# LESSONS LEARNED FROM USING A LIVINGSTONE MODEL TO DIAGNOSE A MAIN PROPULSION SYSTEM

Adam Sweet
QSS Group Inc., at NASA Ames
MS 269-1, Moffett Field, CA 94035

Anupa Bajwa
USRA-RIACS at NASA Ames
MS 269-4, Moffett Field, CA 94035

## ABSTRACT

NASA researchers have demonstrated that qualitative, model-based reasoning can be used for fault detection in a Main Propulsion System (MPS), a complex, continuous system. At the heart of this diagnostic system is Livingstone, a discrete, propositional logic-based inference engine. Livingstone comprises a language for specifying a discrete model of the system and a set of algorithms that use the model to track the system's state. Livingstone uses the model to test assumptions about the state of a component – observations from the system are compared with values predicted by the model.

The intent of this paper is to summarize some advantages of Livingstone seen through our modeling experience: for instance, flexibility in modeling, speed and maturity. We also describe some shortcomings we perceived in the implementation of Livingstone, such as modeling continuous dynamics and handling of transients. We list some upcoming enhancements to the next version of Livingstone that may resolve some of the current limitations.

## PURPOSE

This paper is intended to give an overview of some strengths and weaknesses of the model-based diagnosis tool Livingstone 2 (L2). The authors have seen sufficient interest in L2 to make this overview worthwhile. This paper also lists mitigations of L2's weaknesses that have been used on the applications project PITEX, and what weaknesses will be addressed in the next version of Livingstone, L3. This paper is not intended as a general survey of the model-based diagnosis field; different model-based techniques each have their own strengths and weaknesses.

## INTRODUCTION

Livingstone is an inference engine that has been developed at NASA's Ames Research Center. It accepts a model of the system, such as a spacecraft, and notes the commands to and the observations from the system. Using these, it can monitor the system and diagnose its current state. It does this by comparing behavior predicted from the model with actual observations during operation. This approach is called model-based diagnosis.

Research in model-based diagnosis progressed throughout the 1980s. Several excellent papers are in [1]. The precursors to Livingstone are Sherlock and General Diagnostic Engine (GDE), which used propositional logic and conflict-directed search. However, they lacked the concept of state of the system. The idea of state, and constraints that could identify the current and next state, was added in Livingstone.

Livingstone has been implemented for diagnosis on several systems, such as the liquid propulsion feed system for the X-34 [7], the electro-mechanical actuators on the X-37 [6], hybrid combustion rocket engine testbed, advanced life support system testbed, ships' cooling system, as well as remote spacecraft and low earth orbit satellites. In its most famous application, Livingstone formed the Mode Identification and Recovery component of the Remote Agent Experiment that flew on Deep Space 1 in 1999 [3]. It monitored the Attitude Control System of DS-1. That version of Livingstone, written in Lisp, was L1. A newer version of Livingstone, in

C++, is called L2, and included a new capability of tracking multiple time histories of possible past trajectories of a system.

An L2 model includes discrete modes and mode transitions, discrete variables, and constraints between the variables. Internally, L2 uses qualitative propositional theory to represent the model using a conjunctive normal form, i.e. a set of clauses. A transition system tracks the component modes over time. L2 checks for consistency (tries to make all clauses true to make theory consistent) using boolean constraint propagation. If the theory is not consistent with nominal transitions, L2 searches the space of fault transitions to make it consistent again. The internal algorithms used in L2 are more fully described in [5].

Propulsion IVHM Technology Experiment (PITEX) [7] uses an L2 model of an X-vehicle's Main Propulsion System (MPS) to perform near real-time diagnosis. The model is compositional, i.e., the nominal and failure modes of the individual components are modeled and a complete representation of the system emerges when the components are connected [8]. This model-based approach is in contrast to a rule-based approach, where expected system behavior is predefined by a set of rules.

