Knowledge-Based Support for Requirements Engineering

P. Loucopoulos and R.E.M. Champion

**Department of Computation,
UMIST,
P.O. Box 88,
Manchester, M60 1QD,
United Kingdom,
Tel. 061 236 3311.**

ABSTRACT

The accurate capture and representation of user requirements plays a critical role in the construction of effective and flexible information systems. However, despite the introduction of development methods and CASE tools in the project life-cycle, the process of developing a requirements specification remains problematical.

This paper proposes that future development in CASE environments should provide facilities which more closely match the activies of expert system analysts. These activities include: the creation of informal models and scenarios about the modelled domain prior to the formalisation of captured concepts in a schema according to the chosen development method's model; extensive use of domain knowledge; use of method specific knowledge; consideration of multiple views about the modelled domain; the creation of hierarchies of concepts; formulation of hypotheses about modelled structures; and resolution of different hypotheses and decision formulation. To this end, the paper reports on a prototype system which provides facilities for the support of these activities by exploiting knowledge-based techniques for the capture of concepts about an application domain and their specification in JSD constructs.

ACKNOWLEDGEMENT

The work reported in this paper was conducted as part of the Analyst Assist project. This is a collaborative project involving Data Logic, UMIST, Scicon, MJSL, MOD(ARE) and Istel. The work undertaken by the authors of this paper is supported by the UK Science and Engineering Research Council Grant No. GR/D 72648 - SE/113 as part of the Alvey Programme of Research and Development.

1. Introduction

In recent years there has been a growing realisation that the development of large information systems is becoming increasingly more difficult as user requirements become broader and more sophisticated. The complexity which is exhibited by most contemporary application domains and the subsequent problem of developing automated information systems for these domains has proved that the traditional, informal approach is no longer feasible since the gap, between initial requirements statement and the verification of these requirements in terms of the implemented system, is too large. An alternative to the traditional approach has been the adoption of paradigms which attempt to establish appropriate management procedures within a systematic framework which recognises well identified tasks and milestones. Examples of such methods are Information Engineering [MacDonald, 1986], JSD [Jackson, 1983], NIAM [Verheijen & van Bekkum, 1982], SADT [Ross, 1977], SASD [DeMarco, 1978], STRADIS [Gane & Sarson, 1979] and a plethora of other more or less similar methods (cf [Layzell & Loucopoulos, 1987] for a bibliography).

These proprietory methods and their associated CASE tools pay particular attention to the construction of a high level specification of a system before the implementation of software. Emphasis is placed in establishing a clear understanding of the functional requirements of the information system, since it has often been reported that over 50% of software malfunctions have their origin in the process of requirements capture, the effect however not being detected until later stages thus giving rise to a figure of 80% of personnel resources being committed to the debugging and testing of source code [Chapin, 1979, van Assche et al, 1988].

Despite the increasing use of these methods, however, users continue to experience major difficulties [Morris, 1985]. From the viewpoint of requirements specification, a major criticism levelled at contemporary approaches is their poor handling of the capturing and modelling of the knowledge of the universe of discourse [Greenspan, 1984; Balzer et al, 1983; van Assche et al, 1988]. These shortcomings can be attributed partly to the inherent nature of the *process* of capturing, modelling and verifying concepts of the modelled domain and partly to inadequate *formalisms* used for the representation of these concepts.

This paper argues that future development methods and CASE environments must support the early stages of requirements specification by providing modelling formalisms and functions which permit the development of *informal models* and *experimentation* prior to developing a specification according to the constructs of the chosen method. To this end, the paper reports on a prototype system which is used as part of a larger CASE environment in the context of the Jackson System Development (JSD) method [Jackson, 1983]. Section 2 discusses the process of capturing concepts of an application domain and derives a set of desirable features of a requirements engineering support system. Section 3 introduces the prototype system in terms of its underlying formalism and functions and section 4 describes the way the system may be exploited in terms of the system's three main facilities of concept elicitation, concept resolution and decision tracing.

2. Requirements Specification

A requirements specification represents both a model of what is needed and a statement of the problem under consideration [Rzepka and Ohno, 1985]. The activities involved are long and iterative, and involve much informality and uncertainty. Consequently, many of the problems associated with requirements specification can be attributed to the nature of the task itself. The following are key characteristics of the task of requirements specification [Vitalari & Dickson, 1983]:

- because the task is carried out at the early stages of the development lifecycle it is often difficult to define the boundaries of the universe of discourse in an exact way
- because there is little structure of the problem domain before hand there is a considerable degree of uncertainty about the nature and make-up of the possible solutions
- analysis problems are dynamic that is, they change while they are being solved because of their organisational context and the multiple participants involved in the definition and specification process
- solutions to analysis problems require interdisciplinary knowledge and skill
- the process of analysis itself, is primarily cognitive in nature, requiring the analyst to structure an abstract problem, process diverse information, and develop a logical and internally consistent set of models.

Within requirements specification two basic activities take place: modelling and analysis [Dubois et all, 1986]. Modelling refers to mapping real world phenomena into basic concepts of a requirements specification language. These basic concepts serve as the means of building various structures which, in their totality, constitute the requirements specification for a problem domain. Analysis refers to techniques used to facilitate communication between requirements engineers and end-users using the developed requirements specification as the basis of that communication.

Following this view, *requirements engineering* can be defined as the systematic process of developing requirements through an iterative process of analysing the problem, documenting the resulting observations and checking the accuracy of the understanding gained.

In order to manage this process there is a need for support in three areas. Firstly, an analyst's reasoning process must be guided by some underlying process which is appropriate to the task as well as the problem domain. Such an underlying process must be generic in nature, so that all analysts may use its constructs, and must represent a standard approach within an organisation, so that a specification generated by a development team is achieved in an integrated way. Secondly, facilities must be provided for locating information about an evolving specification. Many facts are gathered during the process of constructing a requirements specification and these facts must be correlated, irrelevant ones discarded and appropriate facts organised in meaningful structures. Thirdly, assistance is needed in the communication between analysts and end users during the phases of facts acquisition and specification verification. Capturing and verifying requirements are labour-intensive activities which demand skilful interchange between those that understand the problem domain and those that need to model the problem domain.

Contemporary approaches provide support in all three areas through the use of method steps, project encyclopedias, and diagramming tools. Certainly, these approaches are a major improvement on the ad-hoc traditional approach. However, further major benefits can only come about if the three areas of design discipline, documentation and communication are supplemented by support facilities which closely match the behaviour of expert analysts.

