1991 NASA/ASEE SUMMER FACULTY FELLOWSHIP PROGRAM

JOHN F. KENNEDY SPACE CENTER
UNIVERSITY OF CENTRAL FLORIDA

KATE'S MODEL VERIFICATION TOOLS

PREPARED BY:	Steve Morgan, Ph.D., P.E.
ACADEMIC RANK:	Associate Professor
UNIVERSITY AND DEPARTMENT:	Baylor University Department of Engineering and Computer Science
NASA/KSC	
DIVISION:	Engineering Development Laboratories
BRANCH:	Artificial Intelligence Laboratory
NASA COLLEAGUE:	Carrie Parrish, Ph.D.
DATE:	August 9, 1991
CONTRACT NUMBER:	University of Central Florida NASA-NGT-60002 Supplement: 6

Acknowledgements

Thank you, Carrie Parrish, for helping define and execute the Model Verification Tools project this summer. Thanks especially to four of KATE's modelers, Charlie, Goodrich, Bob Merchant, Steve Beltz, and Scott Budzowski, for telling how models evolve, what tools they use, and what tools they need. Thanks to NASA's Mark Beymer and Dennis Armstrong, and to UCF's Ray Hosler for making the NASA/ASEE Summer Faculty Fellowship Program possible.

Abstract

Kennedy Space Center's Knowledge-based Autonomous Test Engineer (KATE) is capable of monitoring electromechanical systems, diagnosing their errors, and even repairing them when they crash. A survey of KATE's developer/modelers revealed that they were already using a sophisticated set of productivity enhancing tools. They did request five more, however, and those make up the body of this report. Inside is the tested code of their new: 1) transfer function curve fitter and 2) Fortran-Lisp translator. 3) To aid in syntax checking their modeled device frames, three existing structural consistency checkers are also documented here. 4) An automated procedure for calibrating knowledge base admittances is designed here to protect KATE's hardware mockups from inadvertent hand valve twiddling. 5) And three alternatives are described for the "pseudo object," a programming patch that currently apprises KATE's modeled devices of their operational environments.

Summary

When KATE's modelers revealed their need of five new productivity aids this summer, I set about the task of creating them. They now have a new Lisp program that fits straight lines or exponential curves to random time functions, and another that translates transfer functions written in Fortran into Lisp's stilted prefix format. When I started to streamline their favorite syntax checker, I discovered that some modelers already had what others needed, so I ended up documenting three existing checkers instead. Guided by a modeler whose hardware mockup keeps changing, I designed a control procedure by which KATE automatically recalibrates admittances in his knowledge base. An essential programming patch that the modelers call a "pseudo object" hasn't fit neatly into either KATE's knowledge bases or her shell. Three alternatives are offered here. Solving modeled fluid dynamics equations off-line could provide KATE the necessary flow time functions when she needs them. If these complicated equations must be reconfigured and solved on-line, a signal processing analog computer model offers a fast way of doing it. Modeling KATE's devices as Thevenin's equivalent two-port networks could enable each object to reveal its remote environment (much as a transformer reflects its load resistance to its input) without the need of extra pseudo objects or equation solving. KATE is about to get a much needed pass through the Software Development Life Cycle as part of her forthcoming translation to the C language.

Table of Contents

LIST OF ILLUSTRATIONS

LIST OF TABLES

I
INTRODUCTION

1.1 KATE

Kennedy Space Center's Knowledge-based Autonomous Test Engineer (KATE) mimics human engineers as they test and repair electromechanical systems. Working in KSC's Artificial Intelligence Laboratory, KATE's modelers painstakingly create software simulations of the hardware that KATE monitors. (KATE's models typically take on an appearance like that shown in Figure 1-1.) KATE later delivers the same inputs to this software model that the hardware sees, and she compares their outputs. Showing complete confidence in her software, she blames the hardware for any deviations in modeled and actual outputs. Her diagnoser then experimentally fails selected software "devices" in an effort to duplicate the deviant hardware outputs. Thus KATE is able to isolate the one failed device that is responsible for the errors. She may then advise her user to bring up a redundant device, or she may make the repair herself, if the user has seen fit to give her that much control ahead of time. Lately, users have not seen fit to give her that much control in her only production application, the shuttle's liquid oxygen tanking operation. In fact, firing room personnel are unwilling to hear KATE's advice when real problems arise. NASA is conservative. Systems must log hundreds of hours of trouble-free service or show some sort of quality assurance pedigree before being placed in a critical position. KATE is not there yet.

1.2 KATE'S MODEL VERIFICATION TOOLS

Like all Knowledge Base Engineers, KATE's model builders have difficulty mixing human-like reasoning with computer speed and thoroughness in their software product. My job this summer has been make those modelers more productive, so that KATE's models may go forth and multiply throughout KSC. I became intimately acquainted with KATE by observing her modelers and streamlining their tools. I coded two new tools for them from scratch in Symbolics ZMACS Lisp. Also intending to enhance the code of an existing syntax checker, I ended up documenting three modelers' favorite checkers for use by all. I also designed an automated calibration procedure for one of KATE's knowledge bases, and I offered three design alternatives to KATE's philosophically awkward "pseudo object." Though not immediately executable, these designs helped train me for next summer's project, and they may well provide proposal content for some productivity enhancing projects this fall.

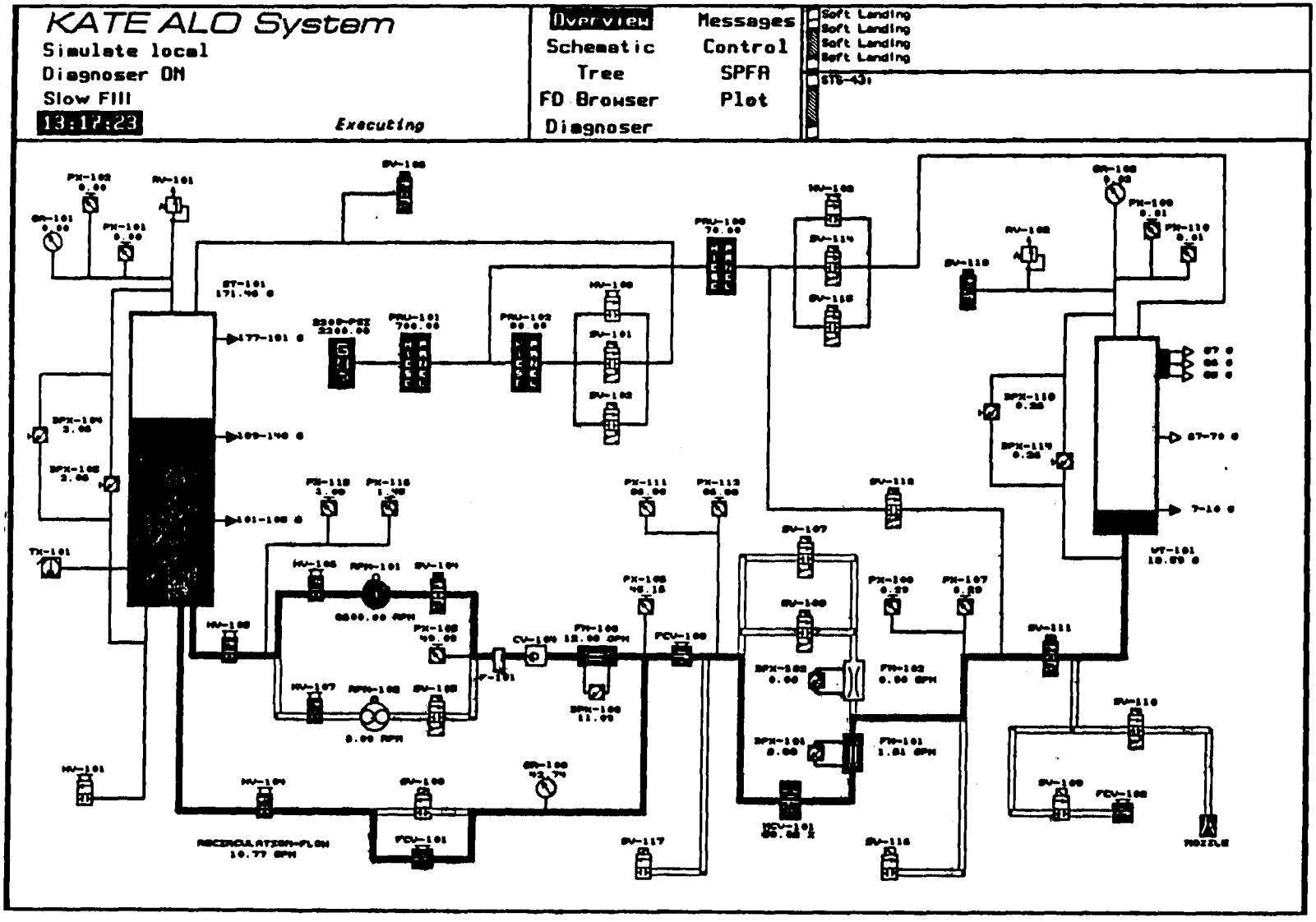


Figure 1-1. KATE's Overview Screen (ALO Model).

II
BUILDING KATE'S MODELS

2.1 KATE'S CURRENT APPLICATIONS

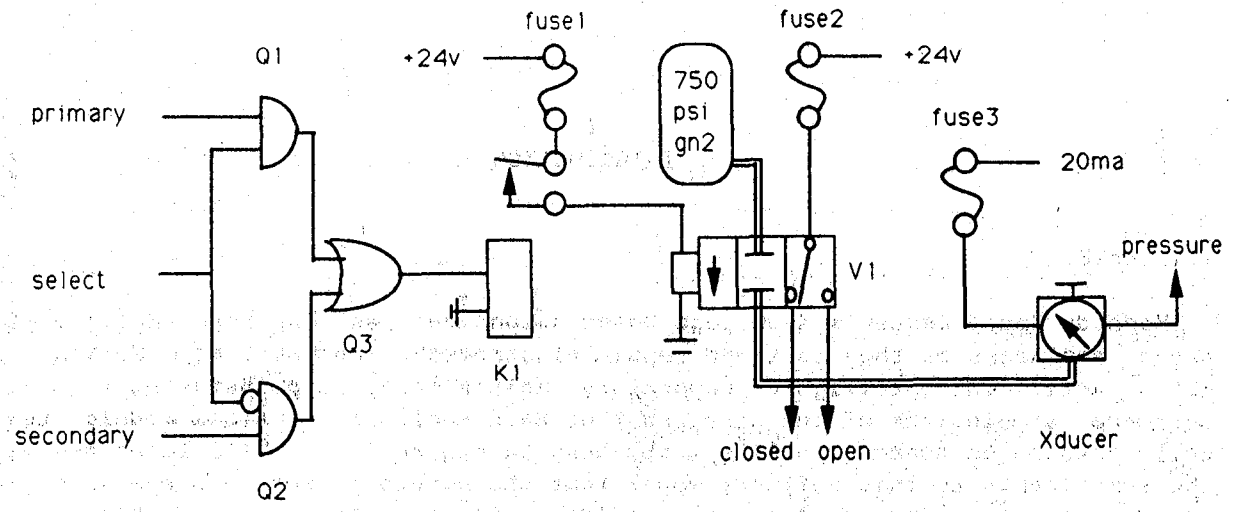
KATE's most visible application today is LOX, her shuttle liquid oxygen tanking advisor. (KATE does not actually advise firing room personnel, but she is tested as if she were advising them on every shuttle launch.) With 1000 to 1500 device frames in her knowledge base and 50- to 75-thousand lines of Lisp code in her shell, KATE's LOX model is as complicated as her models will likely ever get. Figure 1-1 shows LOX (actually this is a schematic overview of another very similar KATE model) to be essentially a cryogenic fluid dynamics model, with a large number of sensors and actuators. Because the oxygen tanking equipment is critical to any launch, failed devices in it can be replaced quickly by bringing up standby pumps, valves, and sensors along redundant pathways.

