Computational Mechanisms For Reuse of Domain Knowledge During Requirements Engineering

N.A.M. Maiden & A.G. Sutcliffe

Centre for Human-Computer Interface Design
City University, Northampton Square,
London, EC1V OHB, UK

Tel: +44-71-477-8412
E-mail: N.A.M.Maiden@city.ac.uk

1 : Introduction

Requirements engineering is complex, error prone and in need of intelligent support for capturing complete, consistent requirement specifications and guiding the requirements engineering process. One solution advocated by the ESPRIT Nature project (Jarke et al. 1993a) is to populate tools with domain abstractions to assist requirement specification (Maiden & Sutcliffe 1992a). Domain abstractions represent the fundamental behaviour, structure and functions of a domain class. This paper proposes that domain abstractions can aid modelling, specification and critiquing of complex system requirements and describes computational mechanisms from an artificial intelligence perspective for that purpose. Therefore, the Nature project is a AI-provider in the context of the workshop.

Reuse of domain abstractions during requirements engineering has been investigated by several authors (e.g. Reubenstein & Waters 1991, Fugini et al. 1991, Maiden & Sutcliffe 1992a). The Requirements Apprentice (Reubenstein & Waters 1991) retrieved clichés to validate the completeness and consistency of requirement specifications. The Advisor for Intelligent Reuse (Maiden & Sutcliffe 1993a) also used domain abstractions, which are a form of complex clichés, as a starting point for specifying new requirements. This paper proposes that retrieving domain abstractions can assist requirement specification and critiquing as well as guide capture of important requirements and domain facts throughout the process. However, complex computational mechanisms from artificial intelligence are needed to retrieve and explain domain abstractions.

Two complex computational mechanisms have been designed and implemented to maximise the leverage from retrieval of domain abstractions for the requirements engineering process. The domain matcher (Maiden & Sutcliffe 1993d) is a hybrid computational mechanism for analogical reasoning between a requirement specification and one or more domain abstractions. It integrates structure matching algorithms and heuristic-based reasoning with domain semantics (Maiden &

Sutcliffe 1993b) to infer consistent fact-pair and object-pair mappings with abstractions. It is similar to existing computational models of analogical reasoning, such as the Structure-Matching Engine (Falkenhainer et al. 1989) and the Analogical Constraint Matching Engine (Holyoak & Thagard 1989), and hybrid case-based reasoners such as Grebe (Branting 1991). The domain matcher is supported by an intelligent dialogue manager (Maiden & Sutcliffe 1994) which explains retrieved domain abstractions to requirements engineers. Explanation strategies are needed to ensure effective comprehension and adaptation of domain abstractions. Strategies for explanation include visualisation of domain structure and behaviour, presentation of prototypical examples, animation of domain behaviour and gradual exposure to domain abstractions to guide comprehension (Maiden & Sutcliffe 1994). The effectiveness of these explanation strategies has been shown during empirical studies of domain abstraction reuse (e.g. Maiden & Tyndale, in press).

The problem classifier (Maiden 1993) uses mappings inferred by the domain matcher to detect incompleteness, overspecification, inconsistencies and noise in the requirement specification. It complements existing approaches such as propositional deduction and dependency maintenance (e.g. Reubenstein & Waters 1991) for detecting inconsistencies in requirement specifications. The problem classifier is integrated with a cooperative tool for explaining complex problem situations to requirements engineers. This notepad lists detected problems in order of importance and duration. The requirements engineer can browse additional explanation and rationale inferred from the match trace by the domain matcher and remedies to these problems also inferred by the domain matcher.

The remainder of this paper presents the computational mechanisms implemented in the domain matcher and problem classifier and their application to Nature's library requirements example (Jarke et al. 1993b). First however, the set of domain abstractions are described in brief to provide background for describing the computational mechanisms.

2: A Theory of Domain Knowledge

A theoretical model of domain knowledge has been proposed to determine the semantic content, boundaries and granularity of software engineering abstractions (Maiden & Sutcliffe 1993c). Reuse of domain abstractions mimics expert software engineering behaviour. Domain abstractions are composed of object and information system models. Object systems define the fundamental structure and behaviour which discriminate domains and underpin the requirement specification. Object system models are discriminated by knowledge types at descending levels of the specialisation hierarchy:

- state transitions of objects with respect to object structures;
- object structures representing simple set-theoretic constructs;
- positive or invariant goal states to be achieved or avoided;
- stative pre/postconditions on state transitions;
- domain or time events which trigger or stop state transitions.

