Discovering Early Aspects through Goals Interactions

Jonathan Lee*, Kuo-Hsun Hsu†, Shin-Jie Lee* and Wen-Tin Lee‡
*Department of Computer Science and Information Engineering
National Taiwan University, Taipei, Taiwan 106
Email: {jlee, jielee}@csie.ntu.edu.tw
†Department of Computer Science, National Taichung University of Education, Taichung, Taiwan 403
Email: glenn@mail.ntcu.edu.tw
‡Department of Software Engineering, National Kaohsiung Normal University, Kaohsiung, Taiwan 824
Email: wtlee@nknu.edu.tw

Abstract—Aspect-oriented approaches have resulted in a great impact on the processing of system properties during the development of software systems. However, a systematic way for eliciting early aspects from requirements is still underdeveloped to better integrate early aspects with the analysis results. As an attempt towards the investigation of the interactions among goals, scenarios, and early aspects, we proposed, in this work, a goal-driven approach to the discovery of early aspects by means of a bidding process that organizes goals into goal clusters for discovering aspectual candidates by exploring the interactions among goals and use cases. By introducing early aspects, the goal-driven approach can be further enhanced to deal with the crosscutting properties in the analysis stage of software development. The proposed approach is illustrated using the problem domains of meeting scheduler system.

I. INTRODUCTION

Pohl [1] emphasizes that one of the main objectives of requirements engineering process is to accomplish a common agreement on the views of the involved users. Many researchers [2]–[6] have recognized that the coupling of goal-based and user-centered approaches is a good way to reach such aim. The tenet of goal-based approaches focuses on why systems are constructed, which provides the motivation and rationale to justify software requirements [7]. The user-centered approach is useful in elicitation, analysis and requirements documentation [8].

Recently, aspect-oriented approaches have resulted in a great impact on the processing of the system properties, such as recurring properties or important stakeholder’s concerns during design and coding phases. By addressing these properties, such as performance, reusability, reliability, and scalability in an effective way, aspect-oriented approaches have attracted an increasing attention in the coding phase [9]–[14] as well as in the design and requirements phases [14], [15].

In this work, we propose a goal-driven approach to the discovery of early aspects through goals interactions by means of a bidding process as an attempt towards the analysis of software systems. Early aspects are defined as crosscutting concerns that are discovered in the early life cycle phase of a software system’s development, including requirements analysis, domain analysis and architecture design phases [16].

The proposed approach is an extension to our previous work on goal-driven use cases (GDUC) [5], [17]–[20], in which use cases are derived based on the analysis of goals interactions. By introducing early aspects, the goal-driven approach can be further enhanced to address crosscutting properties in the early stage of software development. There are two main features involved in the proposed approach (see Fig. 1):

1) To analyze a system by formulating goals and use cases. Goals are formulated through an extended goal structure, and use cases are established to achieve these goals.

2) To discover early aspects by means of a bidding process that organizes goals into goal clusters for discovering aspectual candidates by exploring the interactions among goals and use cases. To further elaborate, aspectual candidates are established by utilizing: (1) the bid function with similarity degrees among goals for bidding by means of goal clusters, and (2) the stability function with total interaction degrees for checking the validity of the bidding, and scattering degrees together with tangling degrees for balancing the total number of goal clusters, which serves as a check-and-balance in the proposed approach.

As pointed out in [21], using a benchmark could help examining the research contribution rigorously, and improving the
tools and technique being developed. Therefore, the meeting scheduler system [22] is chosen as an illustrative example throughout this paper to demonstrate the proposed approach as it has been adopted as a benchmark problem in requirements engineering [23]. The full set of requirements can be found in [22].

In the sequel, we first describe the related work on early aspects in Section II. We then discuss how to identify goals through the use of an extended goal structure based on our previous work in Section III. In Section IV, early aspects are discovered by the notion of goals interactions along with the bidding process. An enhancement of use case diagrams with early aspects is fully discussed in this section. Finally, we conclude by outlining benefits of the proposed approach and our future research plan in Section V.

II. RELATED WORK

Many works have been done in the area of aspect-oriented software development (AOSD). In what follows, we outline two dimensions of related work: one is research on early aspects [14], [15], [24]--[30], and the other is work on the relationships between early aspects and use cases [31]--[33].

A. Early Aspects

Aspects are behaviors that are tangled and scattered across a system [24]. In requirements documents, aspects may reveal themselves as interleaving and interdependent behaviors. Some aspects may be easily identified, as specifications of typical crosscutting behavior, while others may be more subtle and difficult to discover. Many recent studies have attempted to identify and apply the concept of aspects to the early stage of software development, in the hope of better addressing important stakeholder's concerns in the requirements analysis and design phases. Elisa Baniassad et al. [14] also emphasize the importance of identifying and managing requirements-level and architecture-level aspects instead of merely focusing on the implementation phase in the software life cycle. In their work, early aspects are identified and captured explicitly in requirements and architecture activities, and carried over the entire software development life cycle.

