

# **A Cinematographic<sup>1</sup> View of Conceptual Specifications**

*V. Lalioti*

Department of Computation

UMIST

P.O. BOX 88

Manchester M60 1QD

U.K.

e-mail address : vali@sna.co.umist.ac.uk

## **I. Introduction**

The development of a large information system is generally regarded as one of the most complex activities undertaken by organisations. The success of this activity is dependent on the use of a pertinent methodology for identifying the requirements on the target system and make sure that the produced system will actually meet these requirements. Thus, the first step in Requirements Engineering, namely the knowledge acquisition step, has the purpose of abstracting and conceptualising relevant parts of the application domain. The knowledge elicited during this first step is then formally specified by the use of conceptual modelling formalisms.

---

<sup>1</sup> The word cinematographic has a Greek origin and consists of two words 'κίνημα' and 'γραφική'. The meaning of the first word is movement and of the second is graphic.

The process of requirements capture results in a conceptual schema which states the desired structure and functionality of the information system, as defined by the users. Until recently this conceptual schema was then used as an input for the next development phase, namely the design. As a result, the user's participation in the early phases of system's development was limited to the task of knowledge acquisition. After that was accomplished, the user had to wait until the final implementation of the system in order to see how it works. The result of this situation has been the ability to test and evaluate the system's behaviour only after the implementation phase, which is late in the software life cycle.

Boehm has reported that although only 6 percent of project cost and between 9 and 12 percent of the project's duration is spent in the requirements phase, it costs between five and ten times more to repair errors during coding than during the requirements phase; and it costs between 100 and 200 times more during "maintenance" [1]. Development and customer organisations could save a lot of time and money if they could detect and correct a fraction of the errors then, rather than later. This task is supported by the process of verification and validation of requirements specifications, which is the third step in the Requirements Engineering process. A definition of the terms verification and validation of software is given in [2], via the following questions :

- Verification: 'Am I building the product right?'
- Validation: 'Am I building the right product?'

It follows from the above that the basic objectives in verification and validation of requirements are to identify and resolve software problems and high-risk issues early in the software life cycle. Verification and Validation activities produce their best results when performed as soon as possible and involve user feedback.

Experiences from the use of visual environments in programming tasks have encouraged researchers in Requirements Engineering to make use of similar techniques, in order to visualise the conceptual specifications. These techniques use static or dynamic graphical displays to visualise conceptual specifications. In the first category fell visual editors used to graphically define the conceptual specifications [3] [4], while in the latter, techniques such as Petri Nets have been used in assisting the activity of validating conceptual specifications [5] [6].

The CineVali<sup>2</sup> approach combines scenarios with animation and visualisation techniques in order to validate the Conceptual Specifications produced within the requirements capture phase. The use of visualisation techniques provides a more dynamic view of the models, than the static one of visual editors, and with a more familiar, to the user, presentation than that of Petri Nets. Multiple 2-D graphical Views, movement and colour are used to provide an indication of the dynamic behaviour of the specifications [7][8][9].

Section II of this article, provides an overview of the requirements engineering and validation methods. Section III provides a historic perspective of the Visualisation techniques and their use in various areas of Computer Science. In section IV, an architecture for visualising and validating conceptual specifications is described in detail. The subsections of this section provide an overview of the conceptual modelling formalisms used, the validation scenarios and the visualisation technique. finally, section V concludes this article.

## **II. Requirements Engineering and Validation**

The term Requirements Engineering has been defined as the systematic process of developing requirements through an iterative co-operative process of analysing the

---

<sup>2</sup> The CineVali project is a fully funded by the CEC under the Human Capital and Mobility program via an Individual Fellowship. The title of the project is : Cinematographic Validation of Conceptual Specifications.

problem, documenting the resulting observations in a variety of representation formats and checking the accuracy of the understanding gained [10]. This definition is based on the premise that the RE process involves aspects of concern along three dimensions: the representation, cognitive and social , as shown in figure 1, adopted from Reference [11].

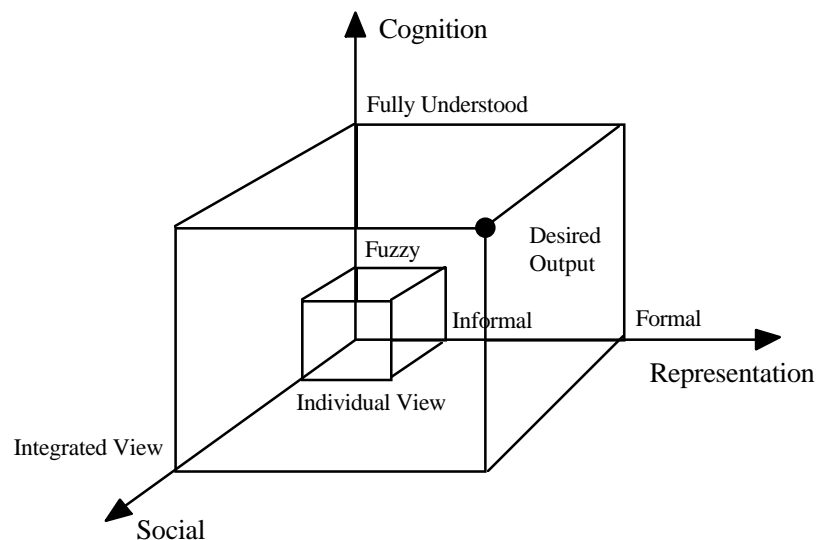


Figure 1. A three dimensional View of Requirements Engineering; based on Reference [11]

Issues of representation range from informal descriptions such as natural language expressions and hypertext to formal conceptual modelling languages. Issues in the cognitive domain concern different orientations of models in terms of understanding the process itself and validating requirements. In the social domain, consideration is given to the complex social process, in which the communication and co-operative interaction between the stakeholders of the requirements, determines the quality of the final product. The process of RE represents a continuous interplay between factors from all three domains.

## A. The representation dimension

### 1. Informal methods

The problem of capturing requirements is, in its initial stages, one of understanding business needs and the needs of users within specific organisations. Methods which address the initial stages fall in two major categories. A set of methods with a strong business orientation which are linked closely with business planning techniques. Examples of these are BSP (IBM's Business System Planning method) and QFD (Quality Function Deployment)[12] [13].

The second set of methods can be thought of as user-centred or socio-technical, i.e., much emphasis is placed not only on user job satisfaction but also on good technical design. These include the Open Systems Approach [14], ETHICS (Effective Technical and Human Implementation of Computer-based Systems) [15].

## **2. Formal methods**

Formal approaches to Requirements Engineering generally form a part of a more downstream set of models, techniques and tools, whose aim is to support the entire development process. With particular reference to requirements specification, there are two key activities that underpin any methodological framework: *environment (enterprise) modelling* and *conceptual modelling*. Enterprise modelling is a structured technique that aims at acquiring a comprehensive examination and documentation of some aspects of the business for a particular purpose. Conceptual modelling is concerned with deriving a schema for the information system that is related to the enterprise schema, and represents the decision of which of the possible products in the problem area will be realised.

This article is more concerned with conceptual modelling. The classification framework proposed by [16] for comparing conceptual modelling techniques, is based on six dimensions: *formal foundation*, *scope*, *level of formality*, *degree of specialisation*, *specialisation area* and *development method*. Of these dimensions the most appropriate to

this article is that of formal foundation, which is concerned with the theory that underlines a particular technique. Five formalisms can be identified on this dimension, namely, *dataflow*, *data-oriented*, *activity-behaviour*, *data-event*, *object-centred*, *rule-based approaches*.

