Proceedings of NASA's 2013 Annual Workshop on Independent Verification and Validation of Software

September 10-12, 2013

Robert H. Mollohan Research Center
Fairmont, W. Va.

Table of Contents

JAXA's IV&V Activity and Value Concept

Tsutomu Matsumoto, Japan Aerospace Exploration Agency (JAXA)
Umeda Hiroki, JAXA

ABSTRACT

JAXA has applied IV&V activities to software development of satellites and space crafts for more than ten years since the International Space Station Program. JAXA's IV&V activities have achieved results by detecting critical problems in software. The IV&V activities are becoming established for all types of spacecraft in JAXA.

As stakeholders understand IV&V more, they need the IV&V with higher quality and explanations of verification to relieve them. Moreover, not only detecting problems as usual but also providing a new IV&V value to contribute to development is expected. This presentation shows an idea and examples of new IV&V activities.

IV&V Lessons Learned From On-Orbit Anomaly Research

Joseph D. Painter, NASA IV&V / TMC
Stephen M. Pukansky, NASA IV&V
Stephen Husty, NASA IV&V
Koorosh Mirfakhraie, NASA IV&V / TMC

ABSTRACT

In this presentation, a number of software-related anomalies experienced in the course of some of the NASA missions will be discussed. These anomalies encompass various components of flight software, such as Command and Data Handling. A description for each anomaly will be provided, along with the factors leading to the anomaly and the impact of the anomaly on the mission. The focus will be on the role that software has played in the anomaly, either as a culprit or in its inability to prevent or contain the anomaly.

Based on the information learned about the role of software in these anomalies, lessons are drawn to be applied to the future IV&V analysis of space mission software. The resulting improvements in performing IV&V during the development of software should help with identifying issues, whose satisfactory resolution may avert similar anomalies in the future, or will otherwise increase the quality of software and reduce the number of faults introduced during its development.

Orion Crew Exploration Vehicle Model-Based Design Implementation – An IV&V Perspective

Joel Henry, NASA Johnson Space Center
David Frazier, NASA IV&V / TASC
Steve Driskell, NASA IV&V / TASC
Leonard Frost, NASA IV&V / SAIC

ABSTRACT

The Orion Crew Exploration Vehicle (CEV) Guidance, Navigation and Control (GN&C) design and analysis team is developing onboard GN&C flight software (FSW) algorithms using the MATLAB®/Simulink® tool suite to embrace a Model-Based Development approach to FSW development. Various aspects of this modern approach are described – including software architecture, design approach and modeling standards using MATLAB®/Simulink® for the GN&C executive and its algorithmic Computer Software Unit (CSU) components. The developer methods employed for unit-level and closed-loop testing simulation, test environments and the test and verification of the auto-generated code products are also presented. Modeling benefits, process challenges and lessons learned to date are summarized.[1,2] The NASA's IV&V Program analysis team is developing activities to assess the GN&C auto-generated code that will fly on the first CEV operational test flight, OFT1. This team discusses the approaches used to provide software implementation and design for the model-based auto-generated FSW products. With NASA agreement, the development does not produce standard artifacts typically used in software IV&V. This presentation identifies approaches to analysis based on what is available from the tools which contain this information. One of the CSUs under analysis will be reviewed. Process challenges and lessons learned to date will be provided, along with a scope discussion for FSW assessment vs. other software build products (which are out of scope for IV&V).

References

[1] 978-1-4244-3888-4/10/$25.00 ©2010 IEEE
[2] IEEEAC paper#1491, Version 3, Updated 2010:01:07

NASA IV&V for the New York City E911 Program

Shirley Savarino, NASA IV&V / TASC
Mike Facemire, NASA IV&V
Hendrik Strydom, New York City

ABSTRACT

New York City's Emergency Communications Transformation Program (ECTP) is a multi-year initiative to enhance call taking and dispatch operations for NYPD, FDNY and FDNY EMS. Under the program, each agency will benefit from upgraded computer dispatch systems, improved integration and data sharing between agencies, new 911 telephony networks and software, and other significant improvements. In August 2013, NASA's Independent Verification and Validation Program entered into an agreement with the City of New York to perform IV&V on the upgrade of the City's E911 call system. NASA's Independent Verification and Validation Program in West Virginia usually tests systems for the space program, like telescopes orbiting in space and rovers crawling over the surface of Mars. The City hired NASA to evaluate the $2 billion upgrade to provide an "additional set of eyes" to evaluate this complex system of systems that converges user needs and technology upgrades in the new call center which will be operational in 2015. Existing processes used for performing IV&V of complex space and manned missions were adapted for use on the ECTP activity.

This paper discusses the startup of the IV&V activity, application and adaptation of NASA IV&V processes to a new customer and domain. We address challenges faced by the City and the NASA IV&V team in starting up the activity in terms of staffing, schedules and technical. Successes and lessons learned for other such activities are presented.

IV&V of ECTP Radio System

Pradip Maitra, NASA IV&V / TASC

ABSTRACT

This presentation describes the IV&V effort on a portion of the Emergency Communication Transformation Program (ECTP) for the City of New York. The ECTP program is aimed at enhancing the 911 emergency dispatch systems. A 911 call originates from primarily three emergencies: situations requiring police intervention, situations requiring a medical emergency management team or a situation requiring fire trucks to combat a fire emergency. The part under focus in this presentation is the radio system used by NYPD (New York Police Department).

A substantial portion of the communication path between a dispatch center and an officer on the street is actually implemented as radio communication. Some of this path is microwave and some is in the cell phone frequency range typically using one of two well-known protocols. This presentation attempts to show the various aspects of these communication methods and how to perform IV&V on these areas.

Robustness of PSAP Radio System

Roman Mezhericher, NASA IV&V / TASC

ABSTRACT

This presentation was created as Technical Reference material for the FDNY Radio Console Project – a part of NYC ECTP2 (Emergency Communication Transformation Project). The presentation describes a role of Radio Dispatch in the 911 call handling process. It reveals project scope and subsystems involved, reflecting subsystems' interactions, as well as Radio Console integration with neighbor systems and particularly the FDNY Emergency Control Station (ECS) project.

The presentation contains preliminary dependability analysis showing multiple solutions applied to the Radio Console project in order to increase system availability and maintainability, including hardware redundancy and diverse paths, utilization of advanced Cloud and Virtualization technologies, as well as monitoring and control.

Also, the presentation reflects IV&V analysis of the project and already identified problems – finalized in the form of submitted TIMs and Risks.

IV&V Coverage of NASA Software Guidelines

Jacob Cox, NASA IV&V / TASC

ABSTRACT

This paper will discuss the NASA IV&V Program's coverage of the NASA Software Guidelines from Appendix H of the NASA Software Safety Guidebook (NASA-GB-8719.13). The Guidebook has guidelines for generic languages, such as C and C++. These will be discussed with respect to how and how well IV&V covers them when analyzing code from projects. Also discussed will be the techniques used that are relevant to specific guidelines and suggestions for improvement.

An Introduction to AMF

Donald Kranz, NASA IV&V / TASC
Tom Gullion, NASA IV&V / TASC
Neal Saito, NASA IV&V / TASC
Gary Marchiny, NASA IV&V / TASC

ABSTRACT

The Analysis Management Framework (AMF) provides a framework for organizing and communicating IV&V evidence-based assurance data. The AMF leverages existing IV&V tools and resources to develop NASA's IV&V Program concepts and definitions of evidence; create methods for refining IV&V assurance data into information and assurance statements; and provide a common set of process assets, Catalog of Methods inputs and educational materials.

For those interested in the Analysis Management Framework, this presentation provides an overview of the AMF concepts. The AMF is based on a 3-layered architectural approach to the IV&V domain space. The 2013 AMF Capability Development Initiative focused on fleshing out the documentation for the business and data layers, as well as sample user interfaces in MS Office and Eclipse. Since these underpinning concepts of the AMF have only recently been pursued and developed, this brief outline should help attendees recognize which of the AMF's various aspects are most relevant to them and where to find information to about utilizing AMF on their projects.

NASA Operational Simulator (NOS)

Justin R. Morris, NASA IV&V

ABSTRACT

The NASA Operational Simulator (NOS) is a generic software-only simulation architecture for NASA missions. NOS was developed by the NASA's Independent Verification & Validation (IV&V) Independent Test Capability (ITC) Team to provide a complete software V&V environment. NOS is utilized by developers and (independent) testers to verify the functionality of a spacecraft's flight software from a system-wide perspective. Use of NOS on two NASA spacecraft is described: the Global Precipitation Measurement spacecraft and the James Webb Space Telescope. While NOS has primarily been utilized on NASA missions, its generic architecture can be easily applied across domains to support V&V of complex systems.

NOS is capable of executing the spacecraft's unmodified flight software executable on readily deployed environments such as laptops and thumbdrives. NOS consists of reusable hardware models, simulators, and custom-developed middleware that provides simulated MIL-STD-1553 and SpaceWire busses. A key feature is its dynamic error injection capabilities via intuitive GUIs and open APIs. The error injection is critical for performing V&V of the flight software; hardware faults can be simulated and off-nominal tests can be executed without additional effort. NOS is easily integrated with ground systems and other components to support complete mission analysis.

MPCV IV&V with a Dynamic Twist

Ricky Beamer, NASA IV&V / New-Bold Enterprises
David H. Ho, NASA IV&V / TASC

ABSTRACT

Traditionally, IV&V analysis work is performed statically with artifacts on requirements, models, test cases and procedures, and source codes, etc. With the availability of simulation and test tools provided by the Orion Multi-Purpose Crew Vehicle (MPCV) prime contractor, Lockheed Martin, and NASA Johnson Space Center, the MPCV IV&V team was able to enhance and broaden the traditional analysis by adding a new run time capability. This capability provides the MPCV IV&V team the ability to analyze the MPCV Flight Software functionally with in-house test cases, test procedures, and test scripts. This presentation will provide the details on the development of this dynamic capability with two different simulation and test tools, SOCRRATES and PLATO.

Using Static Code Analysis Tools for Detection of Security Vulnerabilities

Katerina Goseva-Popstojanova, West Virginia University
Andrei Perhinschi, West Virginia University

ABSTRACT

The advances in technology and broadband connectivity, combined with the ever increasing number of threats and attacks, require information assurance and cybersecurity to be integrated in the traditional verification and validation process. NASA develops, runs and maintains many systems for which one or more security attributes (i.e., confidentiality, integrity, availability, authentication, authorization, and non-repudiation) are of vital importance. These aspects of cybersecurity are especially critical in command and control systems. Therefore, it is becoming imperative to extend the current IV&V capabilities to cover information assurance and cybersecurity concerns of NASA projects.

Static analysis of source code provides a scalable method for security code review and helps ensure that secure coding policies are being followed. Tools for static analysis have rapidly matured in the last decade; they have evolved from simple lexical analysis to using much more complex techniques. However, in general, static analysis problems are undecidable (i.e., it is impossible to construct an algorithm which always leads to a correct answer). Therefore, static analysis tools do not detect all bugs (and thus vulnerabilities) in the code and are prone to false positives, i.e., they may report findings which, on closer examination, turn out not to be security vulnerabilities. A high number of false positives requires significant manual analysis effort and thus wastes valuable resources that may be used better elsewhere. To be of practical use, a static code analysis tool should find as many vulnerabilities as possible (ideally all) with a minimum amount of false positives (ideally none), can be customized to define additional rules to enforce internal coding policies and provides information on vulnerabilities and potential remediation. However, the actual efficiency of the static code analysis tools in detection of security vulnerabilities is not known.

The goal of our NASA's IV&V Program-funded FY13 Capability Development Initiative is to examine several static code analysis tools using both quantitative and qualitative criteria. The assessment is based on using the Juliet test suites for Java and C/C++, which were created by the NSA evaluation effort and have been made publicly available at the National Institute of Standards and Technology (NIST) website. In addition, to assess the tools' performance on applications with realistic complexity, we built a test suite of representative programs with known vulnerabilities. The results and products of this CD initiative will contribute towards leveraging NASA's IV&V Program capabilities and services by using state-of-the practice tools that support information assurance of NASA projects that have high security risk. This work is

done in collaboration with the Space Network Ground Segment Sustainment (SGSS) IV&V and Software Assurance Tools (SWAT) teams.

