Interval Reduction through Requirements Analysis

Gerard J. Holzmann*
Margaret H. Smith*

ABSTRACT

Our premise is that a notable part of the delays that can be encountered in system design projects are caused by logical inconsistencies that are often inadvertently inserted in the early phases of software design.

Many of these inconsistencies are ultimately caught in design and code reviews, or in systems testing. In the current design process, though, these errors are rarely caught *before* the implementation of the system nears completion.

We show that modeling and verifying the requirements separately, before system design proper begins, can help to intercept the ambiguities and inconsistencies that may otherwise not be caught until testing or field use. By doing so, one can prevent a class of errors from entering the development phase, and thereby reduce time to market, while at the same time improving overall design quality.

We demonstrate this approach with an example where we use the LTL model checking system SPIN, developed at Bell Labs, to mechanically verify the logical consistency of a high level design for one of Lucent's new products. It is estimated that interval reductions of 10 to 20 percent can be achieved by applying early fault detection techniques of this type.

1 Introduction

An industrial software design project often begins with a requirements capture and documentation phase. The requirements are largely implementation independent statements of various aspects of the system to be built, or modified. Many of these requirements are documented in graphical form: as message or data flows through an abstract representation of the main system components. Each major functional unit in such a representation is represented by a black box, and only its interactions with other functional

*Bell Labs, Lucent Technologies, 700 Mountain Avenue, Murray Hill, NJ 07974. Email: {gerard,margaret}@lucent.com

units is indicated with numbered or annotated arrows connecting the functional boxes. These pictures help both system engineers and developers in gaining an initial understanding of the new system, and its anticipated behavior.

Rigorous testing of a design can only be performed when the design itself has has been expressed in enough detail that it can be implemented, or minimally, simulated. Before that, one has to rely mostly on discipline, experience, and systematic peer review. The strongest error detection tools (i.e., testing tools) are not applicable until the design has been almost completely implemented. It is well understood, though, that the earlier a design error is found, the least costly and time consuming it will be to repair it.

It is generally agreed that interval reduction can be achieved if at least part of the design errors are intercepted in the earlier phases of a design. Capturing the errors early means that one can prevent inconsistencies from being introduced into the design requirements, and ultimately into the implementation. In an earlier study it was estimated that the interval reduction achieved with a comparable method was approximately 15 percent [1].

A growing collection of *Early Fault Detection Tools* [2, 3, 5, 6, 8, 4, 1] has been developed. Many of these tools appear to require little or no overhead to introduce into the design cycle. As an example of this, in the application studied here, it took two weeks of MTS effort, to build and verify a formal prototype of the design requirements for a new Lucent product. A number of logical inconsistencies in the design were identified mechanically with the help of an existing model checking system [3].

2 Prototyping Requirements

In civil engineering it is common-place that new designs, and design requirements, are verified thoroughly with the help of prototypes and computer models. It is unthinkable that one would start on the construction of a new bridge or skyscraper with just a set of paper instructions, illustrated with pictures, as a guideline for the builder - no matter how carefully groups of human designers may have scrutinized these instructions and illustrations in peer review sessions. Without a thorough verification of physical and computer models of the design, the structure cannot be built. In most cases, the design itself will be modified as a result of the verification process, thus avoiding costly reconstruction and adjustment later.

The use of early fault detection tools is similarly based on the construction of *prototypes* of the design or of the design requirements. The prototypes are sufficiently abstract that they can be constructed with minimal effort, and yet sufficiently formal that they can be verified thoroughly by

special purpose software, called *model-checkers*. The existing tools, such as the Bell Labs model checker SPIN that we will use in this study, are powerful enough that they can exhaustively check prototypes for substantial systems, [3] with billions of distinct, and potentially relevant, system configurations, and billions of distinct and potentially relevant system executions. Often a verification of this type will consume no more than a few seconds or minutes of CPU time on an average workstation or PC.

