RAAP: A Requirements Analysis and Assessment Process Framework for Component-Based System

(Invited Paper)

Richard Lai
Department of Computer Science and Computer Engineering,
La Trobe University, Melbourne, Australia
Email: lai@cs.latrobe.edu.au

Sajjad Mahmood
Information and Computer Science Department,
King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia
Email: smahmood@kfupm.edu.sa

Shaoying Liu
Faculty of Computer and Information Sciences,
Hosei University, Tokyo, Japan
Email: sliu@hosei.ac.jp

***Abstract*— Success of a Component Based System (CBS) depends heavily on the selection of the right components. In reality, components are usually designed for general purposes and finding the ideal ones is often very difficult. The CBS requirements process is hence more complicated than the conventional approach. In this paper, we present a Requirements Analysis and Assessment Process (RAAP) for CBS that can provide quantitative information and guidelines for stakeholders to evaluate the suitability of the components for a given set of requirements. Subsequently, they will be able to select the most appropriate components that best satisfy their needs, taking into consideration the risks involved and the conflicts that could arise as a result of selecting certain components earlier in the process. RAAP consists of three phases: (i) requirements characterization which elicits user requirements; (ii) top-down analysis which calculates the degree of satisfaction of a component and the amount of risks involved; and (iii) trade-off analysis which identifies and resolves the potential conflicts in requirements after certain components have been selected. We also present an application of RAAP to the Seven Eleven Japan system.**

***Index Terms*— Component based system, software metrics, requirements analysis, risk assessment**

I. Introduction

Component Based System (CBS) requirements analysis and component selection is widely recognized as an interrelated process, which plays a central role in overall CBS development. Software literature [1-5] shows that CBS success depends on the ability to select suitable components. An inappropriate component selection can lead to adverse effects such as short-listed components hardly fulfilling the required functionalities, and introducing extra costs in integration and maintenance phases [2]. Individual components usually provide capabilities that might not satisfy all system requirements and some of them may be unnecessary for a given system. This reduces the chance of a good match between a component and stakeholder requirements. Therefore, it is difficult to find a supplier who can meet all stakeholder requirements [6]. In this paper, we adopt Szyperski's definition of a component [7]: "A software component is a unit of composition with contractually specified interface and explicit context dependencies only. A software component can be deployed independently and is subject to composition by a third party".

Recent research [3, 5, 8] suggests that CBS requirements analysis does not need to be completed before starting component selection. However, current CBS development approaches have a requirements process that is based on strict requirements definition. This implies that either candidate components have to be eliminated because they do not meet the stated requirements or they will need to be changed significantly in order to satisfy such restrictive requirements [3]. It reduces the scope of requirements negotiation [8] and makes it difficult to evaluate how components fit in with overall system requirements [9].

Fundamental to CBS success is the need for a collaborative process whereby both stakeholders and candidate components balance the conflicting interests between what is needed and what is available [10, 11]. This collaborative process needs to focus on how to share knowledge between stakeholders and components and facilitate negotiation of individual interests during component selection. Component selection involves a continuous process of requirements negotiation and for an effective requirements negotiation, it is necessary to analyze the impact of the negotiation [12]. This involves balancing a component's satisfaction against the involved risks. Risk assessment is another important attribute affecting overall CBS development as it provides a basis for comparing candidate components by focusing on their risk profiles.

In this paper, we present a Requirements Analysis and Assessment Process (RAAP) framework for CBS that can provide quantitative information and guidelines for stakeholders to evaluate the suitability of the components for a given set of requirements. Subsequently, they will be able to select the most appropriate components that best satisfy their needs, taking into consideration the risks involved and the conflicts that could arise as a result of selecting certain components earlier in the process. RAAP consists of three phases: (i) requirements characterization which elicits user requirements; (ii) top-down analysis which calculates the degree of satisfaction of a component and the amount of risks involved; and (iii) trade-off analysis which identifies and resolves the potential conflicts in requirements after certain components have been selected. We also present an application of RAAP to the Seven Eleven Japan system [13].

The contribution of our work is the development of a collaborative process, which provides a platform to quantitatively analyze individual interests and come to an agreement in a conflicting scenario. We present the notion of specifying requirements at two abstraction levels so as to minimize early component exclusion and accept functionality limitations that cannot be met. We propose a requirements analysis algorithm that enables a systematic evaluation of requirements based on requirements priorities. Further, we have developed satisfaction and risk metrics, which enable a system analyst to better, understand what is needed and what is available. These metrics allow us to derive the distribution of components that are appropriate for the resolution of conflicts in different scenarios. Our work provides a technique to investigate proposed resolutions and evaluate the risks associated with each proposal. Further, it helps in assessing conflicts that may arise when different components are integrated into a CBS.

II. Related work

Off the Shelf Option (OTSO) method [14] is a process that directly addresses the issue of component identification. The process is based on hierarchical evaluation, which decomposes the requirements into a set of hierarchical criteria. It facilitates a systematic, repeatable and requirement-driven component identification and selection process. OTSO provides a systematic component selection process and uses an analytic hierarchy process to provide support in decision-making. However, it does not discuss how to conduct requirements acquisition and how to compare candidate components.

Procurement Oriented Requirements Engineering (PORE) [1] is based on an iterative process of requirements acquisition and component evaluation. It uses a template-based approach for refining the candidate component list until a suitable component has been selected. It proposes to use strict criteria to analyze how candidate components satisfy requirements. However, there is a lack of specific detail on how requirements are used in the evaluation process. Furthermore, there is a lack of assessment of the compliance process.

COTS Aware Requirements Engineering (CARE) [4] is a goal-oriented requirements engineering approach which highlights the importance of keeping requirements flexible. CARE classifies requirements as: stakeholder requirements and component requirements, with emphasis on reducing the gap between these two requirements groups. However, CARE lacks clear guidelines for handling possible mismatches between the stakeholder requirements and the candidate component features.

Kotonya et al. [8] propose a method for CBS requirements engineering based on the notion of viewpoints. It is a service-oriented requirements approach that interleaves the process of requirements with component verification, negotiation and planning. However, this approach lacks guidelines on how to rank requirements and on verification of requirements against component features.

Alves et al. [3] propose an approach to evaluate components in terms of how well they match customer requirements and provides a conflict management framework to identify the components based on resolution proposals and risk evaluation. The modeling of goals starts with the elicitation of high-level goals that represent the stakeholders' concerns. Each goal is then further divided into sub goals and represented by a graph structure using AND/OR tree. However, there is no detailed discussion on how to perform risk evaluation. Further, there is no quantifiable process for evaluating candidate components based on their satisfaction and risk assessments.

TABLE I.
COMPONENTS FOR THE SEJ SYSTEM

Component Name	Description
Smart Scan	Provides ability to read barcodes
IPWorks	Provides ability to write connected applications
PowerTCP	Supports data communication
SocketTools	Integration of internet communication functionality
SuperCom	Provides ability for serial communication
Socket Wrench	Provide TCP/IP networking functionality
Component Space	Data communication within windows and web applications
VSView	Provides ability to format text
XtraReport	Provides ability to create reports
TrueDBInput	Provides ability to acquire and format user inputs
PureComponentEntrySet	Enables data customization
ComponentOneInput	Enables data customization
Xceed	Provides ability to support front-end and back-end application development
InputPro	Enables creation of data entry interfaces
Dxperience	Enables creation of windows forms

III. Seven Eleven Japan System

In this section, we describe our case study, the Seven Eleven Japan (SEJ) system [13] requirements. We selected the SEJ system for the following reasons: (1) it is an application of significant size; (2) availability of SEJ research literature [13, 15]; and (3) its application as a validation of requirements techniques [16-18]. SEJ system manages a national franchise of independently owned convenience stores and uses its software system to leverage information to coordinate a supply chain of business partners. It ensures the stores are stocked with precisely the products that consumers want and when they want them [18]. The stakeholders of the SEJ system include product supplier, delivery centers, franchise stores and SEJ customers [18].

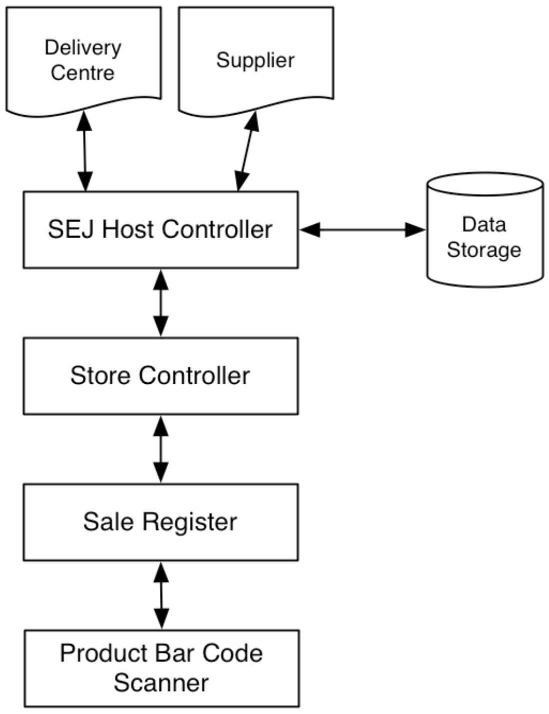


Figure 1. The SEJ System Architecture.

