Rule-Based Requirements Specification and Validation

A. Tsalgatidou

EDP Department
Greek P.T.T
Megalou Vassiliou 6-8,
Rouf, Athens 118 54
Greece

V. Karakostas P. Loucopoulos

Dept. of Computation,
UMIST, P.O. Box 88,
Manchester M60 1QD,
United Kingdom

Abstract

Requirements specification has only recently been acknowledged as one of the most important phases in the overall software life cycle. Since the statement of a complete and consistent set of requirements involves user participation, our approach investigates how user oriented formalisms and techniques could be employed for the specification and capturing of requirements. We propose the use of rules as a natural means for expressing the application domain knowledge, and introduce a number of techniques such as *semantic prototyping* and *animation* for the validation of the requirements.

Keywords: *requirements specification, executable specifications, rule bases, animation, Petri-nets, logic programming, conceptual modelling.*

Introduction

The expansion of the Information Technology sector in recent years has been responsible for increasing demands for bigger and more-complex computer applications. As, however, the computer systems' sophistication increases, the inadequacy of the traditional software development approaches becomes apparent. The major drawbacks of conventional software development methods are identified to be in the phase of requirements capturing/specification. While most of the approaches [deMarco 78] [Jackson 83] are good in describing the *artifact* (software system) through its various phases (i.e. as specifications, design and code) they fall short in their provision of adequate expressive power for describing the application domain. Even worse, many methods neglect to provide support for the analyst during the important phase of validating the captured requirements. Consequently, the state-of-the-art practice results in systems which do not meet user requirements, and which are expensive to maintain, since it is well known that the fixing of errors occurring due to misunderstanding of user requirements, is more expensive when the system has been implemented [Yeh et al 84].

We see the requirements specification phase as consisting of two major activities, namely requirements capturing, and requirements analysis [Dubois Hagelstein 86].

The objective of the requirements capturing phase is to depict the desired contribution of the software system in terms of application domain concepts and their interrelationships. The objective of the requirements analysis phase is to identify how the modelling assumptions are interrelated, and how they affect the future software system. As a consequence, the two phases pose different demands on the employed requirements formalism and technique. In order to model application domains of significant complexity we need an adequately rich formalism which provides a repertoire of concepts that is sufficiently rich for our ontological assumptions about the application domain and semantic accounts about every modelled aspect of the application domain [Mylopoulos 86]. In order to identify the consequences of our modelling assumptions we need a model with *deductive power* [Dubois et al 86].

Our approach is particularly suited for a class of applications known as *data intensive, transaction-oriented* information systems.These systems are characterised by large, often decentralized databases containing persistent application information, accounting for more than 80% of the investments in information systems in use today. We have observed that the requirements for such systems can be captured in terms of rules, conveying information about various aspects of the structure and behaviour of the domains. In this respect our line of research is similar to the one carried by approaches which advocate the rule-based specification of information systems [van Assche et all 88]. However, we pay particular attention to the

validation aspect, and this is where this paper's discussion focuses on.

The structure of this paper is as follows. Section 1 introduces the modelling formalism used for capturing the static aspects of the application domain knowledge. In Section 2, the modelling techniques for the dynamic aspects of the application are modelled. A validation technique known as *semantic prototyping* is the subject matter of Section 3, whilst in the next section a *rule animation* technique, used for validating the system's dynamic aspects is discussed. We conclude with an overview and summary of our approach.

1 Static Modelling Constructs

The conceptual modelling formalism employed by our approach is an extension of entity-relationship based models [Chen 76] [Nijssen 89] enriched with the addition of constructs used for specifying domain knowledge which cannot be expressed by entities and relationships alone. In this respect, the formalism comes closer to contemporary conceptual modelling languages [Greenspan 84] and knowledge representation formalisms [Sowa 84].

