Explanation Constraint Programming for Model-based Diagnosis of Engineered Systems[1][2]

Sriram Narasimhan, Lee Brownston
QSS Group Inc.,
M/S 269 – 3,
NASA Ames Research Center,
Moffett Field, CA 94035
650 604 0832
sriram@email,lbrownston@ptolemy.arc.nasa.gov

Daniel Burrows
Penn State University,
burrows@cse.psu.edu

Abstract—We can expect to see an increase in the deployment of unmanned air and land vehicles for autonomous exploration of space. In order to maintain autonomous control of such systems, it is essential to track the current state of the system. When the system includes safety-critical components, failures or faults in the system must be diagnosed as quickly as possible, and their effects compensated for so that control and safety are maintained under a variety of fault conditions. The Livingstone fault diagnosis and recovery kernel and its temporal extension L2 are examples of model-based reasoning engines for health management. Livingstone has been shown to be effective, it is in demand, and it is being further developed. It was part of the successful Remote Agent demonstration on Deep Space One in 1999. It has been and is being utilized by several projects involving groups from various NASA centers, including the In Situ Propellant Production (ISPP) simulation at Kennedy Space Center, the X-34 and X-37 experimental reusable launch vehicle missions, Techsat-21, and advanced life support projects. Model-based and consistency-based diagnostic systems like Livingstone work only with discrete and finite domain models. When quantitative and continuous behaviors are involved, these are abstracted to discrete form using some mapping. This mapping from the quantitative domain to the qualitative domain is sometimes very involved and requires the design of highly sophisticated and complex monitors. We propose a diagnostic methodology that deals directly with quantitative models and behaviors, thereby mitigating the need for these sophisticated mappings. Our work brings together ideas from model-based diagnosis systems like Livingstone and concurrent constraint programming concepts. The system uses explanations derived from the propagation of quantitative constraints to generate conflicts. Fast conflict generation algorithms are used to generate and maintain multiple candidates whose consistency can be tracked across multiple time steps.

TABLE OF CONTENTS

1. INTRODUCTION

Model-based Diagnosis (MBD) uses a general-purpose model of the internal structure and/or behavior of systems to perform the diagnosis task [1]. Such models may be structural and/or functional models. The diagnosis based on these models takes advantage of the analytical redundancy in the models, which is captured as static and/or dynamic relations between the inputs to the system and the outputs from the system. The basic principle of MBD can be understood as the interaction between predictions and observations (Figure 1). The system behavior is measured by sensors. The computational model of the system can be used to predict what these measurement values should be under nominal conditions. The predicted behavior is compared to the observed behavior to identify any discrepancy.

[1] 0-7803-8155-6/04/$17.00© 2004 IEEE
[2] IEEEAC paper #1139, Version 3, Updated December 02, 2003

The Livingstone fault diagnosis and recovery kernel [2] and its temporal extension L2 [3] are examples of model-based diagnosis engines. Using a simple language developed specially for L2, domain experts can program discrete-state models of components of systems like spacecraft. Each component's various nominal and failure modes, along with constraints upon the transitions between modes, are specified. Multiple components can then be composed into one complete model representing a finite-domain concurrent transition system description of a complete device to be monitored, diagnosed, and possibly controlled. At runtime, L2 accepts input in the form of commands issued to the system and the observations (through sensors) from the system. Commands are propagated through the model system and when conflicts between the model state and the sensed state are detected, a diagnosis can be performed. The diagnostic process consists of a search, starting from the current hypothesized state, for a model state consistent with current observations. Usually this results in the diagnosis of a failed component or set of components. Failure candidates are found in order of their likelihood, based upon *a priori* individual component failure probabilities. The use of conflicts (discovered by the truth maintenance system representation of the problem) to focus the search allows for real-time performance on relatively complex spacecraft models.