Figure 1 shows the high-level system architecture of the SEJ system. It consists of two main parts: 'SEJ Host Controller' and 'SEJ Store Controller'. The 'SEJ Store Controller' interacts with end customers via 'sale register' and 'product scanner'. It communicates with the 'SEJ Host Controller' to pass information about a store's inventory levels, customer behavior etc. The 'SEJ Host Controller' shares information with the delivery centers to coordinate the supply chain. It also orders products from suppliers for delivery to the stores. Further, it provides all the administration and reporting functionalities of the SEJ system.

A. Components for the SEJ Software System

In our case study, we identified eleven requirements from the SEJ literature and used component source[1] as the component repository. We have selected component source as the repository because it provides over 1,700 components grouped into 90 categories. We assume that the SEJ software system is developed on a Microsoft .Net platform [19] and we need to buy a single license for the components. Table 1 shows the list of components selected for the SEJ system.

[1] www.componentsource.com

IV. The RAAP framework

CBS development is a complex and risk-prone process [20] which needs a flexible requirements analysis technique that provides an opportunity for both stakeholders and component vendors to reach a mutually acceptable agreement. We believe that a CBS requirements model needs to address three key issues: understanding stakeholder requirements and component features; quantifying alternatives based on satisfaction and risk analysis; and balancing a component's satisfaction against the involved risk during conflict resolution. By analogy with Mendonca et al.'s measurement framework [21], our Requirements and Analysis and Assessment Process can be summarized the framework shown in Figure 2.

RAAP consists of three phases: namely, requirements characterization, top-down analysis and trade-off analysis. The first phase - requirements characterization - starts with a process to elicit stakeholder requirements. We propose the use of a goal-oriented requirements engineering process to specify stakeholder demands. After this process, requirements are ranked according to their priority, view and matching potential. The ranked requirements are analyzed and represented as a directed graph to specify the relationship between requirements. The second phase - top-down analysis - presents a metrics hierarchy to quantify requirements that match component features. It uses satisfaction and risk metrics to select suitable components for each requirement. The degree of satisfaction measure involves the evaluation of a component's syntactic properties and configuration constraints against a given requirement. The risk is measured as a function of the complexity and severity assessment of a component. The third phase - trade-off analysis - is executed to identify and resolve the potential conflicts in requirements. A conflict between requirements is detected by analyzing the relationship between requirements, and potential resolutions are generated using a set of heuristic rules. Finally, we propose a set of resolution selection rules for a trade-off between requirements and components.

Figure 2 also shows the information flow and control flow of our proposed approach. The control flow (represented as solid lines) is determined by the interaction between the phases. A requirement acts as a unit of analysis and the top-down and the trade-off analyses is applied incrementally. The requirements characterization and associated algorithm act as a prerequisite for the top-down and trade-off analyses. The top-down and trade-off analyses phases also interact with each other when the degree of satisfaction and risk metrics calculated during the top-down analysis phase are used for relevance index and negotiation at the trade-off phase.

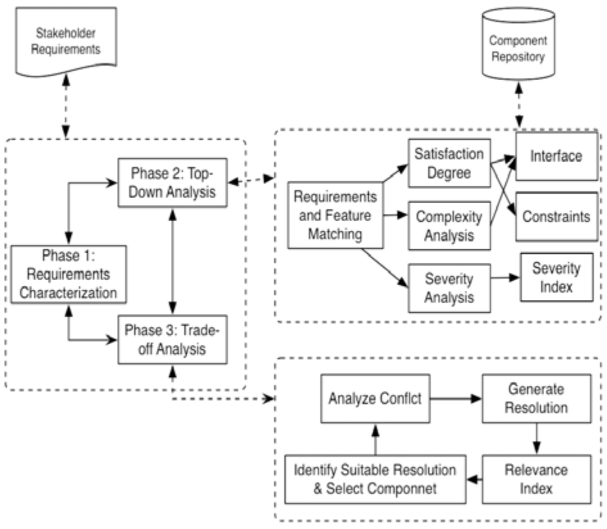


Figure 2. The RAAP Framework.

V. Requirements Characterization

Requirements characterization is used to elicit, rank and represent stakeholder requirements, and determine how they relate to each other. Requirements characterization consists of three steps and we discuss them in detail as follows:

Step 1: Requirements Elicitation - CBS requirements need to start with a less specific and more flexible definition [3]. A flexible requirements definition increases the probability of finding matching components as requirements either exclude the use of components or require a large modification because they are less stringent. We propose to elicit stakeholder demands using goal - scenario coupling approach [22]. In this paper, we adopt the goal definition [23]: "an objective the composed system should meet". Similarly, we adopt the scenario definition [24]: "a proposed specific use of the system". In the goal scenario coupling approach, goal discovery and scenario authoring are complementary activities. After goal discovery, scenario authoring is initiated, followed by goal discovery. The "goal-discovery, scenario-authoring" sequence is repeated to incrementally elicit requirements [22].

The goal-scenario approach [22] is used to elicit CBS requirements because it allows requirements to be represented at different levels of abstraction. This provides a systematic process for refining high-level requirements into objectively measurable sub-requirements. The aim of these abstraction levels is to identify component alternatives that satisfy stakeholders' requirements. We propose to organize CBS requirements, as shown in Figure 3, into two hierarchy levels, namely, high-level requirements and concrete-level requirements. The high-level requirements cover the overall business objectives of an organization. The concrete-level requirements represent a set of services that can be used to achieve the high-level requirements. These abstraction hierarchy levels are used to elicit requirements into concrete sub-requirements, which can be objectively measured against the component features. Requirements are elicited into these two levels based on refinement rules defined in [22].

High-Level Requirements (HLR): **-** The aim of the HLR is to identify an initial set of minimum CBS requirements which correspond to a given business objective. This initial set of requirements represents a possible methodology for fulfilling the overall business vision of a CBS. This emphasis on HLR means the selection process relies less on pre-emptive decisions about the candidate component. HLR captures the requirements as a pair <Gh, Sh> where Gh is a high-level goal and Sh is a high-level scenario. A high-level goal in our approach represents a business objective, and the associated high-level scenario represents the process for achieving a high-level goal.

Concrete-Level Requirements (CLR): - At the CLR, the focus is on refining high-level requirements into sub-requirements that can be used to quantify candidate components. High-level requirements are refined by considering the interaction between the system and the users. These interactions represent a possible method for achieving a high-level goal defined at HLR. CLR specifies requirements as a pair <Gc, Sc> where Gc is a concrete-level goal and Sc is a concrete-level scenario. A concrete-level goal expresses the manner of realizing a high-level goal. The associated concrete-level scenario describes the flow of interactions between a system and its user to fulfill the concrete-level goal.

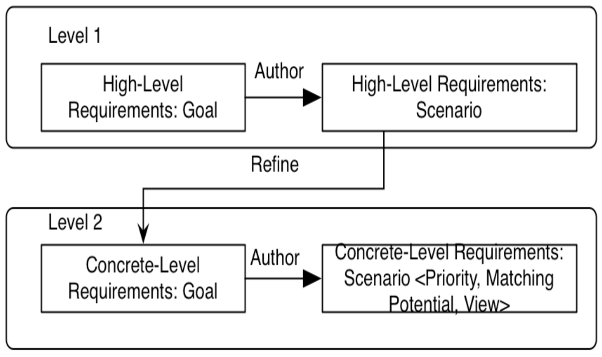

Level 1
High-Level Requirements: Goal
Author
High-Level Requirements: Scenario
Refine
Level 2
Concrete-Level Requirements: Goal
Author
Concrete-Level Requirements: Scenario <Priority, Matching Potential, View>

Figure 3. Requirements Elicitation Model.

Furthermore, it is important to classify requirements to enable distinction between core and peripheral requirements. By analogy with Lee et al. [25, 26] facet classification model, we classify requirements under three facets: priority, matching potential and view. This classification helps in performing an objective measure between requirements and component features.

Priority - The facet priority represents stakeholders' desire for a CLR satisfaction. We propose to classify each CLR priority as mandatory, very important, important or optional. A mandatory CLR represents a minimum set of requirements that need to be satisfied in order for the system to succeed. Very important CLR represents requirements that ensure significant functionality of the system. Important CLR represents requirements that ensure sufficient functionality of the system. Similarly, optional CLR represents requirements that are desirable but do not affect the success of the system.

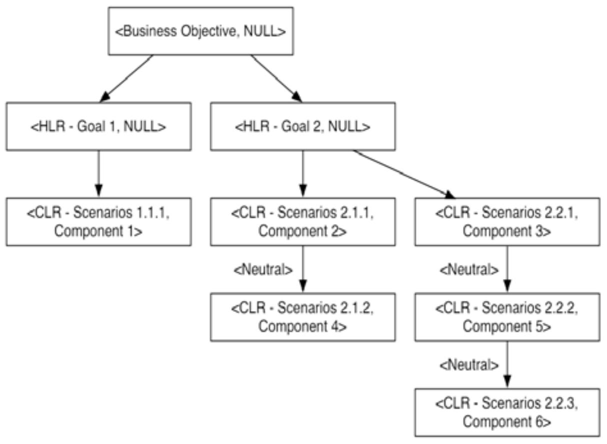


Figure 4. Requirements Graph.

Matching Potential - The facet matching potential represents the CLR prospect of finding a matching component in the repository. We define the matching potential as the percentage of the number of candidate components with the potential to match the CLR to the total number of components in a repository. Low value of matching potential indicates a high number of constraints associated with the CLR realization. High value of matching potential indicates a low number of constraints associated with the realization of CLR. The value of matching potential is computed as follows:

$$\text{Matching Potential} = \sum_{i=0}^{n} CC_i / TC \qquad (1)$$

where *n* is the total number of categories in the repository with the potential to match the CLR and *CC* is the number of components classified in each of these categories. Finally, *TC* is the total number of components in the repository.