Evidence-Based Assurance with the Analyst Workbench

Zachary Seamon, NASA IV&V / TASC
Donald Kranz, NASA IV&V / TASC

ABSTRACT

The Analyst Workbench (AWB) is a NASA's IV&V Program-developed Eclipse tool that interfaces with the Analysis Management Framework (AMF) data structure to provide dynamic traceability views between requirements, design documentation, analyst assessments and other project data. The AWB provides a powerful visual representation of project data and data relationships that other tools lack. This demonstration will provide an overview of the capabilities and features of the AWB, showing the tool's ability to enhance both the quality and quantity of work for analysts and managers. Attendees will see a general overview of the tool and attain the knowledge necessary to assess if the AWB may be a valuable tool for their team's analysis efforts. The AWB will become a core tool in the analysis and management efforts of NASA's IV&V Program going forward.

IV&V Analysis Management Framework Overview

The Analysis Management Framework (AMF) is a capabilities and development (CD) Initiative to provide a three-layered analysis framework (data, business, and user interface) that applies best practices of information system development to organize and communicate data at NASA's IV&V Program. Currently, there are many different user interfaces and data storage types being utilized in the program. Whenever a new interface or database is added to a project, schemas to interface the new product with existing ones must be constructed. This process becomes extensively time consuming and messy, as illustrated in Figure 1.

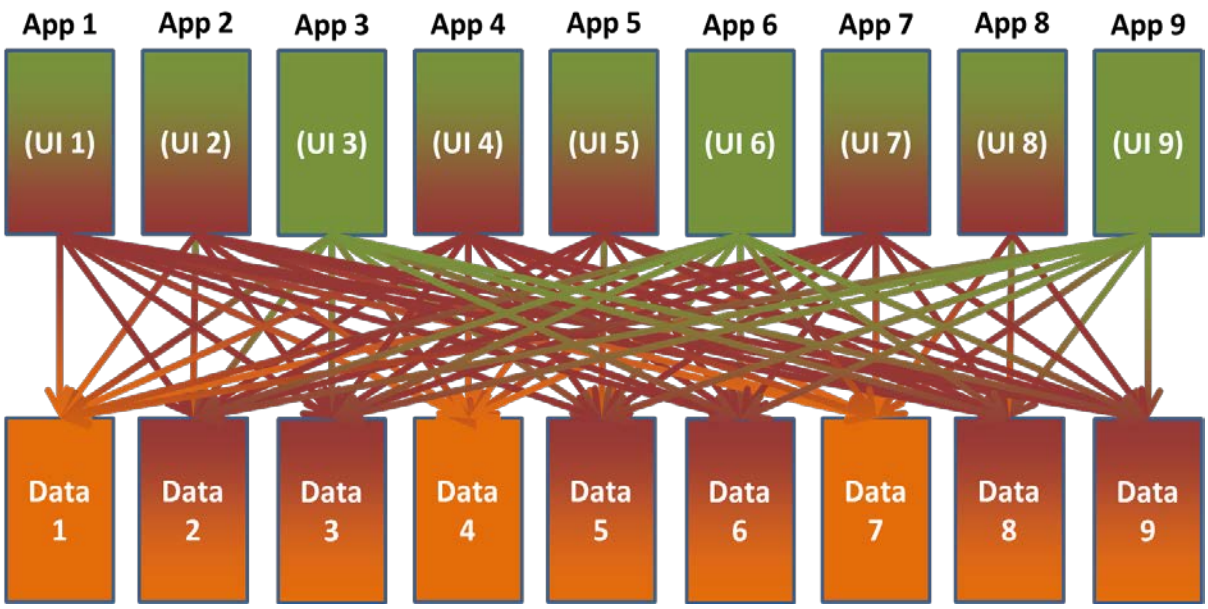

Figure 1: UI to Data relationship without AMF

The AMF solves this problem with the implementation of a central business layer, which provides a common interface between any number of user interfaces (UI) and data. When a new UI or data source needs to be integrated, it simply needs to plug into the business layer to be integrated with the rest of the project. This concept is illustrated in Figure 2.

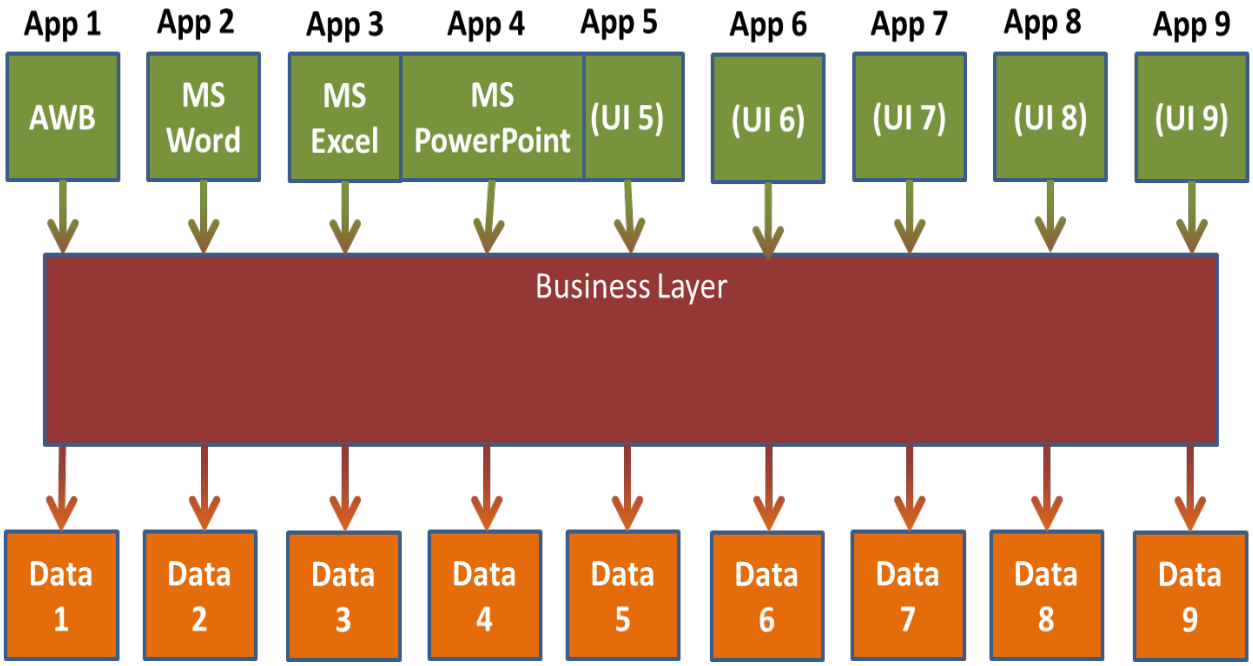

Figure 2: UI to Business Layer to Data relationship in AMF

In addition to a simplified UI and data management on an individual project, the deployment of the AMF program-wide allows for a standardized practice of data management. This reduces the amount of time spent within each project dealing with data manipulation, increasing the time and effort available for analysis work.

Analyst Workbench Overview

The Analyst Workbench (AWB) is a plugin that provides one interface option in the AMF. Originally developed for the SMAP team, the AWB provides a dynamic user interface to view project artifacts, IV&V assessments, and tracing between them. The tool interfaces with a database, via a business layer, to dynamically show the trace relationships between data, as well as allow analysts to write and attach assessments to data. Figure 3 shows a screenshot of the default AWB interface.

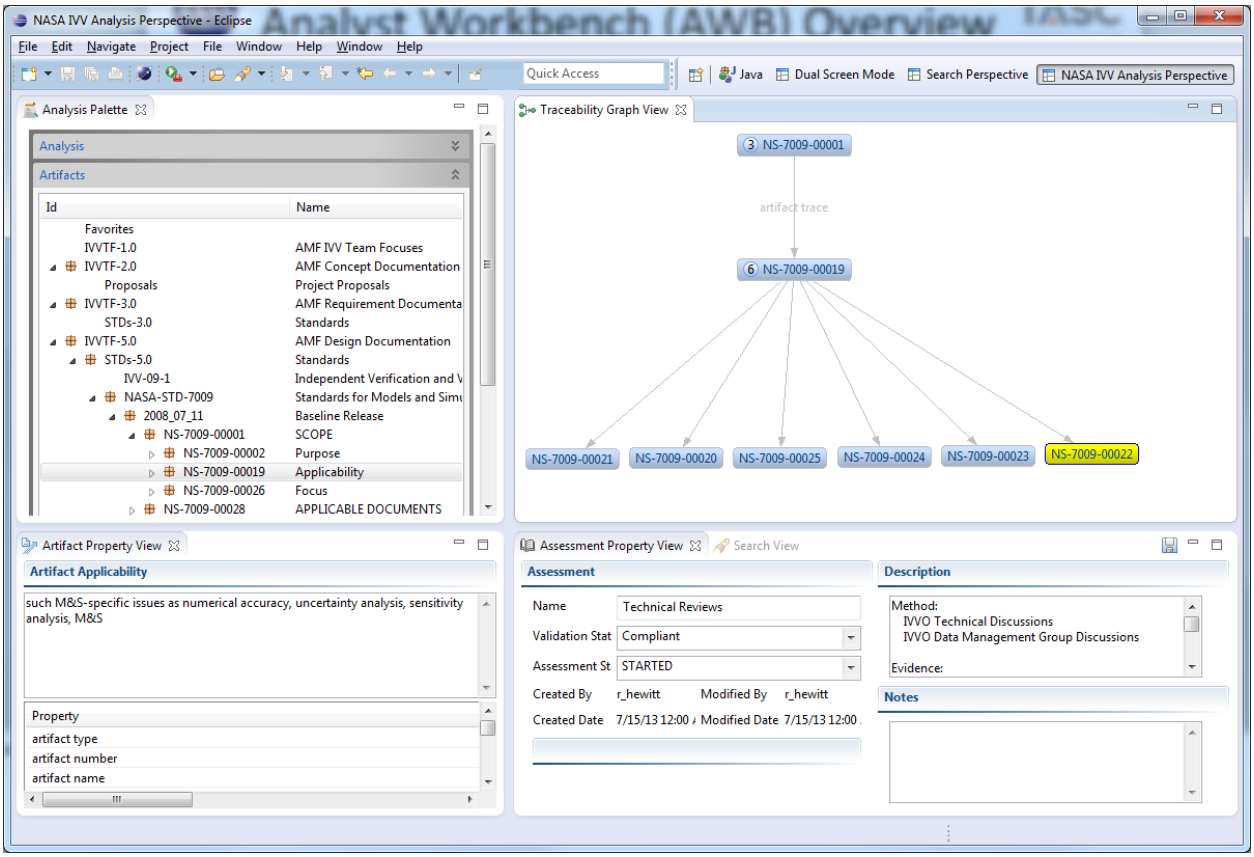

Figure 3: AWB User Interface Example

Perspectives, Views, and Preferences

The AWB consists primarily of five sub-windows called views (Analysis Palette, Traceability Graph View, Artifact Property View, Assessment Property View, and Search View) that each provide a different functionality. Each of these views can be moved, resized, minimized, or closed. This allows the user to customize the interface to his or her preference. The collective

layout of these views is called a “perspective”. The default perspective is shown above in Figure 3.

Users have the ability to manipulate the views to their liking, and then save the perspective with a distinct name. In some cases, a user may have multiple perspectives, each designed for a specific task or type of analysis. The user can then switch to whatever perspective best supports the task at hand. Users also have the ability to switch between different databases (different projects, development versus production, etc.), or toggle historic data. Historic data is any artifact that isn’t the most recent version of that artifact.

Analysis Palette

The Analysis Palette is perhaps the most used view in the AWB. It allows the user to view the data in the database in a hierarchical tree type layout, very similar to a folder structure in Windows. The data is split into three sections: Analysis, Artifacts, and Participants.

The Analysis partition contains data of the type ‘Assessment’, which is anything that was produced by IV&V. Assessments could be design assessments written against a Functional Design Document (FDD), requirement verification, code implementation analysis, etc. In short, Assessments are pieces of evidence created by IV&V.

The Artifacts partition contains data of the type ‘Artifact’, which is anything produced by the project (as in the JPL/Godard/etc. project team), and has not been changed or edited by IV&V. Some common examples of Artifacts are FDDs, requirements, code, etc.

The Participants partition contains a list of participants for the project. This shows who has permissions to manipulate the data in this database. Participant’s user ID will also be tied to any Assessments they make.

The Analysis Palette is used for basic navigation for the data in the database. The trees follow a parent-child hierarchy, based on the traces stored in the database. Documents are broken down into individual parts by section, paragraph, figure, or table. For example, if this paper was imported to the database, this paragraph would be an artifact that is a child to the “Analysis Palette” section artifact, which would be a child to the “Evidence Based with AWB” artifact.