A number of empirical studies have been carried out in an attempt to better understand the process of developing a requirements specification. The differences in behaviour between high and low-rated analysts were investigated in three separate projects [Vitalari & Dickson, 1983; Fickas, 1987; Adelson & Soloway, 1985]. Based upon the results of these studies, it is possible to identify several major (not mutually exclusive) types of working practices by system developers, as follows:

a. Use of Analogy

Developers use information from the environment to classify problems and relate them to previous experience. On the other hand lack of information provides triggers to search for missing data. Empirical studies have shown that experienced developers begin by establishing a set of context questions and then proceed by considering alternatives. One basic prerequisite for using analogy during requirements elicitation is the knowledge that the analyst has about the domain under examination. Loucopoulos and Champion [1988] argue that such knowledge is crucial to the development of a requirements specification.

b. Hierarchies of Models

Expert developers tend to start solving a problem by forming a mental model of the problem at an abstract level. This model is then refined, by a progression of transformations, to a concrete model, as more information is obtained. Developers are aware of the various levels of policy within a domain and use this knowledge to guide a user during a requirements capture session.

c. Formulation of Hypotheses

Hypotheses are developed as to the nature of the solution, as information is collected. An experienced developer uses hypothetical examples to capture more facts from a user but also to clarify some previously acquired information about the object system. Experience in the application domain seems to be an important factor in formulating likely outcomes of the solution space.

d. Summarisation

Developers almost always summarise in order to verify their findings. It has been observed that during a typical user-analyst session the analyst will summarise two or three times and each time the summarisation will trigger a new set of questions. Summarisation may be used in order to clarify certain points from previous discussions or to encourage participants to contribute more information about the modelled domain.

e. Domain Knowledge

Expert analysts normally have a good knowledge of the concepts involved in the business processes of the organisation being investigated. Some concepts will be common to information systems, regardless of the underlying organisation. This common knowledge enables an expert analyst to approach a familiar analysis problem in a new domain with some expectations, which can be used to guide the investigation.

A further important finding of a study on the use of the JSD method which has implications on the use of development methods at the early stages of the project lifecycle is the use of informal models before any captured concepts are formulated according to the method's prescribed formalism [Gibson & Harthoorn, 1987]. The first step in the JSD method is the derivation of the entity stucture model which considers entities of the modelled domain and actions suffered by each entity. The investigation on the use of the method by experienced analysts revealed that, in their attempt to structure the problem domain and identify what are appropriate entities and actions for modelling, these analysts consistently used informal models (or even models of other methods with which they were familiar). As shown in figure 1, these informal models, were considered along with domain expertise and JSD method expertise in deriving a first-cut JSD entity model.

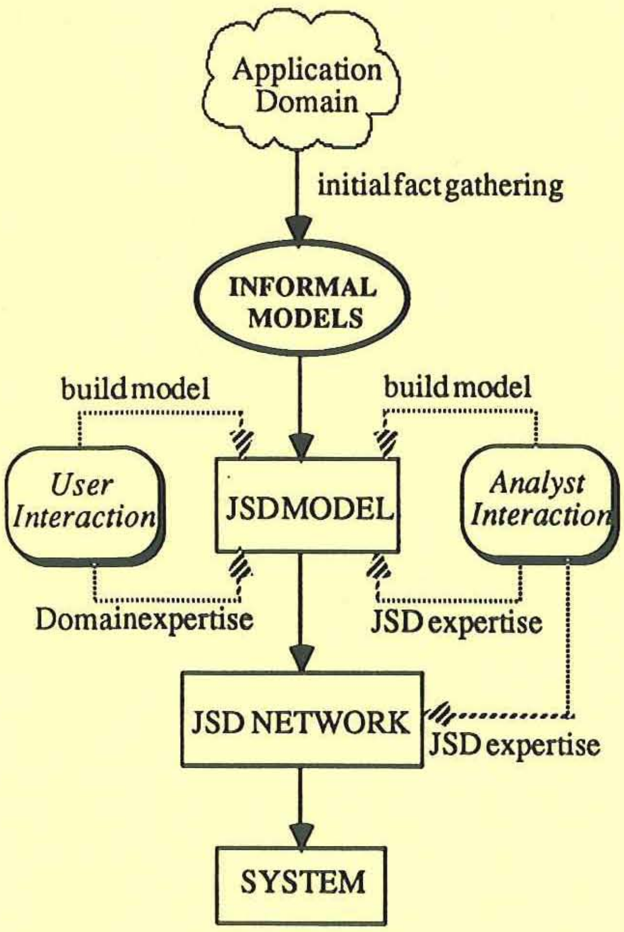


Figure 1: Working Method of Expert JSD Analysts

On the basis of the findings of these studies a prototype tool known as the Requirements Elicitation Support Tool (REST) has been developed in order to provide assistance at the very early stages of requirements capture. This tool has been developed within the context of the *Analyst Assist* system [Loucopoulos et al, 1988] and attempts to provide assistance in capturing informal requirements, specifying and documenting requirements using the Jackson System Development Method (JSD), and validating the specification by prototyping and animation. REST provides facilities which:

- encourage the development of informal models in the form of *scenarios* about the modelled domain
- maintain many different *views* about captured concepts
- enable the *tracing* of decisions taken by analysts about discarding or accepting particular scenarios
- assist the identification of candidate concepts by exploiting *domain knowledge* and
- assist in the mapping of informal models onto JSD model structures using JSD knowledge.

3. The Requirements Elicitation Support System

3.1 The System Architecture

An abstract architecture of REST is shown in figure 2.

The *user fact base* (UFB) is concerned with the storage of application dependent concepts *before* these concepts are interpreted in terms of JSD constructs. This database is populated using a *Fact Input Tool*. This tool is guided by an *Elicitation Dialogue Formulator* which makes use of the *domain knowledge base* and the current state of the UFB. It is feasible for the UFB to contain several different and possibly contradictory views of the concepts pertaining to the modelled domain. At any stage, one of these views is considered to represent the *current view*, that is the view which most closely reflects the analysts opinion about the modelled domain (but may be replaced at any point by any of the other views should the analyst's opinion is modified).

The *JSD specification* holds an evolving system specification in terms of JSD structures for the model and network stages of that method. This specification is developed via the REST (making use of the UFB) and with the assistance of two diagramming tools shown in figure 2 as the *JSD Input Tool* component.

The *JSD Method Advisor* acts as an assistant on the method steps of JSD and provides consistency

