This article has been accepted for publication in a future issue of a journal/proceedings, but has not been fully edited. Content may change prior to final publication. Citation information: Burford, M.A.J., Belli, F., CADAS: A Tool for Rapid Prototyping and Testing of Embedded Software, Proc. IEEE Symposium on Application and Assessment of Automated Tools for Software Development, pp. 27-34

CADAS: A TOOL FOR RAPID PROTOTYPING AND TESTING OF EMBEDDED SOFTWARE

M.A.J. Burford

ESG/FEG
Vogelweideplatz 9
D-8000 München 80 FRG

F. Belli

Hochschule Bremerhaven
Columbusstr. 2
D-2850 Bremerhaven FRG

Abstract

The development of embedded software systems has todate been problematic and subject to high costs due to the lack of non-project specific tools which support the development of such systems over all life cycle phases, from requirement validation through to "in-service" support. In order that prototyping may be carried out in a cost effective manner, all interested parties ought to be in the position to define and evaluate the possible solutions in a dynamic closed loop user friendly manner.

In an attempt to meet the above requirements, and to provide a flexible framework for future needs, a design has been conceived and is being realised through the development of CADAS.

Introduction

The problem of rising costs in the development of embedded software may only partly be combatted via the language of implementation. A concerted effort will only be successful when the programming support environment is also taken into consideration. In this paper a tool will be briefly described which is aimed at providing a solution to this problem.

Because the development of embedded software poses particular problems, it is important first to define the special needs of a complete software production environment. The points that were felt to be of prime importance in the development of CADAS are outlined in the following section. The key elements in the concept of CADAS are followed by a brief description of the basic structure of CADAS. The paper concludes with an outline of the successes of CADAS todate, an indication of the areas requiring further development and finally future topics.

Review of the Role of Prototyping in the Development of Software

Prototyping has, for many years, been used in one form or another in the development of complex embedded software. However, its impact has for many reasons been minimised, with the net effect that its use has only focused on isolated elements of the design during the concept validation phase (1). The resulting fragments were incompatible and had a short life expectancy. The software tended to be written on an "ad-hoc" basis and was not subject to the same configuration control that was exercised on the target software.

ESG in 1980, faced with the joint problems of rising software development costs and the need to quickly, effectively evaluate "in-service" system refits undertook a review of present tools and methodologies used in the development and support of such systems. With the growing interest in fault tolerant systems (2) and in particular the topic of exception handling (3) the importance and problems associated with prototyping were acknowledged as taking on a new importance.

In addition, the then available validation and verification tools were not only project specific, but also life cycle phase specific, focusing mainly on the detail design phase. An analysis of the problems associated with both topics, prototyping and the verification and validation of real time closed loop systems, underlined the need for a tool which fully supported both activities. An important product of such a common tool would be the provision of full traceability on a comparable basis from requirement validation through to "in-service" maintenance for any given project.

As a result of the review, a design for such a tool which would enable the development of prototypes and the simulation of the appropriate operating environments, in a cost effective manner (i.e. a "software breadboard") was conceived and is in the process of being implemented.

Development of Embedded Software poses Special Problems

Embedded software, i.e. systems in which a computer is directly connected to some apparatus or plant which it monitors and/or controls (4), is particularly problematic due to the need to provide suitable environments and stimuli in a time critical manner. Unfortunately the operational hardware is specialised, expensive to source and not available until late in the development cycle. They have severe timing and memory contraints which only serves to reinforce the need for a good prototyping capability (5). The problem is further complicated by the fact that it is not easy and often impossible to manipulate the hardware such that

i) faults or noisy signals are generated as and when desired

and ii) the system downgrades into reversionary modes when required.

The resultant target software is complex due to its multipath nature and requires special environments to support its development.

Fortunately, the mechanism for providing a rapid prototype is very similar to that of providing a suitable simulated operating environment. Indeed the frozen elements of the prototype provide a base line version for the validation and verification. This provides the link, which until now has been missing, between the requirements and the test system (6).