The typical *dataflow model* consists of processing activities and data arcs denoting the flow of data between the activities (e.g., SADT [17]). *Data-Oriented formalisms* concentrate on the modelling of the data that need to be maintained (entity-relationship-attribute [19]). The *Activity-Behaviour formalism* offers the possibility for analysing the behaviour of a system's processing by treating it as a mapping which takes the current state and an incoming stimulus and produces a new state and a response (e.g. Petri-Nets [20]).

In recent years, researchers have attempted to derive formalisms which model static and dynamic objects in a unified manner. One such formalism is *the data-event formalism* as advocated by the ERAE model [21] and the ACM/PCM model [22]. In the *Object-Centred formalisms*, concepts of the world are considered in terms of units known as objects. The behaviour of the universe of discourse is modelled in terms of the creation, modification and manipulation of the perceived objects within that universe of discourse (RML [23], [24] and CML [25]).

*Rule-based approaches* attempt to use knowledge engineering concepts and techniques within a software engineering environment. Examples of this is the work conducted in the RUBRIC project [26], the CIAM method [27] and the TAXIS methodology [28].

## **B. Social Dimension**

A taxonomy of the social assumptions made by the requirements methods is given by [29] and summarised in figure 2. Two meta-narratives namely the "unitary", in which society is

considered to be an organic self regulating whole and the "pluralistic, where society is seen as consisting of distinct groups. Bickerson and Siddiqi in order to classify the requirements methods, have further expanded the modern meta-narrative to produce six meta-narratives.

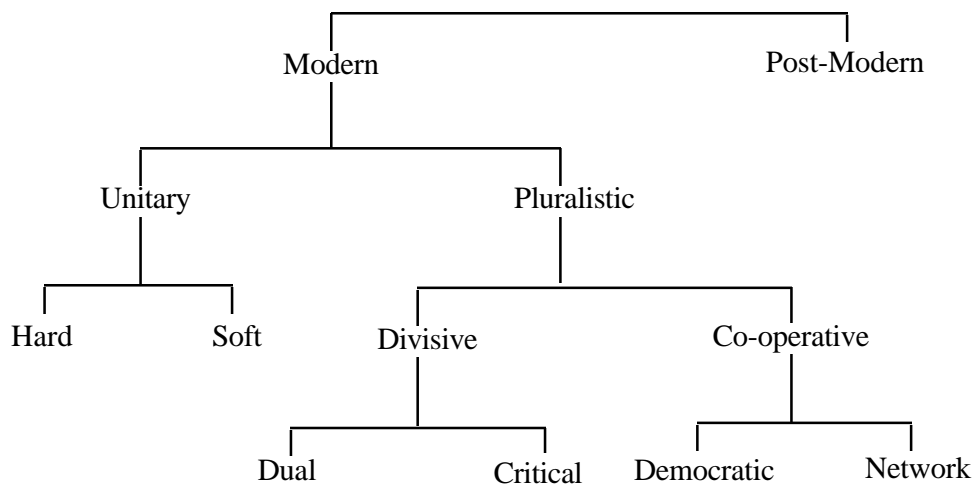


Figure 2. A Taxonomy of the Social Assumptions made by requirements methods

The *Unary Hard* methods make the assumption that the organisational structure is hierarchical, stable and formally definable (e.g. Structured Analysis [17]) and are currently the most commonly adopted in system development. The *Unary Soft* approach assumes that there are differing world views, so it is impossible to obtain a single objective statement of a system's purpose and the systems analyse must facilitate the transformation between the various viewpoints found in the organisation (Soft Systems Methodology [30]).

The *Dualistic approach* assumes society has a fundamental infrastructure that contains a conflict between those who own the means of production and the workers. The *Critical approach* proposes that finding alternatives to the existing social conditions is more important (e.g. Total Systems Intervention [31]). The *Democratic approach* assumes that power lies with the populace and this power can be exercised through representative

forms of management (e.g. Joint Applications Development Method [32]). Finally the network approach has emerged from the study of the sociology of science [33], where both the form and the content of scientific knowledge is seen as a social product. It is common with this approach to provide structures and tools that allow users to build their own systems.

### C. The cognitive dimension

Little is known about how organisations handle product requirements. Anecdotes abound about problems that arise on projects when customer requirements are not fully understood. However, little empirical evidence exists about the way that organisations deal with requirements. Perhaps the two most well known studies of software engineering practice in real organisations were conducted the first by Curtis, Krasner and Iscoe and the other by Lubars, Potts and Richter [34] [35].

The Curtis study had a much wider scope in that it examined not only the requirements process but also many other facets of software development such as project management, implementation and maintenance. From the point of view of requirements practices two of their most significant conclusions confirm most people's intuitions and experiences about requirements for large, complex systems; that accurate problem domain knowledge is critical to the success of a project and that requirements volatility causes major difficulties during development.

<b>Market-Driven Projects</b>	<b>Customer-Specific Projects</b>
<ul style="list-style-type: none"> <li>requirements are sketchy and informal</li> </ul>	<ul style="list-style-type: none"> <li>requirements are voluminous and more formal</li> </ul>
<ul style="list-style-type: none"> <li>use of techniques from manufacturers rather than SE (e.g. QFD)</li> </ul>	<ul style="list-style-type: none"> <li>use techniques from SE</li> </ul>

<ul style="list-style-type: none"> <li>• specification is in the form of a marketing presentation</li> </ul>	<ul style="list-style-type: none"> <li>• specification may be in hundreds of pages of documentation</li> </ul>
<ul style="list-style-type: none"> <li>• not readily identifiable 'customer', developers tend to have less experience in application domain</li> </ul>	<ul style="list-style-type: none"> <li>• make use of domain expertise, developers have in-depth knowledge of domain</li> </ul>
<ul style="list-style-type: none"> <li>• projects rely on consultants for advice on desirable features</li> </ul>	<ul style="list-style-type: none"> <li>• projects rely on in-house personnel</li> </ul>
<ul style="list-style-type: none"> <li>• less structured approaches adopted, task force used in 'brainstorming' sessions</li> </ul>	<ul style="list-style-type: none"> <li>• structured approach following a particular approach</li> </ul>

Figure 3. Comparisons Between Market-Driven and Customer-Specific Projects

The Lubars study concentrated exclusively on requirements issues in an attempt to identify practices that yield better understood and more easily changed requirements and practices that lead to undesirable consequences. For the purpose of exposition projects were divided into two categories: Customer-specific and Market-driven project. Some of the most important differences of the way requirements are interpreted and the methods and nature of the requirements specification adopted, are summarised in figure 3. The results from the study covered several areas, such as requirements definition, specification of the requirements, validation, system evolution and the interaction between requirements definition and project management and planning.

#### **D. Analysis and Validation Tools**

Requirements Engineering consists of the *knowledge acquisition* and the *verification and validation cycle*. The “end product” of this process must be a complete, consistent, valid and formal requirements specification. the verification and validation step has the purpose

of analysing the knowledge captured during the acquisition step. Analysis refers to the activities involved in understanding the acquired knowledge about the domain and is of great importance to the development life cycle, because a review of this knowledge may reveal errors that can be corrected before the requirements phase is completed and thus, reduce the cost of the developed system [1]. The techniques that have addressed verification and validation so far can be distinguished into two major categories: manual techniques and automated techniques.

## 1. Manual techniques

Five relatively simple and easily implemented manual techniques can be identified that provide much valuable information for meeting verification and validation criteria, namely, reading, cross-referencing, interviews, checklists and scenarios.

The *reading* technique is concerned with having someone other than the originator read the specification to identify potential problems. This is particularly helpful if the reader is one of the users or program developers. The *cross-referencing* technique involves constructing cross-reference tables and diagrams in order to clarify the interactions among the different items of a specification. This technique is effective for the consistency and part of the completeness properties of a requirements specification. Both techniques can be cumbersome for large applications. *Interviews* involve discussing the specification with the users and identify potential problems like misunderstandings or blind spots.

