Real-time monitoring of ECLSS flight rules

Scott Bell and David Kortenkamp
TRACLabs Inc.
1012 Hercules
Houston Texas 77058
korten@traclabs.com

Jack Zaientz
Soar Technology Inc.
3600 Green Court, Suite 600
Ann Arbor, MI 48105

Flight rules are the overarching constraints that govern the operation of equipment on the International Space Station (ISS) and the Space Shuttle. There are thousands of flight rules for ISS, all documented in Microsoft Word. Monitoring of flight rules is currently done manually by flight controllers who are highly trained in understanding and interpreting flight rules. In this paper we describe a system that allows flight rules to be expressed in a computer-understandable language and that then generates an automated monitoring system that connects to the ISS telemetry stream. For testing purposes we represented and monitored Environmental Control and Life Support Systems (ECLSS) flight rules for ISS smoke detectors. We describe how this flight rule is represented in an eXtensible Markup Language (XML) format that captures relationships between different pieces of raw telemetry. We describe a drag-and-drop editing environment that allows for easy authoring and debugging of the XML. We then describe how this XML file is translated into a monitoring capability that automatically executes the appropriate functions to validate or invalidate the flight rule. Finally, we show examples from actual ISS telemetry of the real-time monitoring of this specific flight rule. A flight controller interface has been designed using the next generation of mission control tools to allow easy configuration, manipulation and supervision of flight rule monitoring. This flight controller interface will work in concert with the editor and execution engine to enhance the safety of space missions by automatically detecting flight rule violations.

I. Introduction

Modern space systems such as satellites, human spacecraft, planetary probes and space robots are highly sensored and generate large amounts of data. For this data to be useful to humans monitoring these systems and to automated algorithms controlling these systems it will need to be converted into more abstract data. This abstracted data will reflect the trends, states, and characteristics of the systems and their environments. Currently this data abstraction process is manual, *ad hoc*, and intermingled with control systems. It is manual in the sense that either humans do the abstraction in their heads or the data abstraction is done by hand-coding computer programs for each data item. It is *ad hoc* in the sense that each data abstraction is developed on its own with no representation of how it relates to the tasks being performed or to other data abstractions. It is intermingled with the control systems in that data abstractions are irreducible and difficult for other programs, like displays and analysis tools, to access. In this paper we describe a Data Abstraction Architecture (DAA) that allows engineers to design software processes that iteratively convert spacecraft data into higher and higher levels of abstraction. The DAA also formalizes the relationships between data and control and the relationships between the data themselves. We apply this Data Abstraction Architecture to the problem of monitoring ISS Environmental Control and Life Support Systems (ECLSS) flight rules, which are abstractions of raw telemetry.

II. Data abstraction architecture

A Data Abstraction Architecture (DAA) formalizes the data abstraction process for space systems. The DAA provides a canonical way to assemble and interact with data abstraction. Similar to control architectures (e.g.,[1,2]) a data abstraction architecture provides a tool-box of components and connections that allow engineers to build and maintain data abstraction systems. Here are some of the key components in our system:

Data abstraction architecture (DAA) A series of mathematical or logical transformations of telemetry data to provide appropriate inputs from a hardware system to a hardware system controller, system engineer, or crew.

Data elements Define the data upon which the DAA operates, including telemetry, derived data, symbols and triggers.

Data abstractors A defined transformation of data signals from one form to another, usually more abstracted or specialized, form.

Data Abstraction Reasoning Engine (DARE) Encodes the DAA in a computer program that is connected to the data stream, runs in real time and produces outputs for higher-level control systems, system engineers or crew.

Abstraction network The set of interlinked data abstraction nodes that comprise a DAA.

Sensor Event Abstraction Language (SEAL) The Sensor Event Abstraction Language (SEAL) is an XML grammar that defines the data abstractors, the abstractor's message handling operations, and the directed graph connecting the abstractors (called a Data Abstraction Network (DAN)). Users do not need to manually program SEAL files; the necessary files are generated by using a graphical editor.

Data source The generator of a continuous data signal input into the DAA. This generator may include either raw telemetry data signals generated by hardware sensors or preprocessed data signals from a low-level controller or other abstraction architectures.

Data sink The receiver of a continuous data signal output from the DAA. This receiver may include high-level control systems, crew displays, logging or maintenance systems, or other abstraction architectures.

Development environment The editor is an end-user oriented software tool to aid in the construction, debugging, and viewing of SEAL files. The editor creates a file, which is read in by the Data Abstraction Reasoning Engine (DARE) and used to output abstracted data. The abstracted data can be used by displays intended for human consumption or by higher-level controllers.