Livingstone successfully flew on Deep Space One as part of the Remote Agent experiment. L2 is currently being used for diagnosis of systems from a wide variety of domains including International Space Station [4]. However, when L2 was applied to the propulsion system of the X-34 reusable launch vehicle as part of Propulsion IVHM Technology Experiment (PITEX), several limitations were exposed. Many artificial artifacts had to be introduced into the models and sophisticated "monitors" had to be developed to convert the sensed data into a form that is suitable for input to L2. These problems are direct consequence of the inability of L2 to handle quantitative and/or continuous dynamics in the system including transients. Similar problems have manifested themselves when trying to use L2 in other applications like the Reverse Gas Water Shift (RWGS) system and the ISS Command & Data Handling (C&DH) system.

In this paper, we identify these key limitations of the L2 health management system and present an extension called Livingstone 3 (L3) that addresses these limitations. We will present the new modeling framework that provides the support for development of models with new features like continuous dynamics and the diagnosis architecture that contains the algorithms that use these models for diagnosis of hybrid systems (systems that exhibit discrete as well as continuous behavior). Finally we will show the functioning of the diagnostic engine through an example and tabulate results from tests on applying L3 to a set of examples.

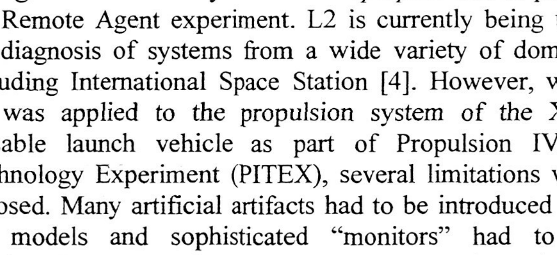


Figure 1: Model-based Diagnosis

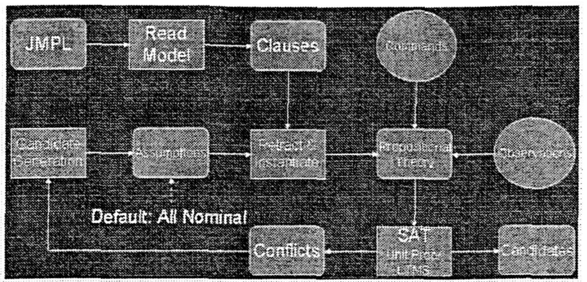


Figure 2: L2 Diagnosis Architecture

2. L2 AND ITS LIMITATIONS

Livingstone and L2 have focused on modeling the system to be diagnosed using a set of high-level, qualitative models that focus on a discrete characterization of the system's behavior. The underlying model that L2 uses for diagnostic reasoning has to be in propositional logic form, more specifically in clausal form. The two main entities of this framework are *Propositions* and *Clauses*. Propositions are variables that are Boolean valued and can take on one of three values TRUE, FALSE, or UNKNOWN. Propositions are used in L2 to represent the variables in the system as well as operating modes of the system and the mode transitions between the operating modes. Faults in the system are modeled as transitions to a faulty mode of operation. It is assumed that the behavior of the system in this faulty mode can also be specified (sometimes unknown faulty modes that allow any behaviors are used to capture all unanticipated faults). The diagnostic problem is to determine which if any of this faulty mode transitions occurred. Clauses are disjunctions of propositions or negations of propositions. These represent the constraints on the behavior of the system.

The Livingstone (L2) diagnosis approach is presented in Figure 2. Diagnosis is performed as a two step procedure, with the steps alternating. In the first step, possible candidates are generated and then in the second step, these candidates are tested for consistency with the commands and observations seen from the system. A candidate is a set of faulty mode transitions (possibly empty indicating no fault) that the system is assumed to have taken at each time step in its behavior evolution. Candidate generation is based on a conflict directed search [5] algorithm. Each candidate is tested for consistency through Boolean constraint propagation (unit propagation) through the set of clauses that the candidate entails. Commands and Observations are also added as clauses before propagation. In order to handle transition systems, clauses at several time steps are maintained as well as transition clauses between time steps.

If the propagation results in an inconsistency (the set of clauses cannot be satisfied by any assignment to propositions), a truth maintenance system is used to identify minimal conflicts to guide the candidate generation.

