Analogical Reuse of Requirements Frameworks

Philippe Massonet and Axel van Lamsweerde

Université catholique de Louvain, Département d'Ingénierie Informatique
B-1348 Louvain-la-Neuve (Belgium)
{phm,avl}@info.ucl.ac.be

(Proceedings RE'97 - 3rd IEEE International Symposium on Requirements Engineering, Jan.1997, pp. 26-37)

Abstract. *Reusing similar requirements fragments is among the promising ways to reduce elaboration time and increase requirements quality. This paper investigates the application of analogical reasoning techniques to complete partial requirements specifications. A case base is assumed to be available; it contains requirements frameworks involving goals, constraints, objects, actions, and agents from systems already specified. We show how a rich requirements meta-model coupled with an expressive formal assertion language may increase the effectiveness of analogical reuse. An acquisition problem is first specified by the requirements engineer as a query formulated in the vocabulary of the specification fragments built so far. Source cases and partial mappings are found by query generalization followed by search through the case base. Once analogies have been confirmed, mappings are completed by use of relevance rules that distinguish in the formal assertions what is relevant to the analogy from what is irrelevant. Best analogies are then selected and extended in such a way that logical properties of the answers to the query may be verified, thus increasing confidence in the analogy. The approach is illustrated by analogical acquisition of specifications of a meeting scheduler in the KAOS goal-oriented specification language.*

Keywords: Specification reuse, analogical reasoning, goal-driven requirements engineering, formal methods.

1. Introduction

Software reuse is intended to decrease software development costs and increase software quality [Kru92]. Specification reuse has in particular been given some attention recently as it provides support for the difficult, critical task of requirements elaboration; moreover one may expect that the reuse of a specification entails the reuse of the software components implementing the specification. In [Fin88], a technique based on matching of simple patterns and causal chains is suggested for reusing specifications expressed in the CORE notation. The Requirements Apprentice allows specifiers to build and check specifications written in a frame-based language by instantiation and specialization of reusable domain-specific clichés [Reu91]. In the same spirit, [Rya93] introduces a graph matching technique as a basis for reusing previously developed specifications expressed by conceptual graphs.

Specification reuse raises a number of technical issues, notably, the representation of specifications for reuse, the organization of the library of reusable components, the retrieval of appropriate specifications through component matching, and the customization of the specification obtained. Various approaches have been proposed to address those issues, e.g., specialization graphs with multiple inheritance [Reu91], faceted classification [Dia89, Ost92], similarity-based retrieval with various similarity measures [Spa93, Ost92, Lau94], and signature matching [Zar93]. The use of analogical reasoning technology [Hal89] for specification reuse and validation has also been suggested [Mai91], [Sut93]. Requirements analysts do rarely start from scratch when they model and specify a new system; they tend to reuse past experience with similar cases to fill in the holes. Analogical reasoning techniques seem therefore quite appropriate in this context.

An analogy is defined as a *mapping* from elements of a *source* domain to elements of a *target* domain, where each domain contains related knowledge and facts. Analogical inference consists of mapping well defined source elements to partially defined or unknown target elements so that additional knowledge about the latter can be obtained by source-target transposition. The source and target elements are called source and target analogs, respectively. The following (overlapping) steps can be identified in an ideal analogical inferencing process [Lam91]. *Presentation:* the problem to be solved is first defined which results in the identification of target elements; *Recognition:* source analogs are identified from target elements; *Mapping:* an analogical match is established between the source and target analogs [Gen88], [Fal89], [Hol89]; *Transfer:* relevant knowledge about the source is transposed to the target according to the match found; *Validation:* the transferred features are checked for appropriateness in the new target situation [Fal87], and revised if necessary; *Consolidation:* the analogy may be analyzed against utility criteria and generalized for later reuse in new situations [Min88].

In the various approaches to specification reuse we are aware of, the aspects of a specification that are formalized and used for retrieval are limited to a global, syntactic "declaration" level --e.g., frame slot names/values, facet names/values, names of nodes and edges in a graph, signature of an abstract data type, etc. The actual requirements on the declared concepts/attributes/links/operations are not formally represented nor used; they are considered as informal assertions annotating the declared objects.

In this context the paper has three objectives:

- to show how a richer ontology for declaring requirements, combined with formal assertions to express them, may result in a more accurate and meaningful level of specification matching and reuse,
- to propose techniques for adapting the various steps of the analogical process above to the problem of reusing *formal* requirements specifications,
- to extend the scope and granularity of reuse from single concept to a collection of related concepts (called framework) though incremental analogy extension.

The perspective in this paper is more on using analogies to complete and/or validate partial specifications already elaborated, rather than starting from almost empty targets. Source and target specifications are written in the KAOS language, a multi-paradigm specification formalism which combines semantic nets [Bra85] for the conceptual modelling of goals, consraints, agents, objects and operations in the system; temporal logic [Man92], [Koy92] for the specification of goals, constraints and objects; and state-based specifications [Pot91] for the specification of operations. The language has a rich ontology which is explicitly defined and accessible at the meta-level [Dar93].

The paper is organized as follows. Section 2 introduces some minimal background on the KAOS framework. Section 3 provides an overview of our approach by outlining the different steps of the analogical reuse process. Section 4 then explains how a reuse problem is defined as an analogical query. Section 5 shows how source cases are identified from such a query; partial, declaration-level target-source mappings are elaborated therefrom. Section 6 then shows how the formal assertion level is used to complete such mappings on more semantic grounds; a completed mapping produces an answer to the original query and provides the basis for analogical transfer. Section 7 summarizes our results and outlines ongoing work on formal verification, extension, and consolidation of analogies.

2. Requirements specification in Kaos

The KAOS methodology provides a specification language to capture *why*, *who*, and *when* aspects in addition to the usual *what* requirements; a goal-driven elaboration method; and meta-level knowledge used for local guidance during method enactment. Hereafter we introduce some features of the language and the meta level that will be used later in the paper; see [Dar93], [Lam95], [Dar96] for more details.

2.1 The Kaos language

The specification language provides constructs for capturing a rich variety of *concepts* involved in the requirements engineering lifecycle, namely, goals, constraints, agents, entities, relationships, events, actions, views, and scenarios. There is one construct for each type of concept. The following types of concepts will be used in the sequel.

- *Object:* an object is a thing of interest in the domain whose instances may evolve from state to state. It is in general specified in a more specialized way -as an *entity*, *relationship*, or *event* according as the object is autonomous, subordinate, or instantaneous, respectively. Objects are described formally by invariant assertions.
- *Action:* an action is an input-output relation over objects; action applications define state transitions. Actions may be *caused/stopped* by events. They are characterized by pre-, post- and trigger conditions.
- *Agent:* an agent is an object acting as a processor for some actions. An agent *performs* an action if it is effectively allocated to it; the agent *knows* an object if the states of the object are made observable to it. Agents can be humans, devices, programs, etc.
- *Goal:* a goal is an objective the system should meet. *Refinement* links relate a goal to a set of subgoals. The goal refinement structure for a given system is in general an AND/OR directed acyclic graph. Goals often *conflict* with others. Goals *concern* the objects they refer to.
- *Constraint:* a constraint is an implementable goal, that is, a goal that can be assigned to some individual agent in the system. Goals must be AND/OR *refined* into constraints. Constraints in turn are AND/OR *operationalized* by actions and objects through strengthenings of their pre-, post-, trigger conditions and invariants, respectively. Alternative ways of assigning responsibilities for a constraint are captured through AND/OR *responsibility* links; the actual assignment of agents to the actions that operationalize the constraint is captured in the corresponding *performance* links.
- *Case:* a case is an aggregation of interrelated instances of the various concept types above. Cases capture well understood areas, abstract or real, with distinguishable boundaries. They may be organized into taxonomies; concepts from a more specific case are linked to corresponding concepts of the same type in a more general case through *IsA* inheritance links.