There are various options available by right clicking on an artifact in this view:

- **Create assessment** – creates an assessment (requirement, design, semantic, or test) traced to this artifact.
- **Show traceability of artifact** – resets the Traceability Graph View and adds this artifact to the view.
- **Add artifact to traceability graph** – adds the artifact to the existing Traceability Graph View without resetting it.

- **Add artifact(s) to current assessment** – creates a trace between the selected artifact(s) and the currently selected assessment.
- **Create artifact trace** – creates a trace between two selected artifacts (popup appears to choose direction of child/parent relationship)
- **Compare selected elements** – compare the text between two artifacts with a popup side-by-side window.
- **View selected element(s)** – creates a popup window for selected artifact(s) displaying all of the artifact(s) information.
- **Search artifacts** – brings up a popup window to keyword-search artifacts.
- **Admin** – provides options to delete, edit, and refresh artifacts.
- **Add to Favorites** – add the artifact to the “Favorites” section for quick access.
- **Remove from Favorites** – remove the artifact from the Favorites section.

Traceability Graph View

The Traceability Graph View is a powerful visual tool inside the AWB. Right clicking on an artifact or assessment in the Analysis Palette or Search View gives the user options to show or add the selected artifact or assessment to the Traceability Graph. A visual element representing the data is then placed on the graph. Artifacts appear blue, assessments are green, and the currently selected element is yellow. Users can add further visual distinction by creating custom color schemes for specific elements. By right clicking on an element, the user can add child artifacts, parent artifacts, or assessments to the graph. These options will add additional elements to the graph that are traced to the selected element and fit in the selected category. Traces are shown with an arrow from the parent to the child. See Figure 4 for an example.

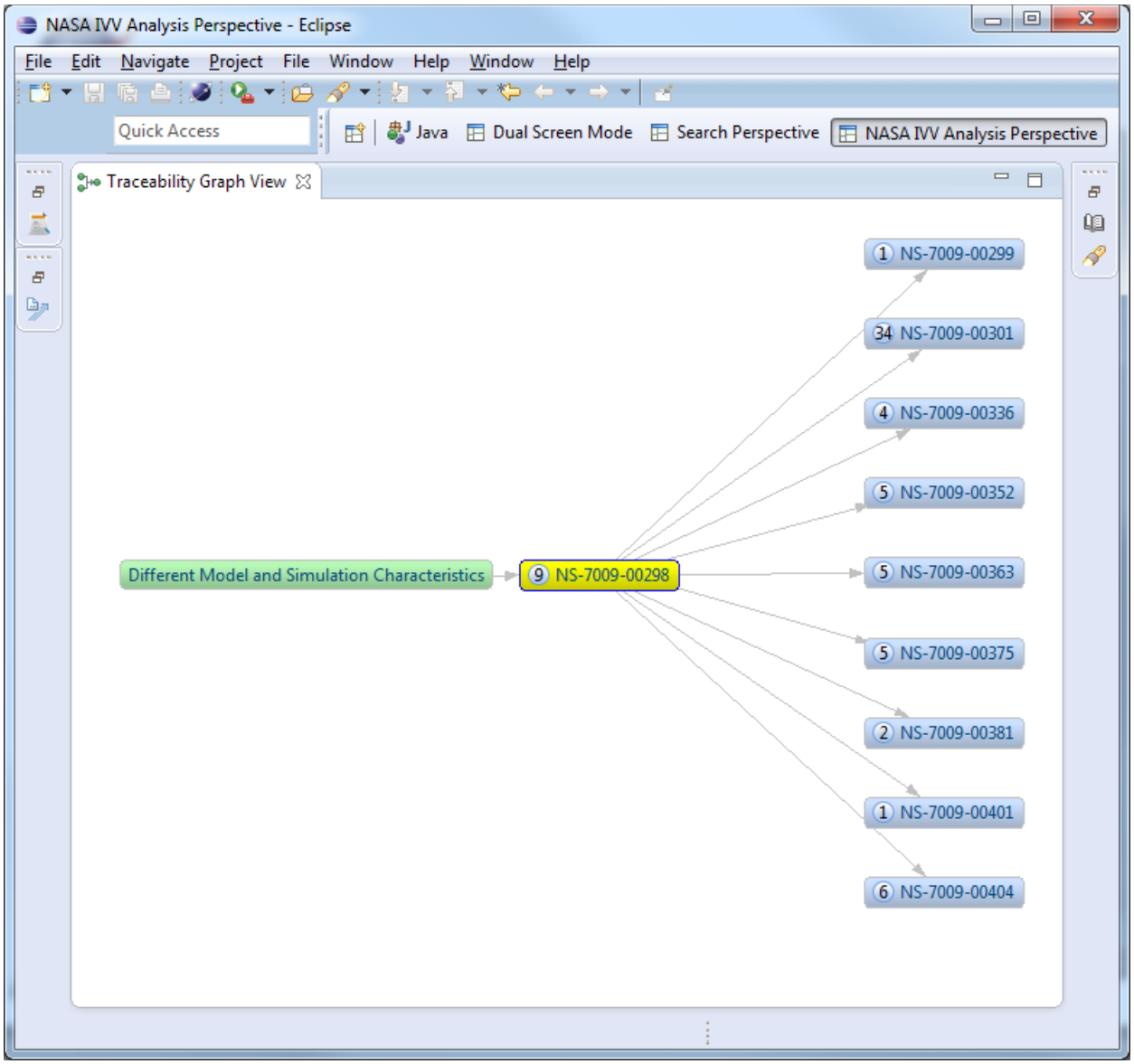

Figure 4: Traceability Graph View Example

After a few link expansions, a dynamic tree begins to form, displaying the relationships between linked data. This functionality provides a unique visual interpretation of data that can enhance analysis by showing relationships, holes, etc. in a way that may not otherwise be possible. The user can quickly see the breakdown of high level requirements, trace a lower level requirement back to its source, see where issues are linked to specific lines of code, see linked analysis work on an FDD, or trace additional evidence to claims.

When right clicking on an element, the user may choose from the following options:

- **Add child artifacts** – add traced children to the graph
- **Add parent artifacts** – add traced parents to the graph

- **Add assessments** – add traced assessments to the graph
- **Refresh the graph** – reset the graph to its origin, with only the selected item remaining
- **Remove node from graph** – remove the selected element and any related traces from the graph
- **Create trace** – create a trace between two elements on the graph
- **Layout** – select from a list of automatic layouts to rearrange the graph
- **Save view** – save a view for later viewing by user or other participants
- **Restore view** – restore a previously saved view
- **Manage views** – delete previously saved views
- **Save as image** – export the graph as an image file

Artifact Property View

The Artifact Property View displays the detailed information of any artifact selected in the Analysis Palette, Traceability Graph View, or Search View. The view shows all of the attributes stored in the database for the selected artifact. The top half shows the “description” in a large text field. This is actual artifact itself (requirement text, paragraph in an FDD, piece of code, etc.). Below that, all of the other attributes are listed:

- **Artifact Type** – The type of artifact (top level, folder, document, section, requirement etc.)
- **Artifact Number** – ID number for artifact (from project, often a document or requirement ID)
- **Artifact Name** – Name of the artifact
- **Artifact Description** – actual text of the artifact
- **Artifact State** – state of artifact in the project (initial release, Rev A, etc.)
- **Artifact Version State** – state of the artifact in the database (New, Modified, Deleted, Unchanged)
- **Artifact Created** – date that the artifact was created
- **Artifact created by first** – participant that created the original artifact
- **Artifact created by last** – participant that created the current version of the artifact
- **Artifact created by id** – ID of artifact creator
- **Artifact created by active** – state of the participant, true for active, false for inactive
- **Artifact ID** – unique number for artifact in database
- **Artifact historic** – true if this is not the newest version of this artifact

Assessment Property View

Similar to the Artifact Property View, this view displays detailed information about a selected Assessment. There are four subsections of this view:

- **Assessment** – basic information, includes: Name, Validation State, Assessment State, Created By, Created Date, Modified By, and Modified Date.
- **Assessment Attributes** – additional information specific to the assessment type
- **Description** – formal text field, this is where the formal evidence, claim, or analysis should be recorded
- **Notes** – additional information, questions, concerns, etc. related to the assessment can be recorded here.

Search View

The search view provides an alternative to the Analysis Palette for finding data. The user can choose several filters and then search the database based on those filters. The user first selects if they are searching for an Artifact or Assessment. Next, the user can select from a list of attributes to search on (artifactName, description, number, state, versionState). Search text is entered into the text field to the right of that selection. At this point, the user can either initiate the search or apply a second set of search criteria, with an additional attribute and search item. After the search, the results can be selected to view them in the Artifact Property or Assessment property views. The user can also right click to add the item to the Traceability Graph or view in a popup window. See Figure 5 for an example.

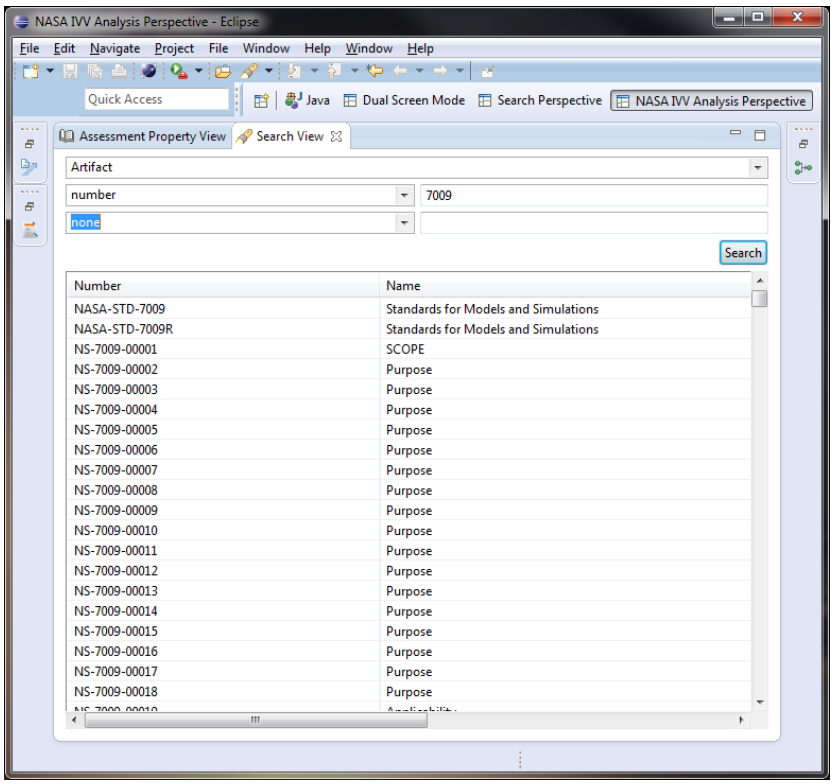

Figure 5: Search View Example

Evidence-Based Assurance

Evidence-based assurance (EBA) is a key goal of NASA's IV&V program, and therefore should be a key consideration during both planning and execution of IV&V projects. EBA provides a simple structured way to formulate assurance cases to support the project goals.

Figure 6 shows a visual explanation of EBA and how it fits into the AMF with other IV&V data.