Specialised *checklists*, based on experience, of significant issues for ensuring successful software development can be used effectively in combination with other verification and validation techniques. This method is very good in identifying omissions, but it does not offer much assistance in the areas of consistency and testability. *Scenarios* show how the system will work once it is in operation. The most common form of scenarios is the man-computer dialogues. This technique can have good results in determining whether the

system under development will actually behave as intended by the users but is once again appropriate for small-sized applications.

## 2. Automated techniques

The need for computer-assisted verification and validation aroused from the fact that real world applications are very large to be verified and validate by manual or informal methods. Automated approaches to verification and validation *include automated cross-referencing and syntax checking, executable specifications, rapid prototyping and animation or simulation.*

*Automated Cross-Referencing and Syntax Checking* involves the use of a machine-analysable specification language. Once the user requirements have been specified in such a language, they can be automatically analysed for consistency and part of completeness by ensuring proper use of the language's concepts and presence of all the necessary information (e.g. SREM-RSL (Software Requirements Engineering Methodology - Requirements Statement Language) [36]). However, these tools are still somewhat inefficient on very large specifications.

*Executable specification languages* are languages which have a quite natural language syntax with pictorial representation and the additional capability of code generation. The main purpose of such languages is to write requirements specifications in both visual and/or linear form. These specifications can then be executed and tested either directly or indirectly on the development platform or even on the target system platform. (e.g. [37]). Rule-based languages is another style of executable specifications (e.g. [38]).

*Prototyping* is a term that is used not only for specifying a particular verification and validation technique but also for identifying a new approach to systems development which emerged due to problem that the traditional waterfall model exposes. Rapid

prototyping involves the fast construction of a prototype version of a system in order that it may be evaluated by a customer or end-user and subsequently refined in the light of the feedback generated during this evaluation. A prototype is a mechanically processable and executable description of a simplified model of a proposed software system. The use of prototyping as a verification and validation technique consists of constructing a prototype of the system at the specification level. This prototype is then demonstrated to the users who evaluate its actual behaviour against its expected behaviour (e.g. Rapid Iterative Production Prototyping [39], PROM [40], Software Storming [41]).

*Animation and Simulation* techniques have been mainly used for visualising programs and algorithms by creating graphical snapshots and movies correlated with the program's actions. Animation of a specification is the process of providing an indication of the dynamic behaviour by walking through a specification fragment in order to follow some scenario of interest. Most of the earlier approaches that use animation and simulation are based on the Petri-Net formalism (e.g [42], [43]).

### **III. Visualisation - A Historical Perspective**

In the past decade the rapid decline of graphics-related hardware costs has resulted in the proliferation of powerful workstations and high resolution graphic displays in most information systems environments. This development has made possible the introduction and effective use of visual environments. Visual and iconic environments have proved to be highly beneficial for human/computer communication in general and for programming, in particular.

#### **A. Pictures versus text**

It is commonly acknowledged that the human mind is strongly visually oriented and that people acquire information in a significantly higher rate by discovering graphical relationships in complex pictures than by reading text.

- Random vs. sequential access. Our eyes provide instant, random access to any part of a picture, and to detailed and overall views. Text on the other hand is sequential. When we scan paragraphs, headlines and material in bold type to speed up text search, we really acting in the "pictorial mode" to overcome this limitation.
- Concrete vs. abstract. The use of objects from the real world in pictures that are used to illustrate abstract ideas makes the ideas simple to think about, since the apparatus we use to assess our physical surroundings can be applied to any picture.

### *Dimensions of expression*

Text is a one-dimensional stream of words. Pictures are much richer, providing as they do three (or more) dimensions along which to lay out information and the opportunity to use a host of properties borrowed from the physical world, such as shape, size, colour, texture, direction, and distance. Since the "language" of pictures is richer than that of text, the encoding of each piece of information can be more compact, with a resultant decrease in decoding time.

### *Transfer rate*

Comparisons of pictures and text make it clear that pictures generally transfer information faster than text: Both accessing and decoding is more efficient. The human sensory system

is set up for "real-time image processing"; text processing is much more recent phenomenon.

## **B. Visual Environments**

Visual and iconic environments have proved to be highly beneficial for human/computer communication in general and for scientific data representation [44] [45], performance evaluation of parallel/distributed applications [46] and programming, in particular.

A program is a complex, abstract object. It includes many components with many different attributes that are interrelated in many different ways. When we reason about a program, we think about all these aspects in a fairly unstructured, random fashion - our thoughts run freely over large parts of it. Visual environments which explore the use of pictures for all phases of the programming process can be divided into two categories, namely *visual programming* and *program visualisation* environments.

### *Visual Programming*

Visual programming is the use of various two-dimensional or diagrammatic notations in the programming process [47]. As defined in [48] a visual programming language is a language which uses some visual representations (in addition to or in place of words and numbers) to accomplish what would otherwise have to be written in a traditional textual programming language. Visual programming languages, that can be found in the literature, differ significantly from each other in approaches, design philosophy and appearance, since they have come into existence from varied backgrounds and directed for different areas of interest. Examples of visual programming languages in the first category are languages for office and business automation [49], for automatic programming [48], for algorithmic programming [50] and for database queries on relational [51], object-oriented [52] and knowledge bases, for visualising the structural aspects of complex software

designs [53], for maintaining software [54], for supporting conceptual modelling [55] and for the design of software by reusability [56].

### *Program Visualisation*

Program Visualisation uses the technology of interactive graphics and the crafts of the graphic design, typography, animation and cinematography to enhance the presentation and understanding of computer programs [57]. Program Visualisation is used for designing, developing and debugging programs, or monitoring their performance. Numerous visualisation environments have been developed addressing different areas of interest; for graphics interface development [58], for visualising concurrent processes [59] teaching or research. *Program animation* is a form of program visualisation that is concerned with dynamic and interactive graphical displays of a program's fundamental operations. A few algorithm animation systems have been developed and used for a variety of applications; Balsa [60], Zeus [61], [62].

Visual and Iconic systems have been used apart from programming in many other areas of computing. For example in the area of Computational Geometry, the geometric algorithms are often easiest to understand visually, in terms of the geometric objects they manipulate [63]. Visualisation systems have also been used in presenting graphs [64].

## **IV. An Architecture for Visualising Conceptual Specifications**

The feasibility of implementing a large information system is dependent on the communication and understanding among the actors of the system (i.e. managers, developers, etc.). Accordingly, a crucial factor to the success of the validation process is the level of support provided for the interaction between the people involved. Support in this direction has traditionally been provided by system development methods in terms of informal techniques such as structured walkthroughs or more formally by the development

of prototypes. Whilst both techniques have their merits, the extent to which an informal approach can be applied to large complex specification is questionable. Furthermore, the use of prototyping requires a developer to take certain design decisions which fall firmly in the implementation domain.

In order to avoid these shortcomings a more appropriate technique namely that of visualisation is put forward in the CineVali approach. Visualisation has been applied successfully in programming environments in order to provide an indication of the behaviour of the program. In the context of conceptual specifications, visualisation involves the animation of the behaviour of a system and a visual interface reflecting the results of events upon the graphical - and where appropriate the textual - components of the specification [8].

The advantage of visualisation over prototyping is that design decisions will not have to be made prematurely during Requirements Engineering when things are still vague. A requirements specification is likely to change many times before proceeding to design and visualisation should help in deriving a succession of specification which are increasingly closer to end users' perceptions about the application domain.