View - The facet view classifies a CLR as either actor specific or system specific. Actor specific CLR are the objectives of the users who interact with the proposed system. System specific CLR are the objectives of entities that do not interact directly with the system but will hold a stake in the system requirements [8]. A System CLR provides a mechanism for expressing organizational goals and constraint requirements that apply to the system as a whole.

Step 2: Requirements Ranking - In the second step, each CLR is ranked according to its priority, view and matching potential. We rank a CLR in a descending order of priority. Requirements with the same priority are classified based on their view. Actor specific requirements get precedence over system level requirements. Requirements with the same priority and view are further classified based on their matching potential values. Requirements with a small 'matching potential' value are ranked higher than requirements with a bigger 'matching potential' value because we aim at selecting a component for a requirement that has limited number of choices rather than those with a higher number of choices. The reason is that by selecting components for requirements with a bigger number of choices after component selection for requirements with a smaller number of choices, we have a better chance of providing alternatives if a conflict occurs.

Step 3: Requirements Graph - In the third step, we construct a directed graph called Requirements Graph (RG), to represent the ranked requirements and the relationship between them, as shown in Figure 4.

Definition 1: Requirements Graph: A RG is defined as a tuple {N,E, Root}, where {N,E} is a directed graph; and Root is the first (starting) node; N is a set of nodes in a graph, with $N = \{ n_i \}$, i = 1 …. |N|; and E is a set of edges in the graph, with $E = \{ e_i \}$, i = 1 … |E|.

Definition 2: Node "n": $n \in N$ represents requirements R_i, which is defined by a tuple $<R_i, C_i>$, where R_i is the ith requirements of a CBS; and C_i is the selected component for the requirement R_i.

Definition 3: Root Node "nr": $nr \in N$ is the first (starting) node of a RG and is defined as a special node that represents the overall customer's business objective and has NULL value for the selected component for the overall business objective requirement.

Definition 4: Directed Edge "e": $e \in E$ represents the association between requirements at consecutive RG levels. It is denoted by <impact>, which represents the 'Requirements Associations' relationship between n_i and n_j at two consecutive RG levels.

Definition 5: Requirements Associations "RA$_{ij}$": RA_{ij} is the association relationship such that node n_i can have either 'Negative' or 'Neutral' impact on n_j. A 'Negative' association represents a conflict between two requirements and a "Neutral' association represents the situation that there is no conflict between them.

Table 2 shows the characterization algorithm that defines a set of activities to construct an RG. We discuss the algorithm in detail below:

1. For a given set of HLR goals {HLR1, HLR2, … HLRn}, each HLR goal is inserted as a first level node in a RG such that HLR1 will be the leftmost first level node and HLRn will be the rightmost first level node. Since the component selection process starts for CLR, each first level node in RG is represented as <HRL, NULL> where NULL signifies no component is selected for the requirement.
2. For each CLR-scenario associated with a HLR-goal, requirements characterization starts with analyzing a CLR relationship with leaf nodes of RG (step 9 in Table 2). The relationship between a CLR and leaf node is analyzed based on the rules defined in the section of Trade-off Analysis (step 1: conflict identification). This relationship investigation considers only the leaf nodes associated with the same HLR as that of a given CLR.

3. For each neutral relationship between a CLR and a leaf node, a suitable component is identified (step 17 in Table 2) based on satisfaction and risk metrics, discussed in detail in the section of Top-down Analysis. First, candidate components' satisfaction and risk values are calculated. Next, these components are ranked according to their satisfaction to risk ratio. Finally, a component with the highest value of satisfaction to risk ratio is selected.
4. For a *negative* relationship between a CLR and a leaf node, we use the resolution generation rules (as discussed in the section of Trade-off Analysis) to identify possible resolutions to the problem of a conflicting scenario. Candidate components are assigned a relevance index that is used to calculate suitability of each resolution. The resolution with the highest value of suitability is used to select a component.
5. Finally, a tuple <CLR, selected component> is inserted into the RG as a leaf node.
6. Repeat steps 2 to 4 for all requirements in the stack.

TABLE II.
THE REQUIREMENTS CHARACTERIZATION ALGORITHM

```
1.  Algorithm ( )
2.  {
3.      Insert Root node.
4.      Insert all HLR at level one of the RG.
5.      Push CLR into the Stack from lowest to highest ranking.
6.      While Stack Not Empty do
7.      {
8.          Pop a CLR from the stack ;
9.          If (characterize (CLR, Leaf Node of RG) ! = SUCCESS)
10.             Return FAILURE;
11.     Else
12.         Return SUCCESS;
13.     }
14. }
15. characterize (CLR, Leaf Node of RG)
16. {
17.     If (Identify_Relationship (CLR, Leaf Node of RG) ! = CONFLICT)
18.     {
19.         Select_Component (CLR);
20.         Insert CLR as new leaf node <CLR, selected component>;
21.         Return SUCCESS;
22.     }
23.     Else If (Trade-off (CLR, Leaf Node of RG) != FAILURE)
24.     {
25.         Generate resolutions for the conflicting scenario;
26.         Identify suitable resolution;
27.         Select_Component (CLR);
28.         Insert CLR as new leaf node <CLR, selected component>;
29.         Return SUCCESS;
30.     }
31.     Else
32.         Return FAILURE;
}
```

A. Applying Requirements Characterization to the SEJ System

The first step in requirements characterization is to elicit requirements. We start eliciting SEJ system requirements by identifying the SEJ business objective. The overall business objective of the SEJ system is ' to create a chain of convenience stores where you can find a solution for any of your daily life problems at hours when needed' [13]. There are five high-level requirements, which represent the overall objectives of SEJ software system. The high-level requirements of SEJ software system is to reduce loss of costumers, maximize use of limited floor space, minimize unsold perishable goods, shorten inventory turnover time and stock products according to changing consumer needs. In this paper, we will consider two high-level requirements, namely, HLR 4 - shorten inventory turnover time; and HLR 5 - stock products according to changing consumer needs.

Figure 5 shows requirements elicitation for the HLR4. The HLR4-goal 'shorten inventory turnover' describes a method of fulfilling the SEJ business objective. The associated HLR4–scenario describes the flow of interactions among the SEJ entities to achieve the HLR4–goal. Based on refinement rules defined in [22], we identify that HLR4–scenario consists of three basic interactions: (i) 'co-ordinate supply chain network'; (ii) 'provide stock ordering decision support'; and (iii) 'control store inventory'. Next, we consider each HLR–scenario interaction as a CLR-goal and elicit an associated CLR–scenario to help achieve the goal. CLR–scenario actions or events are identified by considering the interactions between the system and its users. For example, the CLR-goal to 'provide stock ordering decision support to stores' is realized by three events: 'display sale performance reports', 'display SEJ stock order recommendations' and 'accept or update stock order recommendation'.

Similarly, HLR5 'stock products according to changing customer needs' is another method of fulfilling overall SEJ business objectives. We identify that the HLR5–scenario consists of three basic interactions: (i) 'correlate purchase data with customer profile'; (ii) 'deliver stocks to store in time'; and (iii) 'update store inventory in real time'. In this paper, we consider only the first HLR–scenario interaction to show the conflicting requirements situations. The first HLR5–scenario interaction is considered as a CLR–goal and we elicit associated CLR–scenario to help achieve the goal. The CLR–goal to 'correlate purchase data with customer profile' is realized by two events: 'store customer profile'; (ii) 'maintain customer privacy'; and (iii) 'update customer profile information'.

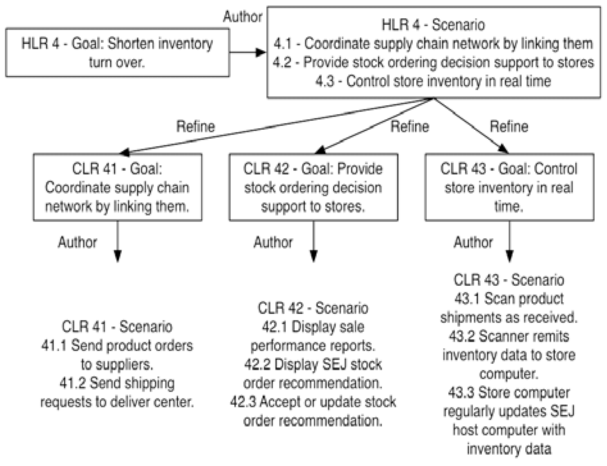

HLR 4 - Goal: Shorten inventory turn over.
Author
HLR 4 - Scenario
4.1 - Coordinate supply chain network by linking them
4.2 - Provide stock ordering decision support to stores
4.3 - Control store inventory in real time
Refine
Refine
Refine
CLR 41 - Goal: Coordinate supply chain network by linking them.
CLR 42 - Goal: Provide stock ordering decision support to stores.
CLR 43 - Goal: Control store inventory in real time.
Author
Author
Author
CLR 41 - Scenario
41.1 Send product orders to suppliers.
41.2 Send shipping requests to deliver center.
CLR 42 - Scenario
42.1 Display sale performance reports.
42.2 Display SEJ stock order recommendation.
42.3 Accept or update stock order recommendation.
CLR 43 - Scenario
43.1 Scan product shipments as received.
43.2 Scanner remits inventory data to store computer.
43.3 Store computer regularly updates SEJ host computer with inventory data

Figure 5. Requirements Elicitation for the SEJ System

In the second stage of requirements characterization, we rank each interaction of the CLR scenario. Each CLR scenario interaction is assigned priority, matching potential and view. For example, CLR 42.1 has priority, matching potential and view as ‘very important’, ’15.52’ and ‘system’, respectively. The priority and view is assigned by a domain expert. Furthermore, we identify that CLR 42.1 can be matched by the components classified in ‘reporting’, ‘database reporting’, ‘charting and graphics’, ‘spreadsheet’ and ‘database management’ categories of component source repository. The number of components in these categories is 38, 49, 82, 42 and 53, respectively. We calculate ‘matching potential’ of CLR 42.1 as ((38 + 49 + 82 + 42 + 53)/ 1700) * 100) = 15.52, where 1700 is the total number of components in the repository. Similarly, all the remaining CLR scenario interactions are assigned priority, matching potential and view values, as shown in Table 3.

