Visualisation for Validation

V. Lalioti and P. Loucopoulos

Information Systems Group
Department of Computation
U.M.I.S.T.
P.O. Box 88
Manchester M60 1QD
U.K.

Abstract

Animation is a multiple graphical view of a process in action. Animation has been successfully employed in programming for designing, developing and debugging programs or monitoring their performance.

This paper advocates that many benefits can be accrued from the use of visualisation techniques for the purpose of validating conceptual specifications during Requirements Engineering.

To this end, the paper describes a visualisation system which makes use of three interrelated conceptual models and their metamodel represented uniformly in a repository and an animation algorithm which generates graphical views corresponding to the behaviour of an application domain as specified by the conceptual models.

1 Introduction

The feasibility of implementing a large information system is dependent on the use of a pertinent method for identifying the requirements on the target system and to make sure that the produced system will actually meet these requirements. Requirements Engineering consists of the knowledge acquisition, conceptual modelling and validation cycle. The acquisition step has the purpose of abstracting and conceptualising relevant parts of the application domain. The modelling step is concerned with the formal specification of aspects of the application domain for the purpose of analysis. Validation is the process of investigating a model (in this case an IS specification) with respect to its user perceptions.

The purpose of validation in the development of information systems is to ensure that a specification really reflects the user needs and statements about the application. Its importance is widely recognised by most developers but still there is a lack of formal theory for efficiently carrying out validation [33]. In recent years some quality assurance tools for systems specifications have emerged. They usually fall in two categories: either

one uses design metrics to make judgements of entire designs [11] or one uses heuristics to pin-point single problems [34, 24, 8].

A key factor to the success of the validation process is improvements in communication and understanding among the actors of the system (i.e. managers, developers, etc.). Accordingly, a crucial factor is the level of support provided for the interaction process 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 this paper. 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.

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 paper is organised as follows. Section 2 gives a brief historical overview of the use of visualisation techniques. Section 3 puts forward an architecture for visualising conceptual specifications. Such an architecture has three requirements which are detailed in this section: a set of formal conceptual models, a metamodel which integrates the different viewpoints expressed by the conceptual models and an animation algorithm for handling the behaviour of a system. Section 4 gives an in-depth description of the visualisation tool in terms of the algorithm for forming scenarios, the animator and the graphical views. Section 5 shows an application of the visualisation tool on a fraction of an-industrial size application. Section 6 concludes this paper.

2 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.

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 is the use of various two-dimensional or diagrammatic notations in the programming process [1]. Program Visualisation uses the technology of interactive graphics and the crafts of graphic design, typography, animation and cinematography to enhance the presentation and understanding of computer programs [4].

Visual Programming

As defined in [28] 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. These different approaches however, can be classified into tow broad classes of programming languages. Firstly, languages have been designed to handle visual information, to support visual interaction and allow programming with visual expressions. Secondly, graphical techniques and pointing devices are used to provide a visual environment for program construction and debugging, for information retrieval and presentation and for software design and understanding.

Examples of visual programming languages in the first category are languages for office and business automation [35], for automatic programming [27], for command languages [30], for algorithmic programming [10, 13] and for database queries [2].

Examples of languages in the second category include those for visualising the structural aspects of complex software designs [29], for maintaining software [23], for supporting conceptual modelling [14] and for the design software by reusability [21].

Program Visualisation

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 [9], for visualising concurrent processes [26], for teaching or research [6].

Myers has classified program visualisation systems by whether they illustrate code or data, and whether displays are either passive or static [20]. In addition, dynamic displays are either passive, such as a videotape, or interactive, such as those found in interactive systems running on workstations.

Typical static displays of program code are flowcharts, scooping diagrams, and module interconnections, as well as text itself when enhanced through formatting and typography. Systems have been developed to display one or more such diagrams automatically from programs coded in high-level procedural languages, and to use the diagrams for editing the underlying program. Static displays of program code can be animated automatically by highlighting the appropriate parts as the code runs [25].

Program Animation is a form of program visualisation that is concerned with dynamic and interactive graphical displays of a program's fundamental operations [5]. A few algorithm animation systems have been developed and used for a variety of applications. For example, at Brown University, BALSA was used for instruction in computer sciences courses and for research in the design and analysis of algorithms [7]. London and Duisberg at Tektronix used animated programs for performance tuning and debugging of large systems [18], and at Bell Labs, Bentley and Kernighan exploited animated displays for algorithm research and for generating sequences of static displays of data structures [3].

3 An Architecture for Visualising Conceptual Specifications

3.1 Overview

Experiences from the use of visual environments in programming tasks has encouraged researchers in Requirements Engineering to make use of similar techniques, normally referred to as animation techniques, in assisting the activity of validating conceptual specifications [16, 32].

Animation of a specification is the process of providing an indication of the *dynamic* behaviour of the system by walking through a specification fragment in order to follow some scenario. Animation can be used to determine causal relationships embedded in the specification or simply as a means of browsing through the specification to ensure adequacy and accuracy by reflection of the specified behaviour back to the user.

The major reason behind errors detected by validation, is limited knowledge and ad hoc use of the selected models [12]. 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. The developer needs to have a better understanding of how the models affect each other, in order to check for any contradiction or redundant specification. Therefore, the models should be interrelated and these relationships should be formally defined and used in the validation process.

Integration of different conceptual models can be effectively achieved through the use of metamodelling. 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.

The relationship between application-dependent models and time-invariant metamodels within the context of representing and validating a conceptual specification is shown in figure 1.

In the scheme shown in figure 1 both the metamodel and the conceptual specification are represented within a single repository. In fact a conceptual specification represents an

instantiation of the conceptual model. To this end, an object-oriented representation scheme has been chosen as the most appropriate storage mechanism for the repository [19].