Each construct in the KAOS language has a two-level generic structure: an outer semantic net layer for *declaring* the concept, its attributes and its various links to other concepts [Bra85]; an inner formal assertion layer for *formally defining* the concept. Goals, constraints and object invariants are formalized in a real-time temporal logic borrowed from [Koy92]. The following operators will be used for temporal referencing: $\circ$/$\bullet$ (in the next/previous state), $\Diamond$/$\blacklozenge$ (some time in the future/past), $\square$/$\blacksquare$ (always in the future/past). Real-time restrictions are indicated by subscripts (e.g., $_{\leq Cd}$ means within C days). Actions are formalized by first-order pre-/postconditions (in the state-based tradition [Pot91]) and trigger conditions.

Figure 1 illustrates the KAOS declaration level for a fragment of a conference organization system. (This fragment will be used as source case later in the paper.)

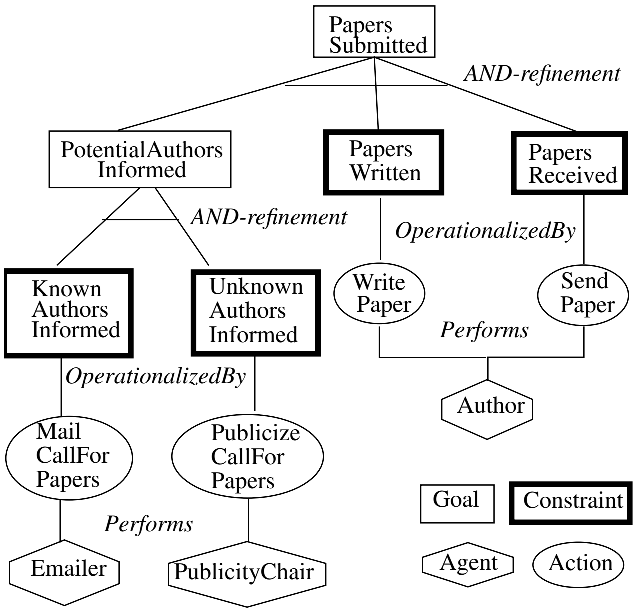

Fig. 1 - Conference organization

The goal PapersSubmitted requires collaboration between authors and conference organizers to be achieved. It may be refined into goals such as PotentialAuthorsInformed, PapersWritten and PapersReceived; the latter is a constraint as it can be can be under responsibility of a single type of agent, namely, Author; authors can perform the SendPaper action under restricted time conditions to ensure that the constraint PapersReceived is met. The specification below gives an alternative textual declaration of the goal PapersSubmitted together with its inner formal assertion. The latter says that if an author is a PotentialAuthor for a CallForPaper, then a paper will be submitted to the ProgramChair by her some time in the future before the Deadline in the call for papers. (Clearly this goal is too ideal and should be weakened.)

Goal Achieve [PapersSubmitted]
InstanceOf SatisfactionGoal
Concerns Author, CallForPaper, Paper, ...
RefinedTo PotentialAuthorsInformed, PapersWritten, ...
FormalDef $\forall$ a: Author, c: CallForPaper, pc: ProgramChair
PotentialAuthor (a,c) $\wedge$ Chairing (pc,c)
$\Rightarrow \Diamond_{\text{c.Deadline}}$ ($\exists$p: Paper) Submission (a,p,pc)

In the formal assertion above, the predicate PotentialAuthor(a,c) means that an instance of the PotentialAuthor relationship links, in the current state, variables a and c of sort Author and CallForPaper, respectively. The PotentialAuthor relationship type, Author agent type and CallForPaper entity type are declared in other sections of the specification, e.g.,

Agent Author
CapableOf WritePaper, ...
Has Expertise: **SetOf** [*SubjectAreaI*]
...

Relationship PotentialAuthor
Links Author {**card**: 1:N}, CallForPaper {**card**: 1:N}
DomInvar $\forall$ a:Author, c:CallForPaper
PotentialAuthor (a,c) $\Leftrightarrow$ a.Expertise $\cap$ c.Scope $\emptyset$

In the declarations above, Expertise is declared as an attribute of Author and used in the invariant defining PotentialAuthor. Object inheritance is also supported, e.g., an agent type PC-Author might be introduced as a specialization of Author with some extra attributes and invariants.

As mentioned earlier, operations are specified formally by pre- and postconditions in the state-based tradition, for example,

Action WritePaper
Input SubjectArea {**Arg**: s}; **Output** Paper {**Res**: p}
PerformedBy Author {**Arg**: a}
DomPre $\neg$ WrittenBy (p,a)
DomPost WrittenBy (p,a) $\wedge$ p.SubjectArea = s

2.2 Meta-level knowledge

Domain-independent knowledge is used for local guidance and validation during goal-driven elaboration. Such knowledge will be used in this paper to constrain analogical mappings.

In particular, a rich *taxonomy* of goals, constraints, objects and actions is defined at the meta level together with rules for specifying concepts of the corresponding sub-type. Here are a few examples.

– Goals are classified by pattern of temporal behaviour they require:
Achieve: $P \Rightarrow \Diamond Q$ or *Cease:* $P \Rightarrow \Diamond \neg Q$
Maintain: $P \Rightarrow \Box Q$ or *Avoid:* $P \Rightarrow \Box \neg Q$
– Goals are also classified by type of requirements they will drive with respect to the agents concerned (e.g., *SatisfactionGoal*, *InformationGoal*, *ConsistencyGoal*, *SafetyGoal*, *PrivacyGoal*, etc.)
– Constraints are in the *HardConstraint* category if they may never be violated, or in the *SoftConstraint* category if they are likely to be temporarily violated.
– Actions are *Modify* or *Inspect* actions according as they modify some object state or not.

Such taxonomies are constrained by rules, e.g.,
– SafetyGoals are AvoidGoals to be refined in HardConstraints;
– PrivacyGoals are AvoidGoals on *Knows* predicates;
– SoftConstraints must have associated ModifyActions to restore them.

3. Overview of the approach

Reusing specifications requires a library of reusable components to be available. Such a library in our context consists of a structured collection of *cases* that have been specified in the past. As seen before, each case is a domain-specific aggregation of interrelated goals, constraints, agents, entities, relationships, events, actions, and scenarios. The collection is structured by *IsA* specialization links that may relate concepts of the same type from different cases; multiple inheritance between different cases is thus possible. Such a library will hereafter be called *case base*.

The analogical reuse problem we consider in this paper is formulated as follows.

GIVEN - a partial specification already elaborated, called *target* specification,
- the identification of some missing part,

FIND - a set of relevant *source* specifications from the case base, *translated* into the vocabulary of the target specification, as candidates for filling in the missing part.

The retrieved source specifications are called solutions to the reuse problem. The analyst then has to select among solutions and evaluate how to adapt and integrate the preferred alternative.