The CineVali approach described in this article combines scenarios with animation and visualisation techniques to validate and visualise the conceptual specifications produced within the requirements capture phase. Three are the main components of the system for Visualising Conceptual Specifications, namely the Repository, the Validation Engine and the Cinematographer as shown in Figure 4. The Repository is common for storing the Metamodels, the Models and finally the instances of Models. The Validation Engine uses this information in order to form scenarios which will investigate the dynamic behaviour and the causal relationships between the structural and dynamic components of the specification of an information system. The Cinematographer will then receive this

information in order to create and manipulate different Views of the information system in two levels of detail, at the model level and at the instance level.

Each scenario starts from one of the conceptual models and walk through a specification fragment, exploring the interrelationships between the three models. However, the way this information is reported back to the developer is very crucial for understanding the dynamic behaviour of the specifications. The technique is that of Cinematography. In particular, multiple graphical Views, movement and colour are used to visualise the dynamic behaviour of the system under development.

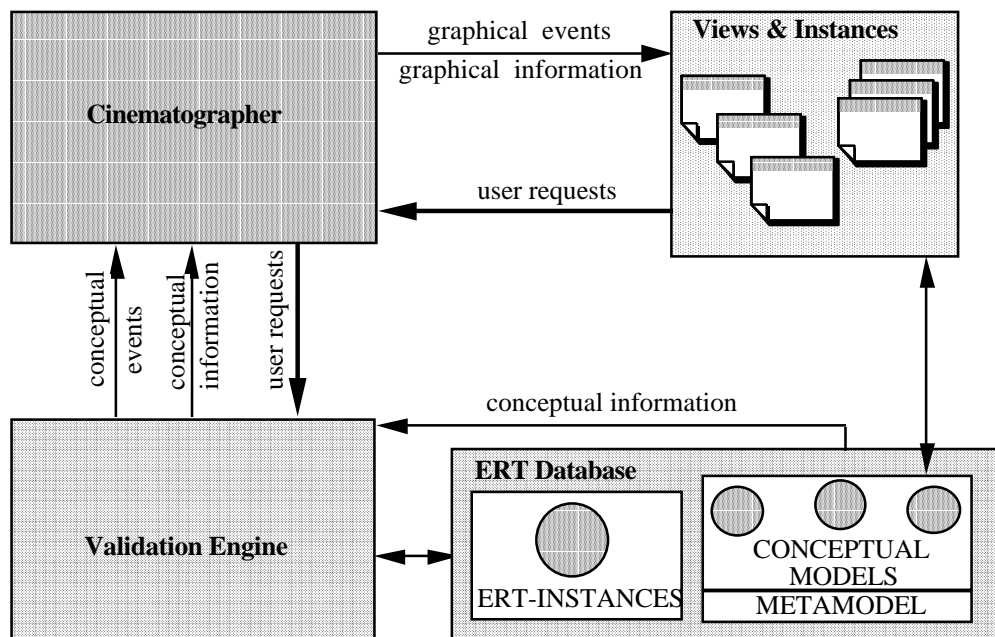


Figure 4. An Architecture for Visualising Conceptual Specifications

The sections that follow, describe each component of the CineVali system separately. The first section provides a summary of the conceptual modelling formalisms used and the conceptual repository. Section B provides an overview of the Validation engine and the different scenarios used to visualise the conceptual models and the last section describes the different views provided by the Cinematographer.

## A. The ERT Database

The conceptual models and metamodels are stored in a common repository. The metamodelling mechanism is used to define causal relationships between the models and a set of fundamental operations which manipulate the instances of the models. The conceptual modelling language which is used, for the task of application domain modelling has been developed within the TEMPORA<sup>3</sup> and ORES<sup>4</sup> projects and provides mechanisms for three conceptual views namely a *structural* view, a *dynamic* view and an *active* view. These three views are represented by the ERT, PID and CRL models respectively. Details of these models can be found in [65] [66].

---

<sup>3</sup> The TEMPORA project is a collaborative ESPRIT project between: BIM, Belgium; Hitec, Greece; Imperial College, UK; LPA, UK; SINTEF, Norway; SISU, Sweden; University of Liege, Belgium and UMIST, UK. SISU is sponsored by the National Swedish Board for Technical Development (STU), ERICSSON and Swedish Telecomm.

<sup>4</sup> The ORES project is a collaborative ESPRIT project between: 01 Plifororiki, GR; Clinica Puerta de Hierro, E; Information Dynamics, GR; Royal Institute of TEchnology, S; UMIST, UK.

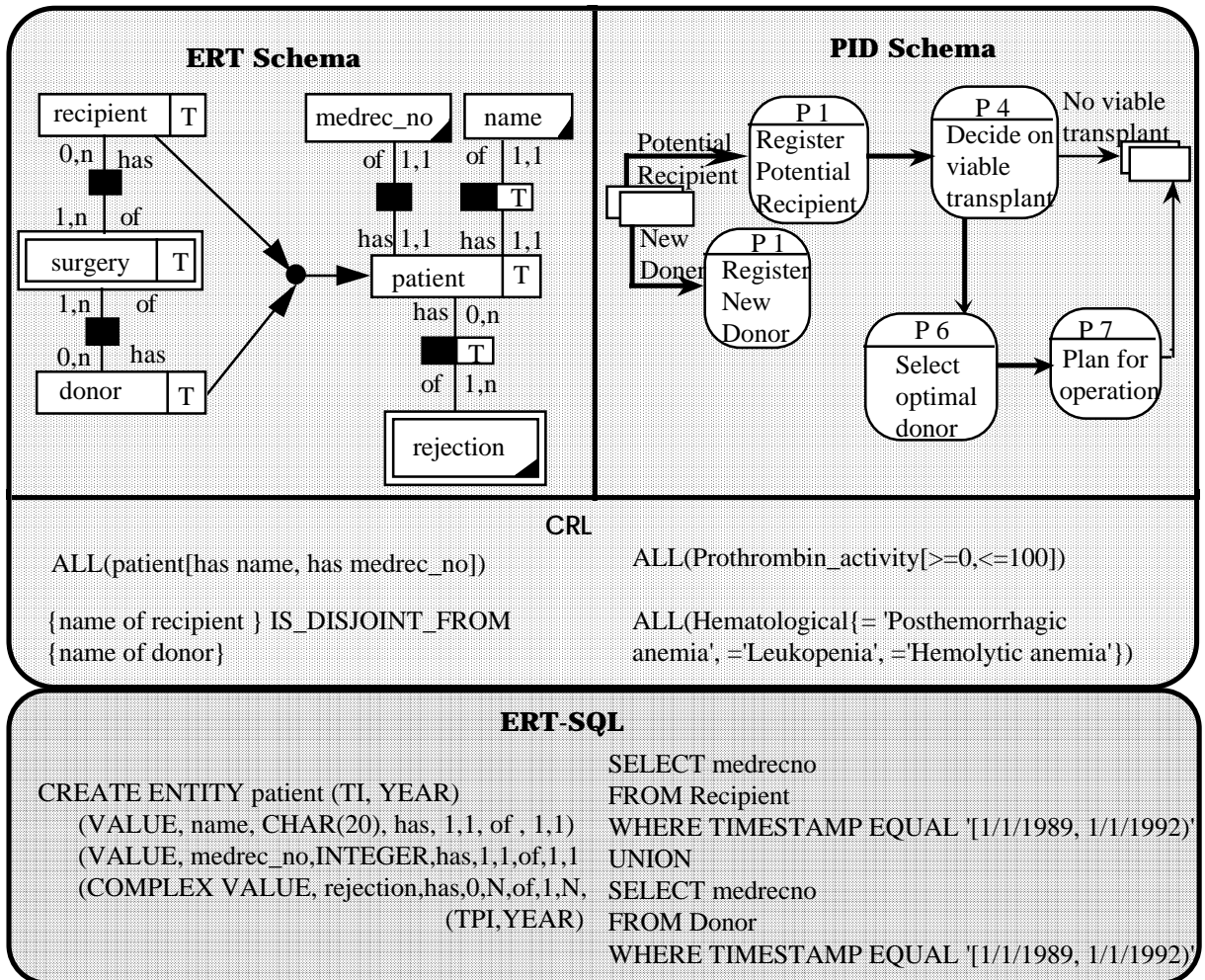


Figure 5. Example Conceptual Specification

Apart from the above mentioned models, a query language which operates on ERT objects has been defined. The ERT-SQL language provides the means of manipulating ERT data, i.e. queering and retrieving data. The language is based on the ERT Algebra [67].

