Domain Knowledge Reuse During Requirements Engineering

Michael D. Gibson and Kevin Conheaney

**School of Computing and Management Sciences,
Sheffield Hallam University,
100 Napier Street, Sheffield S11 8HD, United Kingdom.**

Abstract. The accurate capture, understanding and representation of requirements is a critical step in the construction of effective and usable information systems. This activity is highly cognitive and is people and knowledge intensive; its success or failure is significantly influenced by the skills and especially the previous experience of the development staff involved. Many of the processes of requirements engineering are not well defined and remain difficult and problematic, significantly the initial investigation, elicitation and checking of requirements are poorly supported by current development tools and development methods. This paper describes research within the field of requirements engineering which draws upon AI/KBS research to adopt a knowledge based approach for requirements capture, modelling and validation and encourage the reuse of knowledge about typical information system applications and domains during such activities.

1. Introduction

The nature of modern information systems is changing, systems are becoming more complex and sophisticated; specialised information systems supporting decision making, systems which integrate organisational activities serving business processes rather than business areas and multiple organisation systems for new initiatives such as supply chain networks and virtual organisations. These developments increase the complexity and importance of accurately capturing and specifying the context and requirements of such systems. The incorrect specification of requirements remains one of the principle causes of inadequate information systems; poor requirement definition leading to systems which are ineffective, inefficient or do not satisfy user needs. There are numerous information systems development methodologies in existence, many supporting some aspect of requirements specification. These methodologies differ significantly in the development tasks which they address or the types of application for which they are suitable; the relative merits of such methods are widely documented [22, 9, 10]. A common feature of many of the commercially successful approaches is the increasing use of computer based tools to provide support for using the method or automate aspects of the development process. These development aids range from simple tools manipulating graphical structures

associated with a method, to more sophisticated tools, integrating graphical input with a central development database, performing varying degrees of automated consistency and completeness checking. Recent work in this area has introduced "intelligent" tools which automate development activities or offer guidance to users about performing development tasks, or the heuristics of a particular development method.

The extent to which current development methods and tools are actually addressing the special challenges of the requirements engineering stage of development remains uncertain. Compared to other development stages, the requirements stage is especially difficult; many of the activities are cognitive and the information available to the developer is often unstructured, imprecise, incomplete, or may reflect fluctuating or conflicting user views of system requirements. A knowledge gap exists between user and developer: the developer lacks detailed application knowledge whilst the user lacks an understanding of the computer system development process and terminology. The existence of a knowledge gap can and does lead to communication difficulties between user and developer during the investigation of requirements and importantly during the validation of captured requirements. In an empirical study of how analysts work [12], significant amounts of informal fact gathering and elicitation activity took place unsupported by any sort of method, in a further study [11] over half of the problems cited by analysts were associated with user communication; validation activities were considered to be especially problematic. Another survey [21] observed that users found it difficult to understand and validate requirements and that the use of existing structured notations had little impact in this area. Recent work in the area continues to provide evidence that many development methods and tools do not adequately serve the requirement specification phase of development and reviews of current research indicate that there are many unfulfilled opportunities for further requirements engineering research [5].

This paper describes research which aims to represent knowledge about typical applications and domains and promote the reuse of this knowledge during requirement elicitation, modelling and validation activities. The research is an extension to an existing knowledge based requirements modelling language which was developed to model and validate information system requirements. The remaining sections of this paper review the foundations and aims of the research, introduce the requirement modelling language, explain how this is used to represent typical domain knowledge and how such knowledge might be used in conjunction with computer based tools. The final section of the paper reflects on initial research findings and outlines plans for further work in this area.

2. Research Background

A number of authors have proposed that the representation of many kinds of knowledge about the world is essential to requirements engineering [14, 1, 7]. These works suggest that a requirement specification should represent a wide range of knowledge: domain knowledge about the target system and its environment,

development method knowledge and more general common sense or world knowledge. This view has generated interest in knowledge based representations such as semantic networks, frames and classical logic's as bases for rich modelling and inferencing mechanisms to represent and reason with various types of requirement knowledge.