Taken together these components provide a powerful mechanism for representing and accessing the data necessary to monitor and control Constellation vehicles, habitats and robots. They also provide a mechanism for monitoring flight rules.

II.A. Data abstractors

One of the principal functions of the data abstraction architecture is to define the operations that may be performed on an input message stream. Whether performed on the message content or the message envelope, these are referred to as abstractors, for their resulting product is an abstraction of the input data that is consumed by the next abstractor in the graph. Different classes of abstractors focus on different stages in the transformation process represented by a data abstraction network.

Message Management abstractors are designed to help manage the message flow of the data bus prior to doing anything with the sensor data that is the content of those messages:

- Sampler reduces the number of messages handled to a manageable subset and are generally used at the beginning of a DAN.
- Temporal Alignment ensures that messages coming from different sensors at different rates are grouped to represent events occurring at the same time.

Data Manipulation abstractors focus on transforming the sensor data itself:

- Mathematical Functions (Ratio, Average, Arithmetic) specify math expressions to per-form on the input values

- Unit Transformation changes units of measure

Output Management abstractors focus on which results are to be included in the output, and how they will appear:

- Categorical Binner groups numeric values into symbolic categories (e.g., “low, “med, “high) and display only the symbolic value.
- Trim drops values that lie outside specified ranges.

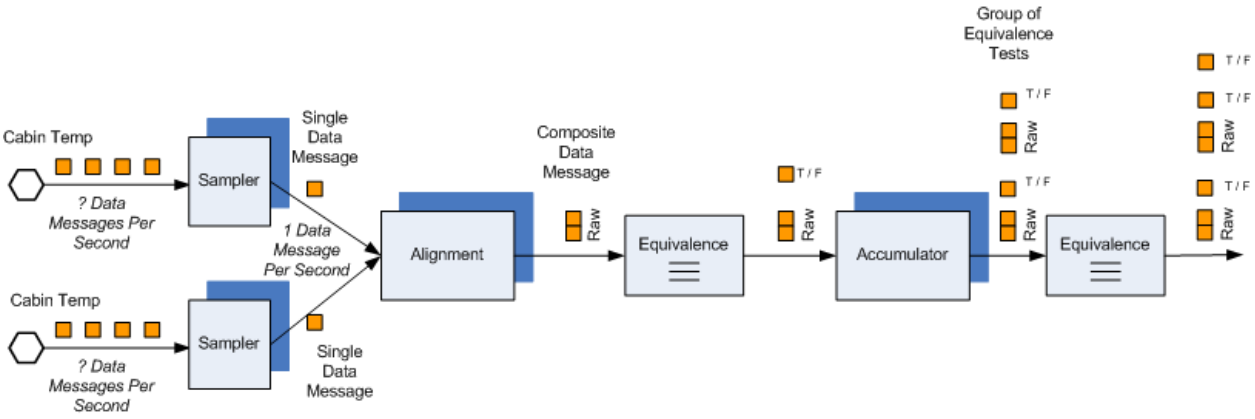

Figure 1. An example SEAL abstraction architecture describing a quiescence filter.

II.B. Sensor event abstraction language

The Sensor Event Abstraction Language (SEAL) is an XML grammar that defines data manipulation and message handling operators, enabling the description of sophisticated transformations on event-based telemetry data. Basically, SEAL defines the flow of data amongst the data abstractors. The SEAL syntax and semantics are intended to support the computational requirements of NASA telemetry and telemetry management processes and align to the conceptual model of those processes held by expert NASA flight control engineers. Finally, the language is intended to support rapid visual development and inspection of data transformation by skilled engineers who are typically trained in disciplines other than software engineering.

A simple example of a data abstraction written in SEAL is a quiescence filter. This filter only passes a value out the far side if that value has remained within tolerances for a prescribed time interval. Figure 1 shows one possible implementation of a Quiescence Filter. First an alignment operator (makes sure both values were acquired at the same time) and equivalence operator (compares two values to determine if they are within a specified tolerance of each other) pair gathers a set of messages together (from different data sources) and evaluates them to see if they are within tolerance. Second, the message, which contains all the original messages and the output of the equivalence test, is passed to another accumulator operator and equivalence operator pair. This pair compares whether the group that is within tolerance has remained in tolerance for the prescribed amount of time. Note that this example makes strategic use of both message-envelope operators (temporal alignment and accumulation) and a simple message-data operator (equivalence) to instantiate the more complex notion of quiescence.