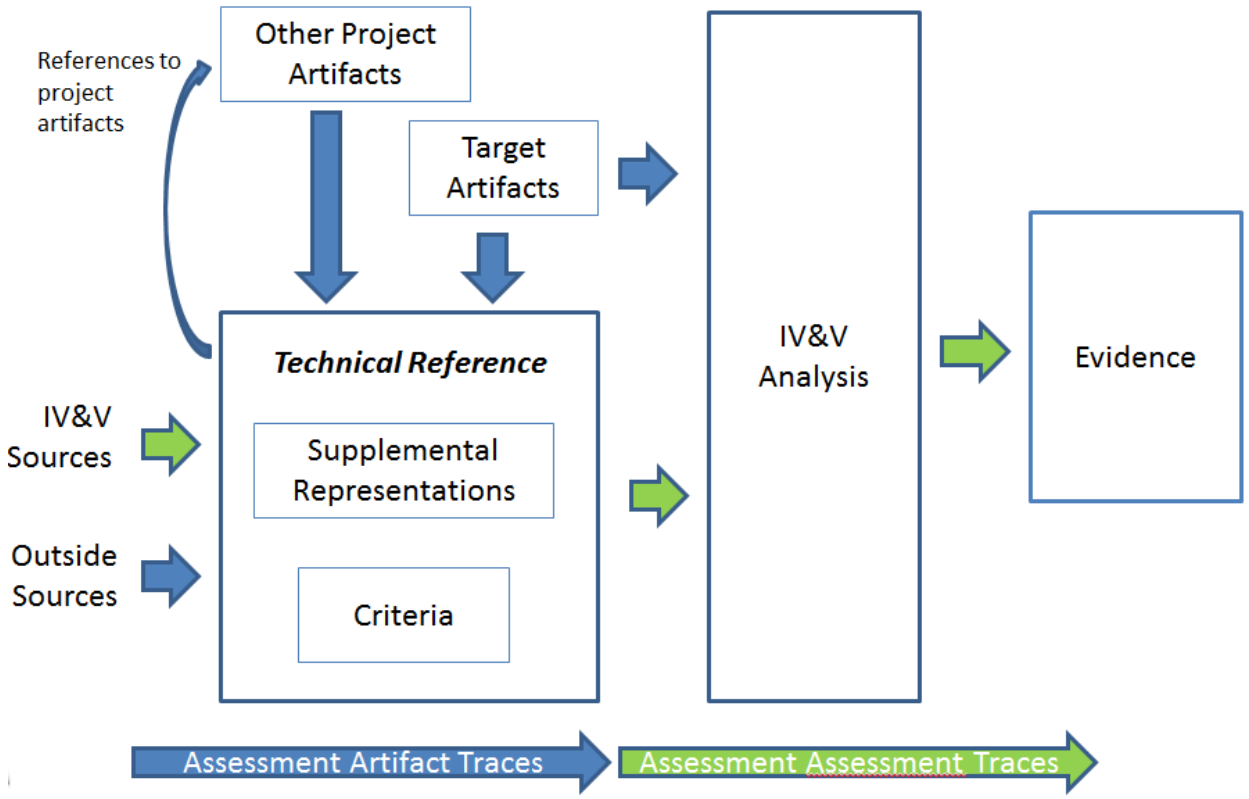

Figure 6: EBA Graphical Explanation

The AWB provides a well-defined, visual, and dynamic solution for NASA's IV&V program and its EBA effort. In AWB, it is easy for an analyst or manager to attach evidence to artifacts and assessments and build an assurance claim.

A claim is represented by an assessment in the AWB. The user makes an initial claim by creating an assessment in the proper folder under the Analysis Palette's Analysis subsection. Claim details are then filled out by using the drop down options and text fields in the Assessment Property View. Additional assessments can be made to record arguments for the claim, and then traced as children to the top level claim. Next, evidence is traced to the claim's arguments. Evidence may be in the form of other assessments, or project artifacts. A completed claim can be followed down the tree by its traces. The AWB displays the claim in either a hierarchical folder structure in the Analysis Palette, or as a graph in the Traceability Graph View.

Following the claim from the top down in a traditional sense is easy in either of the above mentioned views. The Traceability Graph, in particular, is a powerful visual tool for building and understanding the claim. It is also dynamically created as more evidence is added, as opposed to a traditional Assurance Claim diagram that must be created and maintained separately from analysis work. The IV&V project can use the AWB to dynamically build, track, and present evidence throughout the life of the mission automatically.

In addition to following a claim from the top down, the AWB adds the benefit of being able to trace from the bottom up. For example, consider an analyst that is looking at an artifact in the Traceability graph. The analyst can simply “add assessments” to see any linked assessments for the artifact. Perhaps there was previous requirement analysis performed on this requirement, or a trace to an implementation assessment on a related code block. If an argument is linked, the analyst can immediately expand traces on it and see a top level claim that was linked. This dynamic discovery of relationships between sets of data is one of the most powerful advantages of AWB as an analysis and evidence tool.

Conclusion

In conclusion, the AWB is a flexible plugin for the Java Eclipse environment that provides a unique way for users to view, trace, and create data. The AWB’s robust tracing and graphing capabilities make it the ideal tool to seamlessly integrate Evidence-Based Assurance into NASA’s IV&V program in a dynamic, useful, and easy to implement fashion. The AWB’s advantage over other methods is its ability to create Assurance Cases and record evidence dynamically throughout the life of the project, without requiring extensive additional effort from analysts to create separate diagrams or documents to organize and illustrate their evidence.

Supporting Project Management with AMF

Jeremy Fienhold, NASA IV&V / Mountain State Information Systems
Donald Kranz, NASA IV&V / TASC

ABSTRACT

From a project management view, the Analysis Management Framework (AMF) provides an architecture that facilitates full traceability from a technical reference to project artifacts through to the evidence-based assurance claims made by the IV&V team. The AMF provides the ability to bring in multiple project artifacts, for viewing and tracing to, in order for an analyst to have the ability to perform easy viewing and searching of all those artifacts.

Currently, projects use AMF to trace Functional Design Documents (FDDs), IBM® Rational® DOORS® requirements, proprietary database command dictionary entries and even the code base, all of which are then version controlled for change impact analysis. An analyst can write a requirement, design, implementation and/or test assessment against any one of them. Having all of these artifacts available with the trace view provides the project “big picture” view, which allows the analyst to quickly view the relationships between the artifacts.

James Webb Space Telescope (JWST) Integrated Science Instrument Module (ISIM) Independent Testing

Chris Lescinsky, NASA IV&V / TASC
Rick Hess, NASA IV&V / TASC

ABSTRACT

The JWST IV&V Team is developing an independent test campaign for the ISIM Flight Software (FSW) utilizing the Independent Test Capabilities (ITC) Team's JWST IV&V Simulation and Test (JIST) environment. Goals for this independent test campaign are: validation of the ISIM FSW robustness and elasticity; assurance that the ISIM can accomplish the intended mission, identification of areas in the ISIM FSW which could potentially contain unidentified errors, validation of Technical Issue Memorandums (TIMs) and evidence supporting the impact of existing errors specified by TIMs. Accomplishment of these goals will be via test cases identified using performance based requirements, mission operational scenarios (nominal and off-nominal) and TIMs, respectively, in addition to being supplemented by ISIM FSW source code metrics. Test scripts will be developed from the test cases and executed in the JIST environment. Test execution results in the form of data logs, recorded telemetry, break points, memory examination, and other available means may be analyzed to provide the desired evidence and assurance specified by the test campaign goals.

Space Launch System (SLS) Independent Verification and Validation (IV&V) Analysis Processes within Enterprise Architecture (EA)

Mark Lee, NASA IV&V / TASC
Guy Kubic, NASA IV&V / TASC

ABSTRACT

Both the SLS project and NASA's IV&V Program are utilizing Enterprise Architect (EA), a model-driven toolset to provide a model development platform to model the details of the SLS avionics Flight Software (FSW) components, developed for the SLS program. The SLS program is under way with requirements and design of the avionics software (SW) components.

Prior to each avionics FSW release, interim (partial) builds (sprints) are ongoing to develop a certain level of avionics FSW functionality (i.e., a partial list of requirements implemented), which are available by drop for analyst review. The first formal FSW release is scheduled for late November 2013.

The goal of the is paper is to present the "process-based" development of the EA model-driven toolset NASA's IV&V Program utilizes on these interim (near monthly) drops to: 1) understand what has been periodically developed; 2) identify drop functionality, testing and issues; 3) decide what needs to be analyzed based on maturity; and 4) develop the EA toolset for the analysis in anticipation of the formal FSW releases.

The NASA IV&V Program's development of the EA toolset is necessary to maintain evidence-based assurance of the IV&V FSW analysis paradigm on the interim drops. Ultimately, IV&V results of the EA process-based analysis will become a "results-based" analysis of the formal FSW releases.

Mining Technical Issue Memorandums for Knowledge: Challenges and Possibilities

William Stanton, NASA IV&V

ABSTRACT

NASA's IV&V Program has close to 20 years of experience analyzing mission and safety critical software systems. During this time NASA's IV&V Program has accumulated a significant amount of defects in all phases of the software development lifecycle (requirements, design, code, and test). NASA's IV&V Program documents each defect in a Technical Issue Memorandum (TIM), which is subsequently submitted to the development project for disposition. TIMs are archived in repositories by IV&V Projects. NASA's IV&V Program uses metadata about TIMs to build metrics that help ensure it is providing a valuable service, such as:

- How early IV&V gets involved during development
- Phase containment of issues found by IV&V
- Ratio of issues accepted by the development project
- Most frequent types of issues discovered by IV&V

Given that NASA's IV&V Program has several years of experience analyzing software developed for similar purposes there is an opportunity for:

- Discovering useful latent information from the repositories of TIMs at NASA's IV&V Program
- Making useful predictions based on what can be learned from the repositories of TIMs at NASA's IV&V Program

However, given the vast amount of records, manual analysis assisted by a simple keyword search capability will not yield useful results in a timely fashion. Current capabilities in automated text mining offer a promising solution to this problem. This paper will address the possibilities and potential value from:

- Automatically collating a user-specified set of TIMs
- Building predictors that automatically learn from the textual data in a user-specified set of TIMs
- Applying predictors on a user-specified set of TIMs

Along with the potential benefit of using text mining to attain useful latent information from repositories of software defects, there are also challenges and risks with pursuing a solution

for organizations like NASA’s IV&V Program that provide software assurance as a service that also will be discussed in this paper.

Finite States: Base Elements in a Computing Orchestra, Reducing Software Complexity and Improving Software Accuracy, Validation and Verification

Ronald Finkbine, Ph.D., Indiana University Southeast

ABSTRACT

A computer program is generally written in one language and is designed to map a set of inputs to a set of outputs. Data passes through the program in a variety of forms: global variables used by multiple methods, parameters passed to methods, values returned from non-void methods and data either inserted or retrieved from an external database. This variety of data forms and flows give the developing software developer plenty of options in moving data around, it does not affect the ability of the computer to understand the software, but does make life much more difficult for a software maintenance programmer to understand and modify existing programs.

Reducing the size and complexity of software components also leads to areas of code where errors cannot be inserted due to the simplicity of the component. This highly improves software quality and can lead to great savings in software testing and verification. If the software developer concentrates more on the data forms and flows, he/she keeps the correct output of every component very prominent and ready for verification and component testing purposes.

Finite State Machines (FSM) are simple, restricted input computational models that are valuable for their limited computational properties. And FSMs are an easy method to model, code and implement. The difficulty is in combining FSMs into some form of framework, organizing a number of them into an orchestra to accomplish a major program. FSMs have been found to be useful as a basis of functional testing and used in test case selection and adequacy analysis.[2]

This research project concentrates on programming at a system-wide level, allowing the programmer to organize programs by using smaller, more shallow computational units that communicate by putting data into data-paths and separating the responsibilities of larger programs into smaller programs that can be more easily understood by programmers, more easily separated from deep data within a program and thus more amenable to analysis, verification and testing. In addition, these smaller software components are potentially more able to be proved correct by formal software methods. This abstract covers the basic introduction and introduces some positive results that have been generated.

Figure 1 lists the types of FSMs that have been identified and developed in this research.

- byChar
- byString
- byFunction
- byLine
- byRecord

Figure 1: FSM Types

An FSM is a 4-tuple which contains a start state, a set of acceptance states, a set of transitions that connect states and an input alphabet. A visual sample of an FSM is displayed in Figure 2. Each state is a circle and the start symbol is the arrow on the left that signifies the *A* state is the start state. The acceptance states are all double circles. The transitions are labeled with the input character that is accepted (or action that occurs) when the transition is fired. Any character transition that is not expressly defined on the diagram leads to an ERROR state which is not recoverable (exit-able). When an FSM (like Figure 2) has the error states fully accounted for (from every other state) then the diagram is a Deterministic Finite Automata (DFA).

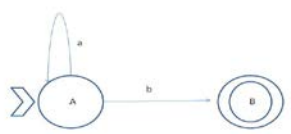

Figure 2: Sample FSM

The pseudo-code equivalent to Figure 2 is demonstrated in Figure 3.

```
state = A
while ((read ch) == a)
     ok;
wnd while
If ch == b then
     state = B
endif
if state == B then print ok;
else print error;
```

Figure 3: Equivalent Code

Software development is very complex and the development and usage of techniques and tools to reduce this complexity is essential. In third generation programming languages there are three paths for data to flow, one is in local parameters, two is global parameters and three is the formal parameters and return values of a method invocation (function call). And it can be worse if it is a recursive function.

