ICSSEA '99

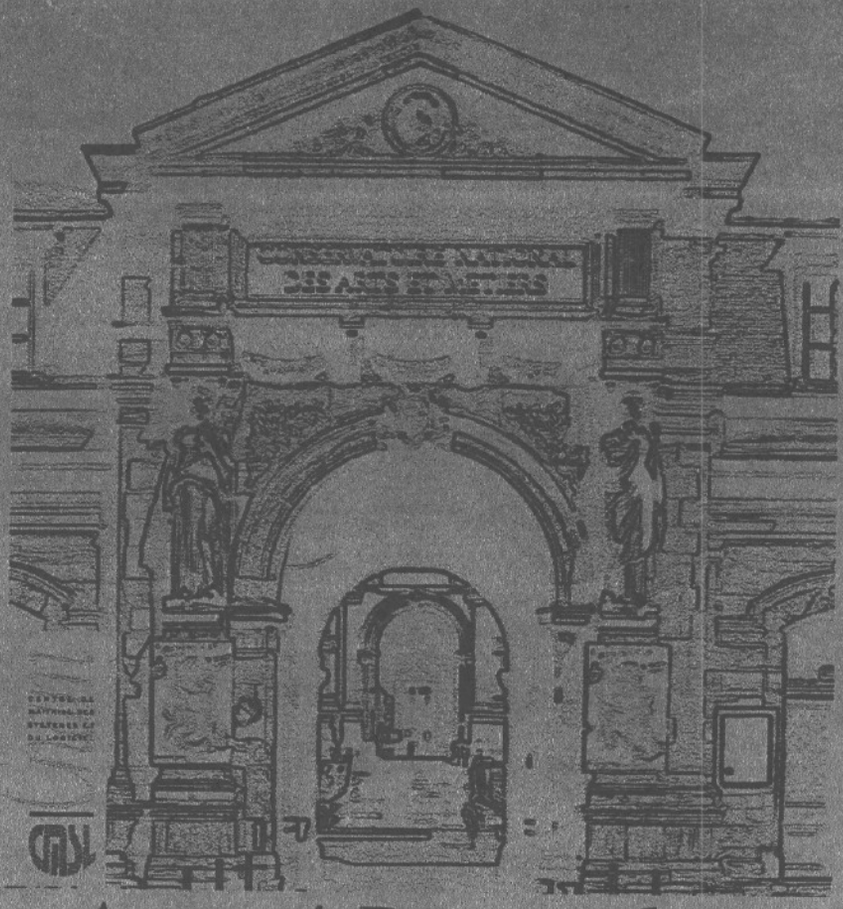
ICSSEA '99

12th International Conference

SOFTWARE & SYSTEMS ENGINEERING *and their* APPLICATIONS

December 8-10, 1999

CNAM — Paris, France



Actes / Proceedings

Volume 3

SESSION 9

TESTING
Co-Chairs: Herm Fischer — Mark V (USA)
& Philippe Baufreton — Snecma (F)

A Development Environment to Support Assisted Definition of Test-Procedures for Complex Systems[1]

Juan Garbajosa±, Mario Piattini¥, Angeles Mahillo¥, Hector Garcia±

±Madrid Technical University. E.U. Informatica. Dpt. OEI. Ctra. Valencia Km. 7, E-28031 Madrid. Spain. Fax: +34-913367520. E-mail: jgs@eui.upm.es, hgarcia@quebrantahuesos.eui.upm.es

¥ University of Castilla - La Mancha. E.U. Informatica. Ronda de Calatrava 5, E-13004 Ciudad Real. Spain. E-mail: mpiattin@inf-cr.uclm.es

¥ Madrid Technical University. E.U. Informatica. Dpt. LPSI. Ctra. Valencia Km. 7, E-28031 Madrid. Spain. E-mail: amahillo@eui.upm.es

Abstract

Operation of complex systems, such as telecom switching systems, power generator plants, satellites systems and subsystems, chemical industry, aeronautics systems, etc., consist of a sequences of operations which interact each other to reach the goal of the system. It occurs that later operations depend on actions taken by earlier operations.

Testing of these systems requires the production of series of commands that validate the system operation, in a repetitive fashion. These series of procedures are often called test procedures. The use of programming languages requires to work at a low level of abstraction, farther away from what the real problem is. A real advantage can be obtained from assisting the test engineer in this task. The satellite domain was a precursor in this issue.

