Virtual Engineering and Science Team - Reusable Autonomy for Spacecraft Subsystems[1]

Sidney C. Bailin
Knowledge Evolution, Inc.
1050 17th Street, Suite 520
Washington, DC 20036
(202) 467-9588 x10
sbailin@waves.kevol.com

Michael A. Johnson
NASA/Goddard Space Flight Center
Code 564
Greenbelt, MD 20771
(301) 286-3170
michael.a.johnson.2@gsfc.nasa.gov

Walt Truszkowski
NASA/Goddard Space Flight Center
Code 588
Greenbelt, MD 20771
(301) 286-8821
Walt.Truszkowski@gsfc.nasa.gov

Michael L. Rilee
NASA/GSFC / Emergent IT
Code 931
Greenbelt, MD 20771
(301) 286-4743
michael.rilee@gsfc.nasa.gov

Bryan Thompson
Global Wisdom, Inc.
1737 Harvard St., NW
Washington, DC 20009
(301) 657-0093
bryan@globalwisdom.org

Abstract—In this paper we address the design, development and evaluation of the Virtual Engineering and Science Team (VEST) tool – a revolutionary way to achieve onboard subsystem/instrument autonomy. VEST directly addresses the technology needed for advanced autonomy enablers for spacecraft subsystems. It will significantly support the efficient and cost-effective realization of on-board autonomy and contribute directly to realizing the concept of an intelligent autonomous spacecraft.

VEST will support the evolution of a subsystem/instrument model that is provably correct and from that model the automatic generation of the code needed to support the autonomous operation of what was modeled. VEST will directly support the integration of the efforts of engineers, scientists and software technologists. This integration of efforts will be a significant advancement over the way things are currently accomplished.

The model, developed through the use of VEST, will be the basis for the physical construction of the subsystem/instrument and the generated code will support its autonomous operation once in space. The close coupling between the model and the code, in the same tool environment, will help ensure that correct and reliable operational control of the subsystem/instrument is achieved.

VEST will provide a thoroughly modern interface that will allow users to easily and intuitively input subsystem/instrument requirements and visually get back the system's reaction to the correctness and compatibility of the inputs as the model evolves. User interface/interaction, logic, theorem proving, rule-based and model-based reasoning and automatic code generation are some of the basic technologies that will be brought into play in realizing VEST.

TABLE OF CONTENTS

1. INTRODUCTION

The design and fabrication of a spacecraft science instrument involves the collaboration of several developers each bringing their own experiences and viewpoints to bear on the problem of meeting mission requirements. Engineers and scientists are involved in various aspects of system specification, implementation, and operation with tasks being divided according to capability, resources, or other reasons.

Very often such instruments use cutting edge technologies whose behavior is not completely understood, in which case post deployment analysis is quite important and is shared by engineers and scientists. Most science instruments used in space are "one-of-a-kind" with lessons learned in previous instruments brought to bear in future generations as quickly as practical. In any event, various instrument components or behaviors are quite similar or identical across a range of instrumentation. For example, many instruments used in space science use high voltages for charged particle optics or signal amplification.

Thus many space science instruments must account for the requirements of high voltage system operation. These operations are often critical for the success of the science mission, yet the high voltage systems are often susceptible to a variety of failures. Many of these failures can be averted through rapid reconfiguration of the systems when faults or undesirable system states arise. Other failures are presaged by trends in health & safety data that may recommend modifications to the standing operations plan to improve system performance or longevity.

Immediately one sees two sets of interests that must be addressed: first are the requirements for instrument and spacecraft health, safety, and continued operation; and second are the science mission requirements for which the instrument was developed and deployed in the first place. Questions which make sense in the context of one viewpoint may not make sense in another. This can make negotiating instrument development or changes in operation difficult and lead to uncertainties in scientific results or suboptimal operations because there are multiple viewpoints and understandings of any particular piece of science instrumentation. By the way, no one viewpoint or understanding need be incorrect to generate misunderstanding, they may be either incomplete or not yet translated from one context to the other. Indeed, this can be a serious issue to consider because often the utility of a particular viewpoint may lie in its disregard for particular aspects that are crucial in other viewpoints.

The Virtual Engineering & Science Team (VEST) is a design tool that allows a number of expert users to construct models of systems and subsystems. These models are constructed within a framework provided by the VEST system. Multiple views of the models provide some flexibility allowing the model developer to choose a preferred method of model input and editing. VEST uses these models to construct control software for the system being designed. As the models are being constructed, their consistency is checked and inconsistencies or potential problems are automatically highlighted by the system. In this way, design or requirements problems may be flagged early in the design or development stage. The system and process models are used to automatically generate lower level (C) code that provide autonomous control functions. The generated system may then be adapted by hand as necessary during system integration. Thus with VEST we are researching a virtual environment that allows a team of researchers (or their agents) to interact with a system model in a manner of their own choosing: whether it be a graph, spreadsheet, or textual code.

