[28] Reubenstein H.B. & Waters R.C., 1991, 'The Requirements Apprentice: Automated Assistance for Requirements Acquisition', IEEE Transactions on Software Engineering 17(3), 226-240.

[29] Rosch, E., 1983, 'Prototype Classification and Logical Classification: The Two Systems'. In New trends in conceptual representation: Challenges to Piaget's theory? (ed. E.K. Scholnick), Lawrence Erlbaum.

[30] Shaw M., 1991, Heterogeneous Design Idioms for Software Architecture', Proceeding 6th International Workshop on Software Specification and Design', IEEE Computer Society Press, 158-165.

[31] Sowa J.F., 1984, 'Conceptual Structures, Information processing in Mind and Machine', Addison Wesley, Reading, MA.

[32] Sutcliffe A.G. & Maiden N.A.M., 1993, 'Bridging the Requirements Gap: Policies, Goals and Domains'. Proceedings 7th Intl Workshop on Software Specification & Design, IEEE Computer Society Press, 52-55.

[33] Sutcliffe A.G. & Maiden N.A.M., in prep, 'A Theory of Domain Knowledge for Requirements Engineering'.

stable over a 12 month period, with the capacity to interpret many different application descriptions such as the IFIP conference system, IWSSD meeting scheduler system and so on. We expect to make refinements some classes and add more information system models, although overall, the models and matcher have proved to be robust. However, evaluation in an industrial environment will provide the real test, and we look forward to that challenge.

Acknowledgements

This work was partially supported by the European Commissions Esprit programme in basic Research Action 6353 NATURE (Novel Approaches to Theory Underlying Requirements Engineering).

References

[1] Arango G., Schoen E. & Pettengill R., 1993, 'Design as Evolution and Reuse', Proceedings of 2nd International Workshop on Software Reusability, IEEE Computer Society Press, 9-18.

[2] Belkin, N.J. 1980, 'Anomalous States of Knowledge as a Basis for Information Retrieval', Canadian Journal of Information Science 5, 133-143.

[3] Constantopoulos P., Jarke M., Mylopoulos J. & Vassiliou Y., 1991, 'Software Information Base: A Server for Reuse', Technical Report, FORTH Research Institute, Univ of Heraklion, Crete.

[4] Croft, W.B. & Thompson, R.H., 1987, 'I3R: A New Approach to the Design of Document Retrieval', Journal of the American Society for Information Science 38(6), 389-404.

[5] Dardenne A., van Lamsweerde A., & Fickas S., 1993, 'Goal-Directed Requirements Acquisition', Science of Programming 20, 3-50.

[6] Dumais S.T., 1988, 'Textual Information Retrieval', In 'Handbook of Human Computer Interaction' (ed. Hellander), Elsevier Science Publishers 673-700.

[7] Falkenhaimer B., Forbus K.D. & Gentner D., 1989, 'The Structure-Mapping Engine: Algorithm and Examples', Artificial Intelligence 41, 1-63.

[8] Fischer G., Girensohn A., Nakakoji K., & Redmiles D., 1992, 'Supporting Software Designers with Integrated Domain Oriented Design Environments', IEEE Transactions on Software Engineering 18(6), 511-522.

[9] Flores F, Graves M., Hartfield B. & Winograd T., 1988, 'Computer Systems and the Design of Organisation Interaction',ACM Transactions on Office Information Systems, 6(2), 87-108.

[10] Gentner D., 1983, 'Structure-Mapping: A Theoretical Framework for Analogy', Cognitive Science 7, 155-170.

[11] Guindon R., 1990, 'Designing the Design Process: Exploiting Opportunistic Thoughts', Human-Computer Interaction 5, 305-344.

[12] Holyoak K.J. & Thagard P., 1989, 'Analogical Mapping by Constraint Satisfaction', Cognitive Science 13, 295-355.

[13] Harandi M.T. & Lee M.Y., 1991, 'Acquiring Software Design Schema: A Machine Learning Perspective', Proceedings 6th Knowledge Based Software Engineering Conf., IEEE Computer Society Press, 188-197.

[14] Jackson M.A., 1983, System Development, Prentice Hall, Englewood Cliffs, NJ.

[15] Jarke M., 1992, ConceptBase V3.1 user manual, RWTH-Aachen, Aachen, Germany.