This underlying propositional logic representation presents some problems when modeling systems that are not Boolean in nature such as circuits and logic gates. Currently Livingstone supports models that involve variables with values from a finite domain and very simple constraints. Finite domain variables are converted to propositions by creating a proposition for each value the variable can take and creating a set of clauses that ensure one and only one of these propositions may be true. For example, if a variable X can take values (low, nominal, high), the following propositions are created:

P1: X=low
P2: X=nominal
P3: X=high

and the following clauses are created:

C1: X=low v X=nominal v X=high
C2: ~X=low v ~X=nominal
C2: ~X=low v ~X=high
C2: ~X=nominal v ~X=high

It is clear that the number of clauses just to represent one variable is $(n + 1 + {}^{n}C_{2})$ where n is the number of values that the variable can take. Obviously it is impossible to represent variables from an infinite domain (e.g., real-valued variables) in this framework.

Since only clauses are allowed to model system behavior, the modeling is restricted. In addition to Boolean formulae, only equality of variables is allowed. Even for representing this simple constraint a large number of clauses are required. For example, X=Y has to be represented as:

C1: ~X=low v Y=low
C2: ~X=nominal v Y=nominal
C3: ~X=high v Y=high
C4: ~Y=low v X=low
C5: ~Y=nominal v X=nominal
C6: ~Y=high v X=high

It is clear that $(2 * n)$ clauses are required to represent an equality constraint where n is the number of values in the variables domain.

When dealing with variables that are from an infinite domain, they have to be converted to a set of finite values using *binning* strategies. For example we might consider three bins, low ($-\infty$ to 0), nominal (= 0), and high (0 to ∞). The binning strategy is dependent on the different behaviors we want to distinguish. There are several problems with both designing and then implementing the binning strategy. The design is non-trivial and requires extensive domain knowledge or lots of simulation (including fault simulation) results. Even after that, there may be too many bins (leading to a large number of clauses as seen earlier) or the mapping may be too complex. The implementation of the mapping from the continuous domain to the discrete L2 representation is sometimes very involved and requires the design of highly sophisticated and complex monitors due to the presence of sensor noise, system transients, etc.

Given these limitations, both the modeling framework and the diagnosis algorithms need to be changed in order to allow efficient diagnosis of systems that include quantitative and continuous behavior. In the next section we will present our new modeling framework for Livingstone 3. In the following section, we will present the Livingstone 3 diagnosis architecture that allows the diagnosis of hybrid systems.

3. Modeling Framework

Our modeling approach is based on the hybrid automata formalism [6]. In this formalism, the discrete state of the system is modeled as a finite state automaton. Each state of the automaton represents a mode of continuous operation of the system. Accordingly, within each state, a set of equations (differential and algebraic equations) are used to describe the behavior of the system in that mode. The transitions between the various states of the automaton may be based on external events/commands (commanded transitions) or internal conditions (autonomous transitions). The two limitations we had with this formalism were that (i) we wanted to generalize beyond differential algebraic equations and (ii) we wanted to use a component connection modeling approach rather than specify a single finite state automaton for the entire system.

We use a hierarchical and component connection model in conjunction with hybrid constraint automata as our modeling framework. The basic building blocks of the model are components, which typically represent a physical device, or a physical subsystem, or a logical grouping of subsystems in a system. Components may be nested inside components to build up a hierarchy to any arbitrary depth. Components contain variables, constraints, and at most one finite state automaton. Variables could be from any domain, for example boolean, enumeration, real valued, interval valued etc. Some variables in the component are identified as port variables. These variables are special variables that can be used create connections between components. A connection between port variables in two different components indicates that the variables are constrained to be equal valued. Variables are also marked as state, observable, and input indicating whether they are state variables, measured or input to the system respectively. Constraints represent relations over variables. The types of relations that can be specified depend on the types of the variables.

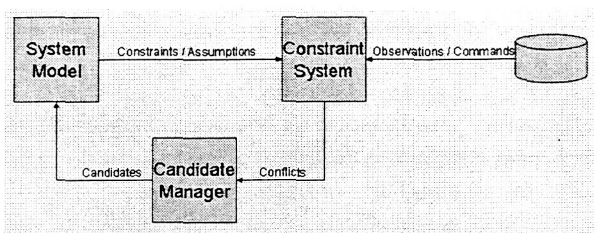


Figure 3: L3 Diagnosis Architecture

For example in the Boolean domain, any Boolean formula can be used as a relation whereas in the real valued domain, any differential and algebraic equation may be used as a relation. Since these constraints do not depend on the mode of the component, we call them persistent constraints implying that variables can only take on values so that these constraints are satisfied at all points in time.