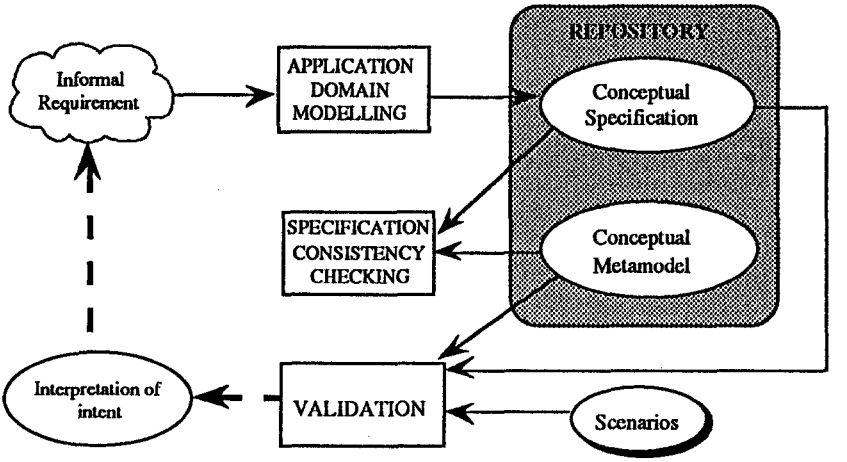

Figure 1. Conceptual Models, Metamodel and Validation

3.2 The Conceptual Models

The conceptual modelling language which is used for the task of application domain modelling has been developed within the TEMPORA project[1] and provides mechanisms for three conceptual views namely a *structural* view, a *dynamic* view and *business rules* view. These three views are represented by the ERT, PID and CRL models respectively. Details of these models can be found in [19]. In summary:

- The semantics of the Entity-Relationship-Time (ERT) model are those of the Entity-Relationship model based on the binary approach (entities, relationships, values, etc.) [22], extended with generalisation/ specialisation hierarchies, complex objects and time modelling. This model is described in more detail in [31].

- The Process Interaction Diagram (PID) model provides graphical notation for the specification of processes and their interaction. This includes both the interaction between the processes at the same level of abstraction and

---

[1] The TEMPORA project is a collaborative 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. The project is partly funded by the CEC under the ESPRIT programme.

the way processes at any level of abstraction relate to their parent process as well as their decomposition. The basic modelling concepts of a PID are: *processes*, *external agents*, *triggers*, *conditions* and *ERT views*.

The Conceptual Rule Language (CRL) model provides the means of specifying the behaviour of an application domain in terms of rules. the model provides facilities for the definition of constraints placed upon the elements of ERT and with the derivation of new information based on existing information. The model also provides facilities for the specification of the conditions under which the actions must be taken, i.e., a set of triggering conditions and/or a set of preconditions that must be satisfied prior to their execution.

An example of the three models is given in figure 2.

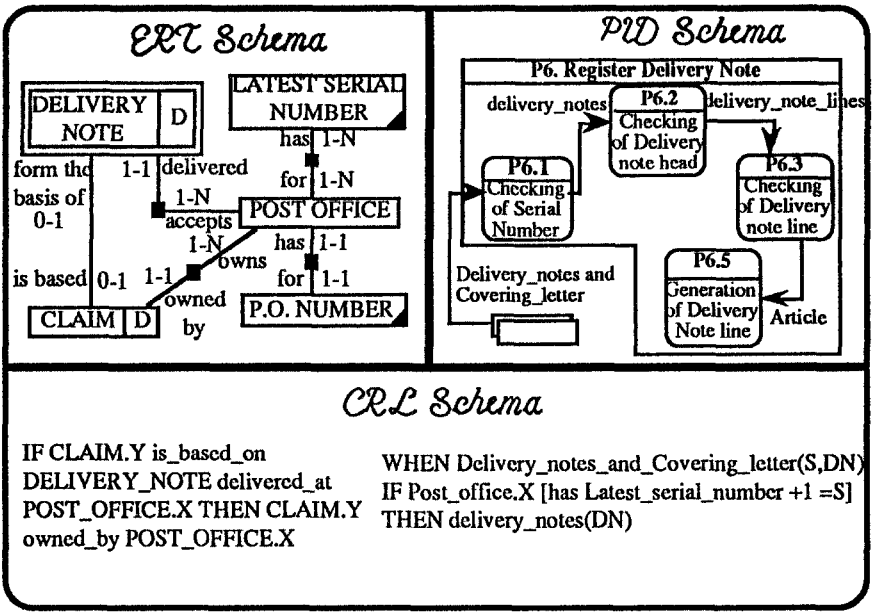

Figure 2. Example Conceptual Specification

3.3 The Conceptual Metamodel

For the conceptual metamodel the metaclass-class mechanism provides the basic structuring mechanism for implementing the metamodel. In this way, a metamodel is represented at the metaclass level, whereas an application model is represented at the class level. A top level view of the metamodel is shown in figure 3. A key element of this view is that the three components of the metamodel which correspond to the ERT, PID and CRL models are fully integrated.