Within this paper we present after this introduction, a concept model for assisted test procedure definition, result of former work of the authors. Then a development environment architecture to support it. After that, we present a case study for the telecom industry (network management area. Fourth we describe, briefly, related work in other areas. Finally we present some conclusions.

Keywords: assisted testing, system testing, system validation, tools and environments, model-based testing

[1] This paper has been written under the ARCO project, Ref. TIC98-0782, which is partially supported by the Interministerial Commission for Science and Technology of Spain.

1. Introduction

Operation of complex systems, such as telecom network management systems, telecom switching systems, power generator plants, satellites systems and subsystems, chemical industry, aeronautics systems, etc., consists of sequences of operations which interact with each other to reach the goal of the system. As explained in [9] the most salient feature of these systems is its dynamic nature. The number and kind of elements composing the system as well as the mode of control of those elements change frequently in response to the environment. Therefore we have that, in many cases, later operations depend on actions taken by earlier operations.

These environments are guided by knowledge described as records, formal procedures, and rules, which command the interaction of the operations. Big amount of data is constantly updated. The bulk of knowledge raises a problem, thus it turns out to be a burden to the test engineers and operators, which handle the system, who have to make decisions based on the current status.

Testing of these systems requires the production of series of commands that validate the system operation, in a repetitive fashion. These series of procedures are often called test procedures. Test procedures are produced using programming languages. Thinking in terms of test engineers, that might be not so familiarised with computer concepts as computer scientists are, the use of these programming languages requires to work at low level of abstraction, farther away from what the real problem is. Programming languages used (C, Atlas, Tcl/Tk ETOL) are frequently not high level ones. But even high level programming languages, such as object oriented, are not yet so close to real life systems as we would like to be.

Therefore the test engineer faces three problems:

- To design –*define*– the test procedures that will validate the system.
- To translate these test procedures into a programming language.
- To execute these tests procedures with little or no help of the test environment.

Derived problems to cope with are the following:

- To maintain the tests procedures, but we prefer to focus on the above mentioned items. Actually maintainability will also get an advantage from the approach we will propose.
- Guarantee a full test coverage

In literature [4] it is reported that generation of test cases for systems for which a static data model is sufficient to capture the system's behaviour can be automated. It is also reported in [4] that for those systems we are concerned this is still an open problem

Concerning the problems associated to the test engineer work, space industry depicted what was called as *generation of test procedures* as explained in [6]. It was intended that the test engineer, instead of having to define test procedures with a low level programming language, could provide the tool with enough information to a tool so that the tool would generate the test procedure itself.

The reason for the space industry being precursor in this field was that satellites require a careful testing before they are launched. Afterwards it is, usually, too late to fix operation mishaps. Therefore the need to research how to facilitate the definition of test procedures was identified as important.

Test procedure generation approach intended to allow the test engineer to focus on his core work, test procedure definition. This way of defining test procedures could actually be denominated as *assisted* test-procedure definition. To support this assisted test-procedure definition there must exist a model behind it, and a development environment.

Our work is focussing is producing a cost effective development environment to support assisted test-procedure definition.

Within this paper we present after this introduction, a concept model for assisted test procedure definition, result of former work of the authors. Then a development environment architecture to support it. After that, we present a case study for the telecom industry (network management area), in which we have implemented the environment model, we relate it to former experiences, and analyse achievements, tasks that are fully supported and that that are not yet so well supported, constraints and cost effectiveness. Fourth we describe, briefly, related work in other areas such as model-based testing, scenario based requirements engineering, and its relationship to our work. Finally we present some conclusions.

2. Assisted Test Procedure Definition: A Concept and a Model

System test procedure definition is a complex activity that requires the test engineer to be fully acquainted with several areas such as the physical systems for which the tests are being defined, and programming techniques. In this respect, the gap between the way the definition is performed and what the engineer should understand as a natural way of working is quite significant; furthermore, the engineer receives little or no guidance from the tools he uses to define the tests.

It may be worthwhile paying some attention to the satellite domain. Satellites are complex artefacts made of many subsystems. Some subsystems aim at commanding the satellite, some other at communicating with the operations centre, and payloads with experimental, scientific or, just, business objectives, such as sending images to Earth. Furthermore a failure in a satellite may mean the lost of the mission. Therefore the need for a systematic system testing was identified in an early time, as described in [6].

