Fuzzy Logic as a Basis for Reusing Task-Based Specifications

Lein F. Lai, Jonathan Lee,* Stephen J. Yang
Software Engineering Laboratory, Department of Computer Science and Information Engineering, National Central University, Chungli, Taiwan

In this paper, we propose an approach to reusing requirements specification, called task-based specifications in conceptual graphs (TBCG). In TBCG, task-based specification methodology is used to serve as the mechanism to structure the knowledge captured in conceptual models, and conceptual graphs are adopted as the formalism to express requirements specification. TBCG provides several mechanisms to facilitate the reuse of formal specifications: a contextual retrieval mechanism to support context-sensitive specifications retrieval and incremental context acquisition, a graph matching mechanism to compute the similarity between two graphs based on the semantic match and fuzzy logic, and a paraphraser to serve as an explanation mechanism for the retrieval specifications. © 1999 John Wiley & Sons, Inc.

I. INTRODUCTION

Software reuse has been recognized as an important research topic in software engineering.[1–3] Reusing components provides many benefits over the traditional designing from scratch: (1) increasing productivity by reducing the production cost in future development efforts, (2) improving system quality by reusing components that have been successfully used, and (3) reusing components that have been proven correct to enhance reliability and thus for ease maintenance.

Software reuse could be implemented at several levels including the specification level, the design level, and the code level. However, design and codes are not only usually too detailed to understand, but also difficult to match against other artifacts. Since the specification level is at higher levels of abstraction than others, matching and adapting specifications are easier. Therefore, reuse at the specification level can yield great benefits in software productivity and quality early in the software life cycle. As a means to precisely characterize a specification and to facilitate automation, a formal representation becomes increasingly important in specification reuse.[4–6]

*Author to whom correspondence should be addressed: e-mail: yjlee@se01.csie.ncu.edu.tw.

INTERNATIONAL JOURNAL OF INTELLIGENT SYSTEMS, VOL. 14, 331–357 (1999)
 CCC 0884-8173/99/040331-27

This paper summarizes an approach to reusing requirements specification, called task-based specifications in conceptual graphs (TBCG). TBCG uses task-based specification methodology[7–11] as the mechanism to structure the knowledge captured in conceptual models, and conceptual graphs as the formalism to express requirements specification. TBCG provides several mechanisms to facilitate the reuse of formal specifications:

- A contextual retrieval mechanism is used to support context-sensitive specifications retrieval and incremental context acquisition. By taking contexts into account, irrelevant specification will be excluded out. Reducing the number of possible candidates for selection thus increases the efficiency of reusing requirements specifications. In addition, TBCG provides incremental context acquisition by using user feedback to either reinforce or correct the system's knowledge in case of success or failure.
- A graph matching approach is used to compute the similarity between two graphs based on fuzzy logic, the semantic match, and the analogical match. The fuzzy similarity between two sets of TBCG graphs can be calculated by computing the degree of implication and the degree of consistency. The notion of the semantic distance and the type hierarchy in CGs provides a basis for matching two types on the semantic aspect. By comparing a complete network of knowledge rather than unrelated facts, the analogical match can address the structural mapping between two graphs.
- A TBCG paraphraser is used to serve as an explanation mechanism for the retrieved specifications. The explanation can increase user's understanding and guide the adaptation of retrieved specifications.

In Section II, we first give an overview on how to express task-based specifications in conceptual graphs. In Section III, basic components for reusing TBCG specifications are introduced. A contextual retrieval mechanism to retrieve and select reusable TBCG specifications is discussed in Section IV. TBCG paraphraser is described in Section V. Section VI outlines the implementation of the TBCG system. Finally, we summarize the potential benefits of TBCG approach and outline our future research plan in Section VII.

II. TASK-BASED SPECIFICATIONS IN CONCEPTUAL GRAPHS

Task-based specification methodology acquires and organizes domain knowledge, functional requirements, and high level problem solving methods around the general notion of tasks. A specification can be described at various abstraction levels and thus pieces of abstract specification can be refined into a more detailed one in a lower abstraction level. The specification has two components: a model specification that describes static properties of the system and a process specification that characterizes dynamic properties of the system. The static properties of the system are described by two models: a model about domain objects, and a model about the problem solving states which we refer to as a state model. The dynamic properties of the system are characterized by (1) using the notion of state transitions to explicitly describe what the functionality of a task is, and (2) specifying the sequence of subtasks and interactions between

subtasks (i.e., behavior of a system) using task state expressions (TSE). A TSE uses the following operators that can be divided into three groups: (1) sequencing: follow operator and immediately follow operator, (2) branching: selection, optional operator and conditional operator, and (3) iteration: iteration operator. Both the model and the process specification can be first described in their high level abstract forms, which can be further refined into more detailed specifications in the next level. The notion of task structure (i.e., task−method−subtask)[12] is adopted for the process refinement. A more detailed description of TBSM can be found in Refs. 10 and 11.

The conceptual graph[13] is a directed, finite, connected graph and consists of concepts, concept instances (referents), and conceptual relations. Concepts and relations represent declarative knowledge. Procedural knowledge can be attached through actors. Actors represent processes that can change the referents of their output concepts, based on their input concepts. Concepts are represented in square brackets, relations in parentheses, and actors in angle brackets. Delugach has extended conceptual graphs to include a new type of node, demons (in double angle brackets), to cause creation and retraction of input and output concepts.[14] A demon's algorithm causes each of its actual output concepts with referents to be asserted (i.e., marked), while each of its actual input concepts is to be retracted. If there is more than one input concept, no demon action occurs until all of its input concepts have been asserted. The notion of constraint overlays has also been incorporated into the conceptual graphs, which provides a method to attach constraints to objects and collections of objects that make up world states.[15,16] Constraint overlays, represented in angle brackets and linked to concepts with dash lines, can overlay actors (procedures or constraints) on a conceptual graph to describe the changes to a model state.

Conceptual graphs have several useful features that can facilitate the mapping from task-based specifications to their counterpart graphs. For example, terms and relations in the domain of TBSM can be directly mapped to concepts and conceptual relations, and the notion of partial conceptual graphs corresponds to the notion of the partial model in our methodology. An overview of the proposed approach depicting the mapping from task-based specifications to conceptual graphs is shown in Figure 1. Components of task-based specifications in conceptual graphs (TBCG) are described below (refer to the Appendix for the notations of conceptual graphs and TBCG).

Domain Model. Since models are similar to entity-relationship models, the transformation of terms, attributes, and relations to conceptual graphs is straightforward. First, a term becomes a concept of type ENTITY. Second, an attribute associated with a term is represented by a relation (attr), and an attribute name is characterized by a concept type as a data type through a relation (chrc). Finally, a relation becomes a conceptual relation.

State Model. A conceptual graph representation for a state model should capture two main semantics in the model: the stages of completion and constraints satisfaction. Our translation rules are summarized as follows: (1) To overlay constraint actors on the conceptual graph of state objects. The constraint overlays are used to show the relationships among state objects. All state

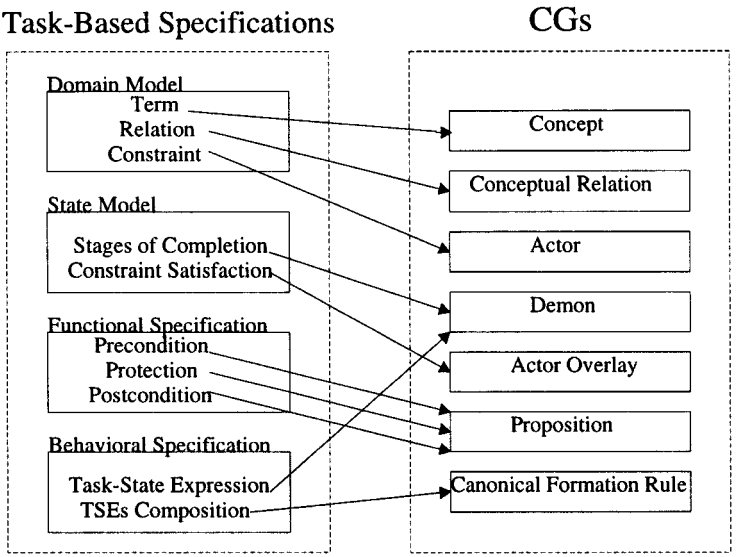


Figure 1. Expressing task-based specifications in conceptual graphs.

objects associated with a constraint actor are required to satisfy the relationship expressed by the actor. (2) To use demons for state transitions whose inputs and outputs are state objects as referents of the type STATE, and input tokens are of the type TASK. No demon action occurs until all input concepts are asserted and all input tokens are enabled. The performance of tasks (in the form of input tokens) is thus essential for state transitions.

Functional Specifications. Preconditions, protections and rigid postconditions are represented by propositions. A soft postcondition can be viewed as a fuzzy proposition with fuzzy concepts and fuzzy conceptual relations.