III. Monitoring ISS ECLSS flight rules

Flight rules govern the operation of space vehicles. Flight rules are currently written in Word and are not monitored by software systems. We converted an existing flight rule into a data abstraction network. This flight rule governs the operation of the smoke detectors on ISS. The flight rule determines when a smoke detector is ‘dirty’ and must be serviced. It does this by looking at specific telemetry coming from that smoke detector, performing calculations on that telemetry and comparing those calculation to pre-defined limits. We manually translated this flight rule into the SEAL language. A visual representation of this SEAL file, including all of the abstractors, is shown in Figure 2.

Starting in the upper left, the hexagons are two sensor inputs from ISS – the scatter voltage and the obscuration voltage. The other hexagon is a constant determined experimentally and referenced in the flight rule. An arithmetic

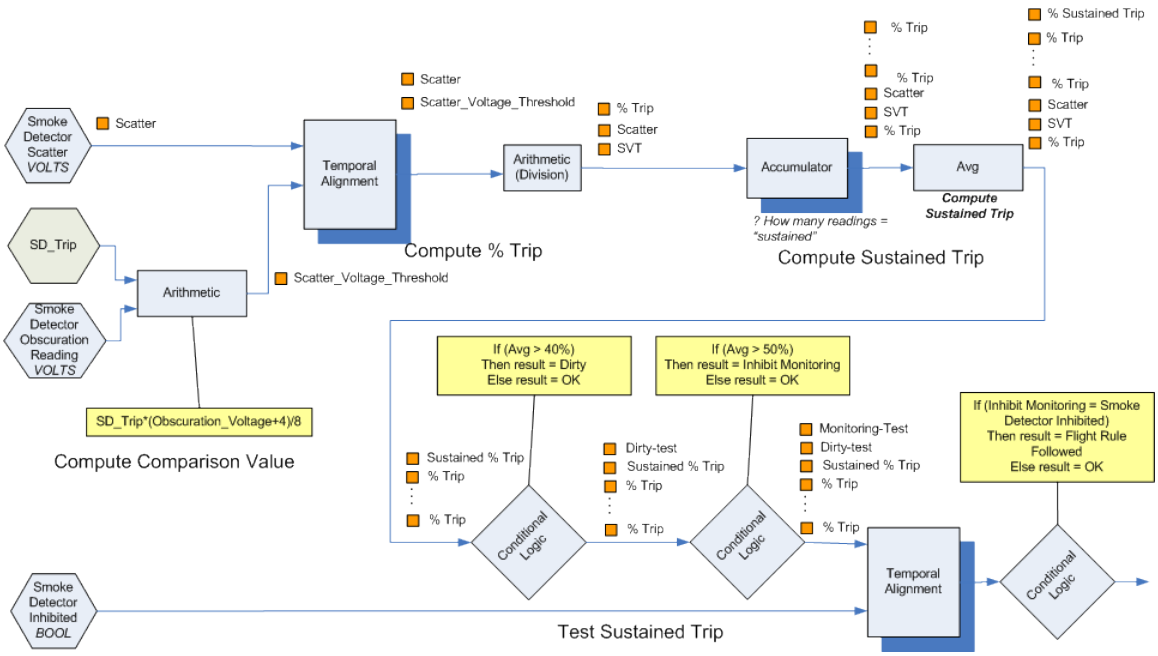

Figure 2. A data abstraction network for the smoke detector flight rule.

abstractor computes a new value from the obscuration voltage and the constant. Next, a temporal alignment abstractor makes sure that we are comparing two values that were obtained at the same time. This prevents stale values from being compared to new values. This abstractor outputs two values that are temporally consistent. Another arithmetic data abstractor computes the percent tripped for that smoke detector. An accumulator abstractor gathers up those readings over time (the "sustained" part of the flight rule) and passes all of those to an average abstractor. If the limit is between 40% and 50% the smoke detector is dirty and must be serviced. If the average is greater than 50% the smoke detector must be inhibited (turned off). Finally, telemetry that states whether or not the smoke detector is inhibited is compared to the result of the computation to determine whether the smoke detector should be inhibited. If they agree, then the flight rule is being followed. If not, then the flight rule is being violated.

III.A. Flight controller interaction with flight rules

The Mission Control Technology (MCT) project is developing new display software for NASA Mission Control Center (MCC).[3] We developed a custom MCT component that connects with DARE to retrieve information. We also created a view of that component so that users can inspect the data abstraction network and see the abstracted data. The view allows a user to see the entire data abstraction network in graphical form (see Figure 3). It also allows the user to inspect the values, units, etc. of any part of the data abstraction network as well as change data abstractor parameters.