Information systems define functions, report structures and constraints which are instantiated to functional and non-functional requirements (Maiden & Sutcliffe 1993b). Each information system accesses many object systems, therefore information systems represent important but non-discriminating requirements in the specification. The knowledge meta-schema and a set of domain abstractions have been implemented using the knowledge representation language Telos (Mylopoulos et al. 1990) and ConceptBase (Jarke 1992), an object-oriented deductive data base. The domain theory is composed of over 120 object and 20 information system models representing key determinants of business and real-time domains. It is this hierarchy which permits active guidance for specification and critiquing during requirements engineering.

Furthermore, domain abstractions are matched to real-world requirement specifications by aggregated object system models, thus enabling advantages from reuse to be applied to real-world problems. This is achieved using two-stage retrieval of domain abstractions: (i) matching parts of requirement specifications to object system models; (ii) inferring mappings between matched object system models to enable mappings with domain abstractions.

3 : The Domain Matcher

The matcher has four components: a match controller, structure matcher, rule-based matcher and domain discriminator, see Figure 1. These components access the requirement specification and object system models and class discrimination rules defined in the theory of domain knowledge. The result of matching a requirement specification to an object system model is a quantitative measure of similarity and two sets of local mappings describing object-pairs and fact-pairs with object system.

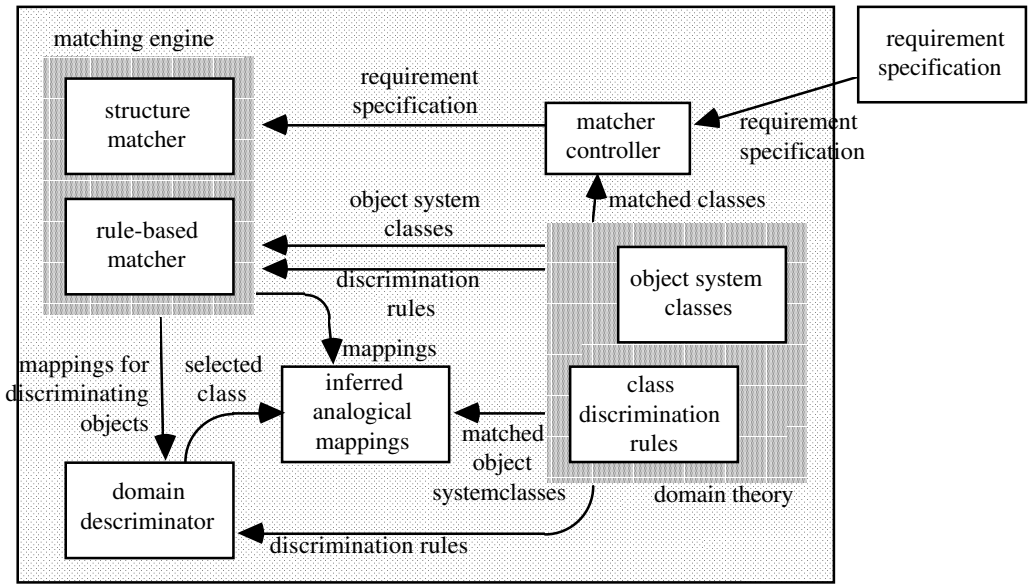

Figure 1 - architecture of the Domain Matcher, showing components needed for retrieval of object system models

The domain matcher is implemented using ConceptBase (Jarke 1992) with Telos (Mylopoulos et al. 1990) and BIM Prolog as part of Nature's coordinated demonstration (Jarke et al. 1993b).

3.1: The Matcher Controller

The matcher controller decides how a requirement specification is mapped to object system models. It is founded on the process model of domain knowledge reuse (Maiden & Sutcliffe 1992b). Its aim is to match a requirement specification to an object system model in the domain specialisation hierarchy. This is achieved by matching the requirement specification to a high-level object system class then specialising this match to specialisations of the object system class until no further specialisation is possible. The controller sends the requirement specification to the structure or rule-based matcher according to the state of the current match in the domain specialisation hierarchy. Structure matching is used to establish an initial match with a high-level object system class which may be specialised to lower-level classes, therefore it is only called once during the matching process. Rule-based matching specialises this initial match by mapping the requirement specification to lower-level object systems which themselves specialise the original object system class. This specialisation continues until no more object system models can be matched or a leaf node of the domain specialisation hierarchy is reached is one of the distinguishing features of the domain matcher. The structure matcher changes algorithms according to knowledge types to be matched. An overview of this process is shown in Figure 2. The integration of hybrid reasoning mechanisms with computational models of analogical reasoning is one of the distinguishing features of the domain matcher.

