case model satisfy at least all actors' rigid and functional goals.

The basic course in an original use case is the simplest course, the one in which the goal is delivered without any difficulty. The alternative course encompasses the recovery course and/or the failure one. The recovery course describes the process to recover the original goal, whereas the failure one describes what to do if the original goal is not recoverable.

In our example, the use case *plan a meeting* covers the case for an initiator to achieve the goal *MeetingRequestSatisfied* (Fig. 3) which is rigid, actor-specific and functional. The use case starts when an initiator issues a meeting request to the system, and lasts until a meeting schedule is generated or canceled. It is the basic course that forms the foundation when specifying a use case and this should be

[3] Similar ideas are advocated in Ref. [3], where achievement and maintenance goals map to actions and nonfunctional requirements, respectively.

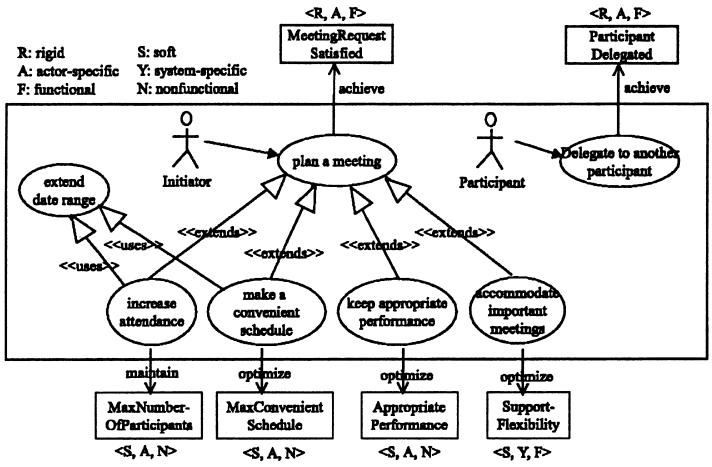


Fig. 3. A goal-driven use case model for meeting scheduler system.

described first. The use case has several alternative courses that may change its flow. An example of this is different ways of recovering the goal *MeetingRequestSatisfied* when there exists a strong conflict in a schedule.

Original use cases are designed to satisfy original goals for modeling users minimum requirements. To extend the model to take into account different types of goals, extension use cases are created. Situations about when to create extension use cases are fully discussed below:

- *To optimize or maintain a soft goal.* By achieving a rigid goal, all its related soft goals can also be satisfied to some extent. To optimize or maintain the soft goals, extension use cases are created (see Fig. 2, the use case E_1). Therefore, the basic course in an extension is to optimize (or maintain) its soft goal, whereas an alternative course describes what to do if it fails to optimize (or maintain) the goal. In our example, to satisfy the rigid goal *MeetingRequestSatisfied* does not guarantee that the meeting is convenient for all participants. To make the schedule as convenient as possible, the extension *make a convenient schedule* is created. If the constraints in the basic course are not satisfied, the alternative course is to recover the optimization of the soft goal, for example, to extend the date range, or to ask participants to add dates to their preference sets.
- *To achieve a system-specific goal.* An extension use case may be created to achieve a system-specific goal (see Fig. 2, the use case E_2). Referring to our example, the original use case *plan a meeting* describes the process to create a meeting from a personal view (the view of the actor *initiator*). The extension use case *accommodate important meetings* extends it to take all initiators into account, that is, to achieve a system-specific goal *SupportFlexibility*.
- *To achieve a nonfunctional goal.* To extend a use case model to capture nonfunctional requirements, extension use cases are added to achieve a nonfunctional goal (see Fig. 2, use case E_3). In this case, an extension use case serves as a constraint to qualify its original use case. In our example, the original use case *plan a meeting* is a direct course to create a meeting, several constraints (may be rigid or soft) on a meeting are ignored: *AppropriatePerformance*, *MaxNumerOfParticipants* and *MaxConvenienceSchedule*. The basic course of *make a convenient schedule* indicates the soft constraints on a meeting schedule. If the constraints are not satisfied, the alternative course is to recover the optimization of the soft goal.

To summarize, an *original goal* is a goal that is ⟨rigid, actor-specific, functional⟩, whereas a goal that is achieved, optimized or maintained by an extension use case is called an *extension goal*. An extensional goal is weakly dependent on its associated original goal, that is, the existence of an extension goal is dependent on an original one. It should be noted that satisfying an original goal does not always make its associated extension goals satisfied (or satisfied to degree), excepting that the extension goal is a soft one.

4. Evaluating goals interactions

Interactions between goals can be evaluated through the following steps: (1) analyze the interactions between use cases and goals by investigating the effects on the goals after the use cases are performed; (2) explore the interactions between goals in the use case level; and (3) derive the interactions between goals in the system level.

4.1. Relationships between use cases and goals

To better characterize the interactions between use cases and goals, we have adopted predicates proposed by Mylopoulos et al.[26]: satisfied, denied, satisfiable, and deniable. A goal can be either *satisfied* or *denied*, if the goal is achieved or ceased utterly, respectively. On the other hand, the predicates *satisfiable* and *deniable* are used to describe a goal that can be satisfied or denied to a degree.

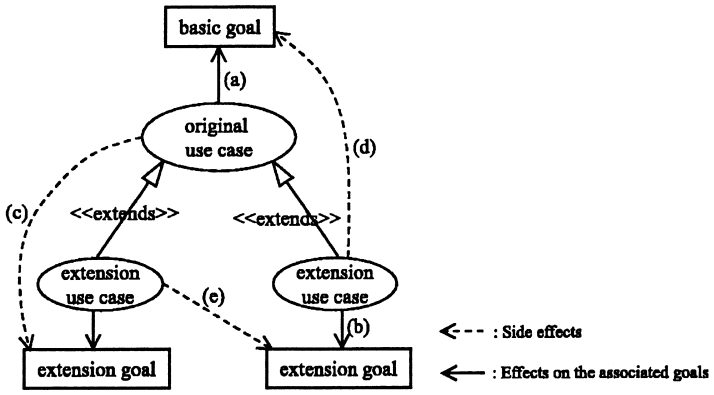


Fig. 4. Interactions between use cases and goals.

In addition, a predicate *independent* is introduced to describe the situation that a goal will not be affected by performing a designated use case.