3.1 Example

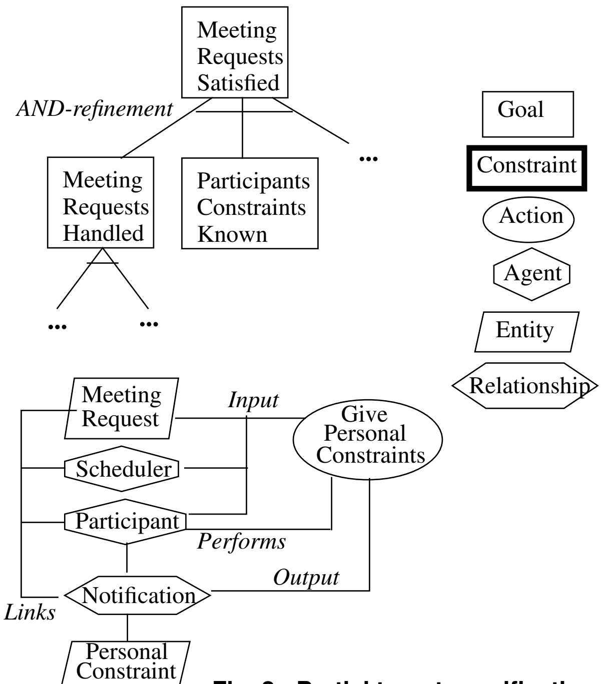

Fig. 2 - Partial target specification

Figure 2 shows two unrelated fragments already elaborated for a target specification of a meeting scheduler. The goal MeetingRequestsSatisfied has been refined into subgoals such as MeetingRequestsHandled, ParticipantsConstraintsKnown, etc. The latter has not been refined yet. On another hand, the action GivePersonalConstraints has been identified and partially defined through an independent process --e.g., from usage scenarios. This action allows a participant to provide his personal constraints, in the time frame prescribed by the request, to the scheduler.

Figure 3 shows one solution obtained by analogical reuse. Graphical symbols in gray highlight the concepts and links which have been found. The formal definitions of those concepts are transferred as well --e.g., there is a strengthened postcondition on the GivePersonalConstraints action to ensure the ParticipantsConstraintsProvided con-

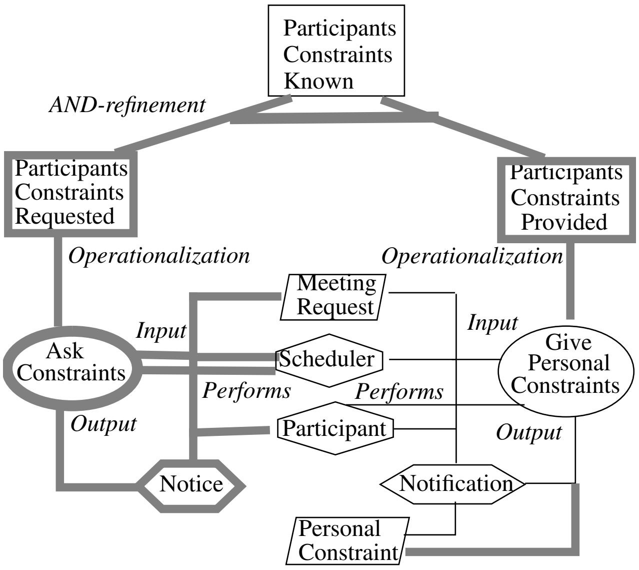

Fig. 3 - Target specification completed

straint, namely, that the participant's constraints delivered fit in the time frame prescribed by the meeting request.

3.2 Constraints on the analogical mapping

Analogies are often seen as mappings from source to target cases such that relations holding in the source also hold in the target [Gen88, Fal89]. The corresponding sources and targets do not necessarily have to be "similar"; they are placed into correspondence according to the *role* they play in the relational structure. The intuition is that analogies are more about relations than about individual features.

This view is customized and extended to the declaration and assertion levels of our KAOS framework. The following constraints are required to hold on any analogical mapping between specifications.

- *One-to-one concept mapping:* a source (target) concept may only be mapped to a single target (source) concept.
- *Same meta-level type:* analogs must be of the same metatype --e.g., if the source concept is an agent then the target must be an agent as well; if the source concept is a constraint, then the corresponding target concept must be a constraint as well.
- *Same relationship/predicate arity:* source and target relationships/predicates may be mapped one onto the other only if they have the same number of linked objects/arguments.
- *Role preservation:* source and target relationships/predicates may be mapped one onto the other only if their arguments can be mapped pairwise in their order of appearance. Consider, for example, the predicates *Notification*(Participant,PersonalConstraint,MeetingRequest) and *Submission*(Author,Paper,CallForPapers). The Notification relationship may be mapped to Submission only if Participant can be mapped to Author, PersonalCon-

straint to Paper, and MeetingRequest to CallForPapers, respectively.

- *Same meta-level category:* source and target concepts may be mapped one onto the other only if they are in the same category. For example, a SatisfactionGoal may not be mapped to an InformationGoal; a HardConstraintl may not be mapped to a SoftConstraint. When formal definitions are compared, they must be in the same category as well; this constraint prevents an Information assertion from being compared to a Privacy assertion even though they may be structurally similar.
- *Same assertion pattern:* source and target concepts may be mapped one onto the other only if they have the same assertion pattern. For example, an Achieve constraint may not be mapped to a Maintain constraint. When formal definitions are compared, they must have the same pattern as well; concepts involved in a source (target) assertion may only be mapped to concepts in a target (source) assertion if the assertions have the same temporal pattern.

The above mapping constraints must be satisfied both at declaration and assertion levels. They drive the entire analogical reuse process. It should be clear at this point that *ontological richness of the language directly impacts on the accuracy of the analogy.*

3.3 Steps of the analogical reuse process

Analogies are built incrementally. First a *structural* match between declarations is sought; the match is then refined through *semantic* matching on assertions; the analogy is finally extended so that KAOS-specific properties may be verified, thus increasing confidence in it. The following steps are distinguished.

1. *Specification of the reuse problem:* a query is formulated to identify missing parts in the target specification for which candidate specification fragments should be found by analogy.
2. *Classification of target concepts:* relevant target concepts in the query are classified with respect to concepts in the case base; the relevant target concepts are the ones being related to the missing ones. Conceptual classification is needed for relating the current situation to past cases.
3. *Query reformulation and satisfaction*: the query is generalized using the classification found; concept taxonomies in the case base are then traversed to retrieve candidate source analogs that satisfy the generalized query. Partial mappings are derived on the basis of *structural* correspondences between target and source declarations.
4. *Refinement of analogical mappings:* the declaration-level mappings are refined on the basis of *semantic* correspondences between source and target assertions. Refined mappings allow relevant assertions to be transferred from source to target.
5. *Source case selection:* this step is necessary when multiple candidate source cases are found. Source cases are compared on the basis of a similarity metrics that measures the number of similar relevant assertions with respect to the total number of relevant assertions.
6. *Verification-driven analogy extension:* a commonly accepted heuristic in analogical reasoning is that things agreeing in one respect will probably agree in yet another. This step aims to extend the original analogy so that more than the initial reuse problem is solved. Extension rules based on KAOS-specific features are used for that purpose. For example, if a partial target AND-refinement is involved in the analogy, an attempt is made to complete the AND-refinement so that its correctness can be verified [Dar96].
7. *Transfer and integration:* analogical declarations and assertions are transferred after translation according to the refined mapping, and integrated into the target specification so that the analyst may assess the result and adapt it.
8. *Consolidation:* effective reuse requires a broad and up to date case base. This interactive step identifies potentially reusable target concepts that are worth being included in the case base; it uses a measure of newness and distance with respect to closest generalizations in the case base.