A premise of our research is that much of the knowledge of interest to requirements specification can be identified, generalised and reused. Studies of how analysts actually work suggest that analysts extensively reuse knowledge about similar or previous applications they have worked on, using common or typical models as starting points and templates for considering the new situation under investigation [12]. It is common for analysts to use various forms of informal static models and animated examples both for their own benefit and during communication with end users. The use of analogy is also recognised by other works proposing the reuse of requirement knowledge in various forms [24, 16, 23, 24, 17, 19, 20]. The importance of analogy is reflected in range of studies of analysts productivity [3, 27] which confirm that previous experience significantly influences developer effectiveness and efficiency and the quality of the work they produce. Initial research in this area indicates that it is possible to identify types of information systems which share common properties and support similar problem domains and to classify these; an example of a partial type hierarchy is shown in Figure 1.

Figure 1. Partial classification of system types and uses

Within this hierarchy of system types there are characteristic objects, domain structure, and behaviour which typifies each type of application and are similar between supertype and subtype. Using the example of a loan control system it is possible to specify a generic description for such a domain, part of which might be:

" a loan object exists, it is usual to lend the object to a hirer or borrower for a period of time and the object is returned at some time. At any point in time the object can be available for loan, can be on loan or under repair and so on..... "

Even from this brief description it is possible to identify and classify domain concepts which typify this type of application:

- common objects such as: *loan object, borrower, hirer, loan period.*

- common states such as: *object available, object on loan, object at repair.*
- common events such as: *lend object, return object.*

These characteristics apply in specialised forms to many types of loan control systems such as the ones shown in the Figure 1 and many others. An aim of this research is to represent such generic knowledge in greater detail, showing common objects, common behaviour, common domain constraints and the dependencies between all of these. Representing the essential structures and common behaviour of target domains will offer guidance to the developer during elicitation, support modelling by analogy and also facilitate further work into the automation of, or active support for requirement model development activities by suitable tools.

The following sections describe a language which is being used to model reusable domain knowledge and show examples of the range of domain knowledge that can be represented.

3. Requirement Knowledge Modelling

An existing Requirement Fact Modelling Language (RFML) [13] is being adapted to specifically represent appropriate aspects of reusable domain knowledge. The RFML was derived from a study of conceptual graph theory, a knowledge representation formalism developed and described comprehensively by Sowa [26] and from work on an earlier conceptual graph based modelling language which formed part of the Alvey Analyst Assist SE/IKBS project [18]. The RFML uses a constrained form of conceptual graph theory within a rule based state transitional framework to model requirement knowledge. The language provides a set of primitive typed concepts and conceptual relations which are used to represent elements of domain knowledge. More meaningful knowledge representation is achieved by combining concepts and conceptual relations to form assertions and by combining assertions to form propositions. An additional knowledge structure, a scenario is introduced by the language. Scenarios are a specialised form of proposition, employing a traditional antecedent consequent rule structure; rule pre-conditions and post-conditions are represented by combinations of concepts, assertions and propositions.

3.1 Language Overview

The basic components of the language are concepts and conceptual relations. A concept is some element of interest from a modelled domain. A conceptual relation represents a directed semantic link between concepts. An assertion is a collection of concepts and conceptual relations which exist between the concepts; forming a directed bipartite graph, representing some aspect of a modelled domain. The following is an example of a simple assertion, introducing the text based linear notation used throughout this paper:

[Person:*] -> (CHAR) -> [Name:*)

The linear notation represents:

- * Concepts - as square brackets containing a concept label and referent value.

- * Relations - as simple brackets containing a relation label.
- * Arcs - shown as a dash and direction arrow.

Concepts are the items of interest from the modelled domain, the item can be real or abstract. Concepts are considered to be of certain types and these types may be classified in a concept type hierarchy. The RFML provides a set of primitive concept types from which domain specific concepts can be specified, some of these are shown in Figure 2.

Figure 2. RFML Primitive Concepts

A concept is represented by a type label and an optional referent value. The label indicates the concept type class and the referent serves as a symbolic identifier or value for an instance or instances of the class. The following examples show some of the concept/referent combinations used by the RFML which appear in the examples given in this paper, a full treatment of referent identifiers is to be found in other work [13].

[Member] - represents the concept class, the highest level of the concept type Member, representing the set of all members known to the model.