The primary static modelling constructs are *entities*, *relationships* and *static rules*. Entities are the phenomena of interest within the application domain. Relationships are associations between the entities which are meaningful and useful from the information system's viewpoint. In contrast to the entity-relationship-attribute model and its variants, our approach does not make any distinctions between entities and attributes, as such distinctions are made usually on subjective criteria of the analyst. According to our viewpoint, attributes are equally important to entities, from an information system's perspective, and should be modelled as such. The purpose of a rule is to constrain the allowable set of entities and associations. The abstraction mechanisms employed by our approach in order to cope with the size and complexity of the application domains are *classification*, *generalization/specialization* and *aggregation* [Borgida et al 84]. Classification refers to the ability to model a set of similar concepts as a separate object, eg. the concept *product* is an abstraction over a set of products. Generalization/specialization refers to the ability to associate classes of concepts using superset/subset relations. A *high demand product* is a subclass of *product* in the sense that high demand products are also products.Similarly, *ordinary product* is another subclass of product. Finally, aggregation is the abstraction technique of viewing a concept as the sum of its parts (constituting components). A *product* can be considered as consisting of a *product code*, *product price* and *product description*.

Rules are pieces of knowledge used to further distinguish the application domain from similar ones. *Static rules* are an important modelling constructs in the sense that they increase our specification power beyond the definition of entities, relationships and cardinality constraints. A static rule is a linguistic expression which describes the state of affairs in the application domain

at any time. A static rule for example may state that no *product* can be a *high demand product* and an *ordinary product* at the same time. This would be stated as follows.

static_rule1: high_demand_product and ordinary_product are mutually disjoint

Static rules can run to any size of complexity, relating for example, a number of different entity and relationship classes as in the following example: "High demand are those products which have associated with them a number of at least ten incoming orders of at least £100 each, over the last six months". This would be stated as follows.

static_rule2: product.X is high_demand_product if #(incoming_order.Y about product.X and incoming_order.Y of value > 100) < 10.

In summary, the static modelling constructs as applied to the modelling of a stock control system are shown in Figure 1.

2 Dynamic Modelling Constructs

The ability to model the dynamic aspects of an application domain is of paramount importance, therefore our approach provides a number of modelling constructs for this purpose. An application domain is perceived as changing due to a number of *events*. Events are the carriers of change within an application domain in the sense that they modify the structure of the domain by introducing, deleting or modifying instances of entities and relationships. Similarly to the modelling of static constructs we provide a number of abstraction mechanisms for modelling the dynamic constructs. Events are stated as *dynamic rules* which consist of three parts, namely

- a *when_part which* is a boolean expression over the state of affairs of the application domain, time conditions, and *signals* which are generated within the application domain (*internal signals*) or within its environment (*external signals*).

- a *precondition* which is a boolean expression over the application domain's state of affairs. A precondition describes the set of states in which an event can take place.

- an *action part* which is the set of actions introduced by the event in terms of

introducing/deleting new entities and establishing/destroying associations between them.