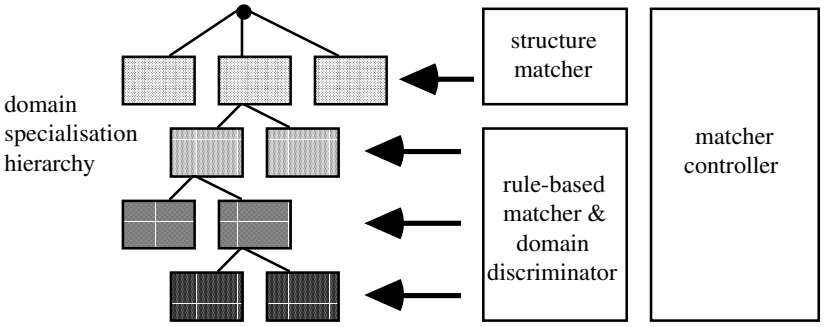

Figure 2 - roles of the structure and rule-based matchers for retrieving object system models

3.2: The Structure Matcher

The structure matcher maps state transitions with respect to object structures, object categorisations and second-order object relations. Structure matching occurs at the beginning of the matching process for three reasons. First, the domain theory claims that these knowledge types are key discriminators between broad object system classes. Second, requirement specifications are ill-defined so guided requirement specification and fact capture using the domain specialisation hierarchy too early in the requirement specification process will be unnatural to the requirements engineer. Therefore, the requirements capturer (Maiden & Sutcliffe 1994) permits the requirements

engineer to enter a sizeable requirement specification which is then matched by the structure matcher maps in a single matching bout. Third, knowledge types mapped by the structure matcher represent interrelated knowledge structures with little low dependencies between instances of knowledge types. Therefore, the matched knowledge types are most amenable to analogical matching. The structure matcher maps the interrelated knowledge structures shown in the informal semantic network in Figure 3.

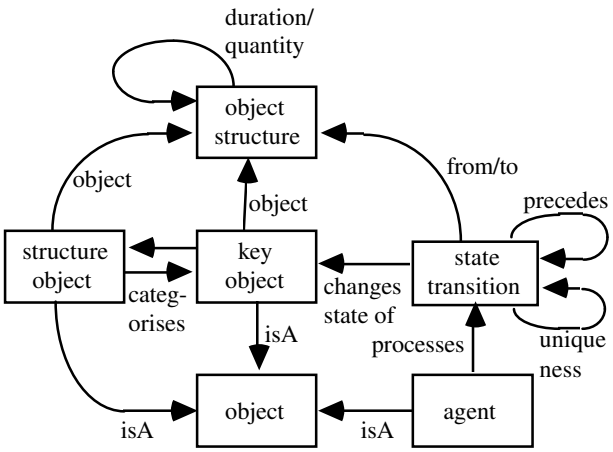

Figure 3 - knowledge meta-schema of types matched by the structure matcher

3.3: The Rule-Based Matcher

Rule-based matching implements the same structure matching algorithm as the structure matcher. However, unlike the structure matcher, it maps a smaller number of facts in the context of an existing structure match. This occurs for two reasons. First an object system model is specialised by and hence matched to small numbers of facts. Full structure matching would involve much redundant mapping to achieve the same goal. Second, the rule-based matcher maps events, actions, stative conditions and goal states representing isolated concepts more amenable to rule-based matching. Matching rules calculate the fit between mappings with specialisations of an object system model and the existing match inferred by the structure matcher. Output from the rule-based matcher is a quantitative score of matching with a object system specialisations and a set of mappings between object-pairs or fact-pairs which define the match.

3.4: The Domain Discriminator

The domain discriminator reasons about key differences between candidate object system models. Discriminator rules are defined as part of the theory of domain knowledge. This reasoning is more typical of rule-based reasoning paradigms in expert systems. The domain discriminator controls matching by the rule-based matcher when specialising a match with a high-level object system model. The discriminator is described elsewhere in Maiden & Sutcliffe (1993d). This paper

examines the structure matching algorithm implemented in the structure and rule-based matchers is examined in more detail.

3.5: The Structure Matching Algorithm

Structure matching borrows algorithms developed in computational mechanisms for analogical reasoning (e.g. Falkenhainer et al. 1989) and matches state transitions, object structures, object categorisations and second-order object relations which form interrelated knowledge structures. Structure matching is in two phases. First, candidate local mappings indicate the likelihood of a match and act as a filter to avoid unnecessary structure matching. Second, structure matching determines a quantitative matching score with each candidate object system model.

Previous analogical matching mechanisms (e.g. Falkenhainer et al. 1989) had excessive runtime and were computationally complex (Holyoak & Thagard 1989), so the structure matcher overcome these problems in two ways. First, definitions of state transitions with respect to object structures in object system classes are kept small, thus reducing runtime. Second, the structure matcher does not attempt to map all possible analogical combinations. Rather it approximates a coherent match between a requirement specification and object system class by identifying a topology of related facts. Each fact about a object system class is mapped to the target domain fact which fits best with mappings between its linked facts about the object system class. Thus, the structure match uses a complex and but pragmatic search strategy. It reduces the search space to avoid exponential combinations in the search space. These complex algorithms are demonstrated using the following example.