2. MOTIVATION

The concepts currently underlying the Virtual Engineering and Science Team (VEST) tool started about two years ago. It was determined that the idea of using model-based reasoning as the infrastructure for supporting on-board autonomous instrument operations

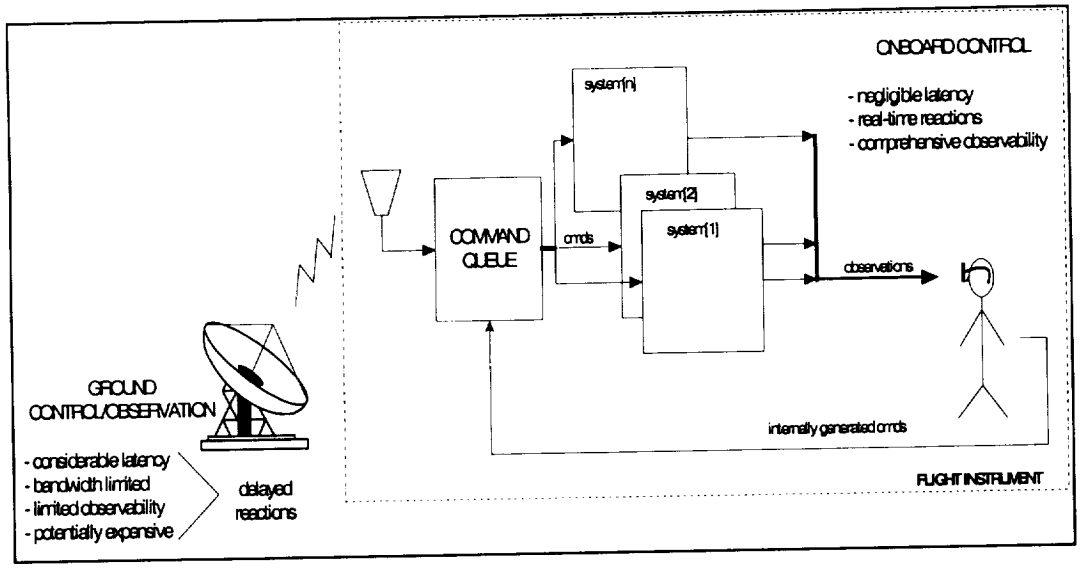


Figure 1. Ground Control vs. Onboard Control

was worth investigating. An effort to develop a prototype system using the IMAGE/Low Energy Neutral Atoms (LENA)[1] instrument was begun. The following figure illustrates the idea at a high level. The concept was to enable an on-board instrument system, augmented with a model-based reasoning capability, to monitor its operation and science agenda behaviors. Based on these inputs the system would be able to identify and correct anomalies and make science-agenda decisions based on the actual science data being collected

Figure 1 shows some of the disadvantages of strictly relying on the ground for total instrument control and the corresponding advantages of relying on on-board control. The reasoning component associated with the instrument package is referred to as the Surrogate Principal Investigator (SPI). The concept SPI is depicted in Figure 2.

development for the SPI can be seen as a process used by ground-based instrument-designers/scientists to develop provably-correct models that can be used to evaluate the behaviors of instruments. Secondly, the mode developed by this process can also serve as the knowledge-base used by an on-board "intelligent" process for the command and control of the corresponding on-board instrument. This dual feature is what is embodied in the Virtual Engineering and Science Team (VEST) tool.

3. APPROACH

Tool Concept

The goal of the VEST tool is to provide a convenient way for the instrument designer to specify the components of an instrument, the interconnections

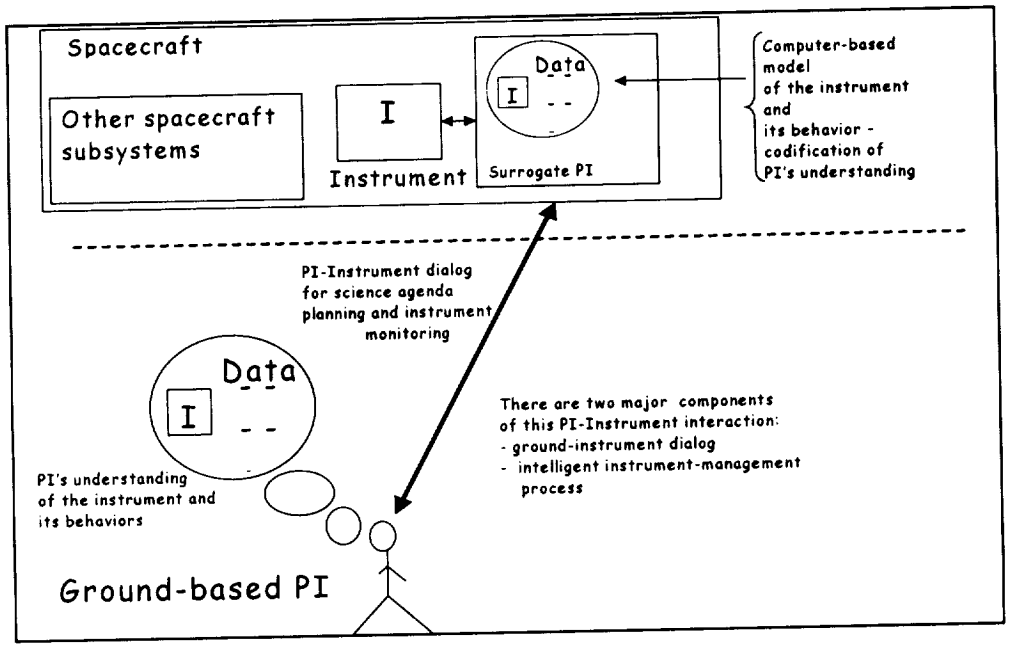


Figure 2. Gound-based PI Mental Model

This figure shows a ground-based PI with his/her knowledge of the instrument, its capabilities and science capabilities. This "mental model" is transformed into a representation that is accessible to an on-board process. This process (the SPI) is capable of using this representation of the ground-based PI's mental model as a basis for its model-based reasoning about the instrument it is controlling.