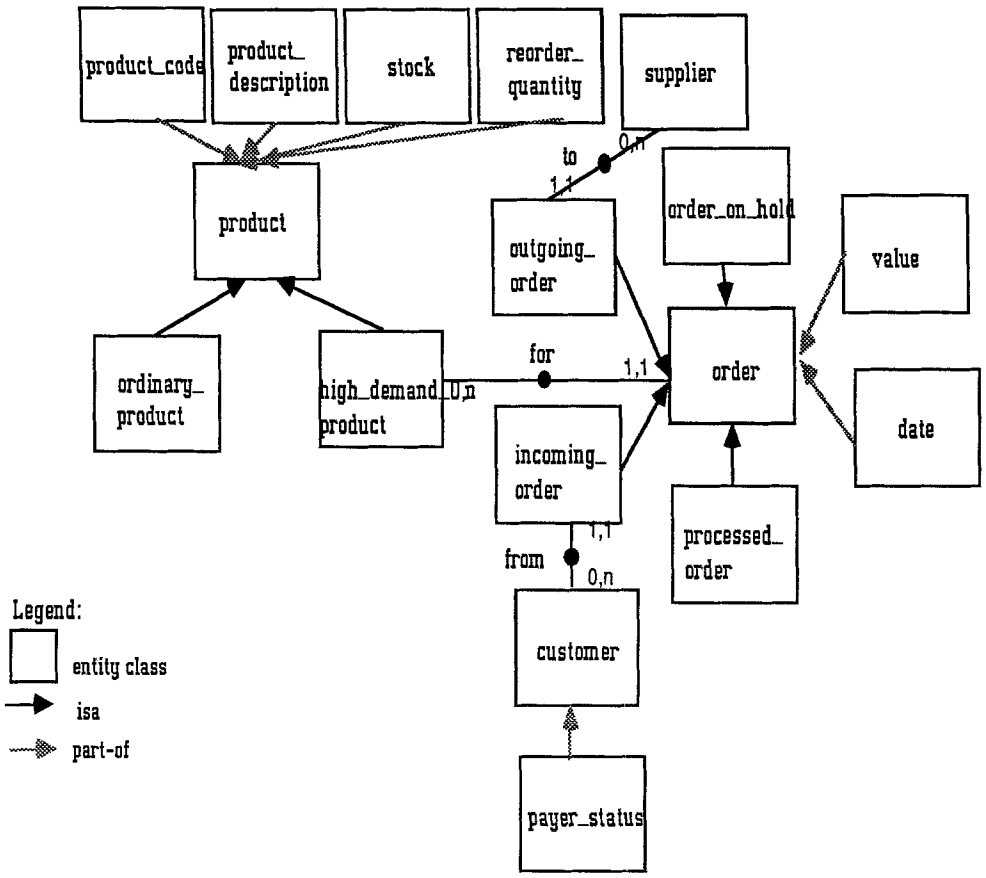

Figure 1: Conceptual model of a stock control system

In order to illustrate these concepts in more detail, consider the stock control system described in the last section.When a signal from the environment indicates that an order from a customer has arrived, then, if the status of the customer as a payer is bad, an instance of a backorder is created. If, however, the payer status of the customer is good then an instance of an order is created. This knowledge of the stock control system's dynamic behaviour is captured in two dynamic rules as follows.

dynamic_rule1:

whensignal_order_arrived(customer.C, product.P, quantity.Q)

ifpayer_status of customer.C = 'bad'

thencreatebackorder(customer.C, product.P, quantity.Q, date(NOW)).

dynamic_rule2:

whensignal_order_arrived(customer.C,product.P, quantity.Q)
ifpayer_status of customer.C = 'good'
thencreateorder(customer.C, product.P, quantity.Q, date(NOW)).

The above two rules, are examples of rules which are triggered by the environment, in the form of external signals. Other rules, however are triggered when a particular operation takes place in the application domain. In the domain of our case study, orders are processed only if, after the order is fulfilled, there remains sufficient stock for the products (above a given reorder point). Because of this, two more dynamic rules related to the activity of process order are entered. The rules are triggered by an internal signal called *request_process_order* and are shown below.

dynamic_rule3:

whensignal_request_process_order(customer.C, product.P, quantity.Q)
if(quantity.Q - stock of product.P > reorder_point of product.P)
thencreateprocessed_order(customer.C, product.P, quantity.Q, date(NOW)).

dynamic_rule4:

whensignal_order_arrived(customer.C, product.P, quantity.Q)
ifnot(quantity.Q - stock of product.P > reorder_point of product.P)
thencreateorder_on_hold(customer.C, product.P, quantity.Q, date(NOW)).

When an internal signal called *process_order_on_hold* arrives, the orders which are on hold are processed. Also, a signal about the arrival of new stock will result in an increase in the product's quantity on stock. These are illustrated as follows.

dynamic_rule5:

whensignal_process_order_on_hold(order_on_hold.O)
then createprocessed_order(customer.C of order_on_hold.O,
product.P of order_on_hold.O, quantity.Q of order_on_hold.O);
destroyorder_on_hold.O.

dynamic_rule6:

whensignal_new_stock_arrived(product.P, quantity.Q)
thenincreasestock.S of product.P byquantity.Q.