In the sequel we will focus on steps 1-4, and briefly discuss the other steps in order to provide a complete picture of our approach.

4. Specifying the reuse problem

The analyst must indicate some missing parts in her specification which she would like to fill in by analogical reuse. This is done by submitting a query in the vocabulary of the specification being elaborated. The answer to the query will be a set of candidate source specifications reformulated in that vocabulary.

The query defines the *scope* of the analogy; all concepts mentioned in it are considered *relevant.* The whole process of analyzing partial specifications to find out missing parts is thereby shortcut. Moreover, retrieving a source case that would match a big partial specification is highly unlikely in real situations. A precise, restricted scope that points out concepts relevant to missing parts has much more chance to result in source analogs being retrieved at lower cost.

Consider, for example, the analogical reuse problem suggested in Fig. 2. This problem may be more precisely specified by the query

set result = **Constraint**
which Refines ParticipantsConstraintsKnown
and which OperationalizedBy GivePersonalConstraints

We use the generic entity-relationship query language described in [Lam87], customized to the KAOS meta-model. Names of meta-level types, links and attributes become language keywords. The language has a set-theoretic framework. The effect of a **set**-statement is to define a *group* of one or more elements (denoted here by variable result) from types or other groups by means of classical set operations and **and**/**or**/**not** logical compositions of predicates on meta-level links (**which**-clause) and attributes (**with**-clause).

Little work has been done on specifying analogical reuse problems that must be solved within larger contexts. Most of work has been devoted to single-concept reuse. Queries seem a natural way to focus the analogical scope within a larger framework. Queries were used in [Lau94] to specify user-defined similarity criteria for locating source analogs; this provides much more flexibility over arbitrary syntactic similarity measures that have been proposed in the past, based usually on weighed counts of similar attributes. Queries have also been used to locate library components by providing desired component features and comparing them to library component features [Dev90], [Fea93], [Mil94]. Signature information can be seen as one such feature [Zar93].

5. Query generalization and satisfaction

Exhaustive search through the case base with structural and semantic matching is clearly unfeasible for a sufficiently rich cases base. Our approach takes advantage of the structure of the case base and the two-level structure of the KAOS language. The basic idea is to first retrieve "promising" source cases through declaration-level *structural* matching, and then determine whether the mapping may be refined through assertion-level *semantic* matching. Sources are retrieved by (i) *reformulating* the query into more abstract terms using conceptual taxonomies in the case base, (ii) *satisfying* the more abstract query using the case base query system, and (iii) deriving a partial mapping for each answer to the query, on the basis of the *structural* match between the query and the answer. Once candidate sources have been found and a partial mapping established, the formal assertions are analyzed pairwise for similarity. This more expensive step is performed only for refining the partial mapping.

5.1 Classification of target concepts

Query satisfaction requires that the concepts involved in the query be related to concepts in the case base. This is done by generalizing the query so as to reach a vocabulary of concepts found in the case base.

We define the *conceptual neighbourhood* of a concept *C* in a specification *S* as the set of concepts related to *C* in *S* through meta-level links. For example, the conceptual neighboorhood of the action GivePersonalConstraints in Fig. 2 is the set {MeetingRequest, Scheduler, Participant, Notification}.

The *scope* of a query is then defined as the set of domain-level concepts appearing in the query together with their conceptual neighbourhood in the target specification.

The preliminary substep for query generalization consists of *classifying* the various concepts in the scope of the query with respect to the contents of the case base. The classification establishes temporary *IsA* specialization links between target concepts and case base concepts to allow comparisons to be made --e.g., to determine whether two concepts have a common ancestor or common features. (Temporary *IsA* links may be frozen later during the consolidation step if the case base is enriched with the target.)

Figure 4 gives one possible result of the classification substep for the sample query above. Note that parent concepts in the classification may come from different cases; the entity MeetingRequest is a specialization of an abstract entity Request from the ReservableResourceManagement case; the action GivePersonalConstraints is a specialization of the Send action from the MessageDelivery case; and the goal ParticipantsConstraintsKnown is a specialization of the abstract goal ConstrainedAnswersGathered from the QuestionAnswering case. The Notification relationship specializes the Limitation relationship (linking Supplier, Request and Requestor not shown there) by specializing the linked entity Request to MeetingRequest. (Lack of

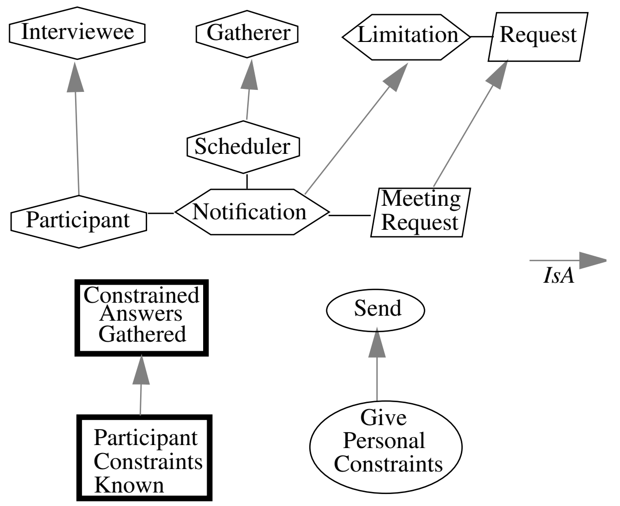

Fig. 4 - Conceptual classification in the case base

space prevents us from describing these cases here; a detailed specification of the ReservableResourceManagement case can be found in [Dar95].)

The classification of target concepts relies on standard classification technology [Dev90], [Bor94]; the classification is further constrained by the meta-level type, category and role preservation constraints introduced in Section 3.2. For example, a human agent must be classified in a human agent taxonomy.

Graphical browsing tools may sometimes be used as a substitute to automatic classification. In particular, if classifications are made to parents from a same case, it may be worth browsing through the whole parent case. Suppose, for example, that the goal ParticipantsConstraintsKnown *and* the action GivePersonalConstraints have been classified under ConstrainedAnswersGathered and ExpressConstrainedAnswer, respectively. One would then expect some constraint from the QuestionAnswering case to refine ConstrainedAnswersGathered and to be operationalized by ExpressConstrainedAnswer; such a constraint might be specialized to fill in the missing part directly. Use of the wordnet software [Mil88] was very helpful in finding some of the generalizations in Fig. 4.

5.2 Query generalization

The vocabulary of the initial query must be translated into a sufficiently abstract version to allow source cases to be retrieved by query satisfaction.

A query *Q2* is said to be a *generalization* of query *Q1* in a classified case base *C* if *Q2* is the result of substituting one or more concepts in the scope of *Q1* by some corresponding more general concept in *C*.

For the classification given in Fig. 4, the following query is a generalization of the sample query above:

set result = **Constraint**
which Refines ConstrainedAnswersGathered
and which OperationalizedBy Send

The search for query generalizations is integrated in the query satisfaction algorithm, see below.

5.3 Structural matching

Elements in the target and source cases are considered structurally similar if there is a correspondence at declaration-level between the source, target, and generalized cases. This correspondence is more precisely defined as follows.