Behavioral Specifications. TSE is an extension of regular expressions, and therefore, can be represented using state transition diagrams. To transform TSEs into conceptual graphs, we have adopted the notion of demons to represent transitions, which possesses the semantics of an actor node with respect to output concepts' referents, with the additional semantics that a demon actually asserts its output concepts and then retracts its input concepts.[14] We also assume that here is a mapping φ that maps an expression to its state where the postcondition after progressing through the expression is true. The distinction between the follow and the immediately follow operator is noted by using the demon for the immediately follow operator with the task before and after the operator being marked, whereas the follow operator is transformed into a demon whose inputs are not yet completely marked.

In the cases of selectional, iteration, conditional and optional operators, two convention are adopted. First, we follow the tradition of demons by using an initiator demon (i.e., $\langle\langle T \rangle\rangle$) and a START concept (a subtype of STATE) for

the beginning state. Second, the convention of viewing a conditional test as a task (e.g., see Ref. 17) is also adopted, denoted as $\beta?$, where β is a condition. That is, $\beta?$ is a special control flow task that is invoked only when β is tested to be true, whereas $\neg\beta?$ is another special control flow task that is invoked only when $\neg\beta$ is tested to be true. A final state is denoted by attaching the monadic relation (final). The difference between selectional and conditional operators is that the expression $\mathscr{E}_1$ is not be performed unless the conditional test for $\beta_1?$ is tested to be true, both of which are not marked. The optional operator is treated as a special case of selectional operator. The iteration operator is implemented using three demons. The first demon is invoked by performing an expression $\mathscr{E}$, while the second demon is invoked by two input tokens, $\beta?$ and $\mathscr{E}$, to represent the notion of iteration condition. The third demon is to indicate the exit condition.

Method Specifications. A method in TBSM is to accomplish its parent task. Generally, a method consists of a collection of subtasks, a guard condition to invoke the method, and a TSE to specify the temporal relationship among those subtasks.

Details of task-based specifications in conceptual graphs can be found in Ref. 8. The library system is used as an example to illustrate various components of task-based specifications in conceptual graphs. The problem description of the library system is summarized below[18]:

Consider a small library database with the following transactions:

(1) Check out a copy of a book. Return a copy of a book.
(2) Add a copy of a book to the library. Remove a copy of a book from the library.
(3) Get the list of books by a particular author or in a particular subject area.
(4) Find out the list of books currently checked out by a particular borrower.
(5) Find out what borrower last checked out a particular copy of a book.

There are two types of users: staff users and ordinary borrowers. Transactions (1), (2), (4), and (5) are restricted to staff users, except that ordinary borrowers can perform transaction (4) to find out the list of books currently borrowed by themselves. The database must also satisfy the following constraints:

- All copies in the library must be available for check-out or be checked out.
- No copy of a book may be both available and checked out at the same time.
- A borrower may not have more than a predefined number of books checked out at one time.

The specifications in conceptual graphs for the task structure and tasks relevant to this example are shown in Figures 2 and 3.

III. REUSING TASK-BASED SPECIFICATIONS IN CONCEPTUAL GRAPHS

Requirements given by users are usually incomplete and inconsistent. Moreover, the problems of incompleteness and inconsistency are aggravated

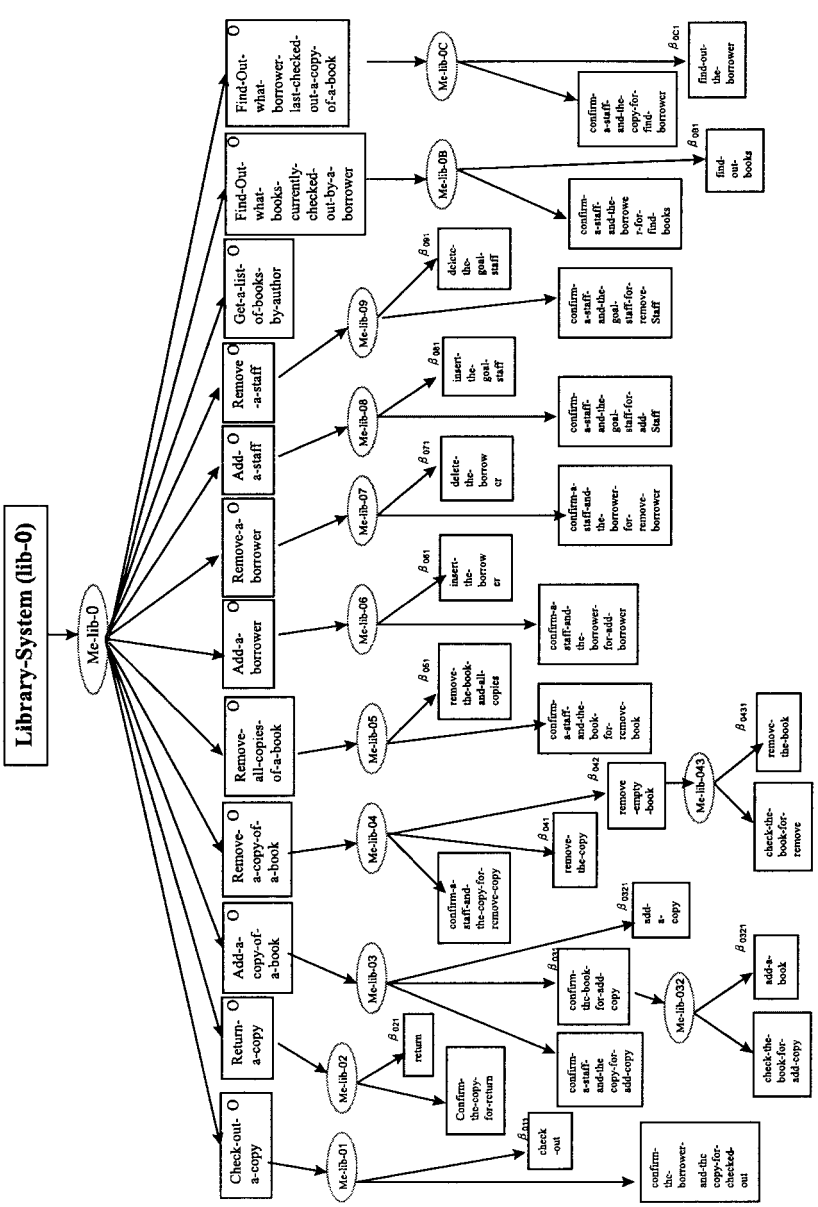


Figure 2. The task structure of the library system.

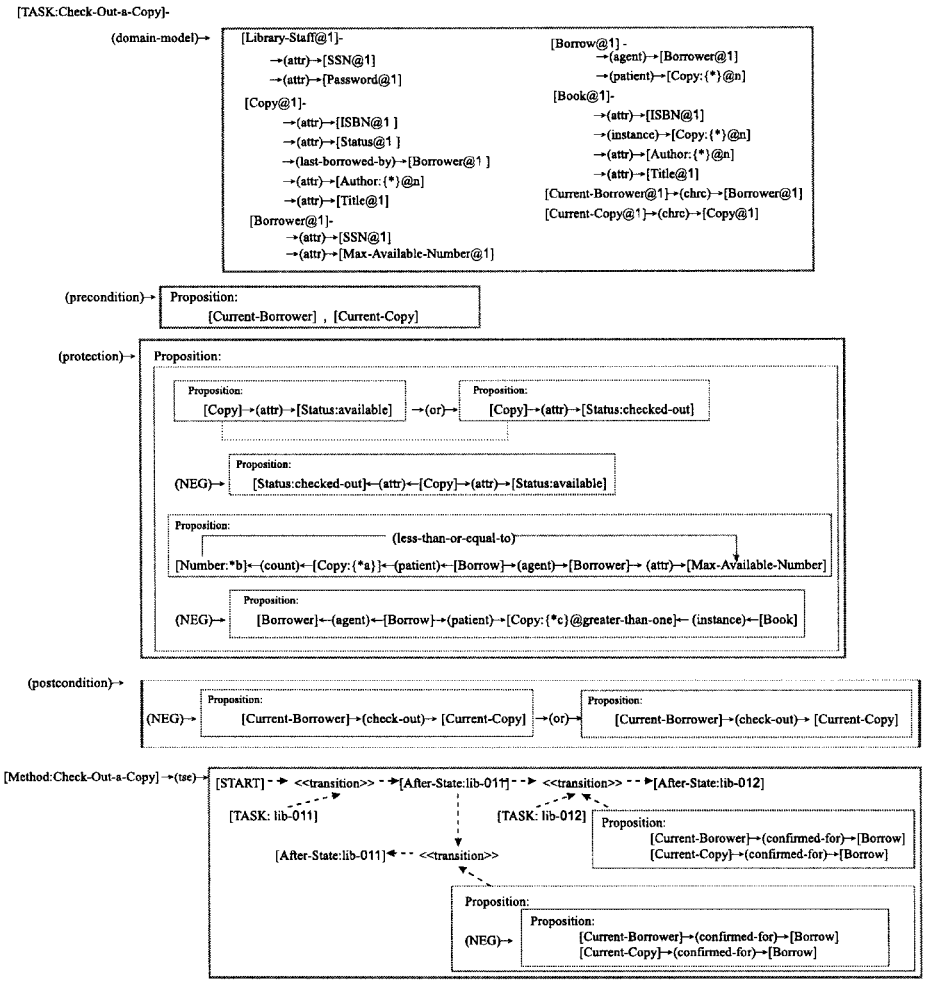


Figure 3. The task Check-Out-a-Copy.

since most of the requirements are expressed by informal representations such as natural languages. In our approach, we construct task-based specifications via the reuse of existing TBCGs to refine skeletal task-based specifications. A skeletal task-based specification is created based on the informal requirements from users. Figure 4 shows an example of skeletal task-based specifications, which consists of a task structure and tasks of an internal telephone directory system.