A variety of KATE's earlier feasibility demonstrations can be found around KSC. An air purge system with redundant air compressors and power supplies introduced the new-born KATE to KSC about eight years ago. A mockup of the Space Station's Environmental Control System (fully equipped with heaters, blowers, and temperature sensors) is attached to KATE's ECS model. A Launch Processing System mockup (called "LPS" and "The Little Red Wagon," because it is mounted on a heavy red cart) is attached to a TI Lisp machine version of KATE. KATE has been translated into ADA, Common Lisp for the IBM PC, a generic version for quickly prototyping new models, and it will soon be translated into C for the IBM PC (more on KATE's future later).

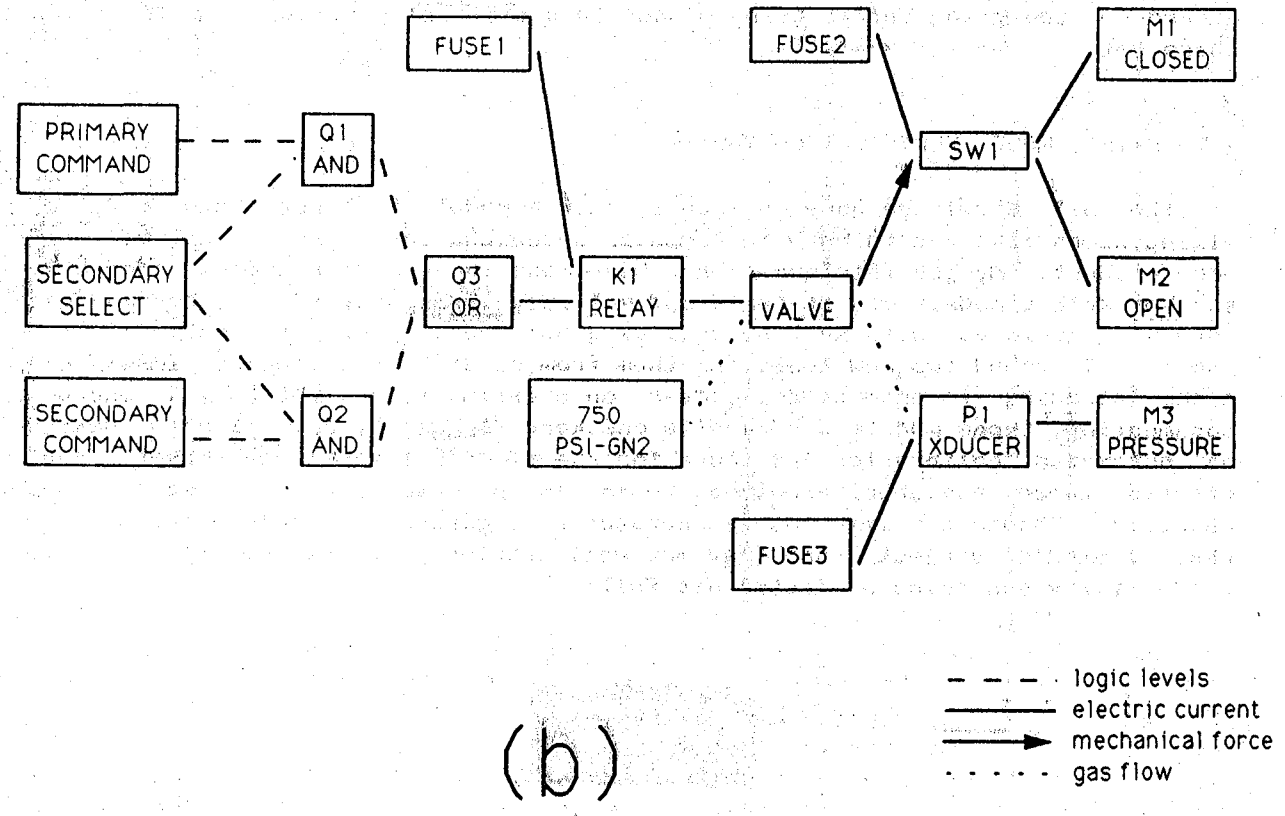
KATE's Autonomous Launch Operations (ALO) application suggests her intended future. Created to supervise unmanned Air Force launches, the ALO model features goal-directed control and unsupervised repair. A hardware mockup at KSC's Launch Equipment Test Facility has demonstrated KATE's ability to complete all 12 LOX tanking procedure steps typically pursued by shuttle firing room personnel, even when devices fail. Water in the hardware mockup will be replaced with liquid nitrogen in a forthcoming demonstration. (Liquid oxygen is not used for obvious safety reasons.) The ALO-H20 model now includes some 472 frames, and the number grows with each requirement for greater modeling precision.

2.2 THE MODEL BUILDING PROCESS

Developing a new application consists of building a software model of the hardware that KATE is to monitor, and then improving it until it works. The model builder usually gets started by studying a ten-pound stack of printed schematic diagrams (see Figure 2-1). He draws fault trees of the devices he sees there to appreciate the impact of every device failure upon the whole model. Finally he types a Lisp frame into the Symbolics ZMACS editor for every device that can fail, every input, and every measurement. The frames look like database records, in which named slots describe every aspect of the part and its connections to other parts [1]. Coding a device transfer function can be as



(a)



(b)

Figure 2-1. KATE's model building process.

simple as transcribing a pressure/flow curve off a printed pump specification or very complicated indeed. Its original electronic designer may have drawn a hardware controller as a tangled feedback network of filters and amplifiers. Ten years later, the KATE modeler is compelled to Lisp-code the simplest possible transfer function for the controller, considered as a single line-replaceable unit.

Before KATE executes a model the first time, the modeler tries to remove frame errors that are obvious by inspection (white box testing) but would be very hard to isolate during execution (black box testing). Automated structural consistency checkers find or repair simple frame syntax errors (e.g., frame A outputs to frame B, but B does not input from A). They may also identify frames that have not been coded yet. Subtle semantic errors (e.g., relays do not accept input from water pipes) generally are obvious in a visual inspection of KATE-generated schematics.

When the frames all look pretty, verification testing begins. The first time KATE runs a new model, actual (hardware) measurements often deviate from modeled values, and KATE blames innocent hardware devices for errors in the model. KATE's diagnoser thus functions as a system-level model verification tester. Taking a closer look at the offending measurement, the modeler often finds a variety of transfer function errors. He may have assumed the wrong time constant in an exponential decay. Perhaps he failed to consider the affect of ullage (the air above the fluid) pressure that impedes flow into a tank. In his zeal to model very accurately, he may have modeled quantizing error that appeared on a prior launch data broadcast. (It has happened!) In the latter stages of knowledge base development, subtle timing errors sometimes occupy the modeler. For example, KATE must guess when firing room personnel transition from stage to stage in the LOX filling procedure (e.g., nobody tells KATE when fast fill ceases and slow fill begins), and her ways of dealing with these ambiguities can be obscure.

2.3 MODEL BUILDERS' PROBLEMS

KATE's developer/modelers face problems that range from clerical to very sophisticated. (See modelers' testimonies in Appendix A.) Typing frames into the LISP machine ZMACS editor is tedious, compared with using an interactive-graphics Computer-Aided Design terminal. It is difficult to write the equations of lines that approximate random time functions. It is just as difficult to recode Fortran transfer functions in Lisp without a few errors, even for bilingual programmers. Thousand-frame knowledge bases often start out full of syntax errors. The structural consistency checkers used to find these errors never find them all. Modelers that use hardware mockups need to recalibrate admittances in their knowledge bases each time valves on the mockup change. When computing input pressure of a device, a modeler sometimes needs more information than the device model can provide (e.g., upstream driving pressure and downstream back pressure in its external environment). That may lead him to create a "pseudo object," having no structural relationship with modeled devices and no relationship with generic objects in KATE's shell either (because it is application-specific). KATE's modelers have invented "soft landing," which

wiggles all modeled measurements to ease the pain of errors that the modeler considers tolerable, and "external influence," which accounts for measurement trends arising outside the declared model domain. With each shuttle launch KATE shows her developers a few new errors, and her design or code changes a little to accommodate an unforeseen operational condition.

III
KATE'S MODEL VERIFICATION TOOLS

3.1 MODEL VERIFICATION PERSPECTIVE

Someday, KATE's very graphical Knowledge Base Editor will make creating frames easy even for the novice end user. Like a CAD terminal, the editor provides a library of device icons that can be picked and placed, moved and interconnected without ever touching the keyboard. Most knowledge about a device will be supplied by the editor. If desired, the user can modify default transfer functions, time delays, units, etc. at the keyboard via a database-like fill-in-the-blanks editor interface. KATE's Model Verification Tools are that part of her Knowledge Base Editor that helps find errors in completed knowledge bases.

Today, KATE's Model Verification Tools already are used to find errors in completed knowledge base frames that are typed in manually using Symbolics' ZMACS editor. (See Appendix A for the details.) Structural consistency checkers find simple frame syntax errors. A family of interactive plotters clarify troublesome deviations in actual and modeled time functions. KATE's overview, schematic, and fault tree graphic displays clarify frame mismatch errors.

Cleverly conceived as KATE's Model Verification Tools are, her modelers agree that a few good tools are missing. More insightful frame syntax checkers are always needed. Coding transfer functions would be much easier if a curve fitter could recite the equations of straight line and exponential plotted data. And careless errors could be prevented if a program could convert transfer functions from familiar algebraic (infix) notation into Lisp's stilted prefix notation. Tedious calibration of KATE's ALO-H20 knowledge base admittances (which happens frequently) should be automated. Clever as KATE's pseudo objects may be, they do not fit well into either her application-specific knowledge bases nor her generic shell. Solutions are offered for all of these model verification tool problems below. They range from executable code to proposal-ready design to white papers on a better pseudo object.

3.2 LEAST-SQUARES CURVE FITTER

Modelers frequently need a transfer function equation that mimics a plotted measurement time function. Though random variables, the time functions generally are approximately straight lines or exponentials that rise or fall toward some asymptote. Symbolics' Genera 8.1 operating system fits least-squares straight lines to plotted random variables, but it does not reveal their equations. KATE's modelers need a least-squares curve fitter with an optional capability of taking the log of its time coordinates.