[16] Jarke, M., Bubenko, J., Rolland C., Sutcliffe A.G., & Vassiliou Y., 1993, 'Theories Underlying Requirements Engineering: An Overview of NATURE at Genesis', Proceedings of IEEE Symposium on Requirements Engineering, IEEE Computer Society Press 19-31.

[17] Katz S., Richter C.A. & The K., 1987, 'PARIS, A system for Reusing Partially Interpreted Schemas', Proceeding 9th International Conference of Software Engineering, IEEE Computer Society Press, 377-385.

[18] Krueger C.W., 1992, 'Software Reuse', ACM Computing Surveys 24, 131-183.

[19] Maiden N.A.M & Sutcliffe A.G. 1992a, 'Exploiting Reusable Specifications Through Analogy', Communications of the ACM 35(4), 55-64.

[20] Maiden N.A.M. & Sutcliffe A.G., 1992b, 'Analogously-based reusability', Behaviour & Information Technology 11(2), 79-98.

[21] Maiden N.A.M and Sutcliffe A.G. 1993, 'Requirements Engineering by Example', Proceedings 1st Intl Symposium on Requirements Engineering, IEEE Computer Society Press 104-112.

[22] Maiden N.A.M. & Sutcliffe A.G., 1994a, 'Requirements Critiquing Using Domain Abstractions', Proceedings of 1st International Conference on Requirements Engineering, IEEE Computer Society Press, 184-193.

[23] Maiden N.A.M. & Sutcliffe A.G., 1994b, 'Computational Mechanisms for Analogical Retrieval of Domain Knowledge during Requirements Engineering', Technical Report CU-94-00C, Centre for Human-Computer Interface Design, City University.

[24] Maiden N.A.M. & Mistry P., in prep, 'How People Categorise Requirements for Reuse: A Natural Approach'.

[25] Matsumoto Y., (1993), 'Experience from Software Reuse in Process Control Applications', Proceedings of 2nd International Workshop on Software Reusability (Advances in Software Reuse), IEEE Computer Society Press, 188-195.

[26] Mylopoulos J., Borgida A., Jarke M. & Koubarakis M, 1990, 'Telos: Representing Knowledge about Information Systems', ACM Transactions on Office Information Systems 8(4), 325.

[27] Prieto-Diaz R., 1991, 'Implementing Faceted Classification for Software Reuse', Communications of the ACM 34(5), 88-97.

which supports the Telos modeling language. Partial validation of this retrieval mechanism has been reported in [23]. The domain abstractions have also been validated through empirical research which elicited the mental categorisations held by experienced software engineers. Card sorts elicited the mental criteria and categories used by such software engineers to divide up software engineering domains for reuse [24]. To be effective, it was hypothesised that these criteria and categories were equivalent to knowledge types and object system models in the domain space. Results supported the basic hypotheses but led to some changes to the domain space. In this section however, we look forward to the development of this work in reuse libraries.

Our modeling framework is related to reuse approaches of software architectures and schemas [17, 18, 30]. However, object system models differ because they describe the problem space rather than the solution space of architectures and schemas. Object system models could serve to reduce the cognitive distance in Kreuger's [18] sense between requirements analysis and eventual satisfaction of those requirements in software. The object system models and retrieval mechanisms are standalone techniques for domain knowledge reuse, but to obtain further leverage, these mechanisms need to be integrated with reuse components libraries containing specifications, design and code.

The matching process can act as a retrieval mechanism with more power than faceted classification schemes because the storage and retrieval of reusable components is based on their underlying abstractions rather than lexical labels on properties. Moreover, evidence from validation studies supports our claim that these abstractions approximate to mental models of domains held by experienced software engineers [24]. The object system models can therefore facilitate communication with software engineers during the requirements engineering. Retrieved object system models are also explained to promote domain and problem understanding [22]. Reusable components suitable for one or more object system models could be retrieved from a repository, so reuse and requirements definition can proceed side by side.

The first problem is to link object system models with reuse component libraries. This could be achieved by reversing the retrieval process. The software documentor enters terms to describe the application context from which a reusable component was derived. Facts are acquired in a directed dialogue [20, 21] which iteratively helps the user home in on an object system model corresponding to general class of the original application. Pointers could then be entered into the library from the reusable component to one or more models. The object system models provide an access path to different groupings of reusable components, depending on the matching process. When a software developer wants to retrieve components the process is reversed, the input description is matched first to the object system models, then pointers are traced from the OSMs to library components which are reusable for the new application.