The use of entity class hierarchies allows us to give inheritance semantics to the dynamic rules. A dynamic rule is said to apply also to a class' subclasses unless otherwise stated. An example of rule overriding is as follows. For high demand products, the orders are processed immediately, irrespectively of whether the stock may fall below the reorder point or not, if the customer who issues the order has a good payer status. Effectively, this means that dynamic rule 3 will be overwritten by dynamic rules 3.1 and 4.1, in the case of high demand products, as follows.

dynamic_rule3.1:

whensignal_request_process_order(customer.C, high_demand_product.P,
quantity.Q)
if payer_status of customer.C = 'good'
then createprocessed_order(customer.C, product.P, quantity.Q, date(NOW)).

```
dynamic_rule4.1:
        when signal_request_process_order(customer.C, high_demand_product.P,
                                          quantity.Q)
        if   payer_status of customer.C = 'bad'
        then create order_on_hold(customer.C, product.P, quantity.Q, date(NOW)).
```

The use of dynamic rule and entity hierarchies with inheritance overridance is illustrated in Figure 2.

3 Static Model Validation

Validation is an all-important phase, since its omission could result in misconceived and inappropriate models of the application domain, which will result in software systems that fail to realize their objectives. The validation phase is essentially an attempt to prove that the model is *internally consistent* and conforming to the users' conceptualization of the domain. Verifying the internal consistency of the model is a task that can be automated to a significant extent [Wohed 87]. This essentially requires that the model is expressed in a formalism with deductive power. We have opted for mapping the modelling constructs to an executable logic language, something that combines the advantages of a rigorous formalism with those of *rapid prototyping* [Budde et al 84]. Checking the internal consistency of the model involves the following activities.

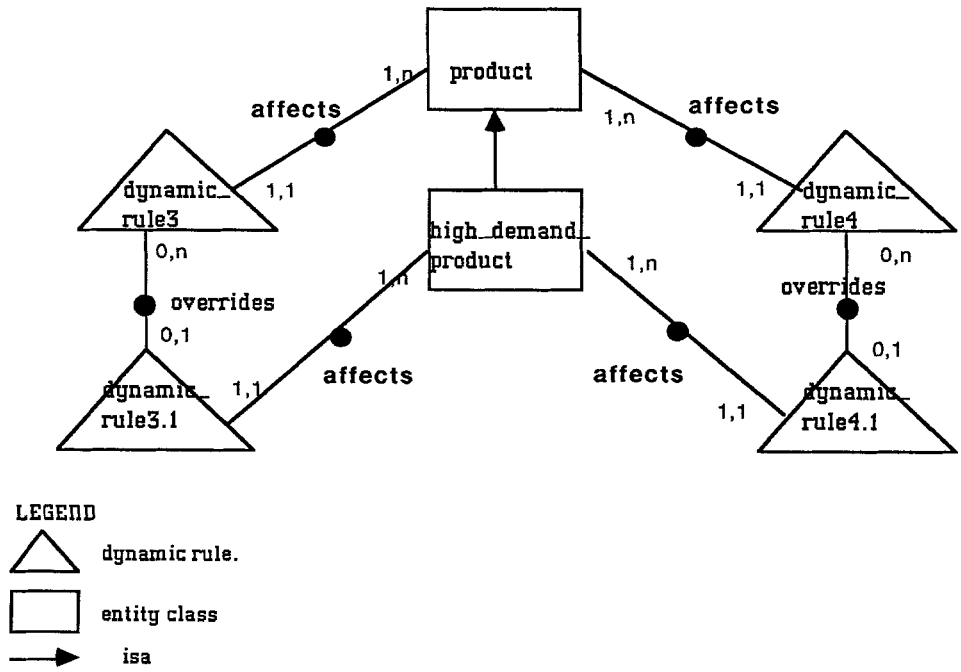

Figure 2. Entity and dynamic rule hierarchies

- Checking the well-formedness of the model, i.e. that it has been constructed according to the syntax rules of the modelling formalism. Checks in this category would include, for example, detecting cyclic *isa* hierarchies (i.e. two entity classes are mutually subclasses of each other), detecting inconsistencies in the use of aggregation relations (two concepts cannot be parts of each other at the same time), etc.

- Checking the consistency and completeness of the rules, i.e. detecting, self-contradicting or inconsistent rules.