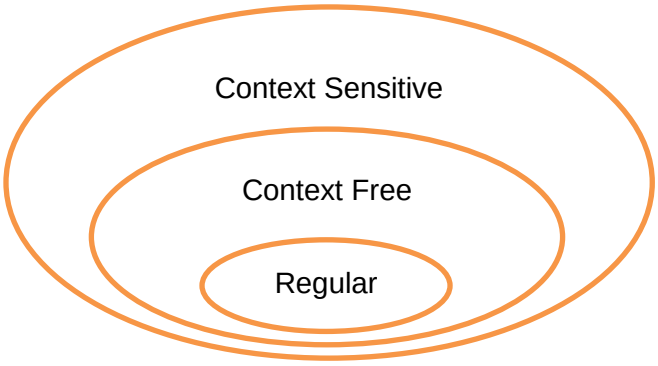

Figure 4: Chomsky Hierarchy

By linking FSMs within a framework the intent is to ensure that all data remains shallow within a program, not buried under the control of an *if* statement within a loop within a recursive function.

Figure 4 shows the Chomsky Hierarchy, a classification of languages and grammars. Any language within any of these levels can be produced and/or accepted by another program in the same level.

Regular languages in computational notation are notated as a*b*, meaning any number (including zero) of a's followed by any number of b's. This also shows the alphabet as being of

two letters, a and b. A program to read and accept strings in this language would need no extensive data structure, no arrays or stacks or database to answer the question, “Does this particular input string belong to this language?” The important characteristic of these languages is that this type of program is so simple that nothing within the program can go wrong (ignoring problems within the operating system or the hardware below, both big assumptions). When something does go wrong, it is obvious and easily detectable. The ease of detection makes these languages and software components designed upon them excellent candidates for verification and validation efforts that too often come late in the software development process, instead of early where they would be more efficient. The main crux is with regular languages, no data structure is needed to compute acceptance/rejection, and therefore there are fewer security risks in using this type of program.

aNbN – context free languages, one stack allowed, this can read the input stream and stack it up, state b can process each item in the stack

aNbNcN – context sensitive languages, two stacks allowed, therefore a symbol table in the nomenclature of compilers.

Other is Turing enumerable, any string of characters a computer program can produce, another program can accept.

Summary

Developing software using the FSM concept as a simple, restricted input computational model using an interactive framework for development will lead to software programs that are able to be coordinated into larger programs that will be higher in reliability, safety, security and lend themselves to software provability analysis. This leads to better software at a reduced cost.

References

[1] Anderson, Michael, (2003), “Diagrammatic Reasoning Website,” http://zeus.cs.hartford.edu/~anderson/

[2] Fantianato, Marcelo and Mario Jono, Applying Extended Finite State Machines in Software Testing of Interactive Systems, Interactive Systems. Design, Specification, and Verification, 10th International Workshop, DSV-IS 2003, Portugal, June 2003. Joaquim A. Jorge, Nuno Jardim Nunes, and João Falcão e Cunha, ISBN: 978-3-540-20159-5

[3] Harel, David, (1987), Statecharts: A Visual Formalism for Complex Systems, Science of Computer Programming, North-Holland.

[4] Harel, David and Eran Gery, (1997), Executable Object Modeling with Statecharts, IEEE Computer , July.

[5] Nisley, Ed, (2002), State of the Machine, Dr. Dobb’s Journal, December. O’Byrne, Brian, (2003), State Machines and User Interfaces, Dr. Dobbs Journal, January.

[6] Samek, Miro, (2002), Practical Statecharts in C/C++, CMP Books, 1-57820-110-1.

[7] Weeks, Kevin, (1992), States and the Art, PC Techniques, Oct/Nov.

Semantic Knowledge Representation

Jim Trawick, NASA IV&V / TASC

ABSTRACT

The task of Knowledge Representation and Retrieval for Independent Verification and Validation is similar in many ways to its task in natural-language processing (needing to correlate seemingly disparate but actually interdependent entities) and semantic networks, such as the long-anticipated Semantic Web (context is everything, and anything in a particular context is related). In IV&V and other technical activities, this is complicated by the difficulties inherent in communication (e.g., terms may be new and not yet agreed upon, the same acronym has multiple meanings, the information may not be in human-readable form, relevant information may be buried several layers deeper than you are looking, etc.). This requires some considerations for ontology, but ontology alone will imply relationships which do not exist.

Semantics is the study of representational meaning – usually in language – but it can be more abstract representations, as well. And ultimately, what IV&V is after is findings – but findings of significance.

Semantic relationships define a context. To be useful across multiple domains, a context definition must be both specific enough to eliminate inapplicable content, and generic enough not to eliminate applicable content from a different domain.

The Semantic Web has been touted for more than a decade, and has yet to be realized. The difficulty realizing the dream is similar to the difficulty in establishing useful arguments to verify and/or validate a technical endeavor – so much data, and so little time. Fortunately, the domain of Software IV&V is much smaller (software requirements, design, implementation and testing), and the context types are limited to that domain. This allows the ideas espoused by the Semantic Web to be applied to engines useful to the IV&V task.

Certain knowledge-based systems employing these ideas can significantly reduce the effort in capturing and retrieving applicable operators, relationships and potential consequences, as well as providing metrics to evaluate effectiveness and suggest reprioritization of resources. Such a system is particularly useful in compiling and retrieving historical data applicable to a current task, as the user can adjust the context to retrieve only those items in which the semantic context is similar, and the probability of applicability is therefore higher. And because IV&V organizations find similar technologies in different projects, a semantic approach can facilitate cross-pollination, at least better than the usual “lessons re-learned.”

Multimedia Steganalysis as Part of Mission Critical Software Independent Verification & Validation (IV&V)

Qingzhong Liu, Sam Houston State University
Noble N. Nkwocha, NASA IV&V
Andrew H. Sung, New Mexico Institute of Mining and Technology

ABSTRACT

To this date, much effort has been invested in developing sound models and rigorous IV&V methodologies to support increased software quality assurance. For mission-critical software, security in addition to reliability and safety is ever more increasingly important today than, perhaps, a few years ago. A German security researcher, for instance, is said to have recently developed and demonstrated an app that can hijack an aircraft remotely using an Android phone. With a flight simulator, the app was shown to have the ability to change the speed, altitude and direction of a virtual aircraft by sending radio signals to its Flight Management System. We propose in this paper that analyzing all multimedia associated with the software development process to ensure that they do not contain hidden and or malicious contents should be part of the IV&V focus; and that the analysis should encompass the multimedia included or used in software engineering tools, the multimedia material that the software development personnel regularly consume or are exposed to during their work and the multimedia material created or generated in the development process as part of the software product.

A focus of multimedia analysis for hidden/malicious contents is steganalysis, where advanced machine learning and pattern recognition techniques are applied in scanning multimedia material for the specific purpose of detecting hidden material, which may implement or carry covert channels and/or malware. This is because steganography, the ancient art of hiding secrets in a media, has found renewed interest and modern reinvention in the Internet age by exploiting the easy manipulability of digital material (images, audios, videos, animations, files, executables, network packets, etc.). For adversaries, saboteurs, criminals, and hackers who share interests in penetrating secure systems, common and ubiquitous multimedia presents the greatest opportunity for embedding secret contents; and when combined with techniques of encryption, malware injection and social engineering (such as spear phishing), steganography has created a tremendous threat to information security that, in fact, includes software security which has been mostly overlooked.

Even in tightly controlled software development environments for secure or classified systems, the engineering team encounters multimedia in various ways: in the tools used (e.g., software engineering toolboxes), in their daily routine activities (e.g., email, phones, entertainment) or multimedia generated as part of the software product (e.g., documents). To ensure a proper cleanroom environment for software engineering, we propose that multimedia steganalysis

should be part of the software IV&V focus. This ensures a higher level of security, at least for the mission-critical components of the system.

In this paper, we will focus on recent technical advancements in multimedia steganalysis accomplished using machine learning, feature mining, and pattern recognition techniques. Results of large sets of experiments on image, audio and video steganalysis demonstrate that the techniques are highly effective in detecting secret contents and are applicable to real-life applications that mandate high levels of security assurance – such as Mission Critical Software.

Static Analysis Tool Comparison with Respect to C++

Jacob Cox, NASA IV&V / TASC

ABSTRACT

A current trend in IV&V has been the inclusion of software beyond flight software, where languages and styles include more than C and the procedure style. Additionally, there are cases where flight software is also including more than procedural programming in C. C++ and Java are object-oriented languages that are becoming more common in NASA's IV&V Program analysis. The object-oriented design of these languages introduces additional challenges for the IV&V analyst particularly with the use of inheritance and polymorphism. The suite of static analysis tools used in IV&V can help in finding latent defects in source code written in C++ and this paper will discuss a comparison of the abilities, strengths and weaknesses of these tools. The tools addressed are Klocwork and Flexelint.

Performance Measurement at NASA Independent Verification and Validation

Arthur Rabeau, NASA IV&V / TASC

ABSTRACT

Performance Measurement is a process that provides quantitative data of program effectiveness and efficiency in order to support informed program decisions. This presentation describes the processes, procedures and methodology that the NASA IV&V Services Contract (ISC) uses to perform Performance Measurement. These Performance Measurement activities include planning and executing the measurements and analyses that will be used to evaluate performance and its impact on several factors, including quality, cost and progress. In addition, Performance Measurement defines not only the makeup and formulation of the measures; it defines the analysis and reporting process to interpret and act on the information received.

Assurance Cases in Planning and Execution of NASA IV&V Projects

Travis Dawson, NASA IV&V / TASC
Samuel R. Brown, NASA IV&V / KeyLogic

ABSTRACT

Evidence-based assurance, that is, providing mission and safety assurance based on documented, objective evidence, is a key goal of NASA's IV&V Program. The use of Assurance Cases is gaining momentum within the Program to help fulfill this goal. Assurance Cases are a type of structured argument that is supported by a large body of literature in industry and academia. The fundamental Assurance Case structure involves using collected evidence to support an argument that proves a claim. Evidence does not simply materialize, however – it results from execution of IV&V analysis activities that can only be properly planned by considering the end goals as captured in the intended Assurance Case. This presentation examines the integration of Assurance Cases into the existing IV&V planning process and the impact on IV&V activity execution.

Practical Assurance Case Design

Samuel R. Brown, NASA IV&V / KeyLogic

ABSTRACT

Assurance case development for a hypothetical, but realistic, spacecraft flight software functional domain is described, along with key arguments and sub-claim patterns. In addition, strategies for selecting top-level claims and organizing those claims are discussed.

The spacecraft flight software is a generalized and somewhat simplified planetary mission, with adequate information to support assurance claim development. Practical and meaningful assurance cases are developed from the software architecture, and both top-down and bottom-up methods are explored with examples. In particular, specific examples of strategies for both successful and unsuccessful assurance cases are explored.

Evidence from a variety of sources is then used to provide support to the arguments, leading to an evidence-based assurance for the spacecraft flight software domain. Examples of evidence sources and their strength are discussed.

Independent Formal Verification of Safety-Critical Systems' User Interface: A Space System Case Study

Manuel Sousa, Universidade do Minho
José Creissac Campos, Universidade do Minho / HASLab / INESC TEC
Miriam Alves, Institute of Aeronautics and Space
Michael D. Harrison, Queen Mary University of London / Newcastle University

ABSTRACT

Safe operation of safety-critical systems depends on appropriate interactions between the human operator and the computer system. Correct specification and verification of such safety-critical systems is an important mechanism for determining precisely what the software must accomplish before deployment, and this extends to the user interfaces. This paper presents a structured, comprehensive and computer-aided approach to formally specify and verify user interfaces based on model checking techniques. The paper also describes how the approach was applied to a case study whose main goal was to independently and formally model and verify the user interface of the Brazilian Institute of Aeronautics and Space's (IAE's) Satellite Launching Vehicle Testing and Preparation Ground System (TPGS). The IVY tool[1] provides support for the approach through a model editor (a component for helping in the development of user interface models); a simulation tool (a component to allow initial validation through direct interaction with the models); a properties editor (a component for helping in the expression of relevant usability related properties as logical formulae that might be verified by a model checker); and a trace visualizer/analyzer (a component for helping in the analysis of the traces that may be produced by the model checker). Independent groups were responsible for: the development of the user interface models based on the TPGS Operator Manuals; the critical and independent verification of the relevant properties of the models; and the critical assessment of the modeling process and suggestions for improvement. During the modeling activity, properties were checked to verify that the model was designed as described by the operator manuals, as well as to verify whether the system had errors or problems. A novelty of the approach is that it allows reasoning about the system's user interface being modeled as well as the quality of its operator manuals. To date, a considerable part of the two TPGS subsystems' user interfaces has been formally specified and verified and the results indicate that the modeling and analysis makes the extension of the model to accommodate the remainder of the subsystem's behavior a feasible task. Experience with this case study has shown that the proposed approach provided a practical and feasible way to systematically specify and automatically verify the user interfaces of complex systems. For most current approaches verification activities are restricted to document inspections and test results analysis, where the interpretation of the results could still be quite subjective, and test scenarios may not cover all the possible combinations of actions that can take

place. The support of a computer-aided tool was fundamental to accomplishing this activity. The overall approach represents a step forward in improving system dependability. The intention for the future is to model other TPGS's subsystems, in particular user interfaces with more complex user-machine interactions than those described herein. The IVY tool is being actively extended to make it more accessible to future developers.