Such a program appears in Appendix B [2]. Given two lists of x and y coordinates, it returns three parameters: the slope, the y-axis intercept, and the sample correlation coefficient of the given points. First the program computes means, variances, and covariances of all of the x and y coordinates:

$$Xmean = (x1 + \ldots + xn) / n,$$

$$Xvar = (x1^2 + \ldots + xn^2) / n,$$

$$XYcovar = (x1 * y1 + \ldots + xn * yn) / n.$$

The equation of the least-squares line, $y = a + b\,x$, is then defined by the slope,

$$b = \frac{XYcovar - Xmean * Ymean}{Xvar - Xmean^2},$$

the y-axis intercept,

$$a = Ymean - b * Xmean.$$

The sample correlation coefficient,

$$r = \frac{XYcovar - Xmean * Ymean}{SQRT(Xvar - Xmean^2) * SQRT(Yvar - Ymean^2)},$$

approaches 100% when the straight line fits the data well or falls below 80% when it fits poorly.

No design diagram is attached to this program, which is obviously crafted to be more readable than efficient. Abelson and Sussman [3] contend that Lisp is so powerful that it may be used as a design language to describe Lisp code. Especially in cases like this one, where control flow is obvious, carefully crafted Lisp can be almost English-like. To the simple test cases for this curve fitter, its modeler-customer (Scott Budzowski) added a graphical output. He composed this enhancement at the keyboard, testing each module as he completed it, in about 10 minutes -- it was a lot of fun to watch.

3.3 FORTRAN-TO-LISP TRANSFER FUNCTION TRANSLATOR

Most good Lispers can translate a device transfer function from ordinary algebraic (infix) notation,

```
p_out = pump_coef * rpm ^ 2 / max_rpms_squared,
```

into Lisp's prefix notation,

```
(setq p_out (* pump_coef (/ (square rpm) max_rpms_squared))),
```

with very few careless errors. But KATE's targeted end users are not Lispers. They are ordinary Electrical Engineering graduates whose first computer language is Fortran. Both KATE's AI Lab developers and her end users can benefit from a

tool that translates functions from infix (algebraic, Fortran) to prefix (Lisp) notation.

Such a translator appears in Appendix C. The simple first version insists that binary operators be in the middle of three-item lists. (It also accepts unary operators in two-item lists.) This list length constraint frees it from having to handle operator precedence (e.g., multiplying before adding in w + x * y), but it forces the user to add a lot of parentheses to a valid Fortran function. The modeler that requested this program thought this burden too great.

The more complicated second translator in Appendix C handles operator precedence very nicely, allowing the user to string any number of operations together in each parenthetical expression. Taken from Winston and Horn's LISP textbook [4], this more complicated version does not handle unary operators gracefully. They must be made to look like binary operators by adding a nil to their lists before translation, and removing the nil after translation.

A Warnier-Orr Data Structured System Design Diagram precedes the second program's Lisp code as documentation. The design shows in plain English that the translator is fiercely recursive; in fact, it hard to follow without a design. Designing code makes it readable to its coder and other code auditors, such as module test engineers, and thus makes code more error free and reliable. The several test cases attached to the translator complete its documentation package, further enhancing its reliability by demonstrating its capabilities and limitations. Sound configuration management requires that design and test data be attached to every coded module in a production software system.

3.4 A BETTER STRUCTURAL CONSISTENCY CHECKER

Perhaps the reader can tell from the survey in Appendix A that a structural consistency (i.e., frame syntax) checker means several different things to several different modelers. Bob prefers to repair his errors one category at a time, so he needs a syntax checker that can be modified to check one frame slot at a time. Charlie's syntax checker keeps him apprised of what frames remain to be coded in his evolving knowledge base, so he doesn't care which slot it checks. Steve wants to be told about as many kinds of syntax errors as possible in his complete and almost perfect knowledge base, with a minimum of user interaction. Mark wants his Mondo syntax checker to quietly fix the errors it finds.

Initially, I understood that KATE's consistency checking file contained just two programs. Mondo-Consistency-Checker repairs five kinds of syntax errors (see Table 3-1), and wimpy but modifiable Check-KB points out all syntax errors of one kind. Most modelers wanted the greater power of Mondo, but without its repair feature, which tended to add unpredictable errors to their already fragile knowledge bases. Intent upon defanging Mondo to satisfy the needs of all, I first drew a module hierarchy chart to document its performance (see Figure 3-1). As I drew, I discovered a third program, Check-All-Frames, which is much more powerful (see Table 3-1 again) than the other two but dusty from disuse. I ended up documenting all three of KATE's unmodified syntax checkers to better serve the diverse needs of KATE's several modelers.

TABLE 3-1. What KATE's Structural Consistency Checkers Test.

CHECK-KB tests only instance-level frames for input/output reciprocity (now):

instance-level frame A	<===>	instance-level frame B
inputs slot: B	<===>	outputs slot: A

MONDO-CONSISTENCY-CHECKER repairs 4 kinds of reciprocal frame references:

top-level frame D	<===>	mid-level frame C
kinds slot: C	<===>	ako slot: D
mid-level frame B	<===>	instance-level frame B
instances slot: A	<===>	aio slot: B
mid-level frame F	<===>	instance-level frame E
parts slot: E	<===>	apo slot: F
instance-level frame G	<===>	instance-level frame H
draw-connects slot: H	<===>	draw-connects slot: G

CHECK-ALL-FRAMES tests frames for the following (optionally interactively):

Top-level Frames:
1. Does a parts, apo, instances, or aio slot accidentally appear here?

Mid-level Frames:
2. Wrong type of input or icon name (e.g., numeric)?
3. Is outputs, output-functions, or units slot missing?
4. Is any of these missing its value?
5. Is an output referred to in an output-functions, units, tolerance, or delay slot not defined in this frame's outputs slot?

Instance Frames:
6. Wrong type of input or output name?
7. Icon plotting coordinates missing or not in (system x y) form?

All Frames:
8. Is it not a frame?
9. No slots?
10. Is a slot empty?
11. Is nomenclature missing?
12. Slots of an unknown type?

Does that frame referred to in a slot of the type below not refer back?

13.	top-level kinds slot	<===>	mid-level ako slot
14.	mid-level instances slot	<===>	instance-level aio slot
15.	mid/instance-level input slot	<===>	mid/instance-level output slot
16.	instance-level parts slot	<===>	instance-level apo slot
17.	instance-level draw-connects	<===>	instance-level draw-connects

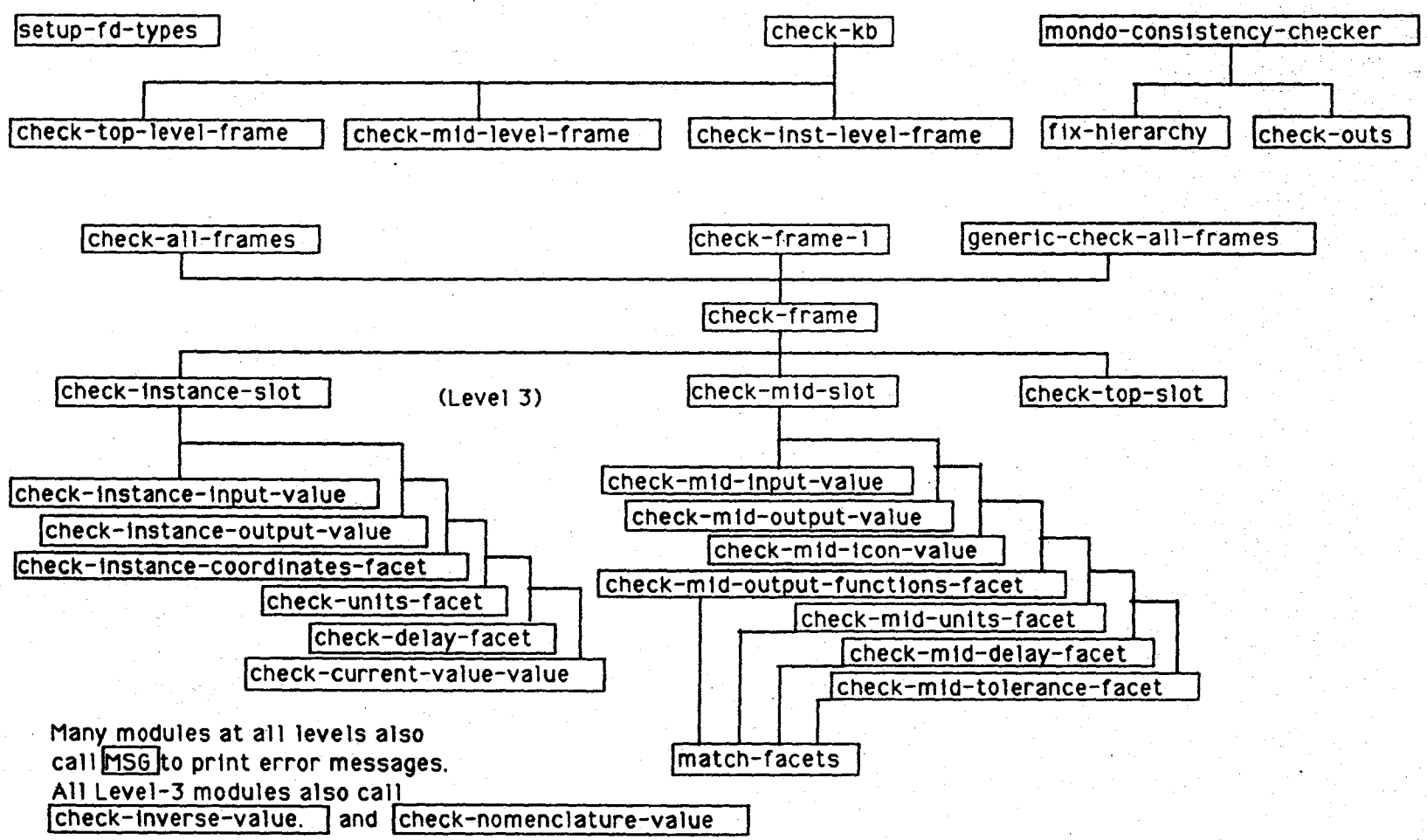


Figure 3-1. Consistency Checking file hierarchy chart.

Mondo's typical printout appears in Figure 3-2a. It counts five kinds of errors (printing the tallies on the terminal screen), and repairs them, but does not point them out to the user. As such, it is less of a tester more of a compiler, accepting a mere skeleton of the intended frames and producing a working knowledge base. Changes Mondo makes in the knowledge base are not permanent, and the original knowledge base can be reloaded over the repaired one if desired.

Check-KB's typical printout appears in Figure 3-2b. It points out syntax errors so that the user can correct them himself (and learn not to make the same mistakes next time). Changing just a line or two of code in Check-KB redirects its attention to any slot of interest in the frames being tested. Check-KB's printout is often lengthy, so it is written to file KB-LISTING instead of the user's screen. Its input nominally comes from file ALO-KB>ALO-H20. Both file names can be readily changed, of course, by modifying its Lisp code.

Written in 1988 for an earlier version of KATE, Check-All-Frames apparently doesn't work at all now. But its ability to point out 32 different kinds of frame syntax errors piques many modelers' interests. (See Table 3-1 and Parrish' dissertation [5] for more detailed documentation.) Upgrading Check-All-Frames to work with today's KATE would really pay premiums in modelers' productivity, and it should be considered in any future KATE contract negotiations. When it is ready, the test plan of Appendix D will verify its performance. KATE's configuration managers should keep a closer eye on such powerful modeling tools in the future and prevent their obsolescence.