[Member:{#}] - represents a specific set of instances of Member; the set is unique within a specific context; the size of the set may be specified or constrained as in:

[Member:{#}@integer]

[Member:*] - represents a specific instance of Member which can be symbolically identified and is unique within a specific context.

[Member:Jones] - represents a specific instance of Member which is symbolically identified and is unique within a specific context.

[Member:#] - represents an individual instance of Member which is unique within a specific context but is not necessarily identified; used to denote the indefinite article a Member or the Member.

The combination of type label and referent allows considerable freedom to specify domain knowledge at various levels of generality and detail, allowing the modelling of a domain at a number of resolution levels appropriate to the type of application and the nature of the domain.

Conceptual Relations are used to specify semantic relationships between concepts. In conceptual graph theory, the treatment of conceptual relations is derived from linguistic research into case roles and is based on Fillmore's case grammar [8]. The RFML follows this approach and provides a set of primitive relations each of which is defined by a formal definition showing the highest level concepts between which the relation may apply. Negated relations are supported and new relations can be defined by specialisation, all relations are classified in a type hierarchy in the same way as concepts.

Concepts and relations are combined to form simple assertions which in turn can be combined to form more comprehensive and more expressive knowledge representational structures. Formal rules determine the combination and manipulation of assertions [26, 13]. The following sections introduce the knowledge representation structures used by the RFML.

3.2 Language Structures

Psychologists have for some time emphasised the importance of chunks of knowledge as elementary patterns in human perception and thinking: a common approach which humans use to structure knowledge is in terms of set situation action rules [15]. The RFML uses assertions, propositions and rule based scenarios, to model elements of domain structure and domain behaviour.

Assertions are an association between at least one conceptual relation and two or more concepts. Assertions may describe knowledge relating to the behaviour or structure of a domain. The following are examples of a number of basic assertions from a domain:

```
[Book:*]->(CHAR)->[Title:*]. [MicroficheSystem:#]->(PART)->[FicheReader:({#})@4].
[Member:*]->(CHAR)->[UserNumber:*]. [ReadingDesk:({#})]->(LOC)->[ReadingRoom:#].
```

Propositions are packages of knowledge containing a number of related assertions or they may contain further nested propositions. Nested propositions may be connected to represent conjunctive, disjunctive or negated combinations of knowledge. A proposition is represented in the RFML as:

```
[[Proposition]] or alternatively as : (PROPREL)->[[Proposition]].
```

The following example shows a propositional declaration for a Library domain:

```
[[[Member:*]->(CHAR)->[AccessLevel:Staff]]->(OR)->
[[Member:*]->(CHAR)->[AccessLevel:Student]]->(OR)->
[[Member:*]->(CHAR)->[AccessLevel:Limited]]->(OR)->
[[Member:*]->(CHAR)->[AccessLevel:Visiting]]].
```

Within the RFML, propositions may be specialised to form context, behaviour or constraint scenarios, as shown in Figure 3.

Figure 3. Proposition Subtypes

Scenarios are packages of knowledge linked to specific concept types. Scenarios are used to represent structural, contextual, behavioural and constraint declarations describing a modelled domain. A scenario is a specialised rule based form of proposition, where the proposition consists of two component propositions linked by a causal relation. A scenario rule structure consists of a left hand side antecedent proposition defining the necessary conditions which must exist before an instance of the concept type associated with the scenario can come into existence. The right hand side defines a consequent proposition showing the necessary conditions which must exist after the concept instance is introduced. Scenarios have the following representation:

```

[Concept]->(PROPREL)->
[[[Proposition]]
->(RESLT)->
[[[Proposition]]].
  
```

4. Representing Domain Knowledge

The section on research aims gave an overview of the classification of common application domains and how concepts typical of these domains can be identified and typed. This sections gives examples of how it is possible to represent further knowledge about a domain, defining concepts, how concepts are usually related and to specifically represent knowledge about domain structure, behaviour, rules and the interaction between these.