The finite state automaton (FSA) in the component is used to represent the continuous modes of operation of the component similar to the hybrid automata approach. Each automaton consists of a set of locations (representing the states of the finite state automaton) and transitions between the locations. Each location consists of a set of constraints that describe the relations that hold over the variables when the component is in that location (mode). We call these volatile constraints to indicate that they need to be satisfied only when the component is in that mode of operation. The transition between locations consists of a guard condition and a probability value. The guard condition is special kind of constraint that evaluates to a Boolean value. For example, in the Boolean domain this could be any Boolean formula but in the real valued domain it is any relational formula. We call the guard condition an "ask" constraint, since if the constraint evaluates to true then the transition is taken. The probability on the transition indicates likelihood of the transition being taken. This probability is especially needed to represent faulty transitions. Some transitions are marked as faulty transitions, indicating that these are autonomous (not commanded) and unobservable events. These are the events whose occurrence we want to diagnose. They have no guard conditions and the probability on these transitions indicate the prior probability of such a fault occurring.

4. DIAGNOSIS ARCHITECTURE

The L3 diagnosis architecture is illustrated in Figure 3. It consists of three main components. The *system model* stores the model of the system and is responsible for tracking the modes of operation of the different components and determining the constraints that are valid at any point in time. The *constraint system* serves the role of tracking the overall system behavior using constraint programming techniques. It receives constraints from the System Model indicative of the current configuration of the system and propagates these constraints in order to try to assign consistent values to variables in the system. When inconsistencies are seen (observations are different from propagated values for corresponding components), the *candidate manager* is responsible for using the conflicts generated as a result of these inconsistencies to generate candidates that resolve all the conflicts and that can possibly explain all of the inconsistencies.

The functioning of L3 is best explained through the set of steps taken during the run of the diagnostic engine. However we will need to define a few terms that will be used in this description.

Location Assumption: An assumption that a component is in a specific location. This typically applies only to nominal locations.

Transition Assumption: An assumption that a component took a specific transition. This typically applies to only faulty transitions.

Theory: The set of variables and constraints over variables in the constraint system.

Conflict: A set of assumptions that make the theory inconsistent and hence cannot all be true.

Candidate: Any set of faulty transition assumptions.

Valid Candidate: A candidate that resolves all conflicts and does make the theory inconsistent.

The diagnosis proceeds in the following manner. Initially the candidate manager has only one candidate called the empty (null) candidate, which has no assumptions. This is used to indicate that initially it is assumed that all components are non-faulty. At each time step the following things happen for each candidate in the candidate manager:

- Initialize Variable values – The values for state variables are obtained from the values at the previous time step (at time 0, state variable values are externally specified). For all other variables the values are initialized to any possible value in their domains.
- Persistent Constraints are added to the constraint system. Persistent constraints correspond to constraints that are satisfied independent of the location of components. In addition all component connections are converted to persistent constraints of the form Variable = Variable.
- Commands issued at the current time step are added as constraints to the constraint system. In this the constraints take on the form Variable = Value where Variable is the input variable and Value is the value it takes.
- Volatile Constraints corresponding to the current locations for all components are added to the constraint system. Volatile constraints correspond to constraints that are only true in specific locations of the components. The location assumptions are used to determine the current locations of the components and then the constraints in those locations are added as volatile constraints.