Theme-based approaches [24], [25] work on the fundamental assumption that two behaviors are related if they occur in the same requirement. Themes are organized into a tree hierarchy by identifying a set of action keywords from the requirements to inspect the relationships among behaviors in a requirements document. A theme is an element of design, and can also be viewed as a collection of structures and behaviors representing one feature. Multiple themes can be combined to form a system. Themes are classified as base themes and crosscutting themes. Base themes may share some structure and behavior with other base themes, while modeling these from their own perspective. Crosscutting themes have behaviors that overlay the functionality of the base themes. Crosscutting themes are treated as aspects. Theme/Doc [24] provides views of requirements specification in the requirements analysis phase, and exposes the relationships among behaviors in a system. Theme/uml [25] provides a models of the structure and behavior of each theme using standard UML at design phase. The disadvantage of theme-based approaches is the excessive effort in grouping the actions into larger themes and identifying aspects.

AORE [15] identifies candidate aspects by representing the relationships between concerns and stakeholder requirements in a contribution matrix based on the negative or positive effects of each aspect on others. Conflicts with stakeholders are resolved by prioritizing concerns. The requirements specification is then revised based on the new priorities. However, a systematic way to identify concerns as candidate aspects would be needed for developers to better evaluate candidate aspects.

Early-AIM [26] adopts corpus-based natural language processing techniques to help automate the identification and modeling of early aspects in a requirements document. The main aim of Early-AIM is to discern the candidate aspects in a document, irrespective of the document structure. EA-Miner [34] is the realization of this concept and offers automated support for identifying and visualizing early aspects from various requirements-related documents.

Aspect-oriented Multi-View Modeling [35] proposed to model a software system from multi-views by utilizing various notations, such as class diagrams, sequence diagrams, and state diagrams. By using the proposed reusable aspect models, it can support aspect dependency chains, which allows an aspect providing complex functionality to reuse the functionality provided by other aspects. However, the focus of the work is to reuse the existing aspects not to discover new aspects.

V-Graph [27] is a goal-oriented approach that adopts a goal model to reason about the interplay between functional and non-functional requirements. A V-graph, represents functional and non-functional requirements by capturing the contribution of goals, soft goals and tasks to the satisfaction of these requirements. Aspects are identified by detecting the tasks that contribute to some soft goals while also satisfying some functional goals. The drawback of this approach is that it only identifies soft goals as aspects without considering the rigid goals, which could also be the source of aspects.

ACE [30] seeks to identify crosscutting concerns through the application of automated clustering techniques. It utilizes a probabilistic model to compute the similarities between different requirements and uses a hierarchical algorithm to cluster similar requirements. Concerns represented by dominant terms were detected during an initial clustering phase, while those represented by less dominant terms were detected by removing away the dominant terms from requirements in subsequent phases. Generated clusters, candidate early aspects, where evaluated using metrics to measure their physical dispersion across the requirements specification and their level of interaction with other dominant concerns. Although ACE addresses the early aspect identification by means of clustering, it is still impeded by the following two problems: one is that the early aspect candidates found in result clusters have a coherent and imprecise problem, namely, the result clusters
may still contain a few unwanted or unrelated requirements; and the other is that a same concern addressed by various requirements may be overlooked due to the expression of the concern by different terms which may impact the similarities of requirements, which is addressed in [36] by Kit et al. with latent semantics analysis.

The main limitation of the related work of early aspects is that early aspects are mainly identified based on keywords across the whole system requirements or artifacts, which could hinder developers from focusing on major system functionalities since the identification of early aspects is accomplished by looking for crosscutting concerns across the whole target system and could possibly divert the crux of the system construction. The theory we proposed might be complex in computation; however, the practice has been simplified for the developers by providing a supporting tool. Comparing with the related work, the benefits of the proposed approach can be summarized as follows: (1) It makes easy for developers to identify early aspects by focusing only on the relationships between goals and use cases in a pairwise fashion. (2) The results delivered to developers are represented in a possibility-based manner, which allows developers to cut a threshold to determine how strong the likelihood is of an aspectual candidate containing an early aspect. (3) Through the use of the numerical representation of the relationships between goals and use cases, the relationships among goals can be more easily processed computationally. (4) The numerical relationships among goals are more informative and can be used as a basis for discovering early aspects by exploring the existence of common properties shard by goals.

B. Use Cases with Early Aspects

Many researchers have adopted use cases in requirements specification, analysis and design, and have attempted to adopt them as test beds for introducing early aspects into the requirements phase.

Sousa et al. [31] propose modeling crosscutting concerns as use cases, and presented a new relationship between use cases, called $\ll$ crosscut $\gg$. Information about the composition between a crosscutting use case and the use cases that it affects is described in a composition table that enables the join points to be defined, instead of in the base use case. The composition between an extension and a base use case can be fully non-invasive. A heuristic rule is provided to determine when to connect two use cases via a crosscutting relationship. That is, if the execution of use case B represents a course that needs to be applied in use case A, but (1) use case A does not depend on the execution of use case B to accomplish its primary goal, and (2) the use case B is not a specific course of use case A, and therefore it can be applied in other use cases, then use case B crosscuts use case A.

AspectU [32] is an aspect language for modularizing crosscutting concerns within a use-case model, and extends the use case model with support for modularizing crosscutting behavior. It introduces an aspect entity structured similarly to an aspect in AspectJ that comprises of pointcuts and advice.

Crosscutting in AspectU has three constructs: (1) join points are points in the model, (2) a pointcut is adopted to identify join points, and (3) advice is a means of affecting the behavior at the join points. These constructs define a join-point model that specifies the relationship of the crosscutting behavior to the underlying use-case model.