A use case is designed to achieve, optimize or maintain its directly associated goals. However, it may occur that goals not directly associated with the use case can also be affected, called side effects. Interactions between use cases and goals can be analyzed by investigating the effects on the associated goals and side effects on other goals after the use cases are performed:

- *Effects on the associated goals.* An original goal can be achieved by performing its original use case, that is, the goal can be *satisfied* either by performing the basic course successfully or by recovering the goal from an alternative course (see Fig. 4, arrow (a)). The goal can also be *denied* under the condition that it is ceased by performing an alternative course. Similarly, an extension goal can be *satisfied* or *satisfied* by performing its associated extension use case (see Fig. 4 arrow (b)).
- *Side effects.* There are three cases that side effects may occur (see Fig. 4, arrows (c), (d) and (e)). (1) By performing an original use case successfully, the extension goals which are directly associated with its extension use cases are also achieved to some extent, that is, the extension goals are *satisfiable* (see Fig. 4, arrow (c)). For example, if the use case *plan a meeting* is successfully performed, the extension goal *MaxConvenienceSchedule* is satisfied to a degree. (2) An extension use case may impair the original goal which is directly associated with its original use case (see Fig. 4, arrow (d)). In this case, the original goal is *denied*. Referring to our example, the extension use case *accommodate important meetings* may cease the original goal *MeetingRequestSatisfied* under the condition that there is a more important meeting that is in conflict with it. (3) An extension use case may achieve or impair an irrelevant goal which is associated with other extension use cases (see Fig. 4, arrow (e)). A typical example of impairing an irrelevant goal in the meeting scheduler system is that performing the extension use case *keep a appropriate performance* may cease the activity to negotiate a convenient schedule, therefore, the soft goal *MaxConvenienceSchedule* is *deniable*.

4.2. Interactions between goals in the use case level

Interactions between goals can be considered in two different levels: use case level and system level. The former concerns the interactions between goals with respect to (wrt) a specific use cases, and the latter focuses on the overall system. In the use case level, two goals are said to be conflicting wrt a use case, if the satisfaction of one goal is increased, while other is decreased after the use case is performed. On the other hand, two goals are said to cooperate with each other, if both the satisfaction degrees of the goals are either increased (positively cooperative) or decreased (negatively cooperative). The third possibility is that the satisfaction degrees of goals remain unchanged. In this case, the goals are said to be irrelevant.[4]

Interactions between two goals wrt a use case can be derived by interactions between the use case and goals. For example, if the interactions between a use case U_k and goals G_i and G_j are *satisfiable* and *deniable*, respectively, it means that the satisfaction degree of G_i is increased and that of G_j is decreased after U_k is performed. The interactions between G_i and G_j wrt U_k is said to be conflicting.

The predicates $cp_{U_k}(G_i,G_j)$ and $cf_{U_k}(G_i,G_j)$ are introduced to describe the relationship between goals G_i and G_j wrt the use case U_k, where $cp_{U_k}(G_i,G_j)$ is true if G_i and G_j are cooperative wrt the use case U_k, and $cf_{U_k}(G_i,G_j)$ is true if G_i and G_j are conflicting wrt U_k. If the goals G_i and G_j are irrelevant wrt U_k, the predicates $cp_{U_k}(G_i,G_j)$ and $cf_{U_k}(G_i,G_j)$ are both false. Referring to our example, the relationships between the use case *make a convenient schedule* and the goals *MaxConvenienceSchedule* and *AppropriatePerformance* are *satisfiable* and *deniable*, respectively. Therefore, we can conclude that the two goals are conflicting wrt the use case *make a convenient*

[4] Similar ideas are also advocated in our previous work [22].

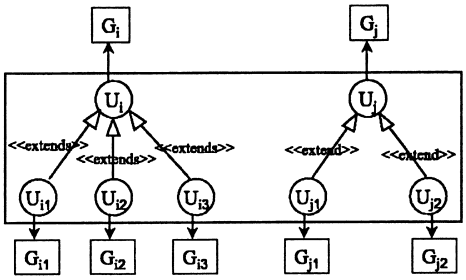


Fig. 5. All illustration of a use case model.

schedule (that is, $cf_{make\ a\ convenient\ schedule}$ (*MaxConvenientSchedule*, *AppropriatePerformance*) = *True*).

4.3. Interactions between goals in the system level

In the use case level, relationships between any two goals are analyzed wrt a specific use case, whereas relationships between goals in the system level are mainly on the use case models where related use cases are amalgamated together. In addition, as Ebert pointed out in Ref. [16], there is more relevance in the case of nonfunctional requirements compared to functional requirements as there are in most systems severe trade-offs among nonfunctional requirements. Our approach focuses on the relationships between nonfunctional goals in the system level.

The interaction between the goals G_i and G_j in the system level is denoted as $R_s(G_i,G_j)$, and is defined as a pair of predicates $\langle cp(G_i,G_j),cf(G_i,G_j)\rangle$, where $cp(G_i,G_j)$ is true if G_i is cooperative with G_j, and $cf(G_i,G_j)$ is true if G_i is conflicting with G_j in the system level. There are four possible interactions between goals in the system level:

- $R_s(G_i, G_j) = \langle False, False\rangle$: G_i and G_j are *irrelevant* in the system level;
- $R_s(G_i, G_j) = \langle True, False\rangle$: G_i and G_j are *cooperative* in the system level;
- $R_s(G_i, G_j) = \langle False, True\rangle$: G_i and G_j are *conflicting* in the system level;
- $R_s(G_i, G_j) = \langle True, True\rangle$: G_i and G_j are *counterbalanced* in the system level.

In our approach, interactions between nonfunctional goals in the system level can be derived based on use case models and relationships between use cases in the use case level. In the following paragraphs, we will describe how to derive interactions between goals in the system level by means of the example in Fig. 5, where G_{i1} and G_{j1} are two nonfunctional goals and U_{i1}, U_{j1} are their associated use cases, respectively. U_i is an original use case of U_{i1}, and G_i is its associated original goal.