Testing of these systems requires the production of series of commands that validate the system operation, in a repetitive fashion. These series of procedures are often called test procedures. Test procedures are produced using programming languages. Thinking in terms of test engineers, that might be not so familiarised with computer concepts as computer scientists are, the use of these programming languages requires to work at low level of abstraction, farther away from what the real problem is. Programming languages used (C, Atlas, Tcl/Tk ETOL) are frequently not high level ones. But even high level programming languages, such as object oriented, are not yet so close to real life systems as we would like to be. Yet, the test engineer requires receiving support and assistance to produce tests procedures. This problem was identified several years ago in the satellite industry as explained in [5].

Satellite test activities involve the generation of actions and the visualisation and analysis of measurements. In this way test activities allow us to determine the impact of commanding an action by visualising the incoming measurement. Measurement and action concepts are the base of test definitions. Actually, measurements verify that the actions performed –by means of the test engineer commands– get the desired effect. These are called Telemetry/Telecommand, usually known as *TM/TC* to collect under the same term a widest set of forms to represent communication between the test system and the Man Machine Interface.

Relating this issue to high order language concepts, taking advantage of the object oriented paradigm, satellite components to be verified can be represented by objects, and actions by operations. Measurements are the result or output of applying an action on an object.

This scheme is valid not only for the satellite domain but for most of the application domains with systems to be tested. In figure 1 we can see the basic functionalities of the assisted test procedure definition concept as opposed to the traditional way of working.

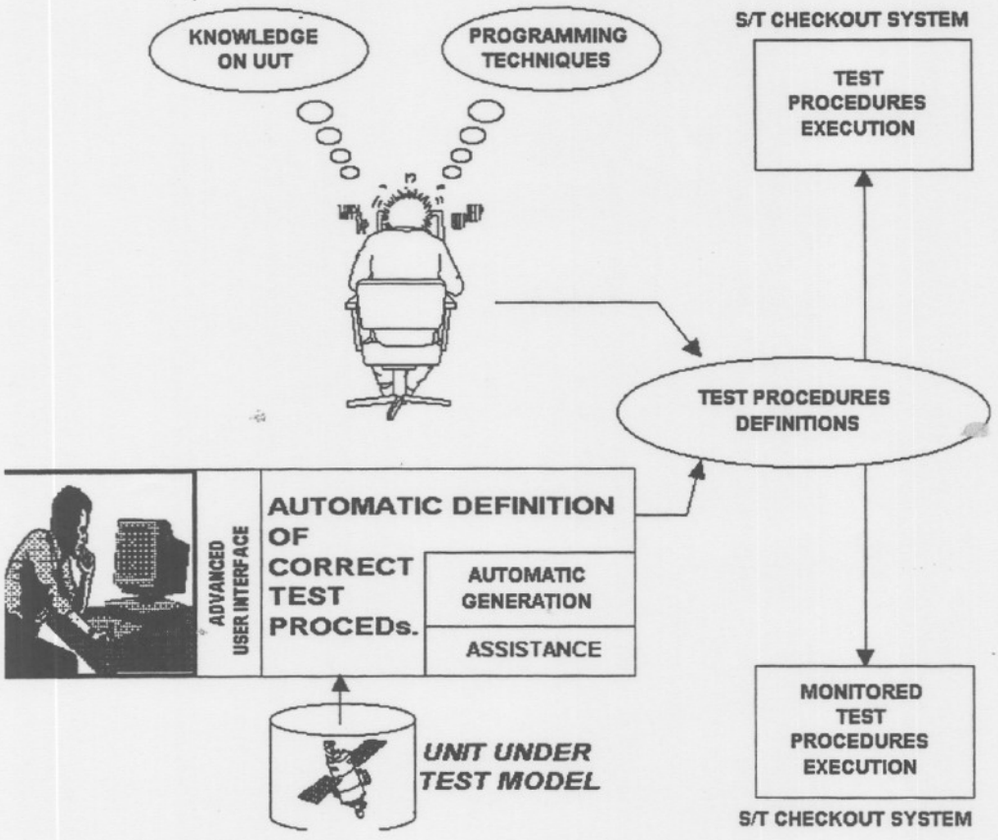


Figure 1: Traditional *versus* Advanced Test Definition Environment

As mentioned in [5] it is possible to provide assistance for the definition of test procedures based on this scheme. Figure 1 shows a comparison between a traditional environment, to define and execute test procedures, and those under development. These new environments can be considered as front-ends that allow the engineer to define test procedures through a graphical Man Machine Interface (MMI) that represents the physical system.