On the other hand, lower level object system models may be directly reused. So far we have concentrated on this route and used the retrieved models for assisting requirements validation as well as providing knowledge for reuse.

7: Discussion

The library architecture and domain matcher offer an alternative solution to faceted classification for reusable repositories. As access to components is through goal-related domain abstractions, the matching should be faithful to the underlying problem structure. This may eliminate the risk of mis-classification and poor precision in retrieval inherent in many indexing schemes, as demonstrated by information science research [2, 6]. However, we know from experience that abstraction incurs a penalty in understanding. Inexperienced software engineers have difficulties when comprehending abstract models [20, 21], although experts fare better [20]. Furthermore, feedback plays a crucial role in the iterative nature of developing the user's understanding of their needs during information retrieval [4]. Explanation will therefore be a necessary part of any intelligent retrieval, and we have extended the domain matcher to provide explanation, although this has focused on requirements validation.

One problem with our current matching process is redundancy in retrieved components. The matcher tends to find an aggregation of object system models which share many objects. This necessitates elimination of duplicates then integration of the different model fragments. This problem is part of our future work which may explore concepts of generalised transaction types [9] as higher order templates binding several object system models and agents together. Further extension of the models may be to add design rationales to object and information system models, following the design book concepts reported in [1].

Current work is evaluating the domain theory using card sorting, laddering and other knowledge acquisition techniques. Comprehension tests of the object system models are also in progress. These models have become

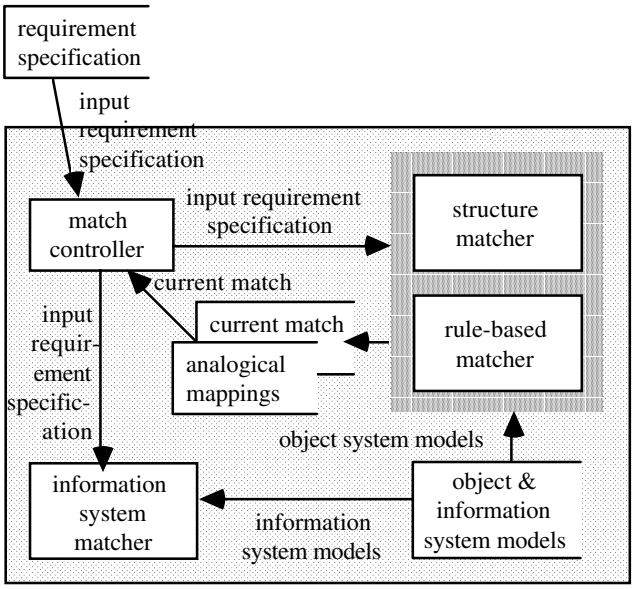

Figure 3. Architecture of the domain matcher.

The matcher is described in detail in [23]. Only an outline is given here. The structure matcher maps primary state transitions, states and semantic relations. It infers interrelated knowledge structures shared by both the input application and one or more object system models. Each fact is mapped if the matcher finds two instances of the same knowledge type which have the same syntactic connectivity in the input and each object system model. This matching occurs in two phases:

- type-based local mappings between all model components to define the potential of a match and filter out unnecessary structure matching.
- structure matching to determine a quantitative score of similarity between the input model and each object system model.

The match controller calls the structure, rule-based or information system matcher to match to the requirement specification at different stages in the matching process. Level-A discriminates between object system models using primary state transitions, primary states, some secondary states and state transitions, and semantic relations. Levels -B, -C and -D discriminate using goal states, events and stative conditions and object properties.

The structure matcher borrows algorithms for analogical reasoning implemented in existing computational mechanisms (e.g. [7]). Mappings can exist between any two knowledge instances of the same type (i.e. state transition, state, semantic relation) which match on the semantics of their attributes, for example two state transitions are the same type of transition. However, the structure matcher constrains mappings to only those which support an interrelated knowledge structure shared by the requirement specification and object system model. Therefore, matching is an exhaustive search of pairwise combinations of knowledge types, constrained by connections from the current locus of search. Structure matching is refined by the use of weights which denote the importance of mappings between different knowledge types. State transitions with respect to structure are more powerful discriminators, therefore the matcher gives greater importance to these mappings than to mappings between semantic relations.