Moreira et al. [33] present a model to identify and specify quality attributes that crosscut requirements, including the integration of quality attributes, into the functional description at an early stage of the software development process. A model of the crosscutting behavior is devised to consider a quality attribute in a new stereotype use case, and made the base use cases include the stereotype use case.

Araujo et al. [37] adopt the concepts of overlapping, overriding and wrapping operators to compose functional requirements with aspects that crosscut non-functional requirements. A non-functional requirement crosses if it affects more than one use case. They model these aspects by defining new stereotype use cases, and adopted the stereotype relationships $\ll$ wrappedBy $\gg$ to connect those wrapped use cases to the crosscutting use case. This means that the behavior described by crosscutting use cases wraps the functional behavior described by these use cases.

Differ from the above methods, use cases in the proposed approach are used as a means to derived relationships among goals, based on which similarity and interaction degrees are established for grouping goals into goal clusters for identifying early aspectual candidates.

III. GOAL-DRIVEN USE CASE MODEL

As a starting point to identify early aspects, it is important to clarify the relationships among system functional and non-functional requirements. Use case driven analysis focuses the expression of requirements on users, starting from the very simple viewpoint that a system is built first and foremost for its users, which provides an important advantage to help manage complexity as it focuses on one specific usage at a time. Our previous work, goal-driven use case model (called GDUC model) [17], [19] that is developed upon the benefit of use case modeling to address the interactions among goals and use cases to provide valuable information in identifying, organizing and justifying software requirements, is adopted as a basis for the discovery of early aspects.

A. Goal Identification and Formulation

Goals identification plays a crucial role in the elicitation of software requirements. In [38], C. Rolland et al. proposed a goal structure to analyze the requirements based on a verb and its parameters. To better capture users’ intention of a system, an extension to the goal structure [39] is used to assist the capturing of software requirements, which was developed along with the following two dimensions: (1) classify a verb from two viewpoints: content and competence, to distinguish different types of requirements based on the notion of requirements satisfiability (e.g. a requirement that needs to be satisfied utterly or can be satisfied to a certain degree); and (2) add
two new types of parameters: view and constraints, to offer separate views in the analysis of the requirements.

In the extended goal structure, a goal is expressed as a clause with a verb and a number of parameters, where each parameter plays a different role with respect to the verb. To be more precise, the verb used in the requirements document will suggest a guideline that helps developers identify the types of the goals. For example, in a requirements document, if it states "Initiator plans a meeting with date and location by asking participants.", a goal can then be identified by using its verb "plan" to depict that the system should provide a function for an initiator to plan a meeting.

In terms of the parameters, there are four types in the extended goal structure: view, target, direction and constraints. The view concerns whether a goal is actor-specific or system-specific. An actor-specific view is an objective of an outside entity that uses a system; meanwhile, a system-specific view is a requirement for the services that a system provides. Targets are entities affected by a goal. There are two types of targets: object and results. An object is supposed to exist before the goal is achieved. Results can be of two kinds: (1) entities that do not exist before the goal is achieved; or (2) abstract entities that exist but are made concrete as a result of achieving the goal. The two types of directions, namely, source and destination, identify the to and from of objects to be communicated, respectively. Constraints represent the pre/post-condition that must be satisfied before or after achieving a goal. Invariant stands for conditions that always hold before and after achieving a goal.

A goal is thus represented as follows:

**Action:** [Actor, Target, Source, Destination, Condition, Competence]

Action is the verb from requirements documents that a goal intends to achieve, which could be either functional or non-functional. Actor refers to people who use this system or the system itself that performs the action. Targets are entities affected by the goal. Source, destination, and condition are optional. Two types of directions, source and destination, to indicate the initial and final location of objects to be communicated, respectively.

Condition serves two purposes: to describe the situation prior to or after performing an action and the invariant that the system must keep before and after performing the action. Competence can be either rigid or soft to show whether the goal must be satisfied utterly or to some extent.

In the meeting scheduler system, 15 goals are identified by utilizing the notion of the extended goal structure and are summarized as follows:

- **$G_{MP}$:** Plan: [initiator, meeting date and location, initiator, participants, $0$, rigid]
- **$G_{MN}$:** Replan: [initiator, meeting date and location, initiator, participants, support flexibility, soft]
- **$G_{SP}$:** Support: [system, conflicts resolution, initiator, participants, support flexibility, soft]
- **$G_{MT}$:** Manage: [system, interactions resolution, initiator, participants, as small as possible, soft]
- **$G_{PH}$:** Handle: [system, plan meetings, initiator, $0$, in parallel, rigid]
- **$G_{DRH}$:** Accommodate: [system, decentralized requests, initiator, $0$, been authorized, rigid]

Fig. 2. Goal-Driven Use Case Model of Meeting Scheduler System

- **$G_{KPC}$:** Maintain: [system, physical constraints, $0$, $0$, not to be broken, soft]
- **$G_{AP}$:** Provide: [system, performance, $0$, $0$, an appropriate level, soft]
- **$G_{RM}$:** Register: [participant, a meeting, participant, $0$, $0$, rigid]
- **$G_{DP}$:** Delegate: [participant, participation, participant, participants, $0$, rigid]
- **$G_{AR}$:** Accommodate: [participant, evolving data, participant, participants, $0$, $0$, soft]
- **$G_{WM}$:** Withdraw: [participant, the meeting, participant, $0$, $0$, soft]
- **$G_{KPS}$:** Enforce: [system, privacy rules, participants, $0$, $0$]
- **$G_{KR}$:** Support: [system, reusability, $0$, $0$, soft]
- **$G_{MU}$:** Maximize: [system, usability, $0$, $0$, non-experts, soft]

It is noted that not all verbs in requirements documents are identified as goals. The extended goal structure is provided as a guideline to assist developers to better capture software requirements based on verbs in requirements documents. It is up to the developers to identify verbs as goals directly from the requirements documents or to re-paragraph requirements documents for identifying goals.