The example matches a partial requirement specification for a library's stock control system to the object containment system model which can be specialised to all key abstractions of stock control domains. Structure mapping occurs in two stages. State transitions, object structures and object categorisations are mapped before second-order object relations due to dependencies between knowledge types. The first bout of structure mapping identifies the structure match between objects and facts shown in Figure 4. Mappings are indicated by equivalent positions in the figures. Object-pair mappings are inferred from local mappings between fact-pairs.

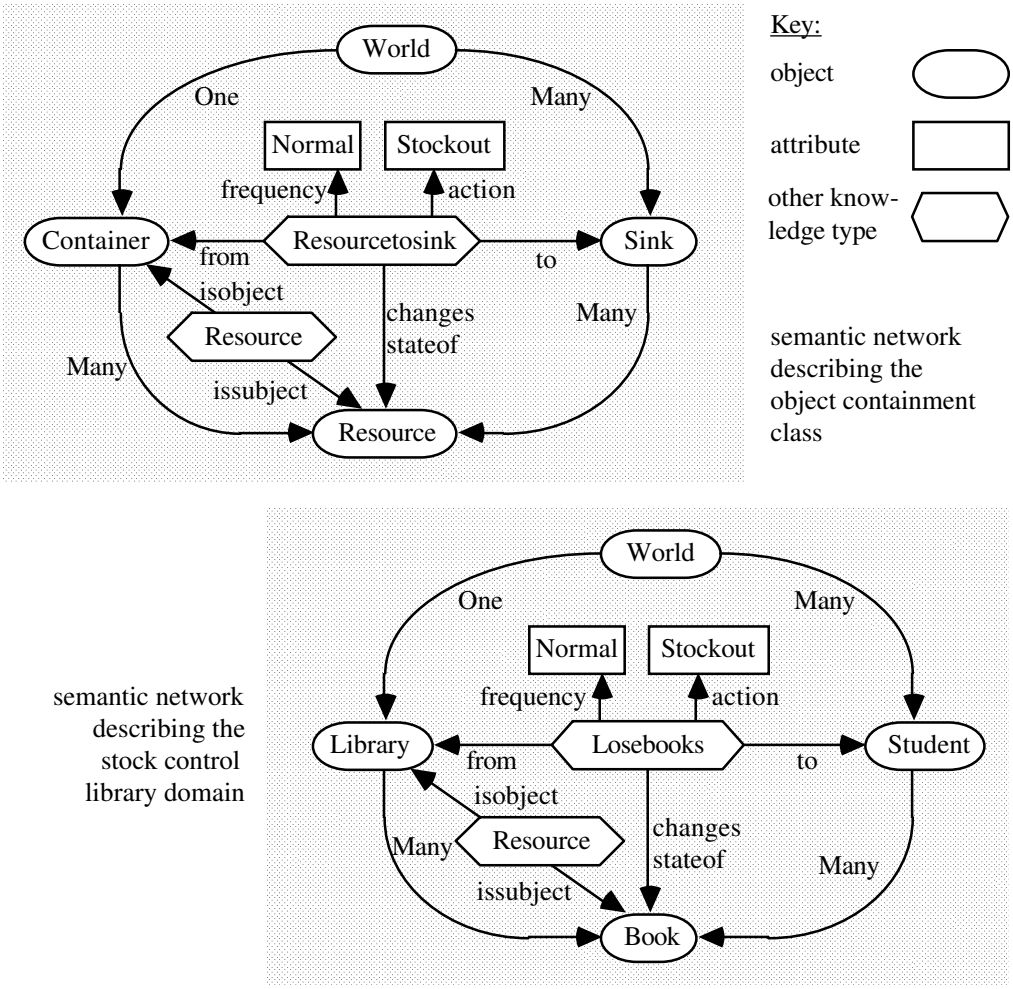

Figure 4 - structure matching in the stock control example

The structure mapper determines the best fit local mapping for each fact in each object system model. Figure 5 shows a fact describing the object structure of an object system model and two facts describing object structures of the requirement specification. Only one of these two target facts can be matched to the class fact, so the structure mapper must determine which abstract fact fits best with the overall match. The best mapping is selected by a quantitative degree of fit for each mapping to its neighbouring facts. To this end, each target fact receives a score for every neighbouring candidate mapping which it is connected to, a candidate mapping also occurring if the neighbouring facts in the target and object system model are semantically equivalent and can be mapped. In Figure 5, the *container* object is also defined in two other facts, hence they are neighbours. Neighbouring facts are matched using local mappings. For the first target fact there are two neighbouring facts in the object system class which are matched locally but for the second target fact there are no local mappings, suggesting that the mapping with the first fact may fit best into the analogical match. Mappings with *resource* object are also counted to give a score of overall fit for each target fact. As expected, the first target fact is the best match to the abstract fact because it is connected to 4 potential neighbouring mapped facts while the second fact only has 2 neighbouring matches. This matching process is repeated for each fact describing the object system model until a degree of match between it and each class has been calculated. The degree of