To explore the interaction between the nonfunctional goals G_{i1} and G_{j1}, the original goals that the nonfunctional goals are weakly dependent on should also be considered. In Fig. 5, the extension goal G_{j1} is achieved (optimized or maintained) under the situation that G_i is satisfied, thus the interaction between G_{i1} and G_{j1} wrt the use case U_i should be also taken into account. More precisely, the interaction between G_{i1} and G_{j1} (i.e. $R_s(G_{i1},G_{j1})$) hinges on: (1) the interaction between G_{i1} and G_{j1} wrt the use case U_i; (2) the interaction between G_{i1} and G_{j1} wrt the use case U_j; (3) the interaction between G_{i1} and G_{j1} wrt the use case U_{i1}; and (4) the interaction between G_{i1} and G_{j1} wrt the use case U_{j1}. That is, $R_s(G_{i1}, G_{j1}) = \langle cp(G_{i1}, G_{j1}), cf(G_{i1}, G_{j1})\rangle$, where

$$cp(G_{i1}, G_{j1}) = cp_{U_i}(G_{i1}, G_{j1}) \vee cp_{U_j}(G_{i1}, G_{j1}) \vee cp_{U_{i1}}(G_{i1}, G_{j1}) \vee cp_{U_{j1}}(G_{i1}, G_{j1});$$

$$cf(G_{i1}, G_{j1}) = cf_{U_i}(G_{i1}, G_{j1}) \vee cf_{U_j}(G_{i1}, G_{j1}) \vee cf_{U_{i1}}(G_{i1}, G_{j1}) \vee cf_{U_{j1}}(G_{i1}, G_{j1})$$

In our example, let $R_s(G_{MCS}, G_{AP}) = \langle cp(G_{MCS}, G_{AP}), cf(G_{MCS}, G_{AP})\rangle$ be the relationship between the nonfunctional goals *MaxConvenienceSchedule* (G_{MCS}) and *AppropriatePerformance* (G_{AP}) (U_{PM} and U_{KAP} are abbreviation for the use cases *plan a meeting* and *keep appropriate performance*, respectively):

$$cp(G_{MCS}, G_{AP}) = cp_{U_{PM}}(G_{MCS}, G_{AP}) \vee cp_{U_{MCS}}(G_{MCS}, G_{AP}) \vee cp_{U_{KAP}}(G_{MCS}, G_{AP}) = False$$

$$cf(G_{MCS}, G_{AP}) = cf_{U_{PM}}(G_{MCS}, G_{AP}) \vee cf_{U_{MCS}}(G_{MCS}, G_{AP}) \vee cf_{U_{KAP}}(G_{MCS}, G_{AP}) = True$$

Therefore, the goals *MaxConvenienceSchedule* and *AppropriatePerformance* are in conflict with each other.

5. Structuring fuzzy object-oriented models specifications through goals interactions

Having developed use cases and analyzed their interactions, we can now focus on the structuring of fuzzy object-oriented models. The tenet of our structuring mechanism is on the utilization of the interactions found among goals. There are two steps involved: (1) to establish a goals hierarchy for obtaining alternative models specified using fuzzy object-oriented models notations; and (2) to construct a stable kernel in an incremental fashion to serve as a basis for integrating alternative models.

5.1. Establishing a goals hierarchy

Instead of putting heavy efforts on resolving conflicts at the beginning of system design which may overload

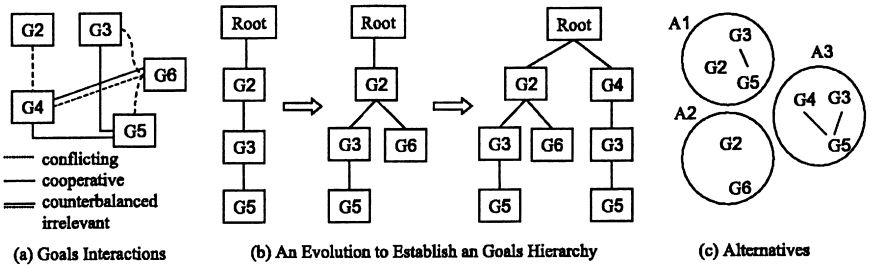


Fig. 6. An illustration on how to establish a goals hierarchy.

analysts, we start with constructing a model to meet a set of conflict-free requirements. To this end, we organize the goals into several alternatives based on the interactions analyzed such that each alternative contains a set of goals that are not conflicting with each other. In another word, no conflict is allowed in an alternative. More specifically, goals organized into alternatives must satisfy the constraints below:

- For any two goals G_i and G_j in an alternative, the interaction between G_i and G_j is either *cooperative* or *irrelevant*.
- For any two alternatives A_i and A_j, there exist goals $G_i \in A_i$ and $G_j \in A_j$ such that the interaction between G_i and G_j is either *conflicting* or *counterbalanced*.
- For any alternative A_k and goal G_i where $G_i \notin A_k$, there exists a goal $G_k \in A_k$ such that the interaction between G_i and G_k is either *conflicting* or *counterbalanced*.

In order to explore all the possible alternatives, we utilize the notion of a goals hierarchy in which alternatives are obtained by tracing each path from a leaf node up to the root. The following algorithm outlines the process involved in establishing a goals hierarchy. As functional goals are usually the basic requirements that must be satisfied, they are placed on the root of the hierarchy and therefore included in every alternative.

Algorithm (Establish a Goals Hierarchy)

1. *Place the functional goals on the top of the hierarchy (i.e. the root of the hierarchy).*
2. *Create an alternative:*
   (a) *Select a nonfunctional goal, say G_i; place G_i as a child node of the root and set it as the current node*;
   (b) *If there exist nonfunctional goals that are not conflicting or counterbalanced with any goal that precedes the current node*;
      i. *Select a goal from those nonfunctional ones, say G_j.*
      ii. *Place G_j as the child node of the current one.*
      iii. *Set G_j as the current node.*
      iv. *Repeat step 2.b.*
   (c) *Else go to step 3.*
3. *Backtrack to create another alternative:*
   (a) *If there exist nonfunctional goals that are conflicting or counterbalanced with the current node, but not conflicting or counterbalanced with any goal that precedes the current node*;
      i. *Select a goal from those nonfunctional ones, say G_j.*
      ii. *Place G_j on the right of the current node.*
      iii. *Set G_j as the current node.*
      iv. *Go to step 2.b.*
   (b) *Else if the current node is the root node, then exit; else reset the current node as the parent of the current node and go to step 3.*

Fig. 6b illustrates how a goals hierarchy is established. The goals G_2, G_3 and G_5 are added incrementally to the root (i.e. the functional goals) to form an alternative. Since adding the goal G_6 or G_4 to the alternative will result in a conflict, we backtrack to create another alternative. By

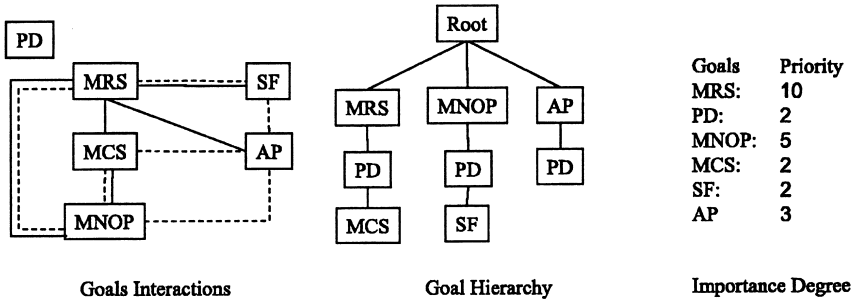


Fig. 7. A goals hierarchy for meeting scheduler system.

doing so, we have come up with three different alternatives: $\{G_{FG},G_2,G_3,G_5\}$, $\{G_{FG},G_2,G_6\}$, and $\{G_{FG}, G_4, G_3, G_5\}$ (Fig. 6c).