TABLE III.
REQUIREMENTS CLASSIFICATIONS FOR THE SEJ SYSTEM

HLR	CLR	Description	Priority	Matching Potential	View
HLR 41	CLR 41.1	Send product order to supplier	Important	16.82	Actor
	CLR41.2	Send shipping requests to delivery center.	Important	16.82	Actor
HLR 42	CLR 42.1	Display sale person order reports.	Very important	15.52	System
	CLR42.2	Display SEJ stock order recommendation.	Very important	8.25	System
	CLR 42.3	Accept or update stock order recommendation.	Important	11.47	System
HLR 43	CLR 43.1	Scan product shipments as received.	Mandatory	13.76	Actor
	CLR 43.2	Scanner remits inventory data to store computer.	Mandatory	8.35	System
	CLR 43.3	Store computer regularly updates SEJ host computer with inventory data.	Mandatory	16.52	System
HLR 51	CLR 51.1	Store customer profile	Mandatory	5.58	Actor
	CLR 51.2	Maintain customer privacy	Mandatory	5.58	Actor
	CLR 51.3	Update customer profile information	Mandatory	16.52	System

Table 4 shows CLR scenario interactions ranking based on the ranking rules defined in section 5. For example, CLR 43.1, CLR 43.2 and CLR 43.3 are all mandatory requirements; hence, the ranking process starts with these requirements. Since CLR 43.1 has ‘actor’ view, it is ranked higher than CLR 43.2 and CLR 43.3. CLR 43.2 and CLR 43.3 are both mandatory and system specific requirements, so we take their ‘matching potential’ into account to rank them. Since ‘matching potential’ of CLR 43.2 is lower than CLR 43.3, we rank CLR 43.2 higher than CLR 43.3. Requirements with lower ‘matching potential’ are ranked higher than requirements with lower ‘matching potential’ because we first want to select components for a requirement that has more limited choices than those with a higher number of choices. We argue that selecting components for requirements with larger choices provides a better chance of providing alternatives in cases of conflict.

Finally, in the third step of requirements characterization, we construct the SEJ requirements graph, as shown in Figure 6. The SEJ requirements graph construction starts with inserting a root node, which represents the overall SEJ business objective. Next, we insert the HLR as first level nodes in the graph. Based on SEJ requirements ranking, as shown in Table 4, we start the component selection process for HLR 4, by considering the highest ranked requirement CLR 43.1.

TABLE IV.
REQUIREMENTS RANKING FOR THE SEJ SYSTEM

HLR	CLR	Rank
HLR 4	CLR 43.1	1
	CLR 43.2	2
	CLR 43.3	3
	CLR 42.2	4
	CLR 42.1	5
	CLR 41.1	6
	CLR 41.2	7
	CLR 42.3	8
HLR 5	CLR 51.1	1
	CLR 51.2	2
	CLR 51.3	3

We identity that ‘SmartScan’ is the most suitable component to fulfill CLR 43.1. The component is identified using satisfaction and risk metrics, as discussed in section 6. Further, CLR 43.2 is selected for identifying suitable component. We analyze its relationship with CLR 43.1 to identify any potential conflicts between the requirements. Since there is no conflict between these two requirements, we identify ‘IPWorks’ as the suitable component for CLR 43.2. Similarly, ‘TrueDBInput’ component is identified as the suitable component for CLR 51.1. CLR 51.2 is the next highest ranked requirement and its relationship is analyzed with CLR 51.1. Since there is a conflicting relationship between

CLR 51.1 and CLR 51.2, we identify 'PureComponentsEntrySet' as a suitable component for CLR 51.2. This conflict identification and subsequent component selection is based on trade-off analysis which is discussed in detail in section 7.

VI. Top down analysis

The top-down approach uses matching criteria between component features and requirements to calculate a component's satisfaction degree and associated risk. The concrete-level requirements are analyzed against candidate component features to obtain satisfaction metrics. We use severity analysis and complexity metrics to derive component risk assessment. We choose a suitable component from a range of candidate components at each level of RG by defining a decision metrics denoted by D (RG, i). Decision metrics value is defined as a percentage of component satisfaction degree and associated risk, as shown in equation 2.

D (RG, i) = (Satisfaction (C) / Risk (C)) x 100 (2)

where i is the depth number where a particular requirement occurs in the requirement graph. We discuss satisfaction degree and risk metrics in detail as follows.

A. Statisfacation Degree

A key challenge in CBS requirements engineering is to reconcile stakeholders' demands against available component capabilities. The component matching involves an evaluation of the degree to which a component satisfies a requirement. The first step is to identify component attributes that contribute to the satisfaction assessment. A component is characterized according to its characteristics and context dependencies [27]. Fundamental to a component are its characteristics that specify the functionality provided. These characteristics define only the individual elements of a component, mainly in syntactic terms. However, a component is subject to configuration dependencies on its use. These dependencies are both on individual elements as well as on the relationship among the elements [28]. Thus, it is essential for a component user to understand the constraints so as to be able to use it properly.

Recently, Cechich et al. [29] defined a measure for early detection of component functional suitability as a function of number of compatible functionality, missing functionality and additional functionality. We believe that in addition to these three attributes, it is important to consider the impact of missing and additional features introduced by candidate components. Further, configuration constraint is another primary attribute, which directly affects the satisfaction degree of a component with reference to a requirement. After a detailed analysis, we identify a set of attributes, as shown in Table 5, which need to be considered during satisfaction degree measurement.

TABLE V.
CRITERIA FOR ASSESSING SUITABILITY OF A COMPONENT

Item	Criterion	Measurement scale
C1	Compatible features – CF	{1,2….n} where n is the number of compatible features.
C2	Missing features – MF	{1,2….n} where n is the number of the CLR scenarios not met by a component.
C3	Impact of missing features – IMF	{negligible, little, moderate, considerable, great}
C4	Additional features – AF	{1,2….n} where n is the number of additional features provided by components and not required by the requirement.
C5	Impact of additional features – IAF	{negligible, little, moderate, considerable, great}
C6	Adaptation effort required – AER	{negligible, little, moderate, considerable, great}
C7	Adhere to system architecture – ASA	{poor, fair, good, very good, excellent}
C8	Non development cost – NDC	{within budget, <5%, 5 – 7 %, 7 – 10 %, > 10%}

We commence the satisfaction measure by considering CLR.. The satisfaction of a requirement is realized at the CLR scenario level and each CLR scenario interaction is objectively measured against component features. Candidate component features are identified from their interface specification and information provided in information brochures, evaluation downloads, user documentation, tutorials and manuals. Table 5 shows the checklist of selection criteria that is used to measure the satisfaction degree of a component with respect to a requirement (CLR).

We classify our checklist into two main groups: component characteristics and configuration constraints. The first group measures the suitability of a component from the perspective of its characteristics. It consists of five selection criteria: Compatible Features (CF), Missing Features (MF), Impact of Missing Features (IMF), Additional Features (AF) and Impact of Additional Features (IAF). CF measures the number of component features that contribute to fulfilling the requirement. Table 6 shows matching metrics where CLR scenario interactions are listed in rows and component features are arranged in columns. A '✓' at the intersection of a CLR scenario interaction and a component feature indicates that the corresponding component feature satisfies it. Similarly, MF and AF are the number of CLR scenario interactions not fulfilled by a component and the number of additional features introduced by the component, respectively. We introduce the notion of IMF, which quantifies the impact of the missing features as 'negligible, little, moderate, considerable or great'. Similarly, IAF quantifies the impact of extra features introduced by the component as "negligible", "little", "moderate", "considerable" or "great'. However, it is important to note that due to human-centric nature of requirements engineering, a domain expert will play a key role in determining how requirements are perceived and how component information are analyzed.

TABLE VI.
MATCHING METRICS

	Interaction 1	Interaction 2	Interaction …..	Interaction n
Feature A1	✓			
Feature A2		✓		✓
…….				
Feature A_n			✓	

The second group evaluates the compatibility of a component from the perspective of context dependency. First, we consider Adaptation Effort Required (AER), which quantifies the amount of adaptation development required to use the component. It is important to consider adaptation cost as, although it usually accounts for less than half the total CBS development effort, the effort per line of adaptation averages three times the effort per line of traditional development code [30]. Adhere to System Architecture (ASA) analyses the compatibility of a component regarding its operating system for development, architecture of product and pre-requires. Non Development Cost (NDC) assesses the cost associated with licensing administration [30].