- The constraint system is propagated. Several propagation algorithms including brute force and arc consistency have been implemented.
- Observations obtained for the current time step are converted to constraints which are then compared against predictions for corresponding variables in the constraint system. To do this, we convert the observation into an "ask" constraint of the form Variable = Value. We can then ask this constraint of the appropriate constraint. Asking the constraint is equivalent to checking if the constraint is satisfied. Note that in this form of ask we are interested in determining if the constraint may possibly be true which can be distinguished from another form of ask that checks if the constraint is necessarily true. The implementation of the possible "ask" should take into account a number of factors like sensor noise, modelling errors etc. So for example in a real valued constraint store, the possible ask should return true if the observation value is within some threshold (dictated by noise characteristics) of the predicted value.
- For each observation that is not consistent with the theory, conflicts are generated from the explanations that were created during the propagation. We first determine the constraints whose propagation caused the variable involved in the "ask" to be inconsistent. We can then trace back from these constraints to any location assumptions that were used to add the constraints to the constraint store. The conflicts generated are across time steps i.e., the inconsistent variable value may have been because of a location assumption at previous time steps. This is possible since state variable values are held across time steps.
- If conflicts are generated, these are added to the candidate manager and valid candidates are generated. The candidate manager is responsible for generating a candidate that resolves all conflicts. A conflict is said to be resolved by an assumption A if any one of its constituent assumptions is resolved by A. A location assumption is resolved by any faulty transition assumption from that location. A transition (faulty) assumption is resolved by any sibling faulty transition assumption. Transition T' is a sibling of transition T if the "from" location of T' and T are the same. A candidate is a set of faulty transition assumptions. A candidate resolves a conflict if the candidate has at least one assumption that resolves the conflict. We have implemented several search strategies like breadth first, depth first, best first and A* searches that generate candidates.
- Transition conditions are evaluated and all enabled transitions are fired to update the current locations of all components. For this, the constraints on the transitions are asked of the constraint system. The form of "ask" we are interested in determining if the constraint is necessarily satisfied. When dealing with real values this might result in problems if we do not use probabilistic methods.
- The time step is advanced.

5. Examples & Results

We have run the Livingstone 3 diagnostic engine on several test examples including some that were used as regression tests for the L2 system. The diagnostic performance of L3 on the L2 examples was comparable. In addition, L3 is able to deal with models that include differential and algebraic equations as constraints. The examples we used for our tests were the nanoCBandLED (3 components), microCBandLED (12 components) and the CBandLED (24 components) models that are used as regression tests for L2. However these models involve only finite domain variables and constraints. We transformed the nanoCBandLED to include real-valued variables and algebraic constraints for testing with L3. Finally we also used tank system models involving tanks, inlet pipes, outlet pipes and connecting pipes to test L3 on systems with differential equations and autonomous transitions. These tank systems have been used as test beds for other diagnostic technologies [7, 8].

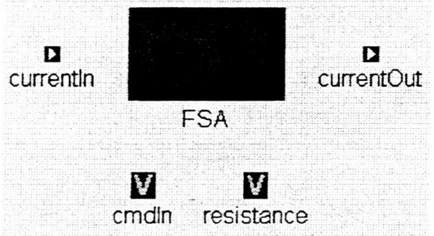


Figure 5: Circuit Breaker Model

We will illustrate the functioning of L3 through a simple nanoCBandLED example that includes algebraic constraints. The nanoCBandLED system is illustrated in Figure 4. It consists of three types of components, current source which is a source of a constant current, a circuit breaker that can be either on or off. If it is on, it passes the current through and if not it does not. The LED is like a bulb in that it lights up when current passes through it. However the amount of illumination is dependent on the amount of current that passes through it. There is also a summing junction that sums the current flowing into it and sends it out. The model of a circuit breaker (CB) is illustrated in Figure 5. It consists of currentIn, currentOut and conductance variables that have real values. It also has a

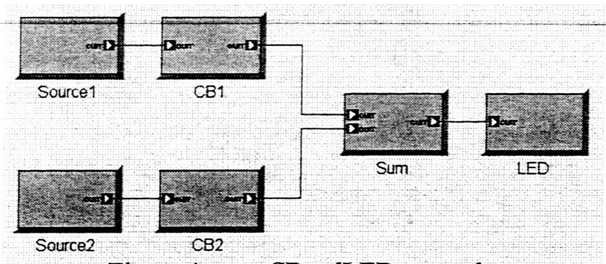


Figure 4: nanoCBandLED example

cmdIn variable that is a signal to turn the CB on and off. The FSA (illustrated in Figure 6) describes the different modes of operation of the Circuit breaker and the constraints that hold true in each of those modes. The constraints in each of the four modes of the CB are given by:

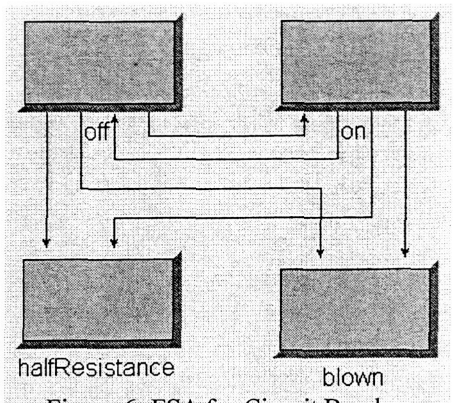


Figure 6: FSA for Circuit Breaker

- off: currentOut = 0.0
- on: currentOut = conductance * currentIn
- halfResistance: currentOut = 0.5 * conductance * currentIn
- blown: currentOut = 0.0

We will consider a scenario where the both CB's are in the halfResistance mode. As a result when they are turned on, they do pass through all the current but only half of it. This results in the LED only receiving only half the normal current from each CB. The fault scenario used is encoded as :

```
0.0 COMMAND CB1.cmdIn on
0.0 OBSERVE LED.Illumination 0.0
1.0 OBSERVE LED.Illumination 2.5
1.0 COMMAND CB2.cmdIn on
2.0 OBSERVE LED.Illumination 5.0
2.0 COMMAND CB1.cmdIn off
3.0 OBSERVE LED.Illumination 5.0
3.0 COMMAND CB2.cmdIn off
4.0 OBSERVE LED.Illumination 5.0
```

where the first column indicates the time, the second column indicates whether a command was issued or an observation was made, the third column indicates the command or observation variable and fourth column indicates the actual observation made. The initial states of both the CB's are off. The source always supplies a constant current of 5 units and all the conductance values are 1. When L3 is run on this fault scenario it returns the following results:

```
Candidates

{ Size: 2 Probability: 4e-05
   Assumptions for time: 1[
      Assumption [CB1 Transition fromLocation='on'
toLocation='halfResistance' ]
   ]
   Assumptions for time: 2[
      Assumption [CB2 Transition fromLocation='on'
toLocation='halfResistance' ]
   ]
}

TimeOfFaultDetection = 1

Candidates visited: 55
```

L3 figures out that both CB's are at halfResistance (only those modes make the observations consistent with the model). In order to achieve this, it looks at 55 possible candidates before finding the correct one. This is because it tests candidates in the order of prior probability of failure (for multiple faults, probability is product of individual probabilities) and as a result all single faults are typically tested before double faults. Also since we are dealing with a transition system, the same fault occurring at different time steps could have different effects on the behavior of the system and so L3 has to consider different combinations of time steps and faults to consider all possible cases.

5. CONCLUSIONS AND FUTURE WORK

We presented the Livingstone 3 model-based diagnosis system that extends Livingstone 2 to allow diagnosis of hybrid systems. We also presented the modeling framework that allows the capture of quantitative and continuous constraints for use by L3. This system is capable of identifying multiple discrete faults in systems that exhibit a mix of discrete and continuous behavior. We have tested L3 on some simple examples, some of which are used as test beds for other diagnostic approaches, and the results are promising.

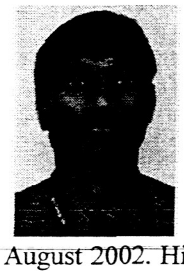
In future work, we would like to develop other constraint systems to be included as part of architecture, specifically graph based reasoning systems like flow graphs, and bond graphs. In addition, we would like to implement stochastic approaches like Kalman filters and particle filters to handle the uncertainties caused by modeling approximations, sensor noise, unknown inputs etc. We will also be working to improve the constraint propagation strategies for more efficient conflict generation. Constraint retraction and re-support methods based on explanations could lead to faster and efficient constraint propagation. We are also working Jet Propulsion Laboratory to develop a fast candidate generation algorithm based on Integer Programming [9].

