IPSN

DES.. 184e.

RAPPORT

RAPPORT DES/184 e

CLAIRE, AN EVENT-DRIVEN SIMULATION TOOL FOR TESTING SOFTWARE

RAGUIDEAU J., SCHOEN D.,
HENRY J.Y., BOULC'H J.

cea INSTITUT DE PROTECTION ET DE SURETE NUCLEAIRE
DEPARTEMENT D'EVALUATION DE SURETE

RAPPORT DES/184 e

CLAIRE, AN EVENT-DRIVEN SIMULATION TOOL FOR TESTING SOFTWARE

RAGUIDEAU J.[*], SCHOEN D.[*],
HENRY J.Y.[**], BOULC'H J.[**]

Specialists' Meeting on
"Advanced Control and Instrumentation Systems
in Nuclear Power Plants :
Design, Verification and Validation"
(IAEA, IWG-NPPCI, IWG-ATWR)
ESPOO (FINLAND), 20-23 juin 1994

[*] LETI, DEIN
[**] DES/SAMS

Juin 1994

CLAIRE
An event-driven simulation tool for testing software

Jacques Raguideau
Dominique Schoen

Commissariat à l'Energie Atomique
LETI (CEA - Technologies avancées)
DEIN - CE/S F91191 Gif-sur-Yvette Cedex

Jean-Yves Henry
Jacques Boulc'h

Institut de Protection et de Sûreté Nucléaire
Département d'Evaluation de Sûreté
CEA CENFAR
B.P. 6 - 92265 Fontenay-aux-Roses

FRANCE

RESUME:

CLAIRE fournit un environnement purement logiciel permettant de valider des applications temps-réel distribuées, soit au niveau des spécifications, soit au niveau du code.

L'outil offre les facilités graphiques de modélisation de l'application et de son environnement; il effectue de façon très efficace la simulation du modèle entré et assure le contrôle de l'évolution des données dynamiquement ou en temps différé.

ABSTRACT:

CLAIRE is a software tool created to perform validations on executable codes or on specifications of distributed real-time applications.

The tool provides graphics facilities to model the application and its environment; it produces an effective simulation of the model entered and manages the simulation process either in real or deferred time.

1 - INTRODUCTION

The Claire software tool was initially designed for use by the Institute for Nuclear Safety and Protection (French acronym IPSN), a CEA body with the task of performing safety analyses to assess the effectiveness of safety measures taken by the operators of nuclear installations.

In connection with this role, the Institute for Nuclear Safety and Protection has to study dossiers for these installations. Some systems are so critical that a simulator is needed to perform special tests using the software installed.

The IPSN needed a tool to describe the software environment in order to evaluate how the programs behaved in response to normal inputs or faults. The tool had to be able to simulate real-time distributed applications without altering their behaviour with respect to time.

The variety and complexity of the environments and the degree of flexibility required made it difficult if not impossible to perform the simulation using real equipment; it was therefore decided to perform all simulation using a computer.

The need for modelling and simulation results from a concern shared by all persons who produce complex programs: the transition from the unit test stage, where debuggers are sufficient, to validating a real-time application in its entirety.

This concern becomes crucial in the case of "safety" programs, which can have catastrophic consequences if they fail. The specification phases for these most critical programs are already covered by methods for verifying them : Petri networks, the synchronous languages provides solutions for their own particular areas: time-oriented verifications, verification of safety properties etc.

Besides the fact that these methods are not always applicable to the size and complexity of the applications, they do not allow validation of executable codes actually used in critical installations.

CLAIRE can be used both to verify the safety properties by modelling the specifications, and also to validate the final code by simulating the behaviour of its equipment and software interfaces. It can be used to observe and provide dynamic control of the simulation process, and also to record changes to the simulated data for off-line analysis.

2 - PRINCIPLES BEHIND THE TOOL

The principles on which CLAIRE is based are modelling by data flows and the event-driven simulation of this modelling. 'Event-driven" simulation was chosen for reasons of code validation: the production of an event is a reaction associated with the transitions of the code through expected states: value of the PC, saving to a given address etc. This option can be applied without any problem to specification validations.