The overall approach behind L2 is to use a discrete system model in order to achieve a computationally fast diagnosis. Many aspects related to working with real-world systems must be addressed outside of the engine. The trade-off for this modeling approach is the additional development effort required to adapt L2 to continuous, real-world systems. In addition to the MPS model and the L2 inference engine, several other key modules, such as monitors and a Real-Time Interface (RTI), make up the PITEX architecture as described in [7]. Briefly, the monitors convert real-valued, noisy sensor data into discrete values. The RTI is responsible for passing the monitor output into the L2 engine, and for requesting L2 to perform diagnoses.

Our experience in modeling the MPS using L2 is the basis of this paper. We note several shortcomings, both in the modeling approach and in the inference engine itself. These findings will help guide the design of the next version of Livingstone, L3.

## RESULTS AND DISCUSSION

PITEX identified several strengths and weaknesses of L2. This may not be an exhaustive list, but covers many issues a new user of L2 might encounter.

### STRENGTHS OF L2

1. Tracking of system state with history – L2 can be used in monitoring execution of commands in nominal and fault cases
2. Fault detection, isolation, and recovery – can detect faults and isolate them, where possible, to the component level, then recommend recovery actions to be taken
3. Multiple faults – can diagnose double/triple faults, limited only by user-defined settings
4. Sensor fusion, unknown sensor values – current state estimate incorporates all available sensor data from multiple components, and can reason when observations are "unknown"
5. Speed – since Livingstone uses discrete representation, it is generally faster than continuous-valued methods; one can set the search parameters to better tune the processing time
6. Automated methods – L2's model representation allows for various automated tests to be performed

7. Maturity – has been used on several applications projects and runs in flight-like environments

Tracking of system state with history

A basic capability of Livingstone is the ability to track the state of a system. L2 tracks a system's state, whether it be nominal operations or fault modes. The modes and variable values within the model are all inferred based on the sequence of commands and the observations. The commands and observations are stored in a history, which allows L2 to revise its estimate of the state of a system based on later observations. This makes L2 useful as a monitoring system even when no faults are present. In the presence of faults it can diagnose which fault is likely to have occurred.

Often engineers are not so much concerned with diagnosing failures as with ensuring that the system behaves as desired. However, sometimes systems do not behave as desired, even though no hard faults have occurred. Consider the case where a spacecraft component spends most of its time in a standby mode, and must be powered on before use. If the component is commanded before it is powered on, it won't respond, although no on-board fault has occurred. L2 will track the system's state as having remained in the standby mode, and will not signal a fault. In other words, L2 tracks the effects of commands, it does not verify if a command sequence itself is correct.

Fault detection, isolation, and recovery

Some confusion can exist about the exact capabilities of a tool called a "diagnosis system". These more specific terms are commonly used to resolve that confusion. L2 has all three capabilities: it can detect that a fault has occurred, can isolate that fault to specific components, and can recommend recovery actions to be taken, if any exist (such as listing the commands needed to switch to a backup system).

Multiple faults

Many diagnosis systems make a single-fault assumption to decrease the amount of computation needed for a diagnosis. Often, this is a reasonable assumption. However, L2 does not need this assumption. L2 is capable of returning multiple hypotheses, each possibly containing multiple faults, to explain the current set of observations. The hypotheses are ordered by likelihood; limits on the number returned and on the likelihood can be set as L2 parameters.

Sensor fusion, unknown sensor values

L2 performs a task similar to sensor fusion. The sensor observations across the system are compared for consistency with the overall mode of the system. If there is an inconsistency, L2 searches for sets of faults that explain all of the observations.

Related to sensor fusion is the robustness of a system to stale or unavailable sensor values. L2 easily handles unavailable sensor values: it will take into account all available observations and return the best diagnosis. If a sensor value is stale, that value would need to be removed from L2 and thus treated as an unknown value.