TABLE VII.
UTILITY FUNCTION FOR IMF, IAF AND AER

	Great	Considerable	Moderate	Little	negligible
IMF	-5	-3	1	3	5
IAF	-5	-3	1	3	5
AER	-5	-3	1	3	5

TABLE VIII.
UTILITY FUNCTION FOR ASA

	Poor	Fair	Good	Very Good	Excellent
ASA	-5	-3	1	3	5

TABLE IX.
UTILITY FUNCTION FOR NDC

	Within Budget	< 5%	5 – 7 %	7 – 10 %	> 10 %
NDC	5	3	1	-3	-5

First, we define a utility function uf1, based on multi-attribute utility theory [31] to classify IMF, IAF and AER. The function uf1 is transformed into the unit interval {-5, 5}, as shown in Table 7. Similarly, we define utility function uf2 to classify ASA. The function uf2 is transformed into the unit interval {-5, 5}, as shown in Table 8. Finally, we define utility function uf3 based on US Department of Defense (DOD) risk management guide [32] to classify NDC. The function uf3 is transformed into the unit interval {-5, 5} , as shown in Table 9. We define characterizes measure for a component c, denoted as CM (c), as:

$$CM (c) = CF + (MF \times IMF) + (AF \times IAF) \quad (3)$$

We define configuration constraints measure for a component c, denoted as CCM (c), as:

$$CCM (c) = AER + ASA + NDC \quad (4)$$

Finally, the degree of satisfaction of a component c, denoted by SD (c), is defined as:

$$SD (c) = CM(c) + CCM (c) \quad (5)$$

B. Risk Assessment

We use risk assessment to quantify the degree of uncertainty associated with the selection of a component during a CBS development. Goseva-Popstojanova et al. [33] defines risk as a combination of probability of malfunctioning (failure) and the consequence of malfunctioning (severity). The probability of failure depends on the probability of occurrence of a fault combined with the likelihood of exercising that fault in a scenario in which a failure will be triggered [33]. Since it is difficult to find exact estimates for the probability of failure of a component in the early phases of CBS development, we use complexity to estimate the fault proneness of a component. Component complexity is chosen as a quantitative factor because it has a proven impact on fault proneness [33, 34]. Further, we perform severity analysis to quantify the impact of the probable failure. Thus, we define the risk for a component c as:

$$Risk (c) = Complexity (c) \times Severity\_Index (c) \quad (6)$$

Complexity Analysis

We propose to perform a complexity assessment for a CBS from the perspective of a system analyst. Fundamental to a component is its interface, which characterizes the functionality provided. The interface defines the services provided by a component and acts as a basis for its use and implementation. Ideally, an interface specification describes the functional properties of a component. Function properties include a signature part to describe the operations, and a behavior part to address the overall behavior of a component. The interface signature delineates the individual elements of a component in a syntactic manner.

TABLE X.
NO/NP COMPLEXITY METRICS

NO/NP	1 – 19	20 – 50	51 +
1	Low	Low	Average
2 - 5	Low	Average	High
6 +	Average	High	High

We present a measure of interface complexity based on the IFPUG [35] function point count (an international standard -ISO/IEC 20926:2003). In IFPUG, data functions are defined as functionality provided to a user to meet internal and external data requirements and are classified into two types: (i) internal logical file (ILF) and (ii) external input file (EIF). The complexities of the ILF and EIF are determined by the data element type (DET) and the record element type (RET). We classify the interfaces that have the same operations and also exchange data with their environment as candidates for function count. For each of the selected candidates, we classify them as either ILF or EIF. Interfaces that have operations that change the attributes of other interfaces in the data exchange are classified as ILF. All the remaining interfaces are classified as EIF.

TABLE XI.
COMPONENT INTERFACE COMPLEXITY METRICS

Data Type	Low	Average	High
ILF*i*	--- x 7	--- x 10	--- x 15
EIF*i*	--- x 5	--- x 7	--- x 10

In IFPUG, each identified ILF and EIF is ranked based on the number of DET and RET using RET/DET metrics [35]. Since RET is a user recognizable subgroup, we count the number of operations (NO) in an interface. Similarly, DET is a unique user recognizable field; we count the number of parameters (NP) in an interface. By analogy with RET/DET metrics [35], we propose NO/NP complexity metrics, shown in Table 10, to rank candidate interface. Ranked interfaces are assigned weights based on IFPUG standard weights, shown in Table 11. Finally, we define an interface complexity measure of a component *i*, denoted by IC*i*, as

$$IC_i = \sum_{i=1}^{n} ILF_i + \sum_{j=1}^{n} EIF_i \qquad (7)$$

where ILF*i* and EIF*i* are the weighted values for a component interface classified based on its complexity. An example showing the approach to measuring the complexity of a CBS specification can be found in [27].

Severity Analysis - In addition to the estimate of the fault proneness of each of the components based on the interface complexities, we need to consider the severity of the consequences of potential failures [33]. For example, a component may have low complexity, but its failure may lead to catastrophic consequences. Therefore, our risk assessment takes into consideration the severity associated with each component, based on how its failure affects the requirement satisfaction. We identify that Volatility of Component (VC) and Supplier Creditability (SC) help to estimate consequence of the malfunction of a component. We use VC and SC to estimate the severity of the component, based on the experience recorded by other users of the component, and stored in component repositories.

TABLE XII.
VC SEVERITY CLASSIFICATION

Volatility of component – VC	Severity Index (VC)
VC <= 1.22 versions per year	1
1.22 < VC <= 1.38 versions per year	2
1.38 < VC < 1.54	3
1.54 < VC <=1.7 versions per year	4
> 1.7 versions per year	5

We calculate VC of a component as the ratio of the number of component releases to the total number of years from the first release. For example, 'Rapid Spell'[2] component has had five releases since its launch in 2003. Therefore, its VC value is 1.25 (five divided by four). We classify and calculate the severity index of VC by calculating confidence interval [36] based on a sample size of 80 components available in the component source repository. We calculate the confidence interval of VC with the confidence coefficient of 95%. Further, we assumed that the severity index of a component c with the VC (c) >= mean is positive. The results of the analysis indicated that, for 95% of components, their VC value was in the interval [1.22, 1.54]. The mean value of VC was 1.38 releases per year. Table 12 shows the severity index for VC based on our confidence interval analysis.

[2] www.componentsource.com

TABLE XIII.
SC SEVERITY CLASSIFICATION

Supplier creditability – SC	Severity Index (SC)
1	5
2	4
3	3
4	2
5	1

SC is the ranking given by the reviewers of the component. The rating reflects the creditability of the component. Component source repository provides a rating of components, based on the reviews, by rating components on the scale of one to five. A component with rating five indicates that the customers of the component are completely satisfied with its functionality, provided documentation and support. A component with rating one indicates that the customer is not satisfied with the provided functionality. Table 13 shows the severity index for a component on a scale of one to five. We propose to assign a severity index (SC) of five to a component, which has a rating of one. This indicates that the component is less credible and thus, will have a high degree of severity of consequences associated with its use. Finally, we define the overall severity index of a component c, denoted by Severity_Index (c) as:

$$\text{Severity\_Index}(c) = \text{Severity Index}(VC) + \text{Severity Index}(SC) \qquad (8)$$

C. Applying Top-Down Analysis to the SEJ System

We illustrate our top-down analysis based on CLR 43.2. We identify that CLR 43.2 is achieved as follows: converting scanner data into a network element, building network document, processing the request/response and, finally, writing the network document to an output stream. We start the satisfaction degree measure of candidate components for CLR 43.2 by developing matching metrics, as shown in Table 14. For example, we identity that the IPWorks component provides the overall required functionality using XML parsers and file transfer support. Table 14 shows that there are four compatible features provided by IPWork and zero missing features. IPWorks also provides three additional features and they have 'little' effect on the overall CLR 43.2 satisfaction. Similarly, it requires 'moderate' adaptation effort and the system architecture requirements match to the overall SEJ software system development platform. However, non-development cost associated with IPWorks is moderate (less than 5% over the budget). Therefore, we calculate satisfaction degree for IPWorks, based on equation 3, 4 and 5, as follows:

SD (IPWorks) = (4 + (0x5) + (3 x 1)) + (1 + 3 + 3) = 7 + 7 = 14 (9)

TABLE XIV.
CLR 43.2 MATCHING METRICS

	Convert data to network element	Build network document to be transferred	Process the request/response	Write the network data to an output stream
Network communication				✓
XML parser	✓	✓		
SOAP support			✓	
File transfer				✓
Email				
Network monitor				
Authentication				

The IPWorks component has eleven interfaces, which are identified from the evaluation version of the component. From eleven interfaces, five qualify as EIF because operations of these interfaces do not change the attributes of other interfaces in exchanging the information. The remaining six are classified as ILF because they affect the attributes of other interfaces during the message exchange. Further, functional complexity of both EIF and ILF interfaces is low and their weighted complexity value is (5 x 7 = 35) and (6 x 5 = 30), respectively. For a detailed discussion of the way to classify interfaces and calculate complexity, please refer to our previous work [27]. Further, we assign a value of 3 to the volatility of IPWorks because it had six versions in the last four years (1.5 versions per year), using Table 12. IPWorks has been assigned rating 3 on the component source website, thus its SC value is 1. Finally, we calculate decision metrics value for IPWorks using equation 2, as shown in Table 15. Similarly, we calculate decision metrics values for all the candidate components for CLR 43.2.