Consistency of the procedures can be checked partially, in an early phase, being possible to execute the tests in a monitored mode. The main output, the test itself in the form to be executed, remains transparent for the engineer. He defines the tests, in an easy way to him, and executes them. He need not be aware of traditional programming languages and operating systems.

3. An Environment for Assisted Definition of Test Procedures: Architecture

In [1] a conceptual three-layer architecture with a conceptual layer, an interaction layer, and a physical layer was depicted. This conceptual architecture can be implemented, in terms of components, as follows: Man Machine Interface, Data/knowledge Repository and code generator. The test engineer

handles the Man Machine interface. A data and knowledge repository, which is used to represent a model of the system; a code generator module, which generates the code that will get into the industrial system; finally a low level interface between the industrial system and environment would be needed. As we shall below these basic components can be refined into a more detailed architecture. Figure 2 presents a scheme for this architecture. We can study briefly the basics for these components.

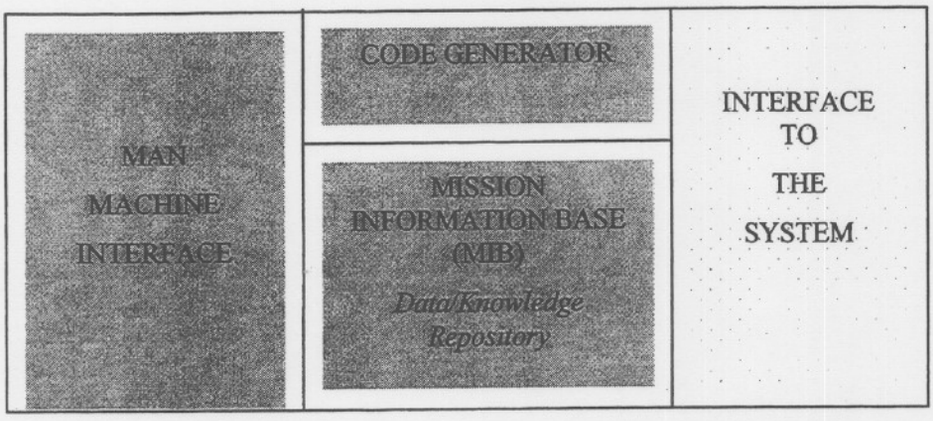


Figure 2: Environment Architecture

Man machine Interface

Man machine interfaces (MMI) are still a subject under research. In our case it is not an exception. The need to support an acceptable representation of the system to be tested together with a way to describe test commands and procedures on this system, receiving information useful on the effect of the test procedures summarises three of the functionalities of the MMI for these environments. The combination of graphic and other technologies such as natural language was studied in [5] and results were report as interesting.

Mission Information Base

One of the most important problems remains in the Mission Information base (MIB), a data Knowledge repository. This system needs to cope with big amount of information, support complex model concepts and structures, and handle it in an efficient way, in order to support the generation of code that will feed the system. In space systems this repository is called Mission Information Base. We shall keep that term. The model in the MIB will be the baseline for providing great part of the assistance and monitoring the execution of test procedures.

The knowledge stored is used to provide advice to the user, and to receive advice on undesirable situations towards the system is driven because of the test procedures he is defining. Object oriented models and system are obviously of great interest but pragmatism of relational ones should not be left aside. It can be understood that this component, at a high level, is similar for all the application systems, but at a lower level, presents specificities that we are studying now.

Code generator

The code generator must translate from the *external form* (pictorial, for instance) on to an *internal form* (abstract syntax) and *target* that the system will understand. This approach gives us a representation independence that is useful and flexible as discussed in [5].

4. A case study for network management system testing

In the market we can find tools with man machine interfaces such as those described here, but not with the assistance functionalities provided by our environment. These tools are neither oriented towards testing, nor they provide assistance in the terms defined in the former sections.