Clearly, checking a model's consistency and completeness is not a task that can be fully automated, because of the difficulty to construct a complete logic theory to check the rules against. To overcome this problem we propose a technique known as *semantic prototyping* [Karakostas Loucopoulos 88] which advocates the active participation of the user in the validation phase. The technique draws results from the theory of logic [Tarski 56], by viewing the model as a logic theory which admits a number of interpretations. The technique proceeds by trying to *refute* the model's validity by trying to prove that it admits an interpretation (i.e. a set of instances of entities, relationships and rules) which is inconsistent. It also attempts to prove the model's agreement with the user's perception of the application domain by trying to find interpretations agreeable to the user. To make this approach clearer, consider the following example. Assume that the analyst manages to find a particular product which is classified both as *high demand* and *ordinary*. This disproves the validity of *static_rule1* which states the disjointness of the two entity classes. This contradiction can be resolved by reconciliations with the user which will result in *static rule1* being dropped, or perhaps in the introduction of an additional category of products.

Another facet of semantic prototyping involves the execution of *realistic scenarios* concerning aspects of the application domain. The user is guided step by step to explain how the test data conform with the static constructs of the application domain. This technique can reveal inconsistencies and omissions in the model definitions. For example, two real customer orders, a valid and an invalid one, could be used as test data. The user would be asked to explain, according to the rules of the application domain model, why he would accept the first, and reject the second order. If he was unable to do so, then a missing or wrongly stated rule could be detected.

The semantic prototyping technique constitutes a significant improvement over traditional validation techniques like *structured walkthroughs*. It can be automated to a large extent due to the use of Prolog as the target language on which the modelling constructs are mapped.

However, the technique requires an analyst with experience in the application domain, able to select the appropriate test cases, and to identify potential sources of inconsistencies within the model. The same applies to the validation of the dynamic aspects, discussed in the next section.

4 Dynamic Model Validation

The need to validate the dynamic aspects of the modelled domain has made necessary the invention of a number of techniques, similar to those used for static model validation described in the previous section. The techniques aim at proving the following things:

- the internal consistency of the model's behaviour
- its consistency with respect to the user's perception of the application domain
- the model's completeness.

Towards these ends, we have adopted a Petri Net representation of the dynamic rules, in order to give them formal semantic accounts and to define their interdependencies in a rigorous manner. The Petri-net formalism [Petri 62] and its variants (*augmented Petri Nets*) have received considerable attention in the area of information systems modelling [Zisman 76]. The graphical formalism introduced in this section uses *places* to represent signals (the WHEN part) and transitions are inscribed with the IF and THEN parts of the rules. Since every rule needs a signal to be triggered, all the transitions will have at least one input representing the triggering signal. In Figure 3 we give a Petri Net representation of the dynamic rules applying to class *product*, introduced in Section 2.

The major advantage of the net model is that its graphical representation coupled to its formal semantics leads to its validation using animation techniques. A prototype tool has been developed to edit and animate nets with places as signals and transitions inscribed by rules [Tsalgatidou 88]. Animation of the model can help in detecting redundant and conflicting situations by highlighting the rules inscribed to every transition every time a transition is enabled. Redundancy occurs when two or more rules can fire in the same situation giving the same results whereas conflict occurs when rules firing in the same situation produce contradictory results.

One of the model's feature's is that the number of tokens that each place can hold is countable. This feature enables the detection of circular rules by the assignment of output places to some transitions which would serve as counters of the number of times the transitions fire. A place assigned as output to a transition receives a token every time the transition fires. If this place is no input to any other transition, its tokens will not be consumed. Therefore the number of tokens that this output place will be holding will correspond to the number of times that the transition

has fired. If such a place is continuously receiving tokens, this may be an indication of the existence of circular rules.