B. Goal-Driven Use Case Diagram

After goals are identified, a goal-driven use case diagram is then established. In the goal-driven use case approach [19], each use case is viewed as a process that can be associated with a goal to be achieved, optimized, or maintained by the use cases. To start with, original use cases to guarantee that the target system will be at least adapted to the minimum requirements are first addressed. Each original use case is associated with an actor to describe the process to achieve an original goal which is rigid, actor-specific, and functional. Building original use cases by investigating all original goals will make the use case model satisfy at least all actors’ rigid and functional goals. To extend the original use cases to take into account various types of goals, extension use cases are created in various situations: optimizing/maintaining a soft goal, achieving a system-specific goal, or achieving a non-functional goal.

In the meeting scheduler system example, use cases are established according to the system goals identified, and the
goal-driven use case diagram is then constructed as shown in Figure 2. A detail discussion of GDUC can be found in [19].

IV. EARLY ASPECTS DISCOVERY

Early aspects are defined as crosscutting concerns that are discovered in the early life cycle phase of a software system’s development, including requirements analysis, domain analysis and architecture design phases [13], [16]. As a continuation of our previous work on goal-driven use case model, we focus our attention in this work on how to discover early aspects based on the notion of goals interactions in the requirements analysis phase.

An early aspect is a concern that crosses modules or components in the target system, identifying clusters that aggregate these modules or components could be helpful in discovering early aspects [30]. The discovery of early aspects is therefore based on the notion of goal clusters as a result of the bidding process to locate groups of goals in which goals in each group response to a set of use cases similarly or, in other words, goals in each group behave similarly.

The early aspect discovery process is summarized below (see Figure 3.)

Step 1 Evaluate the relationships among goals and use cases. The relationships between use cases and goals are analyzed by investigating the satisfaction degree of the goals, and calculating the cooperative and conflicting degrees between goals with a membership function after the use cases are performed, which is described in section IV-A.

Step 2 Obtain goals relationships, including similarity degrees for grouping goals into goal clusters in bid function and interaction degrees for evaluating the validity in the stability function, for the bidding process in step 3. The relationships between goals are calculated by utilizing the proposed formula described in section IV-B for grouping goals.

Step 3 Establish goal clusters with the bidding process that engages similarity degree among goals for bidding by goal clusters, total interaction degree for checking the validity of the bidding, and scattering degree together with tangling degree for balancing the total number of goal clusters. In the other words, goal clusters are established by applying bid and stability functions in a check-and-balance manner. Details of the bidding process is described in section IV-C.

A. Evaluate the Relationships among Use Cases and Goals

The first step in the early aspect discovery process is to evaluate the relationships among use cases and goals. In the analysis of goals and use cases, each goal is achieved, ceased, impaired, optimized, or maintained by its directly associated use case. In addition to the directly achieved, ceased, impaired, optimized, or maintained relationships, the effects of a use case to all goals, called side effects, are also considered.

By investigating all the effects including side effects among goals and use cases, relationships between goals and use cases, that is, achievement degrees, can be determined. In the evaluation, an achievement of a goal is rated from -5 to 5 to represent an achievement degree to the goal while performing a use case. The score is given by domain experts based on a rating table (see Table I as suggested in [40].) In Table I, 5 means the goal can be fully satisfied by the use case; -5 means the goal is fully denied by the use case; and 0 means the use case does not have any effect on the goal. Detail explanation of the rating of satisfaction degree can be found in Table I.

To better model the relationships among use cases and goals, two membership functions are proposed for representing the satisfying and denying degrees of goals with respect to use cases from two viewpoints: satisfying and denying, and are defined as:

**Definition 1:** Let $u_{Sat}(U_i, G_j)(x)$ be a membership function for describing the satisfying degree with respect to a rating $x$ of the base set $X$ (i.e. from -5 to 5). Then, $u_{Sat}(U_i, G_j)(x)=0.2x$, if $x > 0$. $u_{Sat}(U_i, G_j)(x)$ represents the degree that goal $G_j$ is satisfied by use cases $U_i$ wrt score $x$ and $0 \leq u_{Sat}(U_i, G_j)(x) \leq 1$.

**Definition 2:** Let $u_{Den}(U_i, G_j)(x)$ be a membership function for describing the denying degree with respect to a rating $x$ of the base set $X$ (i.e. from -5 to 5). Then, $u_{Den}(U_i, G_j)(x)=0.2x$, if $x < 0$. $u_{Den}(U_i, G_j)(x)$ represents the degree that goal $G_j$ is satisfied by use cases $U_i$. 