An overview of TBCG approach is shown in Figure 5. In TBCG, skeletal task-based specifications is refined one task at a time from the highest level task until all tasks are considered. Each task in the skeletal task-based specification would be matched against existing TBCG specifications by means of contextual

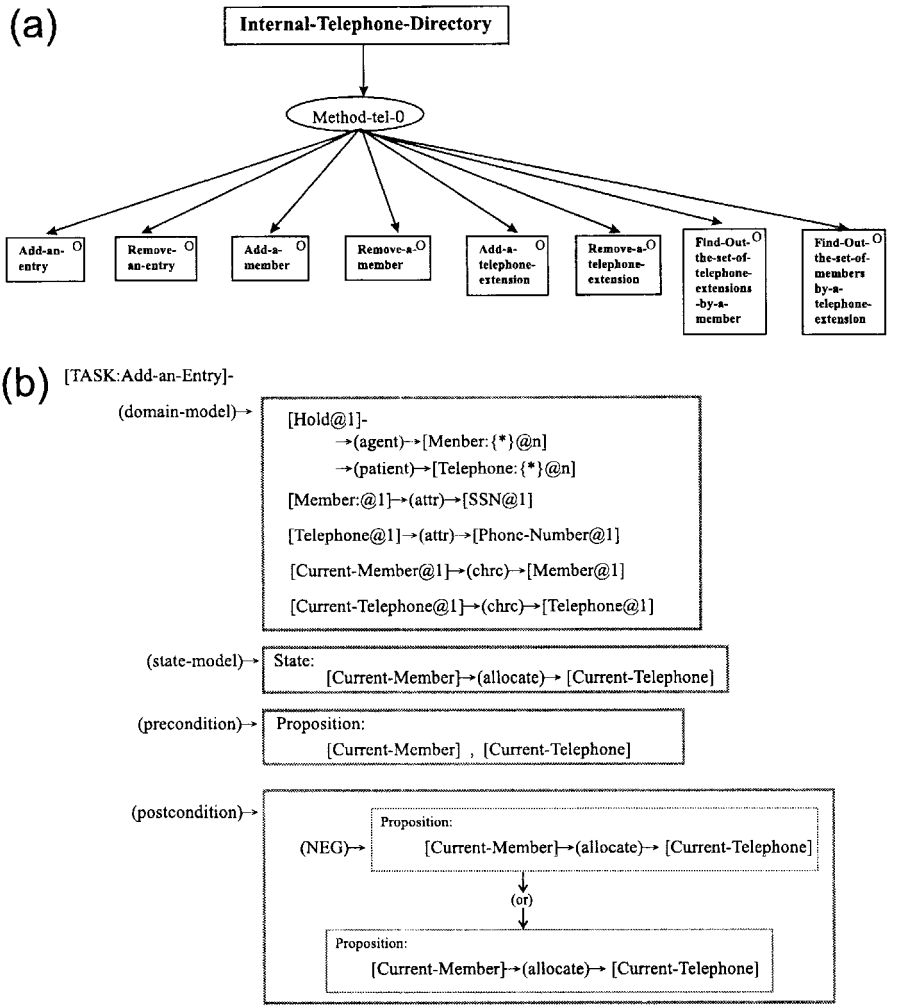


Figure 4. The skeletal task-based specification of the internal telephone directory system. (a) The task structure. (b) The task Add-an-Entry.

retrieval mechanism. This mechanism matches and retrieves reusable TBCG specifications based on the concept of contextual links. Through TBCG matcher, a list of candidate TBCG specifications will be retrieved. The order of these candidates can be ranked by computing their fuzzy similarities to the query graph and their reinforcement values. The reinforcement value expresses the past success rate of each candidate in this context. In the case of failure, the abnormal condition would be recorded and the next candidate TBCG would be selected in turn. A TBCG paraphraser and a verification facility are used to guide the adaptation of TBCG specifications. The paraphraser helps explain

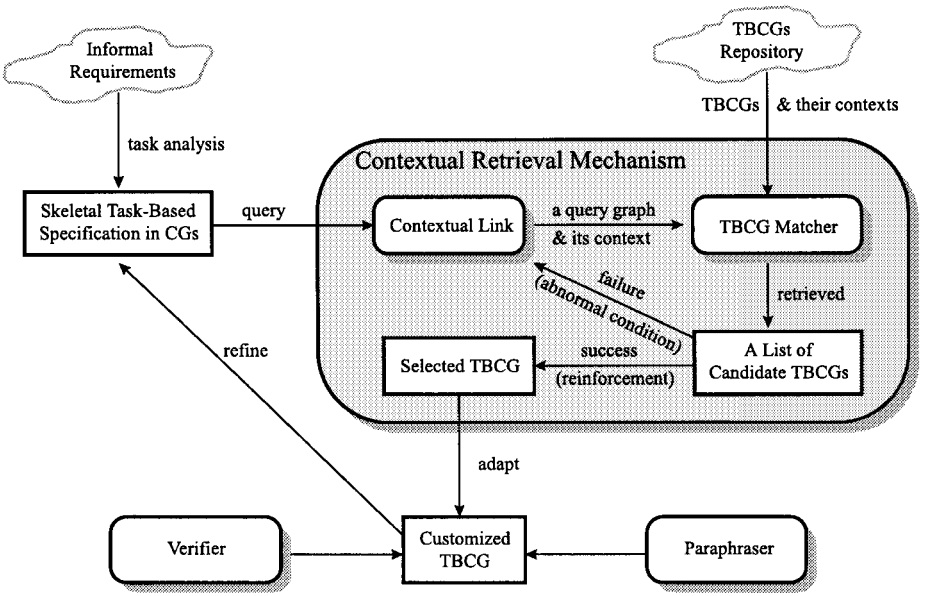


Figure 5. An overview of TBCG approach.

what and why the retrieved TBCG is doing. The verification facility checks the consistency within the TBCG specifications.†

IV. CONTEXTUAL RETRIEVAL MECHANISM

A context is an abstract entity which enables to relativize the truth of a formula.[19,20] That is, both the meaning and assertion of a formula may be different in variant contexts. Since context dependency is an important character of knowledge in requirements specifications, surrounding contexts play a crucial role in describing and reusing formal specifications. Unfortunately, most of the existing techniques for reusing formal specifications neglected the effect of surrounding contexts. To avoid retrieving many unwanted specifications whose contexts are irrelevant, more researches are required to introduce the notion of contexts into reusability.

In this section, we propose a contextual retrieval mechanism to match and retrieve reusable task-based specifications based on the concept of contextual links used in hypertext retrieval.[21] The contextual retrieval mechanism consists of a trigger condition, a set of contextual conditions, and a list of referents (see Fig. 6). A trigger condition can be a query graph which specifies a model of a task (e.g., domain, state, precondition, protection, postcondition, TSE, or

†Since verification is not the focus of this paper, for more detailed discussion of the TBCG verifier, refer to Ref. 9.

(Trigger-Condition $(Graph_{query}))$

(Contextual-Condition $(Graph_{condition1}\ Graph_{condition2}\ Graph_{condition3}))$

(Referents $(Task_{referent1}\ +5\ (Graph_{abnormal\text{-}condition1}\ +1))$

$(Task_{referent2}\ +3\ (Graph_{abnormal\text{-}condition2}\ +2))$

$(Task_{referent3}\ +2)$

$(Task_{referent4}\ +1)$

Figure 6. An example of the contextual retrieval mechanism.

method). Contextual conditions build up the context around the trigger condition. A referent contains a candidate graph that is retrieved from the TBCG repository, a reinforcement slot to indicate how often the candidate graph was successful, and a set of abnormal conditions together with failure rates resulting from them.

Contextual retrieval mechanism offers several advantages that are useful for the reuse of TBCG. First, reducing the number of possible candidates narrows the search by excluding out inappropriate specifications. Second, it provides incremental context acquisition by using user feedback to either reinforce or correct the system's knowledge in case of success or failure. Third, the notion of contextual conditions facilitates the computation of fuzzy similarity between two graphs. To apply the contextual retrieval mechanism to reusing TBCG specifications, there are still several issues that need to be addressed:

- A context cannot be completely described,[22] therefore, a way to construct the context around a TBCG specification is in need.
- As we are interested in spotting the similarity between two sets of graphs rather than that of two graphs, traditional graph matching methods for computing the similarity between two graphs are no longer sufficient. We will be forced to consider both the similarity between two TBCG specifications for *some* contextual condition and for *all* contextual conditions. Therefore, to address the dual concern for what may *possibly* apply and what must *necessarily* hold is needed in graph matching.
- After a query graph is matched, many candidate TBCG specifications may become eligible for reuse. Therefore, the computation for selecting a best fit candidate is required.