4.1 Modelling Domain Contents

The typical structure and contents of a domain are represented by a number of assertions. These are used to specify typical domain concepts and common relationships between such concepts. They model the hierarchical ordering of concept types, relevant instances of types which may exist and the relations which may link concepts of various types at different levels of generality. For each concept typical of a domain, a cluster of assertions is specified for that concept showing all

the important relations it may hold with other concepts in the same domain, some examples for a loan object might be:

```
[LoanObject:*]->(CHAR)->[Location:#].    [LoanObject:*]<-(OBJ)<-[ReturnObject:#].
[LoanObject:*]->(CHAR)->[HireRate:#].    [LoanObject:*]<-(OBJ)<-[LendObject:#].
[LoanObject:*]->(STAT)->[OnLoan:#].      [LoanObject:*]->(STAT)->[Available:#].
```

Assertions represent knowledge about an application domain at a particular level of generality and resolution, they can be successively specialised, refined and extended to model appropriate examples of the application type. The following shows examples of assertion specialisation:

Inherent RFML model assertion:

```
[ANIMATE:*]<-(AGNT)<-[EVENT:#]->(OBJ)->[OBJECT:*].
```

Specialising the above to any loan domain:

```
[ANIMATE]->(SUB)->[Borrower].
[EVENT]->(SUB)->[LendObject].
[OBJECT]->(SUB)->[LoanObject].
[Borrower:*]<-(AGNT)<-[LendObject:#]->(OBJ)->[LoanObject:*].
```

and after further specialisation to a Library domain:

```
[Borrower]->(SUB)->[Member].
[LendObject]->(SUB)->[IssueBook].
[LoanObject]->(SUB)->[Book].
[Member:*]<-(AGNT)<-[IssueBook:#]->(OBJ)->[Book:*].
```

These simple assertions describe aspects of the structure of a domain and serve as building blocks from which more informative assertions and propositions describing the domain can be constructed. They also provide the material for the specification of scenarios to represent typical domain behaviour and important domain rules and constraints.

4.2 Modelling Domain Behaviour

Behaviour scenarios specify units of typical behaviour in a domain, constraints on when such behaviour may take place and the consequences of the behaviour. For each domain event a behaviour scenario models the concepts and relations between concepts which must exist prior to the event and the concepts and relations between concepts which must exist after the event has taken place. The following example shows a simple behaviour scenario for the event 'Lend Object' in a generic loan control domain:

```
[LendObject:#]->(BEHVR)->
[[[Borrower:*]<-(AGNT)<-[LendObject:#]->(OBJ)->[LoanObject:*]]->(AND)->
[[Borrower:*]->(STAT)->[ValidUser:#]]->(AND)->
[[LoanObject:*]->(STAT)->[Available:#]]]
->(RESLT)->
[[[LoanObject:*]->(ALLOC)->[Borrower:*]]->(AND)->
[[LoanObject:*]->(STAT)->[OnLoan:#]]].
```


Each behaviour scenario describes a unit of behaviour and sequences of behaviour scenarios may be used to describe threads of typical domain behaviour. The collection of all behaviour scenarios in the model represent the range of possible behaviour in a domain.

Context scenarios define the states which exist within a domain, further constraints on relationships between states and for each state the concepts and relations between concepts which must exist during that state. A state context scenario describes additional conditions which apply to those concepts related to the state whilst the state exists. The following shows an example of a context scenario associated with a loan control system state 'Available', note the use of negated relations:

```
[Available:#]->(CONTX)->
[[[LoanObject:*]->(STAT)->[Available:#]]
->(RESLT)->
[[[LoanObject:*]->(STAT)->[OnLoan:#]]->(AND)->
[[LoanObject:*]->(STAT)->[AtRepair:#]]].
```

Context scenarios are useful to specify additional structural constraints relevant to a domain and further define the permissible consequences of domain behaviour.

4.3 Modelling Domain Constraints and Rules

In many domains there are application specific rules or general constraints or rules describing the structural aspects of a domain. Constraint scenarios are used to define restrictions on concept type populations, relations between concepts, operational properties of the domain such as integrity constraints, application specific rules and rules for deriving additional information from known information.