Speed

This strength of L2 is relative to model-based systems with continuous dynamics. L2 is fast enough to be deployable on real-world systems: as was shown on PITEX, when running the MPS model (containing 59 components) on flight-like processors, L2 returned a fault diagnosis on the order of 1 second. When the processor was restricted to give only 5% use to L2, most diagnoses were still returned in less than 10 seconds. [7]

Two factors go into the speed of a model-based diagnosis system, candidate generation and candidate testing. The former refers to the method chosen to produce possible states of the system. The latter refers to how these possible states are evaluated to determine if they are in fact consistent with the current observations from the system.

For candidate generation, the algorithm in L2 incorporated a novel search heuristic, called conflict-directed search. This heuristic uses the conflicts between expected observations and the current observations to guide the search for possible faults. In a sense, L2 conducts the fault search based on symptoms that something has gone wrong.

As L2 models are discrete, candidate testing is in general faster than systems based on real-valued numbers. Solving a system of constraints over discrete-valued variables can be done faster than solving (or numerically integrating) complex differential equations. Of course, the speed of candidate testing greatly depends on the size and complexity of the model. As an L2 model size increases, so does the time required for candidate testing.

L3 will keep the conflict-directed search heuristic, and thus will maintain L2's advantage in candidate generation. The speed of candidate testing will depend greatly on the complexity of the model. L3 is not expected to be faster than L2, but specific comments on the speed of L3 cannot be made until it has been used on applications projects.

### Automated methods

The representation of the diagnosis information in a model-based form can be used by automated methods in different ways. Two different automated tools have been developed for verifying L2 models. First is Livingstone Model Verifier (LMV), which is derived from the Symbolic Model Verifier, SMV. LMV can take an L2 model and check it for user-defined diagnosability properties, as well as sanity checks [9]. Second is Livingstone Pathfinder (LPF), which runs L2 both as a simulator and as a diagnosis engine, executing many auto-generated scenarios and checking for various error conditions. For more information, see [10].

### Maturity

Potential users are always interested in the maturity of a software system. As described in the introduction, L2 has been used on several real-world applications projects. This has led to many bug fixes, as well as meeting many software flight requirements and compiling on several operating systems (including VxWorks). L2 also has several support tools built around it, including the graphical model development tools Stanley and Oliver (http://ic.arc.nasa.gov/projects/mba/projects/L2/doc/index.html). L2 has never been used as a critical software system in a mission, but it has been advanced to the "flight experiment" level.

## WEAKNESSES OF L2

The weaknesses of L2 fall into two general categories: those resulting from the discrete modeling formalism of L2, and those resulting from particular design choices of the L2 engine.

Weaknesses of L2's discrete modeling formalism:

1. Continuous dynamics – must be abstracted into a discrete model, which is often difficult
2. Transient periods – delays between the issuance of a command and the response of the system

3. Generic components – not always possible to define components that can be used in more then one model
4. Qualitative arithmetic – can be defined for L2 models but requires special consideration on the part of the modeler
5. Gradual degradation – difficult to model

Weaknesses of the L2 engine implementation:

6. Diagnosis process not apparent to user – the logic supporting a diagnosis is not presented well
7. Closed loops – situations that create casual loops in the model constraints, including physical loops in the system or control loops
8. Autonomous nominal transitions – L2 depends on knowing the command sequence

Some of these weaknesses were addressed in PITEX and more will be addressed by the next version of Livingstone, L3. These are all described in detail below.

## Continuous dynamics

The modeling challenge in L2 lies in creating a discrete abstraction of a system useful for diagnosis. L2's discrete modeling formalism, chosen for speed, makes it difficult to model continuous dynamics directly. All variables have discrete values, and the assigned values are logical functions of the system mode. Even time is treated as an integer value. L2 time is incremented when a system command is received – that is, it is an event-driven system. One increment of L2 time can correspond to any amount of real-world time. In such a framework, one cannot express any relationship that depends on real-world time, such as continuous differential equations.

The difficulty in making the abstraction to a discrete model depends on the system. The MPS tanks are capacitive elements, governed by differential equations: the amount of propellant in the tank changes according to the balance of flow into the tank. PITEX formed the discrete abstraction by defining the state of the tank using the derivative of the pressure in the tank, and not the value of the pressure. The pressure derivative indicates if flow is coming into or out of the tank. This modeling abstraction is used to diagnose the modes of the input and output valves of the tank.