Attributes of an Idealised Support Tool

Prior to the conception of CADAS, the then available tools were reviewed in order to establish if any of them could be used as a basis to meet the identified requirements, which could be summarised by the following key features.

Minimum maintenance and Operator Skill

The rising cost of maintaining software has been clearly recognised, indeed the problem was a prime motivator in the development of ADA. In the case of software whose function is one of support as opposed to one of a saleable product, the need for minimum maintenance takes on a greater meaning, however, unfortunately the need to provide such support receives little or no attention.

The majority of test systems todate, have beem orientated around the activities of a particular life cycle phase and as such tended to be tailored to the needs of certain particular fields of interest. As such, they are then of questionable value in later phases such as "in-service support" as the personnel involved will more than likely have different fields of interest.

Applicable to all life cycle phases

An idealised tool should not only provide the link between the detail design phase and the "in-service" support but should also provide one between the requirements, the validation of the design via prototypes and so on through to the verification of the final product.

Instrumentation of target software

Instrumenation may be achieved either by:

- temporary altering the target software to include the required instrumentation. This approach whilst common is not to be recommended as it is often difficult to assess the side effects of the temporarily included code.

- utilising the data which is present in the I/o of the target system to indicate the instantaneous values of the various parameters which affect the program flow control. Whilst this is to be preferred to that of above, the need to provide software model stubs within the test environment, which simulate the calculation of the various conditional parameters may not only generate time loading problems for the test facility but may be subject to erroneous assumptions.

- At project definition phase, ensure that all condition control parameters are available via the I/o. Whilst this will have the initial disadvantage of increasing the target system I/o, this is more than offset by the simplification of the test environment, requiring only the logging of the appropriate values and times. In the case of exceptions, the first action within the handler could be to register its activation. This would not only provide instrumentation during the development phase but would then provide the basis of an "in-service" health monitoring mechanism.

Replay and analysis of "in-service" data

Due to the interplay between data which leads to possible fault situations, the majority of "in-service" reports or queries tend to be inadequate or worse still, misleading. It is therefore of prime importance that an idealised support tool should be capable of replaying and analysing "in-service" recorded data.

Easily reconfigured

In order that the idealised tool be applicable to all projects it must, by definition be easily reconfigurable. However, this feature also plays an important role within a given project. The operating envelope of a target system is namely broken down into a set of smaller envelopes or worlds. The target system may then be verified and validated by exercising it around the identified critical operating points within the appropriate world. Such a test approach is only cost effective when an easy reconfigurable mechanism is available to assist the definition of the appropriate worlds.

Testing coverage

Whilst the use of such methods as program proving and symbolic execution for verification purposes is highly questionable, they have until now possessed a great advantage over that of testing methods, in that it is fairly easy to generate a measure of coverage (7), (8), (9) and (10). The traditional criticism of program testing methods is that they can only show the presence and not the absence of errors, and as such it is difficult to visualise a measure of test coverage. However, such work as outlined in references (11) and (12) have shown this not necessary to be the case.

Using such methods as symbolic execution in conjunction with specification analysis techniques to identify critical points in a given world and then combining the sum of such points for each world with the total number of worlds required to represent the operating envelope of the target system a 100% test coverage for a given target system may be defined. A test coverage measure may then be achieved by considering the completeness of the definition of available worlds in conjunction with the number of critical points examined within each world. The critical points in any world may be identified according to such criterium as defined by references (12), (13) and (14).

Manual and automatic control of tests

Whilst during the development phase many

scenarios will be driven via a manual input,in order to achieve repeatability during verfication and validation,control of the test must be under automatic control. In addition, in order that certain scenarios are met, certain parameters must take certain values under certain conditions which are not capable of being simulated via human inputs in an open loop manner. The need therefore arises for a test descriptor language. (15)

Rig efficiency

As previously mentioned,hardware tends to be expensive to source and as a result "test rigs" tend to be scarce. A critical item, therefore, from a project management point of view is always, "available rig time". An idealised support tool must therefore ensure that the time available is used in an efficient manner by constraining the user to work in a documented, methodical manner.