similarity between mapped objects is calculated from the total scores of fact mappings which include each object pair, therefore if *Library* and *Container* are mapped in three facts which scores 2, 5 and 3, the score for the degree of similarity between these two objects is 10.

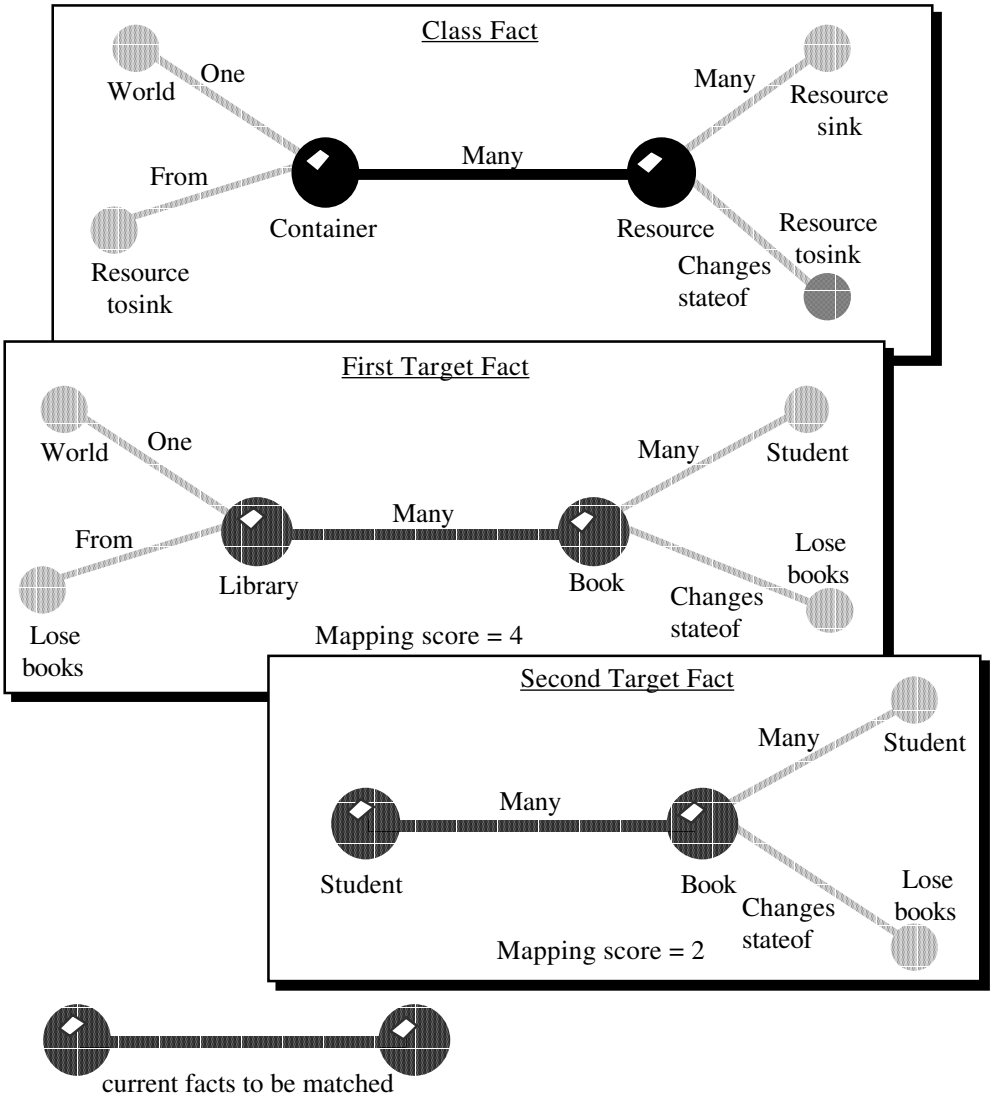

Figure 5 - Example demonstrating the structure matching algorithm to select the most appropriate domain fact for each candidate fact about the domain class. Resultant object mappings are: Container <-> Library; Resource <-> Book.