The initial attempt at developing the SPI was a success. Some of the model components were actually uploaded to the IMAGE spacecraft and utilized in the control of the LENA experiment. Reflecting on the experience gained by developing the prototype two important ideas emerged. The first is that the process of model

between components, and their properties, This is in keeping with the general ideas of Component-Based System Design in which systems are designed from building blocks that already exist or that represent significant, well-defined and well-understood bodies of knowledge. The reuse of existing components is a fundamental part of the VEST process. By importing an existing component into an instrument design, the designer at the same time imports the behavior and properties of the component. The process of reuse allows the designer to avoid "reinventing the wheel."

Component-based design is nothing new in the realm of hardware engineering; it is becoming the norm in software development as well. VEST is distinguished

from other component-based approaches in its orientation specifically to spacecraft instrument design, and its provision of behavioral semantics that go beyond a boxes-and-arrows representation of a design. Computer-assisted design (CAD) tools, for example, provide sophisticated means of drawing schematics and reusing standard parts, but they provide little or nothing in terms of correctness checking or other interpretation of the meaning of a design, i.e., its semantics. Mathematical modeling tools such as Matlab™ or Mathematica™ provide a rich store of basic mathematical semantics; they do not, however, provide built-in component and interconnection types to support instrument design. VEST seeks to combine the convenience of CAD with the semantic richness of mathematical modeling, and to do so in a way that supports the particular patterns and idioms of instrument design.

Model Representation

Analysis of LENA—Inspection of the LENA autonomy code indicated that the chief sources of complexity were in the definition of quantities derived from data directly readable from the instrument hardware, and in the formulation of conditions under which certain actions need to be taken. Examples of derived data include count-rate averages and signatures over a given period of time; complexity of conditions is essentially a matter of multiple levels of nested if-then-else statements.

These observations led us to conjecture that a modeling tool should facilitate the formulation and comprehension of complex data and conditional expressions. This conjecture has implications both for the content of a VEST instrument model and for the capabilities (both analytical and user interface) of the VEST tool.

Content of a VEST Model

In view of the user-oriented goals of VEST and our analysis of the LENA automation software, we decided that a VEST model should consist of 6 levels of information:

1. Instrument components and interconnections. The principal components of LENA are the High Voltage Power Supply (HVPS), the collimator, the Fiber Optic subsystem, and the Time of Flight (TOF) subsystem.
2. Data directly readable from each component. In LENA, this includes the voltage and current of the HVPS, the count rate measured by the TOF, the spin sector of the instrument, and clock time.
3. Derived data, such as average, peak, and standard deviation of count rates, count rate signatures, and projected count rates.
4. Conditions. These are Boolean expressions formulated in terms of both the direct and derived data. Usually, conditions represent constraints, i.e., situations that must hold in order for the instrument to function properly. Along with the logical condition that expresses the constraint, the model includes information about when the constraint is to be checked (e.g., as often as possible, at a given frequency, upon a particular type of event, or when particular conditions hold).
5. Commands. These are the operations by which the instrument can be controlled. Examples in LENA include turning the HVPS on or off, or setting its voltage.
6. Rules, which specify actions to be taken if constraints are violated, in order to re-establish proper instrument operation. A rule consists of a left-hand side, specifying a set of conditions that requires corrective action, and a right-hand side consisting of a series of actions to be taken.

Model checking

There are potentially many ways in which a model could be checked for correctness. Given the emphasis of VEST on monitoring for constraint violations and implementing corrective actions, we arrived at a particular form of model checking aimed at facilitating the instrument designer's thought processes. Model checking in VEST concerns the possibility that actions intended to correct a given anomaly might inadvertently contribute to another problem.

To do this, VEST examines the rules of the model pair-wise. For any two rules R_1 and R_2, VEST examines the action side of R_1 and analyzes whether these actions can cause the condition side of R_2 to become valid where they were not valid prior to the R_1 actions. If this is possible, then from a logical point of view either the conditions of R_1 or those of R_2 are not sufficiently circumscribed. The instrument designer might know that such a situation is impossible, but he has not articulated this knowledge within the model. VEST therefore alerts him to the issue; it is up to the designer to decide whether to refine the conditions of the one or both of the rules.

Rule Analysis Algorithm—Carrying these ideas down to a further level of precision, we came up with the following algorithm. Given two rules in the form of:

$$LHS_1 => RHS_1, LHS_2 => RHS_2$$

where each LHS is a list of conditions (together indicating an undesirable situation), and each RHS is a list of assignments (to fix the situation), do the following:

1. Time order RHS_1 to form RHS_{1a}. This is discussed below.
2. Apply the same variable replacements throughout LHS_2 to form LHS_{2a}
3. Form the expression (LHS_1 and not LHS_2) and (RHS_{1a} and LHS_{2a})
4. Try to find a solution to this expression.