References

[1] J. C. Campos and M. D. Harrison (2008) *Systematic analysis of control panel interfaces using formal tools*. In Interactive Systems: Design, Verification and Specification (DSV-IS 2008), number 5136 of Lecture Notes in Computer Science, pages 72-85. Springer.

Scalable and Flexible Static Analysis of Flight-Critical Software

Guillaume P. Brat, CMU / NASA Ames Research Center
Arnaud J. Venet, CMU / NASA Ames Research Center

ABSTRACT

Static analysis is a fully automatic technology for formally verifying critical properties of software by inspection of the source code. In the past decade, static analysis has transitioned from research lab prototypes to industrial tools used in the production chain of commercial airliners. For example, the Astrée static analyzer has been successfully applied to prove the absence of runtime errors in the fly-by-wire systems of the Airbus A340 and A380. However, in order to achieve this level of assurance on industrial applications, a static analyzer has to be specialized for a specific code or family of codes. This is an effort-intensive process requiring the attention of experts in the field. Unmanned Aircraft Systems (UAS) operations in the National Airspace System, on-board air traffic control and advanced anti-collision systems are some of the objectives of the NextGen Air Transportation System, which are implemented using vastly different types of software. In order to address these important applications, the development of specialized static analyzers needs to be streamlined. In this paper, we describe an effort in this direction initiated at NASA Ames Research Center, which is based on the development of a library of reusable static analysis components named IKOS. The core functionalities of static analysis are encapsulated in a collection of C++ classes that can be assembled to implement a static analyzer specialized for a certain application. IKOS has been successfully applied to the verification of pointer safety in UAS autopilots ranging from 33 KLOC to 270 KLOC. IKOS is also being applied to the verification of auto-generated code obtained from MATLAB®/Simulink® models, which is becoming the standard for the development of flight software. These different applications require specific adaptations of the static analysis algorithms, which can be concisely expressed with IKOS while maintaining a high level of performance.

Accelerating MS Office Work with AMF

Donald Kranz, NASA IV&V / TASC
Neal Saito, NASA IV&V / TASC
Gary Marchiny, NASA IV&V / TASC

ABSTRACT

The goal for the AMF initiative states: “The goal of this effort is to provide a set of process assets, Catalog of Methods inputs, and educational materials from existing IV&V tools and resources to develop IV&V Program concepts and definitions of evidence; to build a framework for organizing and communicating evidence; and to create methods for refining IV&V assurance data into information and assurance statements.” One of the most commonly available tools utilized by IV&V is the MS Office suite of products. The AMF provides a generalized, repeatable, and efficient method for working with the plethora of documents and a SQL database to capture analysis work. A VBA library implementing the AMF user interface, business and data layers is available to help the analyst extract information from word and excel, as well as report back to any MS Office product from analysis work performed. This session will review some of these libraries that are available for MS Excel and MS Word.

NATO IV&V: Work in Progress

Brad Bigelow, NATO Communications and Information Agency (NCIA)
Arend Smit, Ph.D., M&I/Partners

ABSTRACT

NATO has a history of applying IV&V to its IT projects. Dating back to the 1980s, IV&V involvement still continues to this day. We present an overview of the dynamics of NATO's experience with IV&V in the past.

In the first years of NATO IV&V, the involvement of IV&V activities started after the award of the main project. Usually, the contract was based on an initial requirements analysis. In a number of cases, IV&V contractors noted that the project was marred by issues spawned in the pre-contract award phase. Such issues then necessitated contract amendments, sometimes requiring lengthy (re)negotiations, all too often associated with increased project cost and delayed delivery. Faced with this situation, NATO was increasingly willing to fund IV&V activities even before contract award, during the preparation of the Invitation for Bid. This involvement of IV&V at earlier stage in the project tended to reduce the number of contract amendments required during the project's lifecycle.

More recently, NATO has expanded the range of IV&V options available to projects. The original approach involved the procurement of IV&V services under separate IV&V contracts, with dedicated funding. Now, project managers in one of NATO's major capability enhancement programmes can also recruit IV&V services from a central programme support unit. As this type of IV&V activity does not require a separate IV&V contract, it allows more malleable customization of the project's IV&V involvement.

TraceLab Player: From Researchers to IV&Vers

Jane Hayes, University of Kentucky
Jane Huang, University of Kentucky
Adam Czuaderna, University of Kentucky

ABSTRACT

The TraceLab framework has now been demonstrated to assist researchers in quickly developing and validating new ideas related to requirements traceability. At Traceability of Emerging Forms of Software Engineering (TEFSE) in May 2013, applications of TraceLab ranged from feature location to semantic-enhanced tracing. In fact, some researchers used TraceLab to address problems beyond traceability such as generation of software test cases. The latest step in the TraceLab evolution has been to develop a TraceLab player, permitting IV&V agents to take a technique that has been developed and validated in TraceLab and "grab it" for use as a stand-alone "tool." Basically, the IV&V agent edits a batch file to indicate the name and location of input and output files and then runs the technique (such as tracing using Latent Dirichlet Allocation (LDA)). This presentation will demonstrate how IV&V agents can use the TraceLab player.

Use of a Technical Reference in NASA Independent Verification and Validation

Travis Dawson, NASA IV&V / TASC

ABSTRACT

Providing mission assurance for NASA systems, and other customer systems under development, requires documentation of the system under evaluation and the standards and criteria against which the system is evaluated. These two categories taken together are called the Technical Reference, and are being used at NASA's IV&V Program to support evidence-based assurance of customer systems. This presentation describes the intent of the Technical Reference, describes typical contents and usage, and present example products in use.

Model-Based Testing of Spacecraft Flight Software

Maria Hernek, European Space and Technology Center (ESTEC)

ABSTRACT

Software Verification & Validation (V&V) is an important part of the software development lifecycle. Integration test cases for the Flight Software (FSW) are traditionally designed manually based on requirements written in natural language (i.e., black box testing). Unit test cases are traditionally designed manually based on detailed design and also based on the knowledge of the corresponding source code (i.e., white box testing). Due to the high criticality and complexity of the spacecraft FSW, the requirements for V&V are very stringent and usually the manual validation, including both unit testing and integration testing, takes a significant effort.

Model-based testing (MBT) aims at the automation of the design of V&V tests. The difference from the usual black-box validation testing is that rather than manually writing tests based on the requirements documentation, the test implementation is derived from a formal model of the expected system under test (SUT), which captures the functional requirements of the SUT.

The European Space Agency (ESA) is running an activity to evaluate automatic MBT in the context of spacecraft FSW. The main objectives of this study are to evaluate the applicability, scalability, efficiency and cost-effectiveness of a specific MBT method, namely sequence-based specification (SBS) combined with statistical based testing (SBT).

The scope in terms of tasks includes:

- Create a formal model of an existing spacecraft FSW system based on system or software requirements in natural language.
- The resulting formal models are refined into test models which are used for automatic test case generation.
- The generated test cases are integrated into the Software Verification Facility (SVF). Tests are executed and analysed (outcome, time, effort, and coverage).

This presentation reports on the results of this activity.

Model-Based Testing of NASA Systems

Dharmalingam Ganesan, Fraunhofer Center for Experimental Software Engineering (CESE)

ABSTRACT

Software testing is not only expensive but also often not rigorous enough, thus bugs can slip into the field with costly consequences. This is often due to the manual nature of the testing process. In recent years, several software test case execution frameworks (e.g., JUnit) have been developed and used in many organizations. Such frameworks are very helpful to automatically run the test cases during nightly builds. They also help the programmers verify their source code modifications (a.k.a. regression testing). However, the job of designing the test cases is outside the scope of these test execution frameworks. Programmers (or testers) have to construct test cases manually. To overcome this limitation, model-based testing (MBT), which is a technique that derives test cases from an explicit behavioral model, has been proposed. MBT is gaining popularity in the research community and recently in industry, too.

When a software implementation already exists, models of the implementation's expected behavior can be used to generate an innumerable number of test cases. As part of the NASA's Software Assurance Research Program (SARP), researchers at Fraunhofer have developed methods and tools for analyzing models and automatically generating test cases from the model of the expected behavior. The generated test cases are automatically converted into appropriate test execution frameworks such as JUnit, CUnit, etc. Our methods and tools have been applied on several NASA systems. For example, core functions of NASA's Goddard Mission Services Evolution Center (GMSEC) framework were tested using MBT. The same model was used to test several programming languages and middleware technologies that are supported by the GMSEC framework. Similarly, file system APIs of the NASA's OS abstraction layer were tested using MBT, too. In all endeavors, we were successful in detecting previously unknown errors, even in systems that are used in production. Not to mention the fact that several requirements-level issues such as contradictions and incompleteness were detected, too. Work is underway at JPL to apply MBT to test parts of the Soil Moisture Active Passive (SMAP) mission functions.

We use "lightweight" state machines as the modeling notation to kick-off the MBT process. This process requires state machines to be hand-drawn by the testers/engineers. When the number of states grows due to inherent complexity of the system under test, it will be laborious and sometimes error-prone to hand-draw all states and transitions. To scale up MBT, we apply advanced modeling notations and tools (e.g., SpecExplorer) that are capable of automatically generating state machines from a model program, which is similar to a regular program but

enriched with a few modeling constructs. The tester develops a model program, which is usually much simpler than the real system under test because several details are abstracted in the model program. From the model program, different state machines and test cases for different scenarios are automatically derived and tested against the system under test.

Inputs to our methods include requirements, specifications, example usages of the system under test and existing test cases. The output includes requirement issues, a suite of ready-to-run test cases that can be integrated to the build process. Model construction is a manual process. However, model analysis, test case generation and execution are fully automatic. As a return on investment, organizations will get requirements issues (if any), ready-to-run test cases, models, and bugs (if any) of applying model-based testing. In this presentation, we will present our experiences of applying MBT on mission-critical systems.

Architecture of the AMF

Donald Kranz, NASA IV&V / TASC
Tom Gullion, NASA IV&V / TASC
Neal Saito, NASA IV&V / TASC
Gary Marchiny, NASA IV&V / TASC

ABSTRACT

The Analysis and Management Framework (AMF) is a three-tier, evidence-based assurance architecture used to support verification and validation efforts. This session explains the origin and principles used to develop the architecture. Those interested in understanding the basics of the design of the AMF will find this session useful. Passing business objects between different technologies allows sharing of information across platforms. Domain model elements supporting versioning and polymorphic behavior provide a platform for expanding upon the successes of previous projects. Participants that become aware of the AMF's capabilities and rationale can more efficiently implement the AMF's available functionality based on their unique project needs.

IV&V Techniques for Robotics on OSIRIS-REx

Charles R. Price, NASA IV&V / TASC
Ricky Forquor, NASA IV&V
David Turner, NASA IV&V / NEW-BOLD Enterprises

ABSTRACT

This paper will describe recently developed IV&V techniques for the Touch and Go Sample Acquisition Mechanism (TAGSAM) robotic subsystem on the Origins Spectral Interpretation Resource Identification Security - Regolith Explorer (OSIRIS-Rex) spacecraft. The techniques include initial assessment of MRD materials, scale model development, kinematic animation development, facilitated discussions, assessment of the Japan Aerospace Exploration Agency (JAXA) Hayabusa mission, review of the NASA IV&V Program's OSIRIS-Rex Portfolio Based Risk Assessment (PBRA), development of scenarios based on the Design Reference Mission, development of off-nominal branch scenario paths, assessment of developer's fault trees, providing process assets for the NASA IV&V Program's Catalog of Methods and capturing of these techniques into an accessible robotics wiki.