An example of the three models is given in Figure 5 and it is part of the ORES Case Study. In this figure, two layers can be distinguished, namely the conceptual layer, consisted of ERT, CRL and PID, and the ERT database layer consisted of the ERT-SQL

data and schema definition language groups together with the data manipulation language group which is used in the PID and CRL models .

The major reason behind errors detected by validation, is limited knowledge and ad hoc use of the selected models [68]. The problem is compounded when, as is almost always the case, more than one modelling technique is used (for example using different conceptual models for specifying structural and dynamic components). Browsing through each model separately, is not of much help for the developer. Therefore, the models should be interrelated and these relationships should be formally defined in order to check for any contradiction or redundant specification.

Integration of different conceptual models can be effectively achieved through the use of metamodelling [69]. A *metamodel* is a conceptual model of a modelling technique. A metamodel is specification-independent and time-invariant and contains all the necessary knowledge about the constructs and the semantics of the language/model used for specifying requirements during the Requirements Engineering process. In essence, it provides all the building blocks needed for describing an application model that pertain to a given modelling formalism.

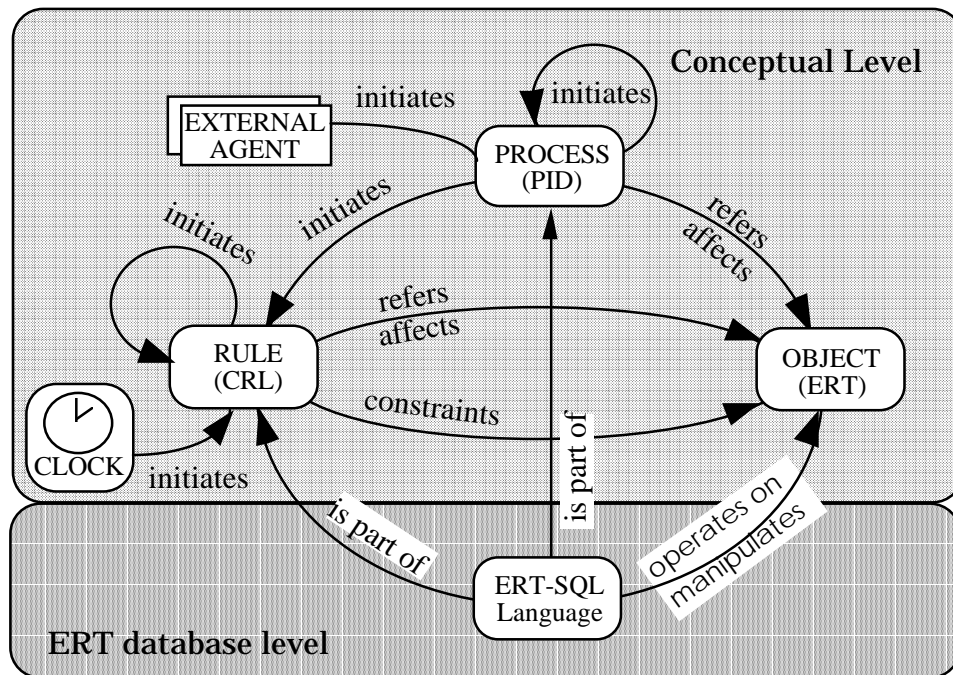


Figure 6. Interrelationships of the Conceptual Models

The three conceptual modelling formalisms used in CineVali, are strongly interrelated and this interconnection is explicitly recognised and represented according to the metamodels of these formalisms. The relationships between the three conceptual formalisms is depicted in figure 6. In particular, the conceptual rule language can be used to constraint and refer or affect ERT objects. A CRL rule can be initiated by a process, or another rule, or the system clock. A process could also refer to or affect an ERT object and can be initiated either by another process, or an external agent. On the other hand, the ERT-SQL language is a data manipulation language which operates on ERT objects. ERT-SQL statements are part of the process and rule body.

## B. The Validation Engine

The *Validation Engine* is the basis of the system and is responsible for gathering the necessary conceptual information from the repository, from the validation scenarios and

create and manipulate instances of ERT objects. The Validation engine also reports, all the interesting conceptual events and information to the Cinematographer to be visualised.

The CineVali approach takes into account the very important relationships between the conceptual modelling formalisms described earlier. The main idea is to follow these interrelationships in order to demonstrate in a very detailed level the system's behaviour. The idea of using scenarios as a means of validating a specification is not new. Actually, this kind of validation support has been extensively used in the past, but its application has been mostly of informal nature. Moreover, scenarios are usually performed manually, the result of which is that the method can only be used for small-sized applications where the bulk of information allows the user to trace the specification without much effort.

The main idea behind the forming of scenarios lies in the fact that it would be desirable to support reasoning about a developed specification. Scenarios aim at demonstrating how the system will work once it is in operation. The most common form of scenarios that has been applied to specifications is the human-computer dialogues. This technique can have good results in determining whether the system under development will actually behave as intended by the users, as far as the interface between the system and its users is concerned.

Scenario formulation, however, can be much more general than this. Many different kinds of queries can form the basis of specific scenario formulation. In [70], a way of applying scenario formulation to a specification developed using the ERT, PID, and CRL is given. Questions like what if X, that is how the system will react if X happens, are processed by the system according to the requirements that have been captured by the developer, and a scenario is produced. Such a scenario actually exhibits the kind of information that it is present in the conceptual schema, as well as the potential behaviour of the system once it is operational. The developed scenario is then examined by the users (or compared to a similar one prepared by them). If significant differences are found, then the captured requirements do not reflect the actual wishes and intentions of the users and they have to

be changed. An earlier system which explored the same initial idea, uses scenarios which are prolog goals together with a semantic prototype to simulate the behaviour of the system under development [71]. If a scenario fails, the tracing of the execution usually reveals the problem.

The results from applying scenarios in all the different phases of requirements engineering are very encouraging and in most cases scenarios are enhancing the communication between the user and the analyst. As Wexelblat asserts "it is clear that the effort required to converse via scenarios will be more than repaid by drastically lowering the cost of failed prototypes" [72].

## **1. The CineVali Scenarios**

The main innovation of the CineVali approach is that the scenarios are formal and automatically generated, while at the same time their representation is one with which both the user and the analyst are familiar. The scenarios could be both process and explanatory scenarios and are more concerned with the way the system will achieve the user objectives and goals. The approach does not address the managerial aspects of the requirements engineering process.

In particular, a set of questions could be asked all having as a starting point one or more concepts of one or more of the conceptual models used:

- (1) *What if event X* arrives at the system, which attempts to show which processes will be initiated if event X arrives at the system. This will be the main means of demonstrating system's behaviour.
- (2) *What are the operations performed by external agent A*, which attempts to show what will be the tasks of an external agent. This could clarify the operations performed by different groups of users.