3 Analysis by Model Checking

The model checker SPIN can verify prototypes of complex distributed systems software. These distributed systems can contain varying numbers of asynchronously executing processes, that interact by accessing shared data, or by sending structured messages to one another via buffered channels or via synchronous rendezvous handshakes. In a physically distributed system, it is inevitable that different system components, or logical processes, will make assumptions about each other. For instance, after a client process submits a request to a server process, it may expect to receive either a failure or success indication within a certain amount of time. That server may, in turn, rely on other processes to handle subtasks, before it can make a final assessment of success or failure. Small misjudgments about possible sequences of events in such a system (e.g., of unpredictable combinations of error conditions) can have large consequences for overall service, and may, in the worst case, bring down the entire system. A model checker can either prove rigorously that a design can survive even the (humanly) most unimaginable error scenarios, or it can come up with clear examples of those scenarios that will not be handled correctly. The model checker can do this as soon as the requirements for the system are enunciated sufficiently well that a prototype can be built. This is typically within the first few weeks of even multi-year design efforts. The prototyping and verification phase often takes just a few days or weeks of effort by a single verification expert, yet it can intercept errors that could cost months of delay and substantial cost later in the project.

The prototype is constructed in a special verification language, called PROMELA. PROMELA is a C-like language with constructs that support automated verification. Given a model of a system written in PROMELA, the model checker SPIN can perform a simulation of the system's execution, and it can generate a C program that performs an exhaustive verification of the system state space. SPIN can check the specification for absence of deadlocks, unspecified receptions, and unexecutable code. The verifier can also be used to prove or disprove specific properties that the user can formulate about the system. We will give an example of such a property later in this paper. In its most general form, SPIN supports the automated ver-

ification of properties stated in linear temporal logic, a standard method for specifying properties of concurrent systems introduced in the late seventies [7].

4 An Example

For an application of these ideas, we studied Lucent's ACTIVIEW Provisioning offering for Wireless operators. ACTIVIEW is a workflow management product sold in a number of markets, such as

- Commercial local-exchange carriers,
- Wireless service providers,
- Domestic local operating companies, and
- International telecommunications operators.

The system plays a central role in automated provisioning applications by managing and tracking the progress of provisioning tasks, which are called *work items*. ACTIVIEW receives a Service Order from an external Service Order Entry System (SOE), converts inputs, and then automatically refers requests for provisioning to external *element management systems*. A given work item may require the processing of several such requests either in parallel or in series. ACTIVIEW analyzes the responses to these requests and at each step determines the next appropriate task to be performed.

A major component of the ACTIVIEW system is the Order Manager (OM). The OM module manages most of the data conversion required by the external components, executes the requests to external element management systems and reads and updates internal databases. Another important ACTIVIEW component is the Work Flow Manager that tracks work items and directs the activities of the OM module. The Work Flow Manager is itself directed by instructions from *task models*, that are designed by ACTIVIEW Systems Engineers (SE).

Task models have nodes and edges. Each node holds a task, which can be an external request to an element management system, a database operation, or an OM method. A typical fragment of a task model begins with an external request to an element management system, which is followed by a success task which updates the work item status or performs a database operation. It can also have one or more failure tasks which can retry some operations or defer the work item to a Manual Work List for human intervention. Task Models are created using a GUI based tool, called the Work Flow Editor.

The ACTIVIEW Wireless product supports automated provisioning of service for wireless handsets. A consumer may purchase a handset at a cellular service store, a general merchandiser, or even from a vending machine. Before service is initiated, the only available outgoing numbers are **911** and the telephone number for the Wireless provisioning system. A sticker on the handset instructs the buyer to dial this predefined number to start telephone service on the handset. When the call is placed, it is connects to an Automatic Call Distribution (ACD) system that routes the call to an available agent. The agent will ask the caller about desired features, such as three-way calling, caller ID, and voice mail, and determines the geographical service areas for incoming and outgoing calls. The customer's choices are recorded by the agent on Service Order Entry (SOE) screens. When the service order is complete it is submitted to ACTIVIEW.

The OM component creates a work item for each Service Order, it analyzes the data fields and selects the appropriate Task Model to perform the provisioning requested. Apart from the SOE system, three external element management systems are used to carry out the instructions issued by ACTIVIEW. Lucent's Over The Air Function (OTAF) System is used to program the handset remotely, using radio frequency transmissions. The Home Location Record (HLR) system stores customer feature subscription information that is used by Mobile Switching Centers during the actual call handling. The Voice Messaging System (VMS) has a co-resident subscription database and provides for message storage for a customer when the wireless handset is turned off or busy.