The time ordering of variables, mentioned in step 1, is an approach to formalizing the temporal relationship between old values and new values of variables in a rule that consists of a set of conditions and a set of actions. We assume that the actions are assignments of expressions (formulated in terms of the variables introduced in the rule's conditions) to variables. The goal of the time ordering process is to represent the "before" and "after" states of the variables so that both types of information can be used in the rule analysis process. We do this by creating an "after" version of each variable that is assigned a value by some action in the rule's right-hand side. For example, given the RHS action

$$x = x + 2y + z$$

we rename the variable on the left-hand side of the assignment to be x_1, yielding

$$x_1 = x + 2y + z$$

so that x still represents the old value (subject to whatever conditions appear on the left-hand side of the rule), while x_1 represents the new value. Note that the occurrence(s) of x on the RHS of the assignment are left intact, since they are intended to represent the "before" value.

Having done this, we propagate the replacement of x with x_1 in all occurrences of x in all succeeding assignments in the rule's right-hand side. The idea is that once the value of x has become the "after" value, all subsequent references to x implicitly refer to that new value, so they should be replaced by x_1.

We do this with the first action on the right-hand side of the rule, then the second, the third, etc. All replacements propagate forward to the remaining assignments, not to previous ones. Once this process is complete, we replace the assignment operator "=" with the equality relation "==". The right-hand side is now a set of conditions itself, representing the state of affairs that obtains after the rule has been executed.

User Interface Requirements

A primary goal of VEST is to facilitate the formulation of an instrument model in high-level terms. In particular, this means describing the required behavior of the instrument in terms that are intrinsic to the instrument. For insight into how to do this, we drew on our experience with LENA.

Most of the LENA automation code consists of conditional assignments and/or function calls. This is amenable to being expressed in a rule notation, i.e., a list of conditions followed by list of actions. Rules simplify the representation by removing the syntactic details of program code: in particular, by flattening out complex sets of nested conditions. However, there is a cognitive tradeoff between program notation, in which conditions are nested, and rule notation in which all conditions are explicitly stated in each rule. The benefit of the rule notation is that you see the exact conditions under which an action is taken, localized (in a rule left-hand side) right above the action. The benefit of program notation is that you do not repeatedly state conditions that apply to a range of inner conditions — so you get a sense of the scope of the conditions.

Although conflicting, each of these benefits aids in comprehension of the instrument logic, and especially in verifying correctness and completeness. In the VEST tool, we aimed to strike a balance between nested and flat representations of complex conditions, in order to facilitate comprehension of the instrument model.

Another requirement is to support comprehension of the many data fields involved in an instrument. To do this, all field references should be hyperlinked to a description of the field. This requirement is based on the notion that hyper-linking is easier than looking up a field in a table of contents or index. In addition, each field declaration should be followed by a description that includes a list of references to the field, hyperlinked to the references themselves.

Similarly, assignment of one field's value (or a defined value) to another should be easily specifiable via point-and-click. This offloads the instrument designer's need to remember and type field names. However, the tool user should be able to type field names in an assignment if he wants to, since it is sometimes easier than point-and-click, especially for rapid input, and it allows greater flexibility.

4. IMPLEMENTATION

For the initial prototype, we wanted to focus on exploring the kinds of model representations and user functionality that would best meet the goals of VEST. We wanted to avoid becoming embroiled in the development of user interface capabilities that are, on the one hand, well understood and widely available, but also a potential drain of development resources. We chose, therefore, to implement the prototype as an extension to Microsoft Office™ tools, in particular Excel™ and Powerpoint™. VEST functions were implemented in Visual Basic and made available as macros through these tools.

Our choice of software platform might be surprising considering the goals of VEST; the following considerations lay behind the choice: First, the Office tools are widely used within the target VEST user community, and their user interfaces are very familiar. Second, the user interfaces have been stressed and refined over a period of time and by a large user base with which few other frameworks can compete. We viewed this as a particularly important fact because the clarity and robustness of the user interface will play a key role in selling VEST to its target users. Finally, the application program interface (API) provided by these tools provided sufficient visibility and flexibility to implement initial versions of the core VEST functionality.

Model Elements—Components and their interconnections are specified initially as a graphic using Powerpoint, as shown in Figure 3.