3.5 AUTOMATED ADMITTANCE MEASUREMENT

Workmen frequently upset KATE's ALO-H20 model by adjusting hand valves in its hardware mockup. When actual admittances disagree with modeled admittances in her knowledge base, KATE's simulated pressures and flows differ from those in the real world, and she blames a hardware device failure. Minor admittance differences can lead KATE to make subtle reasoning errors that cost her modeler significant debugging effort. After discovering the source of these obscure errors, he still has to manual remeasure valve admittances and edit them into the knowledge base. KATE's ALO-H20 modeler needs a way of automatically calibrating these frame admittances before every demonstration.

KATE's control facility can set up the solenoid valves in the ALO-H20 hardware mockup in configurations that enable measuring key flows and pressures. Then KATE can use these pressures and flows to calculate all admittances in the knowledge base automatically.

The test procedure in Appendix E enables KATE to discover all 22 admittances (see Table 3-2 and Figure 3-3 for their definitions) that appear in the ALO-H20 knowledge base frames. It can be executed as a control procedure (see similar examples in the knowledge base file G:>KATE>ALO-KB>CONTROL-PROCEDURES) before each run of the model. The admittances Gm, Gn, and Gp must be updated for every procedure during model execution, since they depend upon valve positions. Pump

```
Placing ITERATE-IT into an inversion slot!
Placing ITERATE-IT into an inversion slot!
Placing ITERATE-IT into an inversion slot!
Placing ITERATE-IT into an inversion slot!
Placing ITERATE-IT into an inversion slot!
Placing ITERATE-IT into an inversion slot!
Placing ITERATE-IT into an inversion slot!
Placing ITERATE-IT into an inversion slot!
Placing ITERATE-IT into an inversion slot!
Placing ITERATE-IT into an inversion slot!
Placing ITERATE-IT into an inversion slot!
Placing ITERATE-IT into an inversion slot!
Placing ITERATE-IT into an inversion slot!
AIO errors: 27
INSTANCES errors: 133
AKO errors: 12
KINDS errors: 45
APO errors: 1
PARTS errors: 4
OTHERS errors: Ø
CONNECTION errors: 42

AIO errors: Ø
INSTANCES errors: Ø
AKO errors: Ø
KINDS errors: Ø
APO errors: Ø
PARTS errors: Ø
OTHERS errors: Ø
CONNECTION errors: Ø
```

(a)

```
Frame RV1Ø2's input P-IN doesn't jibe with frame NIL's output NIL.

Frame CV1Ø4's input P-IN doesn't jibe with frame NIL's output NIL.

Frame TX1Ø1's input TEMP-IN doesn't jibe with frame NIL's output NIL.

Frame RV1Ø1's input P-IN doesn't jibe with frame NIL's output NIL.
```

(b)

Figure 3-2. Consistency Checkers' printouts: a) Mondo's, b) Check-KB's.

TABLE 3-2.
ADMITTANCE DEFINITIONS

Fig.	Name[1]	Definition[1]	ALO-H2O[2]	FlowRatePres[3]
Tank pressurization circuits....				
A1	st_vent_admit	storage tank ullage vent admittance	p.7	-
A2	st_up_admit	" " " pressurization network "	p.9	-
A3	vt_vent_admit	vehicle tank ullage vent admittance	p.24	-
A4	vt_up_admit	" " " pressurization network "	p.25	-
Pump circuits....				
A5	pump_circuit_admit	From ST to PX1Ø6, used by SV1Ø4.	p.11,12	p.2
A6	pumps_admit	Nonlinear flow change with pressure.	p.11,12	-
A7	pump_to_4_way_admit	From pump1(or 2) to PX1Ø5.	-	p.1
A8	suction_line_admit	From ST to PX115.	-	p.6
A9	recirculation_admit	from PX1Ø6 to ST.	p.13	p.2
Aa	recirc_line_admit	From P1 to GA1Ø3.	-	p.6
Ab	No name.	= A9 - Aa, from above.	-	-
As	No name.	= A6 + A7, from above.	-	-
Ar	No name.	From PX1Ø5 to PX1Ø6.	-	-
Vehicle tank fill circuits....				
Ac	transfer_line_admit	Long piping & FCV1Ø3.	p.14	p.6
Ad	fast_fill_circuit_admit	SV1Ø7(8) and FM1Ø2.	p.16	p.4
Ae	final_fill_circuit_admit	MCV1Ø1 @ 33% and FM1Ø1.	p.19	p.4
Af	replenish_circuit_admit	Ditto, adjust % to hold VT level.	p.17	p.4
Vehicle tank drain circuits....				
Ag	tsm_drain_assist_admit	From PRU1Ø3 thru SV112 & SV116 to atmos	p.2Ø	-
Ah	drain_admit	From PX1Ø8 thru SV116 to atmosphere	p.18	-
Ai	No name.	Negligible wrt H2O-carrying drain.	-	-
Aj	nozzle_admit	" " " SV11Ø " " (= 2.5)	p.2Ø,21	-
Ak	bleed_admit	" " " SV1Ø9 " "	p.21	-
(NOTE: "tsm" = tail service mast.)				
Miscellany....				
--	rv_admit_max	= 1Ø (not a measurement)	p.24	-
Am	tank_fill_admit	PX1Ø6 to VT or drain, whichever open	p.2,4	-
An	upper_fill_circuit_admit	PX111 to VT or drain, "	-	p.6
Ap	skid_admit	From PX1Ø8 to PX111.	-	-
(NOTE: tank_fill, upper_fill, and skid admittances vary with procedure.)				

NOTE: Comments in the following listings provided these definitions:
1. G:>KATE>ALO-KB>CONTROL-PROCEDURES
2. G:>KATE>ALO-KB>ALO-H2O
3. G:>KATE>ALO-KB>FLOW-RATE-PRESSURES

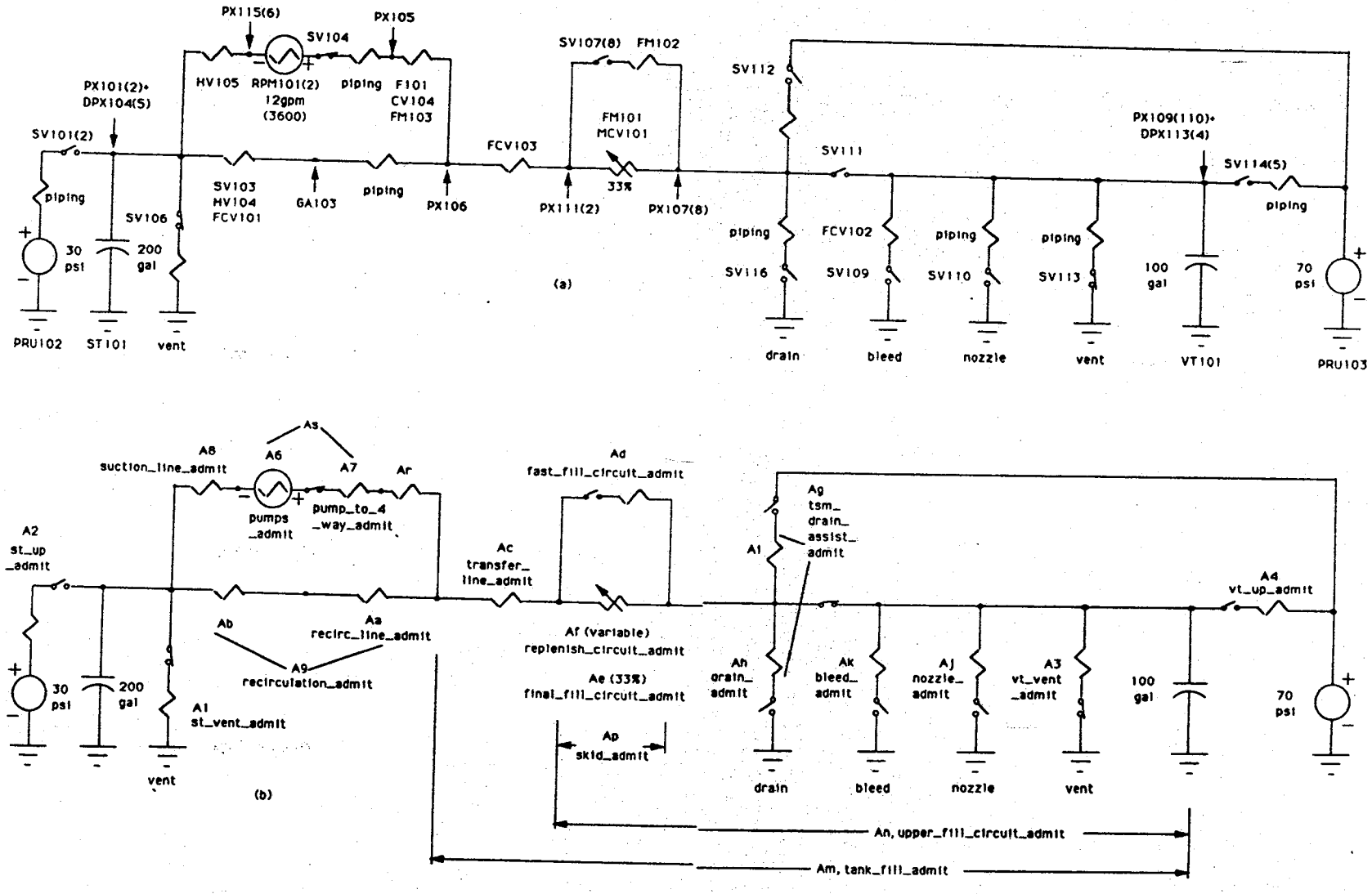


Figure 3-3. The ALO model's admittances: a) perspective (compare with Figure 1-1), b) admittances labeled as they appear in ALO's knowledge base frames.

admittances must be measured in isolation or gleaned from pump specifications before the admittance measuring procedure begins. This test procedure should be coded in Lisp and added to the procedures file above to enhance the ALO-H2O model and modeler productivity.

By the way, the admittance measurement procedure was designed earlier to measure only the admittances of the nine hand valves. This design was abandoned when the modeler realized upon inspection that it would not help him calibrate his 22 knowledge base admittances. Am I glad I completed a design before coding! The aborted design succeeded in pointing out a serious error, and thus served a very useful purpose. The design we abandoned also mistakenly assumed that pipe admittances are negligible compared with valve admittances. Laboratory researchers at Marshall Space Flight Center apparently still think that is true [6].