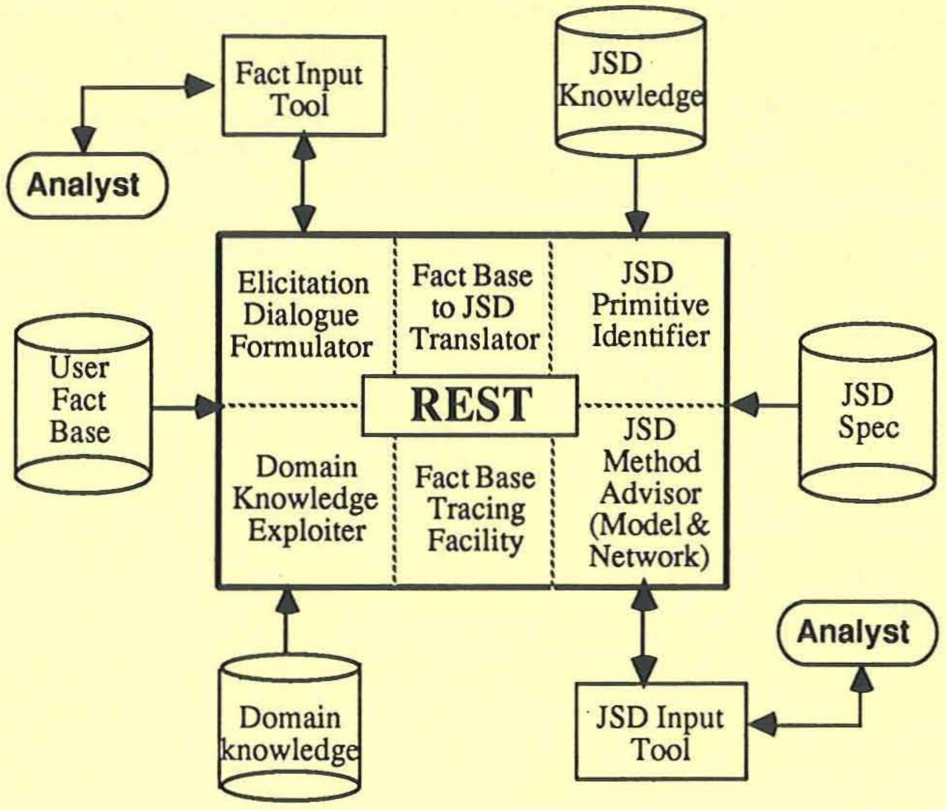

Fact Input Tool
JSD Knowledge
Analyst
Elicitation Dialogue Formulator
Fact Base to JSD Translator
JSD Primitive Identifier
User Fact Base
REST
JSD Spec
Domain Knowledge Exploiter
Fact Base Tracing Facility
JSD Method Advisor (Model & Network)
Analyst
Domain knowledge
JSD Input Tool

Figure 2: The Requirements Elicitation and Specification Tool Architecture

checking on the evolving JSD Specification. The *Fact Base Tracing Facility* is used to link a JSD specification to the concepts in the UFB from which it was derived and to update or annotate information held in it due to information input by the analyst using the *JSD Input Tool.*

Each item stored in the UFB or the JSD specification is linked to its source, in terms of the input session, date and analyst involved. This provides the capabability for tracing the history of the evolving models, and also allows for the summarisation of any subset of the specification.

3.2 The Model for Concept Representation

In order to simplify the system architecture, the knowledge bases and UFB share the same underlying representation. The formalism chosen is broadly based upon the use of *conceptual structures* [Sowa, 1984] for knowledge representation and reasoning. Whilst conceptual structures are often associated with the understanding of natural language, involving large semantic nets, comprehensive lexicons and precise grammar rules, it is important to emphasise that the use of conceptual graph theory within requirements elicitation and formalisation does not involve the same degree of complexity. The concepts and relationships employed can be limited to those which are relevant to the current domain of interest.

The formalism has been chosen for its flexibility and power of representation. Conceptual structures can be mapped onto first order logic statements, which allows for reasoning and consistency checks on the contents of the UFB. Summarisation of the UFB in English phrases is made possible by the relatively simple mapping of conceptual graphs onto the constructs of a natural language.

Conceptual graphs are finite, connected bipartite graphs, the two nodes of the bipartite graph being *concepts* and *conceptual relations.* Every conceptual relation has one or more *arcs*, each of which must be linked to some concept. The collection of all the relationships that concepts have to other concepts is called the *semantic net.* All conceptual graphs are linked to the semantic net, and so access to any graph is possible from any concept or relation.

The following components of conceptual graph theory have been implemented to provide a representation and reasoning medium for use by the *domain knowledge base* and *user fact base* which directly support the process of fact gathering.

- separate hierarchies of concept types and relation types which define the relationships between domain concepts and relations at different levels of generality
- prototype descriptions of each concept which can be linked to the concept type hierarchy
- formation rules to determine how each type of concept may be linked to conceptual relations and to other concepts
- conceptual graphs linked to each concept defining how the concept should be used in a domain (canonical graph) or how it may be used (scenario graph).

A detailed description of the conceptual structures employed in the model for REST is beyond the scope of this paper. The interested reader is referred to [Champion et al, 1988] for a description of the conceptual structures implemented in REST.

Requirements elicitation in REST is based on the premise that an analyst should be able to define any 'thing' of the modelled application domain as a potential concept of interest and that the final set of concepts can only be decided upon after careful consideration of various scenarios about the role that these concepts play. Therefore, 'concepts capturing and resolution' uses:

- The semantics of conceptual graphs. Concepts are placed in semantic networks with the top level concepts being pre-defined as those of ENTITY, EVENT, QUANTIFIER, VALUE, STATE and TIME. Conceptual relations are also classified by type. A hierarchy is defined over the type labels of the conceptual relations defined for the current domain.
- The domain knowledge base (DKB). The DKB may contain information about one or more concepts about the current application domain. This knowledge is used in automatically deriving conceptual graphs for these concepts and later on for their resolution.
- A natural language output facility. All information presented to the analyst during an input session and subsequent resolution, is shown in such a way as to hide the underlying formalism. Where possible, general rules for translating conceptual graphs into English are used; however, where more complicated representation is required for example, in the representation of constraints, these must be dealt with special purpose rules.

4. An Example Requirements Elicitation Session

4.1 Capture and Representation of Domain Specific Facts

The purpose of capturing facts about an application domain is to allow an analyst to reason with all information which is perceived to be relevant so that a system specification can be constructed. Because of the nature of the process of concept identification and capturing, it is advantageous to permit an analyst to carry out this process in a variety of ways. Following the discussion in section 2 about the underlying characteristics of the requirements elicitation process, REST provides two main support facilities for the identification of concepts:

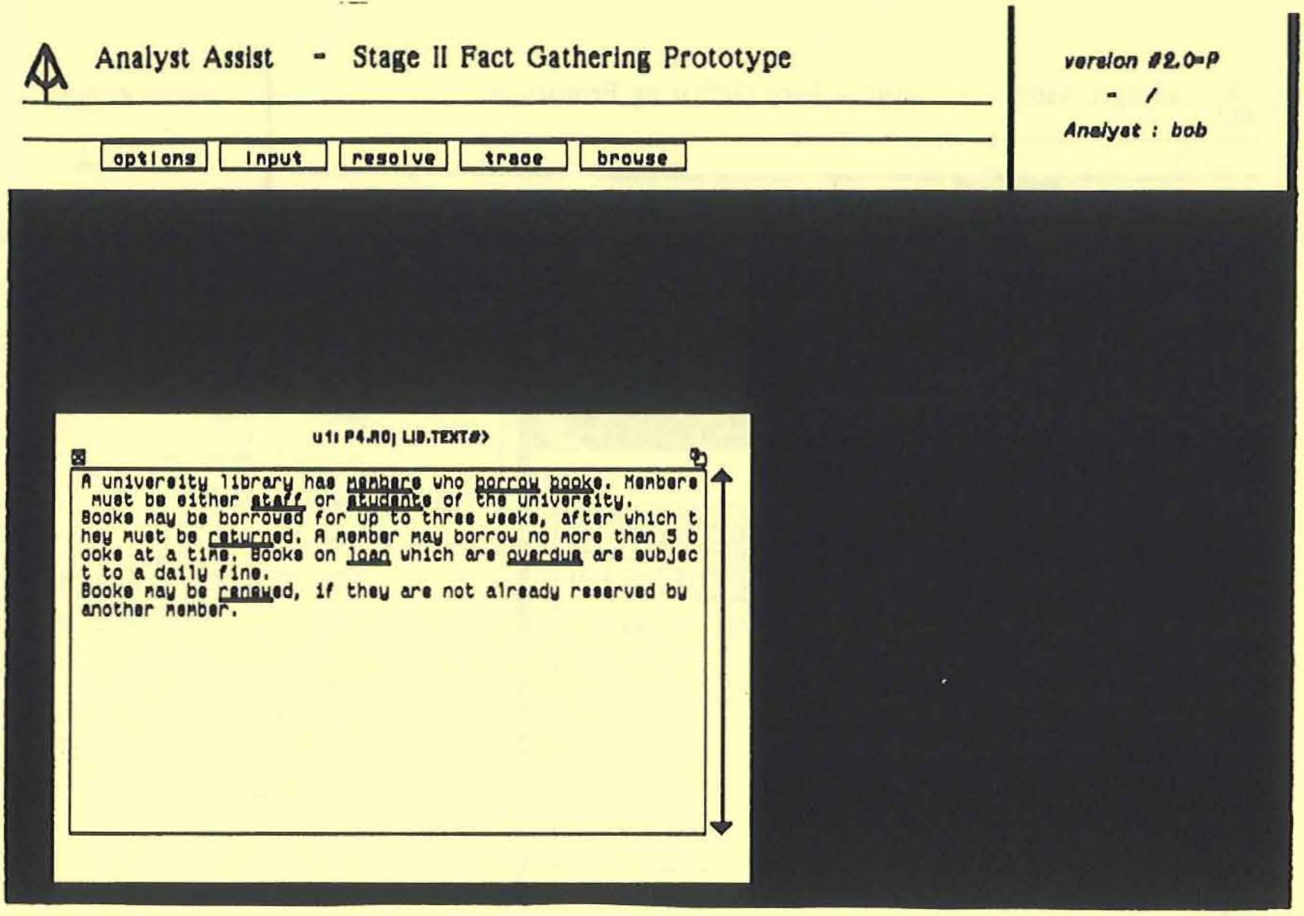
Text Analysis

> This facility allows an analyst to highlight keywords and phrases from a document. Single words are stored in the UFB as concepts. Phrases are analysed to find the concepts of interest and the conceptual relations between them. The concepts include those identified by the analyst and those which are known to the system as part of domain knowledge. Where the concepts are known to the domain knowledge, the appropriate conceptual relations can be derived thus producing conceptual graphs for storage in the UFB.

Concept Hierarchy Building

> The concepts known to the system are described in terms of their relationships with other concepts, and are arranged in a concept hierarchy (in fact a lattice). This facility allows an analyst to classify concepts by placing them directly in the hierarchy in terms of each concept's perceived semantics. The placement of a concept at a particular node has implications as to its allowable relations with other concepts. This in turn facilitates the development of scenarios based on these allowable relations which may be presented to a system developer for further analysis.

As an example of the fact gathering process using REST consider the case of a University Library. Text about this application domain is shown in the text analyser window in figure 3. The underlined text signifies that an analyst has highlighted these concepts as being of interest. The process of storing these concepts in the UFB is demonstrated graphically in figure 4. The schema 'cg1' represents the conceptual graph of the user-supplied information. As a direct result of this, concept type, schemas are created for the concepts "MEMBER", "BORROW" and "BOOK". The



```
A university library has members who borrow books. Members
 must be either staff or students of the university.
Books may be borrowed for up to three weeks, after which t
hey must be returned. A member may borrow no more than 5 b
ooks at a time. Books on loan which are overdue are subjec
t to a daily fine.
Books may be renewed, if they are not already reserved by
another member.
```

Figure 3: The Text Analyser

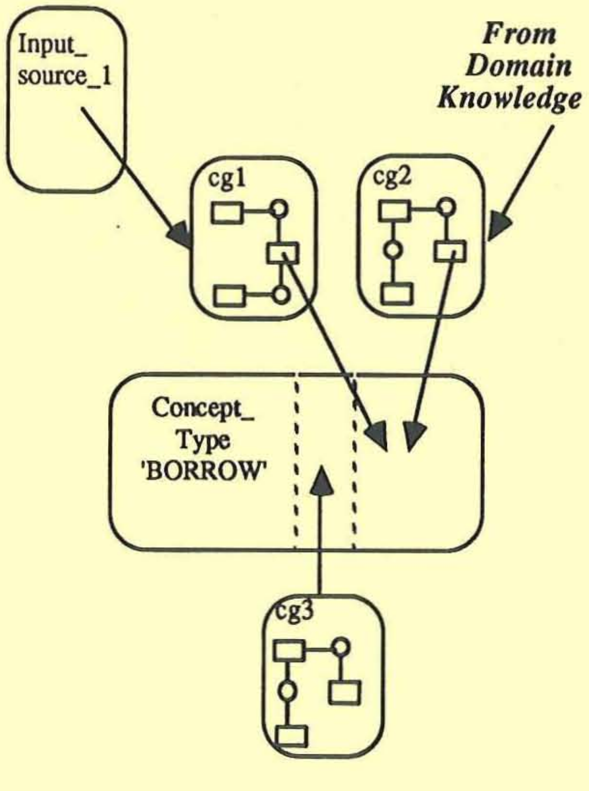
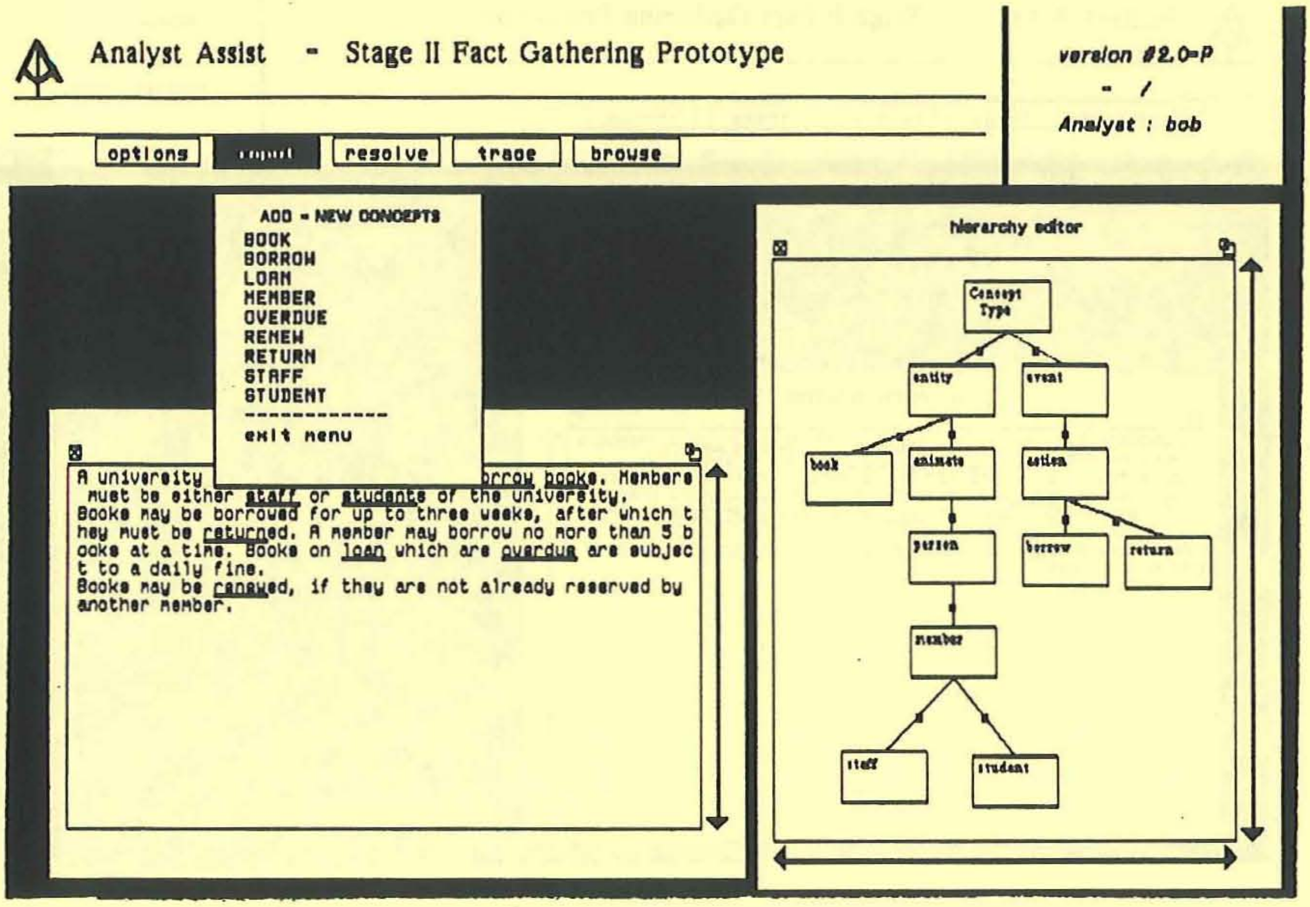