2.1 - Modelling

A data stream model can be expressed quite easily in graphical form. The modelling system adopted in CLAIRE consists in setting out a top-down hierarchy of the various elements of the application, following method similar to SA/RT.

Each unit is shown as a box, each box can be broken down into the boxes which make up the next level below. The boxes of the last level are procedures written in C, setting out the functional role of the box.

These boxes exchange flows of information shown on the diagram by lines. These flows can be either continuous or discrete. Any modification to these discrete flows will trigger the end procedures which are the ultimate destination of the streams.

2.2 - Event-driven simulation

When any flow is modified, an event is created. The informations carried by an event are the variable associated to the flow, the value to be assigned to the variable, the date (the time when the assignment will take place) and finally the list of procedures to trigger during this assignment. The events created by the procedures are placed in a buffer and scheduled according to their date

3 - ROLE OF THE TOOL

CLAIRE can be used to validate specifications and executable code.

3.1 - Validating specifications

The tool can be used to produce models of systems from their specifications. The elements involved in the description are depicted according to the degree of precision required: they can be shown as black boxes if we are only interested in their interactions with the outside world, or as white boxes if we are interested in their internal functions. A 'black box" element can, during more precise modelling, be replaced by its "white box" model, and eventually even by its real algorithm. This means that applications can be validated incrementally.

CLAIRE can be used to produce models similar to Petri networks and hence to validate the real-time behaviour of the application modelled. This transposition consists in associating functions with the transitions and tokens with events: a function is triggered when a token (a data item event) reaches an input place for the transition it is performing. The function evaluates its conditions from the input tokens, and if the conditions are met the function absorbs the tokens, in other words reinitialises its input conditions, executes its body of program and then places tokens at the output points, in other words it creates other events which are the input tokens for other functions. The advantage of this transposition is the possibility of describing and executing large applications, which is difficult with Petri networks.

3.2 - Validating software

The tool can be used to test executable programs distributed over one or more microprocessors, and to dump their behaviour within upper and lower bounds and assess their defect resistance.

The boxes then depict the environment of the programs to be validated: equipment or program interfaces, changes to installation parameters etc.

Boxes no longer associated with the environment but with a particular test strategy can be incorporated into the model: observers in charge of detecting the occurrence of specific events or oracles for comparison with a reference behaviour pattern.

The environment is depicted in accordance with the method set out above: the program to be tested gives rise to a specific box in the graphical description and to a specific procedure (as understood in the context of the event-driven simulation). Inputs and outputs of this procedure correspond to exchanges between the microprocessor and its environment. The procedure is performed by a microprocessor simulator which receives the parameters of the code to be run together with a description of the correspondence between the exchanges and the internal events.

The tool can, even in this case, be used as a code debugger since it allows access to the various registers and values stored in the internal memory of the microprocessor.

4 - TOOL COMPONENTS

CLAIRE is composed of a graphic editor, a code generator, a simulation kernel and a result analyser.

The *graphic editor* provides facilities for drawing boxes and lines, moving a box and its decomposition from one view to another, performing cut and paste operations...It performs checking on input data consistancy.

The *code generator* analyses the graphic description entered by the operator. A box naming formalism allows it to identify "module" boxes that will be C compilation units. It automatically generates the declaration statements of each module and the procedures declaration; it inserts the C code entered by the programmer.
These modules are compiled. The link edit puts together these generated modules and those composing the simulation kernel.

The *simulation kernel* uses a time-stamped event buffer. The events are created by the procedures according to immediate or deferrred modifications of the flows values. They are inserted into the event buffer, with a rank corresponding to the date they are to be executed. On due date, the simulator kernel assigns values carried by the events to the flows, runs the procedures associated to the triggering flows.