- (3) *Which processes or rules are initiated by process P*, which gives all the processes or rules that could be triggered by the execution of process. This could provide information on the general rules which apply to the organisation and their relationship to specific processes. It can also be used in order to follow an execution path and perform a walkthrough of a specification fragment.
- (4) *Which rules constraint object E*, which gives all the rules that are related to object E. This could be used by the user and analyst to check whether constraints put on objects are valid, or too strict or too weak.
- (5) *Which particular object instances violate rule R*, which after checking the triggering conditions of a rule, gives a set of instances which will violate the rule. This could be used to validate the triggering conditions of the rules.
- (6) *Which object instances satisfy the preconditions of process P*, which in a very similar manner with the above question provides instances which will satisfy the precondition of a process. This could be used to either automatically execute a process scenario or to provide fragmentary explanatory information about a process.
- (7) *How does X affect Y*, which will determine the kind of effect of a certain specification item on other specification items (e.g. how a process affects certain entities). This could be used to check interrelationships and side effects between two or more specification items.
- (8) *Which rules are initiated by rule R*, which in a similar manner with the above question will demonstrate an execution path of rules.
- (9) *Why X*, which asks the system to explain how it reached the situation X, by performing backward reasoning (starting from X it has to trace all facts that led to it).
- (10) *When rule R will be initiated*, which answers questions concerning rule triggered by temporal events. This could be helpful in checking tasks that are performed on specific times within the organisation's structure.

- (11) *Which processes or rules affect a particular object*, which will determine the effect rules and processes have on an object. This is particularly useful when checking the way objects are modified and changed by the system.

Most of these questions could be directly used to generate explanatory scenarios, while others could be combined together to form process scenarios. All of these scenarios are formal, since their source is the conceptual modelling formalisms in which the specifications are described. The level of detail varies between them. In fact, the same scenarios could be seen at a different level of detail, thus allowing the expert and non-expert user to understand without being burdened by more than the necessary information. The process scenarios could be either passive or interactive. The way this is achieved is by accepting some assumptions and performing a passive execution of a scenario, or asking for more details from the user to perform a more controlled execution. This type of scenario is therefore reactive and could be changed along its execution, in order to observe different aspects of the captured requirements.

Scenarios can be expressed in three different levels of abstraction. The first very important abstraction is that of the Conceptual and ERT database level. Secondly, scenarios could be restricted only to one conceptual model or could be intermodel scenarios, to allow intermodel checking of the specifications. Another level of abstraction is the one concerned with explanatory scenarios or with process scenarios. Figure 7 summarises these abstraction levels, with the less detailed ones closer to the conjunction of the three axes. In what follows, scenarios are presented by a sequence of questions and include all the possible levels of detail and all relationships between the different models.

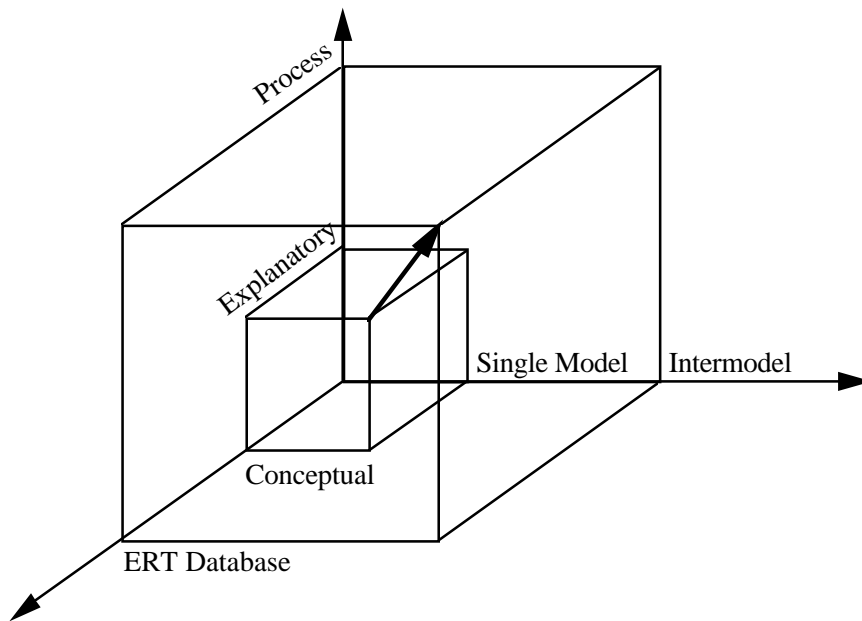


Figure 7. Levels of abstraction of CineVali Scenarios

### Scenario 1

This scenario has the PID model as the starting point and the first question asked is: what if event X arrives at the system (question 1). Then the processes that would be initiated by that event are found and the user could choose one to follow. In the highest level of detail the user should provide all the necessary instances to satisfy any preconditions the process has, that is the question "What ERT instances will satisfy the preconditions of the selected process" is asked (question 6). Then the process will be executed and the ERT objects affected by it will be presented (question 7). Also the rules or processes which are initiated by the output events of the executed process are given to the user (question 3), who could either terminate the scenario or continue by selecting a process or a rule from the new set to follow.

### Scenario 2

This scenario is concerned with the user groups performing the tasks, and checks whether tasks are assigned to external agents properly, or whether the user groups performing the actual tasks within the organisation would be influenced by the system. The scenarios start with the question : what processes are performed by external agent A (question 2). This question will result a set of processes that could be initiated by a particular external agent. There are two options, namely choosing one process and then use scenario 1, or terminate the scenario.

### Scenario 3

This scenario is concerned with the rules and their validation. The scenario starts with the question: how is rule R initiated (question 9), and continues by asking which particular object instances violate rule R (question 5). Then the scenario checks which ERT objects are affected or constrained by this rule (question 7). The scenario proceeds by finding the rules that could be initiated after the execution of the first rule (question 8), and the user can choose which one to execute next.

### Scenario 4

This scenario validates the temporal aspects of a specification by asking the question: when is rule R initiated (question 10). The temporal event that would trigger the rule is presented to the user. After that the scenario gives two options to the user, either to check the ERT objects affected by the rule (question 7) and the rules initiated by it (question 8), thus following a similar path with scenario 3, or to terminate the scenario at that point.

### Scenario 5

This scenario has as starting point the ERT model and asks the question: what rules constraint object E (question 4). Therefore, with this scenario the user is presented with a

set of rules that apply to a particular ERT object and can assess their correctness in terms of how weak or strong the constraints are.

### Scenario 6

We use the same number for denoting a set of scenarios rather than a specific scenario. All the scenarios of this set use backward reasoning to answer the question : Why X, that is why process X is initiated, why rule X is triggered or why entity X is affected (question 9). This set of scenarios involves finding by backtracking the reason and then creating a scenario of type 1, 2, 3 or 4 to demonstrate to the user the way the system arrive in situation X.

## **C. The Cinematographer**

The *Cinematographer* receives conceptual events and information and translates them into graphical ones. Therefore it has methods to create graphical objects which correspond to a process, an ERT object or a CRL rule given their names, types or description, and a set of graphical events that correspond to the interesting events reported by the Validation Engine. It is also responsible for generating a different View for every conceptual model and for updating the Views according to the graphical events reported. Instances should also be visualised when this is necessary.

### **1. Graphical Objects**

The graphical objects of the Cinematographer, are one for every concept of the models. For the ERT model, there are graphical objects that correspond to an entity, value class, complex object, relationship and time stamped classes. For the PID model, there are

graphical objects for representing a process and a Trigger and for the CRL there is a textual object that presents a rule.

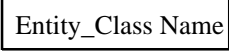
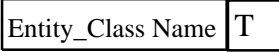
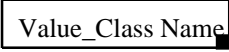
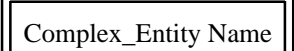
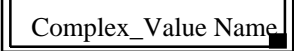


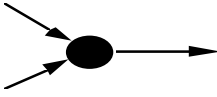
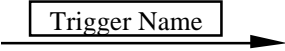
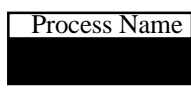
Conceptual Objects	Graphical Objects
Entity class	
Time stamped entity class	
Simple value class	
Complex entity class	
Complex value class	
Relationship	
Timestamped Relationship	
ISA relationship	
Trigger	
Process	

Figure 8: Concepts and their Corresponding Graphical Objects

Figure 8 lists all concepts of the models and their corresponding graphical objects. For every graphical object, the Animator has a method to create it (i.e. create process object P1).