Figure 1 depicts these systems and their relationship to the ACTIVIEW product. Depending on a given work item, ACTIVIEW may request provisioning actions from all three external systems or from just one or two of these. For instance, if the work item is for new service and the customer requests a voice mail feature, OTAF will program the handset while the HLR record and the VMS records are being updated at the request of ACTIVIEW. If the same customer calls back to terminate a three-way calling feature requested earlier, only the HLR system will receive the request for modification from ACTIVIEW. ACTIVIEW knows which Task Model to invoke to accomplish each specific request.

Figure 2 shows a simplified scenario of a typical use of the ACTIVIEW Wireless product.

The correct operation of the wireless service requires a coordinated provisioning of several disparate systems. The design of the ACTIVIEW system is intended to provide this coordination. A complicating factor in this process is that ACTIVIEW is fundamentally a multi-system product, combining both Lucent and non-Lucent systems. Knowledge of especially the non-Lucent components is limited to interface functionality only, which, in some cases, is only partially documented. The construction and formal verification of a model of this design, therefore, can also serve to reassure the designers

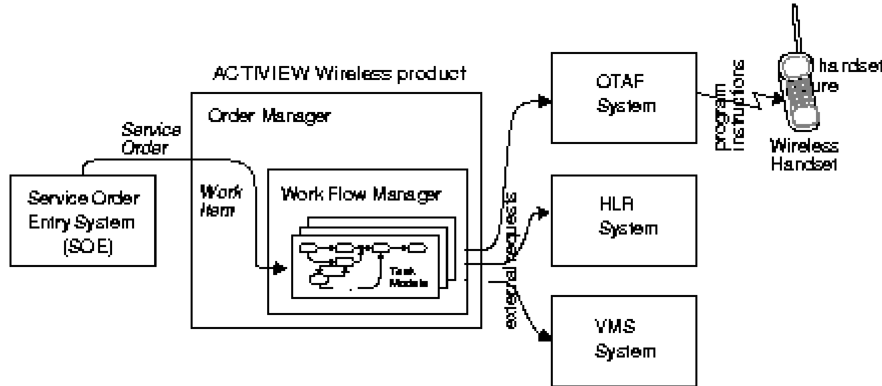

FIGURE 1. ACTIVIEW Wireless Product and External Components.

that the main assumptions made in the design process were justified.

5 The Spin Model

The full description of the requirements for the OM system are documented in approximately 250 pages of text (or roughly 15,000 lines), created over a period of five months. The second author of this paper, was one of the authors for the original requirements for the ACTIVIEW product. The same author constructed the design prototype of the task models for the ACTIVIEW Wireless product in an additional two weeks, which was then reviewed by the first author. The two week period for the construction of the design prototype included a brief learning phase on the details of the SPIN model checker. When more targeted high level design capture tools are used on a project, e.g., such as the Lucent UBET tool [4], the time required to produce the intial verification models could be reduced further, and perhaps even avoided completely by taking advantage of the builtin model extraction capabilities of these tools.

The prototype contains abstract models of the SOE, OTAF, HLR, and VMS components, based on the ACTIVIEW interface specifications. The model was built in PROMELA, the specification language (or *application modeling language*, in the terminology of domain engineering), of SPIN. The final model consists of 438 lines of PROMELA text. The GUI for SPIN, showing part of this prototype, is illustrated in Figure 3.

After the prototype was constructed, the authors spent one afternoon to debug the model; using SPIN for random and interactive simulations. Some unnecessary assumptions were removed, the model was refined, and logical