The structure matching algorithms impose structure as a constraint to confirm or reject local mappings. These complex algorithms are described in [23]. A fact capture dialogue to acquire descriptions of the new application is driven by the knowledge meta-schema. The matcher interleaves fact acquisition with matching and explanation of retrieved object system models as it progresses down the specialisation hierarchy, see Figure 4. The matcher requests different facts as it progress as rule-based matching specialises an existing match using different knowledge types such as events, actions, stative conditions and goal states. The process continues until it finds a leaf model which represents the best fit.

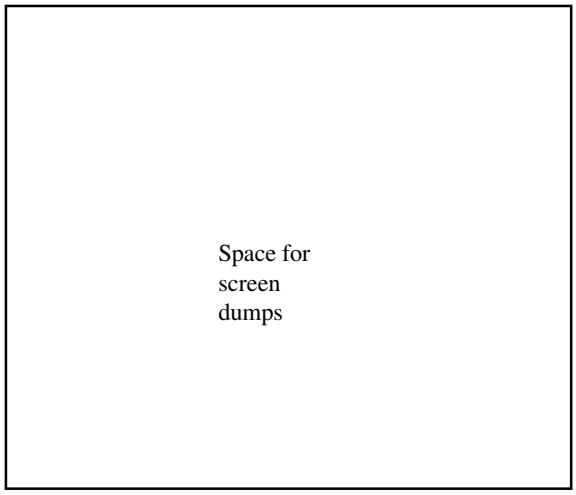

Figure 4. Explanation of one object system model by the requirements critic.

6: Development of domain models for reuse libraries

The domain matcher and data base of object system models have been implemented. The matcher has been constructed using Prolog by BIM version 4.0 under SUN 4.1.3 [23]. It accesses object system models held in ConceptBase, an object-oriented deductive data base

applications, controllers and pilots in air traffic control systems.

Level-1 class: object sensing: the purpose of this class is to monitor movement of objects in the real world and record their relative state with respect to a spatial/physical structure object. Many structure objects exist, each one modeling a significant component of the real world. Key objects move with respect to this structure and these changes are reported to the sensing structure object. This class may include a sensing agent which reports on this behaviour and specialises to spatial object sensing application (e.g. radar detection of air traffic), temporal monitoring and property sensing applications (e.g. monitoring temperature, pressure in process control applications).

Level-1 class: object messaging: the purpose of this class is to communicate key objects which have some information value between structure objects representing the communication topology. Specialisations of this class add agents which are purposeful originators and receivers of messages. The class is characterised by one or more key objects (messages), two or more structure objects representing the topology and at least one state transition for exchange of messages. Specialisations in this class add more structure objects for canonical topologies and more transitions with relationships to represent different communication protocols. CSCW (Computer support co-operative work) applications are usually aggregations of object messaging and the next class, object manipulation.

Level-1 class: object manipulation: the key object does not change state with respect to structure, instead it may undergo many incremental changes which change its secondary state. This model represents a large family of applications which includes computer based tools for word processing, graphics packages, decision support, design support, CAD systems, etc. From the design viewpoint this class also contains all computer programs. However, the focus of the domain theory is on requirements, so the essence of this class is applications which require manipulation a structure object as an artefact in its own right. These systems are characterised by one of more object structures which are manipulated by an agent, the support tool or human assistant, in response to requests from the user agent.

This is the breadth of the object system model space as it currently stands. Further analysis of applications may lead to discovery of new high-level object system models, however the set described above has remained stable over 18 months of successive analysis of applications from real life and case study descriptions in the software engineering literature.

So far detailed specialisation has concentrated on the object containment, object allocation, object sensing, object composition, object decomposition and agent-object control system models, yielding over 150 object system models. A set of 20 information system models have also been defined. Examples of these includes functions such as reporting, summarising, progress checking, simple counts and listing.

It is important to emphasise that most applications are aggregations of object system models. A typical library domain will have an object hiring system to handle book loans, an object supplying system for book stock control and resource consumption for new book stock acquisitions. This leads into consideration of the next part of the domain theory, the matching process.