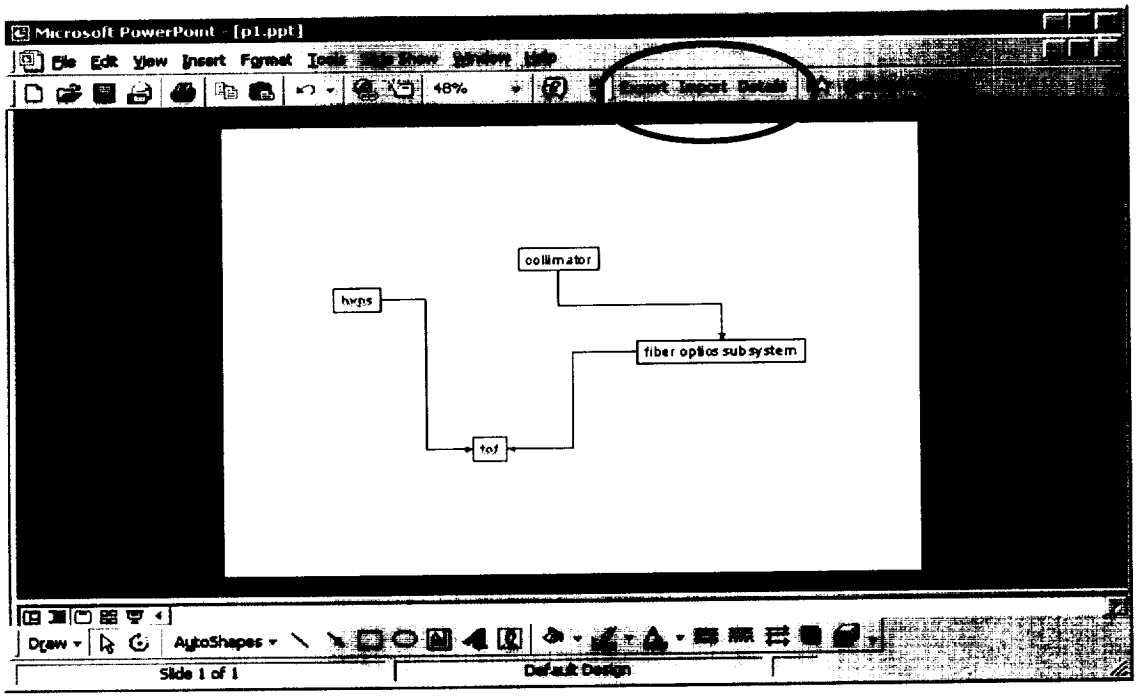


Figure 3. A PowerPoint drawing is used to specify the components and interconnections of the instrument. The Export and Import buttons (within the red circle) transfer the model information to and from the extended Excel tool. The Details button provides a toggled display of the detailed modeling information provided through the extended Excel tool.

The graphical view provides an overall understanding of the structure of the instrument, including the reuse of existing components. This information is then exported to Excel, which is used to provide detailed model information. Excel is the preferred mechanism for the detail level since it consists largely of tables of elements. The type of each element in column two is indicated by an icon, which is automatically created when the element is defined through the Element Editor.

The Excel representation, shown in Figure 4, consists of two main columns, the first of which lists the components imported from the graphical view.
The second column lists, for each component, detailed model elements associated with that component, e.g., raw and derived data, constraints, actions, and rules. Information associated with the instrument as a whole, rather than with a specific component, can also be placed in this column under the grouping "System."

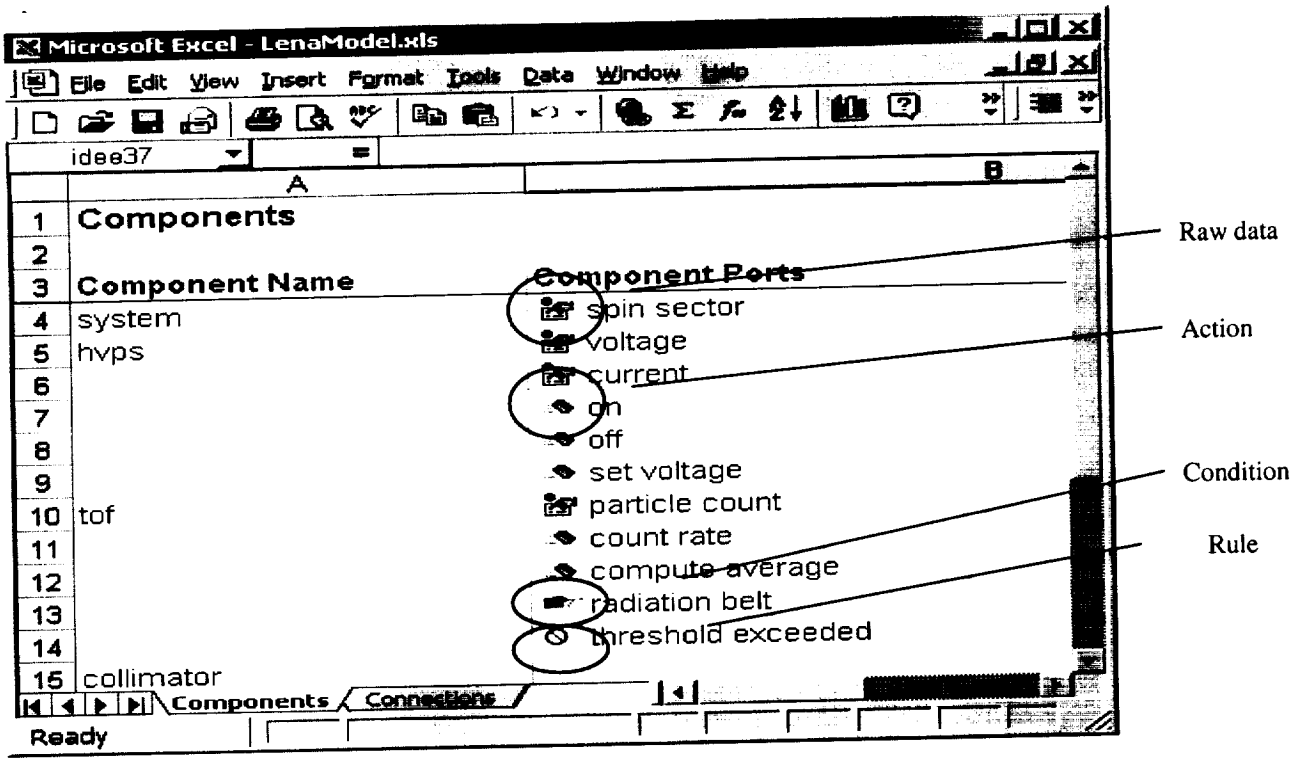


Figure 4. Detailed attributes and derived objects for each instrument component are specified using an extension of Excel.