The Concept of CADAS

The key features of CADAS which enable prototyping to be carried out in a cost effective way are

- the cross referencing of the prototype specification to the requirements
- the provision of an events language
- the capability to generate and insert models automatically into CADAS.

The primary aims of verification and validation may be characterised by the following three tasks

- the exercising of the preferred modes of operation in real time
- the identification of the exceptions within a system with the appropriate elements of the environment
- the exercising of the identified exceptions, i.e. interactive execution of the exceptions in an economic simulated environment.

CADAS has been designed primarily to permit the designer/user to perform these tasks with the minimum of special to project hardware, in a "hands on" environment. This is achieved by utilising the key prototyping features outlined above.

Events language

A highly desirable means of automating a "cause and effect" type of testing is via an events language (16). An events language is a formal means to define values certain variables should take at which time and under what conditions. Whilst most of the requirements for such a language may be fulfilled by any one of the many available test description languages , they are all left wanting in one form or another. Many are open loop with no conditional control. Others whilst possessing conditional control, permit data insertion to be carried out in a static manner (6), (17).

The event language "TADLER" (Testing using the ADA Language for Event Reproduction) is under development in order to meet the requirement for an automatic, conditional, closed loop, self-documenting test descriptor language. It was decided to base it upon the ADA language in order to ease:

- user interface in future
- the migration of the exception handler defined in the events into the target software
- and to reduce development time

Events may be visualised as a set of subroutines, separated by time conditions, with the language itself breaking down into three elements.

- condition and action control flow statements
- general event objects and actions
- target system specific objects and terms

The first element may be directly taken from ADA, whilst the latter two may be envisaged as sets of ADA procedures. The latter of the two is project specific whilst the former is target system specific eg. avionics, robotics etc. The structure of the events language dictionary reflects these three elements. The first element remains static and unique, whilst the third being defined for every project. The number of versions of section two will naturally reflect the number of different types of projects being supported.

An example of a TADLER event file for an avionic application is given in figure 1. The aim of the file is to provide the necessary stimuli and control to simulate an attack on a ground target.

```
 With FLTPRO use MISSION_1
          reset CLOCK;
          press PLN_BUTTON on TV/TAB_2;
 after 2 secs
          press NAV_BUTTON on TV/TAB_1;
 after 3.5 secs
          press KEY_3 on TV/TAB_1;
          press MAIN_BUTTON on NMC_PANEL;
          press AUTO_BUTTON on NMC_PANEL;
          engage BHH_MODE on AUTOPILOT;
 if WAYPOINT_A is OVERFLOWN then
          goto LABEL;
 endif;
 after 2 secs
          press IN_BUTTON on NMC_PANEL;
          disengage AUTOPILOT;
          perform a HIGH_LOFT;
 EVENTS inhibit;
 <<LABEL>>  after 1 secs
while RANGE from AIRCRAFT_POSITION to WAYPOINT_B >> 20 nm loop
       null;
end loop;
start RECORDING;
```

Fig. 1 A set of events taken from a TADLER avionic file

The package FLTPRO contains a set of procedures which define the various flight data cases. After initially setting the events clock to zero, a series of button presses select the appropriate navigation mode within the target system. A test is performed to ensure that the aircraft only performs the attack if the aircraft has not overflown WAYPOINT A. The position of this navigation mark is defined in the procedure MISSION_1.As the aircraft approaches the next navigation point, WAYPOINT B the events processor waits until the aircraft is within a range of 20 nautical miles before switching the data recording device on.

The problem of exception handlers is one future area where event languages may play a major role. Irrespective of whether such mechanisms are

desirable or not (18), their design, implementation and verification will be problematic, due to the need to :

- generate the error condition in a controlled repeatable manner
- prototype and evaluate the handler

An iterative detail design process within the target software is certainly undesirable (19).

The development of exception handlers, external to the target code in the form of an events file, may reduce development problems. In this case use is made of two TADLER event processors, the first to generate the stimuli to force the exception to be raised and the second to provide the definition of events which should occur within the exception handler.

```
with WORKPROGRAM use SCHEDULE_4

     reset CLOCK;
     press TEST_BUTTON on CONTROL_PANEL ;

after 4 secs

      NUMBER_OF_SCREWS:=4;
      PRODUCT_FLOW_RATE: = 12·8 FT/s;

after 5 mins

      perform ACTION_1;

after 3 secs

      while NUMBER_OF_SCREWS>1 loop

          null;

      end loop;

      start RECORDING;
```

Fig. 2a Environment stimuli events file for a programable robot.