5: The domain matcher

The aim of the domain matcher is to determine a degree of similarity between input facts describing a new application and one or more object system models stored in the knowledge base. This is achieved by matching facts to a high-level object system model then specialising this match to lower level object system models until no further specialisation is possible. Structure matching is used to establish an initial match which is then specialised by rule-based matching. Information system models, linked to leaf-node object system models, are then retrieved to complete the process, see Figure 3.

structure object to model the source of new objects, and a further state transition to represent the transfer of objects from a source to the owning object structure. This specialisation corresponds to a variety of inventory management systems with outbound and inbound transfer of objects. Purpose in this case is not only orderly transfer of objects to a client but also replenishment of resources from suppliers.

Level-2 class: object returning*:* this model is another sub-tree branching from the object containment system model. In this case the specialisation adds a single state transition inbound from the client to the owning object structure. This object system model specialises to hiring, servicing and repair applications.

Level-3 class: object hiring*:* the purpose of these object system models is to control the loan, hire, or temporary transfer of ownership of some resource from an organisation which normally holds the resource and a client who requests possession of the resource. The model is characterised by two structure objects, the owning organisation and the client, one key object (the resource) and two transitions representing outbound transfer of the resource to the client and a subsequent return.

Level-3 class: object augmentation: the purpose in this object system is to control the exchange of objects between a client and a servicing organisation. Further specialisations map to service and repair applications in which the key object is in some way improved or has valued added to it by the servicing structure object. This model has the same structure objects as in the object hiring system model, but is differentiated by the concept of augmentation of the key object. This concept is represented by a semantic relationship between the inbound and outbound state transitions.

Level-1 class: object composition: in this model the purpose is to aggregate key objects and thereby synthesise new ones. Concrete instantiations are a wide variety of manufacturing systems which take in raw materials and compose them by some process into a product. This model is characterised by two structure objects (the source of objects to be composed, and the destination of composed objects), two sets of key objects (components and composite objects) and one transition for merging component objects into composites. A controlling agent is also present in this model to describe organisation of the process.

Level-1 class: object decomposition: this is the inverse of composition which maps to concrete examples of systems which disaggregate objects, e.g. disassembly, unpacking or applications involving decomposition. This system is characterised by source and destination structure objects, two sets of key objects, composites and components, and one transition representing decomposition. As with object composition a controlling agent is present.

Level-1 class: financial object exchange: this class is a special case when structure has not been used to differentiate classes at the top level. Financial and accounting systems share many structural properties with other object system models. Furthermore, in many domains accounting systems are aggregated with other transaction processing. Given that finance is a salient descriptor of business information systems this class is discriminated by possessing a financial object property. The top level structure is the same as object containment and this class shares many of the same structural specialisations. Loans, for instance, has the same object structure as the object repair system model as the loan is augmented by being repaid with interest.

Level-1 class: object allocation: this object system models applications that create an object state change that is a precursor to some other transaction. Examples are booking, reservation and allocation applications. The purpose of such systems is to establish an association between a key object and a client so that the further transaction is enabled. This class is modeled by two structure objects, the source of unallocated objects and destination of allocated objects and an agent which is responsible for the activity which determines the link.

Level-1 class: agent-object control: this model represents command and control applications. The class is characterised by an agent and key objects contained in one structure object. Key objects are contained within a structure which models a controlled part of the real world. The transitions represent messages passed from the agent to the objects to change their behaviour in some way. Objects respond and change state within the structure and specialisations of this class refine how that change occurs with respect to structure (e.g. spatial, temporal change, change in property, number, etc.). The purpose of this class is expressed as an agent having some effect on objects' behaviour in a structure. Specialisations of this class are command systems which add further agents for military command and control

primitives have been formalised in the Telos knowledge representation language [26] and implemented in ConceptBase, a deductive database [15]. Telos semantic primitives are mapped to first-order predicate logic definitions, similar to first-order logic definitions in KAOS [5]. Formal definitions also assist reasoning with goal states and behaviour during requirements definition and understanding of domain abstractions.

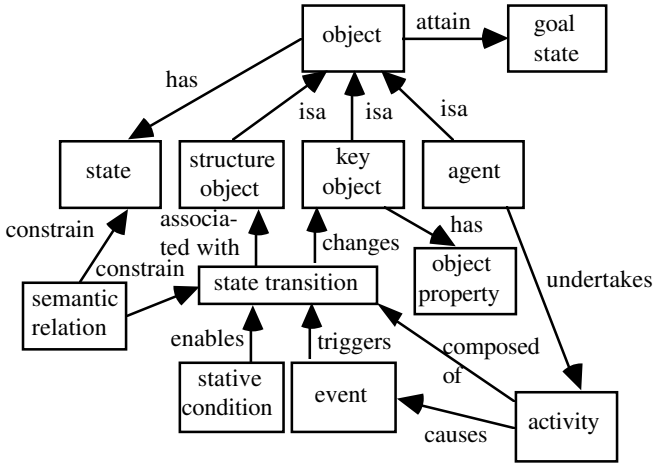

Figure 1. Meta-schema of knowledge types for domain modeling.