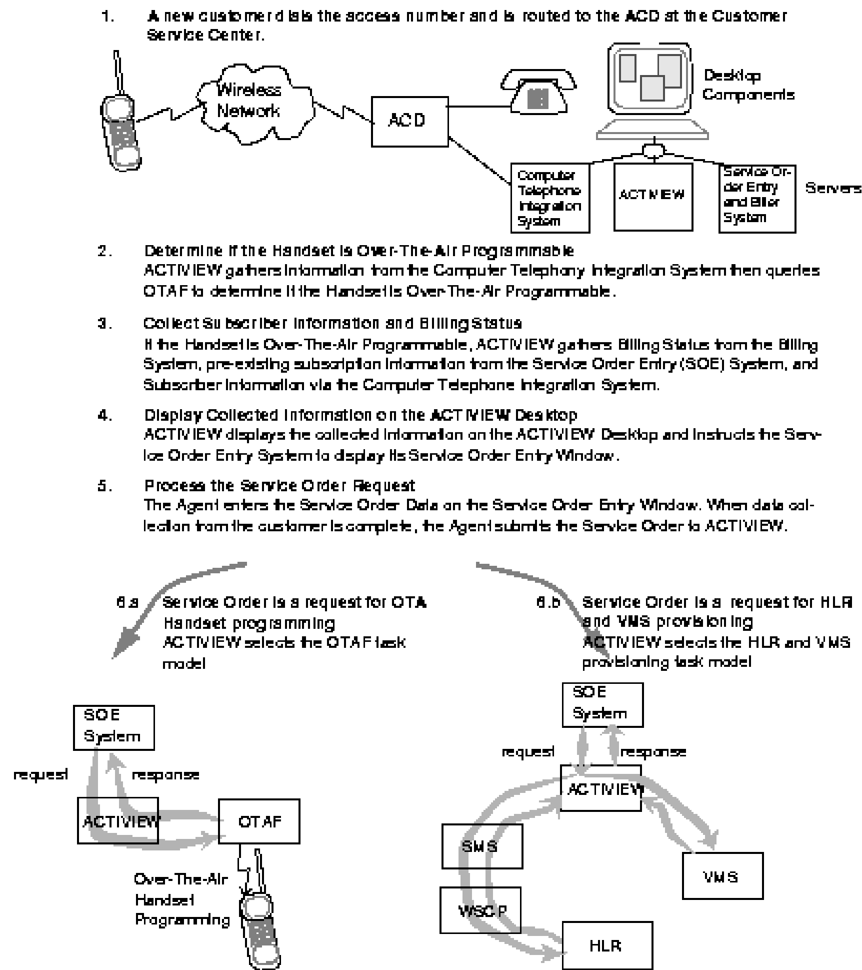

FIGURE 2. ACTIVIEW Provisioning Scenario.

correctness requirements were added. Each simulation run was displayed by SPIN in graphical form as an annotated message sequence chart, as illustrated in Figure 4.

With the help of simulations, unwarranted abstractions that were made to construct the prototype itself from the written requirements were identified and removed. Once the simulation of the prototype conformed, as far as could be observed, to the intended behavior of the system, as far as documented, the actual model checking process could begin. In this reviewing process the initial prototype of 1,100 lines of PROMELA was reduced to a model of 438 lines.

FIGURE 3. Main Control Panel for the SPIN Model Checker.

The model describes the behavior of 17 logical processes within the system, communicating through 20 buffered message channels, each with a capacity to store up to four messages (each message with multiple data fields). Each logical process has between 20 and 50 distinct control states, and has access to between 5 and 10 local variables to store temporary results. The control states can be made visible in automaton form, as illustrated in Figure 5, but for most applications this detail is not needed to perform the verifications.

6 Requirements Analysis by Model Checking

To perform verification, the SPIN system converts the prototype specification from an automata model into an optimized program in standard C. This program is then compiled and executed to perform the verification. The result consists either of a terse statement that the correctness properties are valid for the system as specified, or (more typically) the generation of one or more detailed counter-examples that demonstrate that there are logical inconsistencies in the model. The counter-examples, or error-trails, can be inspected with the help of SPIN's simulation capability, either step by step, or by using the trail for a *guided* simulation run, supported by message sequence charts, data displays, etc.

An exhaustive verification run for the system as specified analyzes up to 7 million distinct system configurations. The longest execution scenario