3.6 BETTER PSEUDO OBJECTS

KATE's modelers invented pseudo objects to apprise ordinary objects of remote pressures and flows in a complicated fluid flow circuit. The fluid flow output of a valve, for example, depends not only upon that valve itself but also upon the backpressure of its downstream load and the driving pressure of its upstream fluid supply. In KATE's ALO-H2O model, PX-1Ø6 is called the "all knowing" pressure. The rather complicated frame of this pseudo object evidences influence of far ranging pump flows and admittances. All other pressures and flows in the circuit can be readily computed from the keystone pressure PX-1Ø6, without the necessity of solving multiple simultaneous differential equations. Equation solving takes too much time, and it would interfere with KATE's failure diagnosis process. [6]

A pseudo object typically starts out looking like any ordinary object (i.e., a frame-modeled device), but then it starts growing like a cancer. Initially the flow through a device may be specified as a function of only local pressures. When modeled flows or pressures deviate from actuals, the modeler searches for any measurable time function which correlates well with actual flow, and then "mixes it in" with his original transfer function. These incremental improvements continue until the pseudo object's transfer function becomes a sort of multiple regression equation, combining the weighted contributions of many remote flow and pressure measurements throughout the circuit. A part of the specific application's knowledge base, the pseudo object describes the functioning of only one fluid flow circuit. But it is distinctly different from those true objects in the knowledge base which define the structure of the circuit. Stepchildren of the modeling process, pseudo objects do not fit well into either KATE's knowledge bases nor her shell. Furthermore, a single keystone pressure cannot be found in circuits that have multiple flow loops.

I resolved to find some alternatives to pseudo objects that would not run aground of the limitations imposed by on-line equation solving. Three alternatives are advanced below as proposal-ready philosophical white papers. The modelers' evaluations follow each description.

3.6.1 OFF-LINE EQUATION SOLVER ALTERNATIVE

If KATE needs an equation solver, but the equations aren't soluble in real time, why not solve all the equations KATE will need off-line and feed their analytic solutions to KATE as she needs them? A different solved equation is placed into the modeled knowledge base for every configuration of valves (e.g., every subprocedure in the LOX tanking procedure). Mathematica easily solves these simultaneous differential equations for all modeled flows and pressures as functions of time.

Consider the application of this technique to KATE's ALO-H2O model. During this model's slow fill subprocedure, only 12 of the 26 valves are open, producing the rather simple two-loop circuit of Figure 3-4. The modeler writes the two flow equations:

$$f1^2\,(1/aHV105^2+1/aHV107^2+1/aSV105^2+1/aF101^2+1/aCV104^2+1/aFM103^2)$$
$$+\,(f1-f2)^2\,(1/aFCV101^2+1/aHV104^2) = k * RPM102, \qquad (1)$$

$$\int f2\,dt \,/\, aVT101 = (f2-f1)^2\,(1/aHV104^2+1/aFV101)$$
$$+\,f2^2\,(1/aFCV103^2+1/aFM101^2+1/aSV111^2+1/aSV113^2). \qquad (2)$$

Off-line, Mathematica solves these for the time functions, f1 and f2, in terms of the circuit constants. The modeler codes f1 and f2 as pseudo object transfer functions for KATE to use throughout the slow fill procedure. Other objects in the knowledge base may refer to these flows in computing their own output pressures:

$$poutSV105 = pinSV105 - f1^2 \,/\, aSV105^2,$$

or even these time functions can be computed off-line.

Solving systems of simultaneous differential equations off-line moves the complexity out of KATE's shell that Whitlow found she couldn't bear [8]. This technique handles multiple loop flow models well. Constraining the valve closures of a particular modeled procedure also simplifies off-line equation solving. A small fraction of the original complexity (i.e., the flow time functions) moves into the knowledge base, where the modeler can handle it more cleverly than KATE could in her shell. Steve laments, however, that KATE's diagnoser would have to been modified to accommodate this procedure. The diagnoser currently twiddles valves, in an effort to duplicate erred hardware measurements, thus invalidating the simplified model and its off-line solution. Apparently these equations must be solved on-line.

3.6.2 ANALOG COMPUTER ALTERNATIVE

Is there a novel way to solve these equations on-line very rapidly? Yes, Abelson and Sussman suggest that the equations be solved by a signal processing analog computer [3]. Including the tank pressurization circuits, the ALO-H2O

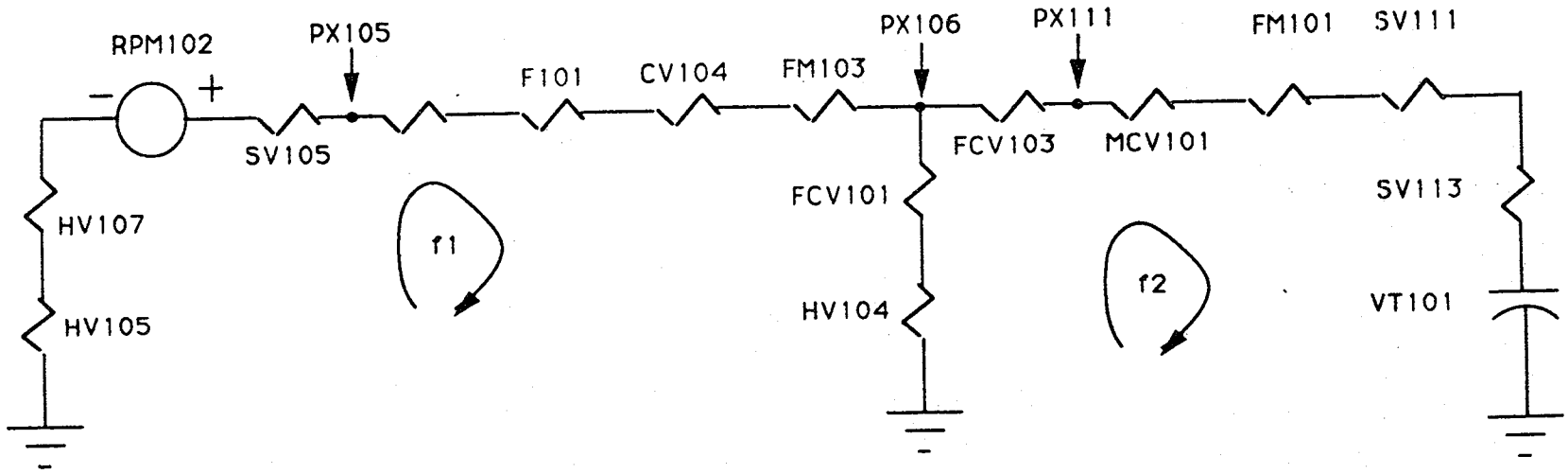


Figure 3-4. The simplified slow fill model.

model in slow fill includes four flow loops. Kirchoff's Law states that the sum of all the pressures around a loop must be zero. Four simultaneous equations state these sums in terms of the four flow rates around the loops of Figure 3-5.

$$\int f1\ dt\ /\ Cst = f1\hat{}2\ /\ A1\hat{}2, \tag{1}$$

$$k * RPM1\emptyset1 = (f2\hat{}2)(1/A8\hat{}2+1/A6\hat{}21/A7\hat{}2+1/Ar\hat{}2) + (f2-f3)\hat{}2\ (1/Aa\hat{}2+1/Ab\hat{}2), \tag{2}$$

$$(f3-f2)\hat{}2\ (1/A1\hat{}2+1/Ab\hat{}2+1/Aa\hat{}2) + f3\hat{}2\ (1/Ac\hat{}2+1/Ae\hat{}2) + (f3-f4)\hat{}2/A3\hat{}2 = \emptyset, \tag{3}$$

$$(f4-f3)\hat{}2\ /\ A3\hat{}2 = \int f4\ dt\ /\ Cvt. \tag{4}$$

After these equations are differentiated, a fast signal processing analog computer can solve them simultaneously.

$$df1/dt = A1\ /\ 2\ Cst. \tag{1}$$

$$\frac{df2}{dt} = \frac{df3}{dt}\ \frac{(f2-f3)\ (1/Aa\hat{}2+1/Ab\hat{}2)}{f2\ (1/A8\hat{}2+1/A6\hat{}2+1/A7\hat{}2+1/Ar\hat{}2) + (f2-f3)\ (1/Aa\hat{}2+1/Ab\hat{}2)} \tag{2}$$

$$\frac{df3}{dt} = \left[\frac{df1}{dt}\ \frac{f3-f1}{A1\hat{}2} + \frac{df2}{dt}\ (f3-f2)\ (1/Ab\hat{}2+1/Aa\hat{}2) + \frac{df4}{dt}\ (f3-f4)/A3\hat{}2\right] \times$$

$$[(f3-f1)(1/A1\hat{}2+1/Ab\hat{}2+1/Aa\hat{}2)+ f3(1/Ac\hat{}2+1/Ae\hat{}2)+ (f3-f4)/A3\hat{}2]\hat{}-1, \tag{3}$$

$$df4/dt = 2\ df3/dt + (A3\hat{}2\ /\ Cvt) * f4\ /\ (f4 - f3) \tag{4}$$

Figure 3-6 renders these equations graphically as an analog computer block diagram. Abelson and Sussman tell how to code signal processors that have troublesome feedback loops like these. Essentially they recommend that the fedback input be a global, given some initial value. That value is used to compute an output, and the output thus provides a new fedback value for the next sampling period.

The simple multiply-accumulate operations inherent in these signal processor difference equations run much faster than a general purpose simultaneous equation solver. Thus KATE's diagnoser can freely reconfigure valves on-line and can expect to harvest accurate flows and pressure values in real time. Integrators in the analog computer are inherently stable, thus combatting the convergence problem Whitlow encountered with his flow solver [8]. Experiments with Gensym's G2 program, however, indicate that any equation solver obscures device failures to some extent [6]. Is there a compromise between equation solving and our current pseudo objects?

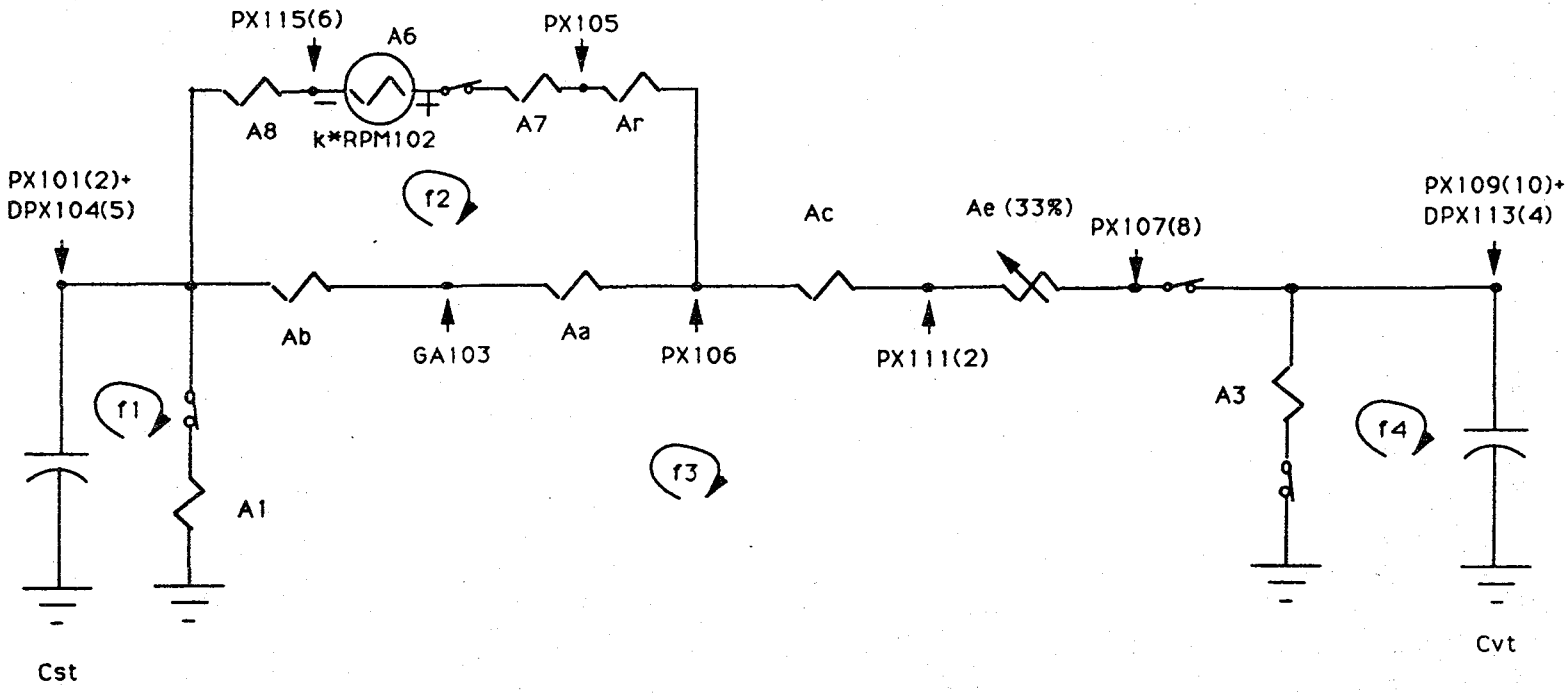


Figure 3-5. Four flow loops in ALO during slow fill.