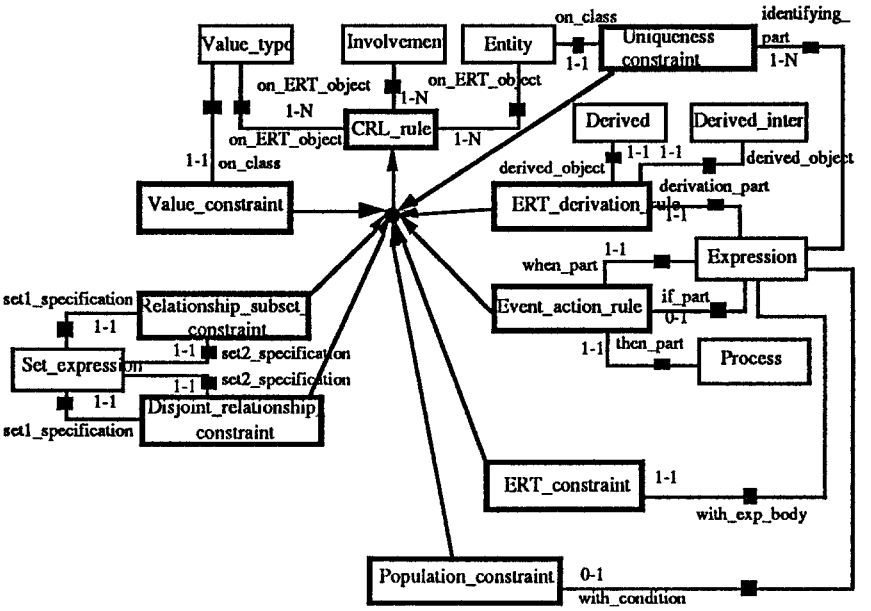

Figure 3. A Top-Level View of the Metamodel

In order to utilise the knowledge contained in the metamodel, an object-oriented layer has been chosen to act as the common repository of the specification. This choice facilitates the storing of both the application models and the knowledge of the modelling formalisms in one place. The application models are mapped to the object-oriented layer in terms of object classes and this mapping is guided and checked against the metamodel, which is also stored in the same layer, but in terms of object metaclasses. These metaclasses contain the description of the building blocks of the models, the way that they interrelate, constraints that act on them and methods that define the behaviour of the objects. An example of metaclasses is shown below:

```
object([
metaclass:   Event_action_rule,
isa:  CRL_rule,
slots,
   property : when_part,
      [type = Expression, presence = mandatory],
   property : if_part,
      [type = Expression, presence = optional],
   property : then_part,
      [type = String, presence = mandatory],
   property : parameters,
      [type = List(Undefined), presence = optional, card = 0-U],
   constraint : possible_parameters_values,
      [category = invariant,
```

```
        definition = (this#parameters is_in oneof
         [ERT_class,String,Numeric_expression,Arithmetic_expression])],
    method : action_rule_init,
        [definition = action_init/2, category = Prolog],
endslots,
endmetaclass]).


object([
metaclass: Process,
isa:   Metaclass,
slots,
    property: process_has_name,
        [type= String, presence = mandatory],
    property: process_has_ID,
        [type= String, presence = mandatory],
    property: is_decomposed,
        [type= List(Process), presence = optional, card = 0-U],
    property: is_part_of,
        [type= Process, presence = optional],
    property: is_described_by,
        [type= List(PLL_Rule), presence = optional, card = 0-U],
    property: has_precondition,
        [type= List(Input), presence = optional, card = 0-U],
    property: process_produces_output,
        [type= List(Output), presence = mandatory, card = 1-U],
    property: process_produces_trigger,
        [type= List(Trigger), presence = mandatory, card = 1-U],
    property: affects,
        [type= List(ERT_Object), presence = mandatory, card = 1-U],
    property: is_controlled_by,
        [type= List(Event_action_rule), presence=optional, card = 0-U],
    method : process_init,
        [definition = proc_init/2, category = Prolog],
endslots,
endmetaclass]).

object([
metaclass:Flow,
isa:   Metaclass,
slots,
    property: flow_has_name,
        [type= String, presence= mandatory],
    constraint: total_involvement_of_flow,
        [category = invariant,
         definition = (this is_in oneof [Input, Output, Trigger])],
    method : Flow_init,
        [definition = flow_init/2, category = Prolog],
endslots,
endmetaclass]).
```

Apart from these definitions, the metamodel also contains a set of axioms which must be verified in order for every set of specification to be correct, consistent and in accordance with the other models. We distinguish two types of axioms (rules): Error rules which are

verified to insure correctness of the specifications and warning rules which suggest better organisation of the specifications. Examples of such axioms are given below:

con4: In a uniqueness constraint: <ERT_id_name> is_identified_as <ERT_access> the ERT_id_name must appear in the ERT_access.

con5: In a relationship disjoint constraint:<set_exp> is_disjoint_from <set_exp> the two set expressions must not be identical.

con6: In the subset relationship and relationship disjoint constraint, all the variables of the set expressions involved in the constraint must have been instantiated.

Following the schema of figure 1, every time the application models are mapped to the object-oriented layer, they are checked against the information stored in the metaclasses and the axioms and when errors or warnings are detected, they are reported back to the user for correction. When the mapping succeeds, the specification will be stored in the repository in terms of classes which are instances of the metamodel's metaclasses. Although this procedure assures the well-formedness of a conceptual specification according to the rules of the metamodel, it provides no facilities for an in-depth analysis of the specification in terms of the wishes and views of end users. This is a task for the validation tools, whose architecture is described in section 3.4 and which makes use of the models and metamodel described in sections 3.2 and 3.3 respectively.

3.4 The Validation Scenarios

Although, interactive graphics and animation were used successfully in programming, these techniques have not been fully applied in conceptual modelling.

The architecture shown in figure 4 uses the same techniques, with program animation, in order to provide an indication of the dynamic behaviour of the specification given in the conceptual modelling formalisms.

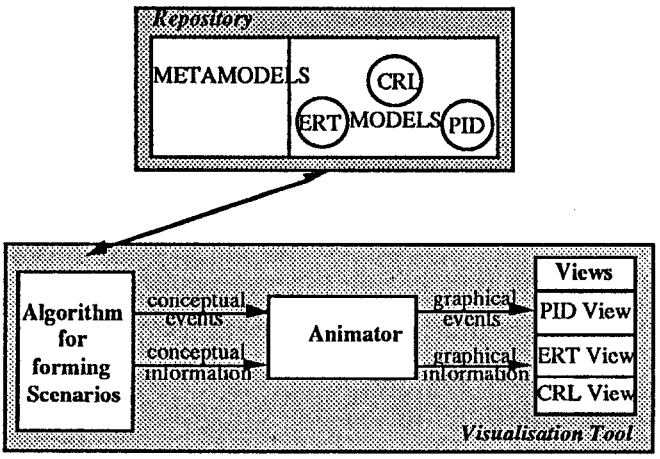

Figure 4. An Architecture for Visualising Conceptual Specifications

Graphic editors would be sufficient in order to browse through each model separately. However, during analysis of specification, a more powerful tool is needed, to assist the developer. All interrelations between the various conceptual modelling formalisms used should be demonstrated in a consistent and easy to understand way.