The system to be tested is made of a number of network elements, such as routers, an agent that manages the network elements, and a number of managers. Each manager would correspond to a network operator. Each manager, implemented as a window, such as that of figure 3, allows a user to perform operations such as to create a network element, or route a message.

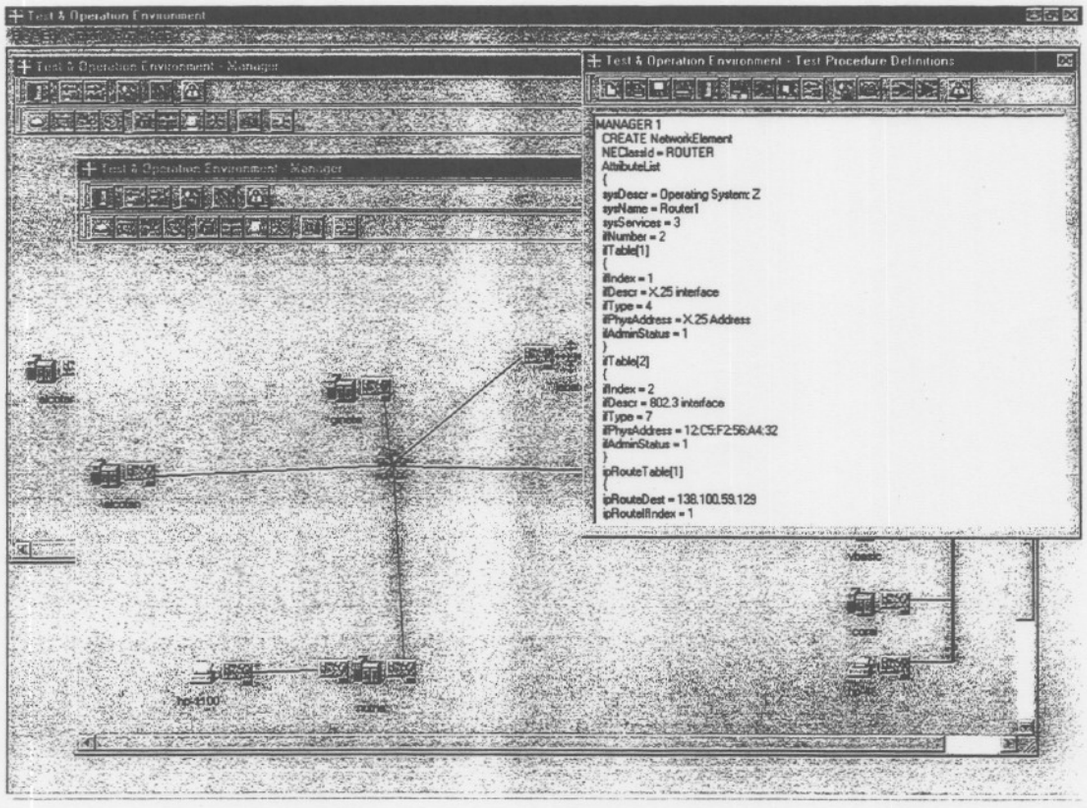


Figure 3. Several managers and a test procedure definitions working simultaneously

In figure 3 we can see an example of a part of the Man machine interface of the environment. It represents several managers working simultaneously. A window corresponds to each manager.

As we explained before, as well as the graphical interface we have an interface to describe test commands in the form of a high level programming language. In this case we are using a subset of CMIP standard, figure 4. It is obvious that, even when useful and unambiguous, this representation can be tough to be used directly.

```
CREATE Network Element
 ManagedObjectClass = ROUTER
 ManagedOrSuperiorObjectInstance = 1
 AttributeList
  {
   sysName = Router_1
   sysServices = 3
   ifNumber = 2
   ifTable[1]
    {
     ifIndex = 0
     ifType = 5
     ifPhysAddress = A X.25 Address
     ifAdminStatus = 1
     ifOperStatus = 1
    }
   ifTable[2]
    {
     ifIndex = 1
     ifType = 9
     ifPhysAddress = A MAC Address
     ifAdminStatus = 1
     ifOperStatus = 1
    }
  }
END CREATE
```

Figure 4. Example of CMIP test procedure

The data knowledge repository has a model of the network. It has been actually implemented with Oracle 8. A fully object oriented representation could have added some additional functionalities, such as better reasoning capabilities, but we preferred to use a data base such as Oracle.