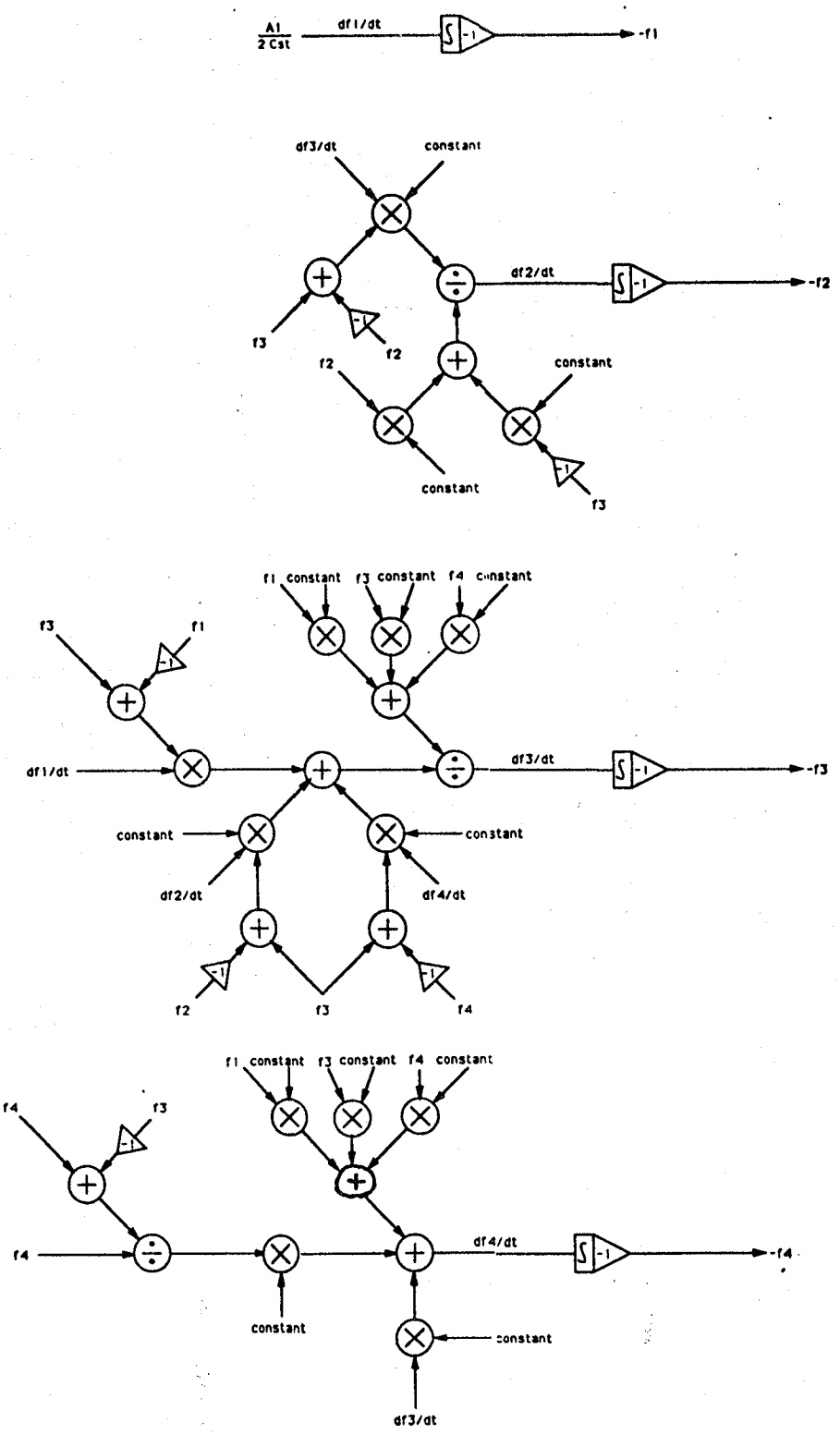


Figure 3-6. An analog computer designed to solve ALO's equations.

3.6.2 TWO-PORT ALTERNATIVE

The final alternative strives to distribute the complexity of KATE's pseudo objects over all the trueobjects, without requiring equation solving. The frame of every object describes the device itself, as well as all downstream (upstream) objects viewed through its input (output) port. Modeled objects are like a series of dirty windows, through which a viewer sees both the nearest window and other windows beyond. We obtain a Thevenin's equivalent circuit (to embody both local and remote devices) for each viewing port of every modeled device. Just the equivalent circuits of the two devices on either side of it are sufficient for computing any modeled pressure.

Thevenin's Law reduces a hydraulic network of any complexity to a single pressure source and a series admittance. It states that the equivalent pressure is the pressure that can be measured at the network's output. Furthermore, the equivalent admittance is that measured at the output when all network pressure and flow sources set to zero. (The admittance of a pressure source alone is zero.)

Figures 3-7 and 3-8 show how Thevenin's Law is applied to two-terminal devices and pipe tees in a hydraulic circuit. Looking into the input port of a two-port device in Figure 3-7a reveals the admittance itself and the equivalent circuit of all devices connected to its output. Furthermore, its internal admittance and the equivalent circuit of all devices connected to its input are visible at its output port. Thus, the device may be rendered as the equivalent circuit in Figure 3-7b, in which its input and output admittances and pressures may be computed.

$$Ain = An + Ao,\ Aout = An + Ai,$$

$$Pin = Po,\ Pout = Pi.$$

Similarly, the equivalent circuit of a pipe tee connected to the three devices shown in Figure 3-8a can be redrawn as the equivalent circuit of Figure 3-8b with the following parameter definitions.

$$Ain = 1 / (1/Ao + 1/Ab),$$

$$Aout = 1 / (1/Ai + 1/Ab),$$

$$Aby = 1 / (1/Ai + 1/Ao),$$

$$Pin = Po + (Pb - Po) * (Ao + Ab)^2 / Ao^2,$$

$$Pout = Pb + (Pi - Pb) * (Ab + Ai)^2 / Ab^2,$$

$$Pby = Pi + (Po - Pi) * (Ai + Ao)^2 / Ai^2.$$

With these definitions, we can model pressures and flows anywhere in a hydraulic circuit simply by referring to their adjacent devices. The affects of all remote devices are handled recursively by equivalent circuit formulas in the

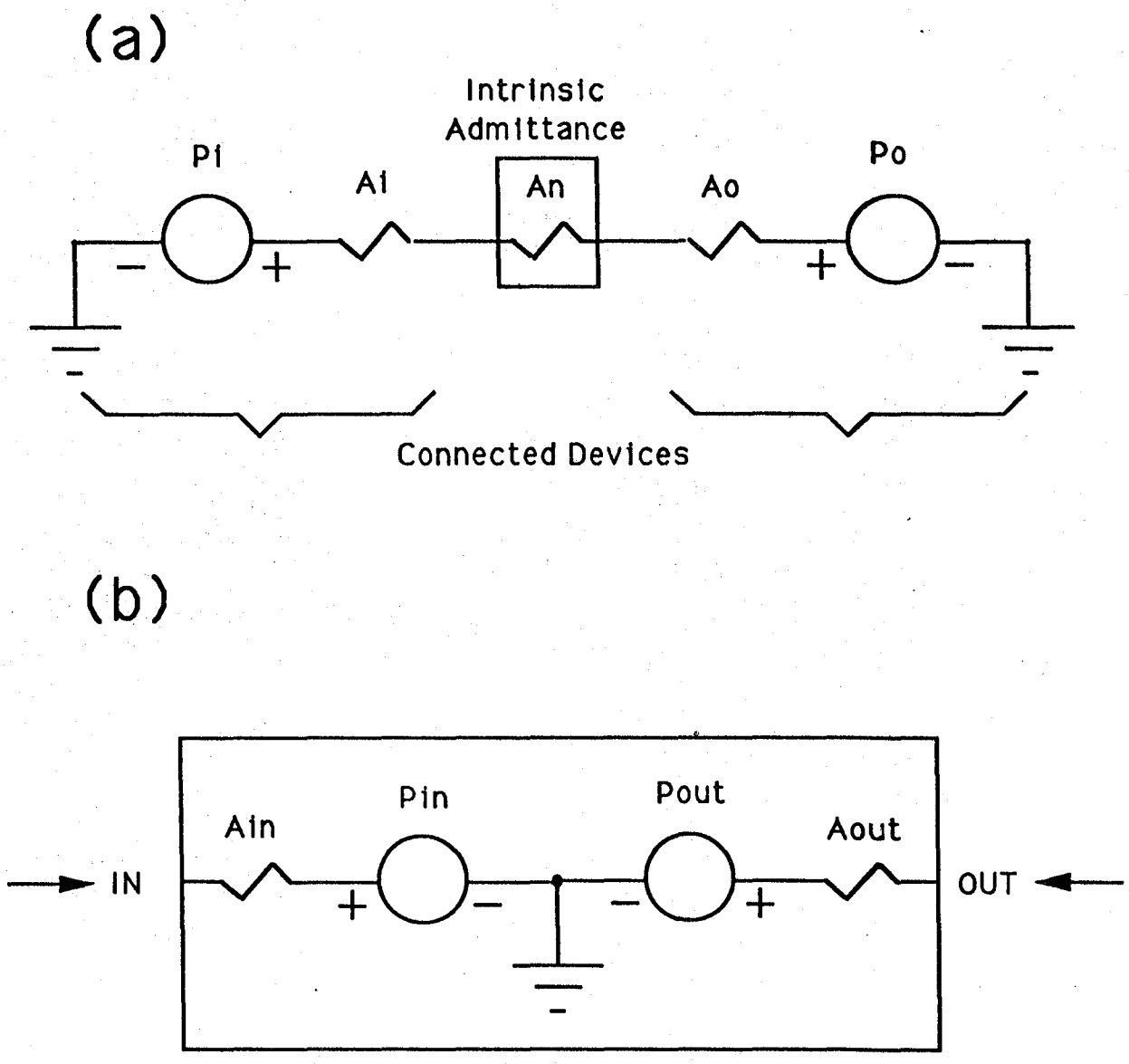


Figure 3-7. Thevenin's equivalent of a two-port device: a) device environment, b) equivalent device model.