The *results analyser* makes it easier to exploit the huge files contaning the flows evolution along the simulation. This off-line analysis of results completes the on-line analysis performed by observers during the simulation. The result analyser allows a navigation through the events generated during the simulation to make a more precise analysis and possibly detect unwanted situations.
The analyser provides graphic facilities, such as plotting curve , drawing timing diagrams or printing arrays of numeric values. Zooming facilities and scale modification are also provided.

5 - PROGRAMMING THE MODEL

Programming consists in describing operation of the end boxes of the graphical description; only the executive part of the procedures is to be carried out by the programmer: the declarative part is taken care of by the code generator. C was chosen as the programming language for procedures modelling the environment. It is also the language in which the simulation kernel was written (this simplified the task of integrating everything).
The C language has been chosen first for its large scale diffusion but also for its suitability to the problem : separate compilation, variables scope , data structures, and almost functions manipulation, which allows to associate a function address to an event, and therefore to achieve a performing simulation process.

Furthermore, by opting for C language to develop his procedures, the user has at his fingertips all the power of this language and all the possibilities it offers for interfacing

with existing products. He can call on all the procedures of the "run time library", use XWINDOW to animate his simulation etc.

There is no need to learn any specific language dedicated to the test in order to develop the model which will form the simulation: any C programmer can develop it.

6 - TOOL APPLICATIONS

6.1 - Experiment to evaluate 1E grade software systems

6.1.1 - Background to the evaluation approach

The process of licensing the operation of nuclear plants includes obligatory stages which give rise in particular to an in-depth examination of the instrumentation and control system. This examination takes into consideration the technological aspects (integrated circuits and software) which were chosen by the manufacturer for programmed systems performing safety grade functions.

The technical support body (IPSN) of the safety authority (Nuclear Installations Safety Directorate) is in charge of conducting all investigations it sees fit in order to ensure that the methods and technology used by the manufacturer and operator guarantee the expected level of safety, for 1E grade software systems, and ensure a sufficient degree of testability and maintainability. In order to do this the IPSN pays particular attention to the following points:

- rational and thorough methods of developing programs following a detailed quality assurance plan (for documentation and code);
- strict programming rules to produce a testable and maintainable program (code);
- tests conducted to ensure a sufficient coverage rate both at the supplier's premises and on site (simulation).

Programs are evaluated mainly by taking into account the results of the following two tests:

- analysis of documents (software design, quality and maintenance procedures),
- dynamic analysis using the CLAIRE workbench, performed on the binary code supplied by the software manufacturer.

The purpose of the dynamic analysis is to show the behaviour of the program when subjected to:

- stimuli whose values are taken from the conditions within the nominal operating range for the system using the program being tested (consistency check),

- stimuli whose values correspond to various scenarios of malfunction by the system using the software being tested (ruggedness check).

The approach adopted to evaluate PWR reactor protection system software is set out below for the part concerning the dynamic analysis.

6.1.2 - Consistency check

The consistency check is used to verify the values taken by system outputs (for example controlling a trip) when the inputs assume values chosen by the analyst from the nominal operating range for the protection system.

The CLAIRE workbench is used to simulate operation by running the binary program without the need for equipment (central processing unit card, peripheral cards etc.) used on site. It can:

- make up an environment which reproduces exchanges between each microprocessor and the circuits (clocks, communication circuits, memory etc.) which are associated with it in each unit of the protection system installed on site;
- run the binary programs of the units of the protection system by means of a microprocessor simulator, generating special files plotting all interactions between the microprocessors and their environments, with the run time being recorded,
- display in synthetic form (timing diagrams, curves etc.) the values assumed by the different variables being monitored in order to analyse the simulation results.

The environment of the binary program, and of the microprocessor which runs it, is reconstructed by developing specific programs which replace the equipment called up by these programs. This is achieved mainly by using a graphical description based on the SA/RT method.

The values of input variables given by the test campaigns designed for this consistency check are taken into account when the programs are run.