This matching procedure is repeated for each state transition, object structure and object categorisation describing the object system model. It enables comprehensive analogical matching of requirement specifications to domain abstractions without the computational costs associated with previous computational models of analogical reasoning. The structure mapper can produce inconsistent mappings between both facts and objects, therefore it must rationalise the result of the mapping process before presenting results to the requirements engineer. However, this

rationalisation should be a cooperative task with the requirements engineer resulting in the acceptance of rejection of proposed mappings with object system models.

Structure matching is then extended to second-order object relations using the higher-order structure mapper component. Results from the structure matcher are passed to the matcher controller to determine the overall validity of the match. Resulting fact-pair and object-pair mappings act as input to the rule-based matcher which operationalises the same algorithms using a different implementation (Maiden & Sutcliffe 1993d).

3.6: Validation

The architecture and algorithms of the domain matcher have been validated through testing of two implementations of the structure matcher. The basic structure matching algorithms were implemented in AIR (Maiden & Sutcliffe 1991) and empirical studies with real but inexperienced requirements engineers revealed that the domain matcher retrieved a relevant domain abstraction for 50% of all requirements and domain facts entered by users, despite the incompleteness and incorrectness of many of these requirements (Maiden & Sutcliffe 1993a). The domain matcher was reimplemented as part of Nature's coordinated demonstrator (Jarke et al. 1993b). It was shown to retrieve correct object system models from partial and incorrect requirements specifications, although evidence remains anecdotal due to the complexity of the algorithm. More comprehensive validation of the algorithm will take place throughout the Nature project.

4: Problem Classifier

The problem classifier is a computational mechanism to detect problem situations in the requirement specifications. These problems are inferred from recorded fact-pair and object-pair mappings. The classifier feeds off the domain matcher and provides explanations using the problem notepad (Maiden & Sutcliffe 1994). It detects problem types defined by a taxonomy of requirement specification problems discussed by Meyer (1985) and Roman (1985), although some these problem definitions have been changed to suit the functionality of the domain matcher. The architecture and algorithms of the problem classifier have been implemented on a SparcStation IPX using BIMprolog (Jarke et al. 1993b). The problem classifier accesses the problem detection ruleset, object- and fact-pair mappings, the requirement specification and domain abstractions. It results in a set of detected problems and provides rationale for these problems which permits their explanation to the requirements engineer. The paper will describe the algorithms for detecting the more important problem situations such as incompleteness, overspecification and inconsistencies.

4.1. Algorithms for Detecting Incompleteness

Two algorithms detect incompleteness in the form of features with no possible mapping to the object system model or improbable fact mappings which are inconsistent with other fact-pair and object-pair mappings. In the latter case, incompleteness is distinguished from inconsistencies by

the degree of inconsistencies inferred by the domain matcher. The first type of incompleteness of a requirement fact Ft is inferred by:

$$\forall(Ft,Fc)\text{incompleteness}(Ft) \Rightarrow \neg\text{factmapping}(Ft,Fc)$$

where Fc is a fact about an object system. Incompleteness is tested for all knowledge types in the knowledge meta-schema. The second form of incompleteness is defined by the following algorithm:

$$\forall(Ft,Fc)\text{incompleteness}(Ft) \Rightarrow \text{factmapping}(Ft,Fc) \;\; \text{mapping-score}(Ft,Fc)^{2}5 \; \text{objectmappings}(Ft)_\text{objectmappings}(Fc)$$

where objectmappings(Fc) and (Ft) represent the best-fit object mappings between the object system model and requirement facts inferred by the domain matcher.

4.2. Algorithms for Detecting Overspecification

Overspecification of requirements is defined by the existence of facts in the specification which have no mappings to facts in the object system model. Two algorithms for detecting overspecification in the requirement specification. For a domain abstraction fact Fc:

$$\forall(Ft,Fc)\text{overspecification}(Fc) \Rightarrow \neg\text{factmapping}(Ft,Fc)$$

where Ft is a fact in the requirement specification. The second type of inconsistencies is defined by the following algorithm:

$$\forall(Ft,Fc)\text{overspecification}(Fc) \Rightarrow \text{factmapping}(Ft,Fc) \;\; \text{mapping-score}(Ft,Fc)^{2}5 \; \text{objectmappings}(Ft)_\text{objectmappings}(Fc)$$

where objectmappings(Fc) and (Ft) represent the best-fit object mappings between the object system model and requirement specification.