The primary motivation for L3 is to address these issues. As well as discrete values, an L3 model will admit real-valued numbers, intervals, and ordinary differential equations. This will allow many more situations to be explicitly modeled, in a way that is more natural to engineers.

## Transient periods

As an L2 model cannot be a function of time, it is difficult to model periods in which a system is transitioning between modes. Of course, real-world systems all take time to transition. During this time, the system observations may or may not have been updated. If a diagnosis is requested before the system has settled, L2 may wrongly indicate faults in the system. The traditional approach to this issue is to wait for a specified period of time to let the transition complete before requesting a diagnosis from the system.

PITEX addressed this issue in the Real-Time Interface (RTI) module. This module is responsible for sending the commands and observations to L2, and requesting a diagnosis. Originally, the RTI followed the traditional approach: it would not allow a diagnosis to be done after receiving any command until after a specified amount of time has elapsed. However, on PITEX there were many cases where a second command would be sent within the settling time of the first command. In some cases, commands were sent rapidly enough that the scenario would be over before L2 could perform a diagnosis! The RTI was developed to better handle these cases. The final version contains a table associating commands to the corresponding observations; this is used to determine which observations will be transitioning after we issue a particular command. To do a diagnosis within the settling time of a command, the RTI removes the observations associated with that command from L2, and reassigns them back to L2 after the settling time is over. This allows L2 to diagnose other parts of the system while one part is transitioning.

It is possible to define transient modes inside of an L2 model. This was investigated during the PITEX expansion period. With this approach, the model contains explicit transitional modes. These transitional modes contain few or no constraints. The RTI no longer keeps information relating the commands and observations: it passes on all observations and only tracks the settling time of each command. In effect, instead of making a more complex RTI which unassigns and re-assigns observations, we make a more complex model that suspends constraints in a transient period and reinstates them after it is over. The advantage of this approach is that the relationships between commands and observations are only specified in the model, and not repeated in the RTI. This makes the RTI less model-specific, and hence more reusable. The initial investigation of this method on PITEX was promising, and we will explore it in future work.

Again, L3 will address this issue. Transient modes can be created if desired, and differential equations can be used in models for the transient modes. All information related to the transient, including the settling time, will be contained within the model.

## Generic components

A common goal in model-based methods is to define a set of components that could be reused in many different models. L2 models are capable of meeting this goal. However, they often encode system-specific behavior in order to achieve a better fault isolation ability. This is once again due to the discrete model representation of L2: the variable domains are defined in terms of the system characteristics relevant to diagnosis.

PITEX was no exception; the best example occurred in the L2 model of the pneumatics line, pictured below in Figure 1.

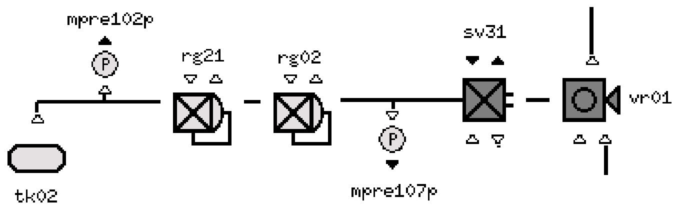

**Figure 1 L2 model fragment of the X-34 feedsystem pneumatics line**

This model fragment contains two pressure regulators in series, with no observations (pressure sensors) between them. Rg21 is the primary regulator, and rg02 is a backup regulator with a higher setpoint. Therefore, by design, if the continuous value from the pressure sensor mpre107p corresponds to rg02's setpoint, one can tell that rg21 specifically has failed. Mapping the sensor readings into the traditional discrete values {low, nominal, high} cannot fully isolate a fault to the rg21 component. For that reason, the pneumatics pressure variable, used throughout this line, was chosen to contain two ranges, one corresponding to rg21 and one corresponding to rg02. The regulator models contain constraints about these ranges, and will implicate rg21 as failed if mpre107p reads in the rg02 range. Hence, this regulator model is specific to the X-34 system, and cannot be used in another model.