Figure 4: Creation of Concept_Type 'BORROW' in the UFB

conceptual graph cg1 is linked to each of these as a scenario for each concept. The relevant domain knowledge for the concept "BORROW" is shown as cg2, and is also provided as a scenario for the concept. The resolution of the two scenarios is trivial in this example, but in many cases an analyst might be faced with a number of different interpretations (most of which would be quite legitimate). By highlighting all possible interpretations of a concept, in particular those guided by the domain knowledge, it is much more likely that an analyst would capture a situation in a more faithful manner. In the example, the resulting conceptual graph, cg3, is linked to the current view slot of the concept schema. The schemas cg1 and cg2 are then marked as 'shadowed', to indicate that they have been resolved by cg3.

The concepts represented in the UFB may be viewed (and further extended) by using the hierarchy editor as shown in figure 5. Figure 5 shows the type of specialisation of each concept in relation to the pre-defined system concepts. This means that each new concept introduced in the UFB inherits relationships with other concepts from its supertype. The placement of a concept in the hierarchy does not preclude subsequent alterations which would change its position in the hierarchy. Such a change may result from further analysis and resolution of requirements.



```
A university                                    orrow books. Members
 must be either staff or students of the university.
Books may be borrowed for up to three weeks, after which t
hey must be returned. A member may borrow no more than 5 b
ooks at a time. Books on loan which are overdue are subjec
t to a daily fine.
Books may be renewed, if they are not already reserved by
another member.
```