4.3. Algorithms for Detecting Inconsistencies

Inconsistencies in requirement specifications are defined by either the existence of two or more requirement facts that define the domain in an incompatible way or by the presence of requirements or domain facts that make it possible to interpret a fact of the problem in at least two different ways. Inconsistencies can either be contradictions or ambiguities in the requirement specification (Meyer 1985). Inconsistencies are detected from mismatches between interrelated knowledge structures describing the requirement specification and retrieved object system model. Three algorithms detect inconsistencies between sets of fact-pair mappings, sets of object-pair mappings and fact- and object-pair mappings. Algorithms for inferring inconsistencies between

fact-pair mappings and between object-pair mappings are defined by an arbitary value representing the degree of closeness of the mapping scores S1 and S2. Therefore inconsistencies between two fact-pair mappings Fm1 and Fm2 are defined as:

$$\forall(Fm1,Fm2)\text{inconsistency}(Fm1,Fm2) \Rightarrow \text{mapping-score}(Fm1)=S1 \;\; \text{mapping-score}(Fm2)=S2 \;\; S1/S2^{2}1.5$$

where Fm1 is the fact-pair mapping which is consistent with explained, best-fit object-pair mappings and Fm2 are all other fact-pair mappings inferred by the domain matcher. Inconsistencies between two object-pair mappings Om1 and Om2 are defined by:

$$\forall(Om1,Om2)\text{inconsistency}(Om1,Om2) \Rightarrow \text{anyobjects}(Om1)_\text{anyobjects}(Om2)$$

where Om1 is a best-fit object-pair mapping inferred by the domain matcher and Om2 represents all other inferred object-pair mappings with the two objects belonging to the best-fit object-pair mappings. Inconsistencies between an object-pair mapping Om and fact-pair mapping Fm are detected using the following algorithm:

$$\forall(Ft,Fc)\text{Inconsistency}(Ft,Fc) \Rightarrow \text{factmapping}(Ft,Fc) \;\; \text{mapping-score}(Ft,Fc)>5 \;\; \text{objectmappings}(Ft)_\text{objectmappings}(Fc)$$

where objectmappings(Fc) and (Ft) are the best-fit object mappings.

The problem classifier has been implemented as part of Nature's coordinated demonstration and has been shown to detect typical problems in requirement specifications (Maiden 1993). However, this prototype has revealed the need for advanced explanation strategies to ensure effective understanding and resolution of detected problems. The advantages of these implemented algorithms to the requirements engineer are shown in the following requirements engineering scenario.

5 : Simple Scenario Showing Benefits from the Domain Matcher & Problem Classifier

This short scenario demonstrates active guidance and critiquing during requirements engineering from successful retrieval of domain abstractions. The requirements critic (Maiden & Sutcliffe 1994) is an intelligent tool which exploits retrieved domain abstractions during requirements engineering. It explains retrieved domain abstractions to a requirements engineer with moderate experience to assist requirement specification for a computerised stock control system in a library. The requirements engineer has conducted a fact gathering interview with a head librarian. This interview revealed the need for a stock control system for a lending library. The system must identify missing or damaged books which are no longer in the library, see Jarke et al. (1993b). The requirements critic exploits the domain specialisation hierarchy shown in Figure 7.

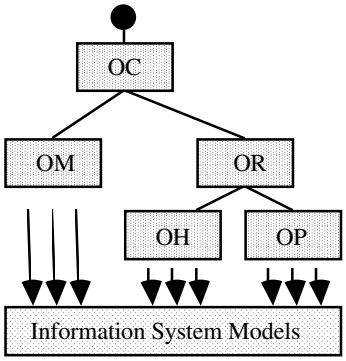

Figure 6 - example domain specialisation hierarchy showing five object system models

The high-level object system represents the concept of resource containment and is specialised by the OM object system to include resource repletion. This specialised abstraction can be instantiated to all stock control domains. This paper reports the final phase of the scenario which results from successful retrieval of an object system model by the domain matcher and problem detection by the problem classifier, see Figure 7.

The OM object system and linked information systems defining functional requirements are explained to the requirements engineer:

- the domain abstraction is explained using visualisation and text-based descriptions of domain structure and behaviour, formal definitions of the abstraction and explanation of object-pair mappings. Further domain explanation and exploration is available by requesting animation of domain behaviour. This ensures domain understanding prior to requirement specification;
- guided explanation of information systems and prototypical examples takes place;
- information systems are explained using formal definitions, visualisation, animation and prototypical examples. Information systems can be reused to define functions for stock takes, purchase authorisation for books and various report functions.

This assistance is possible due to retrieval of domain abstractions by the domain matcher and detection of problem situations by the problem classifier, as shown by the windows in Figure 7.

6 : S u m m a r y

This paper outlines two implemented computational mechanisms which retrieves and exploits generic domain knowledge during requirements engineering. The domain matcher is a computational mechanism of analogical reasoning for retrieving software engineering domain knowledge. It is from an artificial intelligence tradition. The problem classifier is a rule-based reasoning mechanism to exploit inferences made by the domain matcher to resolve typical problems in requirements engineering. The paper outlines both mechanisms and shows their power from part of the ESPRIT Nature project's common example. It is the basis for the provision of some artificial intelligence techniques to real and difficult software engineering problems.