One approach is the use of scenarios which could help the user in understanding the captured information [15]. For example, starting from the functional model one could ask "What If X", that is what would happen if X holds. In the architecture depicted above, the component of the visualisation tool which is responsible for forming the scenarios and walk through a specification fragment in order to follow the scenario, is the *Algorithm of Forming Scenarios*. The algorithm will start from the functional model and analyse the behaviour of the system by checking the interrelationships of the models. It will also report any events and conceptual information to the second component of the tool, namely the *Animator*.

The *Animator* is responsible for translating the events and conceptual information reported by the Scenarios Algorithm, to graphical events and objects respectively. These graphical events, along with the graphical objects are reported to the *Views*.

A *View* has a window associated with it and methods in order to react to any event or graphical information sent by the Animator. More than one View can be used in order to visualise the various interrelationships between the conceptual models. Each View is responsible for updating its window according to the events, and all Views are updated simultaneously. The Views use colour, movement and animation techniques; thus providing the developer with multiple animated views of the dynamic behaviour of the specifications. A detailed description of the components of this architecture is given in section 4.

4 The Visualisation Tool

The three conceptual modelling formalisms described in section 3, are strongly interrelated and this interconnection is explicitly recognised and represented according to the metamodels of these formalisms. This important characteristic is used in visualising and validating a developed specification.

The basic idea, described in [15], was to use the process model as a starting point in order to form scenarios that could help the developer in understanding the captured information. More specifically, given a trigger (usually an external one), a set of processes are initiated and this invocation starts a number of execution paths that change the state of the business world.

Apart from finding these paths, it is possible to demonstrate when each process is going to be executed and what are the business objects that are directly affected by it. However, the way this information is reported back to the developer is very crucial for understanding the dynamic behaviour of the specifications. The Animator and the Views use a graphical representation with which the developer is familiar. The graphical

representation is similar to the graphical notation used to define the models (i.e. PID diagram).

The three components of the visualisation tool are described in detail in the next subsections while an example of the use of the tool is given in section 5.

4.1 The Algorithm of Forming Scenarios

The Scenarios Algorithm is responsible for defining the scenarios and starting from the models to follow their interrelation links in order to gather all the necessary information described previously in this section. The scenario that is currently used is the "what if X", and the starting point of the algorithm is the process model.

Execution Paths

The driving notion of the algorithm is that of trigger. Triggers are considered in as events that take place either in the external world or in the information system itself and cause the system to take some actions in order to respond to them. Three different kinds of triggers have been identified: external ones that originate from the external world (i.e., external agents in the PID model), conditions concerning the business objects (these originate from ERT views in the PID model) and flows from processes carrying ERT information that invoke other processes.

In the current implementation of the algorithm, a list of all the triggers that are presented in the PID model (or in the WHEN part of the corresponding event-action rules) is shown in a separate window and the user is then able to select one of them and *fire* it.

After a trigger has been selected and fired, the algorithm finds the set of processes that are initiated by the specific trigger. Each one of them constitutes the beginning of an execution path that consists of a set of other processes. The path terminates when no other processing is required, i.e., when a process either produces a flow that goes to the external world, or updates the ERT objects. This implies that each execution path is formed by the first process and all those that are fired by an output flow of a process that has been already included in the path. It should be pressed here that, since an output flow of a process may trigger more than one processes, the result of performing the algorithm will be a tree of process paths.

Linking PID with ERT

A very important aspect of the approach described in this paper is the interconnection of the three models. Although the designer develops three separate models (ERT, CRL, PID), there are ways of viewing them as a whole. Actions may be taken in a system and rules may hold or be violated, but there is a common basis that binds all these together namely, the entities, value classes and their relationships.

When a trigger invokes an execution path as discussed above, it is very important to understand not only what sort of actions are taken, but also how these actions affect the ERT model. One way of looking into this is to present to the user the action part of the event-action rules that describe the internal processing of each process. But in most cases, these rules are quite difficult to be understood by a naive user. Moreover, the user

does not usually care about a detailed description of a process' internals. It is desirable, however, to inspect the effects of every process and a way of accomplishing this is by merely finding the set of ERT objects that this process affects. Such a facility would, in essence, give a comprehensive summary of the execution of processes, so that the user may understand what will be the state of the business after carrying out some actions.

This idea has been included in the algorithm which, after finding each process and its preconditions in the initiated execution path, finds the ERT objects (entities, relationships and value classes) that are affected by it.

Linking ERT with CRL

The final step that has been considered in this algorithm deals with the connection of the ERT model with the constraints and derivations expressed in the CRL. Many ERT objects are going to have a number of CRL rules associated with them, which will express constraints on their values or associations or the way that they can be derived by using information already existing in the model. Since the algorithm finds the ERT objects that are affected by the execution of each process, it could be shown what are the constraints (or derivations) relevant to them.

By providing such a facility, the user will be able to inspect the various rules expressed in CRL, see how they are connected with the ERT and consequently with the PID model and finally concentrate on the way these rules will affect what has been modelled in the two other models. This can reveal certain deficiencies of the captured information. If, for instance, there is a process that creates instances of the entity class X and a rule that constraints the population of this entity, by bringing all these together, the user might wonder what will happen if the rule is violated and suggest certain actions to be included in the PID diagram in order to tackle such a situation.

This part of the algorithm has not been implemented yet. However, since the ERT model is connected with constraints and derivations expressed in the CRL, the algorithm can also find the constraints or derivations relevant to the ERT objects affected by a process.

Reporting to the Animator

Finally, the gathered information, should be reported to the user through the Animator and the Views. Whenever an interesting event occurs, as the algorithm executes, it is reported to the Animator. The notion of interesting events was first used in program animation, to describe the events that happen during the execution of a program and are interesting for the user to see. The interesting events in the Scenarios Algorithm are:

- the selection of a trigger
- the invocation of a process
- a process affects an ERT object
- a constraint rules restricts an affected object

When such an event occurs, the Animator is notified and the relevant information is also send from the Algorithm to the Animator (for example, the ERT objects affected).