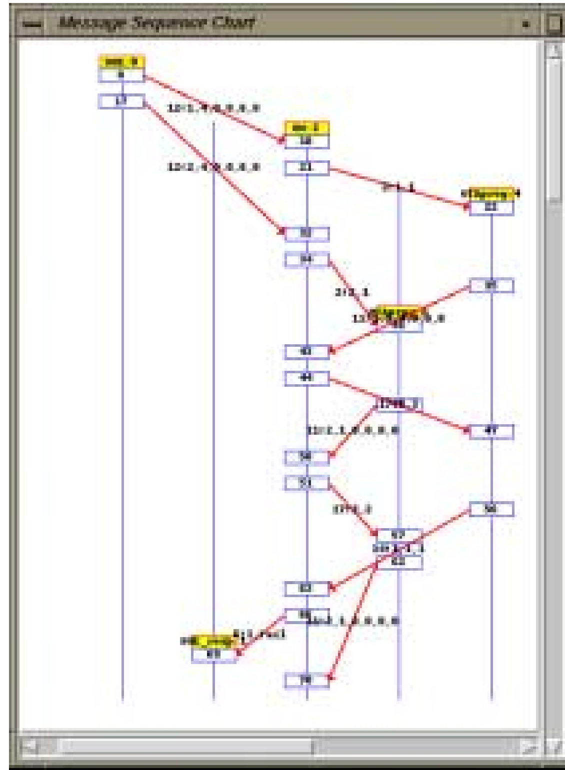

FIGURE 4. SPIN Display of Simulation Run.

encountered in this verification is 6,793 steps. This verification run is completed in approximately 30 minutes of CPU time on an average workstation.

It is not necessary, however, to let the verification run proceed until completion. SPIN's on-the-fly model checking algorithm by default stops the verification attempt as soon as it can prove that at least one of the correctness properties can be violated. In this application, this occurs after just a few seconds of runtime, and the inspection of only a few hundred carefully chosen system configurations.

The property we set out to prove in this verification attempt was:

- *"The status of the wireless handset on any particular feature requested by the customer always corresponds to the last provisioning request that was submitted by the customer for that feature."*

The critical term here is *submitted*. As the model checker demonstrates within seconds, the order of submission of provisioning requests does not

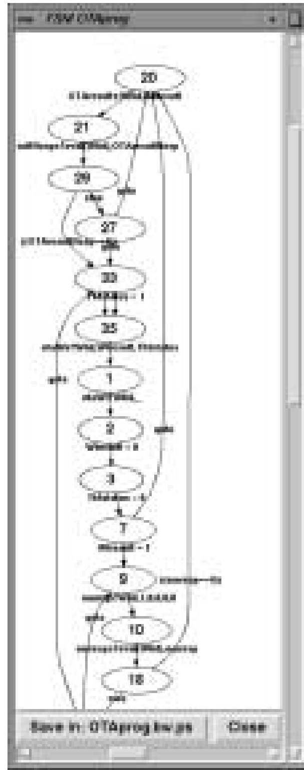

FIGURE 5. Automaton Structure for OTAprog Process Computed by Spin.

necessarily correspond to their order of processing. The counter-example is displayed by the model checker as the message-flow diagram shown in Figure 6.

Consider a customer who requests three-way calling, is informed of the cost and other details by the operator, during the entry phase of the provisioning request, and decides after a short time that the cost is too high and submits a new request to cancel the feature. There is no guarantee in the system as designed that the provisioning for the second request is not completed before the first request. The result of the first processed request would then be void, since it involves the canceling of a feature that was not available to the customer. The result of the request processed second would be a successful installation of the feature, despite the customer's explicit cancellation. In Actiview it is assumed that the status of the handset always corresponds to the last provisioning request. It can be difficult to secure that the design that is selected can indeed never violate this assumption. The designers can reason about the design to establish that for

all reasonable cases that can be imagined, this assumption is valid. With and automated verifier, though, the designer can prove with certainty, that a design guarantees the validity of the assumption for *all* possible executions, both the imaginable and the unimaginable ones.