References

1. Hamscher, W., L. Console, and J. De Kleer, *Readings in model-based diagnosis*. 1992, San Mateo, CA: Morgan Kaufmann Publishers. ix, 520.
2. B. C. Williams, P. P. Nayak A Model-based Approach to Reactive Self-Configuring Systems, AAAI 1996, pp 971-978.
3. J. Kurien, P. P. Nayak Back to the Future with Consistency-based Trajectory Tracking, AAAI/IAAI 2000, pp370-377.
4. P. Robinson, M. Shirley, D. Fletcher, R. Alena, D. Duncavage, C. Lee, "Applying Model-Based Reasoning to the FDIR of the Command & Data Handling Subsystem of the International Space Station," Proceedings of the 7th International Symposium on Artificial Intelligence, Robotics and Automation in Space (ISAIRAS03) May 19–23, 2003.
5. Brian C. Williams, and Robert Ragno, "Conflict-directed A* and its Role in Model-based Embedded Systems," *to appear* Special Issue on Theory and Applications of Satisfiability Testing, Journal of Discrete Applied Math.
6. Alur, R., et al., Hybrid Automata - An Algorithmic Approach to Specification and Verification of Hybrid Systems, in Lecture Notes in Computer Science: Hybrid Systems I. 1994, Springer Verlag. pp. 209-229.
7. Mosterman, P.J. and G. Biswas. Diagnosis of Continuous Valued Systems in Transient Operating Regions. IEEE Transactions on Systems, Man, and Cybernetics, 1999. **29**: pp. 554-565.
8. S. Narasimhan and G. Biswas, "Model-based Diagnosis of Hybrid Systems," Eighteenth Intl. Joint Conf. on Artificial Intelligence, Acapulco, Mexico, Aug., 2003.
9. Fijany, F. Vatan, A. Barrett, M. James, C. Williams, and R. Mackey. "A Novel Model-Based Diagnosis Engine: Theory and Applications", 2003 IEEE Aerospace Conference, 2003.

Biography

Dr. **Sriram Narasimhan** is a computer scientist with QSS group inc., and works as a contractor at the NASA Ames Research Center where he is a member of the Model Based Diagnosis and Recovery Group in the Computations Sciences division at the NASA Ames Research Center. He earned his Ph.D. from the Computer Science department at Vanderbilt University in August 2002. His doctoral dissertation was on model-based diagnosis of hybrid systems. Prior to that, he received his B.E in Computer Science from Birla Institute of Technology & Science in 1995, his M.Sc. in Economics from Birla Institute of Technology & Science in 1995, and his M.S in Computer Science from Vanderbilt University in 1998. Dr. Narasimhan's main research interest is in model-based diagnosis, especially for hybrid systems. At the NASA Ames Research Center, he is the principal investigator for the research effort to develop Livingstone 3, a model-based diagnosis engine for hybrid systems that is a successor to Livingstone. During his doctoral work, he was funded by Boeing to work on fault adaptive control of f15 aircraft. He spent three summers at Xerox PARC developing a model-based diagnosis engine for copier machines. His other research interests are in active diagnosis, diagnosability, measurement selection, and fault adaptive control.



Lee Brownston has been a Computer Scientist V with QSS Group, Inc., since June 2001. He has maintained Livingstone 2 and supported Livingstone 2 users, and has participated in the design and implementation of Livingstone 3. Prior experience includes World-Wide Web development with Blue Pumpkin, Inc. (2000-2001) and EProNet (1998-2001); artificial intelligence programming with Stanford's Knowledge Systems Laboratory (1990-1998) and FMC Corporate Engineering Center (1986-1990) and Carnegie-Mellon University (1982-1986). Prior to that, he was an Assistant Professor in the Department of Psychology, Florida International University (1982-1986). He holds a B.A. in Psychology (University of Michigan, 1971), Ph.D. in Psychology (University of Minnesota, 1977), and a B.S. (1980) and M.S. (1982) in Computer Science (Florida International University).

Daniel Burrows is a Master's student in the Computer Science and Engineering Department at Penn State University. His research interests are in Software Engineering and Constraint Programming. He was a summer intern at the NASA Ames Research Center from June-August 2003 at which time he was involved in the development of the Livingstone 3 model-based diagnosis engine.