In the above example, the package WORKPROGRAM contains a set of procedures (SCHEDULE_1, SCHEDULE_2 etc) which defines the various tasks the robot may be programmed to carry out. When no more screws are available, the target software should raise the exception NO_SCREWS. The exception handler stub within the target systems then performs a rendezvous with the second events processor

```
exception NO_SCREWS

      NO_SCREWS_MARK: = NO_SCREWS_MARK +1;

      STORE_TIME;

      TEXT_10.PUT('DOWN TOOLS, NO SCREWS');

end;
```

Fig. 2b An events file to represent a possible exception handler

Once the exception handler's performance meets the appropriate requirements the contents of the event file may be automatically embedded in the target code.

Automatic generation and insertion of models into CADAS data base

The second key element for prototyping is the method of generating and inserting into CADAS new models.

The test data used for validation and verification should ideally be generated from the requirements (20). The role of the models in CADAS is to act as test data generators. As a project advances, the requirements will be refined. These refined requirements will then be used to generate new model versions.

In order that a model may be used by CADAS its structure must comply with certain restrictions, such as memory mapping conventions, pipe line process etc. In order to ease the process of model generation and integration, whilst at the same time providing a means of controlling the model activity, it was decided to automate the process.

The generation of a model starts by the user defining the model's required attributes using a formal specification language (eg. EPOS (21)). Once the specification is defined as complete, and containing no conflicts, the additional information such as subsystem membership, version number and I/o memory mapping information is retrieved from the appropriate elements in the data base. Once the model has been tested and accepted, it is automatically inserted into the model library and the appropriate real time table is updated. Both of which are elements of the data base. The above process is presented in Fig.3.

Once a model has been accepted it can not be directly modified or removed by unauthorised persons. Should a user wish to use a model with a modified function, then he must use the above process to generate a new model version. The data used to test the functionality of the model is generated automatically via an analysis of the specification.

Basic Structure of CADAS

The above two features are only a part of CADAS, which is an attempt to provide a total environment for prototyping, validating and maintaining embedded software. The basic structure of CADAS is shown in figure 4 in the form of Petri nets. As can be seen the flow of control and data have been clearly separated.

Upon entering the system, the user defines via a menu, which target system he wishes to exercise. This indirectly defines the elements of the data base eg. appropriate model library, target system information files, event language dictionary etc, he is permitted to access. He then selects his required activity or mode via means of a further menu as shown in fig. 5

CADAS defines five independent modes, each of which is a collection of tasks, and the user enters or leaves them via the moding executive menu. In an attempt to structure the process of defining a test, the activitiy has been split into two phases - preparation and initialisation.

Preparation mode

Due to run time loading constraints, it is impossible to encapsulate the operation envelope of the target system in one test environment,

to any great depth.

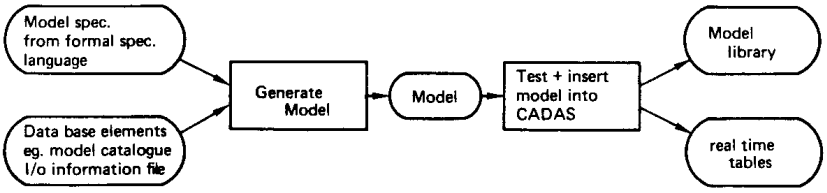
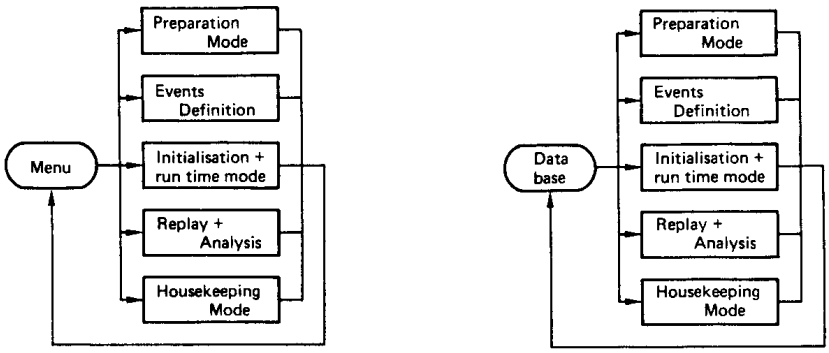