Element Editor—In order to facilitate specifying the details of derived quantities, conditions, actions, and rules, we developed a context-sensitive editor for these element types. The editor is implemented as an Excel macro, written in Visual Basic. Figure 5 shows the user interface, which pops up over the Excel spreadsheet on a single keystroke. Since different types of information are required for each type of element, the editor is implemented as a tab panel, and the fields appropriate for each element type appear when the corresponding tab is selected.

Although free form typing is permitted, the editor provides operator buttons for building complex expressions; the use of these buttons ensures that expressions will be well formed (i.e., correct number of operands, parentheses balanced, etc.). When creating or modifying expressions, the designer can import another element simply by clicking on that element in the spreadsheet and pressing a single command key. Conversely, the user can select an arbitrary sub-expression in the currently edited expression, and with a single keystroke highlight (in the spreadsheet) all the elements that are referenced in the sub-expression. These functions minimize free form typing and facilitate comprehension by linking referenced elements. They represent an initial attempt to address the requirements discussed in the *Content of a VEST Model* Section.

Solver approach

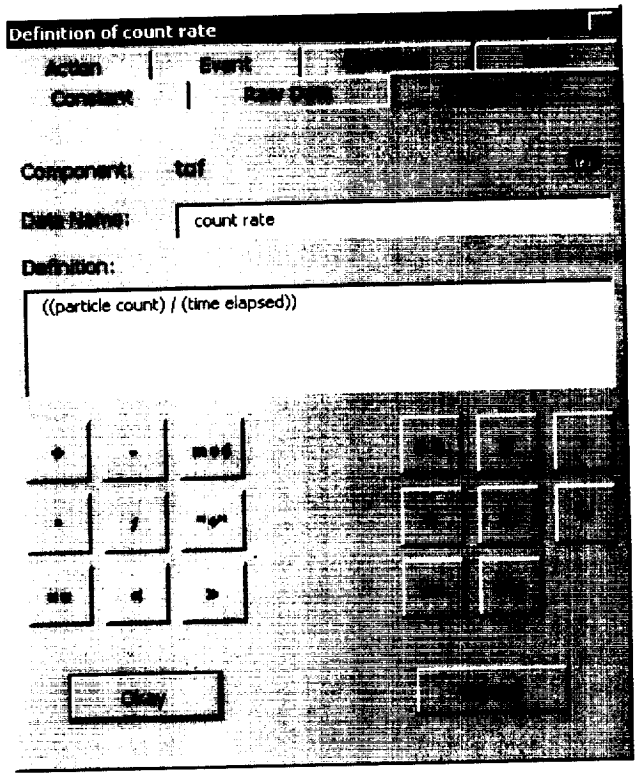


Figure 5. A context sensitive editor allows for the precise definition of model element details.

Rule checking, the algorithm for which was described in the *Rule Analysis Algorithm* section, is implemented using the Solver add-in to Excel. The Solver takes a set of constraints involving a set of spreadsheet cells and tries to find values for the cells that satisfy the constraints. The rule checking algorithm takes a pair of rules in the VEST model and converts them to a set of logical expressions consisting of Boolean combinations (involving the logical operators AND, OR, and NOT) of equations and inequalities.

Unfortunately, the Solver does not take arbitrary expressions of this form, but only conjunctions of equations and inequalities. We handle this by converting the expression into disjunctive normal form, which consists of a disjunction (ORs) of conjunctions (ANDs) of equations and inequalities. Each conjunctions is then passed to the Solver successively. If any of them is solvable, a potential rule conflict has been discovered.

Solver complexity issues

Conversion of logical expressions to disjunctive normal form works, in principle, but we found that computationally it is infeasible. The conversion process involves replicating sub-expressions in the process of pulling OR operators out, and pushing AND and NOT operators in. In general, this results in an exponential rate of replication, so that even moderately complex expressions become intractably large, and the algorithm bogs down.

A further problem with the Excel Solver is that it does not accept strict inequalities (i.e., $x < y$) but only inequalities of the form $x <= y$. To get around this, we convert $x < y$ to $x <= y-\varepsilon$ where ε is a settable constant representing the desired level of precision.

The problems suggest that future versions of VEST must employ a solver that is more oriented towards mathematical and scientific applications. Determining whether a solver exists that would meet the VEST requirements is an open issue, which we will pursue in further work.

Vizualization

Another novel aspect of the VEST tool is the support it provides for visualizing complex logical expressions. This feature is a response to our analysis of the LENA automation software, from which we inferred that much of the challenge in developing an autonomous instrument lies in comprehending the possible states and combinations of conditions. The VEST visualization functions uses 3-dimensional graphics to represent the structure of a logical expression built using the AND, OR, and NOT operators. We experimented with several such representations. We converged on a "close vs. loose stacking" concept in which conjunctions are indicated by a close stacking of one expression on top of another, disjunctions are indicated by a looser stacking, and negation is indicated through color. Figure 6 illustrates the process.