TABLE XV.
CLR 43.2 MATCHING METRICS

Component	Satisfaction Degree								Risk			D (RG, i)
	CF	MF	IMF	AF	IAF	AER	ASA	NDC	IC	VC	SC	
IPWorks	4	0	5	3	1	1	3	3	65	3	1	5.38%
PowerTCP	2	2	-2	2	3	-2	3	1	36	2	3	3.33%
SocketTools	2	2	-5	3	1	-5	3	5	58	4	3	-0.49%
SuperCom	2	2	-5	2	1	-5	3	-2	43	1	1	-11.63%
SocketWrench	2	2	-5	0	0	1	-2	3	45	1	1	-6.67%
ComponentSpace	3	1	-5	0	0	-2	3	3	48	5	3	0.52%

VII. TRADE-OFF ANALYSIS

Trade-off analysis aims at balancing the conflicting interests between stakeholder requirements and negotiating resolutions during component selection. In this paper, we consider only interaction conflicts. Requirements are said to be in interaction conflict if the satisfaction of one requirement may impair or eliminate the satisfaction of another requirement [26]. The key feature of trade-off analysis in our approach is a shift from a requirements-driven conflict analysis to a collaborative process in which both requirements and component features balance the conflicting interests. Traditionally, interaction conflict analysis [37, 38] has been achieved by conflict detection, resolution generation and resolution selection. Conflict between requirements is detected by matching requirements and potential resolutions are generated using analytic compromise and heuristic compensation. Finally, resolutions are selected based on a set of guidelines.

In CBS development, there is a need to find a balance between requirements and available component features. Requirement-driven approaches alone are usually not sufficient as they do not support a collaborative process of negotiating individual interests of stakeholders against a component's features. We propose a trade-off analysis, which attempts to find a mutually acceptable resolution to a conflicting situation by providing an opportunity for stakeholders to trade-off individual interests. The trade-off analysis consists of four stages: conflict identification, potential resolution generation, measuring the component's relevance to each potential resolution and selection of suitable resolution. In the first step, interaction conflict is identified among requirements. In the second step, we propose a set of rules to generate potential resolution to a conflict. In the third step, we calculate the relevance index for each component with reference to the potential resolutions. Finally, a suitable resolution and components are selected using the relevance measure.

Step 1: Conflict Identification

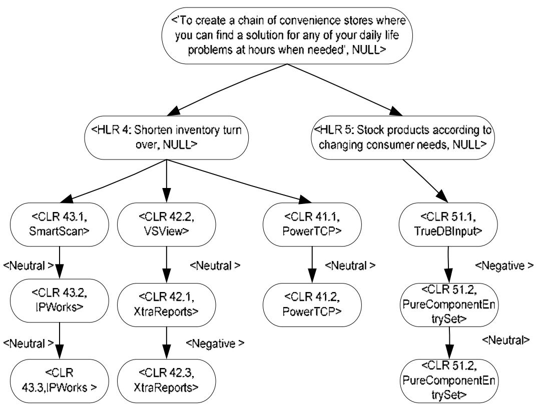
For a given requirement, trade-off analysis starts with investigating its relationships with other requirements. We adopt the concept of *Generic Relationship Question (GRQ)* [21] to represent the relationships between requirements and identify possible conflicts. Suppose we want to investigate the relationship between two concrete-level requirements, (1) CLR - X with ranking i and (2) CLR - Y with ranking i + 1. Since CLR - Y has a higher ranking than requirement X, we assume that a suitable component (Component A) has already been identified and it is represented as a leaf node <CLR - Y, Component A> in RG. We develop the following GRQ to represent the relationship between CLR - X and CLR – Y:

How does 'Requirement X' relate to 'Component A'? (10)

where 'Requirement X' defines a set of interactions associated with CLR X–scenario. Similarly, 'Component A' represents a set of component features, which specify the functionality of the component selected for CLR – Y.

After establishing the GRQ, we propose to start the conflict identification by doing a pair-wise comparison between 'Requirement X' and 'Component A'. The set of interactions identified for 'Requirement X' are mapped to the component features identified for 'Component A'. Further, impact of each 'Requirement X' interaction is analyzed against all features associated with 'Component A'. This impact can be neutral or negative. A neutral impact indicates that the component feature does not influence satisfaction of the interaction. The negative impact indicates that the component feature does influence satisfaction of the interaction. We believe this pair-wise comparison reduces the importance of intuition in detecting conflicts [39] since it forces the comparison of all possible interactions.

Table 16 shows conflict identification metrics where all the interaction steps associated with 'Requirement X' are listed in rows, and all the features associated with 'Component A' are arranged in columns. We define that two requirements have a conflicting relationship if there is at least one negative impact between them. A 'X' at the interaction of a component feature and CLR scenario interaction indicates that the interaction has a negative impact on the component feature. We acknowledge the fact that a domain expert will play a key role in identifying the conflicts between requirements.



<'To create a chain of convenience stores where you can find a solution for any of your daily life problems at hours when needed', NULL>
<HLR 4: Shorten inventory turn over, NULL>
<HLR 5: Stock products according to changing consumer needs, NULL>
<CLR 43.1, SmartScan>
<CLR 42.2, VSView>
<CLR 41.1, PowerTCP>
<CLR 51.1, TrueDBInput>
<Neutral >
<Neutral >
<Neutral >
<Negative >
<CLR 43.2, IPWorks>
<CLR 42.1, XtraReports>
<CLR 41.2, PowerTCP>
<CLR 51.2, PureComponentEntrySet>
<Neutral >
<Negative >
<Neutral>
<CLR 43.3,IPWorks >
<CLR 42.3, XtraReports>
<CLR 51.2, PureComponentEntrySet>

Figure 6. Requirements Graph for the SEJ System

TABLE XVI.
CONFLICT IDENTIFICATION METRICS

	Feature 1	Feature 2	….	Feature n
Interaction 1		X		
Interaction 2	X	X		
….				
Interaction n				

Step 2: Resolution Generation

The next step in trade-off analysis is to generate potential resolutions to a conflict. We adopt the concept of knowledge-based negotiation [40] to propose a set of rules for resolution generation. Knowledge-based negotiation proposes concepts of compensation and dissolution. For compensation, the strategy is to satisfy requirements outside the conflict to increase the overall desirability of the resolution by adding new issues to negotiations. For dissolution, the strategy is to allow conflicting designs to be part of the specification, but remove their negative interaction. Dissolution replaces conflicting items with potentially less contentious items. In our approach, we propose the following heuristics as a guideline to generate resolutions.

Rule1: For each conflict, introduce one or a group of new interactions for the scenario associated with a requirement with lowest priority in the conflict, such that it applies a set of constraints, which result in conflict resolution.

Rule2: Delete an interaction or group of interactions from the scenario associated with a requirement with lowest priority in the conflict such that it removes the conflict.

Rule3: Write an adaptation code for interoperability between selected components such that it masks the conflict.

Step 3: Relevance Index

In order to effectively select a suitable resolution to a conflict, there is a need to analyze the impact of each resolution and quantify candidate components relevance with respect to each resolution. We introduce the notion of a relevance index, which provides a mechanism for understanding and balancing the satisfaction and risk associated with each resolution. The relevance index of a component is defined as a function of a component's satisfaction degree, associated risk and impact of each resolution. We calculate relevance index as follows:

- *Calculating satisfaction and risk metrics for the requirements*. We use the set of metrics defined in section 6 to measure candidate components satisfaction degree and associated risk with reference to the requirement. These measures indicate the potential value for each metrics in the case where there is no conflict between the requirement and its predecessor requirements. We calculate these values to help us in identifying the impact on component satisfaction and risk caused by the conflict. We propose to denote these values as SP and RP respectively.
- *Calculating component satisfaction degree and risk for each resolution*. Similarly, we use the set of metrics defined in section 6 to measure component satisfaction and associated risk with reference to each conflict resolution. These measures indicate the actual value for each metrics with reference to a conflict resolution. We propose to denote these values as SA and RA respectively.
- *Calculating impact of resolution for each metrics*. Whenever a conflict is detected and a set of

resolutions is generated, it is necessary to take into account the impact of the resolutions on the candidate component's satisfaction. When a satisfaction metric value decreases against its potential value, we assign an impact factor to the component's satisfaction degree to quantify the impact introduced by the conflict resolution. We define the Satisfaction Impact Factor (SIF) as

$$\begin{aligned}&\text{if (SA} < 0\text{) then SIF} = (\text{SA} - \text{SP}) / \text{SA}\\&\text{else if (SA} >= 0\text{) then SIF} = (\text{SP} - \text{SA}) / \text{SA}\end{aligned} \quad (11)$$

- *Calculating the relevance index (RIc) for each component.* Relevance percentage for a component in a resolution is defined as the ratio of a component's satisfaction degree and risk subtracted by the associated impact factor.

$$\text{RIc} = ((\text{SA} / \text{RA}) - \text{SIF}) \times 100 \quad (12)$$

Step 4: Situable Resolution Selection

The last step of trade-off analysis is to identify suitable resolution and select relevant components. We propose to identify suitable resolutions and select relevant components by comparing the relevance index measures. However, it is important to note that the final figures of the relevance index cannot be used as the sole criterion for selecting suitable resolutions and relevant components. For example, the same relevance measure may be achieved for two components, but the individual attributes that contribute to the measure may be different from one another. Hence, we define the following steps as a guide on selecting suitable resolutions and components.

Rule 1: Calculating the suitability of each resolution, with the suitability of each resolution defined as the sum of relevance indexes of all components for the resolution divided by the total number of components.

$$\text{Suitability} = \sum_{i=0}^{C} \text{RIc} / C \quad (13)$$

where C is the total number of components.