Again as part of the PITEX extension work, we investigated how to make Livingstone component models more generic, without sacrificing diagnostic resolution. The L2 modeling language had recently been updated to add a feature similar to inheritance in C++ and Java. It allows a user to specify a base component with variables and constraints, and then extend that component and redefine the variables and constraints to be of specific types. In this manner, we were able to create generic base regulator models, and then extend them to create a PITEX regulator model for the model fragment in Figure 1. The resulting model still uses X-34 information, but the constraints specific to the X-34 subsystem are separated from the constraints common to all pressure regulators. More importantly, there is no loss of diagnostic resolution.

This weakness in the L2 modeling representation will again be addressed by L3. Real-valued numbers will make it less necessary to define special-purpose variables. The extent to which this is true remains to be seen: there may always be situations that require special-purpose components.

Qualitative arithmetic

The use of arithmetic is obviously desirable in many modeling situations. One common example is modeling the balance of flow through junctions and branches. Another is modeling leaks. There are also many real systems that contain competing effects. A diagnosis model must distinguish which effect will dominate, which is difficult to do without arithmetic.

This can be handled in L2, usually by defining a qualitative arithmetic table. Table 1 shows one possible addition table for a discrete variable that can take the values {low, medium, high}.

**Table 1 Example of qualitative arithmetic table**

| **(+)** | **Low** | **Medium** | **High** |
|---|---|---|---|
| **Low** | Low | Medium | High |
| **Medium** | Medium | Medium | High |
| **High** | High | High | High |

Given such a table, L2 will easily solve constraints involving the addition. However, the table contains many ambiguous elements. Should "medium" plus "medium" equal "medium" or "high"? Should "low" plus "medium" equal "medium" or "high"? Generating the table requires a modeler to make these decisions. Because of these concerns, leaks, junctions, and other situations requiring arithmetic are often avoided in L2 models.

L3 will directly address this issue. A real-valued constraint system automatically includes arithmetic, and the situations described above can be modeled easily.

Gradual degradation

Degradation is an active area of research, often referred to as "parameter estimation". Obviously, it is desirable to detect potential faults as early as possible, and if a component seems to be degrading, mitigating steps can be taken before the actual fault. However, L2 currently does not handle these issues. As it is concerned with transitions between discrete modes, usually a degraded component will be treated as nominal until it passes some threshold, then will be diagnosed as failed.

Given a monitor capable of detecting degradation, the resulting information about degraded state could be merged into L2. The information could be treated as specialized observations that would put the system into special fault modes. PITEX had developed a valve timer monitor, to measure the time response of the simulated valves. If the time required to open and close the valve begins to increase, it is often an indication of incipient failure. However, due to time limitations, PITEX did not incorporate that information into the L2 models.

### Diagnosis process not apparent to user

This is a weakness of the L2 engine. In model-based systems, the diagnosis process is not defined in advance. However, they are able to provide information on the diagnosis process as it progresses. L2 does contain some features to query the chain of reasoning leading to particular variable assignments, but they were created to support the developers of L2 rather than to provide insight to the users.

L3 plans to address this concern by adding additional support for tracing the diagnosis process. The complete path from observations to the resulting diagnosis will be available to a user.

### Closed loops

Closed loops, in this context, refer to sets of constraints with causal loops. These can be created in several modeling situations: general pipe networks will often contain loops, and controllers create feedback loops.

Internally, L2 uses a unit propagation algorithm to determine if the current modes and observations are consistent. However, unit propagation is not complete: it will not always be able to determine values for variables, even when enough information exists to do so. Having causal loops in a model is a common case that can cause this. A complete propagation algorithm, such as Davis-Putnam-Logeman-Loveland (DPLL), is always able to determine values for variables, if it is possible to do so. The first physical systems modeled in Livingstone did not contain loops, and unit propagation was chosen for simplicity [3].