By applying the above algorithm to our meeting scheduler example, two alternatives can thus be built (Fig. 7): $\{G_{MRS},G_{PD},G_{MCS},G_{MNOP},G_{SF}\}$, and $\{G_{MRS}, G_{PD}, G_{AP}, G_{MI}\}$.

5.2. Constructing a stable kernel

Essentially, a kernel contains all the functional requirements and a set of nonfunctional requirements that are either cooperative or irrelevant to each other (i.e. no conflict is allowed). A kernel is stable in the sense that no engagement in the annoying conflicts resolution is required, and that it can be used to serve as a basis for further refinement in an incremental fashion.

To choose a stable kernel from the alternatives found in the previous step, the *cooperative degree* and *importance degree* of an alternative are introduced. A cooperative degree of an alternative is defined as the total numbers of cooperative interactions within the alternative. In Fig. 6c, the cooperative degree of the alternative A_1 is equal to 1 since there is a cooperative interaction between G_3 and G_5.

The importance degree of an alternative is the sum of the weights of goals in the alternative. We adopt Saaty's pairwise comparison approach to the assignment of weights to goals [35]. That is, the relative weights of each goal pair are used to form a reciprocal matrix, and the absolute weight of each goal is obtained from the normalized eigenvector using eigenvalue method. In the meeting scheduler system (Fig. 7), as the alternatives A_1 and A_2 have the same cooperative degrees, we evaluate the alternatives based on their importance degrees. The alternative A_1 is chosen as the stable kernel since it carries a higher importance degree than that of the alternative A_2.

To construct the kernel model, fuzzy object-oriented models [25] are used to model imprecise requirements. In fuzzy object-oriented models, we have identified several kinds of fuzziness that are required to model imprecise information involved in user requirements.

- classes with imprecise boundary to describe a group of objects with similar attributes, similar operations and similar relationships;
- rules with linguistic terms that are encapsulated in a class to describe the relationships between attributes;
- ranges of an attribute with linguistic values or typical values in a class to define the set of allowed values that instances of that class may take for the attribute;
- the membership degree (i.e. ISA degree) between an object and a class, and between a subclass and its superclass (i.e. AKO degree) can be mapped to the interval [0,1]; and
- associations between classes that an object instance may participate to some extent.

5.2.1. Inside a fuzzy class

Traditionally, a class is used to describe a crisp set of objects with common attributes, common operations and common relationships. In order to model the impreciseness rooted in user requirements, fuzzy object-oriented models extends a class to describe a fuzzy set of objects (called a fuzzy class), in which objects may have similar attributes, similar operations and similar relationships, for example, a set of interesting books or a class of clever students. In the meeting scheduler system, the class *ImportantParticipant* is modeled as a fuzzy class, that is, a participant may be an important one to a degree.

A fuzzy class in fuzzy object-oriented models is an encapsulation of a number of properties that can be classified as static properties or dynamic ones. Static properties are viewed as integral features of an object that exist for its lifetime including identifier, attributes and operations. On the other hand, dynamic properties are optional for an object and can be short-lived such as fuzzy rules.

Since a fuzzy class is a group of objects with similar static properties (i.e. attributes, operations) and similar dynamic properties (i.e. relationships and rules), the membership degree of an instance to a fuzzy class is dependent on the properties, especially the values of attributes and the values of link attributes. In our example, the degree that a person belongs to the class *ImportantParticipant* depends on his *status* and his *role* in the meeting he attends.

5.2.1.1. Attributes with fuzzy ranges. In fuzzy object-oriented models, the fuzziness in the range of an attribute in a class may be due to either a linguistic term or a typical value.

- A class may be fuzzy for the linguistic values its attributes can take. For example, the class *YoungMan* has a fuzzy range for the attribute *age*, since a person may take *young* or *very young* as values for his age.
- The range of an attribute is fuzzy [19] because some of its values are deemed as atypical (i.e. less possible than other values), therefore, each value the attribute may take is associated with a typical degree.[5] In our example, the class *ImportantParticipant* has a fuzzy range {*student*/0.4, *staff*/0.7, *faculty*/1} for the attribute *status*, which means that a faculty is typically an important participant, and a student is an important participant with a typical degree of 0.4.

It is of interest to note that a crisp class may have attributes with fuzzy ranges. For instance, the class *MeetingRegistration* is a crisp class, with an attribute *participant importance*, which is associated with a fuzzy range.

5.2.1.2. Fuzzy rules. Incorporating fuzzy rules in

[5] The notion of typical values is adopted from Ref. [14].