## 2. Graphical Events

The Cinematographer also translates the events reported by the Validation Engine to graphical events which can be understood by the Views. For example the event "process P1 is initiated" is translated into "draw process object P1", or the event "Entity Class Employee is affected by process P1" is translated into "Colour Employee with the colour

of P1". When all the graphical objects of an interesting event have been created, the corresponding graphical event is send to the Views to process.

### **3. The Views**

A View consists of a window, a set of methods to respond to graphical events sent by the Animator. Also a View has a set of general methods in order to automatically re-size or re-draw its window either at user's request or whenever is needed.

A single View can be used to animate all the information. However, this will result an overloaded View and may confuse the developer by reporting too much information at once. In our approach three different Views can be used, one for each conceptual modelling formalism. Each View has a method for reacting in occurrences of events and for displaying the relevant information. There exist three views, namely, the ERT, PID and CRL Views.

## **V. Conclusions**

The process of information systems development can be viewed as a sequence of model-building activities. The quality of each set of models depends largely on the ability of a developer to extract and understand knowledge about an application domain which needs to be acquired from a diverse user population. This implies that developing an information system and in particular capturing requirements is a knowledge intensive activity which requires appropriate mechanisms for knowledge elicitation, knowledge representation and knowledge validation about the modelled application domain.

We have presented a way of validating a specification developed using the ERT, PID and CRL models by animating information captured by the three models in a uniform way. A first version of the prototype has been implemented and used on a number of case studies.

However, our research efforts are focusing in providing a visual environment for validating and symbolically executing conceptual specifications of an information system. Therefore, the visual system envisaged, will graphically animate conceptual prototypes. The term *conceptual prototype* is used to emphasise the difference between executing the specifications and creating a prototype of the information system. In conceptual prototyping there is no need to make any design decisions prematurely. By animating the conceptual prototype, all the actors of the requirements engineering process can understand and inspect the behaviour of the system under development as early as possible in the requirements engineering process.

## References

- [1] Boehm, D.G., Software Engineering Economics, Prentice Hall, Inc., 1981
- [2] Boehm B.W., Verifying and Validating Software Requirements and Design Specifications, IEEE Software, Vol. 1, No. 1, January 1984.
- [3] Auddino A., Amiel E. and Bhargave B., Experiences with SUPER, A Database Visual Environment, Database and Expert System Applications, Proceedings of the International Conference DEXA 91, Berlin, August 21-23, 1991
- [4] Parent C. and Spaccapietra S., ERC+: An Object-Based Entity Relationship Approach, in P. Loucopoulos and R. Zicari (Eds), Conceptual Modelling , Databases, and CASE, pp. 69-86, J. Wiley, 1992
- [5] Kramer J. and Ng K., Animation of Requirements Specification, SPE, 18, 8: 749-774, 1988
- [6] Tsalgatiidou A., Dynamics of Information Systems Modelling and Verification, UMIST, PhD thesis, Manchester, UK., 1988
- [7] Lalioti V. and Loucopoulos P., Visualisation for Validation, Fifth Conference an Advanced Information Systems Engineering, CAiSE, June 1993.
- [8] Lalioti V. and Loucopoulos P., Visualisation of Conceptual specifications, Information systems Journal, Vol. 19, April 3, 1994.
- [9] Lalioti V., Jasmine: A cinematographic representation of algorithms, University of Crete, Iraklion, GREECE, 1990.
- [10] Bubenko J., Rolland C., Loucopoulos P., DeAntonellis V., Facilitating "Fuzzy to Formal" Requirements Modelling, IEEE International Conference on Requirements Engineering, Colorado, pp. 18-22, April, 1994
- [11] Pohl K., The three dimensions of requirements engineering: A Framework and its applications, Information systems Vol. 19, No. 3, pp. 243-258, 1994

- [12] Daetz D., QFD: a method for guaranteeing communication of the customer voice through the whole product development cycle, (Ed.) IEEE International Conference on Communications, Boston, IEEE, pp. 1329-1333, 1989
- [13] Sullivan L. P., Quality Function Deployment, Quality Progress, 19(6), pp. 39-50, 1986
- [14] Eason K., Information Technology and Organisational Change, Taylor and Francis, 1990
- [15] Catterall B.J., The HUFIT functionality matrix, D. Diaper, D. Gilmore, G. Cockton & B. Shckel (Ed.), INTERACT'90, Cambridge, IFIP, pp. 377-381, 1990
- [16] Roman G.C., A Taxonomy of Current Issues in Requirements Engineering, IEEE Computer, Vol. 18, No. 5, April 1985
- [17] DeMarco T., Structured Analysis and System Specification, Prentice-Hall, Englewood Cliffs, N.J., 1978
- [18] Chen P.P., The Entity-Relationship Model - Towards a Unified View of Data, ACM TODS, Vol. 1, No. 1, pp. 9-36, March 1976
- [19] Falkenberg E., Concepts for Modelling Information, Modelling in Database Management Systems, (Ed.) Nijssen G., North Holland, Amsterdam, 1976
- [20] Peterson J.L., Petri-Nets, Computing Surveys, Vol. 9, 1977
- [21] Dubois E., Hagelstein J., Lahou E., The ERAE Model: A Case Study, Information Systems Design Methodologies: Improving the Practice, Proceedings IFIP WG 8.1 Working Conference, Noordwijkerhout, The Netherlands, 5-7 May, 1986
- [22] Brodie M.L., Silva E., Active and Passive Component Modelling (ACM/PCM), Information System Design Methodologies: A Comparative Review, (Eds.) T.W. Olle, H.G. Sol and A.A. Verrijn-Stuart, North Holland, 1982.
- [23] Greenspan S.J., Borgida A. and Mylopoulos J., A Requirements Modelling Language and its Logic, Information Systems, Vol. 11, No. 1, 1986, pp. 9-23.
- [24] Tsai J.J.P., Weigert T., Jang H.C., A Hybrid Knowledge Representation as a Basis of Requirements Specification and Specification Analysis, IEEE Transactions on Software Engineering, Vol. 18, No. 12, December 1992.

- [25] Meirlaen E., Domange P., CML: A Conceptual Modelling Language for Software Engineering, B.I.M. Research Report, October 1985.
- [26] Loucopoulos P., The RUBRIC Project - Integrating E-R, Object and Rule-based Paradigms, Workshop session on Design Paradigms, European Conference on Object-Oriented Programming (ECOOP), July 1989, Nottingham, UK.
- [27] Gustafsson, M.R., Karlsson, T., Bubenco, J.A., A Declarative Approach to Conceptual Information Modelling, in 'Information System Design Methodologies: a comparative Review', (Eds.) T.W. Olle, H.G. Sol and A.A. Verrijn-Stuart, North Holland, 1982.
- [28] Mylopoulos J., Nernstein P.A., Wong H.K.T., A Language Facility for Designing Database Intensive Applications, ACM Transactions on Database Systems, Vol. 5, No. 2, June 1980
- [29] Bickerson M.J. and Siddiqi J., The Classification of Requirements Engineering Methods, IEEE, January 1992, pp. 182-186
- [30] Checkland P. and Scholes J., Soft Systems Methodology in Action, John Wiley, 1990
- [31] Flood R.L. and Jackson M.C., Creative problem solving: Total systems intervention, Wiley, Chichester, 1991
- [32] Andrews D.C., JAD: A Crucial Dimension for Rapid Applications Development, Journal of Systems Management, March 1991
- [33] Latour B., Science in Action: How to follow scientists and engineers through society, Open University Press: Milton Keynes, UK, 1987
- [34] Curtis B., H. Krasner and N. Iscoe, "A field study of the software design process for large systems", Comm. ACM 31(11): 1268-1287, 1988
- [35] Lubars, A Review of the state of the Practice in Requirements Modelling, IEEE, January 1992, pp. 2-31.
- [36] Alford M.W., A Requirements Engineering Methodology for Real-Time Processing Requirements, IEEE Transactions on Software Engineering, Vol. SE-3, No. 1, January 1977.