Let *S* and *T* be fragments of a source and target case, respectively; *S* and *T* are said to *match structurally* if there exists a generalized case fragment *G* such that one can decompose *S*, *T*, and *G* into corresponding pairs of elements that satisfy the following conditions: (i) the corresponding elements in the corresponding pairs of *S*, *T,* and *G* are of the same meta-level type and category; (ii) the elements in the corresponding pairs of *S*, *T,* and *G* are linked together by links of the same meta-level type; (iii) the corresponding elements in the corresponding pairs of *S* and *T* are both specializations of the corresponding element in the corresponding pair of *G*.

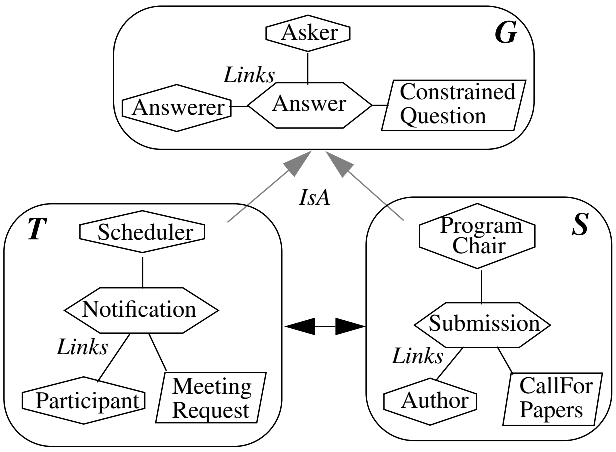

Fig. 5 - Structural matching

Figure 5 shows a structural matching between fragments of the MeetingScheduler and ConferenceOrganization cases. The pair (Participant, Notification) in the MeetingScheduler fragment corresponds to the pairs (Author, Submission) in the ConferenceOrganization fragment and (Answerer, Answer) in the common generalization. It is easy to see that conditions (i)-(iii) hold, e.g., (i) Participant, Author, Answerer are all of type Agent whereas Notification, Submission, Answer are of type Relationship; (ii) Participant and Notification, Author and Submission, Answerer and Answer, are pairwise connected by links of the same meta-level type --namely, Links (other types would be Performs, OperationalizedBy, etc.); (iii) Participant and Author are both specializations of Answerer; Notification and Submission are both specializations of Answer; etc.

A *structural mapping* between *T* and *S* is directly derived from a structural match of *T* and *S*; it is defined by the set of corresponding elements in the corresponding pairs. For Fig. 5 this mapping is:

{ (Notification, Submission), (Participant, Author),
(Scheduler, ProgramChair), (MeetingRequest, CallForPapers) }

The first four mapping constraints in Section 3.2 are clearly satisfied by structural mappings. Such mappings are partial because the formal assertions in *S* and *T* are not taken into account.

5.4 Query satisfaction and structural mapping

The objective of the query satisfaction algorithm is to retrieve source case fragments that structurally match the target case fragment defined by the scope of the query submitted, and to derive structural mappings therefrom. Each concept in the scope of the query is assumed to be classified in the case base (see Section 5.1).

The general principle is to search for query generalizations and source fragments satisfying them in parallel. Generalized scopes are found by climbing *IsA* taxonomies in the case base, starting from the parent concepts determined by the classification. Alternative ways of climbing taxonomies are explored; alternative generalizations are encountered when a concept has multiple parents. In order to explore more promising source fragments first, the search is guided by the following two heuristics:

- *Specificity*: favor scope generalizations that involve more specialized concepts rather than more general ones;
- *Coherence*: favor scope generalizations that involve concepts from a same case rather than from multiple ones.

The specificity heuristics is based on the intuition that two concepts that have a common generalization lower in the taxonomy have more common features by inheritance; the corresponding source is therefore more likely to be relevant. The coherence heuristics is based on the intuition that if the generalized query is specific to a case, then the answers to the query will be specific to specialized cases of it.

The specificity heuristics is implemented by a *breadth-first* traversal of taxonomies; more concrete generalizations are thereby explored before more abstract ones. The coherence heuristics is taken into account through a coherence metrics defined by *s / c*, where *s* denotes the number of elements in the scope of the query and *c* denotes the number of concepts in the smallest case covering the entire scope of the query.

The algorithm dynamically expands a so-called *reformulation graph*. Nodes in this graph represent target-parent concept substitution pairs to be used for generalizing the query; arrows represent ways of moving to more general target-parent concept pairs, by replacing parent concepts by their own parents in the case base taxonomy. *Expanding* a node means computing its successors by making such replacements. A node is said *open* if it is expandable but has not been expanded yet. The set of open nodes in a reformulation graph under expansion is denoted by OPEN.

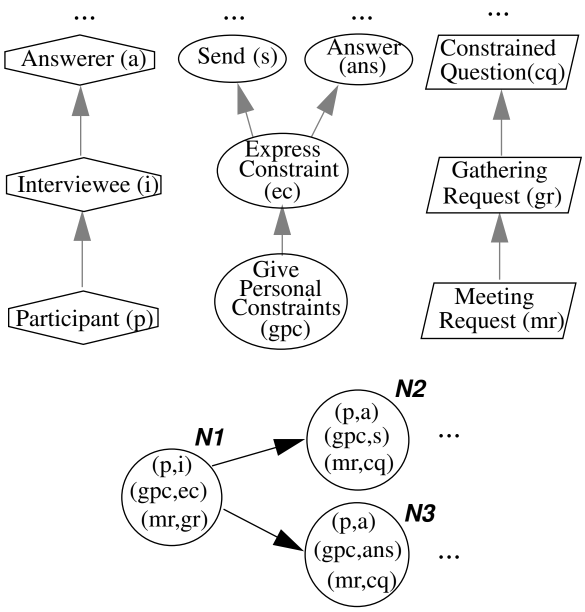

Fig. 6 - Reformulation graph

Fig. 6 shows a reformulation graph for the taxonomies appearing on the upper side. The action GivePersonalConstraints (gpc) has one parent, namely, ExpressConstraint (ec); the initial node in the reformulation graph therefore contains the pair (gpc,ec). Since ec has two parents, namely, Send (s) and Answer (ans) from the MessageDelivery and QuestionAnswering cases, respectively, there are two alternative successor nodes which contain the pairs (gpc,s) and (gpc,ans), respectively. According to the specificity heuristics, node N3 will be expanded before node N2 because the latter contains more abstract substitutions. According to the coherence heuristics, the generalized query

set result = **Constraint**
which Refines ConstrainedAnswersGathered
and which OperationalizedBy Answer

will be preferred over the generalized query

set result = **Constraint**
which Refines ConstrainedAnswersGathered
and which OperationalizedBy Send

that appeared in Section 5.2; the concepts in the former all belong to the QuestionAnswering case whereas in the latter the QuestionAnswering and MessageDelivery cases are involved.

The query satisfaction process can now be described by the following heuristic algorithm.

1. Create root of reformulation graph from classification of target concepts in case base;
2. **while** OPEN is not empty **and** maximum search time not exceeded **do**
   2.1. select a most specific and coherent node in OPEN, say *MSC*;
   2.2. - reformulate query according to substitutions in *MSC*;
   - submit generalized query to case base query system;
   - extend set of answers and derive structural mappings;
   2.3. expand *MSC* and update OPEN accordingly
3. Return set of answers and structural mappings.