III.B. Connecting to ISS data

We tested our data abstraction network against actual ISS smoke detector telemetry. We did this by connecting DARE to the Information Sharing Protocol (ISP) network that publishes ISS telemetry. We accessed this network through a Virtual Private Network (VPN) account with NASA JSC. The data abstraction network was successfully able to monitor a specific smoke detector on ISS. ISP also has the capability to playback a hand-built file of telemetry values. We used this to test the data abstraction network against data that violated the flight rule (which the actual ISS data never did). This test confirmed that the data abstraction network could detect when the flight rule was being violated. This proof-of-concept demonstration was done using the MCT mission control interface. Figure 4 shows the set of processes used for this proof-of-concept demonstration. The XTCE referred to in the diagram stands for the XML Telemetric and Commanding Exchange XML schema. XTCE is an emerging standard for representing telemetry and commands for space systems. It has been adopted by NASA's Constellation program to represent Orion commands and telemetry. XTCE serves as the source of our raw telemetry information. Also, we can output an XTCE file that

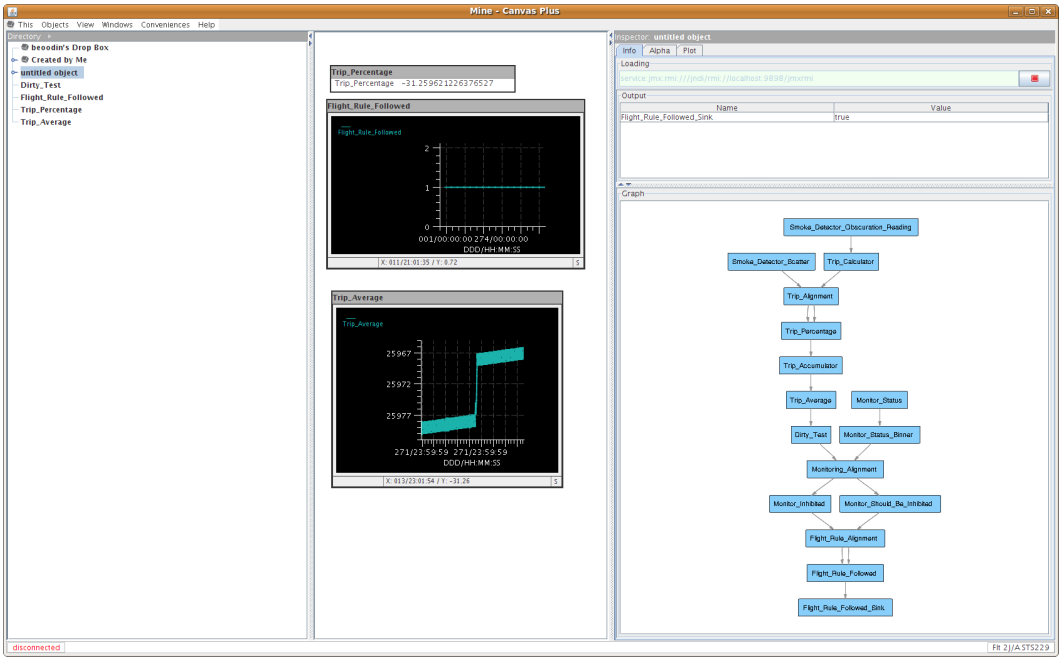

Figure 3. Screen shot of the MCT user interface for the data abstraction network.

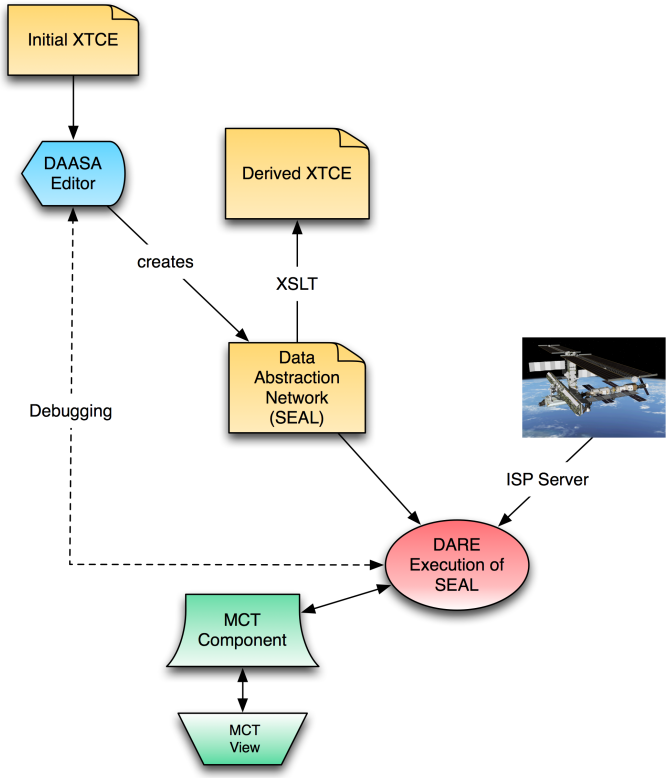

Figure 4. The processes that were used to monitor ECLSS flight rules.

contains the abstracted telemetry generated by DARE.