Fig. 3 Model generation and insertion



Control Flow Data Flow

Fig. 4 Basic structure of CADAS

Indeed usually there is no need to do so. The operating envelope must therefore be broken down into a set of test environments or worlds. The definition of these individual worlds is the function of the preparation mode, and is achieved through the selection of :

- the required models
- the required parameters to be displayed on-line during the test
- the required hardware units eg. MIL 1553B bus interface
- the required events file
- the required recording features
- generation of preparation documentation

MODING EXECUTIVE

SELECT REQUIRED MODE BY TOUCHING SCREEN ACCORDINGLY

1. PREPARATION MODE
2. EVENTS DEFINITION MODE
3. INITIALISATION AND RUN TIME MODE
4. REPLAY AND ANALYSIS MODE
5. HOUSEKEEPING MODE
*. HELP

Fig. 5 Moding executive Menu

Events definition

The activity of defining and debugging an events file is a labour intensive activity, and furthermore the generated file may be used by more than one preparation. In order to reduce the required work input, the user has been provided with an editor and events exerciser.

Initialisation and real time run

The exploration of any one of the worlds defined in the preparation mode will encompass a number of tests. Each different test being defined by its initial start conditions. Their definition is the function of the initialisation mode. Whilst the world or test environment definition will have a possible indefinite life, that of the initialisation is short lived. Its life being terminated by the user defining a new initialisation data set, following a run, or by the user exiting from the mode. Prior to entering the real time run, an active timing check is made. Providing this check is positive the real time run is entered. Intertask communication during the runtime is via the global data area, a fixed part of memory. In order to reduce post test data saturation, full use of on line monitoring is used.

On entering the mode the following actions are carried out

- identification of which preparation data set
- hardware checks
- initialisation of data
- initialisation of recording medium
- timing checks
- generation of initialisation documentation
- entry into real time run

Following the real time run, the user, if he so wishes may obtain a dump of the contents of the global data base prior to and after the real time run.

Replay and analysis

Using this mode the user may not only replay the test in pseudo real time, he has also the possibility to present recorded data in a graphic or tabular form. By replaying the recorded data in conjunction with the appropriate models, he is able to generate and inspect the time dependent value of internal variables.

House keeping mode

This mode is used to perform general data base activities such as model generation and insertion, deletion of preparation files, updating language dictionaries and system I/o specification files. By the definition of a new project, this mode provides the mechanism by which the initial data is defined such as part three of the events dictionary, and the project coverage.

CADAS Use and Development

Use todate

The test environment, as defined by the preparation mode, may be envisaged as being made up of a set of subelements eg. sensors, software requirement models, etc. Since the prototype of CADAS was built as a multi task single processor system, the real time run was made up of a set of tasks primarily based upon run frequencies. However, in the multi processor system under development, the run time tasks will be based upon the functionality of the various subsystems. The prototype has not only validated the concept but has also successfully been used to test flight safety critical software. The work not only entailed the exercising of the target software but also included the detection and identification of discretes released by it, to within 20 microseconds.

Present development work

Although the basic concept of CADAS has been validated, certain areas of work still remain to be completed or realised. Todate over 20 man years have been invested and an estimated 8 to 10 man years remain to complete the present defined work. The areas requiring further activity may be summarised as follows

1. Completion of the facility to automatically generate new models and insert them into CADAS
2. Completion of the event language
3. refinement of the definition of test coverage
4. realisation of the appropriate software to support test coverage reporting

Future work

It is foreseen to include such methodologies so as to be able to establish the appropriate software metrics eg. reliabilitiy, complexity, flexibility (11) etc.