![Fig. 3. Illustration of the Process in Discovering Early Aspects](image)

<table>
<thead>
<tr>
<th>Score</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>-5</td>
<td>The goal is fully satisfied after the use case is performed</td>
</tr>
<tr>
<td>-3</td>
<td>The goal is largely satisfied after the use case is performed</td>
</tr>
<tr>
<td>-1</td>
<td>The goal is partially satisfied after the use case is performed</td>
</tr>
<tr>
<td>0</td>
<td>The goal is not affected after the use case is performed</td>
</tr>
<tr>
<td>1</td>
<td>The goal is partially denied after the use case is performed</td>
</tr>
<tr>
<td>3</td>
<td>The goal is largely denied after the use case is performed</td>
</tr>
<tr>
<td>5</td>
<td>The goal is fully denied after the use case is performed</td>
</tr>
<tr>
<td>2, 4, 2, 4</td>
<td>Represent the degrees between scores listed above</td>
</tr>
</tbody>
</table>
wrt score $x$ and $0 \leq u_{Den}(U_k, G_j)(x) \leq 1$.

For example, to analyze the relationships among goals and use cases in the meeting scheduler system, we examine the relationships among goals and use cases in a pairwise manner based on Table I. Goal $G_{MP}$ (meeting planned) can also be achieved by the use case $U_{RAM}$ (replan a meeting); therefore, a score of 4 is given to indicate that the goal $G_{MP}$ can be largely to fully satisfied by the side effect of performing $U_{RAM}$. By applying the membership functions, we can obtain the satisfying degree between $G_{MP}$ and $U_{RAM}$ as $u_{Sat}(U_{RAM}, G_{MP})(4) = 0.8$ and denying degree $u_{Den}(U_{RAM}, G_{MP})(4) = 0$, which is paired as a tuple – $(u_{Sat}(U_{RAM}, G_{MP})(4), u_{Den}(U_{RAM}, G_{MP})(4)) = (0.8, 0)$. Goal $G_{MI}$ (min. interactions) is partially to largely denied by use case $U_{RAM}$ (replan a meeting) since replanning a meeting could increase the number of communication among participants, which results in a negative number of rating. -2. By applying the membership functions to the relationships among goals and use cases, Table II can be obtained in a tuple form.

B. Obtain Goals Relationships

In order to facilitate the bidding of a goal cluster for its target goals in the bidding process, two important factors are further explored: one is the similarity degree of goals for grouping a goal clusters, and the other is the interaction degrees for evaluating the validity of the grouping of goals from a system-wise point of view.

The similarity degree between two goals is defined as the summation of variances that each use case contributes to these two goals:

**Definition 3:** Let $Similarity(G_i, G_j)$ be a function for representing the similarity degree between two goals. Then,

$$Similarity(G_i, G_j) = 1 - \left( \sum_{k=1}^{n} \frac{|u_{Sat}(U_k, G_i)(x)| - |u_{Sat}(U_k, G_j)(x)| + |u_{Den}(U_k, G_i)(x) - u_{Den}(U_k, G_j)(x)|)}{S_{i,j}} \right)$$

where $i \neq j$, $n$ is the number of use cases in the system, $-5 \leq x \leq 5$, and $S_{i,j}$ is an adjusting factor, which is the counts of satisfying/denying degrees of $G_i$ and $G_j$ with respect to $U_k$ that are not equal to 0.

The similarity degrees are normalized, that is, only those whose satisfying/denying degrees of $G_i$ and $G_j$ with respect to $U_k$ are not equal to 0 will be considered. This is to exclude out those relationships that are rated as irrelevant.

For example, to compute the similarity degree of $G_{MP}$ and $G_{MR}$,

$$Similarity(G_{MP}, G_{MR}) = 1 - (2.0/7) = 0.714$$

where 7 is the adjusting factor, which is the counts of satisfying/denying degrees of $G_{MP}$ and $G_{MR}$ with respect to use cases whose values are not equal to 0, as shown in figure 4.

Table III shows the pair-wise similarity degrees in the meeting scheduler system. In this table, the larger the value is, the more similar the two goals are. For example, $G_{MR}$ is more similar to $G_{MP}$ ($Similarity(G_{MR}, G_{MP}) = 0.71$) than $G_{SF}$ ($Similarity(G_{SF}, G_{MP}) = 0.5$) is.

![Fig. 4. Illustration of Computing Similarity](image)

**Definition 4:** Let $Cooperative_{U_k}(G_i, G_j)$ be a function for representing the cooperative degree of two goals $G_i$ and $G_j$ wrt a specific use case $U_k$. Then,

$$Cooperative_{U_k}(G_i, G_j) = \left( u_{Sat}(U_k, G_i)(x) \cap u_{Sat}(U_k, G_j)(x) \right) \cup \left( u_{Den}(U_k, G_i)(x) \cap u_{Den}(U_k, G_j)(x) \right)$$

where $\cap$ stands for fuzzy AND representing the intersection operation, $\cup$ stands for fuzzy OR representing the union operation, and $-5 \leq x \leq 5$.

**Definition 5:** Let $Conflicting_{U_k}(G_i, G_j)$ be a function for representing the conflicting degree of two goals $G_i$ and $G_j$ wrt a specific use case $U_k$. Then,

$$Conflicting_{U_k}(G_i, G_j) = \left( u_{Sat}(U_k, G_i)(x) \cap u_{Sat}(U_k, G_j)(x) \right) \cup \left( u_{Den}(U_k, G_i)(x) \cap u_{Den}(U_k, G_j)(x) \right)$$

where $\cap$ stands for fuzzy AND representing the intersection operation, $\cup$ stands for fuzzy OR representing the union operation, and $-5 \leq x \leq 5$.