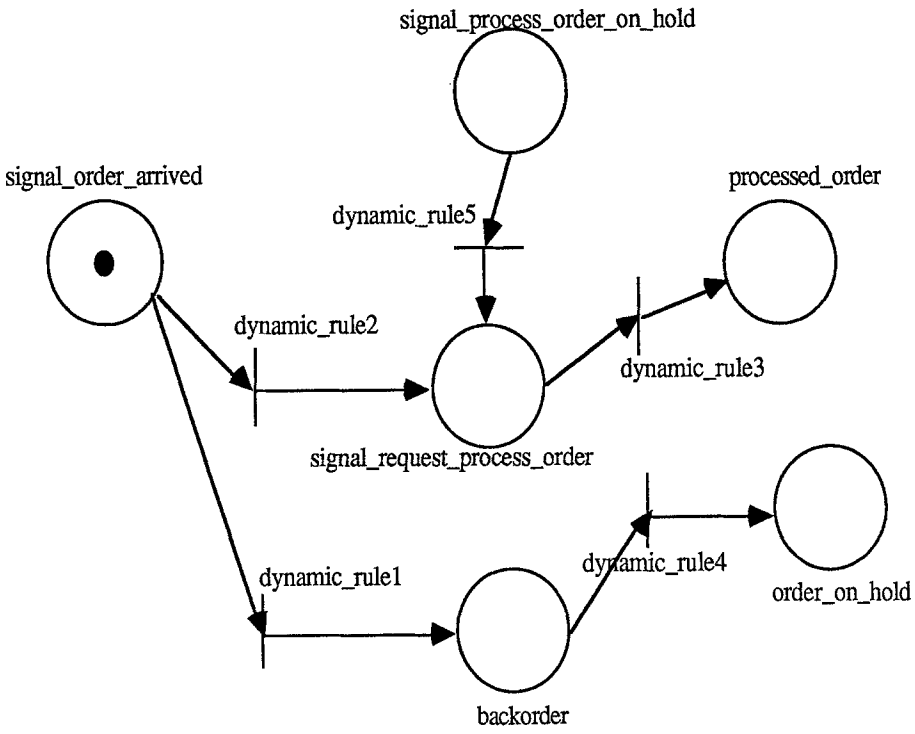

Figure 3: A Petri Net for dynamic rules affecting *product*

Missing rules can also be detected by animation. For example, if a signal, which is expected to trigger some actions, has been generated and nothing is happening, this may mean that some rules are missing. Another indication of missing rules is when there are some transitions which never become enabled. If the triggering part of the rule is internal to the system, this would mean that the necessary rule for the production of the signal is missing.

Conclusions

As the complexity of software applications continues to increase, it becomes more apparent that the traditional approaches focusing in specifying the artifact instead of its role within the application domain will become more unsuitable. For a large category of computer applications like embedded systems, office information systems and other knowledge based/expert

applications the need to model the application domain in user-oriented terms becomes apparent [Borgida et al 85].

Our approach succeeds in providing the following.

- a rich formalism for expressing knowledge about complex domains in user-oriented terms, and

- techniques for validating the models' correspondence to the users perception of the application domain.

Compared to contemporary *knowledge-based* approaches to requirements modelling[Anderson Fickas 89] [Loucopoulos Champion 89] [Reubenstein Waters 89], we put more emphasis on the validation aspects, since it is them which become critical in large and complex applications. Currently, prototype validation tools, implemented in languages like Prolog and Pop-11, are running on Sun workstations. The plans for future enhancements of our approach include the automation of the semantic prototyping technique using *case-based* reasoning [Hammond 86], i.e. guiding the validation process using experience acquired from similar domains.

References

[Anderson Fickas 89]
Anderson, J. S., & Fickas, S. *A Proposed Perspective Shift: Viewing Specification Design as a Planning Problem.* Proc. Fifth Int. Workshop on Software Specification and Design, May 19-20, 1989, Pittsburgh, PA, USA.

[van Assche et al 88]
van Assche, F., Layzell, P. J., Loucopoulos, P., Speltincx, G. *Information Systems Development: A Rule-Based Approach.* Journal of Knowledge Based Systems, September 1988.

[Borgida et al 84]
Borgida, A., Mylopoulos, J., & Wong, H. K. Z. *Generalization/specialization as a basis for software specification.* In Brodie, M. et al (eds.)."On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages. Springer-Verlag, New York, 1984.

[Borgida et al 85]
Borgida, A., Greenspan, S., Mylopoulos, J. *Knowledge Representation as the Basis for Requirements Specification.* COMPUTER, April 1985.

[Budde 84]
Budde, R. (ed.) *Approaches to Prototyping.* Springer-Verlag, Berlin, 1984.