A complete propagation algorithm could be implemented into L2 to overcome this weakness; however, there are no current plans to do so. L3 does plan to implement DPLL, or its equivalent, and will overcome this weakness.

### Autonomous nominal transitions

L2 makes the assumption that all nominal mode transitions will be the result of a visible system command. However, this is not true for all systems. A mechanical vent relief valve, which automatically opens when the pressure is above a threshold, falls into this category. This assumption was made on the basis of speed and simplicity. With this assumption, L2 only needs to check a single transition against the current observations to infer that the system has executed a command nominally. Without this assumption, L2 would need to check all allowable nominal transitions against the current observations. This would result in greater computational requirements of L2, even when no faults have occurred. It also admits the possibility of having multiple system states that would need to be tracked in the nominal case.

The MPS model contained a solenoid valve whose commands were not in the telemetry available to L2. This situation was handled by not giving the valve separate "open" and "closed" modes, but one "nominal" mode where the valve position is allowed to be open or closed. L2 can infer the value of the valve position variable, but will not be able to diagnose specific faults, such as "stuck open", in the valve.

The modeling formalism chosen for L3 will explicitly allow hidden nominal transitions. The guards on the transitions will help avoid the expansion of possible transitions and additional computation time.

## SUMMARY AND CONCLUSIONS

L2 is a good candidate for providing system monitoring and diagnosis. However, PITEX did find several weaknesses of L2, both in the discrete modeling representation and in the L2 implementation. Potential users should evaluate these strengths and weaknesses according to the needs of their particular application.

In order to address the current limitations of modeling in L2, researchers at NASA Ames are working on both the modeling methods and on the capabilities of the inference engine. For instance, we are exploring the use of generic component models, to promote model reuse. Researchers are also working on designing L3, which can use a richer language to express the model, ranging from interval constraints to differential equations, as well as many improvements to the core engine.

## ACKNOWLEDGMENTS

Thanks to Bill Millar and the other PITEX team members, for their discussions which led to the creation of this paper. Also thanks to Drs. Sriram Narasimhan and Mark Shirley for reviewing this paper and providing their insights and comments.

## REFERENCES

1. ***Readings in Model-Based Diagnosis***, Eds. Hamscher W., Console L., and de Kleer, J. (1992).

2. Williams, B. C., and Nayak, P. P., ***A model-based approach to reactive self-configuring systems***, Proceedings, AAAI-96 (1996)

3. Muscettola, N., Nayak, P. P., Pell, B., and Williams, B. C., ***Remote Agent: To boldly go where no AI system has gone before***, Artificial Intelligence 103:5-47 (1998)

4. Bernard ,D., Dorais , G., et al, ***Spacecraft Autonomy Flight Experience: The DS1 Remote Agent Experiment***, AIAA 99-4512, (1999)

5. Kurien, J., and Nayak, P. P., ***Back to the future for consistency-based trajectory tracking***, 7th National Conference on Artificial Intelligence (AAAI) (2000).

6. Schwabacher, M., Samuels, J., and Brownston, L., ***NASA Integrated Vehicle Health Management Technology Experiment for X-37***, SPIE AeroSense (2002)

7. Meyer C., Cannon H., et al, ***Propulsion IVHM Technology Experiment Overview***, IEEE Aerospace Conference, Big Sky, MT (2003)

8. Bajwa, A. and Sweet, A. ***The Livingstone Model of a Main Propulsion System***, IEEE Aerospace Conference, Big Sky, MT. (2003).

9. Cimatti, A., Pecheur, C., Cavada, R., ***Formal Verification of Diagnosability via Symbolic Model Checking***, IJCAI'03, Acapulco, Mexico (2003)

10. Lindsey, A. E., and Pecheur, C., ***Simulation-Based Verification of Livingstone Applications*****,** Proceedings of Workshop on Model-Checking for Dependable Software-Intensive Systems (DSN 2003), San Francisco, California (June 2003)