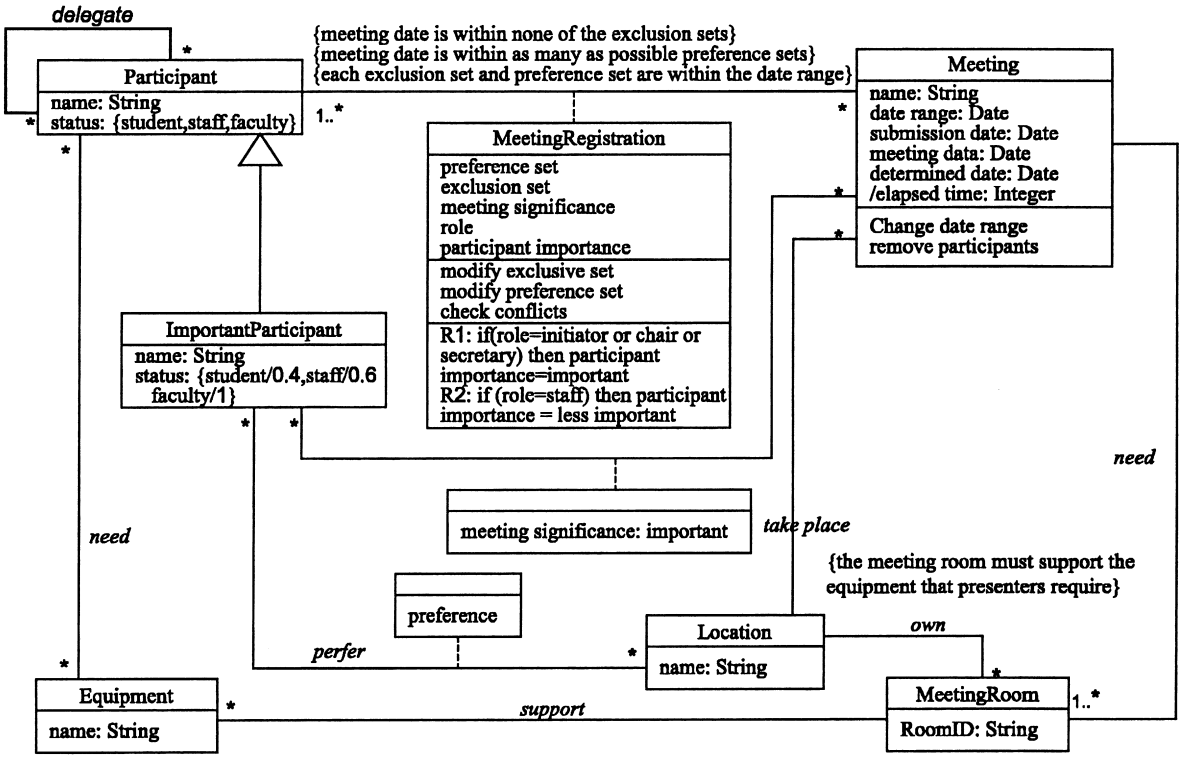


Fig. 8. A kernel model for the meeting scheduler system.

object-oriented analysis can help enrich the semantics of analysis models [17]. Using fuzzy rules is one way to deal with imprecision where a rule's conditional part and/or the conclusions part contains linguistic variables. More specifically, fuzzy rules in a fuzzy class play two important roles: to specify internal relationships between attributes, and to describe triggers more explicitly. Fuzzy rules are used to describe the internal relationship or external relationship. In the former, fuzzy rules describe the relationship between attributes inside a class. For example, a rule "if the *role* is a staff, the *participant importance* is *less important*" describes the relationship between the attributes *role* and *participant importance*. In the latter, fuzzy rules are used to describe the relationship between two different classes.

5.2.2. Fuzzy classification

Perceptual fuzziness refers to the compatibility between a class and an object (i.e. ISA), and the class membership between a class and its subclass (i.e. AKO) [41]. In fuzzy object-oriented models, we extend crisp class memberships to fuzzy class memberships by allowing the existence of perceptual fuzziness. In the meeting scheduler system, a person may belong to the class *ImportantParticipant* to some extent.

The membership degree of an object to a class is established either explicitly or implicitly [6]. In [20], the ISA degree is explicitly given, and used to derive the values of attributes of the object. In fuzzy object-oriented models, the ISA degree is implicitly determined by the structure of classes. The perceptual fuzziness of an object to a class or a subclass to its superclass is calculated by evaluating both the static properties and dynamic properties. Referring to our example, the membership degree of a person to the class *ImportantParticipant* can be obtained by checking his *status* (static property) and his *role* in the meeting he attends (dynamic property).

5.2.3. Uncertain fuzzy associations

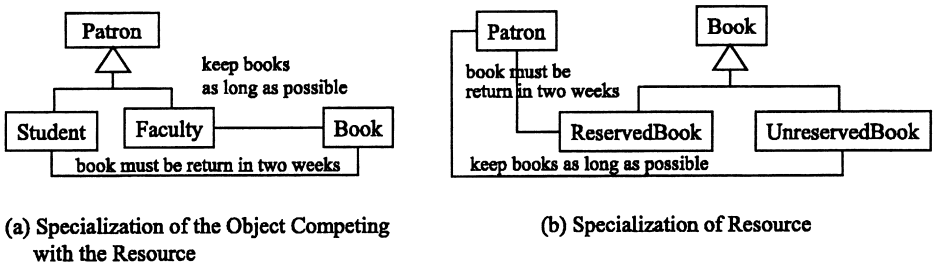
Links and associations are means for establishing relationship among objects and classes. A link is a physical or conceptual connection between object instances. For example, John *work-for* Simplex company. An association describes a group of links with common structure and common semantics. For example, a person *work-for* a company. In traditional object-oriented approaches, only crisp associations are introduced, namely, an object either participates in an association or not.

Usually, certain and precise knowledge about an association is not always available in the user requirements; furthermore, users' observations are sometimes uncertain and imprecise. Therefore, an adequate management of uncertainty and imprecision in the phase of requirements analysis is an important issue. The distinction between imprecise and uncertain information can be best explained by Dubois and Prade [13]: imprecision implies the absence of a sharp boundary of the value of an attribute; whereas,

Goal: to accommodate important meetings (SupportFlexibility: <soft,system-specific, *functional* >).

Basic course When an initiator issues a meeting, the system checks whether there is any meeting whose schedule is inclusive in the date range of the new meeting and has higher priority. If so, the system informs the initiator and asks him to change the meeting's date range.

Fig. 9. An alternative use case to achieve *supportflexibility*.



(a) Specialization of the Object Competing with the Resource

(b) Specialization of Resource

Fig. 10. Resolving conflicts from competing resources by specialization.

uncertainty is an indication of our reliance about the fuzzy information.

A uncertain fuzzy association is allowed in fuzzy object-oriented models. The imprecision of an association implies that an object can participate in the association to some extent, whereas uncertainty is referred to the confidence degree about the association. To represent the imprecision of an association, a special link attribute is introduced in fuzzy object-oriented models to indicate the intensity that objects participate in an association. Fuzzy truth value, such as *true*, *fairly true* and *very true*, is used to serve as the representation of uncertainty for its capability to express the possibility of the degree of truth [23].