4: Object & information system models

Object system models are structured in a class specialisation hierarchy. The models and class structure have been validated using knowledge elicitation studies with expert software engineers [24] and systematic investigation of case studies reported in the literature. The concepts of 'use cases' or 'essential transactions' related to the application problem were used to identify the objects and behaviour included in object system models.

Object system models at different levels of abstraction are distinguished using different knowledge types. Structure and purpose are the most important constructs at higher levels because the analogical retrieval process matches interrelated knowledge structures rather than isolated facts 7, 10, 12]. Structure is modeled as relationships between goal states, states defined using structure objects and key objects, and transitions between states which help attain the goal states. Therefore, purpose expressed as goal states is linked to state transitions which are initiated by agents. The structure of the hierarchy is illustrated in Figure 2.

The first and second levels in the hierarchy are defined by state transitions, agents, states and semantic relations. Lower-level object system models are specialised by adding further knowledge types such as goal states, triggering events, preconditions on transitions and object properties. The top level of the tree has been pruned, as models at such high levels of abstraction do not contain sufficient knowledge for retrieval through structure matching. The model population consists of nine separate trees each of which is specialised to 3-4 levels of sub classes. A brief description of each high-level system is given along with detail of some object system models in the object containment hierarchy.

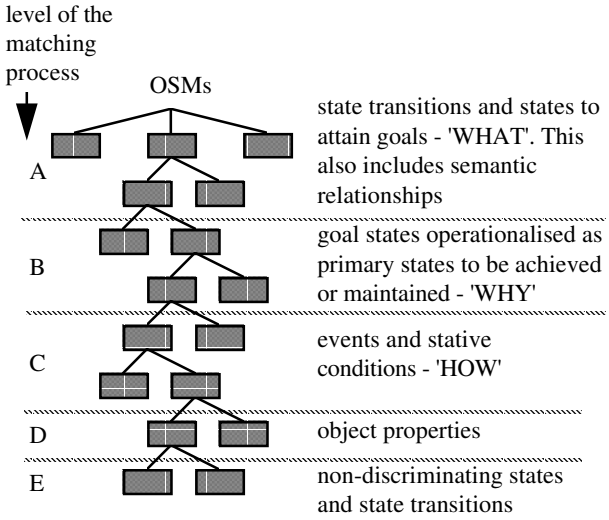

Figure 2. Hierarchical structure of object system models.

Level-1 class: object containment*:* this class is characterised by the unidirectional transfer of key objects from an owning structure to a client structure. More familiar examples are all sales order processing systems and specialisations in this tree also describe inventory and loans type applications. The model has two structure objects, one key object which is consumed and an event which models the request for the transaction. The common purpose of this class is the orderly transfer of resource type object from an owner to a client who requires the resource. This class only models the transfer of objects, while a separate abstraction is used for the monetary aspects of sales. Hence object containment system models a wide variety of inventory type applications and may be specialised to many non- sales systems where resources are transferred, e.g. internal supply of donations or resources.

Level-2 class: object management*:* this specialises the object containment system model by adding another

- structure objects are used to define the start or end states of key objects whose states are changed by state transitions. In a warehousing application, the structure object stock-bins represents the end state of a stock-in transition and the start state of the pick state transition.

Domain states: object system models have two ways of describing state. Primary states record the physical relationship between objects, for example a loaned book changes its relationship with the library structure from being in the library to being loaned by the reader. Secondary states belong to objects independent of the structural aspects of models, for example the book is reserved for a reader but it is still within the library, in that the change is not essential for the system's transaction processing.

State transitions: state transitions are manifestations of causal behaviour which satisfies a purpose and hence are linked to goals. A state transition changes the state of one object by transferring it between structure objects to achieve desired goal states. State transitions change an object's state with respect to structure objects but not the internal state of the object, as the latter change is effected by the object updating its own data structure. State transitions are assigned frequency attributes which represent the probability of transition occurrence in the domain, for instance *normal* transitions such as book returns are likely to occur while *exceptional* transitions such as late or missing returns are less likely. Furthermore, state transitions are aggregated into sequences to model activities or procedural processes initiated by agents.