A. Describing Context

At any time, we might have only a partial description of the context surrounding a TBCG graph. Since no context can be completely described, the mechanism for building up a context incrementally is needed. Our approach provides a way to describe the content of a context in an incremental fashion. In TBCG, the context around a graph can be manifested by diverse views such as

domain, state, functionality, and behavior. Domain and state describe static properties of a context; whereas, functionality and behavior characterize dynamic properties of a context. Based on the sketchy knowledge given by users at the beginning, the context surrounding the query graph can be built roughly by adding views into contextual conditions. As the development of the system progressed, the user's knowledge is increased. Hence, either existing contextual conditions may be refined iteratively or new views can be incorporated into contextual conditions.

As was advocated by Delugach,[23] the multiple-viewed approach alleviates the difficulty in specifying and analyzing requirements of a complex system. Delugach also points out that the main drawback of many existing approaches is the lack of a formal basis from which to automatically analyze the resulting multiple-viewed requirements. To express the context surrounding a requirements specification, a uniform knowledge representation to capture the information in multiple views is in need. In TBCG, different views for constructing a context are represented in their conceptual graphical specification (i.e., in a uniform representation). By reducing the effort of translation between different views, the internal common representation makes the process of specifications matching much easier.

Consider the domain model of the task Check-Out-a-Copy in Library-System (see Fig. 3). The context of this graph is composed of multiple views containing a precondition, a protection, a postcondition, and a method. Each of them can be a contextual condition to match against contextual conditions in other TBCG specifications.

B. Computing Similarity

To compare specifications based on their graphical descriptions, TBCG quantifies its degree of fuzzy similarity by computing a distance between its corresponding graphs. Distances between two graphs can e obtained from aggregating semantic distances of all pairs of corresponding nodes. In conceptual graphs, the semantic distance between two types is evaluated by adding up paths from each type to their most specific common supertype in the type hierarchy.[24] The most specific common supertype of two types t_1 and t_2 indicates the most specific type that subsumes t_1 and t_2. Its formal definition is described below.

DEFINITION 1 (*The Most Specific Common Supertype of Two Types*). *Let the most specific common supertype of two types t_1 and t_2 be denoted by $G_t(t_1, t_2)$. We have*

$$G_t(T_1, t_2) = \{g | general(g, t_1) \wedge general(g, t_2) \wedge (\forall t \neq g)(general(t, t_1) \wedge general(t, t_2)) \Rightarrow general(t, g)\}$$

where $general(x, y)$ is a predicate to indicate that x is more general than y (i.e., x is a supertype of y).

Consider the concept type hierarchy in Figure 7, in which the most specific common supertype of *Borrower* and *Book* is *Entity*. Distances from *Borrower* and *Book* to *Entity* are 3 and 1, respectively. Therefore, the distance from *Borrower* to *Book* is 4.

DEFINITION 2 (*Distance Between Two Types*). *Let the distance between two types* t_1 *and* t_2 *be denoted by* $D_t(t_1, t_2)$. $D_t(t_1, t_2)$ *is defined as the distance of the shortest path from* t_1 *to* t_2 *in the type hierarchy. We have*

(1) $D_t(t_1, t_2) = D_t(t_1, G_t(t_1, t_2)) + D_t(t_2, G_t(t_1, t_2))$
(2) $D_t(t_1, t_2) = 0$. *If* $t_1 = t_2$
(3) $D_t(t_1, t_2) = +\infty$. *If* $G_t(t_1, t_2) = \mathsf{T}$ (*i.e., the universal type in conceptual graphs*)

A TBCG graph is composed of a set of nodes and links. Some nodes may be more valuable to the query graph than others. In our approach, a criticality can be assigned to each node in the query graph based on user's own judgments.

DEFINITION 3 (*Criticality of a Node in the Query Graph*). *The criticality* (*i.e., the degree of importance*) *of a node is quantified by length. The length indicates the maximal admissible path through which a node's type can match other types in the type hierarchy. The higher the criticality of a node, the larger the set of matchable nodes and the propositional weight of this node in the query graph. The criticality of a node n is denoted by* $Criticality_n(n)$.

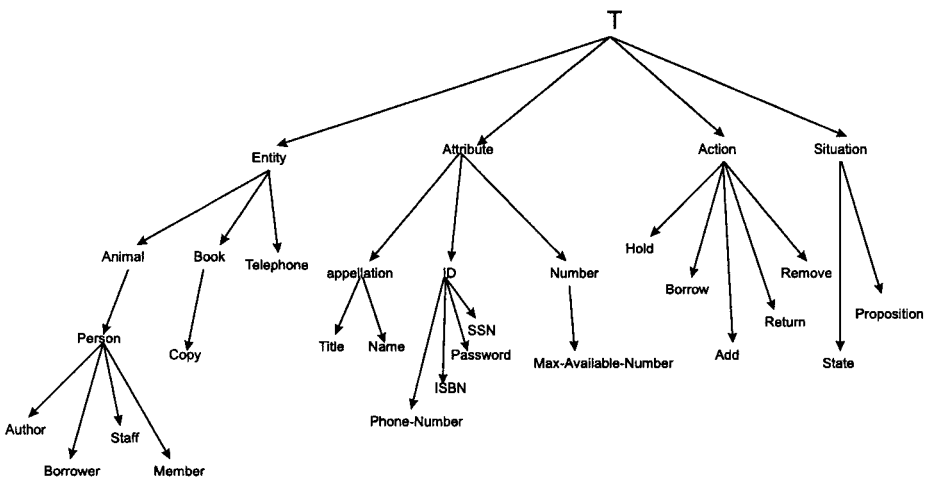


Figure 7. Concept type hierarchy.

The distance between nodes can be calculated by the semantic distance between their types, which serves as the basis for semantic match. Unlike syntactic match that must be satisfied exactly, semantic match permits a relaxed match for close semantics.[25]

DEFINITION 4 (*Distance Between Two Nodes*). *Let n_1 and n_2 be nodes in the query graph and the candidate graph, respectively. The distance between n_1 and n_2 is denoted by $D_n(n_1, n_2)$. We have*

$$D_n(n_1, n_2) = \min\{D_t(t_1, t_2), Criticality_n(n_1)\}$$

where t_1 and t_2 are type labels of nodes n_1 and n_2, respectively.

Our approach achieves graph matching through analogical reasoning that processes a complete network of knowledge rather than unrelated facts. Many matching techniques, especially the keyword-based matching, overleap the structural mapping between specifications. However, specifications are usually too complex to be described comprehensively using keywords. Analogical match in TBCG not only captures the meanings of nodes, but also compares the structural relationship between graphs. We formally define the concept of a compatible set in Definition 5, and outline an algorithm for finding compatible sets below.

DEFINITION 5 (*A Compatible Set Between Two Graphs*). *Let g_1 be a query graph that contains nodes n_i where $i = 1 \cdots p$, and g_2 be a candidate graph that contains nodes m_j where $j = 1 \cdots q$. A compatible set between two graphs g_1 and g_2 contains p pairs of nodes to indicate the possible correspondence of all nodes in g_1 and g_2.*

ALGORITHM 1. *Find compatible sets between two graphs G_1 and G_2.*

```
Create a product pair (N_i, N_j), ∀N_i ∈ G_1 and N_j ∈ G_2
generate a compatible set {(N_i1, N_j1), (N_i2, N_j2)}, whenever (N_i1, N_j1) and
(N_i2, N_j2) are compatible
repeat
    for each compatible set S_u do
        for each product pair P_v do
            if compatible(P_v, S_u) then join(P_v, S_u)
until no change occurs
```

The distance between two graphs can then be computed by summing up all the distances of correspondence pairs of nodes in their compatible sets.

DEFINITION 6 (*Distance Between Two Graphs*). *Let g_1 be a query graph that contains nodes n_i, $i = 1 \cdots p$, and g_2 be a candidate graph that contains nodes m_j*

where $j = 1 \cdots q$. *The distance between two graphs* g_1 *and* g_2 *is denoted by* $D_g(g_1, g_2)$. *We have*

$$D_g(g_1, g_2) = \min_{CS_g(g_1, g_2)} \left\{ \sum_{(n_i, m_j) \in CS_g(g_1, g_2)} D_n(n_i, m_j) \right\}$$

where $CS_g(g_1, g_2)$ *indicates a collection of compatible sets between* g_1 *and* g_2.

Similarly, the criticality of a query graph can be computed by summing up all the criticalities of nodes in the graph.

DEFINITION 7 (*Criticality of the Query Graph*). *Let g be a query graph that contains nodes* n_i, $i = 1 \cdots p$. *The criticality of the query graph g is denoted by* $Criticality_g(g)$. *We have*

$$Criticality_g(g) = \sum_{n_i \in g} Criticality_n(n_i)$$

The distance between a candidate graph and its query graph will decrease the degree of importance (i.e., the criticality) of the candidate graph. Therefore, the relative criticality of a candidate graph can be obtained by subtracting the distance from the criticality of the query graph.