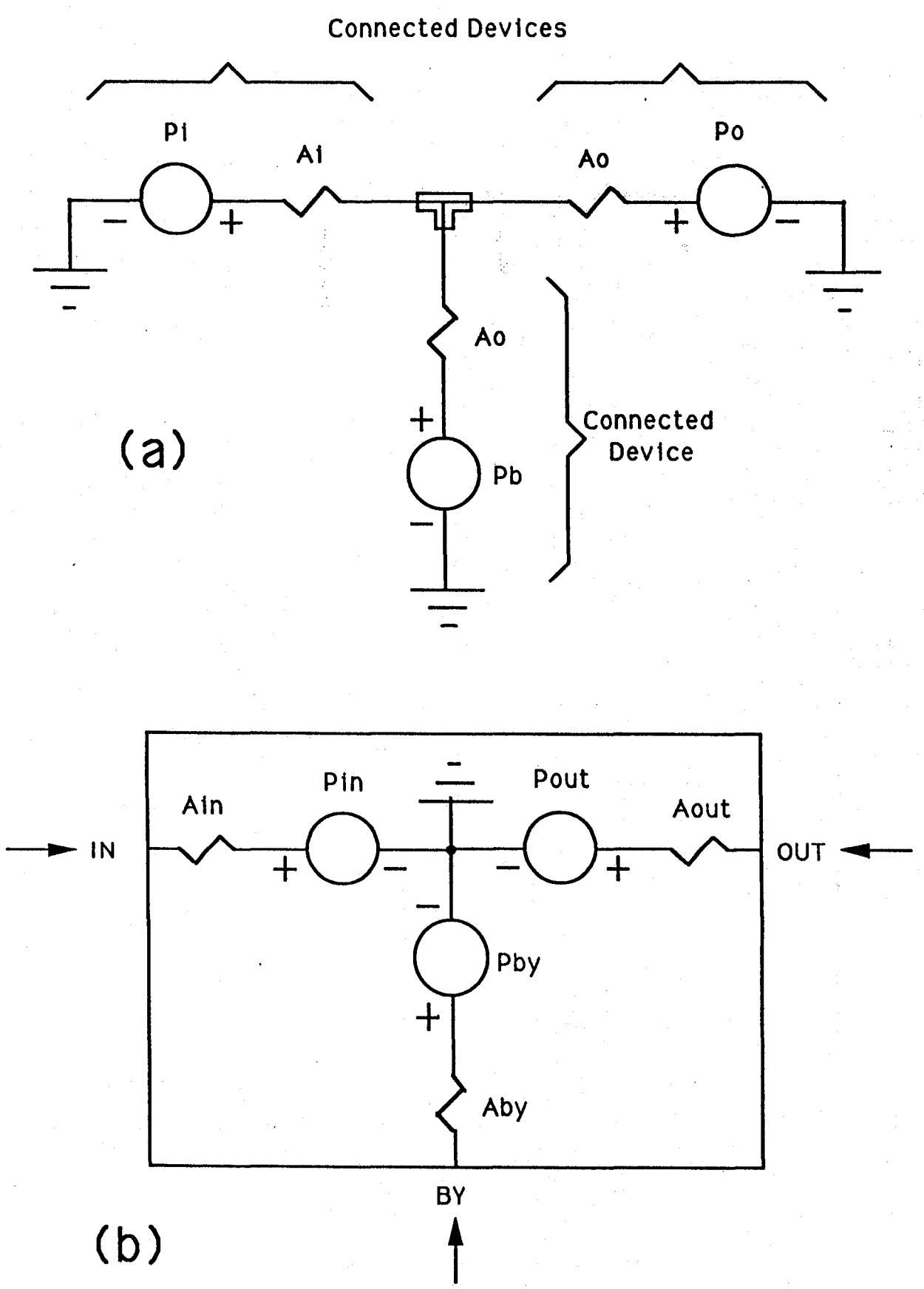


Figure 3-8. Thevenin's equivalent of a pipe tee: a) device environment, b) equivalent device model.

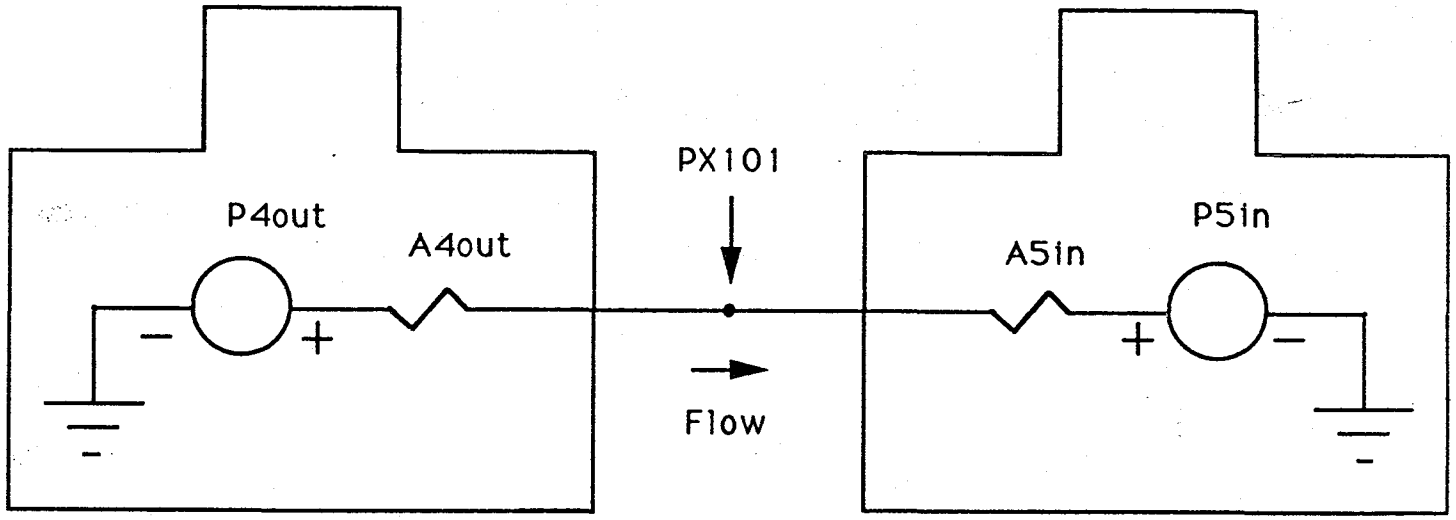


Figure 3-9. Two-port modeling an AL0-H20 pressure.

connected frames. For example, fluid flowing from tee 4 to tee 5 in a section of KATE's ALO-H2O model (see Figures 3-9 and 1-1),

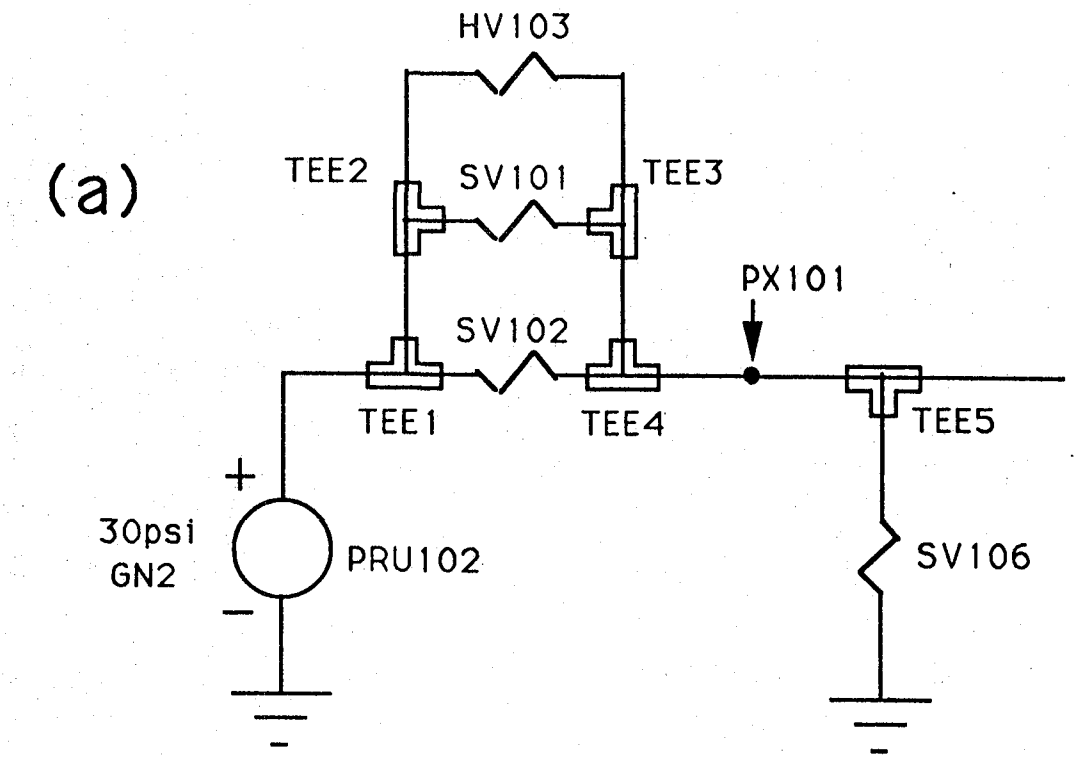
Flow = SQRT(P4out - P5in) * (A4out + A5in).

And the pressure PX1Ø1 at that node,

PX1Ø1 = P4out + (Flow / A4out)^2.

Can the two-port object model flows and pressures in multiple loop circuits, or do infinite recursions result? No problem. Every remote device is viewed in the same direction as the local device for which an equivalent circuit is sought. Shown in Figure 3-1Ø is the former example in which we viewed tee 4's output port. Looking into tee 4's output port, we see the output ports of SV1Ø2 and tee 3. Looking into tee 3's output port reveals the output ports of SV1Ø1 and HV1Ø3. Viewing HV1Ø3, SV1Ø1, and SV1Ø2's output ports reveals the output ports of tee 2 and tee 1 and the bypass port of tee 2. Finally, looking into the bypass and output ports of tee 1 reveals the output port of PRU1Ø2, which is defined. No viewing step reflects back upon the viewer; no infinite recursion occurs in the equivalent circuit modeling process. A coding suggestion appears in Appendix F.

The two-port alternative is attractive because it is so similar to the original pseudo object idea. Thus it would be the easiest alternative to implement under KATE's current architecture. It offers more precision than the ALO-H2O model's all-knowing P1, because it exactly reflects the affects of remote devices (it is not the trial-and-error result of "mixing in" correlated measurements). A two-port object truly is an "object," retaining the devices' structural position in the circuit. Yet in its greater complexity, it accounts for the modeled device's environment, without need to resort to all-function pseudo objects. Like a neural network, each device is aware of its only nearest neighbors, yet that is enough to model the whole circuit, as viewed in one direction from a single point. This alternative does not obscure the affects of failed devices as an equation solver does; thus it is more compatible with KATE's diagnoser. It is hard to say in detail just how diagnosis might be affected by implementing the two-port pseudo object alternative.