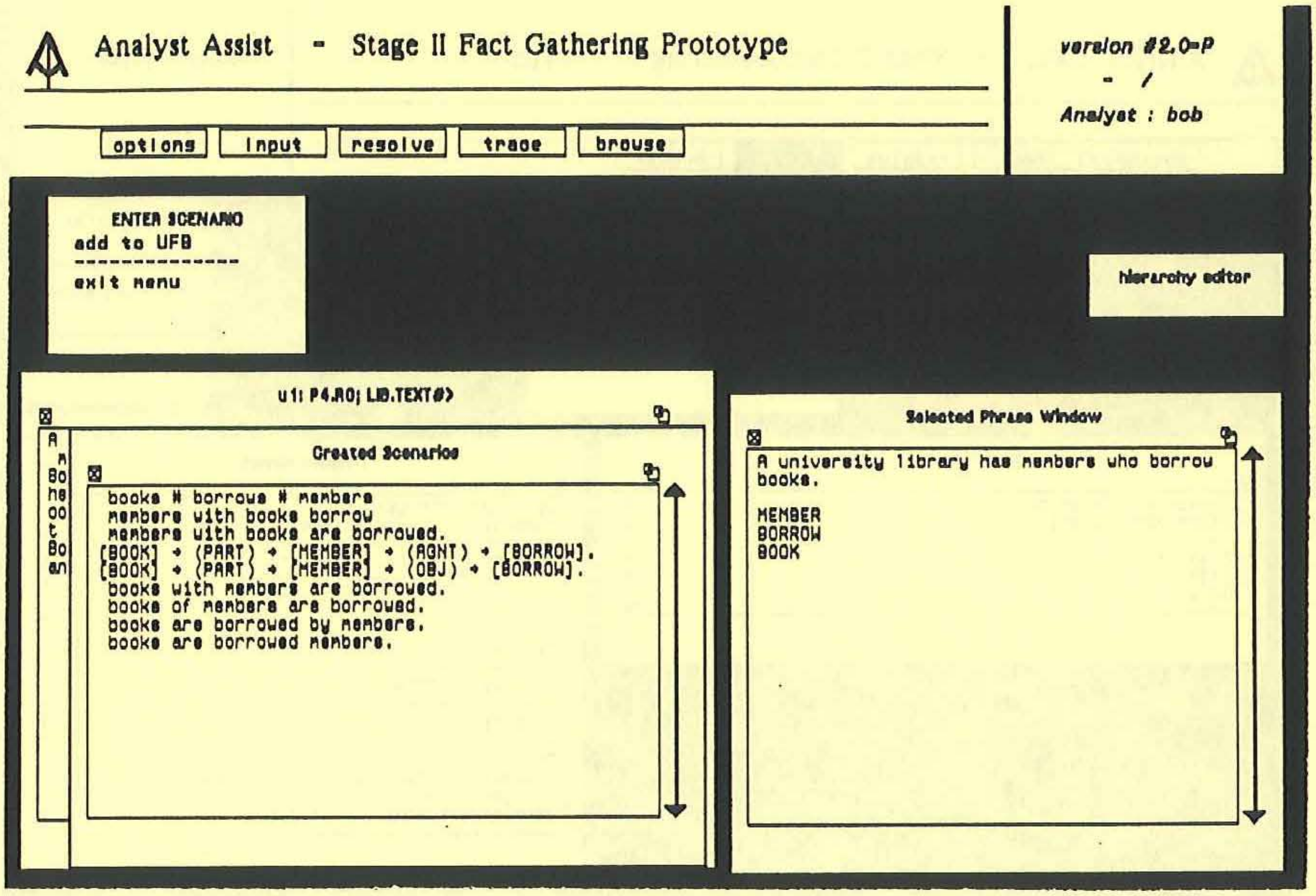
Figure 5: The Hierarchy Editor

In addition to identifying individual concepts such as MEMBER, BORROW and BOOK it is possible to highlight phrases within the text which imply semantic links between the concepts. Then the system can be prompted to derive the nature of these links. An example of this is shown in figure 6. The *selected phrase* window shows the highlighted phrase from the original text, together with a list of all currently known concepts. Links may be established between the three concepts and the system's interpretation of these links is presented in the *created scenarios* window. It is now up to the analyst working together with the end user to select the most appropriate interpretation which will subsequently be stored in the UFB.

4.2 Resolution of Captured Domain Specific Facts

Resolution of information within the UFB takes place under the supervision of the analyst, but some automatic checking of consistency can be carried out. These checks imply the need for a more formal representation of facts in the UFB. Where more complicated dependencies exist in the information, for example in the interdependence of requirements and constraints, a set of rules is required to maintain consistency.

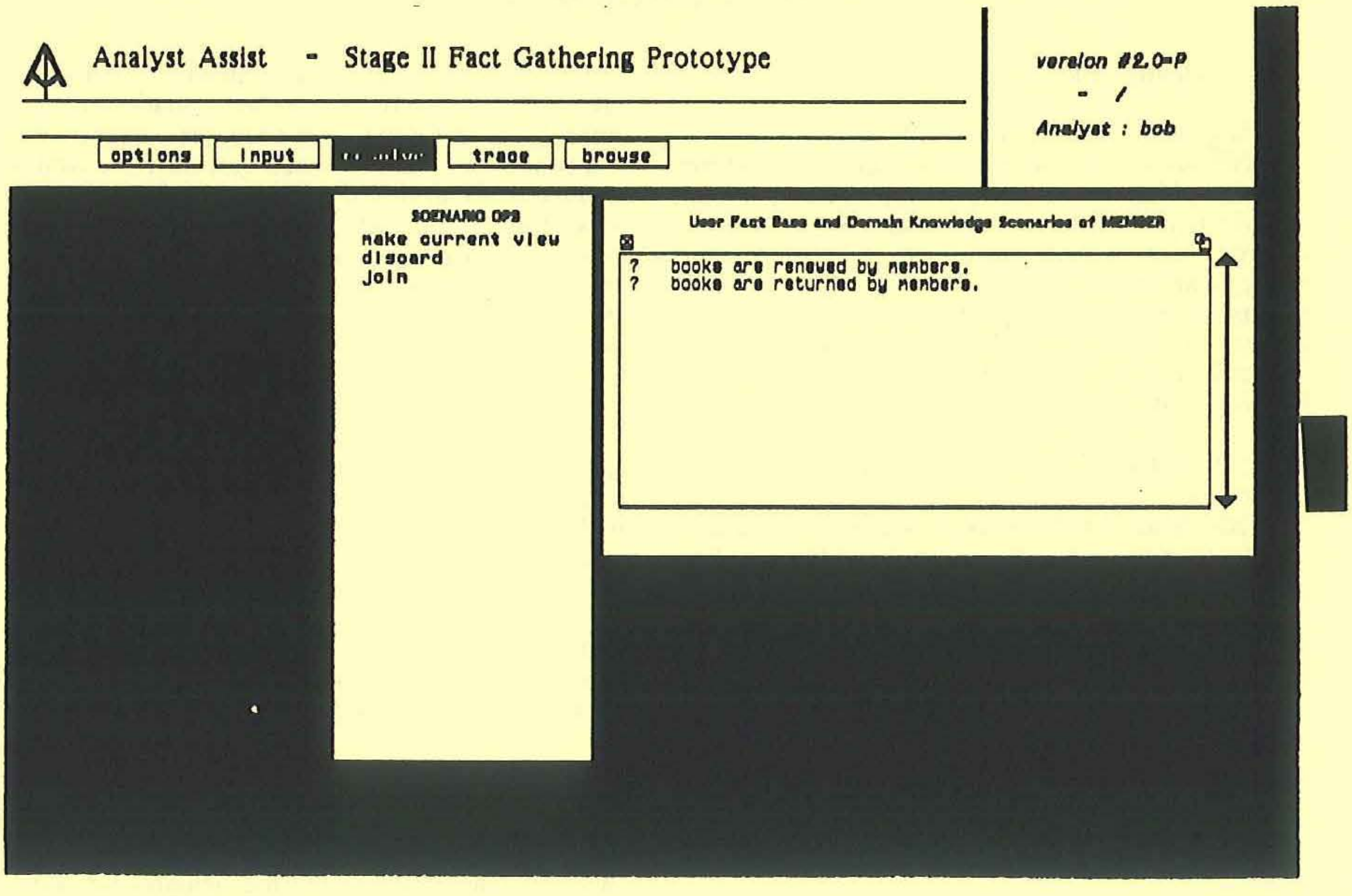
The current version of the prototype includes facilities for viewing the entire contents of the UFB about a particular concept and for the rationalisation of this information. As an example consider again the University Library case study. Figure 7 shows the information stored in the UFB, in the window labelled *User Fact Base and Domain Knowledge Scenarios* about the concept MEMBER. The window *scenario options* shows the available operations that can be performed on the UFB knowledge about MEMBER. The 'make current view' option relates to the agreed position as far as a concept is concerned; the 'discard' option allows the analyst to remove scenarios from consideration (they are nevertheless saved for possible future use); the 'join' option permits the merging of two or more existing scenarios. Figure 7 shows two unresolved scenarios for the concept MEMBER which may be joined and be made the current view. The advantage of the resolution facility is that it allows the incremental development of information about each concept in the UFB. The join operation includes functions to enforce consistency in the scenarios being joined, but in the case where conflicts arise, different views of the same concept may be developed in parallel.



```
 books # borrows # members
 members with books borrow
 members with books are borrowed.
[BOOK] → (PART) → [MEMBER] → (AGNT) → [BORROW].
[BOOK] → (PART) → [MEMBER] → (OBJ) → [BORROW].
 books with members are borrowed.
 books of members are borrowed.
 books are borrowed by members.
 books are borrowed members.
```