IV. Related work

Several autonomous control architectures had explicit data abstraction. One clear example is the Supervenience architecture.[4] The architecture consisted of communicating levels in which lower levels pass data about the world to higher levels. At the same time higher levels pass goals down to lower levels. It is implemented using a blackboard architecture at each level. Each level also contains its own uniform data representation. Several agent-based systems have looked at the information retrieval and integration problem (see[5] and[6]). Some early examples of agent systems for information retrieval and coordination include COLLAGEN,[7] Infomaster[8]and work by Jennings[9] and Lesser.[10] Another example would be the Mobile Agents work of Clancey and Sierhuis, especially with respect to robotics interaction.[11]

V. Conclusions

Automated monitoring of ECLSS flight rules will save ground controller time and effort. Calculations that must currently be performed manually can be done automatically. However, creating software to monitor flight rules is also an expensive process. The work in this paper shows how flight rules can be monitored by composing simple arithmetic and comparison operations to create a simple representation of the flight rule. The calculations in this representation can be executed by an engine that is connected to spacecraft telemetry. The outputs of this engine can be displayed to flight controllers in the same manner as other telemetry. An editor allows flight controllers to create these compositions with very little training or computer science experience. The system was tested using actual ISS telemetry and an ISS flight rule in the ECLSS domain.

VI. Acknowledgments

This work funded under NASA contract NNX08CA11C. The authors wish to thank Jeremy Frank and Jay Trimble of NASA Ames Research Center and Alan Crocker and Christie Bertels of NASA Johnson Space Center for their contributions to the ideas presented in this paper.

References

[1]Bonasso, R. P., Firby, R. J., Gat, E., Kortenkamp, D., Miller, D. P., and Slack, M., "Experiences with an Architecture for Intelligent, Reactive Agents," *Journal of Experimental and Theoretical Artificial Intelligence*, Vol. 9, No. 1, 1997.

[2]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*, Vol. 103, No. 1, 1998.

[3]Trimble, J., Walton, J., and Sadler, H., "Mission Control Technologies: A new way of designing and evolving mission systems," *AIAA Space Operations 2006*, 2006.

[4]Spector, L. and Hendler, J., "Planning and Reacting across Supervenient Levels of Representation," *International Journal of Intelligent and Cooperative Information Systems*, Vol. 1, No. 3, 1992.

[5]Haverkamp, D. S. and Gauch, S., "Intelligent Information Agents: Review and Challenges for Distributed Information Sources," *Journal of the American Society for Information Science*, Vol. 49, No. 4, 1998.

[6]Malone, T. W., Lai, K. Y., and Grant, K. R., "Agents for Information Sharing and Coordination: A History and Some Reflections," *Software Agents*, edited by J. M. Bradshaw, AAAI Press, Menlo Park CA, 1997.

[7]Rich, C. and Sidner, C. L., "COLLAGEN: A Collaboration Manager for Software Interface Agents," *User Modeling and User-Adapted Interaction*, Vol. 8, No. 3-4, 1998.

[8]Genesereth, M. R., Keller, A. M., and Duschka, O. M., "Infomaster: an information integration system," *Proceedings of the ACM SIGMOD Conference*, 1997, pp. 539–542.

[9]Jennings, N. R., "Coordination Techniques for Distributed Artificial Intelligence." *Foundations of Distributed Artificial Intelligence, Sixth-Generation Computer Technology Series,*, edited by G. M. P. O'Hare and N. R. Jennings, John Wiley and Sons, New York, 1996.

[10]Lesser, V. R., "Reflections on the Nature of Multi-Agent Coordination and Its Implications for an Agent Architecture," *Autonomous Agents and Multi-Agent Systems (AAMAS-98)*, 1998.

[11]Sierhuis, M., Bradshaw, J. M., Acquisti, A., van Hoof, R., Jeffers, R., and Uszok, A., "Human-Agent Teamwork and Adjustable Autonomy in Practice," *Proceedings of the 7th International Symposium on Artificial Intelligence, Robotics and Automation in Space: i-SAIRAS 2003*, 2003.