Rule 2: Identifying the most suitable resolution by selecting the one with highest value of suitability.

Rule 3: Formulating the most suitable resolution and selecting the component with the highest value of relevance index.

Rule 4: If two, or more than two, components have the same relevance index, select the component with the highest value of satisfaction.

Rule 5: If two, or more than two, components have the same value for relevance index and satisfaction, then select the component with the lowest value of risk.

A. Applying Trade-off Anlysis to the SEJ System

The first step in trade-off analysis is to analyze the relationship between requirements and identify potential conflicts. Figure 6 shows an example of a negative relationship between CLR 51.1 - 'Store customer profile' and CLR 51.2 – 'Maintain customer privacy'. Since, CLR 51.1 is the highest ranked requirement for HLR 5, the 'TrueDBInput' component has already been selected using top-down analysis. We represent the relationship between CLR 51.1 and CLR 51.2 by defining the following GRQ template.

How does 'Requirement 51.2' relate to 'Component TrueDBInput'? (14)

We identify that 'Component TrueDBInput' consists of three features, namely, 'data validation', 'data storage and 'support of user friendly data entry'. Similarly, 'Requirement 51.2' consists of two interactions which help achieve CLR 51.2 – scenario. These interactions are: (i) control access to customer data and (ii) protect customer data.

TABLE XVII.
CONFLICT IDENTIFICATION METRICS FOR CLR51.1 & CLR 52.1

	Data validation	Data storage	User friendly data entry
Control access to customer data			
Protect customer data		X	

We start the trade-off analysis by developing conflict identification metrics, as shown in Table 17, to perform a pair-wise comparison between 'Requirement 51.2' and 'Component TrueDBInput'. We select the first 'Requirement 51.2' interaction and analyze its impact on all the 'Component TrueDBInput' features. This interaction has a neutral impact on all the three features. For example, the 'data validation' feature does not influence the satisfaction of 'control access to customer data'. Next, we select the second 'Requirement 51.2' interaction and analyze its impact on all the 'Component TrueDBInput' features. This interaction has a neutral impact on 'data validation' and 'user friendly data entry' features. However, it has a negative impact on 'data storage' feature because stakeholders of the SEJ system wants to obtain all customer's consent before data storage. Therefore, it is important to resolve this conflict before selecting a suitable component for CLR 51.2.

TABLE XVIII.
RELEVANCE INDEX METRICS

Component	SA	Risk	SP	SIF	RI1
PureComponents EntrySet	14	130	17	0.214286	-10.66%
ComponentOne Input	12	190	15	0.25	-18.68%
Xceed	-10	315	-2	0.8	-83.17%
AspLib	4	258	9	1.25	-123.45%
Input Pro	-7	220	-6	0.142857	-17.47%
Dxperience	-10	565	-8	0.2	-21.77%

The second step in trade-off analysis is to generate resolutions to a conflict. We identify the following potential resolutions for the abovementioned conflict based on the resolution generation rules just defined.

- Resolution 1: Add a new interaction 'notify customer before data entry' in CLR 51.2 - scenario to provide a compromise for the identified conflict.
- Resolution 2: Add a new interaction 'collect information only if customer agrees to data collection' in CLR 51.2 - scenario to provide a compromise for the identified conflict.

TABLE XIX.
RESOLUTION SELECTION METRICS

Component	RI1	RI2
PureComponents EntrySet	-10.66%	-10.95%
ComponentOne Input	-18.68%	-17.86%
Xceed	-83.17%	-83.94%
AspLib	-123.45%	-124.10%
Input Pro	-17.47%	-18.23%
Dxperience	-21.77%	-20.96%
Suitability	-45.87%	-46.01%

For each resolution, we calculate the relevance index for the candidate components, as shown in Table 18. For example, we calculate the relevance index for 'PureComponentsEntrySet'. Firstly, we calculate components satisfaction degree and risk with reference to original CLR 51.2. The 'PureComponentsEntrySet' component has two CF, zero MF with 'great' impact; and three AF with negligible impact. Thus, its potential satisfaction value is measured as 17. Next, we calculate its satisfaction degree with reference to resolution 1 which modified CLR 51.2 by adding a new interaction. Since, 'PureComponentsEntrySet' component does not support the new interaction; it now has two CF, one MF with considerable impact and three AF with negligible impact. Thus, its actual satisfaction value is measured as 14. It is important to note that the risk value remains the same because it does not depend on the set of CLR interactions. Finally, we calculate the impact factor for the component 'PureComponents EntrySet', based on equation 11, as .21 ((17 – 14) /17). Similarly, the relevance index of the component is calculated, based on equation 12, as -10.66% ((14/130) - .21)) x 100.

Similarly, we calculate the suitability of each resolution based on equation 13. Finally, based on rules defined in step 4 of section 7, we identify that a suitable resolution for the conflict is resolution 1 and 'PureComponents EntrySet' is the component which best satisfies the requirement CLR 51.2, as shown in Table 19. It is important to note that the suitability values for both resolutions are very similar because one new interaction was introduced for both resolutions and both of them were not directly supported by candidate components. Further, all components have negative relevance index values which also highlights the fact that some sort of tailoring code is required by the components in order to meet stakeholder requirements.

VIII. CONCLUSIONS AND FUTURE WORKS

In this paper, we have presented RAAP, a Requirements Analysis and Assessment Process framework for CBS that can provide quantitative information and guidelines for stakeholders to evaluate the suitability of the components for a given set of requirements. RAAP guides stakeholders through a collaborative process of requirements elicitation, matching and trade-off analyses. This collaborative process involves continuous requirements analysis by providing a set of metrics for balancing the stakeholder expectations against potential risks. We propose the notation of a relevance index and identify suitable resolutions to a conflicting scenario. RAAP investigates proposed resolutions and evaluates the risk associated with each proposal. It also helps in assessing types of conflicts that may arise when different components are integrated into a CBS. Using the SEJ system as a case study, we demonstrated the usefulness RAAP. This SEJ case study consists of eleven requirements, two conflicting requirements situation and a RG containing fourteen nodes.

In this work, we have gained a few insights. The collaborative assessment of requirements provides a road map for identifying risks and improving the quality of the process by selecting suitable components. Our requirements analysis algorithm evaluates requirements in a descending order from high priority to low priority requirements, thus, first selecting components which satisfy high priority requirements. It helps in developing a CBS with high stakeholder satisfaction. One of the limitations of RAAP is that we consider only the functional requirements. An important aspect of a CBS is its non-functional requirements, like security and performance. It is important to incorporate these non-functional requirements during CBS development because the non-functional requirements can also conflict with one another and need to be represented and analyzed. Furthermore, in the course of applying RAAP to the SEJ software system, we observed the following points.

- *Comprehensive assessment of candidate components helps in making informed decisions during component selection*. Our case study shows that satisfaction and risk measures aid in making informed decisions about component selections. Furthermore, they provide a mechanism for understanding and measuring the impact of changes during conflict resolution. For example, Table 15 indicates how candidate components match a requirement by quantifying their satisfaction degrees and associated risks.
- *Development of CBS with likelihood of high priority requirements satisfaction*. Our process starts selecting components by ranking requirements based on their priority, view and matching potential. Selecting components based on this ranking helps CBS satisfy at least the actor specific and high priority requirements.

- *Early detection of components that will require adaptation effort*. As shown in our case study, during the top-down and trade-off analysis, a component can have a negative satisfaction measure, which indicates that the component will need some sort of wrapper code to either satisfy the requirement or avoid the conflict. Thus, these satisfaction numbers enable a system analyst to identify these components early in the requirements and specification phases of a CBS life cycle.
- *Easy back tracking of risks*. Our process also develops a requirements graph, as shown in Figure 6, which can be used as a possible way to backtrack any earlier decisions regarding component selections and required adaptation efforts.
- *The Need for a standard notation*. Component information in a component source repository is available in a number of formats ranging from natural language descriptions; help files, to sample evaluation copies. A domain expert plays a role in how requirements are perceived and component information is analyzed. Thus, there is a need to extend Unified Modeling Language (UML) [41], an industry de-facto standard, to describe the RAAP specifications. For example, Hussein et al. [42] proposed a UML profile to specify intrusion detection facilities during CBS development.
- *The Need for developing an automated tool*. An apparent obstacle to the use of the RAAP is the effort and time required to collect the relevant metrics. In our case study, on average five candidate components were evaluated for each CLR; and it involved on average thirty computations for matching between CLR scenarios and component features. We plan to develop an automated tool for RAAP. The tool will consist of six modules: namely, requirements elicitor, component analyzer, graph generator, metrics collectors, conflict resolver and user interface. The requirements elicitor module will be used for eliciting stakeholder requirements, which contain high-level and concrete-level requirements. The requirements elicitor module will rank requirements based on priority, matching potential and view. The component analyzer module will extract component feature information contained in a repository. The graph generator module will construct a directed graph to represent the requirements, the selected components and the relationship between different requirements. The metrics collector module will compute the satisfaction and risk metrics for each requirement of the system. The conflict resolver module will describe the conflicts between requirements, generate the resolutions and calculate the relevance indexes to help software designers select a suitable resolution for a given conflicting requirements situation. Finally, the user interface module will facilitate users to interactively enquire about the selected components.

For our future work, we plan to investigate the benefits of satisfaction, risk, resolution selection metrics; and relevance index for a CBS. We also aim to extend UML so that a CBS using RAAP can be specified in UML. This will help the automation as both stakeholder requirements and component specifications can be based on machine-readable UML.