The environments stores, not only, the model for the MIB, but also the test procedures in abstract syntax. This procedure is retrieved and executed whenever the test engineer wishes so. We can also consider that the abstract syntax approach allows us to produce SNMP code (the one that network management system understands) or any other that is required. This adds flexibility to the environment

5. Overview of Research Activities on System testing

As we have explained the work presented within this paper has started in the satellite-testing domain. In this section we report other research directions with a number of items in common with our work. The approaches presented here are most applicable to systems for which a data model is sufficient to capture the system's behaviour, that is, control information is not required in the model.

Model-based testing has a number of items in common which includes an underlying philosophy with respect to the importance of having a rich model representation. However, as reported in [4], testers using this approach concentrate on a data model and generation infrastructure instead of handcrafting individual tests. This is because the behaviour of the systems they support can be fully described with a model.

Generation of test cases from requirement specifications is another approach. In [3] an algorithm to automatically generate event scenarios is presented. This algorithm extracts the needed information

from the requirements forms and produces a set of scenarios that can be used to test transaction oriented systems.

Alternatively, in [8] and [7] the use scenarios as a start to define requirements are proposed. In [3] scenarios are used to represent paths of possible behaviour through a use case, and these are investigated to elaborate requirements. From requirements and use cases, tests can be generated.

Finally, following [2], it is possible to bridge the gap between formal verification and simulation by using hybrid techniques.

6. Conclusions

Within this paper we have presented the assisted test concept procedure and an environment to support this concept. The environment architecture, described for a network management system, shares basic guidelines with other systems in the satellite domain, as we discuss within the article. At present we are studying how to extend this architecture to other application domains. The MIB modelling is one of the aspects that present more problems to make general.

One of the main features of the environment is it openness. Therefore it may be integrated with other tools, such as those to support model-based testing. Actually, under the present architecture only a new application component would be necessary to be added.

Acknowledgements

Authors are indebted to Francisco Sanchis, Carlos Cuvillo, Francisco Arizmendi, Esperanza Marcos, and Maria Alandes for their useful comments.

References

[1] P. Assis-Arantes, E. Bernard, G. Leonis, T. Racaud, J. Reynolds, and R. Smillie. Object Oriented User Language for Satellite Check-Out. In EGSE Workshop, ESA/ESTEC, The Netherlands 1992.

[2] Mike Benjamin, Daniel Geist, Alan Hartman, Yaron Wolfsthal. Gerard Mas, Ralph Smeets. A Study in Coverage-Driven Test Generation. ACM DACS´99.

[3] S.J. Cunning, J.W. Rozenblit. Test Scenario Generation from a Structured Requirements Specification. IEEE ECBS´99. S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton B. M. Horowitz.

[4] S.J. Dalal, A. Jain, J.M. Leaton, C.M. Lott, G.C. Patton, B.M. Horowitz. Model-Based Testing in Practice. IEEE Int. Conf. on Software Engineering. ICSE '99 Los Angeles CA.

[5] J. Garbajosa, O. Tejedor and M. Wolff. Natural language Front End to Test Systems. AIRTC'94 Symposium in Artificial Intelligence in Real Time Control. October 1994.

[6] Juan Garbajosa, and Mikael Wolff. Automatic Generation of Satellite Test Procedures. Proc. International Symposium Computing Tools, Systems Engineering and Competitiveness. Association Aeronautique et Astronautique de France (AAAF)Paris, March 1997.

[7] Axel van Lamsweerde,Laurent Willemet. Inferring Declarative Requirements Specifications from Operational Scenarios. IEEE Transactions on Software Engineering. Vol. 24, No. 12: December 1998, pp. 1089-1114

[8] Alistair G. Sutcliffe, Neil A.M. Maiden,Shailey Minocha, Darrel Manuel. Supporting Scenario-Based Requirements Engineering.IEEE Transactions on Software Engineering. Vol. 24, No. 12: December 1998, pp. 1072-1088.

[9] Vijay K. Vaishnavi, Gary C. Buchanan, and William L. Kuechler Jr. A Data/Knowledge Paradigm for the Modeling and Design of Operations Support Systems. IEEE Transactions on Knowledge and Data Engineering, Vol. 9, No. 2, pp. 275-291. March-April 1997