The algorithm may in general return a set of alternative answers. In our ongoing example, one possible answer is the constraint PapersReceived from the ConferenceOrganization case introduced in Fig. 1, together with its conceptual neighborhood. Additional relevant source concepts are obtained iteratively through new queries involving the answer returned, e.g.,

set result = Action {**which Isa** Answer
and operationalizes *PapersReceived* **and** ...}

which in turn yields the SendPaper action.

The coresponding structural mapping obtained is:

{ (ConstrainedAnswersGathered ← [ParticipantConstraintsKnown, PapersSubmitted]),
(Answer ← [GivePersonalConstraints, SendPaper]),
(ConstrainedQuestion ← [MeetingRequest, CallForPapers]),
(Answerer ← [Participant, Author]),
(Asker ← [Scheduler, ProgramChair]),
(Answer ← [Notification, Submission]) }

where the form (CommonGeneralization ← [TargetConcept, SourceConcept]) is used to highlight the common parent.

6. From structural to semantic mappings

The structural mapping resulting from query satisfaction is only partial because the formal assertions attached to the answers are not taken into account; such assertions should be eventually transferred (at least partially) to the target after suitable translation of the concepts involved in them. Moreover source assertions may refer to concepts which were not covered by the structural mapping; finding target analogs of such concepts may result in the identification of missing elements in the target specification.

Consider the constraint PapersReceived that was retrieved as an answer to the analogical query above.

SoftConstraint Achieve [PapersReceived]
InstanceOf SatisfactionConstraint;
Concerns Author, CallForPapers, Paper, ...
Refines PaperSubmitted;
OperationalizedBy SendPaper
FormalDef
∀ a: Author, c: CallForPapers, p: Paper, pc: ProgramChair
PotentialAuthor (a,c) ∧ Chairing (pc,c) ∧ WrittenBy (p,a)
⇒ $\Diamond_{\text{c.Deadline}}$ Submission (a,p,pc)

Reformulating the above formal definition into the target vocabulary is not possible from the partial mapping obtained so far; the relationships WrittenBy, Chairing and the entity PersonalConstraint are not covered by the structural mapping above.

Our next step is to complete the structural mapping so that assertions that need to be transferred to the target can be translated. The general principle is to select "relevant" source assertions that involve unmapped concepts, translate them into target assertions with the unmapped concepts being replaced by variables, and then attempt to match them with existing target assertions. A successful matching instantiates variables to target concepts, that become analogs of the source ones.

6.1 Relevance rules

The rules for deciding which assertions are relevant for refining the structural mapping are specific to the KAOS meta-model; they depend on the meta-level types, links and attributes of the concepts appearing in the query. Fig. 7 gives a small portion of the KAOS meta-model (see [Dar93] for more details). We give a sample of relevance rules associated with this portion.

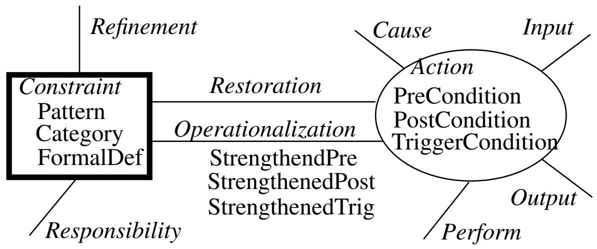

Fig. 7 - Portion of the KAOS meta-model

- If the query is about a target Action and the Input and Output meta-level links are mentioned in it, then the Precondition and Postcondition of the source Action are relevant.
- If the query is about a target Action and the Input meta-level link is mentioned in it, then the corresponding Output link in the source Action and its arguments along with the Precondition and Postcondition are relevant.
- If the query is about a target Action and the Operationalization meta-level link is mentioned in it, then the StrenghtenedPrecondition and StrenghtenedPostcondition for Operationalization of the source Action are relevant.

Consider, for example, the action GivePersonalConstraints. Suppose that we only specified its signature so far and want to define it further through analogical reuse. For the query

set result = **Action**
which Inputs MeetingRequest, Participant, PersonalConstraint
and which Outputs Notification ,

the only assertions considered relevant for refinement will be the pre- and postconditions of the analog source action. If the above query is extended with "... **and which** Operationalizes ParticipantsConstraintsGathered", the rules above will extend the transfer of the pre/postconditions to the strengthened pre- and postconditions to ensure ParticipantsConstraintsGathered. The principle is to focus the refinement on the parts of the source case that seem relevant to the analogy and ignore all the others for the moment (they will be taken into account later in the extension step).

6.2 Semantic matching

Candidate target analogs of unmapped concepts in source assertions are obtained by matching target assertions with relevant source assertions in which the unmapped concepts appear.

Let *SA*, *TA* and *M* denote a source assertion, target assertion, and structural mapping of source/target case fragments to which SA/TA are attached, respectively; let *GTrans*(*SA*,*M*) denote a generalized translation of *SA* according to *M,* in which unmapped source concepts have been replaced by meta-variables of the corresponding meta-level type.

SA and *TA* are said to *match semantically* if there exists a substitution of the source variables in *GTrans*(*SA*,*M*) by target concepts that meets the mapping constraints and makes *TA* and *GTrans* logically equivalent. Such a substitution, where source variables are reinstantiated to the corresponding source concepts, is called *semantic mapping extension*. The mapping constraints are those introduced in Section 3.2; logical equivalence may be established using the proof theory of temporal logic [Man92].

To illustrate this notion of semantic matching, consider the following two assertions. The first one captures the fact that, if a call for papers refers to a conference and an author's paper is accepted, then the author will attend the conference; the second one captures the fact that, if a meeting request results in a meeting being planned and a participant's personal constraints have been addressed, then the participant will attend the meeting:

$\forall$a: Author, c: CallForPapers, co: Conference, p: Paper
RefersTo (c,co) $\wedge$ *Accepted (*p, co*)*
$\Rightarrow \Diamond_{co.begin}$Attends (a, co)] [*MustAttendConference*]

$\forall$ p: Participant, mr: MeetReq, m:Meeting, *pc: PersonalConstraint*
PlannedInto (mr,m) $\wedge$ Addressed (*pc*, m)
$\Rightarrow \Diamond_{m.begin}$Participates (p, m) [*MustAttendMeeting*]

Suppose that all concepts in these two assertions are structurally mapped except the Accepted relationship and the PersonalConstraint entity. Assuming the source assertion to be MustAttendConference, the generalized translation *GTrans*(*SA*,*M*) yields the following assertion:

$\forall$ p: Participant, mr: MeetReq, m: Meeting, x : <?X: Entity>
PlannedInto (mr,m) $\wedge$ <?Y: Relationship> (x, m)
$\Rightarrow \Diamond_{m.begin}$Participates (p, m)

The substitution {(<?X:Entity>, PersonalConstraint) , (<?Y:Relationship>, Addressed)} makes *TA* and *GTrans* logically equivalent. This substitution meets the various mapping constraints, viz. one-to-one concept mapping, same meta-level type and category, same relationship/predicate arity, role preservation, same assertion pattern (Achieve), and same assertion category (Satisfaction). The semantic matching thus extends the analogical mapping with the pairs (PersonalConstraint, Paper) and (Addressed, Accepted).

Attempting to semantically match a source and target assertion makes sense only if the assertions are comparable, that is, if they have a same pattern and same category.

Compound assertions may therefore need to be decomposed into basic assertions so that semantic matching may be achieved. Basic assertions are assertions that have a pattern and a category; compound assertions are ones that are made up of basic and compound assertions using logical connectors such as $\wedge$ and $\vee$.