[Chen 76]
Chen, P. P. S. *The Entity-Relationship Model: Towards a Unified View of Data.* ACM TODS, Vol. 1, No. 1, March 1976.

[Dubois Hagelstein 86]
Dubois, E. & Hagelstein, J. *Reasoning on Formal Requirements: A Lift Control*

System. Proc. Fourth Int. Workshop on Software Spec. and Design, April 3-4, 1987, Monterey, CA.

[Dubois et al 86]
Dubois, E., Hagelstein, J., Lahou, E., Ponsaert, F., Rifau, A., Williams, F. *The ERAE Model: A Case Study*. In "Information System Design Methodologies: improving the practice", Olle, T., W., Sol, H., G., Verrijn-Stuart, A., A. (eds). North-Holland Publishing Company, IFIP 1986.

[Greenspan 84]
Greenspan, S., J. *Requirements Modelling: A Knowledge Representation Approach to Software Requirements Definition*. Technical Report No. CSRG-155, University of Toronto, 1984.

[Hammond 86]
Hammond, K. *"CHEF": A Model of Case-Based Planning*. In Proc. of the Fifth National Conf. on Artificial Intelligence, Philadelphia, PA, 1986.

[Jackson 83]
Jackson, M. *System Development*. Prentice-Hall International, London, 1983.

[Karakostas Loucopoulos 88]
Karakostas, V. & Loucopoulos, P. *Verification of Conceptual Schemata Based on a Hybrid Object Oriented and Logic Paradigm*. Journal of Information and Software Technology, Vol. 30, No. 10, December 1988.

[Loucopoulos Champion 89]
Loucopoulos, P. & Champion, R.E.M. *Knowledge-based Support for Requirements Engineering*. Journal of Information and Software Technology, Vol. 31, No. 3, April 1989.

[deMarco 78]
deMarco, T. *Structured Analysis and System Specification*. New York:Yourdon, 1978.

[Mylopoulos 86]
Mylopoulos, J. *The Role of Knowledge Representation in the Development of Specifications*. In "Information Processing 86". Kugler, H. J. (ed.) Elsevier Science Publishers B. V., IFIP 1986.

[Nijssen 86]
Nijssen, G. M. *On Experience with Large-scale Teaching and Use of Fact-based Conceptual Schemas in Industry and University*. In Proc. IFIP Conference on Data Semantics (DS-1), Meersman, R. & Steel, T. B. Jr. (eds.), Elsevier North-Holland, Amsterdam 1986.

[Petri 62]
Petri, C. A. *Communication with Automata*. Suppl. to Tech. Rep. RAD C-TR-65-337, Vol. 1, Grifiss Air Force BAse, NY, 1966 (translated from "Kommunication mit Automaton", University of Bohn, Germany, 1962.

[Reubenstein Waters 89]
Reubenstein, H. B. & Waters, R. C. *The Requirements Apprentice: An Initial Scenario*. Proc. Fifth Int. Workshop on Software Specification and Design, May 19-20, Pittsburgh, PA, 1989.

[Sowa 84]
Sowa, J. F. *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley Publishing Company, 1984.

[Tarski 56]
Tarski, A. *Logic Semantics and Metamathematics*. Oxford Univ. Press, 1956.

[Tsalgatidou 88]
Tsalgatidou, A. *Dynamics of Information Systems: Modelling and Verification.* Ph.D. thesis, Dept. of Computation, University of Manchester Institute of Science and Technology, June 1988.

[Wohed 87]
Wohed, R. *Diagnosis of Conceptual schemas.* SYSLAB Report No. 56, Univ. of Stockholm, Sweden, 1987.

[Yeh et al 84]
Yeh, R. T., Zave, P.. Conn, A. P. & Cole, G. E. Jr. *Software Requirements: New Directions and Perspectives.* In "Handbook of Software Engineering", Vick, C. R. & Ramamoorthy, C/ V. (eds.), Van Nostrand Reinhold Company Inc., 1984.

[Zisman 76]
Zisman, M. D. *A Representation of Office Processes.* Dept. of Decision Sciences, Univ. of Pennsylvania, WP 76-1-03, 1976.