An interaction relationship between two goals at the system level can then be obtained by applying fuzzy union operation to the results of Definitions 4 and 5 with respect to all use cases.

**Definition 6:** Let $Sys(G_i, G_j)$ be a function for representing an interaction degrees between two goals at the system level. Then,

$$Sys(G_i, G_j) = \bigcup_{k=1}^{n} \left( Cooperative_{U_k}(G_i, G_j) - Conflicting_{U_k}(G_i, G_j) \right)$$

where $n$ is the number of use cases in the system.

Table IV shows the interaction degrees between goals of the meeting scheduler system.

C. Establish Goal Clusters with Bidding Process

In the making of a target system, it is usually desirable to aggregate goals with a high cooperative degree, which makes the system conform to the principle of high cohesion in software design. On the other hand, a goal cluster with a large number of goals would probably violate the principle of low coupling, which may lead to a system design flaw.

Based on this belief, the bidding process begins with the bidding of goals to form a goal cluster, followed by the checking of total interaction degrees of all goal clusters, and finally through the use of scattering and tangling degrees to validate each bid.
TABLE II

A TUPLE FORM REPRESENTATION OF RELATIONSHIPS BETWEEN GOALS AND USE CASES OF THE MEETING SCHEDULER SYSTEM

<table>
<thead>
<tr>
<th>GMP</th>
<th>GMR</th>
<th>GSP</th>
<th>GMR</th>
<th>GDR</th>
<th>GKR</th>
<th>GDR</th>
<th>GEM</th>
<th>GEM</th>
<th>GMR</th>
<th>GPR</th>
<th>GMR</th>
<th>GMR</th>
</tr>
</thead>
<tbody>
<tr>
<td>U_MP</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>U_MR</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>U_SR</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>U_M</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>U_PR</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>U_SR</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>U_M</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>U_PR</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>U_SR</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>U_M</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

TABLE III

SIMILARITY DEGREES AMONG GOALS OF THE MEETING SCHEDULER SYSTEM

<table>
<thead>
<tr>
<th>GMP</th>
<th>GMR</th>
<th>GSP</th>
<th>GMR</th>
<th>GDR</th>
<th>GKR</th>
<th>GDR</th>
<th>GEM</th>
<th>GEM</th>
<th>GMR</th>
<th>GPR</th>
<th>GMR</th>
<th>GMR</th>
</tr>
</thead>
<tbody>
<tr>
<td>U_MP</td>
<td>x</td>
<td>0.9</td>
<td>0.9</td>
<td>0.8</td>
<td>0.8</td>
<td>0.7</td>
<td>0.7</td>
<td>0.4</td>
<td>0.4</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>U_MR</td>
<td>0.9</td>
<td>x</td>
<td>0.8</td>
<td>0.8</td>
<td>0.7</td>
<td>0.7</td>
<td>0.4</td>
<td>0.4</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>U_SR</td>
<td>0.8</td>
<td>0.8</td>
<td>x</td>
<td>0.8</td>
<td>0.8</td>
<td>0.7</td>
<td>0.7</td>
<td>0.4</td>
<td>0.4</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>U_M</td>
<td>0.8</td>
<td>0.8</td>
<td>0.8</td>
<td>x</td>
<td>0.8</td>
<td>0.8</td>
<td>0.7</td>
<td>0.7</td>
<td>0.4</td>
<td>0.4</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>U_PR</td>
<td>0.7</td>
<td>0.7</td>
<td>0.7</td>
<td>0.7</td>
<td>x</td>
<td>0.7</td>
<td>0.7</td>
<td>0.4</td>
<td>0.4</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>U_SR</td>
<td>0.7</td>
<td>0.7</td>
<td>0.7</td>
<td>0.7</td>
<td>0.7</td>
<td>x</td>
<td>0.7</td>
<td>0.7</td>
<td>0.4</td>
<td>0.4</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>U_M</td>
<td>0.7</td>
<td>0.7</td>
<td>0.7</td>
<td>0.7</td>
<td>0.7</td>
<td>0.7</td>
<td>x</td>
<td>0.7</td>
<td>0.7</td>
<td>0.4</td>
<td>0.4</td>
<td>0.2</td>
</tr>
<tr>
<td>U_PR</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>x</td>
<td>0.4</td>
<td>0.4</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>U_SR</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>x</td>
<td>0.4</td>
<td>0.4</td>
<td>0.2</td>
</tr>
<tr>
<td>U_M</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>x</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>U_PR</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>x</td>
<td>0.2</td>
</tr>
<tr>
<td>U_SR</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>x</td>
</tr>
<tr>
<td>U_M</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
</tr>
</tbody>
</table>

**Definition 7:** Let $\text{Bid}(\text{GoalCluster}_k, G_i)$ be a function for representing the bidding value from a goal cluster $\text{GoalCluster}_k$ to a goal $G_i$. Then,

$$\text{Bid}(\text{GoalCluster}_k, G_i) = \sum_{j=1}^{m} \text{Similarity}(G_j, G_i)/m$$

where $m$ is the size of goal cluster $\text{GoalCluster}_k$ (i.e., the number of goals in goal cluster $\text{GoalCluster}_k$).