R e f e r e n c e s

- Branting L.K., 1991, 'Building Explanations from Rules and Structured Cases', International Journal of Man-Machine Studies 34, 797-837.
- Falkenhainer B., Forbus K.D. & Gentner D., 1989, 'The Structure-Mapping Engine: Algorithm and Examples', Artificial Intelligence 41, 1-63.
- Fugini M.G., Guggino M. & Pernici B., 1991, 'Reusing Requirements through a Modeling and Composite Support Tool', Proceedings of 3rd International Conference CAiSE'91, Trondheim (Norway), May 1991, ed. R. Anderson, J.A. Bubenko & A. Solvberg, Lecture Notes in Computer Science 498, Springer-Verlag, 50-78.
- Holyoak K.J. & Thagard P., 1989, 'Analogical Mapping by Constraint Satisfaction', Cognitive Science 13, 295-355.
- Jarke M., 1992, 'The ConceptBase Manual, Version 3.0', RWTH-Aachen.
- Jarke M., Bubenko Y., Rolland C., Sutcliffe A.G. & Vassiliou Y., 1993a, 'Theories underlying requirements engineering: an overview of NATURE at genesis', Proceedings of IEEE Symposium on Requirements Engineering, IEEE Computer Society Press, 19-31.
- Jarke M., Pohl K., Jacobs S., Bubenko J., Assenova P., Holm P., Wangler P., Rolland C., Plihon V., J. Schmitt, Sutcliffe A.G., Jones S., Maiden N.A.M., Till D., Vassilou Y., Constantopoulos P. & Spandoudakis G., 1993b, 'Requirements Engineering: An Integrated View of Representation', to appear in Proceedings 4th European Software Engineering Conference, Garmesh-Partenkirchen, September 1993.
- Maiden N.A.M., 1993, 'AIR's Problem Classifier', Nature Report CU-93-00H, Department of Business Computing, City University.
- Maiden N.A.M. & Sutcliffe A.G., 1994, 'Requirements Critiquing Using Domain Abstractions', to appear in Proceedings First International Conference on Reqiuirements Engineering, IEEE Computer Society Press.
- Maiden N.A.M. & Sutcliffe A.G., 1993a, 'Requirements Engineering by Example: An Empirical Study', Proceedings of IEEE Symposium on Requirements Engineering, IEEE Computer Society Press, 104-112.
- Maiden N.A.M. & Sutcliffe A.G., 1993b, 'The Domain Theory: Object System Definition', Nature Report CU-93-00A, Department of Business Computing, City University.
- Maiden N.A.M. & Sutcliffe A.G., 1993c, 'The Domain Theory: Information System Definition', Nature Report CU-93-00B, Department of Business Computing, City University.
- Maiden N.A.M. & Sutcliffe A.G., 1993d, 'The Domain Matcher: Architecture and Algorithms', Nature Report CU-93-00D, Department of Business Computing, City University.
- Maiden N.A.M. & Sutcliffe A.G., 1992a, 'Exploiting Reusable Specifications Through Analogy, Communications of the ACM. 34(5), April 1992, 55-64.
- Maiden N.A.M. & Sutcliffe A.G., 1992b, 'Process of Domain Knowledge Reuse in Nature's Initial Integration Prototype', Nature Report CU-92-00A, Department of Business Computing, City University.
- Maiden N.A.M & Sutcliffe A.G., 1991, 'Analogical Matching for Specification Retrieval', Proceedings 6th Knowledge-Based Software Engineering Conference, IEEE Computer Society Press, 108-116.
- Maiden N.A.M. & Tyndale D.E., in press, 'Reuse of Domain Abstractions During Requirements Engineering: an Explanation is Required', submitted to journal.
- Meyer B., 1985, 'On Formalism in Specifications', IEEE Software, January 1985, 6-26.
- Mylopopoulos J., Borgida A., Jarke M. & Koubarakis M,, (1990), 'Telos: Representing Knowledge about Information Systems', ACM Transactions on Office Information Systems 8(4), 325.
- Reubenstein H.B. & Waters R.C., 1991, 'The Requirements Apprentice: Automated Assistance for Requirements Acquisition', IEEE Transactions on Software Engineering 17(3), 226-240.
- Roman G., 1985, 'A Taxonomy of Current Issues in Requirements Engineering', IEEE Computer, April 1985, 14-22.

Figure 7 - assistance provided by the requirements critic from a retrived object system model instantiated to stock control