6.3 Completing the mapping

The principle is to extend the structural mapping until all relevant source assertions that need to be transferred are translated into the target vocabulary. A mapping *M* is said to be a *completed mapping* if (1) it extends a structural mapping, (2) all concepts from relevant source assertions attached to the answer to the query are in the range of *M*, (3) their analogs are defined in the target case, and (4) *M* meets the pattern and category mapping constraints.

In our ongoing example, the mapping is completed when all concepts from the formal definition of the constraint PapersReceived are in the range of the mapping.

A mapping completion algorithm based on the above definition is sketched below.

1. Apply relevance rules to select relevant assertions *A*
2. Identify in *A* concepts unmapped by structural mapping
3. **while** concepts still unmapped **do**
   3.1. select one, say *C*
   3.2. identify relevant source assertions *SA* about *C*
   3.3 **while** *C* is unmapped **and** there are still *SA* about *C* **do**
      - select one *SA*
      - **if** semantic match with relevant target assertion
        then add analogs to mapping
4. Return completed mapping or failure

For the PapersReceived answer in our ongoing example, step 1 applies the rule "if the query is about a Constraint and the Operationalization meta-level link is mentioned in it, then the formal definition of the constraint is relevant"; therefore the formal definition of PapersReceived needs to be transferred. Step 2 identifies the relationships PotentialAuthor, WrittenBy, Chairing and the entity Paper as unmapped by the structural mapping (see the mapping at the end of Section 5.4 and the formal definition at the beginning of Section 6). Step 3.1 selects an unmapped concept such as Paper. Step 3.2 identifies relevant assertions about Paper, such as the invariant attached to Submission, and decomposes it into basic assertions like MustAttendConference. An assertion is then selected (e.g. MustAttendConference) and a semantic match with relevant target assertions is attempted. As shown earlier MustAttendConference can be matched with the target assertion MustAttendMeeting; as a result the pairs (PersonalConstraint, Paper) and (Addressed, Accepted) are added to the mapping. Failure to return a completed mapping indicates that no analogy can be found due to dead end; in this case the source case should be dropped and another one should be taken from the answer set. A completed mapping allows the formal definition of the source constraint PapersReceived to be translated to a candidate formal definition for the missing target constraint (the name PersonalConstraintsGathered is supplied by the analyst), like the following.

HardConstraint Achieve [PersonalConstraintsGathered]
InstanceOf SatisfactionConstraint;
Concerns Participant, PersonalConstraint, ...
OperationalizedBy GivePersonalConstraints
FormalDef
$\forall$ p:Participant, mr:MeetReq, pc:PersonalConstraint, s:Scheduler
Invited (p, mr) $\wedge$ Scheduling (s, mr) $\wedge$ ConstraintProduced (pc, p)
$\Rightarrow \Diamond_{\text{mrDeadline}}$ Notification (p, pc, s, mr)

Compared with other methods for declaration-level reuse through structural matching, analogical transfer through semantic matching results in a much richer, meaningful requirements fragment. The transposed requirements have of course to be checked against adequacy by the analyst and adapted if necessary.

7. Discussion

Analogical reasoning techniques are aimed to mimic the frequent situations where requirements engineers rely on past experience to elaborate and validate new requirements. With respect to existing work on analogical reasoning techniques [Gen88], [Fal89], [Hal89], [Lam91] and their application to specification reuse [Mai91], [Sut93], the focus of the paper was on showing how a *rich ontology* for capturing requirements combined with *formal assertions* to express them may yield more meaningful matching for specification reuse. Richer meta-level taxonomies result in more accurate structural matching (thanks to the constraints on same meta-level type, category and assertion pattern imposed in addition to the one-to-one and relational consistency constraints from [Gen88], [Fal89], see Section 3.2); formal assertions provide additional semantic accuracy. Analogical reuse is thereby extended from semi-formal to formal specifications. Such reuse may contribute to decreasing the cost of applying formal methods in requirements engineering while retaining their benefits.

Missing parts in a partial specification are identified by queries. The scope of the query may involve a wide variety of concepts whose type is defined at the meta-level: goals, constraints, agents, entities, relationships, events, actions, or scenarios. The target concepts in this scope are first classified with respect to concepts in the case base. Generalized queries and specialized answers are then found by climbing case base taxonomies up and down, respectively, under structural and semantic mapping constraints. The answers produce source cases and structural mappings; the latter are refined through semantic matching of relevant formal assertions. Analogical transfer is achieved by applying completed mappings to the source declarations and their relevant assertions. The process can be iterated step by step so that entire requirements frameworks can be reused.

The search-intensive phases of retrieving source cases and elaborating structural analogies do not require the analyst's intervention. Heuristic search for structural matches is aimed at guaranteeing reasonable performance. Semantic matching may raise more serious problems as a powerful theorem prover would be needed to fully automate the process. Semantic matching is currently performed manually; our experience suggests that manual assertion matching is already quite helpful for elaborating and/or validating requirements specifications. Automating semantic matching at reasonable cost would probably require some compromise between formalism expressiveness and efficient assertion matching (as already shown by classification technology). Our principle of introducing specialized assertion patterns and categories that constrain the mapping may be seen as a preliminary step in that direction.

Two important aspects of the analogical reuse process were left aside from this paper: the validation of transferred requirements and the consolidation of the case base.

Once candidate source specifications have been transposed, the analyst has to assess whether the resulting specifications are appropriate to the target situation and, if so, adapt them if necessary.

Besides, an extra benefit of integrating a formal assertion layer is that formal verification technology may be used to increase the level of confidence in reused specifications. We are currently investigating techniques for verification-based analogy extension. For example, when a partial goal refinement has been transferred, the analogy may sometimes be extended from the source refinement towards a provably complete refinement; the correctness of the transposed specification may be verified formally using techniques described in [Dar96].

The approach to reuse described in this paper relies on a rich, well-structured case base. The increasing popularity of design patterns [Gam95], [Bec96] makes us confident that this assumption is realistic for the future. The incremental update of the case base with new and "interesting" cases remains a big issue though. As a last step of the analogical reuse process, consolidation is aimed at determining whether parts of the target specification obtained should be integrated into the case base due to their potential reusability. This is known as the utility problem in explanation-based learning [Min88]. Our consolidation step is based on three utility criteria. *Generality* estimates the variety of concepts that could be defined as specializations of the concept being considered for inclusion in the case base. *Newness* measures how many new features the concept introduces with respect to its closest parent in the case base. Declaration-level newness is the number of new attributes and links the concept adds; assertion-level newness is based on a count of the number of basic assertions the concept adds with respect to the closest parent. *Conceptual distance* is aimed to identify conceptual holes in case base taxonomies. Defining *Distance(c,c')* as the number of intermediate concepts on the shortest path between concept *c* and his ancestor *c'*, one may use the ratio *Distance(s,cg) / Distance(t,cg),* where *s* is a source concept, *t* the target analog and *cg* the common generalization found during structural match, to identify conceptual holes on the path from *t* to *cg*. Generalized versions of the target concept may then be introduced to fill in such holes. Case base consolidation is thus an interactive process that involves the evaluation of utility criteria, the integration of useful target concepts according to the classification links established during query satisfaction (see Section 5.1), and the generalization of these concepts when utility needs to be improved.