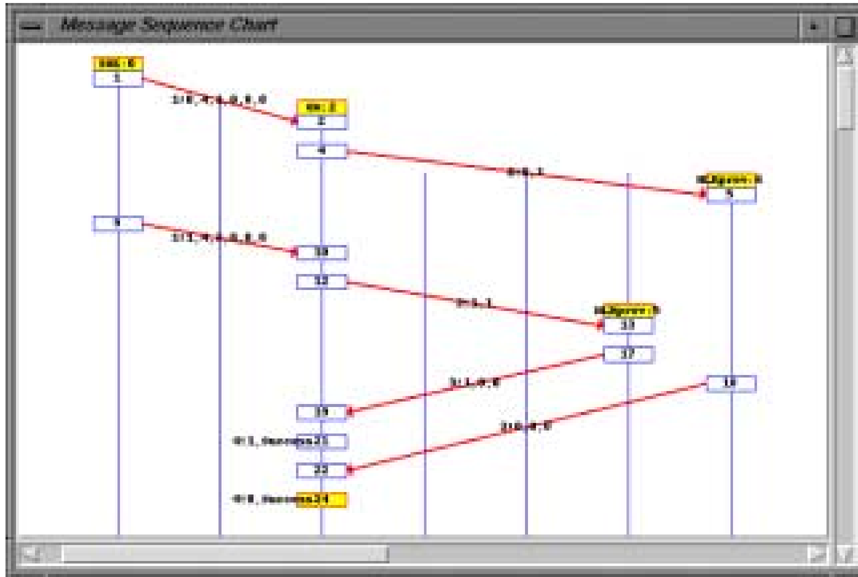

FIGURE 6. Scenario Generated by SPIN Showing Violation of Requirement

The discovery that the model fails to reliably provide a stated property can initiate a discussion with the customer about the relative importance of the fault. In this case, a minor redesign of the Order Manager suffices to secure the required coordination between related workitems, an guarantees reliable service under all circumstances.

7 Conclusions

The verification of design requirements in the early phases of, especially the larger, software design projects requires a relatively small investment of skill and effort. There is a potentially large return on this investment in reduced interval time and project cost, by avoiding unpleasant discoveries of design faults late in the design cycle. Early fault detection tools of the type we have discussed have not yet been exploited fully in industrial design projects. The skills that are required to use these tools well are limited to basic skills in model construction, and abstraction, i.e., skills that may also

prove valuable in other aspects of systems design.

Model checking tools have been studied for almost two decades. The SPIN system, for instance, is based on a development at Bell Labs that dates back to 1980. The increase in the speed of computers and in the amount of memory they can access, by several orders of magnitude each, and the graphical capabilities that most desktop computers today support, have all contributed to turning our tentative first efforts towards developing software verification tools into the availability of powerful new desktop tools that can be used with almost push-button ease by engineers in the development of robust software products.

Acknowledgements

The following are gratefully acknowledged for their technical input on ACTIVIEW: Myrna Papier, Elliot Pludwinski and Shel Rockowitz.

8 REFERENCES

[1] A.R. Flora-Holmquist, E. Morton, J.D. O'Grady, and M.G. Staskauskas, 'The Virtual Finite-StateMachine Design and Implementation Paradigm', *Bell Labs Technical Journal*, Vol. 2, No. 1, 1997, pp 96-113.

[2] G.J. Holzmann, 'Early Fault Detection Tools,' *Software Concepts and Tools*, Vol. 17, No. 2, 1996, pp. 63-69. Also in: Lecture Notes in Computer Science, Vol. 1055, pp. 1-13.

[3] G.J. Holzmann, 'The Model Checker Spin,' *IEEE Transactions on Software Engineering*, Vol. 23, No. 5, May 1997, pp. 279-295.

[4] G.J. Holzmann, D.A. Peled, and M.H. Redberg, 'Design Tools for Requirements Engineering,' *Bell Labs Technical Journal*, Vol. 2, No. 1, 1997, pp. 86-95.

[5] R.P. Kurshan, *Computer-Aided Verification*, Princeton University Press, 1994.

[6] K.L. McMillan, *Symbolic Model Checking*, Kluwer, 1992.

[7] A. Pnueli, 'The temporal logic of programs,' *Proc. 18th IEEE Symposium on Foundations of Computer Science*, 1977, pp. 46-77.

[8] SPIN Webpage, containing tool sources, test cases, and documentation. `http://netlib.bell-labs.com/netlib/spin/`

Acronyms Used

- *ACD - Automatic Call Distribution*
- *AML - Application Modeling Language*
- *GUI - Graphical User Interface*
- *HLR - Home Location Record*
- *LTL - Linear Temporal Logic*
- *MSC - Mobile Switching Center*
- *OM - Order Manager*
- *OTAF - Over The Air Function*
- PROMELA - *Process Meta Language, the AML of* SPIN
- *SOE - Service Order Entry System*
- *SE - System Engineers*
- *SOE - Service Order Entry*
- SPIN - *The LTL Model Checker used*
- *TM - Task Model*
- *VMS - Voice Messaging System*
- *WFE - Work Flow Editor*
- *WFM - Work Flow Manager*