4.2 The Animator

The Animator translates the conceptual information given by the Algorithm to a graphical one. Therefore, the Animator 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 Scenarios Algorithm.

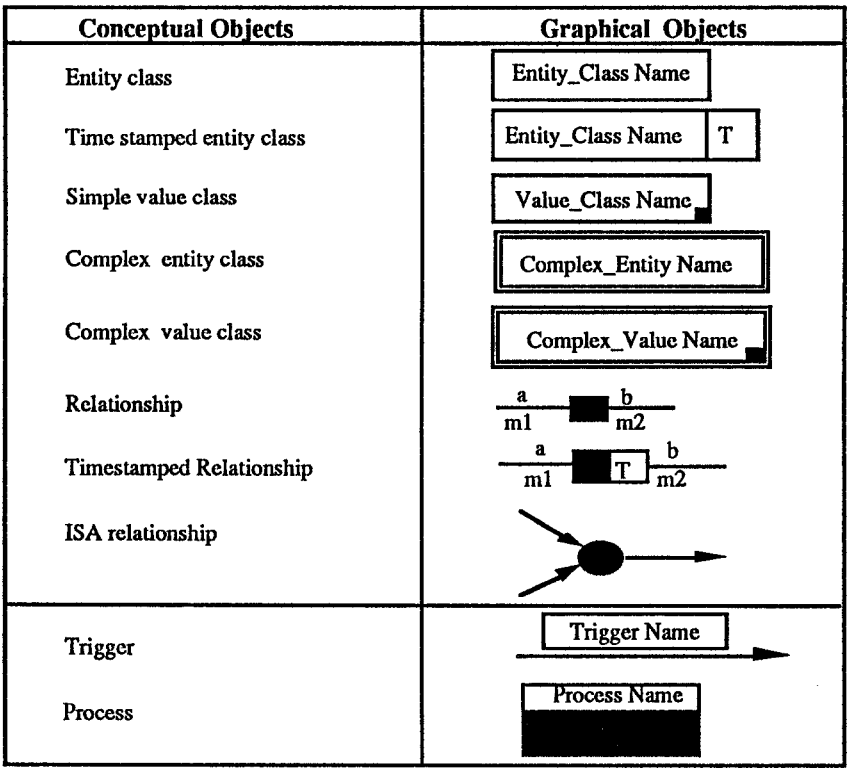

Figure 5. Concepts and their Corresponding Graphical Objects

Graphical Objects

The graphical objects of the Animator, 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.

Figure 5 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).

Graphical Events

The Animator also translates the events reported by the Scenarios Algorithm 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.

4.3 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 at user's request.

The way a View reacts in a graphical event is by drawing the corresponding graphical object in its window, or by changing a graphic attribute (i.e. colour) of a graphical object. For example, for the event "draw process P1", the View will automatically calculate the position and size of the graphical object P1 and draw it on the window, while for the event "colour Employee with the colour of P1", the View will find the right colour (the one that the graphical object P1 has), change the colour attribute of the graphical object Employee and re-display it, if necessary.

A single View can be used to animate all the information reported by the Scenarios Algorithm. 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.

All Views have a set of general methods in order to support automatic re-sizing and repaint either at user's request or whenever is needed (i.e. a window hiding part of a View was moved).

PID View

This View is responsible for drawing the execution paths. The events which this View recognises are the selection of a trigger and the invocation of a process. The graphical objects it uses are the ones representing triggers and processes.

Whenever a trigger is selected the View reacts by drawing this trigger and whenever a process is initiated the View, draws the process with a different colour. The overall result of this animation will be the drawing of a tree of execution paths in a depth first manner.

ERT View

This View is responsible for displaying any ERT objects that are affected by the execution of a process. The graphical objects which this View uses are the ones representing entities, value classes, complex objects and relationships.

Whenever an ERT object is affected by the execution of a process, the View is updated by drawing the corresponding object with the same colour that the process has. This View is updated simultaneously with the PID View, that is whenever a new process is displayed in the PID View, the ERT that are affected by it, are displayed in the ERT View with the same colour the process has in the PID View.

CRL View

This view displays either a CRL rule that constrains an ERT Object affected by the initiation of a process, or the event-action rule that invokes a process. The first View links the ERT and CRL models, while the latter links PID and CRL models. Two possibilities exist for either Views. One is to always display the CRL View, and another is to allow the user explicitly ask for the View. In the latter case it is possible to allow the user to select a process by clicking in the PID View and only then the CRL View will appear and display the event-action rule, which triggers the selected process.

4.4 Implementation Platform for the Visualisation Tool

The prototype of the visualisation tool, described in the previous subsections was implemented using the Algorithm of Forming Scenarios [15] and *Jasmine* [17]. Figure 6 shows the internal structure of this prototype.

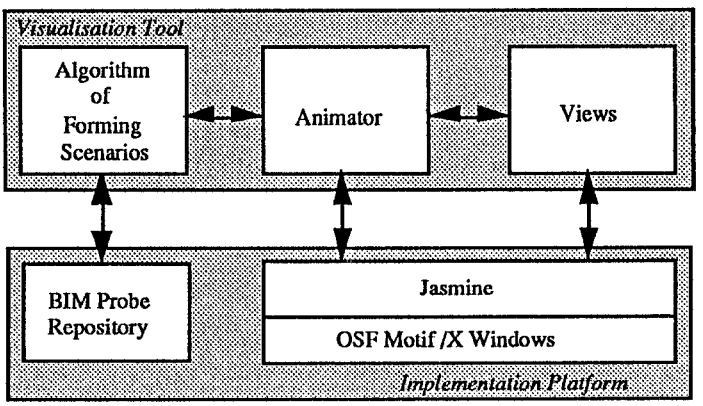

Figure 6. Structure of the Animator and of the Views

The Algorithm of Forming Scenarios was implemented in BIM Prolog, while the metamodels and models are stored in a BIM Probe repository. Jasmine formed the basis for the Animator and the Views components of the visualisation tool.