A link between x and y which is an instance of the association R is represented as a canonical form in fuzzy object-oriented models:

(*link attribute*, $\langle x, y\rangle$, *degree of participation*, τ)

where the first component of the quadruple is a link attribute of an association. The value that a link $\langle x,y\rangle$ takes for the link attribute is described in the *degree of participation*, which represents the degree that objects x and y participate in the association R. The value is a linguistic term such as *very high*, *high* or *low*. The fuzzy valuation τ is a confidence level of the fuzzy association, whose value is a fuzzy truth value.

For example, an important participant can identify his preference for locations and the intensity of his preference. Sometimes it is not certain whether a participant prefers a specific location or not. To model the relationship between important participants and locations to be an uncertain fuzzy association *prefer* will help the meeting scheduler system to resolve conflicts and make a most convenient schedule. A link attribute *preference* is associated with the association *prefer* to indicate the degree of preference. By stating that a link between *John* and *L102* is (*preference*,⟨*John*, *L102*⟩, *strong*,*very true*), we mean that it is very true that *John* strongly prefers the location *L102*.

Constraints with imprecise information are also allowed in fuzzy object-oriented models, called soft constraints. For example, the requirement "a meeting location should be as convenient as possible for all important participants" is modeled as a soft constraint on the associations *prefer* and *take place*.

Fig. 8 is a kernel model for the meeting scheduler system in fuzzy object-oriented models notations. Goals *MeetingRequestSatisfied*, *ParticipantDelegated* and *MaxConvenienceSchedule* are included in the kernel model.

5.3. Integrating alternatives

Since the rest of the requirements are conflicting with the requirements in the kernel, conflicts resolution is required in this stage. In order to handle conflicts, we explore the root of conflicts to lead us to the resolution of conflicts or even better the prevention of conflicts.

Table 2
Conflicts resolution for the conflict between the goals G_{MCS} and G_{AP}

Degree of impairment	Degree of importance	Solution
$I_{G_{AP}}(G_{MRS}) > I_{G_{MRS}}(G_{AP})$	$P(G_{MRS}) > P(G_{AP})$	Competition: selecting G_{MRS}
$I_{G_{AP}}(G_{MRS}) > I_{G_{MRS}}(G_{AP})$	$P(G_{MRS}) > P(G_{AP})$	Compromise
$I_{G_{AP}}(G_{MRS}) < I_{G_{MRS}}(G_{AP})$	$P(G_{MRS}) < P(G_{AP})$	Compromise
$I_{G_{AP}}(G_{MRS}) < I_{G_{MRS}}(G_{AP})$	$P(G_{MRS}) < P(G_{AP})$	Competition: selecting G_{AP}

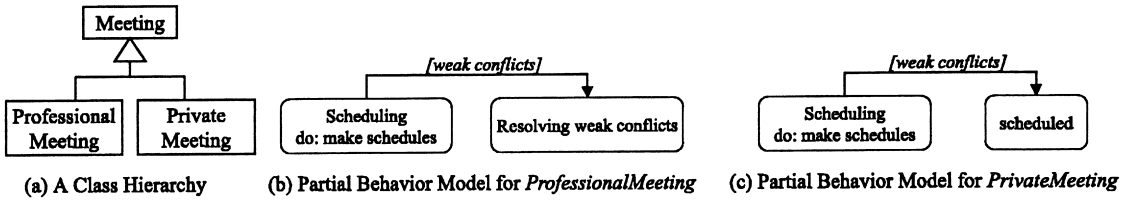


Fig. 11. Resolving conflicts from divergent expectations by specialization.

5.3.1. Root of conflicts

5.3.1.1. Competing resources. A conflict occurs when goals are competing with a mutually exclusive resource (i.e. the resource can not be shared by at the same time). In the library system, patrons wish to keep books as long as they need, whereas the librarians wish to keep books available in the library. A conflict arises because patrons and librarians are competing with the resource "books".

5.3.1.2. Divergent expectations. A conflict occurs whenever goals have divergent expectations on their common interest. For example, the goal *MaxProfits* from producers and the goal *MinCosts* from customers are both interested in the cost of a product. A conflict arises in that they have divergent expectations on the production's price. The common interest of goals may be an object's properties (e.g. the price of a product), an object's behavior, or a relationship between objects. In the meeting scheduler system, the goals *ParticipantPreferenceKnown* and *ParticipantPrivacy* are both interested in the relationship between participants and their preferences: whether a participant can know other participants' preferences. A conflict arises in that one goal expects that a participant can know other participants' preferences, whereas another goal does not.

5.3.1.3. Side effects. A conflict between goals arise whenever a use case to achieve a goal has a side effect on other goals. Conflicting goals in this case do not necessarily have a common interest. Referring to the meeting scheduler system, the goal *MaxConvenienceSchedule* is interested in the convenience of a meeting and *AppropriatePerformance* is interested in the elapsed time to make a schedule. Applying the use case *make a convenient schedule* to optimize the goal *MaxConvenienceSchedule* leads to the behavior of resolving weak conflicts (i.e. a meeting schedule is not preferred by all participants). On the other hand, applying the use case *keep appropriate performance* leads to the prohibition of resolving weak conflicts. It should be noted that the conflict is caused by the side effects of use cases, rather than divergent expectations on a common interest or competing with a common resource.

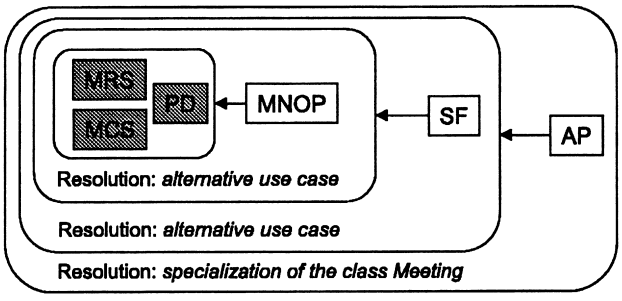


Fig. 12. Integrating alternative for the meeting scheduler system.

Another example is the conflict between *MeetingRequestSatisfied* and *SupportFlexibility*. The use case *accommodate important meetings* illustrates the process to optimize the goal *SupportFlexibility*: if a meeting's schedule is determined but conflicting with another meeting with a higher priority, it may be canceled in order to accommodate a more important goal. Note that this kind of conflict is tightly related to the use cases we constructed, and can be prevented by constructing alternative use cases (Section 5.3.2).