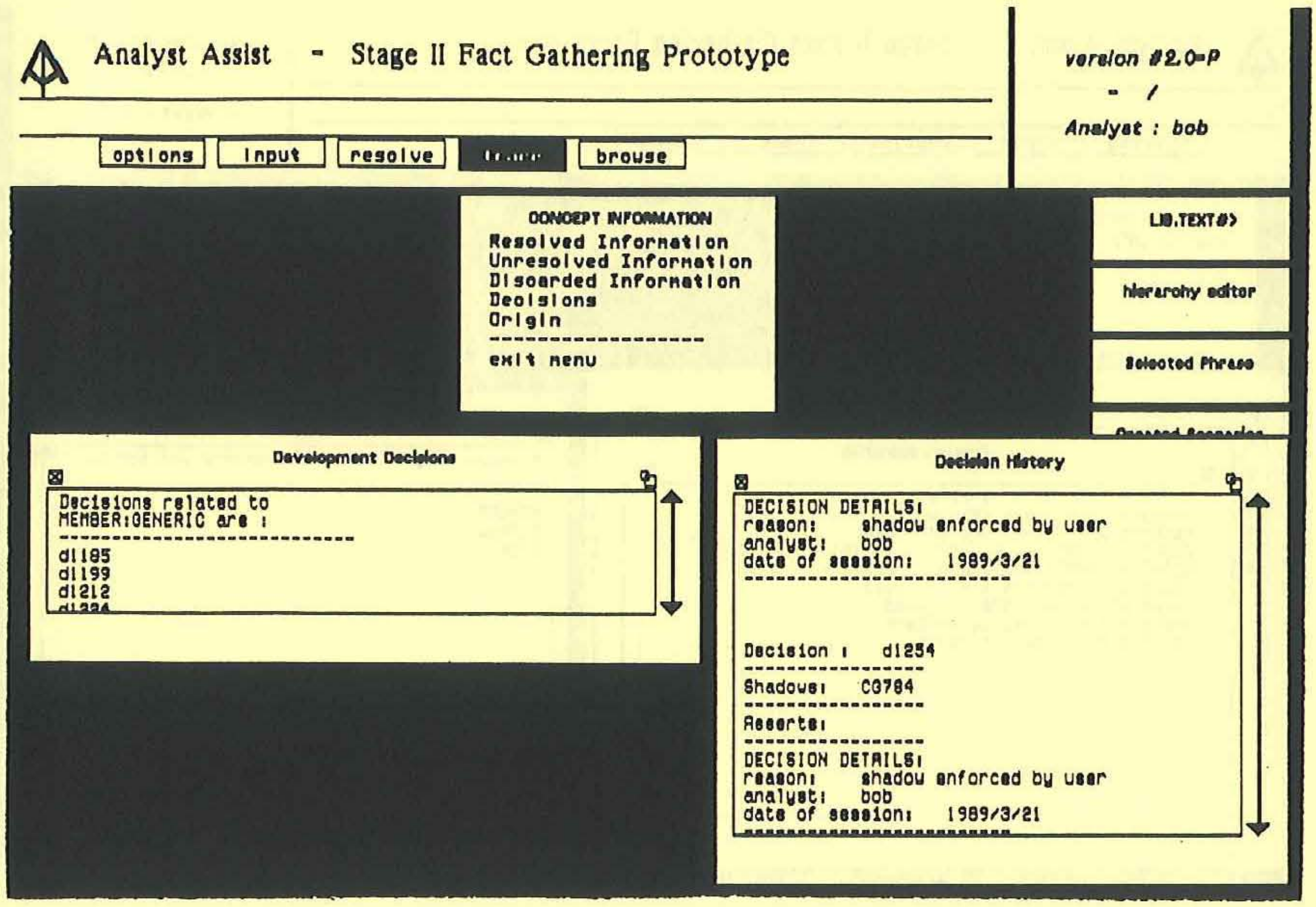
```
A university library has members who borrow
books.

MEMBER
BORROW
BOOK
```

Figure 6: Scenario Formation


Analyst Assist - Stage II Fact Gathering Prototype
version #2.0-P
- /
Analyst : bob
options
input
resolve
trace
browse
SCENARIO OPS
make current view
discard
join
User Fact Base and Domain Knowledge Scenarios of MEMBER

```
?    books are renewed by members.
?    books are returned by members.
```

Figure 7: Resolution Options



```
Decisions related to
MEMBER:GENERIC are :
----------------------------
dl185
dl199
dl212
```

```
DECISION DETAILS:
reason:      shadow enforced by user
analyst:     bob
date of session:     1989/3/21
--------------------------


Decision :   d1254
-----------------
Shadows:   CG784
-----------------
Asserts:
-----------------
DECISION DETAILS:
reason:      shadow enforced by user
analyst:     bob
date of session:     1989/3/21
```

Figure 8: Tracing Analysts' Decisions

4.3 Tracing Analyst Decisions

During the development of a requirements specification, analysts adopt certain lines of action, decide which information is appropriate, discard previously considered options and so on. In other words, the construction of a requirements specification is non-monotonic and therefore, tools to assist in the tracing of the diverse decisions taken by analysts in the process of deriving a specification are considered as providing another source of assistance.

The current version of REST allows the source of information in the UFB to be traced. This may include details of the source document, the analyst or the date of the input session. In addition to this, the system provides traceability of analyst decisions. In order to facilitate tracing of decisions, two types of information need to be stored. Firstly, facts which the analyst wishes to add without providing a documentary source, i.e. the analyst's own ideas and notes. Secondly, specific information concerning the decisions made about the data in the UFB, in particular decisions to do with the reconciliation of differing views.

In the UFB, conceptual graphs attached to concept types are marked as either 'shadowed' or 'current'. At any given time only one current view of a concept is allowed. Each current view would be linked to a decision showing how it was derived and hence which scenarios and previous views are shadowed by it. New scenarios and current views may be shadowed by subsequent decisions, but traceability is maintained by including these changes in the decision history.

The tracing of analysts' decisions in the current version of the prototype is shown in figure 8. The 'decision history' window shows the details of the decision labelled 'd1254' which has been selected by the analyst as relevant to the current view of the concept MEMBER. These details include a description of the type of decision made, the analyst responsible, the date and a list of the scenarios which have been shadowed as a result of the decision. Subsequently, the analyst can review any of the scenarios listed.

5. Conclusions

This paper has argued that the demand for more reliable and cost effective information systems should force us to seek solutions in areas most likely to yield maximum benefits. One of the major emerging themes in the area of information systems development research is the realisation that correct requirements specification holds the key to the construction of effective and flexible systems. At the same time it is recognised that the area of requirements capture is fraught with problems such as uncertainty and vagueness in the users' requirements, complexity about the modelled domain and lack of adequate techniques and tools which can bridge the gap between users and developers.

The work reported in this paper represents an attempt to improve requirements capture and specification through the use of knowledge engineering techniques. It is proposed that requirements specification is primarily a knowledge-intensive activity and the work reported here follows the premise that the next generation of requirements engineering environments will be knowledge-based environments. To this end, the work presented complements other research efforts in the wider sphere of software engineering, notably those of [Waters, 1985; Rich et al, 1987; Pietri, 1987; Keiser & Feiler, 1987; Lubars & Harandi, 1987].