Object properties: three types object properties are allowed: (i) physical objects which are subject to physical laws such as gravity or momentum. Examples include library books in the lending library domain and aircraft in an air traffic control; (ii) financial objects represent currency or entities with some monetary value. Examples of financial objects include cash in an automatic telling machine and the currency in a foreign exchange transaction; (iii) conceptual objects representing all key objects which are not physical or financial objects, for example classes to be booked in a university timetabling domain.

Events: events are sub-divided into domain and time events. An event is a single point in time when something happens. Events are implicit at the beginning and end of every state transition. To avoid redundancy we only model initiating events as 'triggers'. For instance, the dispatch of goods from the warehouse to customer (a transition) is triggered by a customer order (a request event). Domain events model the result of activities by an internal agent or received from the real world while time events emanate from a clock.

Preconditions on state transitions: preconditions are tests on objects' primary states, object quantities and durations of an object's state, as well as tests on the object's internal state. Conditions determine whether or not a state transition can occur.

Goal states: goals describe a required state which the system should satisfy. Requirements statements are usually expressed linguistically as intentions for goals, aims and objectives to be achieved. Goal statements can be refined into more formal expressions of future states which the system should, or should not, attain, and are linked to object system models in three ways: (i) requirements describe states which the object system must achieve. These requirements are satisfied by state transitions on objects; (ii) functional requirements describing some operational system behaviour, e.g. activities to be carried out by an agent to satisfy the goal state; (iii) functional requirements will necessitate production of information which may be used by an agent.

Semantic relations: these relations add further structural information to further discriminate between object system models. Semantic relations occur between primary domain states and between state transitions. Two forms of semantic relation between primary states are: (i) temporal relations which indicate that one single key object A persists in one structure object A' for a significantly longer time than key object B does in structure object B', where this duration represents the times from the first existence of A in A' and B in B'. It is defined as A longer B. It represents an occurrence which typical but does not hold for every instance of the object system model; (ii) scale relations which indicate that a greater number of key objects A are held in structure object A' than key objects B in structure object B' at any time t. It is defined as A greater B. Again, this represents an occurrence which typical but does not hold for every instance of the object system model.

Semantic ordering relations link state transitions to define sequential ordering, so that one transition precedes another in changing the state of that same key object instance.

The structure of the meta-schema is shown in the Figure 1 as an object-relationship diagram. All the meta-schema

give design ideas for a reuse library manager based on the domain abstraction space.

2: Modeling framework

A hierarchy of domain abstractions describes generalised classes of software engineering domains. The framework is based upon cognitive models of memory (e.g. [10], [29]) and experimental evidence that experienced software engineers tend to recall and reuse domain abstractions [11]. This suggests that natural units of abstraction may be aggregates of objects linked by purpose, similar to the 'use case' concept in object-oriented design. Hence the framework we propose can be viewed as object-oriented reuse at the network level rather than of individual objects.

Object systems models define the structure and behaviour of the problem space, i.e. the application objects or entities, while information system models have processes which produce reports or query screens containing information about the object system [33]. This distinction is a development of Jackson's [14] ideas on separating the fundamental entity model of a system from the functional processes which acquire information from that model, although we depart from Jackson by introducing the concept of agents into object systems. Object and information system models are distinguished by the following features:

- object system models represent domain structure and behaviour. Objects may be physical and subject to laws such as gravity and momentum, or financial or conceptual objects without physical instantiation. These models representation what is often terms the 'transaction processing' of the system;
- information system models are composed of processes which produce reports or query screens with information acquired from the object system.

Within any one application class a domain model is an aggregation of several object and information system models, hence the framework can handle applications of different sizes and complexity. A matching process (see section 5) delivers aggregation by retrieving an appropriate set of object and information system models according to input descriptions of a particular application.

Agents may be external to the automated system (i.e. users) and are involved in events that must be processed (triggers) or internal agents and initiate state transitions within the object system. Having outlined the conceptual modeling approach, the next step is to examine the knowledge meta-schema. This defines the modeling language for all the model variants in the theory.