First of all, some of the normal operating conditions of the protection system will be selected to ensure that the model obtained from the environment developed for the purposes of this study is adequate. Then, system programs will be run to check on the behaviour of the system under specific operating conditions (degradation of the 2-out-of-4 voting logic for example) provided for in the specifications.

The simulated system and its test campaigns will be reused to check that each version of the programs operates well and without regression.

6.1.3 - Ruggedness check

The main purpose of this study is to evaluate the performance of programs from the representative whole when subjected to test campaigns, determined in advance, which represent malfunctions of the protection system or of the systems which provide it with information. The test campaigns are focused on the critical or sensitive components

discovered during the previous stages. The check makes provision for an analysis covering an aspect complementing the tests already carried out by the manufacturer.

This study makes use of simulation tools defined for the consistency check to produce a fuller environment so that certain internal program variables representative of the malfunctions in question can be dealt with.

The results of the simulations obtained using the different ruggedness test campaigns must be analysed to identify the state of each program output variable involved in these simulations.

The analysis is continued on the system outputs in order to identify the consequences of the malfunctions introduced and to draw conclusions from them regarding whether or not the system performs adequately in view of the missions it has to perform.

6-2 - Evaluation of a installation and control architecture

The ESCRIME project, currently in development phase in the CEA, consists in evaluating instrumentation and control architectures for future nuclear power plants. These instrumentation and control applications are very large distributed systems, in which real-time concerns are preponderant, due to the increase of automation in the systems, process optimisation constraints, and safety requirements.

The instrumentation and control applications manage a hierarchy of objectives :

- At the highest level, the global objective is expressed as an external request, for example, it can be the plant ouput power profile.

- This global objective is dynamically converted into a set of current objectives (e. g. put the plant in the hot shutdown state), depending on the reactor state, or on the subsystems availability.

- The current objective is finaly translated into functional objectives, related to physical data, such as pressure, temperature, reactivity, ... These objectives are met by bringing into play physical subsystems and low level regulation loops.

Reports and error messages coming from a lower level make the upper level review its strategy to meet its objectives, or, if it is not possible, to report the error.

This leads to run complex decision making software.

The evaluation of such a system requires in first place a tool to evaluate its dynamic behaviour, by playing predefined or interactive scenarios, while using a graphical representation of simulated physical data.

The CLAIRE tool is used to perform these evaluations : the system is described in the CLAIRE formalism using hierarchically organized boxes, exchanging data and control

flows. Last decomposition level boxes are written in C, or can incorporate existing complex code, for example the one simulating the physical process.

The simulation phase is interactive, some variables of the simulated system being represented as graphical objects, such as potentiometers, indicators... These elements can also influence the simulation process by reflecting operator's actions on the simulator variables and controls.

The user can dynamically choose the variables to be represented, in order to be able to deal with large applications, incorporating a large number of variables.

Observers can be added to the simulated system, in order to perform online detection of situations such as deadlocks, indeterminisms, CPU contention, non-respect of timing constraints, non-respect of events sequences.

A coverage test can be done as well.

The simulation results allow to play it back, for example to look for the cause of an abnormal situation and to display the results.

7 - EXTENDING POSSIBILITIES

To meet the requirements of the above project, work is currently in hand into extending the possibilities for operator interaction during the simulation process. At present, dynamic monitoring of the simulation process is carried out in text mode. This monitoring would be simplified by the use of graphical displays and controls.

There are plans to:

– allow the operator to select the most suitable representation mode (potentiometers, indicators, text etc.) for monitoring the dynamic evolution of his variables;

– modify the current value of a variable using this same mode of representation,

– extending current simulation control by allowing the possibility of operating in step-by-step mode, and alteration of the speed of execution etc.

8 - AVAILABILITY

CLAIRE is written in C and XWINDOW; it is available for Vax/VMS machines and is currently being translated for use on SUN stations.

Microprocessor simulators currently available on VAX machines are those of the M6800, M68000, M68010, M68020, I8051 and I8086 models.