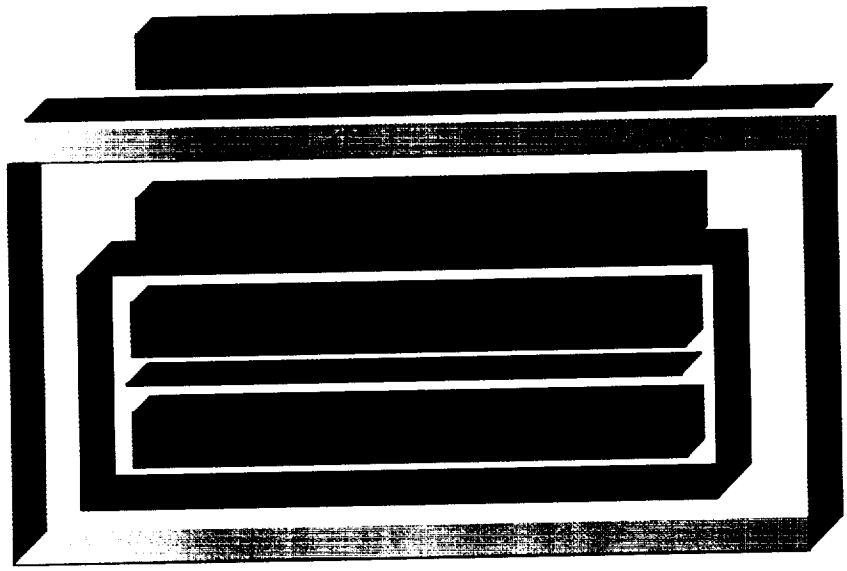


Figure 6. Visualization of the expression:
(u = v) OR (x = y AND NOT (a >= b OR c < .0001))
The red horizontal separator indicates OR, while vertical stacking without the red separator indicates AND. Complex expressions are enclosed in a "bookcase." The color of the bookcase defaults to blue, while negation is indicated via a black bookcase.

Code Generation

The VEST tool generates C code that represents each component as a structure (representing the raw data fields) with associated definitions (representing derived data and conditions) and functions (representing actions, and rules).

Fault isolation code is also generated for the instrument as a whole. These functions use a standard model-based approach of fanning out from an observed anomaly in order to determine the source of a problem. If a constraint governing a particular component is violated (for example, a threshold exceeded), that component is first checked for discrepancies between its input values and output values. If all rules governing the relations between the component's inputs and outputs are satisfied, it is inferred that the source of the problem lies elsewhere and those components that provide direct input to the original component are checked. The process continues iteratively in this way until a faulty component is found or there is nothing more to check.

4. FUTURE DIRECTIONS

Generalization to other instruments

The application of what we have learned working with VEST is readily transferable to space systems with high voltage components that may suffer faults or degrade over time. These components include electronic optics and detector components such as found in IMAGE/LENA. These particular components are quite common in the various kinds of particle sensors that are used in science and industry. Related components are found in some light amplification technologies that suffer similar vulnerabilities to over stimulation from the science target or interference. VEST may enable the fairly direct transfer of some software functions between such closely related components. However, there is more to VEST than the modeling of high voltage power supplies and the control of particle optics and detectors. VEST also provides an entire layer of control functions that allow the production of a software system that seeks to maintain the health and safety of the system to be deployed.

Each different kind of system that uses VEST will require its own set of interfaces and models, perhaps built on or extending prototypes already built using VEST. Once constructed, these models and interfaces can then be linked or mixed-in with software functions that aid the implementation of instrument autonomy. In this way common software themes arising in these systems may be shared between projects. What enables this sharing across products is the strict enforcement of a design and development protocol: the VEST tool both provides and enforces this protocol. However, flexibility is still maintained by providing escape hatches to lower levels of code development.

That said, the application of VEST to the development of systems involving high voltage electronics and particle detectors follows the example provided by its application to LENA-like instruments. Electromagnetic imagers, e.g. X-ray, XUV, etc, have their own operational characteristics, often involving high voltage electronics, that would benefit from VEST. Even instruments with geometrical or pointing constraints, e.g. Sun-avoidance, could benefit from VEST's consistency checking. In fact, it is likely that nearly any Boolean or finite ranged function defined on instrument state could be made part of a VEST model. In that case, models with even purely geometric states, e.g. CAD models, could be integrated and checked with VEST. One interesting possible application of VEST is to aid the development of fault tolerant control systems for high fault rate systems or components, e.g. non radiation-hardened electronics applications in the space environment.

Validation

The next step we plan to take with VEST is to validate the approach described in this paper by generating autonomous control systems (ACS) for two space science sensors. The first sensor for which we will construct a model, rule system, and operating scheme will be LENA itself. Comparing the VEST-produced ACS with the manually produced one will provide interesting insights into the tradeoffs between the two approaches. We expect to gain insight into what was easily accomplished in one, but not so easily accomplished in the other. The way errors, e.g. design or coding, manifest themselves in the two systems will be compared. How the existence of models and formal consistency checks affects testing and quality assurance for flight qualification is of great interest. Finally, the performance of the VEST-produced ACS will be compared with the manually generated system. In this way we will validate the VEST-aided design and development process, as well as the VEST-produced ACS itself.