Evaluating the *t*-way Combinatorial Technique for Determining the Thoroughness of a Test Suite

Charles R. Price, NASA IV&V / TASC
Ricky Forquer, NASA IV&V
Adelbert Lagoy, NASA IV&V
D. Richard Kuhn, National Institute of Standards and Technology (NIST)
Raghu N. Kacker, NIST

ABSTRACT

An innovative technique developed by NIST that determines the thoroughness of a test suite by measuring the *t*-way combinations of input and configuration variables is being evaluated as an IV&V capability development task. This evaluation of the *t*-way technique targets the testing done on the Global Precipitation Measurement (GPM) mission heater control software by its developer and subsequently analyzed by IV&V.

Automated Design-Time Analysis for the GOES-R System

David Hall, NASA Ames Research Center
Corina Pasareanu, NASA Ames Research Center

ABSTRACT

The National Oceanic and Atmospheric Administration (NOAA) operates a system of Geostationary Operational Environmental Satellites (GOES) to provide continuous weather imagery and monitoring of meteorological and space environment data to protect life and property across the United States. Two GOES satellites remain operational at all times providing coverage for the eastern United States and most of the Atlantic Ocean and the western United States and Pacific Ocean basin. The NOAA/NASA GOES-R project represents the next generation of GOES. It provides continuity of the GOES mission and improvement of its remotely sensed environmental data. The system will be software intensive and will consist of many interacting components.

The GOES-R system must be extremely reliable and correct. Testing is typically used to ensure software reliability. However, testing is often manual and time-consuming, and it is used late in the software life cycle, after the code has been written and when it is very expensive to fix discovered interaction errors.

We describe an effort at NASA Ames that performs systematic analysis of a portion of the state transition behavior of the ground segment of the GOES-R system using design-time information. Identifying and correcting errors at design time is typically easier and more cost-effective. But even if the system is already implemented, behavior identified from the design specifications can be used to guide testing and assess completeness of the test cases. Various design components of the GOES-R ground segment have been modeled using MATLAB®'s Stateflow® notation, which have been automatically translated into an intermediate executable representation using NASA Ames' Polyglot system. Automated test case generation and verification with respect to prescribed requirements have been performed using NASA Ames' PathFinder tool-set.

IV&V Guidance for IV&V for Product Line Software

Charles R. Price, NASA IV&V / TASC

ABSTRACT

Product line software is a series of software deliveries produced by a common software developer for use in different space mission applications, where succeeding software deliveries contain heritage components that are reused or modified components from previous deliveries.

This paper will describe the recently developed *Guide for IV&V of Product Line Software* that resulted from a capability development task that reviewed all NASA IV&V Program heritage reports and sought crowdsourced information, commentary and opinions from the NASA IV&V community.

Assurance Cases for Software Releases in ISS Sustaining Phase of Development

Sarma Susarla, NASA IV&V / TASC

ABSTRACT

The presentation gives an overview of the development process for new software releases during sustaining phase of International Space Station (ISS) and how IV&V interfaces with the development throughout the life cycle to analyze the various artifacts in performing Computer Software Configuration Item's (CSCI's) requirement review, design/code review, test review, software integration review, and software on-orbit transition review. The presentation describes how assurance cases are developed in each of those reviews culminating in the final assurance case that the software meets mission objectives for on-orbit transition

Applying NASA-STD-7009 Standard to Models and Simulations

Darilyn Dunkerley, NASA IV&V / TASC

ABSTRACT

NASA's *Standard for Models and Simulations* (NASA-STD-7009), created in response to the Columbia Shuttle Accident, addresses the dynamic development environment associated with projects developing model-based software and systems. The Columbia Accident Investigation Board (CAIB) observed that development projects that employ models and simulations (M&S) suffer from complexity of configuration management, since both the models/simulations as well as the modeled systems and software components themselves are under constant revision. M&S techniques are increasingly applied in the development projects which receive NASA's IV&V Program analysis services. This paper documents key concepts from NASA-STD-7009 to assist analysts in evaluating the extent to which a given development project's M&S artifacts (which include safety-critical components) comply with this emergent standard.

Data-Driven IV&V Decision Support

Chris Williams, NASA IV&V / TMC

ABSTRACT

NASA's IV&V Program produces various types of data in support of its activities, and this data is not always able to be used in an effective manner to support decision-making at both the lower levels and the higher management levels. The NASA IV&V Program's Software Assurance Tools (SWAT) team has been working with others in the Program (e.g., IVVO Data Management Planning, TQ&E) to prepare a foundation for data capture and use that will provide a more robust decision support capability.

Some of the key data related items that will be addressed during this presentation include (1) data availability, (2) data presentation, (3) ad hoc reporting, (4) metrics, and (5) IV&V decision support.

Some specific examples that may be addressed in this topic include:

- JIRA/Confluence gadgets and plugins
- 'ORBIT Snapshot' data
- COMPASS tool (IV&V method utilization, technical guidance for Technical Scope & Rogor (TS&R) creation)
- SQL Server reporting services (ad hoc reporting)
- Static code and dynamic analysis metrics
- On-orbit issue anomaly data
- REST web services (e.g., RiskManager tool data access)

Modeling the Image-Processing Behavior of the NASA Voyager Mission with ASSL

Emil Vassev, University College Dublin
Mike Hinchey, University of Limerick

ABSTRACT

NASA exploration missions increasingly rely on the concepts of autonomic computing, exploiting these to increase the survivability of remote missions, particularly when human tending is not feasible. This paper presents initial results of long-term research targeted at the design and implementation of prototype models for future Voyager-like missions that rely on principles of autonomic computing. Here, we employ the Autonomic System Specification Language (ASSL) to build a formal model and to generate a prototype for the image-processing behavior of the NASA Voyager Mission. This helps to validate existing features and perform experiments through simulation. Moreover, this prototype lays the basis for future experiments whereby autonomic features are added in a stepwise manner.

1. INTRODUCTION

There are few engineering activities as complex as the effective design, construction, and maintenance of the spacecraft employed in exploration missions. Autonomic Computing (AC) has emerged as a promising approach to the development of large-scale self-managing complex systems [1]. The general idea of AC is the handling of complexity in computer systems through self-management based on high-level objectives.

The building blocks of AC systems are architectural components called autonomic elements (AEs) [1, 2]. In general, an AE extends programming elements (i.e., objects, components, services) to define a self-contained computational unit with specified interfaces and explicit context dependencies. Essentially, an AE encapsulates rules, constraints and mechanisms for self-management, and can dynamically interact with other AEs. From a more applied perspective, AC builds upon existing technology, with the goal of developing management capabilities that can be applied to both new and legacy systems.

NASA is approaching AC with interest, recognizing in its concepts a bridge towards “the new age of space exploration” where spacecraft should be independent, autonomous, and “smart” [1]. Both the Autonomous Nano-Technology Swarm (ANTS) concept mission [3, 4] and the Deep Space One (DS1) mission [1] represent the new generation of AC-based unmanned missions. AC software makes spacecraft autonomic systems capable of planning and executing many activities onboard the spacecraft to meet the requirements of changing objectives and harsh external conditions.

We investigate some hypotheses regarding the design and implementation of future Voyager-like missions that incorporate some of the principles of AC. Our objective is to build prototype software models that help in the comparison of features and issues of the actual Voyager mission with hypothesized possible autonomic approaches, thus giving significant benefits to the development of future space-exploration systems. To realize these goals, we experiment with ASSL (Autonomic System Specification Language) [5], an AC-dedicated framework providing a powerful formal notation and computational tools to help AC researchers with problem formation, system design, system analysis and evaluation, and system implementation. However, to improve the efficiency of the autonomic-features modeling, our next step is to use requirements engineering to capture the necessary requirements identifying autonomic features.

The rest of this paper is organized as follows. In Section 2, we present the Voyager Mission together with our research objectives and goals. Section 3 describes how we used ASSL to specify autonomic features and generate a prototype of the NASA Voyager Mission. Section 4 presents experimental results, and finally, Section 5 concludes the paper with the benefits for space missions and a discussion on autonomy requirements engineering along with concluding remarks and future work.

2. RESEARCH OBJECTIVES

The great success of the NASA Voyager Mission, designed and built over 30 years ago, and the fact that autonomous behavior persists in the Voyager requirements, make the same a good example for future space missions. Here, it is our understanding that both prototyping and formal modeling, which will aid in the design and implementation of future Voyager-like missions,

are becoming increasingly necessary and important as the urgent need emerges for higher levels of assurance regarding correctness.

2.1. The NASA Voyager Mission

The NASA Voyager Mission [6] was designed for exploration of the Solar System. The mission started in 1977, when the twin spacecraft Voyager I and Voyager II were launched (see Figure 1). The original mission objectives were to explore the outer planets of the Solar System. As the Voyagers flew across the Solar System, they took pictures of planets and their satellites and performed close-up studies of Jupiter, Saturn, Uranus, and Neptune.

Figure 1. Voyager spacecraft [6]

After successfully accomplishing their initial mission, both Voyagers are now on an extended mission, dubbed the "Voyager Interstellar Mission". This mission is an attempt to chart the heliopause boundary, where the solar winds and solar magnetic fields meet the so-called *interstellar medium* [7]. All this makes Voyager the most successful planetary exploration mission of all time. This success is due to the fact that:

- The spacecraft were designed and implemented rigorously, and as a result both are still "healthy" today;
- NASA engineers designed the spacecraft hardware "for the long haul", by installing a system that allows for enhanced remote control programming "to give the spacecraft even greater capability than they possessed when they left Earth".

In the course of this research, we explored the image-processing system implemented on board the Voyager spacecraft. In order to take pictures, Voyager II carries two television cameras on board – one for wide-angle images and one for narrow-angle images, where each camera records images with a resolution of 800x800 pixels. Both cameras can record images in black-and-white only, but each camera is equipped with a set of color filters, which helps in the reconstruction of images be as fully-colored ones.

To transmit pictures to Earth, Voyager II uses its 12-foot dish antenna (see Figure 1) to send streams of pixels. It uses the same microwave frequencies used for radar. However, due to the long distance and to fundamental laws of physics, the strength of the radio signal is diminished proportionally and it reaches antennas on Earth with a strength 20 billion times weaker [8]. To counter this, the signals are received by a network of enormous antennas located in Australia, Japan, California, and Spain. Next, all the faint signals received from Voyager II are combined and processed by the Voyager Mission base on Earth to reduce electronic noise, blend, and filter the composed pictures.

2.2. Our Research Objectives

Our long-term objectives are the modeling and implementation of autonomic system prototypes of future Voyager-like missions, thus allowing for benchmark experiments to compare prototyped autonomic features and issues with the actual Voyager Mission. To achieve these goals, we intend to apply ASSL to build formal models and generate functional prototypes for the Voyager mission. The generated prototypes will help us to validate the features in question and perform further investigations based on practical results under simulated conditions. Note that knowledge of the Voyager Mission enables us to compare issues arising in the mission itself with potential approaches to their mitigation.

Here, our first objective is to specify with ASSL and generate a prototype model for the *image-processing behavior* observed in the NASA Voyager mission. Note that while exploring the Solar System, the Voyagers were able to detect interesting objects and take pictures of the same on-the-fly. This reveals a form of autonomic event-driven behavior, which we specify with / ASSL.

3. RESEARCH

This research is centered around the ASSL framework. We use ASSL to specify the Voyager Mission in a stepwise manner (feature by feature) and generate a series of prototypes, which we evaluate in simulated conditions. The latter are usually modeled as events that trigger special autonomic policies in the generated prototypes. We evaluate the behavior of the generated prototypes through special log records produced by any ASSL-generated application. These log records inform us about important state-transition operations allowing us to trace the behavior of the prototype in question.

3.1. ASSL

In general, ASSL considers ASs as composed of AEs interacting over interaction protocols. To specify autonomic systems, ASSL uses a multi-tier specification model [5] that is designed to be scalable and to expose a judicious selection and configuration of infrastructure elements and mechanisms needed by an AS. The ASSL tiers are abstractions of different aspects of the AS under consideration, such as *self-management policies*, *communication interfaces*, *execution semantics*, *actions*, etc. There are three major tiers (three major abstraction perspectives), each composed of sub-tiers (see Figure 2):

- AS tier — forms a general and global AS perspective, where we define the general system rules in terms of *service-level objectives (SLO)* and *self-management policies*, *architecture topology*, and global *actions*, *events*, and special *metrics* applied in these rules.
- AS Interaction Protocol (ASIP) tier — forms a perspective that defines the means of communication between AEs. The ASIP tier is composed of *channels*, *communication functions*, and *messages*.
- AE tier — forms a unit-level perspective, where we define interacting sets of individual AEs with their own behavior. This tier is composed of *AE rules* (SLO and self-management policies), an *AE interaction protocol (AEIP)*, *AE friends* (a list of AEs forming a circle of trust), *recovery protocols*, special *behavior models* and *outcomes*, *AE actions*, *AE events*, and *AE metrics*.