DEFINITION 8 (*Relative Criticality of a Candidate Graph*). *Let* g_1 *be a query graph and* g_2 *be a candidate graph. The relative criticality of the candidate graph* g_2 *is denoted by* $Criticality_r(g_2|g_1)$. *We have*

$$Criticality_r(g_2|g_1) = Criticality_g(g_1) - D_g(g_1, g_2)$$

Due to the imperfect information inherited in contexts, it is difficult to come up with a crisp similarity degree between two contexts. Therefore, we adopt the notion of "necessity" and "possibility" in fuzzy logic as an interval to capture the fuzzy similarity between two contexts. Our approach defines fuzzy similarity between two graphs based on the degree of implication and the degree of consistency advocated by Enrique Ruspini.[26] The interpretation of the degree of implication and the degree of consistency is shown in Figure 8. By stretching the context of the query graph to encompass the context of the candidate graph, formulas that are true in the candidate graph may hold to a degree α in the context of the query graph. Let Q and and C be contexts surrounding the query graph q and the candidate graph c, respectively. We can say that the context C implies the context Q to a degree α, if and only if for *every* contextual condition x in C there exists a contextual condition x' in Q that is at least α-similar to it.

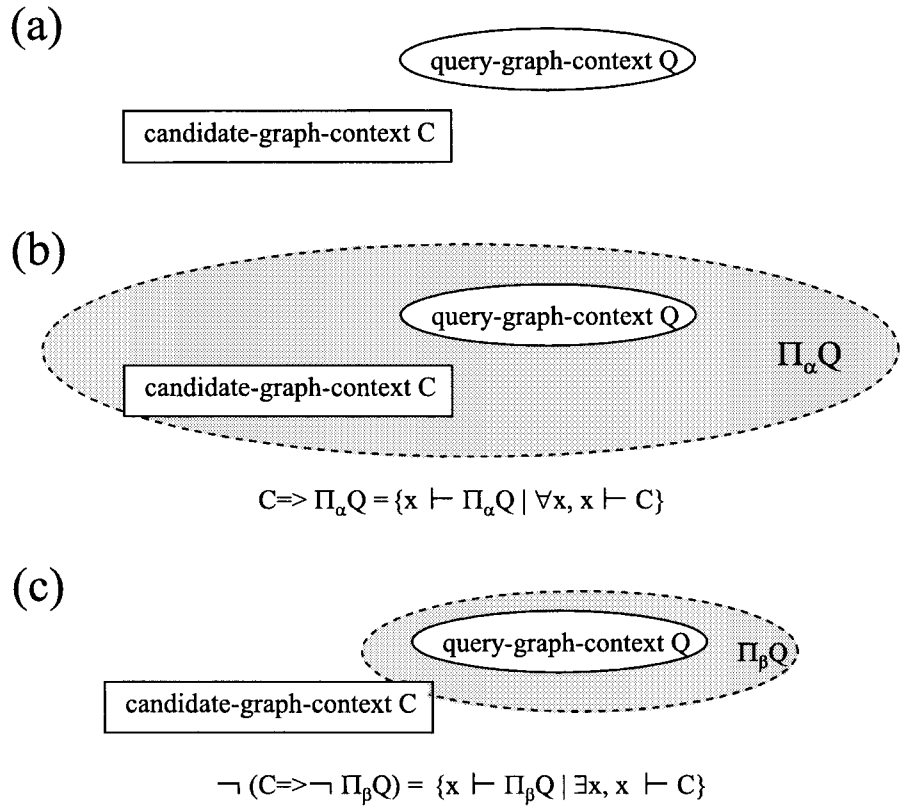


Figure 8. (a) Contexts of a query graph and a candidate graph. (b) The degree of implication. (c) The degree of consistency.

The formal definition of the degree of implication is described below:

$$C \Rightarrow \Pi_\alpha Q = \{x \vdash \Pi_\alpha Q | \forall x, x \vdash C\}$$

where Π_α means "possibly true to a degree α" and $\vdash$ stands for "is true in." On the other hand, the degree of consistency can be viewed as stretching the context of the query graph to intersect some contextual condition in the context of the candidate graph. We can say that the context C is consistent with the context Q to a degree β, if and only if there exists *some* contextual condition x in C and *some* contextual condition x' in Q that are at least β-similar. The degree of consistency is formally defined below,

$$\neg (C \Rightarrow \neg \Pi_\beta Q) = \{x \vdash \Pi_\beta Q | \exists x, x \vdash C\}$$

Figure 8(b) and (c) show the dual notion of the degree of implication and the degree of consistency, where the former reflects the degree of "inclusion" with

respect to the notion of *necessity* and the latter expresses the degree of "intersection" with respect to the notion of *possibility*. By means of comparing contextual conditions of the candidate graph with that of the query graph, an interval (i.e., [implication degree, consistency degree]) is formed to represent the similarity between two graphs.

DEFINITION 9 (*Fuzzy Similarity Between Two Graphs*). *Let g_1 be a query graph in a task k_1 and g_2 be a candidate graph in a task k_2. The task k_1 contains several contextual conditions x_i, $i = 1 \cdots p$ and the task k_2 contains contextual conditions y_j, $j = 1 \cdots q$. The fuzzy similarity between two graphs g_1 and g_2 is denoted by $FS_g(g_1, g_2)$. We have*

$$FS_g(g_1, g_2) = \left[\inf_{x_i \in k_1} \sup_{y_j \in k_2} \frac{Criticality_r(y_j|x_i)}{Criticality_g(x_i)}, \sup_{x_i \in k_1} \sup_{y_j \in k_2} \frac{Criticality_r(y_j|x_i)}{Criticality_g(x_i)} \right]$$

C. Selecting Candidate Task-Based Specifications in Conceptual Graphs Specifications

As the query graph has been matched against specifications in the repository, many candidate TBCG specifications may be retrieved. Each of them is recorded as a referent in the contextual link. To enhance the accuracy of selection, we adopt the notion of purpose-directed analogy[27] to ensure that the mapped analogy of each selected candidate can satisfy the purpose of the query task. Our approach consists of four stages: retrieve, explain, map, and justify (see Fig. 9). For each retrieved candidate specification, the system first explains how this base task satisfies its purpose. A task can satisfy its purpose only when the purpose is true in its after state description. The after state description of a task can be obtained from progressing through its TSE. States and conditions in TSE will be further explained by corresponding state models which specify the satisfied constraints. Next, the system maps the explanation derived for the base task to that of the target task, and attempts to justify that the mapped task satisfies its purpose. If the mapped TSE and explanation cannot justify the purpose of the query task, this base task will be excluded from the referent list. By explanation-based learning, we can justify that the selected reusable task is appropriate to achieve the purpose of the query task.

To rank order the list of referents, a rule-based approach is proposed. Several crucial factors are exploited to form our heuristics: reinforcement value, implication degree, and consistency degree. The heuristics implemented in the rule base are summarized below. First, rules for selecting a best fit reusable TBCG are focused on the reinforcement in referents. The reinforcement of a referent expresses how often the candidate graph was successful. The higher the reinforcement value of a candidate, the higher the past success rate of the candidate TBCG in the contextual link. Second, the candidate with the higher degree of implication would be selected under the situation of a same reinforcement. Finally, the degree of consistency is also considered as a basis for

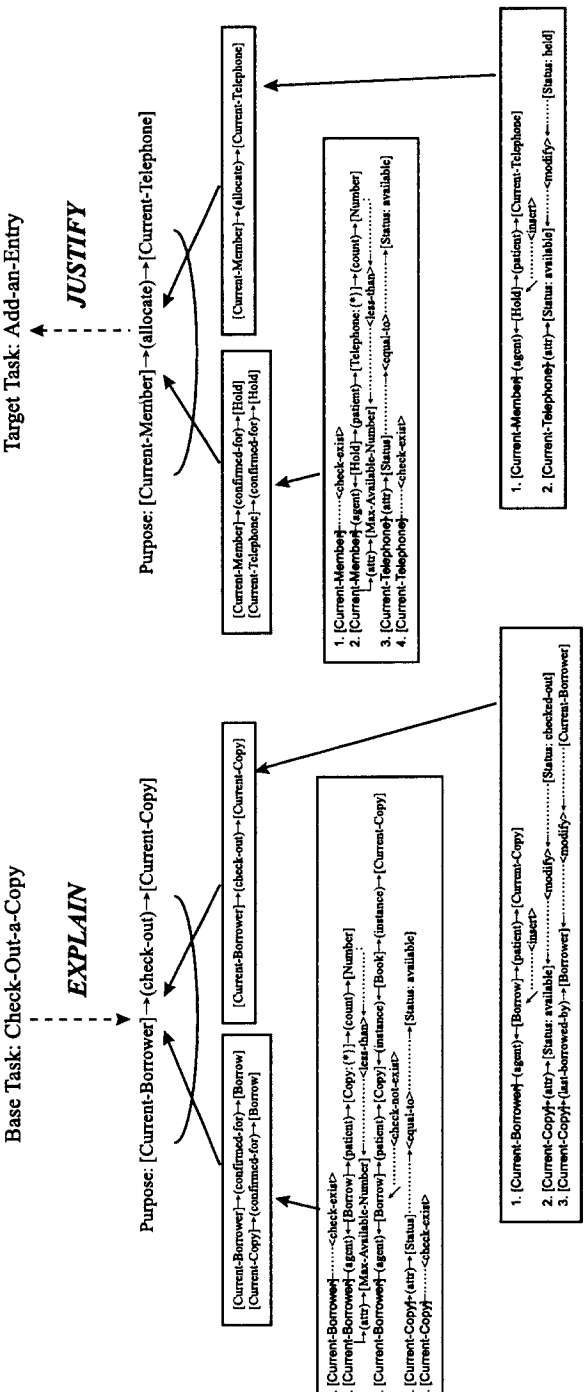


Figure 9. Purpose-directed analogy.

selection. The candidate with the higher degree of consistency would be selected when both the same reinforcement value and the same degree of implication are obtained.