Second, we hope to show that higher level ACS functions could be brought to bear for science instruments that are not so similar to LENA. This will show that VEST can provide cross-project reuse and generalization. We will choose a key instrument type to examine based on the availability of expert knowledge, instrument data, and resources. One option is to apply VEST to an imaging instrument, e.g. a Solar X-ray imager, that has a mix of health & safety requirements, perhaps including high voltage electronics, but may also feature a wider range of operational functions and

requirements. For example, many such imagers have multiple modes including field-of-view size, data gathering intervals, and pointing requirements. Switching between modes can be induced for either health & safety concerns or science operations. Autonomy can be important for such instruments because the reaction time available to set modes for optimal science data gathering is usually much greater than the communication latency inherent in the ground-based command and control. Using VEST to develop such science operations software would be a significant generalization from LENA, while showing some reuse due to common requirements and related hardware components.

The Next Generation

After prototyping and verifying VEST, we then plan to review the approach we have taken and see how the decisions we made along the way affected the results of our work. Implementation issues concerning the usability and robustness of the commercial software in this work will be reviewed, and lessons learned will instruct future implementations of the VEST application. For example, we expect that it will be profitable to review commercial mathematical modeling applications and expression solvers for model building, state specification, simulation, and visualization. However, any of these systems will have to be augmented with the VEST framework of interconnections between system components, their behaviors, and constraints thereupon. A library of functions for implementing science instrument autonomy will have to be ported to, or developed for, the new programming environment from which the embeddable target code would be generated. Though it is quite early to say definitively, it may be possible to allow communication between a software-side modeling & development tool like VEST, and a domain specific tool, e.g. a mechanical CAD application, to check for the consistency of an even richer spaces of states and constraints.

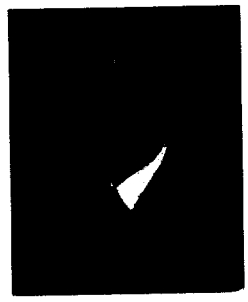
REFERENCE

[1] T. Moore et al., "The Low-Energy Neutral Atom Imager for IMAGE", Space Sci. Rev., v91, p155-195, 2000.

BIOGRAPHIES

Sidney C. Bailin is founder and President of Knowledge Evolution, Inc., a firm that develops and promulgates innovative knowledge sharing technology. Prior to forming Knowledge Evolution he was a Vice President of Engineering at Computer Technology Associates, where for 12 years he played a leading role in that company's software technology program. His software experience, which spans 22 years, has ranged from the development of production real-time communications systems to research and development in automated reasoning and in information agents. He was one of the originators of Object-Oriented Analysis. He has been active in the software reuse community for the past 14 years, and is best known in that community for introducing the KAPTUR methodology, which links reuse to rationale capture. His most recent work concerns the dynamic negotiation of ontologies between agents. Dr. Bailin has published roughly two dozen papers on various aspects of software engineering. He currently sits on the Editorial Board of the Encyclopedia of Software Engineering, where he oversees the topics areas of Software Reuse and Artificial Intelligence in Software, and shares responsibility for entries concerning Object Oriented Design. He is the principal author of a recent book on the LIBRA approach to knowledge sharing, published by the IEEE Press.

Michael A. Johnson is a senior electrical engineer in the GSFC Microelectronics and Signal Processing Branch. He has served as the lead electrical engineer for numerous flight systems at GSFC, including the IMAGE/LENA Electronics Systems, the Cassini/CAPS Spectrum Analyzer, and the Triana/PlasMag Electron Spectrometer Controller. His current research interest involves developing autonomous control strategies for spaceflight instruments that have severely constrained resources.



Prior to coming to Goddard, he was a lead electrical engineer at MIT/Lincoln Laboratory responsible for the design and development of space and ground-based systems. He received his B.S, M.S and Electrical Engineer degrees from MIT.

Michael L. Rilee is a scientist with Emergent IT supporting the GSFC Laboratory for Extraterrestrial Physics and the Science Computing Branch of the GSFC Earth and Space Science Computing Division. For NASA's Remote Exploration and Experimentation project he has led development of the Plasma Moment Application and the Radio Astronomical Imager, which are science data analysis applications designed for space borne supercomputers. He is currently researching a High Performance Computing System that may fly on Magnetospheric Multi Scale (launch 2007). At GSFC he has been active in Nano-Satellite technology development and the application of parallel computing to data analysis and astrophysical fluid simulation (PARAMESH). He received his Ph.D. and M.S. in Astrophysics (Plasma Physics) from Cornell University, and his B.A. in Astrophysics and Mathematics from the University of Virginia in Charlottesville, VA.

Walt Truszkowski is currently the Senior Technologist in the Advanced Architectures and Automation Branch at NASA's Goddard Space Flight Center. In that capacity he is responsible for managing the agent technology research for the Branch. Currently work is underway to establish an agent-based system for the ESA/NASA satellite SOHO. He also serves as the Lead of the Information Technology Research Group in the Branch. In that capacity, he is leading an effort to create a repository of information on technologies of importance to researchers in the organization. He is also leading the research in the areas of human factors of website design/use and the application of agents for the intelligent access and management of web-based information.

He is a National Research Council (NRC) accredited Fellow participating in the Resident Researcher's Associate (RRA) program at the Goddard Space Flight Center.