Jasmine consists of two tools. The first tool is a set of functions to define and manipulate simple graphical objects (i.e. boxes, circles, text and lines) or complex ones (i.e. lines that connect two objects together) and install them on multiple views and windows. As a consequence the user of the tool can create animations of a program without having the knowledge of window systems or graphics. The implementation of *Jasmine* was done on SUN workstations running SUN UNIX using C and C++ programming languages and X window system.

The Animator uses Jasmine's set of functions to create the graphical objects which represent the concepts of the models, while the Views use the functions provided by Jasmine in order to manipulate the graphical objects.

5 Example

In order to demonstrate the functionality of the Visualisation tool, an industrial case study has been used[2] . Part of the PID model developed for this case study is depicted in figure 7.

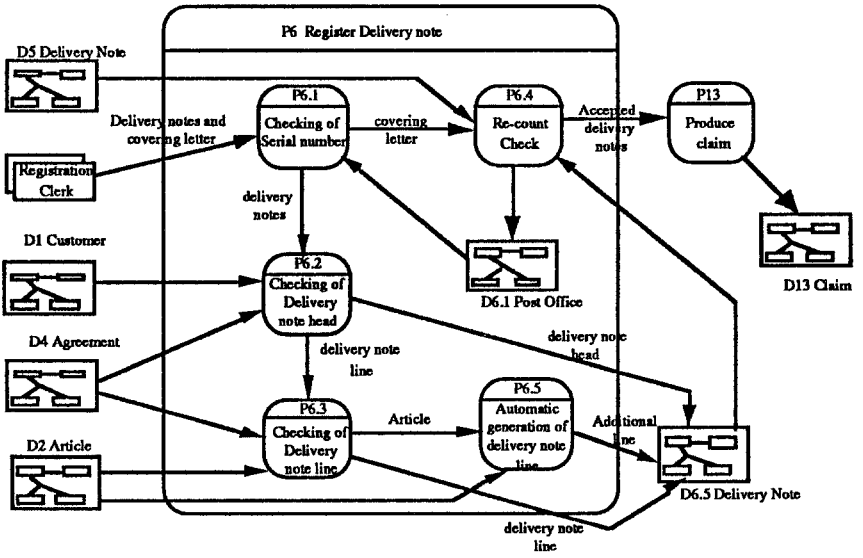

Figure 7. Example PID from the Sweden Post Case Study

The Scenarios Algorithm will start from the external trigger *Delivery Notes and covering Letter*, and find all the processes that are initiated by it, process P6.1 in this example. In

---

[2] This case study has been used fo all aspects of developing a conceptual specification, using state of the art CASE tools as well as representing the specification and metamodel in an object-oriented environment. The example given here represents a small subset of the entire case study.

the Scenarios Algorithm, all preconditions are considered to be satisfied, therefore process P6.1 will "execute" immediately and the Scenarios Algorithm will proceed by finding all the affected ERT objects.

All these events will reported to the Animator as they happen (in real time). The Animator will then, using the type of event and the information associated with it, translate every conceptual object to its corresponding graphical one, and every event to one that the Views will understand and send it to the Views.

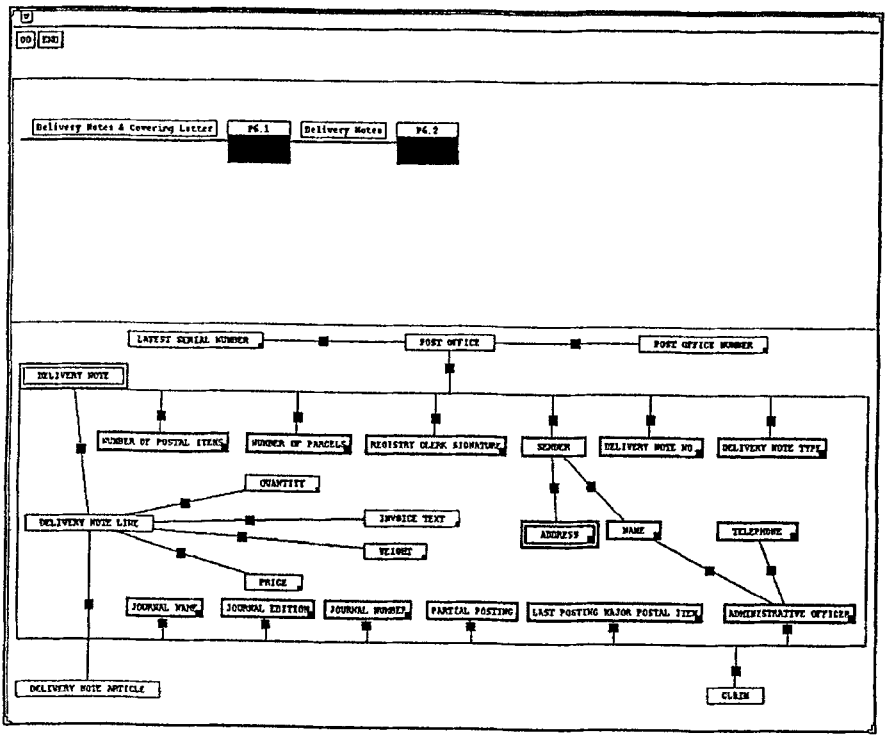

Figure 8. First Snapshot of PID and ERT Views

The Views will then, update themselves according to the events occurred. For example, when the event of trigger selection is reported from the Animator to the Views, the PID View, will display the graphical representation of the trigger.

The PID View is responsible for finding the right position within the Views' window, and the right size of the graphical object, so as the name of the trigger can fit in the graphical object. In a similar way, the ERT and CRL Views will be updated, when the affected ERT objects and the corresponding CRL rules will be reported by the Animator. Views also have a mechanism for re-size and re-draw their corresponding windows, whenever necessary.