Experience with our analogical reuse approach has been limited so far to small requirements fragments from different domains including various specializations of resource management systems, distributed communication systems, and simple transportation systems. Real, large scale experimentation would require a realistic size case base. The building of such a case base using the KAOS requirements engineering environment under development is among our high-priority tasks. On another hand, recent experience with the use of the KAOS methodology in two industrial projects has confirmed our feeling that requirements elaboration should be more viewed as a reuse-based activity. Beside shortcutting elaborations and supporting validation, specification reuse guarantees better uniformity and comparability.

Acknowledgement. This work was partially supported by the Belgian Ministry of Science (SPPS Project IT/IF/10 "Oasis") and the "Communauté Française de Belgique" (FRISCO project, Actions de Recherche Concertées Nr. 95/00-187 - Direction générale de la Recherche). Thanks are due to the RE'97 reviewers for helpful feedback.

References

[Bec96] K. Beck, R. Crocker, G. Meszaros, J.O. Coplien, L. Dominick, F. Paulisch and J. Vlissides, "Industrial Experience with Design Patterns", *Proc. ICSE-18 - 18th International Conference on Software Engineering*, Berlin, 103-114.

[Bor94] A. Borgida and P.. Patel-Schneider, "A Semantic and Complete Algorithm for Subsumption in the CLASSIC Description Logic", *Journal of Artificial Intelligence Research*, vol. 1, June 1994, 207-308.

[Bra85] R.J. Brachman and H.J. Levesque (eds.), Readings in Knowledge Representation, Morgan Kaufmann, 1985.

[Dar93] A. Dardenne, A. van Lamsweerde and S. Fickas, "Goal-Directed Requirements Acquisition", *Science of Computer Programming*, vol. 20, 1993, 3-50.

[Dar95] R. Darimont, "Process Support for Requirements Elaboration", PhD Thesis, Department of Computer Science, University of Louvain (Belgium), 1995.

[Dar96] R. Darimont and A. van Lamsweerde, "Formal Refinement Patterns for Goal-Driven Requirements Elaboration", *Proc. FSE-4: Fourth ACM Symposium on the Foundations of Software Engineering*, October 1996.

[Dev90] P. Devambu, "LASSIE: A Knowledge-Based Software Information System", *Proc. 12th Intl. Conf. on Software Engineering*, Nice, 1990, 249-261.

[Dia89] R. Prieto Diaz, "Classification of reusable modules", in: *Software Reusability, Volume 2: Applications and Experience*, T.J. Biggerstaff, AJ. Perlis (Eds.), Vol. 2, ACM Press, 1989, 99-123.

[Fal87] B. Falkenhaimer, "An Examination of the third stage in the Analogy Process : Verification-Based Analogical Learning", *Proc. 10th International Joint Conference on Artificial Intelligence*, 1987, 260-263.

[Fal89] B. Falkenhaimer, K. D. Forbus and D. Gentner, "The Structure-Mapping Engine: Algorithm and Examples", *Artificial Intelligence*, vol. 41, 1989, 1-63.

[Fea93] M. Feather, "Requirements Reconnoitering at the Juncture of Domain and Instance“, *Proc. RE'93 - First International IEEE Symposium on Requirements Engineering*, San Diego (Ca), January 1993, 73-77.

[Fin88] A. Finkelstein, “Re-use of formatted requirements specifications”, Software Engineering Journal, September 1988, 186-197.

[Gam95] Gamma, E., Helm, R., Johnson, R., Vlissides, J., *Design Patterns — Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[Gen88] D. Gentner, “Analogical Inference and Analogical Access“, in: *Analogica*, A. Prieditis (Ed.), Pitman, 1988, 63-88.

[Hal89] R.P. Hall, “Computational Approaches to Analogical Reasoning: A Comparative Analysis”, *Artificial Intelligence*, vol. 39, 1989, 39-120.

[Hol89] K.J. Holyoak and P. Thagard, "Analogical Mapping by Constraint Satisfaction", *Cognitive Science,* vol. 13, 1989, 295-355.

[Koy92] R. Koymans, *Specifying message passing and time-critical systems with temporal logic*, LNCS 651, Springer-Verlag, 1992.

[Kru92] C. W. Krueger, “Software Reuse”, *ACM Computing Surveys*, vol. 24, no. 2, June 1992, 131-183.

[Lam87] A. van Lamsweerde, B. Delcourt, E. Delor, M.-C. Schayes and R. Champagne, “Generic Lifecycle Support in the ALMA Environment”, *IEEE Transactions on Software Engineering*, vol. 14, no. 6, June 1988, 720-741.

[Lam91] A. van Lamsweerde, “Learning Machine Learning”, in: *Introducing a Logic Based Approach to Artificial Intelligence*, A. Thayse (Ed.), Vol 3, Wiley, 1991, 263-356.

[Lam95] A. van Lamsweerde, R. Darimont and P., “Goal-Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learnt“, *Proc. RE'95 - 2nd International Symposium on Requirements Engineering*, York (UK), March 1995, 194-203.

[Lau94] D. Lauzon and T. Rose, “Task-Oriented and Similarity-Based Retrieval", *Proc. 9th Knowledge-based Software Engineering Conference*, Monterey, September 1994, pp. 98-107.

[Mai91] N. Maiden and A. Sutcliffe, “Analogical Matching for Specification Reuse”, *Proc. 6th Knowledge-based Software Engineering Conference*, Syracuse (NY), September 1991, 101-112.

[Man92] Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems*, Springer Verlag, 1992.

[Mil88] G. Miller. C. Fellbaum, J. Kegl and K. Miller, “WordNet: An electronic lexical reference system based on theories of lexical memory“, Princeton University Cognitive Science Laboratory Technical Report 11, 1988.

[Mil94] A. Mili, R. Mili and R. Mittermeir, “Storing and Retrieving Software Components: A Refinement Based System“, *Proc. ICSE-16 - 16th International Conference on Software Engineering*, Sorrento (Italy), 91-100.

[Min88] S. Minton, “Quantitative results concerning the utility of Explanation-Based Learning”, Proc. AAAI'88, 564-569.

[Ost92] E. Ostertag, J. Hendler, C. Braun and R. Prieto Diaz, “Computing Similarity in a Reuse Library System: an AI-Based Approach”, *ACM Transactions on Software Engineering and Methodology*, vol. 1, no. 3, July 1992.

[Pot91] B. Potter, J. Sinclair and D. Till, "An Introduction to Formal Specification and Z". Prentice Hall, 1991.

[Reu91] H. Reubenstein and R. Waters , “The Requirements Apprentice: Automated Assistance for Requirements Acquisition”, *IEEE Transactions on Software Engineering*, vol. 17, no. 3, March 1991, 226-240.

[Rya92] K. Ryan, B. Mathews, “Matching conceptual Graphs as an aid to Requirements Reuse”, *Proc. RE'93 - First IEEE International Symposium on Requirements Engineering*, San Diego, January 1993, 112-120.

[Spa93] G. Spanoudakis and P. Constantopoulos, “Similarity for Analogical Software Reuse: A Conceptual Modelling Approach”, *Proc. 5th International Conference on Advanced Information Systems Engineering*, LNCS 685, June 1993, 483-503.

[Sut93] A.G. Sutcliffe and N.A.M. Maiden, “Use of Domain Knowledge for Requirements Validation“, *Proc. IFIP WG8.1 Conference on Information System Development Process*, 1993, 99-115.

[Zar93] A.M. Zaremski and J. M. Wing, “Signature Matching: A Key to Reuse“, *ACM SIGSOFT Software Engineerig Notes*, vol. 18, no. 5, December 1993, 182-190.