In the bid function, a goal cluster with the largest bid to a goal is claimed as the winner of a bid in accordance with the definition of the similarity degree. To validate the bid, a stability function is proposed and is controlled by three factors: total interaction degrees, scattering degrees and tangling degrees.

- total interaction degrees defined as a summation of interaction degrees described in section IV-B in all goal clusters,
- scattering degree defined as the number of goals in a goal cluster, and
- tangling degree defined as the number of goal clusters that a goal participate.

The bid is granted to a goal cluster if the inclusion of a goal to the goal cluster increases the value of the total interaction degree of all goal clusters. On the contrary, if the value of a total interaction degree of all goal clusters remains intact after the bidding, the scattering degree and tangling degree are considered as a second criterion to control the changes of the total number of goal clusters in order to balance the stability. The system is moving toward a more stable state as the total number of goal clusters converges in the bidding process, which suggests that the bid should not be granted if the scattering degree and the tangling degree increase as a result of the bidding. The definition of total interaction degree of all goal clusters $\text{InteractionDegree}(\text{systemstate})$, scattering degree $\text{ScatteringDegree}(\text{systemstate})$, and tangling degree $\text{TanglingDegree}(\text{systemstate})$ are elaborated as follows:

**Definition 8:** Let $\text{InteractionDegree}(\text{systemstate})$ be a function for representing the total interaction degree as a summation of interaction degrees in all goal clusters. Then,

$$\text{InteractionDegree}(\text{systemstate}) = \sum_{k=1}^{m} \sum_{i=1}^{n} \sum_{j=1}^{n} S_{kp}(G_i, G_j)$$

where $m$ is the number of goal clusters in a $\text{systemstate}$, $n$ is the number of goals in the $k$th goal cluster, and $i \neq j$. A system state is a snapshot of the grouping of goals, i.e. the goal clusters, at one specific time.

**Definition 9:** Let $\text{ScatteringDegree}(\text{systemstate})$ be a function for representing the scattering degree as a summation
of the number of goals in all goal clusters in the system. Then,

\[ \text{ScatteringDegree(systemstate)} = \sum_{k=1}^{m} \text{size}(\text{GoalCluster}(k)) \]

where \( m \) is the number of goal clusters in a systemstate, \( \text{size}(\text{GoalCluster}(k)) \) is number of goals in \( \text{GoalCluster}(k) \).

Definition 10: Let \( \text{TanglingDegree(systemstate)} \) be a function for representing the tangling degree as the summation of the number of goal clusters that every goal participates in the system. Then,

\[ \text{TanglingDegree(systemstate)} = \sum_{i=1}^{n} \text{participate}(\text{Goal}(i)) \]

where \( n \) is the number of goals, \( \text{participate}(\text{Goal}(i)) \) is the number of goal clusters in a systemstate that \( \text{Goal}(i) \) participates in.

The initial system state in the beginning of the bidding process is established by having each goal as a goal cluster of its own. Goals in the system are bid and granted one at a time with one exception that a goal cluster can’t bid a goal that is already inside the goal cluster.

The check-and-balance process is applied to determine the next system state by checking for the largest total interaction degree of all goal clusters, and then balance the changes of the total number of goal clusters by means of evaluating scattering and tangling degrees among various system states. The system state with a higher total interaction degree among all goal clusters is chosen as the next system state. Otherwise, scattering and tangling degrees are calculated as a second criterion for selecting the next system state whose scattering and tangling degrees remain unchanged. The bidding process algorithm for discovering the aspectual candidates is as follows:

**Algorithm 1:** (Bidding Process Algorithm)

1) Initialize
   a) For \( n \) goals to be bid, create \( n \) sets of goal clusters with each of these goal clusters contains a unique goal to form the initial system state, \( S_{n0} \), and denoted these goal clusters as \( g_{c_1}, g_{c_2}, \cdots, g_{c_m} \), and the size of a goal cluster \( \text{size}(g_{c_i}) \) is defined as the number of goals in a goal cluster \( g_{c_i} \).
   b) Place \( n \) goals in a bidding pool, and denote the number of goals in the bidding pool as \( \text{poolsize} \).

2) Bid
   a) For each goal \( g_{c_i} \) in the bidding pool, \( 1 \leq t \leq \text{poolsize} \), do steps 2b to 2d in parallel with \( g_{c_i} \):

   b) Mark \( g_{c_i} \) as processed.
   c) For these \( n \) goal clusters \( (g_{c_1}, g_{c_2}, \cdots, g_{c_n}) \), compute the bid function \( \text{Bid}(g_{c_i}, g_{c_j}) \) where \( g_{c_i} \subset g_{c_j} \) and \( 1 \leq k \leq n \).
   d) Set goal clusters with highest value of the bid function as the set of winning candidates, denoted as \( wgc_1, wgc_2, \cdots, wgc_m \) and \( \text{Bid}(wgc_i, wgc_j) \leq \text{Bid}(g_{c_i}, g_{c_j}) \), where \( 1 \leq i \leq m \), \( 1 \leq k \leq \text{poolsize} \).