In figures 8, 9 and 10 snapshots of the ERT and PID Views are given, for the example PID diagram shown in figure 7. In order to produce these snapshots and make the changes on the ERT View visible on paper, as well as on a black and white screen, instead of colouring, the ERT objects are highlighted when a process affects them and turn to normal when the output trigger of this process are displayed on the PID View.

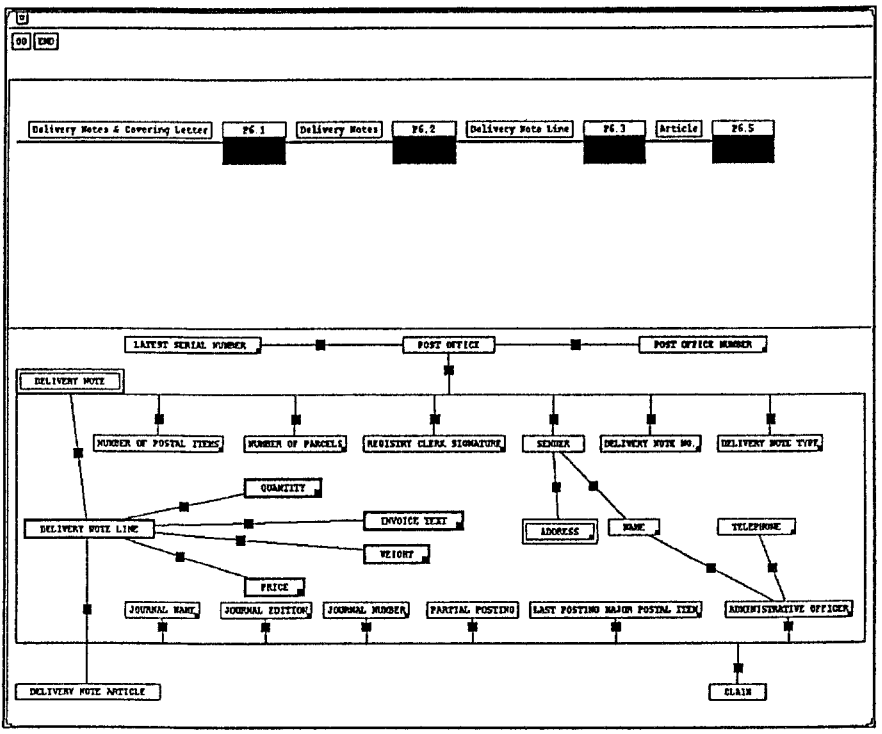

Figure 9. Second Snapshot of PID and ERT Views.

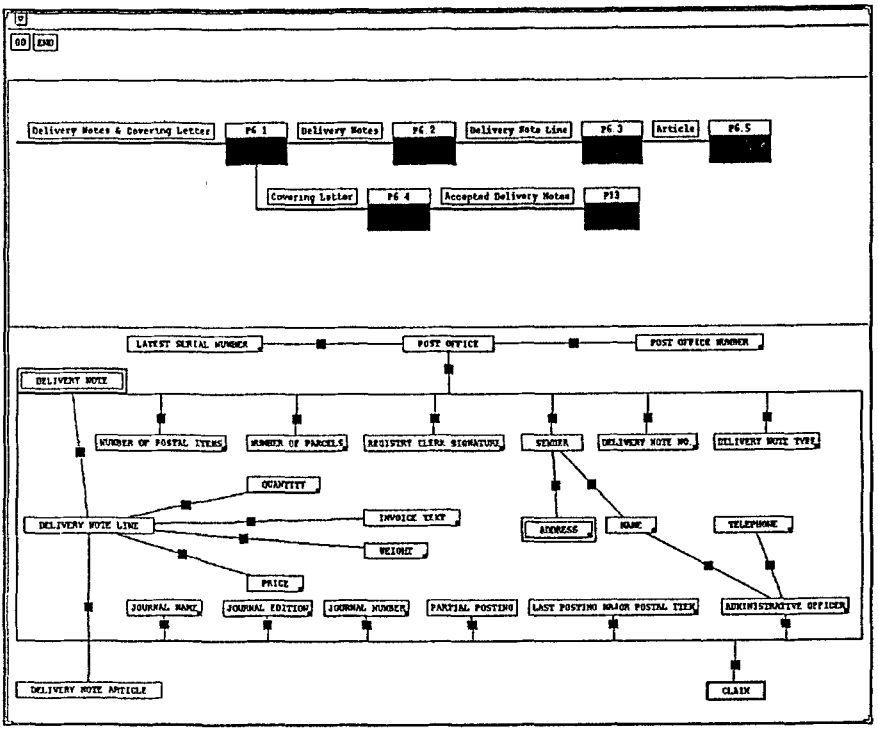

Figure 10 Third Snapshot of PID and ERT Views.

6 Conclusion

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.

The requirements engineering paradigm discussed in this paper considers a declarative approach to the task of representing functional requirements. The core modelling components are those of the objects of the modelled domain and the constraints, both of static and transitional nature, that are associated with these objects. Object modelling is carried out using the ERT model whereas rule modelling uses the CRL as a textual language for expressing constraints. Due to the limited scope of viewing rules in a textual

formal, particularly those concerned with behavioural aspects, a CRL schema can also be viewed in terms of the PID model. The ability to view a requirements schema in familiar or easily explained terms to end users is regarded by the authors as central to the validation process of requirements. This process involves the agreement of many different stakeholders and system developers about, amongst other things, the intention of the target system.

This paper also argues that the reviewing process involved in validating specifications should involve not only static mechanisms in terms of models that one can reason about but also animation of the specification that can provide requirements stakeholders and system developers with an early opportunity to observe the functional behavioural of the proposed system. Such a mechanism is presented in this paper by animating information captured by the three models of ERT, PID and CRL, in a uniform way. A first version of the prototype has been implemented and used on a number of case studies.