After a referent has been selected, the user responds either "success" or "failure." The system automatically records this selection by adding +1 to the reinforcement slot in the case of success. When a failure occurs, the system attempts to obtain from users the reason for this failure. Users may state that some condition of the candidate graph is not suitable for the query graph. Hence, users either describe an abnormal condition or select an existing abnormal condition in the contextual link. If an abnormal condition is observed for three times, its negation will be added into contextual conditions in the contextual link automatically.

V. PARAPHRASING TASK-BASED SPECIFICATIONS IN CONCEPTUAL GRAPHS SPECIFICATIONS

To make the reuse of TBCG specifications easy, it is important that the specifications can be explained to and understood by the users. With the direct mapping to natural languages, the conceptual graph serves as an intermediate language for translating TBCG specifications to natural language.‡ Paraphrasing TBCG specifications provides an explanation to enhance the user's understanding for the retrieved graph. Understanding what and why the retrieved TBCG is doing would facilitate the adaptation of the retrieved TBCG.

Important features in translating CGs to English in TBCG paraphraser include:

- Subject–verb agreement. The paraphraser obeys basic English grammar rules, including the subject and verb agreement rule. For example, [*Person*: {∗}@*many*] → (*eat*) → [*Pie*] will be translated into "Many persons eat one pie."
- Treatment of enclosure. An enclosure delimits a group of information. Our approach deals with several types of enclosure, including proposition, graph, state, etc. For example, (*NEG*) → [*Proposition*: [*Borrower*: {*John*, *David*}@*two*] → (*check-out*) → [*Book*: {∗}@3-*to*-5]] will be translated into "There is a negation of the proposition as follows: Two borrowers John and David check out 3 to 5 books."
- Isolated concepts. The system will precede the translation with the phrase "There exist(s)" as shown in the next example. [*Person*: *John*] will be translated into "There exists one person John."
- Actors. Actors are used for expressing constraints in state models of TBCG specifications. This is achieved by preceding the word "must" in the sentence. [*Number*: 20] → ⟨*equal-to*⟩ → [*Number*: ∗ *x*] will be translated into "The number $*x$ must be equal to the number 20."
- Demons. Demons represent state transitions in TSEs and methods of TBCG specifications. Figure 10 shows an example of paraphrasing the method of the task Check-Out-a-Copy (in Fig. 11). The paraphraser explains the sequence, trigger events, and state changes in the method of T_8.

‡This is achieved by adopting the work by Slagle et al.[28] that translates CGs to English.

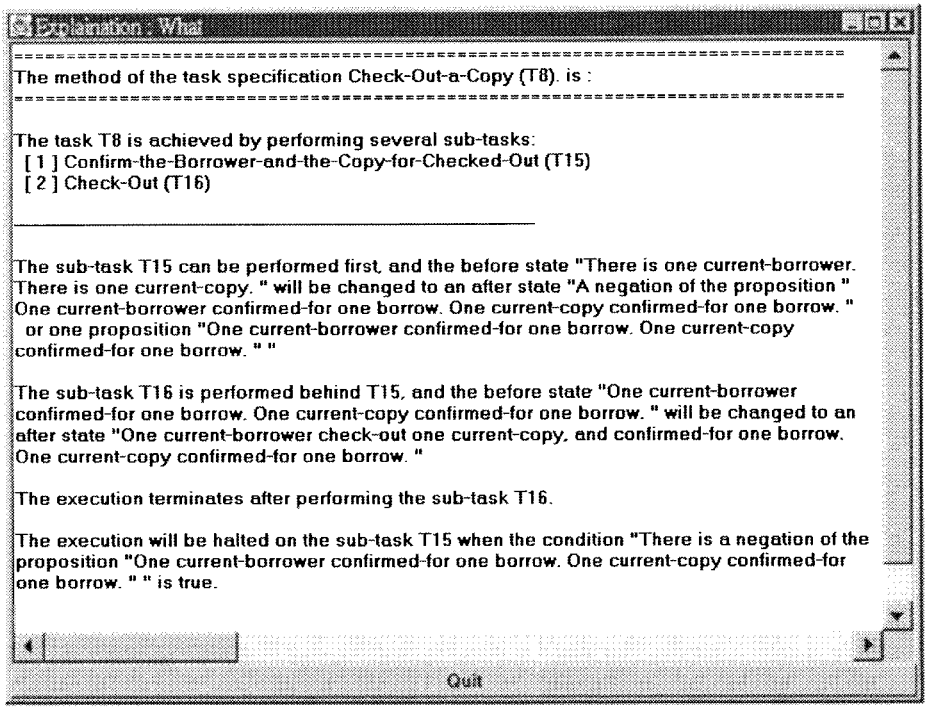


Figure 10. Paraphrasing the method of the task Check-Out-a-Copy.

VI. IMPLEMENTATION

The TBCG case tool provides a Java interface as the front-end. Users can use this tool on a HTML browser in the Internet. We use CLIPS, an expert system development tool, to implement the reasoning engine for the contextual retrieval, graph-matching, and paraphrasing components, each of which is described below (see Fig. 12 for an overview of the system).

We adopted Java as the programming language for the TBCG case tool. It provides the capability to run on multiple platforms and in the Internet. Java calls C library at run-time by the native method. We use this characteristic to invoke the CLIPS engine when users request the tool to match or retrieve a TBCG specification at run-time. The client provides the basic notation of the conceptual graph and task-based specifications (e.g., concept, conceptual relation, actor, demon, and propositions), and users use these notations to specify the specifications on the workplace directly. Users can construct the task structure and the type hierarchy of a system and specify the model (e.g., domain, state, precondition, postcondition, protection, TSE, method model) of tasks in the system. The canonical graph of a type can also be described. Figure 11 is a demonstration of the method of the task Check-Out-a-Copy.

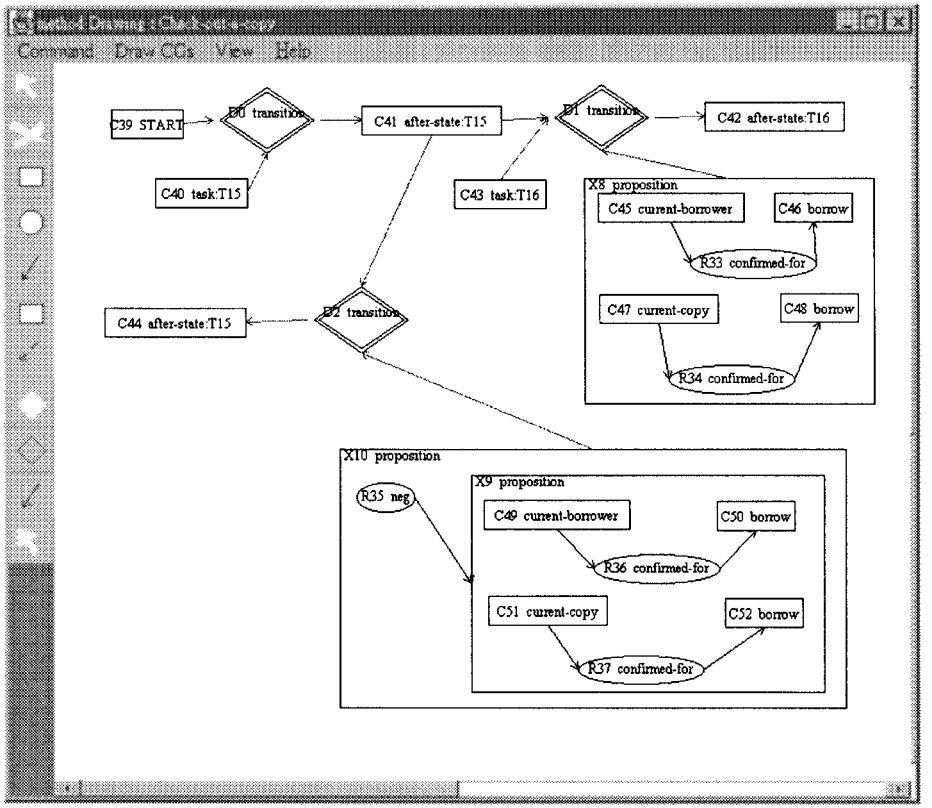


Figure 11. The method of the task Check-Out-a-Copy.