- [37] Henderson P., Functional Programming, Formal Specification and Rapid Prototyping, IEEE Transactions on Software Engineering, Vol. SE-12, No. 2, February 1986, pp. 241-249
- [38] Kiper J.D., Structural Testing of Rule-Based Expert Systems, ACM Transactions on Software Engineering and Methodology, Vol. 1, No. 2, April 1992, pp. 168-187
- [39] Goodwin C., Rapid Reaction, Computing, 21 January 1993
- [40] Peltu M., Spiral of success, Computing, 28 January 1993.
- [41] Jordan P.W., Keller K.S., Tucker R.W., Vogel D., Software Storming, Combining Rapid Prototyping and Knowledge Engineering, IEEE COMPUTER, May 1989, pp. 39-48
- [42] Dahler J., Gerber P., Gisiger H.P., Kundig A., A Graphical Tool for the Design and Prototyping of Distributed Systems, ACM SIGSOFT, vol.12, No. 3, July 1987
- [43] Tsalgatidou A. and Loucopoulos P., Rule-based behaviour modelling: specification and validation of information systems dynamics, Information and Software Technology, Vol. 33, No. 6, July/August 1991, pp. 425-432
- [44] Pickover C.A. and Stein A., The Scientific Visualisations Laboratory at the IBM T.J. Watson research Center, Research Report, RC 14629 (#65459) Computer Science, 5 November 1989.
- [45] Appino P.A. and Farrell E.J., A Visualisation Tool for the Scientist/Engineer, Research Report, RC 15445 (#68407) Computer Science, 1 October 1990.
- [46] Bernstein D., Bolmaricich A. and So K., Performance Visualisation of Parallel Programs on a Shared Memory Multiprocessor System, Research Report, RC 14635 (#64061) Computer Science, 1 September 1989.
- [47] Baecker R.M., An Application Overview of Program Visualisation, Computer Graphics, 20, 4: 325, 1986.
- [48] Shu N.C., A Visual Programming Language Designed for Automatic Programming, 21st Hawaii International Conference on System Sciences (HICSS-21), Hawaii, IEEE, 2 Software Track: 662-671, 1988

- [49] Zloof M.M. A Language for Office and Business Automation, AFIPS Office Automation Conference Digest, AFIPS Press, 249-260, 1980
- [50] Edel M., The Thinkertoy Graphical Programming Environment, COMPSAG, IEEE, 466-471, 1986
- [51] Batini C., Catarci T., Constabile M.F. and Levialdi S., Visual Query Systems Universita degli Studi di Roma La Sapienza, Report, 04.91, March 1991
- [52] Cruz I.F., DOODLE: A Visual Language for Object-Oriented Databases, ACM SIGMOD, pp. 71-80, 1992
- [53] Smart J.C. and Vemuri V., A-Vu: A Visualisation Tool for Complex Software Systems, IEEE, August: 172-182, 1992
- [54] Osborne W.M., Fitting pieces to the maintenance puzzle, IEEE Software, January: 11-12, 1990
- [55] Kangassalo H., Concept-D : A Graphical Language for Conceptual Modelling and Data Base Use, Workshop on Visual Languages, IEEE, 2-11, 1988
- [56] Nierstrasz O., Tschritzis D., de Mey V. and Stadelmann M., Objects + Scripts = Applications, ESPRIT '91, Brussels, 534-552, 1991
- [57] Brown M.H., Perspectives on Algorithm Animation, CHI'88: Human Factors in Computing Systems Washington, D.C., 33-38, 1988
- [58] Clemons E.K. and Greenfield A.J., The SAGE System Architecture: A System for the Rapid development of Graphics Interfaces for DEcision Support, IEEE Computer Graphics and Applications, 5, 11: 38-50, 1985
- [59] Roman G. and Cox K.G., A Declarative Approach to Visualising Concurrent Computations, Computer, October: 25-36, 1989
- [60] Brown M.H. and Sedgewick R., Techniques for Algorithm Animation, IEEE Software, 28-39, January 1985
- [61] Brown M.H., Zeus: A System for Algorithm Animation and Multi-view Editing, DEC Systems Research Center, Research Report, 75, February 28, 1992

- [62] Bentley J.L. and Kernighan B.W., A System for Algorithm Animation: Tutorial and User Manual, AT&T Bell Laboratories, Technical Report Computer Science, 132, January, 1987
- [63] Brown M.H. and Hershberger J., The Second Annual Video Review of Computational Geometry, DEC Systems Research Center, Research Report, 101a and 101b, May 4, 1993
- [64] Mendelson A.O., Declarative Database Visualisation: Recent Papers from the Hy+/GraphLog Project, Technical Report CSRI-285, University of Toronto, June 1993
- [65] Theodoulidis, C., P. Loucopoulos and B. Wangler, A Conceptual Modelling Formalism for Temporal Database Applications, *Information Systems*, 16, 4: 401-416.
- [66] Loucopoulos, P., P. McBrien, F. Schumacker, B. Theodoulidis, V. Kopanas and B. Wangler, Integrating database technology, rule-based systems and temporal reasoning for effective information systems: the TEMPORA paradigm, *Journal of Information Systems*, 1, 2, April: 129-152.
- [67] Karvelis G., An Algebra and a Query Language for the Entity Relationship Time Data Model, MSc. Thesis, University of Manchester, Computation Department, 1993
- [68] Falkenberg, E.D., Information Modelling - Subjective Forever, Database dag, Eindhoven.
- [69] Loucopoulos, P., McBrien, P., Persson, U., Schumacker, F., Vasey, P., TEMPORA-Integrating Database Technology, Rule Based Systems and Temporal Reasoning for Effective Software, ESPRIT Conference, Brussels, November 1990.
- [70] Katsouli, E., Verification and Validation for Rule-based Requirements Specifications UMIST, 1992.
- [71] Loucopoulos P. and Karakostas V., Modelling and Validating office information systems: an object and logic oriented approach, *Software Engineering Journal*, March 1989, pp. 87-94

- [72] Wexelblat A., Report on Scenario Technology, MCC Technical Report STP-139-87, Microelectronics and Computer Technology Corporation, Austin, Texas.

## **Bibliography**

### **[Bubenko et al, 1992a]**

Bubenko J., Rolland C., Loucopoulos P., DeAntonellis V., Facilitating "Fuzzy to Formal" Requirements Modelling, IEEE International Conference on Requirements Engineering, Colorado, pp. 18-22, April, 1994.

### **[Davis, 1990]**

Davis A.M., Analysis and Specification of Systems and Software Requirements, "R. Thayer & M. Dorfman (Ed.), System and Software Requirements Engineering Tutorial, IEEE Computer Society Press, Los Alamitos, California, 1990.

### **[Boehm, 1984]**

Boehm B.W., Verifying and Validating Software Requirements and Design Specifications, IEEE Software, Vol. 1, No. 1, January 1984.

### **[E. P. Glinert, 1990]**

Visual Programming Environments, Applications and Issues, IEEE Computer Society Press Tutorial, Elpraim P. Glinert (Editor), 1990.