ACKNOWLEDGMENT

The second author would like to thank King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia for its continuous support in research.

REFERENCES

[1] N. A. Maiden and C. Ncube, "Acquiring COTS Software Selection Requirements," *IEEE Software*, vol. 15, pp. 46-56, 1998.

[2] K. R. P. H. Leung and H. K. N. Leung, "On the Efficiency of Domain based COTS Product Selection Method," *Information and Software Technology*, vol. 44, pp. 703 - 715, 2002.

[3] C. Alves and A. Finkelstein, "Investigating Conflicts in COTS Decision-Making," *International Journal of Software Engineering and Knowledge Engineering*, vol. 13, pp. 1-21, 2003.

[4] L. Chung and K. Cooper, "Knowledge Based COTS Aware Requirements Engineering Approach," In Proceedings of the 14th international conference on Software engineering and knowledge engineering, pp. 175-182, 2002.

[5] I. Sommerville, "Integrated Requirements Engineering: A Tutorial," *IEEE Software*, vol. 22, pp. 16 - 23, 2005.

[6] S. Lauesen, "COTS Tenders and Integration Requirements," In Proceedings of 12th IEEE International Requirements Engineering Conference, Kyoto, Japan, pp. 166-175, 2004.

[7] C. Szyperski, D. Gruntz, and S. Murer, *Component Software - Beyond Object - Oriented Programming*, Second edition ed: Addison-Wesley, 2002.

[8] G. Kotonya and J. Hutchinson, "Viewpoints for Specifying Component Based Systems," In Proceedings of *7th International Symposium on Component Based Software Engineering*, vol. LNCS 3054, pp. 114 - 121, 2004.

[9] C. Rolland, "Requirements Engineering for COTS Based Systems," *Information And Software Technology*, vol. 41, pp. 985 - 990, 1999.

[10] B. Boehm and C. Abts, "COTS Integration: Plug and Pray?," *IEEE Computer*, vol. 32, pp. 135 - 138, 1999.

[11] C. Alves, "COTS Based Requirements Engineering," *Component Based Software Quality, LNCS 2693*, pp. 21 - 39, 2003.

[12] C. Alves and A. Finkelstein, "Requirements Negotiation for COTS-based Systems: Challenges and Open Issues," In Proceedings of Second International Workshop on Requirements and COTS Components (RECOTS'04), Koyoto, Japan, 2004.

[13] A. Ishikawa and T. Nejo, *The Success of 7-Eleven Japan: Discovering the Secrets of the World's Best-Run Convenience Chain Stores*: World Scientific Publishing Co., 1998.

[14] J. Kontio, S.-F. Chen, K. Limperos, R. Tesoriero, G. Caldiera, and M. Deutsch, "A COTS Selection Method and

Experience of its use," In Proceeding of 20th Annual Software Engineering Workshop, Greenbelt, Maryland, 1995.
[15] S. Whang, C. Koshijima, H. Saito, T. Ueda, and S. V. Horne, *Seven Eleven Japan (GS18)*: Stanford University Press, 1997.
[16] S. J. Bleistein, A. Aurum, K. Cox, and P. K. Ray, "Strategy Oriented Alignment in Requirements Engineering: Linking Business Strategy to Requirements of e-Business Systems using the SOARE Approach," *Journal of Research and Practice In Information Technology*, vol. 36, pp. 259 - 276, 2004.
[17] S. J. Bleistein, K. Cox, and J. Verner, "Validating strategic alignment of organizational IT requirements using goal modeling and problem diagrams," *Journal of Systems and Software*, vol. 79, pp. 362 - 378, 2006.
[18] S. J. Bleistein, K. Cox, J. Verner, and K. T. Phalp, "B-SCP: A Requirements Analysis Framework for Validating Strategic Alignment of Organizational IT based on Strategy, Context and Process," *Information and Software Technology*, vol. 48, pp. 846-868, 2006.
[19] J. Lowy, *Programming .NET Components*: O'Reilly Media, Inc, 2005.
[20] K. C. Wallnau, S. A. Hissam, and R. C. Seacord, *Building Systems from Commercial Components*: Addison- Wesley, 2002.
[21] M. G. Mendonca and V. R. Basili, "Validation of an Approach for Improving Existing Measurement Frameworks," *IEEE Transactions on Software Engineering*, vol. 26, pp. 484 - 499, 2000.
[22] C. Rolland, C. Souveyet, and C. B. Achour, "Guiding Goal Modeling Using Scenarios," *IEEE Transactions on Software Engineering*, vol. 24, pp. 1055 - 1071, 1998.
[23] A. V. Lamsweerde and E. Letier, "Handling Obstacles in Goal-Oriented Requirements Engineering," *IEEE Transactions on Software Engineering*, vol. 26, pp. 978 - 1005, 2000.
[24] C. Potts, K. Takahashi, and A. I. Anton, "Inquiry-based Requirements Analysis," *IEEE Software*, vol. 11, pp. 21 - 32, 1994.
[25] J. Lee and N.-L. Xue, "Analyzing User Requirements by Use Cases: A Goal-Driven Approach," *IEEE Software*, vol. 16, pp. 92 - 101, 1999.
[26] J. Lee, N.-L. Xue, and J.-Y. Kuo, "Structuring Requirement Specifications with Goals," *Information and Software Technology*, vol. 43, pp. 121 - 135, 2001.
[27] Sajjad Mahmood and Richard Lai, "A Complexity Measure for UML Component System Specification," *Software-Practice and Experience*, Vol. 38, Issue 2, pp. 117 – 134, 2008.
[28] J. Han, "A comprehensive interface definition framework for software components," In Proceedings of 998 Asia Pacific Software Engineering Conference, Taipei, Taiwan, pp.110-117, 1998.
[29] A. Cechich and M. Piattini, "Early Detection of COTS Component Functional Suitability," *Information and Software Technology*, vol. 49, pp. 108 - 121, 2007.
[30] V. R. Basili and B. Boehm, "COTS-based Systems Top 10 List," *IEEE Computer*, vol. 34, pp. 91-93, 2001.
[31] N. E. Fenton and S. L. Pfleeger, *Software Metrics: A Rigorous & Practical Approach*: Thomson Computer Press, 1996.
[32] "Risk Management Guide for DOD Acquisition," *Department of Defense*, 2003.
[33] K. Goseva-Popstojanova, A. Hassan, A. Guedem, W. Abdelmoez, D. E. M. Nassar, H. Ammar, and A. Mili, "Architectural-level risk analysis using UML," *IEEE Transactions on Software Engineering*, vol. 29, pp. 946-960, 2003.
[34] Khaled El Emam, Walcelio Melo, and J. C. Machado, "The Prediction of Faulty Classes Using Object Oriented Design Metrics," *Journal of Systems and Software*, vol. 56, pp. 63 - 75, 2001.
[35] IFPUG, *Function Point Counting Practices Manual, Release 4.1*: International Function Point Users Group, Princeton Junction NJ, 2000.
[36] W. Mendenhall and T. Sincich, *Statistics for Engineering and the Sceinces*: Maxwell Macmillan International, 1992.
[37] A. v. Lamsweerde, R. Darimont, and E. Letier, "Managing Conflicts in Goal-Driven Requirements Engineering," *IEEE Transactions on Software Engineering*, vol. 24, pp. 908 - 926, 1998.
[38] W. N. Robinson and S. D. Pawlowski, "Managing Requirements Inconsistency with Development Goal Monitors," *IEEE Transactions on Software Engineering*, vol. 25, pp. 816 - 835, 1999.
[39] L. M. Cysneiros and J. C. S. d. P. Leite, "Nonfunctional Requirements: From Elicitation to Conceptual Models," *IEEE Transactions on Software Engineering*, vol. 30, pp. 328-350, 2004.
[40] W. N. Robinson and S. Fickas, "Supporting Multi-Perspective Requirements Engineering," In Proceedings of First International Conference on Requirement Engineering, Silver Spring, MD, 1994.
[41] J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual*: Addison-Wesley, 2005.
[42] M. Hussein and M. Zulkernine, "Intrusion detection aware component-based systems: A specification-based framework," *The Journal of Systems and Software*, vol. 80, pp. 700-710, 2007.

Richard Lai is with the Department of Computer Science and Computer Engineering at La Trobe University, Australia. Prior to joining La Trobe, he had spent more than 10 years in the computer and communications industry. He was ranked as the world's number one scholar in systems and software engineering consecutively for four years (1999–2002), according to an annual survey published in the Journal of Systems and Software. His current research interests include component-based software system, software measurement, requirements engineering, and global software development.

Sajjad Mahmood received the PhD degree in computer science, from La Trobe University, Melbourne, Australia. He is an assistant professor of the Information and Computer Science Department at the King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia. His research interests include software reuse, component-based software engineering and software product lines.

Shaoying Liu is a professor in the department of computer science at Hosei University. He received his Ph.D in formal methods at the University of Manchester, UK, and has experienced in working at Xi’an Jiaotong University, the University of York, the University of London, and Hiroshima City University before 2000. His research interests include formal methods, formal engineering methods, software inspection, testing, and intelligent software engineering environments. He has published over 110 papers in refereed journals and international conferences, one book titled “Formal Engineering for Industrial Software Development” with Springer, and another four edited conference proceedings. He is a fellow of British Computer Society, senior member of IEEE Computer Society, and member of Japan Society for Software Science and Technology.