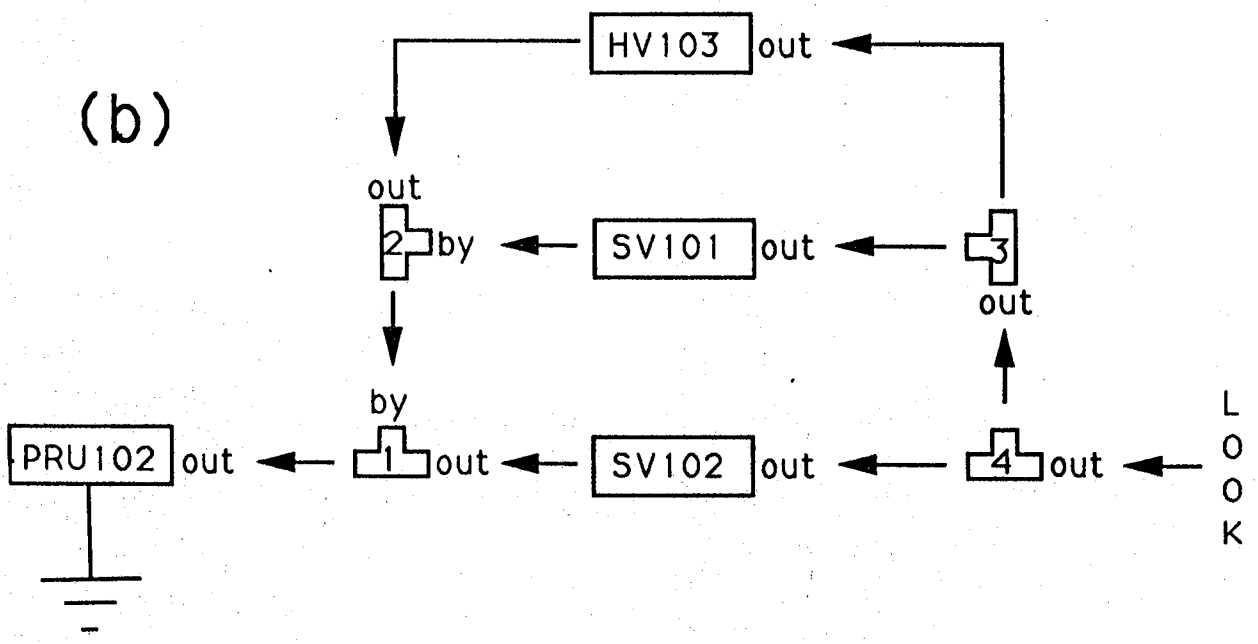


Figure 3-1Ø. No infinite recursions in multiple loops: a) ALO circuit segment, b) views always leftward.

IV
RESULTS AND DISCUSSION

4.1 MY MODEL VERIFICATION TOOLS

I added five useful tools to KATE's Model Verification Toolkit this summer. The already coded transfer function curve fitter and translator will make building new models easier, starting immediately. Documentation of KATE's three structural consistency checkers makes two of them available right away to all modelers, and the more powerful third will be useful as soon as it can be enhanced to match KATE's new knowledge bases. Bob has already decided to code the my automated admittance calibrator design in his spare time, with or without funding. Evidently he expects it to boost his productivity. Several stimulating discussions of the three pseudo object alternative white papers revealed some useful ideas in the first two, even if they are not altogether workable. The third alternative may enable KATE to embrace multiple flow applications that previously could not be modeled.

4.2 KATE'S FUTURE

How else can we make KATE's modelers more productive? There are two answers to that question. Traditional computer-automation aids, such as Computer Aided Design terminals, serve the end user. Mostly clerical, this modeler's task can be described in a series of very predictable steps for which KATE's responses are (hopefully) well known. KATE's end user needs a highly interactive graphical Knowledge Base Editor for creating frames, a Check-All-Frames syntax checker that doesn't miss a thing, and more refined on-line plotting facilities for debugging transfer functions. But KATE's end user hasn't shown up for work yet. And it's a good thing, because his Knowledge Base Editor isn't ready. KATE's developer/modelers have more difficult productivity problems. Redesign and recoding efforts (e.g., softlanding and external influences) spring up regularly in response to modeling problems that are currently beyond KATE's grasp. Weekly reports of these incremental changes in KATE provide evidence that she is a prototype, not a production program.

Production programs always pass through a Software Development Life Cycle (see Figure 4-1) before their release to end users. The development steps along the left of this "waterfall" chart start with a proposal phase, and proceed to a rapid prototyping sessions with the new customer if the proposal is successful. The rapid prototype uses powerful computers (like Symbolics) and powerful languages (like Lisp) to quickly define required system performance and user interactions before design begins. Design occurs in three top-down phases: system design to allocate jobs to subsystems, preliminary design to find the best path to module implementation, and detailed design to specify the performance of every software module. Coding follows. According to plans written during the design phases, testing proceeds from the bottom-up along the right side of the waterfall chart. Module testers diligently seek errors in modules, so that coders can remove them and increase product quality. Errors repaired here are

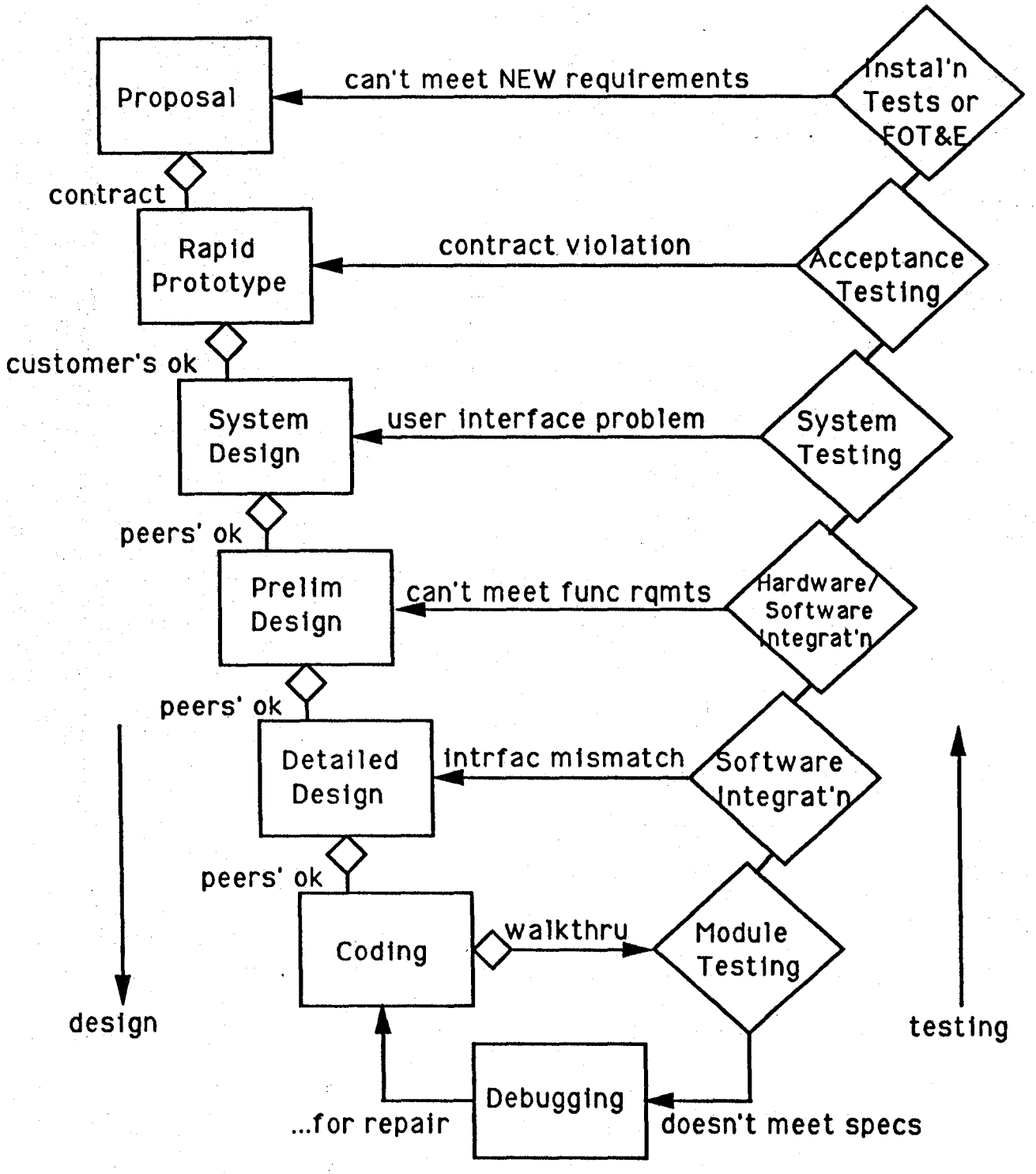


Figure 4-1. The Software Development Life Cycle.

27Ø times cheaper to fix than those found during production use. [7] In the software integration phase, modules are added to the evolving system, one at a time from the bottom-up, as module interface errors are discovered and eliminated. The hardware/software integration phase sees working subsystems being joined together with a similar testing goal. The system testing phase scrutinizes the complete system's responses to user commands, looking for errors the system engineer inadvertently designed in. Acceptance tests convince the customer that the system is ready for production use. Installation tests qualify each copy of the system for work at the user's site, or... they may provide content for a new proposal. Thus begins another trip through the Software Development Life Cycle.

For the past several years, KATE has been stuck in one proposal and rapid prototyping cycle after another. Her errors have been costly to fix, but her conceptual design has gradually matured to the point of convincing some that she will be able to handle critical tasks at KSC someday. Soon KATE will win her long-awaited ride through the Software Development Life Cycle. Folks here refer to the event as "recoding KATE in the C language for higher speed and lower cost," but you and I realize that Symbolics' Lisp machines are for prototyping and that C-programmed PCs are KATE's logical target system. Given an alert Computer Scientist as Configuration Manager, KATE will soon be as error free and incredibly reliable as KSC's other launch equipment. She will have the pedigree that she needs to win positions of responsibility and respect around KSC.

IV
CONCLUSIONS

KATE had some wonderful Model Verification Tools before I came, and now she has a few more (transfer function curve fitter, translator, and three documented syntax checkers). She has the prospect of some even more powerful ones in the foreseeable future (automated admittance calibrator and better pseudo objects). Her forthcoming "recoding in C" promises a error-purging turn through the Software Development Life Cycle.

With NASA and Boeing's able leadership, I have trained myself for KATE development work next summer by:

1. Surveying KATE's modelers.
2. Coding two new Model Verification Tools.
3. Enhancing some existing tools.
4. Designing a new tool.
5. Writing white papers on a better modeling technique.

I'm already looking forward to next summer.

KATE is an incredibly clever artificially intelligent computer program whose time has come. It is thrilling to be counted part of her development team just as she begins to turn a profit.