3) Check-and-Balance
   a) For each winning candidate \( wgc_i \), calculate total interaction degrees of the three possible next system states \( SS_{i+1}, SS_{i+1}' \) and \( SS_{i+1}'' \).
   b) If there are more than two total interaction degrees are equal, proceed to step 3e otherwise proceed to step 3d.
   c) Choose the one with largest value of the sum of the two functions \( (\text{ScatteringDegree}(i), \text{TanglingDegree}(i)) \) for the three possible next system states \( SS_{i}, SS_{i}' \) and \( SS_{i}'' \) and denote it as \( CSS_y \), \( 1 \leq y \leq \text{poolsize} \). Proceed to step 3e.
   d) Choose the system state with the largest total interaction degree and denote it as \( CSS_y \), \( 1 \leq y \leq \text{poolsize} \). Proceed to step 3e.
   e) For candidate system states \( CSS_y \), select \( CSS_y \) such that \( \text{InteractionDegree}(CSS_y) \)
   f) do the following steps for each \( CSS_y \) in concurrent mode.
   i) Remove goal \( g_{c_i} \) from the bidding pool.
   ii) If there are goals in the bidding pool, proceed to step 2.
   iii) Otherwise put the \( CSS_y \) in the set of a final system state pool (FSSP) and proceed to step 4.

4) Finalize
   a) If all candidate system states are processed, calculate the number of the occurrence of a specific goal cluster in all final system states and denoted as total number of occurrence of a specific goal cluster \( t \) (number of system states in FSSP) + 100%.

Based on this algorithm, goals are grouped into goal clusters as a starting point for discovering early aspects. Each goal cluster with a frequency indicates the odds of goals sharing a same common property, i.e. the early aspect.

A supporting tool is constructed for developers as an assistant to input the relationships between goals and use cases in a pairwise manner. The judgment of the effect from a use case to a goal is determined by developers and could be subjective. However, the judgment made in the proposed approach is in a pairwise manner, which, we believe, can greatly reduce variations made by different developers. The final results (as in Figure 5) are represented in a table listing all goal clusters along with goals inside and the frequency of the occurrence of each goal cluster appearing in all final system states. For example, a goal cluster with 33% means that at the end of the bidding process, in which a total number of 100 final system states are derived, goals in the goal cluster are grouped
Aspects are discovered through performing the use case "Keep Performance" and are denoted as:

- Aspect\_SF: \{Handle Flexibility, \{G\_MP, G\_MR, G\_SF\}\}
- Aspect\_AP: \{Keep Performance, \{G\_MI, \ G\_AP\}\}
- Aspect\_MNMP: \{Max. Number of Meeting, \{G\_MHP, G\_DRH\}\}
- Aspect\_VP\_C: \{Verify Program Correctness, \{G\_SR, G\_KPC\}\}
- Aspect\_MSI: \{Maintain System Integrality, \{G\_EPR, G\_WM, G\_MU, G\_RM, G\_DP, G\_AED\}\}

The benefit of the proposed approach is that it makes easy for developers to identify early aspects by focusing only on the relationships between goals and use cases in a pairwise fashion. Furthermore, the results delivered to developers are represented in a possibility-based manner, which allows developers to cut a threshold to determine how strong the likelihood is of an aspectual candidate containing an early aspect.

In the meeting scheduler example, the result shows that seven goal clusters with various occurrences in frequency in all final system states are listed and suggested to developers.

V. CONCLUDING REMARKS

Aspect-Oriented software development (AOSD) has emerged as a significant development and maintenance approach to software engineering. It provides explicit means to model important stakeholders’ concerns that tend to crosscut multiple system components. The early discovery of concerns results in untangled and non-scattered designs and codes; and therefore reduces the cost of implementation and enhance maintainability.

In this work, we propose a goal-driven approach to the discovery of early aspects through goal clustering by means of a bidding process as an attempt towards the analysis of software systems, in which two main features are devised: (1) evaluating the relationships among goals and use cases to obtain the degrees of similarity and interaction relationships among goals; and (2) discovering early aspects through the exploration of interactions among goals, use cases, and early aspects, which engages similarity degree among goals for bidding by goal clusters, total interaction degree for checking the validity of the bidding, and scattering degree together with tangling degree for balancing the total number of goal clusters in the bidding process in a check-and-balance manner.

The key strength of the proposed approach can be viewed in two ways: one is from the viewpoint of using numerical representation of relationships among goals, and the other is from the developers’ viewpoint. The benefits of using numerical representation of relationships among goals are twofold: (1) the relationships among goals can be more easily processed computationally, and (2) the numerical representation relationships among goals are more informative and can be used as a basis for discovering early aspects by exploring the existence of common properties shared by goals.

Therefore, it makes easy for developers to identify early aspects by focusing only on the relationships between goals and use cases in a pairwise fashion, which, we believe, can greatly reduce variations made by different developers, and efforts on looking for crosscutting concerns or common properties across the whole system. The results delivered to developers
are represented in a possibility-based manner, which allows developers to cut a threshold to determine how strong the likelihood is of an aspectual candidate containing an early aspect. Moreover, by setting a different threshold, the proposed approach gives an experienced requirements engineer a prism into all the possible potential aspectual candidates to prevent any neglect or overlook that may occur based purely on intuition.

Our future research plan includes: (1) to extend AOP with the capability of accommodating early aspects addressed in the design phase; and (2) to make an automatic transformation of early aspects identified in the requirement phase from modeling to aspect-oriented programming.

ACKNOWLEDGMENT

This research was sponsored by National Science Council, Taiwan. (NSC 100-2221-E-002-001-MY3 and NSC-101-2221-E-142-004).

REFERENCES