In this paper a clear distinction is made between requirements and design specifications. It is argued that a requirements specification should be concerned with the understanding of a problem whereas a design specification should pay attention to logical and physical structures that implement the requirements. Therefore, the development of a requirements specification involves modelling the relevant application domain resulting in a model which is cognitive in nature. In the past, requirements specifications have played a passive role and have been viewed as fixed throughout the life of an information system. The thesis put forward in this paper is that a requirements specification needs to evolve to reflect changes in the application domain and that these changes must be implemented in such a way so as to ensure that users and developers are clear on the implications that the changes will have on the design and operational characteristics of the system. Because of the nature of requirements elicitation, the paper proposes that, this objective can be achieved by a knowledge-based approach, such as that followed in the *Analyst Assist* system.

The system described in this paper has been developed on Texas Instruments Explorers using ART and Common Lisp. The prototype system has also been ported on SUN workstations also running under ART and Common Lisp. It has been the intention of the authors to adopt a single unifying representation formalism for the expression of facts in the *user fact base* and knowledge within the *domain knowledge-bases* since such an approach has several advantages. The informality which characterises much of the initial requirements elicitation process is supported by the linguistic basis of conceptual structures. The representation is based on a user-defined type hierarchy of concepts occurring in the domain of interest. Linked to each concept in the hierarchy is a definition, together with examples of semantic and episodic information about the concept, expressed as conceptual graphs. Our choice of formalism also has implications for the validation of the requirements *independently* of a design method. The linguistic basis of the concept and relationship definitions allows both the domain knowledge and the existing application knowledge to be presented to the user in terms of the semantics of the domain rather than the semantics imposed by any design method, thus broadening the applicability of the general approach to any system development method.

REFERENCES

Adelson, B. and Soloway, E. (1985) *The Role of Domain Experience in Software Design* , in IEEE Transactions on Software Engineering, Vol. SE-11, No. 11, November 1985.

Adhami, E. (1988) *An Environment for the Execution and Graphical Animation of JSD Specifications*, International Workshop on Knowledge-Based Systems in Software Engineering, UMIST, Manchester, U.K., March 1988.

van Assche F., Layzell, P.J., Loucopoulos, P., Speltincx, G.(1988) RUBRIC: A Rule Based Representation of Information System Constructs, in Proc of the 5th Annual ESPRIT Conference, Brussels, Nov 14-17, 1988, Nort-Holland, pp.438-452.

Balzer, R., Cheatham, T.E, Green, C., (1983) *Software Technology in the 1990's: Using a New Paradigm*, Computer, November 1983, pp. 39-45.

Bird, B. (1988) *Building of JSD Specifications*, International Workshop on Knowledge-Based Systems in Software Engineering, UMIST, Manchester, U.K., March 1988.

Chapin, N. (1979) *Software Lifecycle*, INFOTEC Conference in Structured Software Development, 1979.

Champion, R.E.M., Gibson, M., Harthoorn, C. (1988) Conceptual Structures in the Fact Gathering System, Analyst Assist internal report AA-U0067, UMIST, 1987.

DeMarco, T. (1978) *Structured Analysis and System Specification*, Yourdon Press.

Dubois, E. et al (1986) *The ERAE model: A case study*, in 'Information Systems Methodologies: Improving the practice' Olle T.W., Sol H.G. and Verrijn-Stuart A.A., (eds), pp.87-106, IFIP-North Holland, 1986.

Fickas, S. (1987) *Automating the Analysis Process: An Example*, 4th International Workshop on Software Specification and Design, Monterey, USA.

Gane, C. and Sarson, T. (1979) *Structured Systems Analysis: Tools and Techniques*, Prentice-Hall

Gibson, M. and Harthoorn, C. (1987) *The Use of JSD*, Analyst Assist internal report AA-U0010, UMIST, 1987.

Greenspan, S.J., (1984) *Requirements Modeling: A Knowledge Representation Approach to Software Requirements Definition*, Technical Report No. CSRG-155, University of Toronto, 1984.

Jackson, M. (1983) *System Development*, Prentice-Hall.

Keiser, G.E. & Feiler, P.H. (1987) *An Architecture for Intelligent Assistance in Software Development*, 9th International Conference on Software Engineering, March 30 - April 2, 1987, Monterey, USA.

Layzell, P.J. & Loucopoulos, P. (1987) *Systems Analysis and Development*, Chartwell-Bratt, 2nd edition.

Loucopoulos, P. and Champion, R.E.M. (1988) *Knowledge-based approach to requirements engineering using method and domain knowledge*, Knowledge-Based Systems, Vol. 1, No. 3, June 1988.

Loucopoulos, P., Layzell, P.J., Champion, R.E.M., Gibson, M. (1988) A Knowledge-based Requirements Engineering Environment, Proc, Conf. on Knowledge Based Software Assistance (KBSA), Utica, USA, 2-4 August, 1988.

Lubars, M.D. & Harandi, M.T. (1987) *Knowledge-Based Software Design Using Design Schemas*, 9th International Conference on Software Engineering, March 30 - April 2, 1987, Monterey, USA.

MacDonald, I. (1986) *Information Engineering- An improved, automatable methodology for designing data sharing systems*, in 'Information Systems Methodologies: Improving the practice' Olle T.W., Sol H.G. and Verrijn-Stuart A.A., (eds), pp.173-224, IFIP-North Holland, 1986.

Morris, E.P. (1985) *Strengths and Weaknesses in Current Large DP*, Alvey/BCS SGES Workshop, Jan 1985, Sunningdale, U.K.

Pietri, F. et al (1987) *ASPIS : A Knowledge-based Environment for Software Development* , in ESPRIT '87 : Achievements and Impact, pp 375-391, North Holland, 1987.

Rich, C., Waters, R.C., Reubenstein, H.B. (1987) *Toward a Requirements Apprentice*, 4th International Workshop on Software Specification and Design, Monterey, USA.

Ross, D.T. (1979) *Structured Analysis: A Language for Communicating Ideas*, IEEE Transactionson Software Engineering, Vol SE-3, No.1.

Rzepka, W. & Ohno, Y. (1985) *Requirements Engineering Environments: Software Tools for Modelling User Needs*, IEEE Computer, April 1985.

Sowa, J.F., (1984) *Conceptual Structures: Information Processing in Mind and Machine*, Addison-Wesley Publishing Company, 1984.

Verheijen, G. & van Bekkum, J. (1983) *NIAM: An Information Analysis Method*, in 'Information systems design methodologies : a comparative review', OlleT.W, Sol H.G. and Verrijn Stuart A.A., (eds), IFIP WG 8.1 CRIS I, North Holland.

Vitalari, N.P. and Dickson, G.W. (1983) *Problem Solving for Effective Systems Analysis An Experimental Exploration* , in Communications of the ACM, Vol. 26, No. 11, November 1983.

Waters, R.C. (1985) *The Programmer's Apprentice: A Session with KBEmacs*, IEEE Transactions on Software Engineering, 11(11) pp. 1296-1320, November 1985.