At the moment, the development of such features are still in a state of flux. However, provision has been made within the replay and analysis mode to include such important features once they have achieved a matured state of development, that is acceptable to industry.

Conclusion

An automated tool for developing embedded software has been presented which allows the target system to be modelled and permits all interested parties to exercise and evaluate the target system, not only in the preferred mode of operation but also in all degredated modes in a "hands on" environment.

This is achieved by means of the event language and the automatic generation and insertion of models into the appropriate libraries.

CADAS may be characterised by the following key features:

- the ability to perform real time closed loop testing
- through the use of a disciplined approach the appropriate elements of the target system and its operating environment may be simulated in a cost effective manner
- the tool is not only non project specific but also is not constrained to a particular life cycle phase
- through the nature of its structure it is easy to learn and use, maintain and when and where necessary be furhter developed.

The true power of CADAS, of course, will only be realised when it is used in conjunction with the other important elements in the programming support environment. One of the most significant environments will most certainly be the ADA Programming Support Environment (APSE (22)).

The availability of suitable prototyping and verification and validation tools will have a direct influence on the life cycle costs of embedded software and as such should be included in a "base line" APSE. However, the provision of such tools require a significant investment and todate this important element has not received the attention it deserves and needs. The development of CADAS is an attempt to plug this omission.

References

(1) Special Issue on Rapid Prototyping, Working Papers from the ACM SIGSOFT Rapid Prototyping Workshop Columbia, Maryland (1982)

(2) Belli F. et al " A strategy for the development by means of regular expressions". Proceedings of International Symposium on fault tolerant computing. IEEE Computer Society (1983)

(3) Cristian F. "Reasoning about programs with exceptions". Proceedings of International Symposium on fault tolerant computing. IEEE Computer Society (1983)

(4) Pyle I.C. "The ADA Programming Language", Prentice Hall , May 1981

(5) Burford M.A.J. et al "CADAS - A computer aided design tool for avionic systems" AGARD - Advanced concepts for avionics/weapon systems design, development and integration, April 1983

(6) Gmeiner U. "Dynamic Analysis and Test Data Generation in an automatic test system". Workshop on Reliable Software, Hauser Verlag, München 1979

(7) Ramamoorthy C.V. "Techniques in Software Quality Assurance" Proceedings Software Qualitätssicherung, Teubner Verlag, Stuttgart, March 1982

(8) Morten, Taylor F.N. " A survey of methods of achieving reliable software" Computer Feb. 1977

(9) Benson et al, " A software quality assurance experiment" GRC Report, Proceedings of Software Quality and Assurance Workshop, San Diego, 1978

(10) Myers G.T. "The Art of Software Testing", John Wiley and Sons, New York 1979

(11) DACS, Rome Air Development Center, Quantitative SW Models (1979)

(12) Goodenough J.B. et al, "Toward a theory of test data selection", IEEE Trans. on SW Engineering June 1975

(13) Howden W.E.,"Theoretical and Empirical Studies of Program testing",IEEE Trans on SW-Engineering, July 1978

(14) Infotech, State of the Art Report SW testing 1979

(15) Miller E.,"SW Testing and Test Documentation" (Workshop Report), IEEE Computer, March 1979

(16) Fairley R.E. "Tutorial: Static Analysis and Dynamic Testing of computer Software" Computer April 1978

(17) Ramamoorthy C.V. "Testing large Software with automated software evaluation systems" IEEE Trans, March 1975

(18) Hoare A. " The emperor's new clothes" CACM 1981

(19) Barnes J.G.P. "Programming in ADA", Addison-Wesley 1982

(20) Miller E.F. "Dynamic Analysis and Test Data Generation in an automatic test system", Workshop on Reliable Software, Hauser Verlag, München 1979

(21) EPOS - "Entwurfsunterstützendes Prozess-orientiertes Spezifikationssystem" - developed at Stuttgart University under the leadership of Prof. Dr. Lauber.

(22) Buxton J.N. Stenning V. "Stoneman", Feb. 1980