When users finish inputting the TBCG specifications on the client, it sends these data to the server through the network. After the server processes the received data, it sends the result back to the client. The client then shows the result as conceptual graphs for users to evaluate. Users evaluate the retrieved graphical specifications as an aid for reusing. Figure 13 is a demonstration of matching the domain model of task Add-an-Entry with contextual conditions of precondition and postcondition (in Fig. 4). Information appearing in the Referent List window indicates that a task T_8 is a referent with a reinforcement value 4, and an interval of $[0.5625, 0.7840]$ for representing the degree of similarity.

When the message dispatcher in the server receives the messages from the client, it analyzes the messages to determine whether to invoke the JDBC layer or the CLIPS engine. If the message is a database operation request, such as inserting a conceptual graph or deleting a graph from a specification, the message dispatcher uses SQL APIs on the database repository through the JDBC layer. JDBC is a set of SQL-based APIs that Java uses to retrieve and

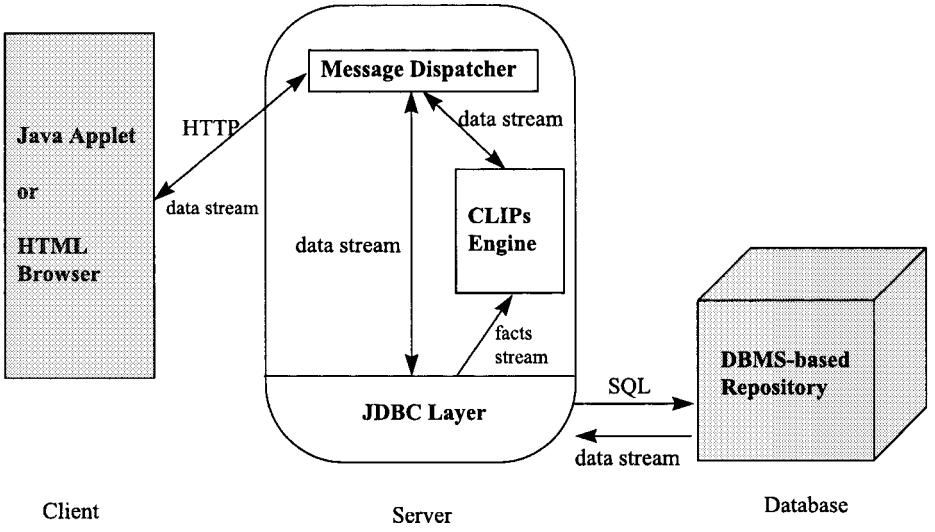


Figure 12. An overview of the system.

modify the data stored in the DBMS. However, if the messages are CLIPS operation requests, then the server makes up the facts necessary to the CLIPS program and invokes the CLIPS engine to execute. The CLIPS uses these facts to do the contextual retrieval, graph matching, and paraphrasing the TBCG specification. The server then sends these results back to the client.

We use DBIII as the database file format. This is because the ODBC (open data base connectivity) takes it as one of the standard data exchange format. ODBC can work with the standard database through SQL, and JDBC can incorporate with ODBC. In this way, Java can manipulate the database through JDBC and ODBC. In addition, we can use SQL statement to query the graph data much easier.

VII. CONCLUSION

In this paper, we propose an approach to reusing the requirements specification, called task-based specifications in conceptual graphs (TBCG). In TBCG, task-based specification methodology is used to serve as the mechanism to structure the knowledge captured in conceptual models, and conceptual graphs are adopted as the formalism to express requirements specification. TBCG provides several mechanisms to facilitate the reuse of formal specifications: a contextual retrieval mechanism to support context-sensitive specifications retrieval and incremental context acquisition, a graph matching mechanism to

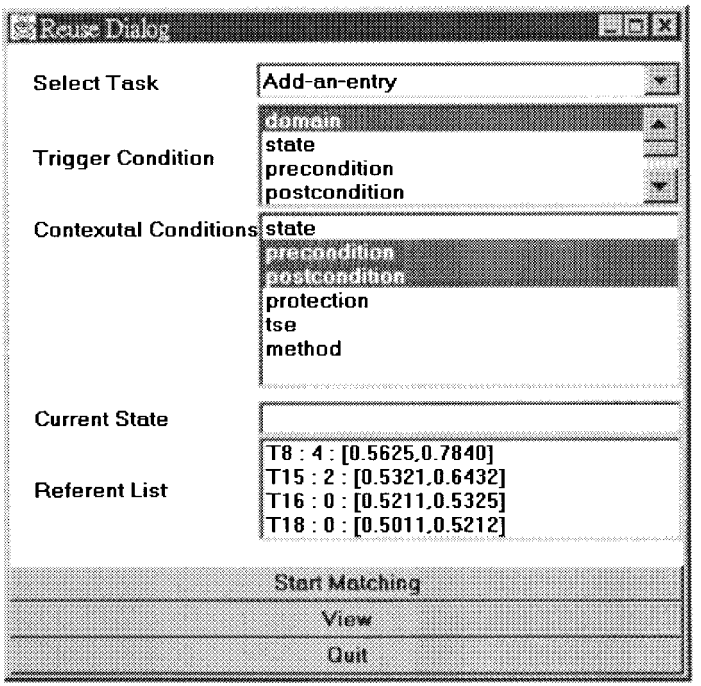


Figure 13. The result of the graph matching.

compute the similarity between two graphs based on the semantic match and fuzzy logic, and a paraphraser to serve as an explanation mechanism to explain retrieved specifications.

Contextual retrieval mechanism offers several benefits that are useful for reusing formal specifications:

- To exclude irrelevant specifications to reduce the number of possible candidates for selection. By this way, the efficiency of reusing requirements specification will be increased.
- To provide an incremental context acquisition to help elicit detailed information of contexts. The multiple-viewed approach also helps to build up a context incrementally.
- Support the computation of fuzzy similarity, semantic match, and analogical match. The computation of fuzzy similarity not only facilitates the matching between two sets of graphs but also deals with the uncertainty inherent to contexts that cannot be completely described. Unlike syntactic match that must be satisfied exactly, semantic match in TBCG permits relaxed match for close semantics. Unlike keyword-based match that overleaps the structural mapping, analogical match in TBCG addresses the structural relationship between graphs.
- To offer a plug-in environment that facilitates the reuse of different specifications. The matching engine in contextual retrieval mechanism can be used as a plug-in component for different kinds of modeling notations.

- To provide a TBCG paraphraser to explain what and why the retrieved TBCG specification is doing. The explanation mechanism offers a basis to increase user's understanding and guide the adaptation of retrieved specifications.

Our future research plan will consider the following tasks: (1) to extend the current framework to fuzzy logic for modeling imprecise requirements, and (2) to apply the contextual retrieval mechanism to unified modeling language (UML).

We thank Yu Fei for his early work on this subject. This research is partially sponsored by National Science Council (Taiwan, R.O.C.) under Grant NSC86-2213-E-008-006.

References

1. Krueger, C. W. ACM Comput Surveys 1992, 24, 131–183.
2. Mili, H.; Mili, F.; Mili, A. IEEE Trans Software Eng 1995, 21, 528–562.
3. Ostertag, E.; Hendler, J.; Diaz, R. P.; Braun, C. ACM Trans Software Eng Methodology 1992, 1, 205–228.
4. Cheng, B. H. C.; Jeng, J. J. IEEE Trans Knowledge Data Eng 1997, 9, 341–349.
5. Maiden, N. A.; Sutcliffe, A. G. Software Eng J 1996, 281–292.
6. Zaremski, A. M.; Wing, J. M. ACM SIGSOFT 1995, 6–17.
7. Lee, J. Int J Intell Syst 1997, 12, 167–190.
8. Lee, J.; Lai, L. F.; Huang, W. T. IEEE Expert 1996, 11, 60–70.
9. Lee, J.; Lai, L. F. Information and Software Technology 1998, 39, 913–923.
10. Lee, J.; Yen, J.; Passtor, J. Int J Intell Syst 1994, 9, 839–851.
11. Yen, J.; Lee, J. IEEE Expert 1993, 8, 8–15.
12. Chandrasekaran, B. AI Mag 1990, 11, 59–71.
13. Sowa, J. F. Conceptual Structures: Information Processing in Mind and Machine; Addison-Wesley: Reading, MA, 1984.
14. Delugach, H. S. Dynamic Assertion and Retraction of Conceptual Graphs, Sixth Annual Workshop on Conceptual Graphs; 1991; pp 15–26.
15. Eshner, D.; Hendler, J.; Nau, D. Incremental Planning Using Conceptual Graphs, Sixth Annual Workshop on Conceptual Graphs; 1991; pp 283–294.
16. Pfeiffer, H. D.; Hartley, R. T. In Conceptual Structures: Current Research and Practice; Nagle, T. E.; Nagle, J. A.; Gerholz, L. L.; Eklund, P. W., Eds.; Ellis Horwood: England, 1992; pp 87–107.
17. Warren, D. H. D. Generating Conditional Plans and Programs, Summer Conference on Artificial Intelligence and Simulation of Behavior; 1976; pp 344–354.
18. Wing, J. M. IEEE Software 1988, 66–76.
19. McCarthy, J.; Buvac, S. Formalizing Context: Expanded Notes; Tech Rep STAN-CS-TN-94-13; Computer Science Department, Stanford University, Stanford, CA, 1994.
20. Akman, V.; Surav, M. AI Mag 1996, 55–72.
21. Boy, G. A. Indexing Hypertext Documents in Context, Third ACM Conference on Hypertext; 1991; pp 51–61.
22. Guha, R. V. Contexts: A Formalization and Some Applications; Tech Rep ACT-CYC-423-91; MCC, Austin, TX, 1991.
23. Delugach, H. S. J Syst Software 1992, 19, 207–224.
24. Foo, N.; Garner, B. J.; Rao, A.; Tsai, E. Conceptual Structures; Current Research and Practice; Ellis Horwood: London, 1992; pp 149–154.