5.3.2. Conflicts resolution

A conflict can be resolved based on the causes of the conflict, the possibility of occurrence of the conflict, and the severity of the conflict consequences. Three resolution techniques are proposed to handle conflicts: to avoid the occurrence of conflicts by *prevention*, to reach a consensus between conflicting goals by *compromise*, and to achieve a goal without regard to another by *competition*.

5.3.2.1. Prevention. The best strategy of handling conflicts is to prevent conflicts from happening. Conflicts can be prevented by adding arbitrators or seeking alternative use cases, based on the causes of conflicts.

- To prevent a conflict from competing resources, a resource arbitrator may be introduced to distribute the resource. Using this heuristic rule, the conflicts between patrons and librarians can be prevented by introducing a book arbitrator.
- To prevent a conflict from side effects, an alternative use case is proposed to substitute for the original one. As mentioned in Section 5.3.1, the conflict between *SupportFlexibility* and *MeetingRequestSatisfied* occurs is due to the use case *accommodate important meetings* which may cancel a meeting. The use case described in Fig. 9 is an alternative use case to optimize the goal *SupportFlexibility* without having to cancel a meeting: a meeting

excludes all meetings' schedules out of its date range to avoid possible conflicts with other meetings.

5.3.2.2. Compromise. To reach a compromise, each goal involved in the conflict should make some concessions. Compromise can be reached by specializing an object class into disjoint subclasses with different associations linked to the corresponding subclasses.[6]

- Conflicts from competing resources can be resolved by specializing the resource or the objects competing with the resource. Consider the conflict between the goal *BookKeptAsLongAsNeeded* and *BookAvailableInLib*, we can resolve it by specializing the *Patron* (the object competing with the resource) into two subclasses: *Student* who must return books in two weeks and *Faculty* who can keep books as long as he/she needs, or specializing the *Books* (the resource) into *ReservedBook* and *UnreservedBook* (Fig. 10).
- Conflicts originated from divergent expectations on a relationship can be resolved by specializing the objects involved in the relationship. The conflict between *ParticipantPreferenceKnown* and *ParticipantPrivacy* can be resolved by specializing *Participant* into *PrivilegedParticipant* and *NonPrivilegedParticipant*, and restricting the privileged participants to have the right to access other participants' preferences.
- Conflicts due to divergent expectations on an object's behavior can also be resolved by specialization. For example, the conflict between *MaxConvenience Schedule* and *AppropriatePerformance* can resolved by specializing *Meeting* into two subclasses: *Professional Meeting* that must be scheduled as convenient as possible, and *PrivateMeeting* that must be scheduled as soon as possible (Table 2, Fig. 11).

5.3.2.3. Competition. Resolving conflicts by competition refers to satisfying one of the conflicting goals without regard to the others. The competition between goals depends on: (1) the importance degree of each goal involved in the conflict; and (2) the degree of impairment[7] to each goal when it cannot be satisfied. Supposing that G_1 conflicts with G_2, $I_{G_2}(G_1)$ is the degree of impairment to G_1 when G_2 is satisfied. $I_{G_2}(G_1) > I_{G_1}(G_2)$ denotes that achieving G_1 is more beneficial than achieving G_2 (since the *overall* impairment of achieving G_1 is less than that of achieving G_2).

In our example, if we use competition as a way of resolving the conflict between *MaxConvenientSchedule* and *AppropriatePerformance*, both the degrees of importance and impairment should be considered. The goal *MaxConvenientSchedule* is more important than *AppropriatePerformance* (Fig. 7). If $I_{G_{MCS}}(G_{AP}) > I_{G_{AP}}(G_{MCS})$, we prefer to achieve the goal *MaxConvenientSchedule* without regard to the goal *AppropriatePerformance*. If $I_{G_{MCS}}(G_{AP}) < I_{G_{AP}}(G_{MCS})$, a compromise strategy is recommended to resolve the conflict because none of the goals can be neglected.

Fig. 12 illustrates the steps involved in resolving conflicts when integrating the goals *MaxNumberOfParticicpants*, *SupportFlexibility* and *AppropriatePerformance*.

6. Conclusion

As was pointed by Lamsweerde [37], goal information should be captured in the requirements acquisition phase, which is useful for analyzing conflicting requirements and nonfunctional requirements. The proposed approach is spawned based on this belief to fuse the goal-oriented and object-oriented modeling techniques in requirements engineering.

Our approach offers several benefits in: (1) serving as a structuring mechanism to facilitate the derivation of use case specifications and objects model; (2) bridging the gap between the domain description and the system requirements, that is, the interactions between functional and nonfunctional requirements; (3) making easy the handling of soft requirements, and the analysis of conflicting requirements; and (4) extending traditional object-oriented techniques to fuzzy logic to manage different kinds of fuzziness that are rooted in user requirements.

Our future research plan will consider the following tasks: to investigate the worth-oriented domain advocated by Rosenschein et al. [31] to evaluate dynamic behaviors without actually executing the statechart diagrams; and to conduct a case study on using the proposed approach for modeling university timetabling of our university (National Central University).

Acknowledgements

This research was supported by the National Science Council (Taiwan) under grants NSC88-2213-E-008-006.

References

[1] B. Nuseibeh, A. Russo, J. Kramer, Restructuring requirements specifications for managing inconsistency and change: a case study, Proceedings of the 20th International Conference on Software Engineering, 1998.
[2] R. Darimont, A. van Lamsweerde, E. Leitier, Managing conflicts in goal-driven requirements engineering, IEEE Transactions on Software Engineering 24 (11) (1998) 908–926.
[3] A.I. Anton, Goal-based requirements analysis, Proceedings of the International Conference on Requirements Engineering, 1996, pp. 136–144.
[4] R. Balzer, N. Goldman, Principles of good software specification and

[6] A similar idea is also proposed in Ref. [2].
[7] A use case is said to impair a goal if the performance of the use case may result in the dissatisfaction of the goal.