3: Knowledge meta-schema of domain knowledge

The knowledge meta-schema defines the semantic primitives and relationships within the object and information system models. The semantic primitives consist of 10 knowledge types, most of which are familiar from other modeling languages. However we propose addition of a new concept 'structure object' which models an approximation to physical structure in the real world. This concept bridges the gap between domain-specific models [8] and modeling languages [5, 26] which provide general semantics. We introduce the structure concept as an intermediate level of abstraction between linguistically based knowledge representation, such as conceptual structures [31] and more formal semantics. The components of the meta-schema are as follows:

Objects: objects have properties and states, so unlike standard object-oriented languages, operations and methods are not part of our basic definition. These dynamic aspects are modeled in agents as sub-classes of objects. Objects are subject to state transitions and therefore change state. Key objects are the specialisations which undergo processing by the system.

Agents: agents are specialisations of objects which carry out actions resulting in events which, in turn, initiate state transitions. Each agent can either be a human or an automated process. Agents undertake behaviour and hence cause change within the system, or may acquire information about an object's state in order to produce information outputs.

Structure objects: structure objects model parts of the real world domain which have spatial properties and preserve aspects of physical structure which are important for understanding object behaviour and states, e.g. containment or possession. Structure objects add components which are normally lost from conceptual models because they represent passive entities or the whole universe of discourse, for example a library, warehouse or airspace in air traffic control. Objects can change state by reference to a structure (for example the object is removed from the warehouse and thereby deleted from stock). Structure objects have the following properties:

- structure objects must have a persistent existence which expresses containment of objects, for example one library contains many books or many aircraft can be in one airspace;

Domain Modeling for Reuse

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

Centre for Human-Computer Interface Design, City University
London, UK

Abstract

This paper presents a theory for reuse of domain knowledge during requirements engineering. The theory includes a set of formal and reusable domain abstractions, a computational analogical reasoning mechanism for their retrieval and tool support to enable both effective requirements modeling and critiquing and guided fact capture prior to retrieval. It was developed as part of the ESPRIT 6353 'Nature' basic research action. The paper argues that the theory has implications for domain modeling as well as domain knowledge reuse during requirements engineering.

1: Introduction

One of the motivations for domain analysis is the creation of libraries of reusable components. While some success has been achieved within restricted domains, such as aerospace engineering [27] and telecommunications software [25], domain analysis is invariably conducted within one organisation so that transfer of components between domains is difficult. As a result a gap exists between domain analysis which create libraries of components specific to an applications area and organisation on one hand and general specification languages (e.g. pragmatic languages such as entity-relationship diagrams or formal as in Z) on the other. To empower reuse we propose that bridging models are required at a level of abstraction between specific domains and abstract languages. Such models, we argue, are a necessary part of a reuse library architecture, as a classification framework and furthermore, as the basis for a retrieval mechanism for reusable specifications, designs and code.

Notable characteristics of successful reuse libraries are that they are small (circa 100-250 components) and contain components at what may be termed the design level of abstraction. To increase the size of libraries and to push reuse from the design level towards specification and even requirements, new approaches to component retrieval are required. Faceted classification schemes have proved useful, however, they suffer from limitations in the fidelity of terminology and discipline in use [25]. The bridging models we propose are intended to advance both the storage architecture and retrieval mechanism of large scale reuse libraries. The concept of bridging models is not new. Indeed a variety of models at varying levels of abstraction have been proposed, for example clichés [28], patterns [13, 30] and generalised application frames [3], to cite but a few. However, most reports have failed to provide more than a few examples of abstract models. No comprehensive model set or even modeling framework has been proposed to cover even several different types of application.

This paper is part of our ongoing work on analogy and specification reuse [21, 32]. It forms part of the domain theory for requirements engineering and reuse which is being developed in the ESPRIT 6353 'Nature' project [16]. The domain theory consists of three components: (i) a knowledge meta-schema defining the modeling language for bridging models; (ii) a set of domain abstractions which act as generic classes for specific application instances; (iii) a computational matching process, based on cognitive and artificial intelligence theories of analogical reasoning, that enables retrieval of appropriate models for facts describing a specific problem domain.

The matching process has been reported elsewhere [23], so this paper will concentrate on the first two components of the domain theory, although a description of its operational use in matching is covered.

The paper is organised in five sections. First, the modeling framework is outlined, then the meta-schema is introduced. This is followed by description of the domain abstractions. Retrieval of the abstractions in the matching process is described and this is elaborated to