25. Ryan, K.; Mathews, B. Matching Conceptual Graphs as an Aid to Requirements Re-Use, IEEE International Symposium on Requirements Engineering; IEEE Press: Piscataway, NJ, 1992; pp 112–120.
26. Ruspini, E. Int J Approx Reasoning 1991, 5, 45–88.
27. Kedar-Cabelli, S. Readings in Machine Learning; Shavlik, J. W.; Dietterich, T. G., Eds.; Morgan Kaufmann: San Mateo, CA, 1990; pp 647–656.
28. Dogru, S.; Slagle, J. R. A System that Translates Conceptual Structures into English, 7th Annual Workshop on Conceptual Structures; 1992; pp 283–292.

APPENDIX: A SUMMARY OF CONCEPTUAL GRAPHS AND TASK-BASED SPECIFICATIONS IN CONCEPTUAL GRAPHS NOTATIONS

For a summary of conceptual graphs and task-based specifications in conceptual graphs see Figures A.1–A.4.

Summary of Conceptual Graphs Notation

Nodes in Conceptual Graphs:

Node	Notation	Explanation
concept	[T:R]	A concept contains a type label to indicate the concept type and a referent field to specify the instance.
relation	(T)	A relation specifies the relationship between concepts.
actor	<T>	An actor represents a process that can change the referent of its output concepts based on its input concepts.
demon	<<T>>	A demon causes each of its output concepts to be asserted, and each of its input concepts to be retracted.

** Remarks: (1) T denotes the type label of a node. (2) R denotes the referent field of a concept. **

Referents in Concept Nodes:

Referent Types	Expression	Semantics	Example	English Expression
Generic	[T:*]	$\exists x, x \in T$	[Cat:*]	a cat or some cat
Individual	[T:#n]	$\#n \in T$	[Cat:#02]	the cat #02
Proper Name	[T:Name]	$Name \in T$	[Cat:Kitty]	the cat named Kitty
Unique	[T:@1]	$\exists! x, x \in T$	[Cat:@1]	one and only one cat
Set	[T:{N_1,...,N_n}]	{N_1,...,N_n}$\subset$T	[Cat:{Kitty,Garfield}]	Kitty and Garfield
Plural Set	[T:{*}]	$\exists S, S \subset T$	[Cat:{*}]	cats or some cats
Counted Plural Set	[T:{*}@n]	$\exists \Sigma, \Sigma \subset T$ and $\|S\|=n$	[Cat:{*}@4]	four cats
Definite Set	[T:#{*}]	$\#S \subset T$	[Cat:#{*}]	the cats
Universal	[T:∀]	$\forall x, x \in T$	[Cat:∀]	every cat
Negative	[T:¬]	$\forall x, x \notin T$	[Cat:¬]	no cat

** Remarks: (1) T denotes the type label of a concept. (2) S denotes a set. (3) #S denotes a designated set. **

Figure A.1.

Basic Concepts in TBCG

Concept Type	The General Form	Explanation
[TASK]	[TASK]- (domain-model)→[GRAPH] (state-model)→[GRAPH] (precondition)→[Proposition] (protection)→[Proposition] (postcondition)→[Proposition] (tse)→[GRAPH]	A concept [TASK] represents a task which has a domain model, a state model, a functional model, and a behavior model. The relation (domain-model) is used to connect a task concept [TASK] with its domain model which is represented as a graph concept [GRAPH]. The referent of a concept [GRAPH] is a context composed of a conceptual graph. The functional model contains preconditions, protections, and postconditions. The behavior model is spcified by task-state expressions (TSEs).
[Method]	[Method]- (parent-task)→[TASK] (guard-condition)→[Proposition] (part)→[TASK:{*}] (tse)→[GRAPH]	A concept [Method] represents a method to accomplish its parent task. A method consists of a collection of subtasks, a guard condition to invoke the method, and a TSE to specify the temporal relationship among those subtasks.
[Constraint]	[Constraint]- (part)→[Argument:{*}] (attr)→[Local | Global]→(chrc)→[Scope] (attr)→[C_0| · ·|C_n]→(chrc)→[Strength] (attr)→[Single-Valued | Multivalued] →(chrc)→[Mapping] (attr)→[Positive | Negative]→(chrc)→[Property] (accm)→[Rule]→(body)→[[Input]⋯> <actor>⋯> [Output]]	A concept [Constraint] represents a constaint which is characterized by parts, facets, and rules. Parts are inputs and outputs to a constraint. These facets are scope, strength, mapping, and property. The relation (attr) is used to connect a concept and its attribute. The relation (accm) is used to connect a constraint and its rule that consists of a rule name and a rule body. The rule body is implemented by an actor that can attach procedural knowledge to change the referents of their output concepts based on their input concepts.

Figure A.2.

State Transitions in TBCG

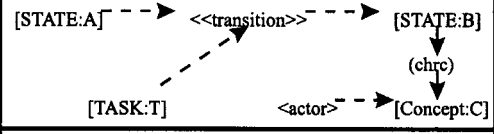
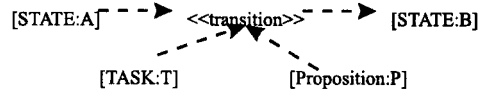
Different Types	The General Form	Explanation
with actors overlay (in state model)		The state model means that "When the task T is performed and the constraint overlay (i.e. actor) is satisfied, the state A will be changed to state B."
with conditions (in TSE)		The state transition means that "When proposition P is true and the task T will be performed, the state A will be changed to state B."

Figure A.3.

TSE Operators in TBCG

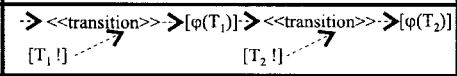
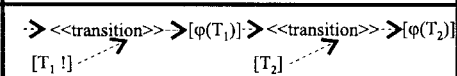
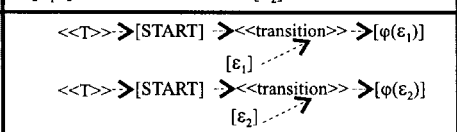
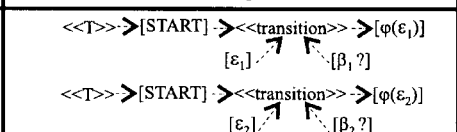
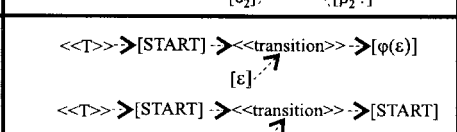
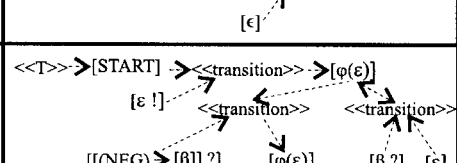
Operator	Syntax	In Conceptual Graphs	Explanation
Immediately Follow	$[T_1,T_2]$		The task T_1 is invoked first, and the task T_2 immediately follows the task T_1.
Follow	$[T_1;T_2]$		The task T_2 follows the task T_1 after a sequence of intermediate, unspecified tasks (T_1,...,T_2).
Selectional	$[\varepsilon_1 v \varepsilon_2]$		The initiator demon <<T>> and a [START] concept are used for expressing the beginning state. ε_1 and ε_2 are TSEs representing two mutually exclusive control flows.
Conditional	$(\beta_1\varepsilon_1 v \beta_2\varepsilon_2)$		β? is a special control flow task that is invoked only when β is tested to be true, where β is a condition. The expression ε_1 will be performed if the condition test for β_1? is tested to be true. Similarly, the expression ε_2 will be performed if the condition test for β_2? is tested to be true. β_1 and β_2 are mutually exclusive.
Optional	$[\varepsilon]$		ϵ is a null operation. ϵ and ε are representing two mutually exclusive control flows.
Iteration	$(\beta\varepsilon)^*$		β? is a special control flow task that is invoked only when β is tested to be true, where β is a condition. The expression ε will not be performed iteratively unless the conditional test for β? is tested to be true.

Figure A.4.