their implications for specification languages, Proceedings of the IEEE Conference on Specifications of Reliable Software, 1979, pp. 58–67.
[5] R. Balzer, N. Goldman, D. Wile, Informality in program specifications, IEEE Transactions on Software Engineering 4 (2) (1978) 94–103.
[6] B.S. Blair, Object-Oriented Languages, Systems, and Applications, Pitman, London, 1991.
[7] B. Boehm, H. In, Identifying quality-requirement conflicts, IEEE Software 13 (2) (1996) 25–35.
[8] A. Borgida, S. Greenspan, J. Mylopoulos, Knowledge representation as the basis for requirements specification, Computer April (1985) 82–91.
[9] C. Souveyet, C. Rolland, C.B. Achour, Guiding goal modeling using scenarios, IEEE Transactions on Software Engineering 24 (12) (1998).
[10] A. Cockburn, Goals and use cases, Journal of Object-Oriented Programming 10 (7) (1997) 35–40.
[11] B. Dano, H. Briand, F. Barbier, Progressing towards object-oriented requirements specifications by using the use case concept, Proceedings of the International Conference on Requirements Engineering, 1996, pp. 450–456.
[12] A. Dardenne, A. van Lamsweerde, S. Fickas, Goal-directed requirements acquisition, Science of Computer Programming 20 (1993) 3–50.
[13] D. Dubois, H. Prade, Possibility Theory: an Approach to Computerized Processing of Uncertainty, Plenum, New York, 1988.
[14] D. Dubois, H. Prade, J.P. Rossazza, Vagueness, typicality and uncertainty in class hierarchies, International Journal of Intelligent Systems 6 (1991) 161–183.
[15] S. Easterbrook, Domain modelling with hierarchies of alternative viewpoints, Proceedings of the IEEE International Symposium on Requirements Engineering, 1993, pp. 65–72.
[16] C. Ebert, Putting requirement management into praxis: dealing with nonfunctional requirements, Information and Software Technology 40 (1998) 175–185.
[17] G. Eckert, P. Golder, Improving object-oriented analysis, Information and Software Technology 36 (2) (1994) 67–86.
[18] C. Francalanci, A. Fuggetta, Integrating conflicting requirements in process modeling: a survey and research directions, Information and Software Technology 39 (1997) 205–216.
[19] R. George, R. Srikanth, F.E. Petry, B.P. Buckles, Uncertainty management issue in the object-oriented data model, IEEE Transitions on Fuzzy Systems 4 (2) (1996) 179–192.
[20] I. Graham, Fuzzy objects: inheritance under uncertainty, Object Oriented Methods, Addison-Wesley, Reading, MA, 1994, pp. 403–433.
[21] I. Jacobson, Object-Oriented Software Engineering: a Use Case Driven Approach, Addison-Wesley, Reading, MA, 1992.
[22] J. Lee, J.Y. Kuo, New approach to requirements trade-off analysis for complex systems, IEEE Transactions on Knowledge and Data Engineering 10 (4) (1998) 00.
[23] J. Lee, K.F.R. Liu, W.L. Chiang, A fuzzy petri net based expert system for damage assessment of bridges, IEEE Transactions on System, Man, and Cybernetics, Part B: Cybernetics 29 (3) (1999) 350–370.
[24] J. Lee, N.L. Xue, Analyzing user requirements by use cases: a goal-driven approach, IEEE Software 16 (4) (1999) 92–101.
[25] J. Lee, N.L. Xue, J. Chen, Modeling imprecise requirements with fuzzy objects, Information Sciences: an International Journal (1999).
[26] J. Mylopoulos, L. Chung, B. Nixon, Representing and using nonfunctional requirements: a process-oriented approach, IEEE Transactions on Software Engineering 18 (6) (1992) 483–497.
[27] J. Mylopoulos, L. Chung, E. Yu, From object-oriented to goal-oriented requirements analysis, Communication of ACM 42 (1) (1999) 31–37.
[28] C. Potts, K. Takahashi, A.I. Anton, Inquiry-based requirements analysis, IEEE Software 11 (2) (1994) 21–32.
[29] W.N. Robinson, S. Fickas, Supporting multi-perspective requirements engineering, Proceedings of the First International Conference on Requirement Engineering, IEEE Computer Society Press, Silver Spring, MD, 1994, pp. 206–215.
[30] W.N. Robinson, V. Volkov, Supporting the negotiation, Communications of the ACM 41 (5) (1998) 95–102.
[31] J.S. Rosenschein, G. Zlotkin, Rules of Encounter, MIT Press, Cambridge, MA, 1994.
[32] T. Rowlett, Building an object process around use cases, Journal of Object-Oriented Programming 11 (1) (1998) 53–58.
[33] K.S. Rubin, A. Goldberg, Object behavior analysis, Communications of the ACM 35 (9) (1992) 48–62.
[34] J. Rumbaugh, Getting started: using use cases to capture requirements, Journal of Object-Oriented Programming 7 (5) (1994) 8–12.
[35] T.L. Saaty, Decision Making for Leaders: the Analytic Hierarchy Process for Decisions in a Complex World. Lifetime Learning, Atlanta, Georgia, 1982.
[36] S. Spaccapietra, View integration: a step forward in solving structural conflicts, IEEE Transactions on Knowledge and Data Engineering 6 (2) (1994) 258–274.
[37] A. van Lamsweerde, R. Darimont, P. Massonet, Goal-directed elaboration of requirements for a meeting scheduler problems and lessons learnt, Technical Report RR-94-10, Universite Catholique de Louvain, Departement d'Informatique, B-1348 Louvain-la-Neuve, Belgium, 1994.
[38] A. van Lamsweerde, E. Letier, Integrating obstacles in goal-driven requirements engineering, Proceedings of the 20th International Conference on Software Engineering, 1998.
[39] J. Vanweljenhuysen, Using dre to augment generic conceptual design, IEEE Expert 10 (1) (1995) 50–56.
[40] J. Vanwelkenhuysen, Quality requirements analysis in customer-centered software development, Proceedings of International Conference on Requirements Engineering, 1996, pp. 117–124.
[41] V. Wuwongse, M. Manzano, Fuzzy conceptual graphs, in: G.W. Minean, B. Moulin, J.F. Sowa (Eds.), Conceptual Graphs for Knowledge Representation, 1993, pp. 430–449.
[42] J. Yen, X. Liu, Approximate reasoning about properties of imprecise conflicting requirements, International Journal of Uncertainty, Fuzziness and Knowledge Based Systems 3 (2) (1995) 143–162.
[43] J. Yen, W.A. Tiao, A systematic tradeoff analysis for conflicting imprecise requirements, Proceedings of the Third IEEE International Symposium on Requirements Engineering, 1997, pp. 87–96.
[44] L.A. Zadeh, Test-score semantics as a basis for a computational approach to the representation of meaning, Literacy Linguistic Computing 1 (1986) 24–35.
[45] H.-J. Zimmermann, Fuzzy Set Theory and its Applications, Kluwer Academic, Boston, MA, 1991.