Constraint scenarios specify constraints on specific objects; expressed by relating an object to an appropriate constraint scenario. The scenario defines a pattern of concepts and relations showing conditions which must exist (or cannot exist) in the domain if an instance of the constrained object exists in the domain. The post-condition of the constraint scenario contains an assertion or assertions which indicate the possible effect on the domain if the constraint pre-condition is satisfied. The following scenario introduces the object 'Loan Duration' as relevant to a loan control domain and possible characteristics of the concept:

```
[LoanDuration:*]->(CONST)->
[[[Borrower:*]<-(AGNT)<-[LendObject:#]->(OBJ)->[LoanObject:*]]->(AND)->
[[[LoanObject:*]->(STAT)->[OnLoan:#]]->(AND)->
[[[OnLoan:#]->(DURN)->[LoanPeriod:*]]->(AND)->
[[LoanDuration:*]->(GTR)->[LoanPeriod:*]]
->(RESLT)->
[[[LoanObject:*]->(STAT)->[Overdue:#]]->(AND)->
[[Borrower:*]->(STAT)->[OverdueBorrower:#]]].
```

The examples of domain knowledge presented in this section have been simplified and represent modelling at a very general level in the application type hierarchy. It is when these knowledge structures are specialised and extended to represent specific application domain examples that they become more meaningful and useful; the following section shows how such structures can be interpreted and used during requirement specification activities.

5. Using Domain Knowledge

An objective of this research is to model each level of a specified application type hierarchy by appropriate assertions, propositions and scenarios. Each of these declaratively describes an aspect of the application type and collectively they form a typical/generic model for that type of application and domain. Using declarations allows a modular approach to specifying domain knowledge and provides a number of useful abstraction levels to view the domain model from different perspectives.

Individually the declarations can serve as templates and prompts to the developer about aspects of an application type when considering a new development or they can be used as communication examples in user analyst dialogues for elicitation. For example, from the generic loan control event 'LendObject' described in a previous section:

```
[LendObject:#]->(BEHVR)->
[[[Borrower:*]<-(AGNT)<- [LendObject:#]->(OBJ)-> [LoanObject:*]]->(AND)->
[[Borrower:*]->(STAT)-> [ValidUser:#]]->(AND)->
[[LoanObject:*]->(STAT)-> [Available:#]]
->(RESLT)->
[[[LoanObject:*]->(ALLOC)-> [Borrower:*]]->(AND)->
[[LoanObject:*]->(STAT)-> [OnLoan:#]]].
```

To specialise this to a library domain generates a number of possible questions which may need to be addressed, examples are: *what is the loan object involved? what is the borrower called? when is a loan object considered available? what is a valid user in the domain? how is the allocation of object to borrower recorded?*

Addressing such questions can guide elicitation activities and the further study of existing domain descriptions and documentation or they can provide a framework for developer-user communication. The following example shows an interpretation of the original scenario specialised for the event 'IssueBook' in a library domain:

```
[IssueBook:#]->(BEHVR)->
[[[Member:*]->(AGNT)-> [BorrowBook:#]->(OBJ)-> [Book:*]]>(AND)->
[[Member:*]->(CHAR)-> [Loans:{#}@<6]]->(AND)->
[[Book:*]->(STAT)-> [BookAvailable:#]]->(AND)->
[[Book:*]->(~STAT)-> [BookReserved:#]]
->(RESLT)->
[[[Book:*]->(STAT)-> [BookOnLoan:#]]->(AND)->
[[Member:*]->(CHAR)-> [Loans:{#}]<-(INCL)<[Book:*]]].
```

A central theme of our approach is that a new requirement model is developed by specialising, refining or adapting selected parts of the relevant domain model if this is appropriate. Additionally, the existing knowledge representation structures can be used by developers as patterns for modelling their own interpretation of the new domain. Currently these activities are analyst directed, an aim of future research is to provide intelligent tool support to guide and monitor domain knowledge reuse activities. The ability of the RFML to represent many of the domain declarations in a rule based format provides the basis for the operational interpretation of the language by suitable tools. This allows tool support for assisting with completeness and consistency checking and for executing the model to assist in the understanding and validation of modelled knowledge. By selecting and executing behaviour scenarios, the developer can experiment with domain models and the evolving requirement model. Typical or alternative patterns of domain behaviour can be tested; examining the ordering and dependencies of domain events and states and the validity of resulting behaviour on modelled domain rules and constraints.