The next stage of development will involve a number of important enhancements. Firstly, the third part of the algorithm that deals with connecting the CRL model with the ERT and PID models will be fully implemented. To achieve this, the CRL metamodel has to be studied and possibly changed so as to facilitate easy acquisition of the stored rules. Secondly, a set of further enhancements are currently under development in order to provide an interactive facility for users for guiding the animator about the possible execution of scenarios to be followed. For example it is desirable to involve the user each time a precondition is encountered in the process paths. In this way, instead of being a passive viewer the user will be able to drive the algorithm and choose possible paths that are of interest.

References

1. Baecker, R.M., *An Application Overview of Program Visualization*, Computer Graphics, 20, 4: 325, 1986.

2. Batini, C., T. Catarci, M.F. Constabile and S. Levialdi, *Visual Query Systems*, Universita degli Studi di Roma La Sapienza, Report, 04.91, March, 1991.

3. Bentley, J.L. and B.W. Kernighan, *A System for Algorithm Animation: Tutorial and User Manual*, AT&T Bell Laboratories, Technical Report Computer Science, 132, January, 1987.

4. Brown, M.H., *Perspectives on Algorithm Animation*, CHI'88: Human Factors in Computing Systems, Washington, D.C., 33-38, 1988.

5. Brown, M.H., Hershberger, J., *Animation of Geometric Algorithms: A Video Review*, DEC Systems Research Center, Research Report, 87a, June 6, 1992.

6. Brown, M.H. and R. Sedgewick, *A System for Algorithm Animation*, ACM Computer Graphics, 18, 3: 177-186, 1984.

7. Brown, M.H. and R. Sedgewick, *Techniques for Algorithm Animation*, IEEE Software, 28-39, 1985.

8. Cauvet, C., C. Proix and C. Rolland, *Information Systems Design: an Expert System Approach* , Artificial Intelligence in Databases and Information Systems (DS~3), R. A. Meersman, Z. Shi and C. H. Kung, Canton, China, North-Holland, 1-28, 1988.

9. Clemons, E.K. and A.J. Greenfield, *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.

10. Edel, M., *The Tinkertoy Graphical Programming Environment*, COMPSAG, IEEE, 466-471, 1986.

11. Eick, C.F.,*Metoden und Rechnergestütze Werkzuege für den Logischen Databankentwurf* Dept. of Informatics, University of Karlsrue, 1984.

12. Falkenberg, E.D., *Information Modelling - Subjective Forever*, Database dag, Eindhoven, 1989.

13. Hirakawa, M., S. Iwata, I. Yoshimoto, M. Tanaka and T. Ichikawa, *An Environment for HI_VISUAL Iconic Programming*, Workshop on Visual Languages, 305-314, 1987.

14. Kangassalo, H., *Concept-D : A Graphical Language for Conceptual Modelling and Data Base Use*, Workshop on Visual Languages, IEEE, 2-11, 1988.

15. Katsouli, E.,*Verification and Validation for Rule-based Requirements Specifications* MSc Thesis, UMIST, 1992.

16. Kramer, J. and K. Ng, *Animation of Requirements Specification*, SPE, 18, 8: 749-774, 1988.

17. Lalioti, V.,*Jasmine: A cinematographic representation of algorithms* MSc Thesis, University of Crete, Iraklion, GREECE, 1990.

18. London, R.L. and Duisberg, *Animating Programs Using Smalltalk*, Computer, August: 61-71, 1985.

19. 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, 1991.

20. Myers, B.A., *Visual Programming, Programming by Example, and Program visualization: a Taxonomy*, SIGCHI '86 Conference on human Factors in computing Systems, ACM, 59-66, 1986.

21. Nierstrasz, O., D. Tsichritzis, V. de Mey and M. Stadelmann, *Objects + Scripts = Applications*, ESPRIT '91, Brussels, 534-552, 1991.

22. Nijssen, H.,*Conceptual Schema and Relational Databases - A Fact-Oriented Approach*, Prentice Hall, 1989.

23. Osborne, W., M., *Fitting pieces to the maintenance puzzle*, IEEE Software, January: 11-12, 1990.

24. Pietri, F., Puncello, P.P., Torrigiani, P., Casale, G., Innocenti, M.D., Ferrari, G., Pacini, G., Turini, F., *ASPIS : A Knowledge-Based Environment for Software Development* , ESPRIT '87 : Achievements and Impact, North Holland, 375-391, 1987.

25. Reiss, S.P., *PECAN: Program Development Environments that support Multiple Views*, IEEE Transactions on software Engineering, 11, 3: 276-285., 1985

26. Roman, G. and K.G. Cox, *A Declarative Approach to Visualizing Concurrent Computations*, Computer, October: 25-36, 1989.

27. Shu, N.C., *FORMAL: A Forms-Oriented and Visual-Directed Application System*, IEEE Computer, 18, August: 38-49, 1985.

28. 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.

29. Smart, J.C. and V. Vemuri, *A-Vu: A Visualization Tool for Complex Software Systems*, IEEE, August: 172-182, 1992.

30. Smith, D.C., C. Irby, R. Kimball, W. Verplank and E. Harslem,*Designing the Star User Interface* 7: 242-282, 1982.

31. Theodoulidis, C., P. Loucopoulos and B. Wangler, *A Conceptual Modelling Formalism for Temporal Database Applications*, Informations Systems, 16, 4: 401-416, 1991.

32. Tsalgatidou, A.,*Dynamics of Information Systems Modelling and Verification*, PhD Thesis UMIST, Manchester, UK, 1988.

33. Wallace, D.R., Fujii, R.U., *Software Verification and Validation : An Overview*, IEEE Software, May 1989.

34. Wohed, R., *Diagnosis of Conceptual Schemas* , Artificial Intelligence in Databases and Information Systems (DS-3), Proc. IFIP Working Conference on The Role of Artificial Intelligence in Databases and Information Systems, R. A. Meersman, Z. Shi and C. H. Kung, Canton, PR China, North-Holland, 437-456, 1988.

35. Zloof, M.M., *A Language for Office and Business Automation*, AFIPS Office Automation Conference Digest, AFIPS Press, 249-260, 1980.