6. Tool Support for the Approach

At present, tool support for the approach is based on an existing RFML workbench which was implemented to demonstrate the use of the language for building a requirement model. The RFML workbench is implemented using an extended version of Smalltalk/V Windows Release 2.0 [6] on a 486 PC. The Smalltalk environment has been modified to include an extended Prolog interpreter which provides a partial implementation of the Prolog language [2]. The Prolog facilities provide a foundation for implementing the workbench knowledge bases and appropriate knowledge inference mechanisms. Figure 4 provides an overview of the workbench architecture and functionality.

Figure 4. The Architecture of the RFML Workbench

The common representation of domain knowledge and requirement knowledge allows exploitation by common tools, the developer is able to browse a populated domain knowledge model alongside the building and population of a new requirement model. This encourages modelling by analogy, allowing the study of typical domain concepts, domain behaviour and structure and experimenting with this; providing templates of domain knowledge from which to specify application specific models. Work is ongoing to specify how to browse generic models to guide elicitation and model building activities and guidance on execution and experimentation with behaviour scenarios.

7. Conclusions and Further Work

The research described here is in its early stages, but has already given useful insight into the potential of domain knowledge reuse, especially for initial fact gathering and elicitation activities which at present are poorly supported.

One of the first and important steps in requirements engineering is bridging the gap between informally stated requirements and a model containing structured and formalised requirements. The reuse of domain knowledge and development by analogy appear to be a useful and supportable way of making this step. There are significant similarities between common types of business information systems, our research to date indicates it is possible to classify and define these types. By providing generic models of application types, and appropriate support tools, direct support can be given to requirement elicitation and modelling by refinement and adaptation rather than reinvention. Further work is required to establish the extent to which information systems can actually be classified and generalised and the appropriate formats of effective generic models. The approach also needs to be tested to evaluate the usefulness or limitations of generic models when applied to real life examples of specialisation's of the application type.

A research aim for further work on tool support is to investigate and develop intelligent tools to exploit the domain knowledge reuse approach. A set of RFML development heuristics have been specified [13]; these in conjunction with the formality of the language, the contents of a part populated requirement knowledge base and appropriate domain knowledge base give a foundation for developing active support tools for a number of model development activities:

- requirement elicitation activities,
- selection and explanation of appropriate domain knowledge
- model building guidance and critiquing
- requirement traceability and decision justification
- automation of completeness/consistency checking and conflict resolution.
- validation guidance tools

Finally one of the important benefits of the research has been the appreciation that once a populated and validated RFML model is derived, this in turn can serve as a long term knowledge base of user requirements and application domain knowledge,

providing a foundation for later stages of development and a reference point for revision and maintenance of the system after implementation. This also introduces the possibility of reusing such a model at a future date for system updates or as an additional source of domain knowledge to support the development of similar applications or other applications in the same domain.

References

1. A. T. Borgida, J. Mylopoulos: "Knowledge Representation as the basis of Requirements Specification". *COMPUTER*, Vol. 18, No 4, April 1985.
2. W. F. Clocksin, C. S. Mellish: "*Programming in Prolog*". Springer-Verlag, Berlin, 1984.
3. B. Curtis, H. Krasner, N. Iscoe: "A Field Study of the Software Design Process for Large Systems". *Communications of the ACM*, Vol. 31, No 11, pp 1268-1287, November 1988
4. A. J. Czuchry: "Where's the Intelligence in the Intelligent Assistant for Requirements Analysis". *Proceedings Second Annual Knowledge-Based Software Assistant Conference*, Published by Rome Air Development Center, NY 13441-5700, January 1988.
5. A. M. Davis, P. A. Freeman: "Guest Editors Introduction: Requirements Engineering". *IEEE Transactions on Software Engineering*, Vol. 17, No 3, pp 210-211, March 1992.
6. DIGITALK Inc. "*Smalltalk/V Object-Oriented Programming System (OOPS) Tutorial and Programming Handbook - Windows Series*", Digitalk, California, USA, 1992.
7. E. D. Falkenberg, H. Van Kempen, N. Mimpen: "Knowledge-Based Information Analysis Support". in *Proceedings of IFIP TC2/TC8/WG 2.6/WG 8.1 Artificial Intelligence in Databases and Information Systems*, Guangzhou, China, 1990.
8. C. J. Fillmore: "The Case for Case". in *Universals in Linguistic Theory*, E. Bach, R. T. Harms (Eds.), Holt Rhinehart and Winston Inc, 1968.
9. G. Fitzgerald, N. Stokes, J.R.G. Wood: "Feature Analysis of Contemporary Information Systems Methodologies". *The Computer Journal*, Vol. 28, No 3, pp 223-230, 1985.
10. C. Floyd: "A Comparative Evaluation Of System Development Methods", in [Olle et al 86], 1986.
11. D. J. Flynn, R. Warhurst, M. D. Gibson, D. Browne, R. Summersgill: "An Empirical Study into the Process of Validating a Specification". *Proceedings 2nd International Conference on Information Systems Development Workbenches*, Gdansk, Poland, pp 401-420, September 1990.
12. M. D. Gibson, C. Harthhoorn: "*The use of JSD*". Analyst Assist Project Research Report AA-U0010, UMIST, Manchester, UK, 1987.
13. M. D. Gibson: "RFML: An Executable Requirement Specification Language for Information System Development", PhD Thesis, Faculty of Technology, UMIST, Manchester, England, 1992.

14. S. J. Greenspan, J. Mylopoulos, A. Borgida: "Capturing More World Knowledge in the Requirements Specification". *Proceedings of 6th International Conference on Software Engineering*, England, UK, pp 225-234, September 1982.
15. F. Hayes-Roth: "RULE-BASED SYSTEMS". *Communications of the ACM*, Vol. 28, No 29, pp 921-932, September 1985.
16. D. R. Harris: "An Overview of the Knowledge-Based Requirements Assistant". *Proceedings Second Annual Knowledge-Based Software Assistant Conference*, Published by Rome Air Development Center, NY 13441-5700, USA, January 1988.
17. M. Lloyd-Williams: "Knowledge-based CASE tools: improving performance using domain specific knowledge". *Software Engineering Journal*, pp167-173 July 1994.
18. P. Loucopoulos, P. J. Layzell, R. E. M. Champion, M. D. Gibson: "A Knowledge-Based Requirements Engineering Support Environment". *Proceedings of the 3rd Annual Knowledge-Based Software Assistant Conference*, Utica, NY, USA, pp 139-154. 1989.
19. N. A. M. Maiden: "Analogical Specification Reuse During Requirements Analysis", PhD Thesis, Department of Business Computing, City University, London, July 1992.
20. N. A. M. Maiden, A. G. Sutcliffe: "Requirements engineering by example: An empirical study", *Proceedings of IEEE symposium on Requirements Engineering*, IEEE Computer Society Press, pp 104-112, 1993
21. J. T. Nosek, R. B. Schwartz: "User Validation of Information System Requirements: Some Empirical Results". *IEEE Transactions on Software Engineering*, Vol. 14, No 9, pp 1372-1375, September 1988.
22. T. W. Olle, H. G. Sol, A.A. Verrijn-Stuart (Eds.): "*Information Systems Design Methodologies: improving the practice*". North-Holland Publishing Company, IFIP, 1986.
23. P. P. Puncello, F. Pietri, P. Torrigiani: "ASPIS: A KNOWLEDGE-BASED ENVIRONMENT FOR SOFTWARE DEVELOPMENT". *ESPRIT Achievements and Impact*, North-Holland, pp 375-391, 1987.
24. C. Rich, R. C. Waters, H. B. Rubenstein: "Towards a Requirements Apprentice". *Proceedings of the 4th International Workshop on Software Specification and Design*, London, 1987.
25. H. B. Rubenstein, R. C. Waters: "The Requirements Apprentice: Automated Assistance for Requirements Acquisition". *IEEE Transactions on Software Engineering*, Vol. 17, No 3, pp 226-240, March 1991.
26. J. F. Sowa: "*Conceptual Structures: Information Processing in Mind and Machine*". Addison-Wesley, Reading, MA, USA, 1984.
27. G. M. Wijers, H. Heijes: "Automated Support for the Modelling Process". in *Advanced Information Systems Engineering Second Nordic Conference CAiSE 90*, Sweden May 90, Springer Verlag, New York, pp 88-108, 1991.