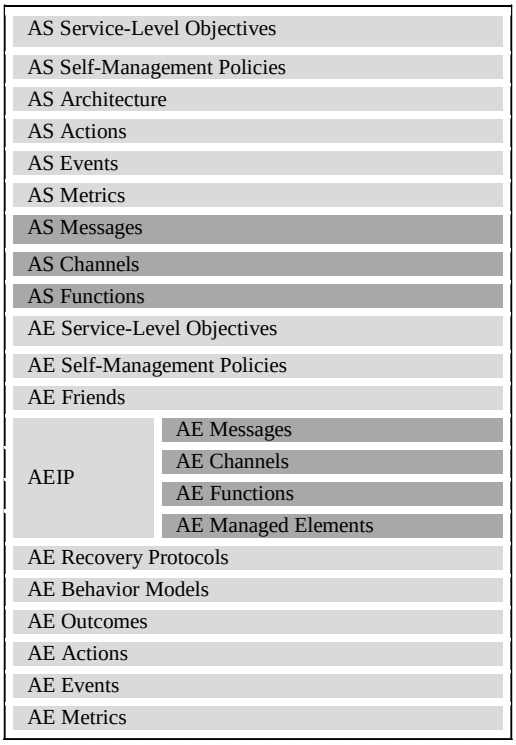

Figure 2. ASSL multi-tier specification model.

3.2. Specifying and Generating Prototypes with ASSL

The ASSL tiers are intended to specify different aspects of the AS in question, but it is not necessary to employ all of those in order to model an AS. Usually, an ASSL specification is built around self-management policies, which make that specification AC-driven. The ASSL formal model addresses policy specification at both AS and AE tiers. Policies are specified with special constructs called *fluents* and *mappings*:

- Fluents are states with duration and when the system gets into a specific fluent, a policy may be activated.
- Mappings map particular fluents to particular actions to be undertaken by the specified AS.

ASSL expresses fluents with *fluent-activating* and *fluent-terminating* events, i.e., the self-management policies are driven by events. In order to express mappings, conditions and actions are considered, where the former determine the latter in a deterministic manner.

The following ASSL code presents an example specification of a self-healing policy. The interested reader is advised to consult [6] for more details on the ASSL specification model and grammar.

```
ASSELF_MANAGEMENT {
    SELF_HEALING {
        FLUENT inLosingSpacecraft {
            INITIATED_BY { EVENTS.spaceCraftLost }
            TERMINATED_BY { EVENTS.earthNotified }
        }
        MAPPING {
            CONDITIONS { inLosingSpacecraft  }
            DO_ACTIONS { ACTIONS.notifyEarth }
        }
    }
}
```

Once a specification is complete, it can be validated with the ASSL built-in consistency checking mechanism and a functional prototype can be generated automatically. The prototypes generated with the ASSL framework are fully-operational multithreaded event-driven applications with embedded messaging.

3.3. Voyager Image-Processing Behavior Algorithm

An autonomous-specific behavior is observed in the Voyager spacecraft when a picture must be taken and sent to Earth (see Section 2.1). The following elements describe the algorithm we applied to specify the image-processing behavior observed in the Voyager mission with ASSL.

1) The Voyager II spacecraft:
   - 1.1) uses its cameras to monitor space objects and decide when it is time to take a picture;
   - 1.2) takes a picture with its wide-image camera or with its narrow-image camera;
   - 1.3) notifies the antennas on Earth with “image session start” messages that an image transmission is about to start;
   - 1.4) applies each color filter and sends the stream of pixels for each filter to Earth;
   - 1.5) notifies antennas on Earth for the end of each session with “image session end” messages.
2) The antennas on Earth:
   - 2.1) are prompted to receive the image by the “image session start” messages (one per applied filter);
   - 2.2) receive image pixels;
   - 2.3) are prompted to terminate the image sessions by “image session end” messages;
   - 2.4) send the collected images to the Voyager Mission base on Earth.
3) The Voyager Mission base on Earth receives the image messages from the antennas.

4. RESEARCH RESULTS

In the course of this project, we successfully specified the image-processing behavior of the NASA Voyager Mission with ASSL. Here we applied the ASSL multi-tier specification model [5] to specify the Voyager II Mission as an autonomic system (AS) composed of the Voyager II spacecraft and four antennas on Earth, all specified as distinct AEs. Next, we generated the Java application skeleton for the prototype of the Voyager II Mission and experimented with it to explore important state-transition operations ongoing in the system at run-time and to trace the behavior of the generated system.

4.1. Specifying Voyager Mission with ASSL

In order to specify the algorithm described in Section 3.3, we applied the ASSL multi-tier specification model and specified the Voyager II Mission at the three main ASSL tiers – AS (autonomic system) tier, ASIP (autonomic system specification protocol) tier, and AE (autonomic element) tier (see Section 3.1). Hence, in our specification, we specified the Voyager II spacecraft and the antennas on Earth as AEs that follow their encoded autonomic behavior and exchange predefined ASSL messages over predefined ASSL communication channels. The Voyager mission autonomic behavior is specified at both AS and AE tiers as a self-management policy called **IMAGE_PROCESSING**. Thus, the global autonomic behavior of the Voyager II Mission is determined by the specification of that policy at each AE and at the global AS tier.

Due to space limitations, we cannot present the entire specification, which is rather long (over 1100 lines of ASSL code). An appendix is added at the end of this paper to present the full ASSL specification (see Appendix A).

4.1.1. AS Tier Specification

At this tier, we specified the global AS-level autonomic behavior of the Voyager Mission. This behavior is encoded in the specification of an **IMAGE_PROCESSING** self-management policy. At this tier, that policy specifies an image-receiving process taking place at the four antennas on Earth (located in Australia, Japan, California, and Spain). In fact, as specified at the AS Tier, this policy forms the autonomic image-processing behavior of the Voyager Mission base on Earth.

Here, we specified four "**inProcessingImage**_" fluents (one per antenna), which are initiated by events prompted when an image has been received, and terminated by events prompted when the received image has been processed (see Appendix A). Further, all the four fluents are mapped to a **processImage** action. The following specification sample shows a fluent specification together with its mapping:

```
FLUENT inProcessingImage_AntSpain  {
    INITIATED_BY { EVENTS.imageAntSpainReceived }
    TERMINATED_BY { EVENTS.imageAntSpainProcessed }
}
MAPPING {
    CONDITIONS { inProcessingImage_AntAustralia}
    DO_ACTIONS {ACTIONS.processImage("Antenna_Australia") }
}
```

Here, the specification of the events that initiate and terminate that fluent is the following:

```
EVENT imageAntSpainReceived {
    ACTIVATION  {
        RECEIVED { ASIP.MESSAGES.msgImageAntSpain }
    }
 }
EVENT imageAntSpainProcessed { }
```

Note that the **processImage** action is an **IMPL** action [5], i.e., it is a kind of abstract action that does not specify any statements to be performed [6]. The ASSL framework considers the **IMPL** actions as "to be manually implemented" after code generation. The following is a partial specification of that action:

```
ACTION IMPL processImage {
    PARAMETERS { string antennaName }
    GUARDS {
        ASSELF_MANAGEMENT.OTHER_POLICIES.
        IMAGE_PROCESSING.inProcessingImage_AntAustralia
            OR
        ASSELF_MANAGEMENT.OTHER_POLICIES.
        IMAGE_PROCESSING.inProcessingImage_AntJapan
            …
    }
    TRIGGERS {
        IF  antennaName  =  "Antenna_Australia"  THEN
            EVENTS.imageAntAustraliaProcessed
        END ELSE  …
    }
}
```

Here, the **processImage** action is specified to accept a single parameter. The latter allows that action to process images from all four antennas. Moreover, there is a special **GUARDS** clause that is specified to prevent execution of the action when none of the

four fluents is initiated. The action triggers an **imageAnt**[antenna name]**Processed** event if the action is performed with no exceptions.

4.1.2. ASIP Tier Specification

At this tier, we specified the AS-level communication protocol – the autonomic system interaction protocol (ASIP) (see Section 3.1). This communication protocol is specified to be used by the four antennas when these communicate with the Voyager Mission base on Earth. Here, at this tier we specified four *image messages* (one per antenna), a communication channel that is used to communicate these messages, and communication functions (e.g., **sendImageMsg** and **receiveImageMsg**; see Appendix A) to send and receive these messages over that communication channel. Note that the communication functions accept a parameter that allows same communication functions to send or receive messages to and from different antennas. Please refer to Appendix A for the ASSL specification of the Voyager ASIP.

4.1.3. AE Tier Specification

At this tier, we specified five AEs. The Voyager II spacecraft and all four antennas on Earth (the antennas located in Australia, Japan, California, and Spain), are specified as AEs. Note that here, we specified the **IMAGE_PROCESSING** self-management policy at the level of single AE and thus, this policy is realized over all AEs specified for the Voyager Mission.

In this sub-section, we present important details of this specification. Please, refer to Appendix A for the complete AE Tier specification.

AE Voyager. The most complex AE is the one specified for the Voyager II spacecraft. To express the **IMAGE_PROCESSING** self-management policy for this AE, we specified two fluents: **inTakingPicture** and **inProcessingPicturePixels**. The following ASSL listing presents that self-management policy with both fluents and their mapping sections.

```
AESELF_MANAGEMENT {
    OTHER_POLICIES {
        POLICY IMAGE_PROCESSING {
            FLUENT inTakingPicture {
                INITIATED_BY { EVENTS.timeToTakePicture }
                TERMINATED_BY { EVENTS.pictureTaken }
            }
            FLUENT inProcessingPicturePixels {
                INITIATED_BY { EVENTS.pictureTaken }
                TERMINATED_BY { EVENTS.pictureProcessed }
            }
            MAPPING {
                CONDITIONS { inTakingPicture }
                DO_ACTIONS { ACTIONS.takePicture }
            }
            MAPPING {
                CONDITIONS { inProcessingPicturePixels }
                DO_ACTIONS { ACTIONS.processPicture }
            }
        }
    }
} // AESELF_MANAGEMENT
```

Here, the **inTakingPicture** fluent is initiated by a **timeToTakePicture** event and terminated by a **pictureTaken** event. This event also initiates the **inProcessingPicturePixels** fluent, which is terminated by the **pictureProcessed** event. Both fluents are mapped to the actions **takePicture** and **processPicture** respectively.

In addition, we specified an AEIP (autonomic element interaction protocol) (see Section 3.1), which is used by the Voyager AE to communicate with the four antenna AEs and to monitor and control the two cameras (wide-image camera and narrow-image camera) on board. Thus, with this AEIP we specify (see Appendix A):

- ASSL messages needed to send an image pixel and messages that notify the antenna AEs that an image-receiving session is about to begin or end;
- a private communication channel;
- three communication functions that send the AEIP messages over the AEIP communication channel.
- Two special managed elements (termed **wideAngleCamera** and **narrowAngleCamera**) to specify interface functions needed by the Voyager AE to monitor and control both cameras. Through their interface functions, both managed elements are used by the actions mapped to the fluents **inTakingPicture** and **inProcessingPicturePixels** to take pictures, apply filters, and detect interesting space objects.

The following specification sample shows a partial specification of one of these managed elements.

```
MANAGED_ELEMENT wideAngleCamera {
    INTERFACE_FUNCTION takePicture { }
    …
    INTERFACE_FUNCTION countInterestingObjects  {
        RETURNS { integer }
    } }
```

Moreover, an **interestingObjects** metric is specified to count all detected interesting objects, which the Voyager AE takes pictures of. The source of this metric is specified as one of the managed element interface functions (**countInterestingObjects**); i.e., the metric gets updated by that interface function.

```
METRIC interestingObjects  {
    METRIC_TYPE {  RESOURCE          }
    METRIC_SOURCE {  AEIP.MANAGED_ELEMENTS.
        wideAngleCamera.countInterestingObjects  }
    THRESHOLD_CLASS  {  integer [ 0~ )  }
}
```

Note that the **timeToTakePicture** event (it activates the **inTakingPicture** fluent) is prompted by a change in this metric's value. Here, in order to simulate this condition, we also activate this event every 60 seconds on a periodic basis.

```
EVENT timeToTakePicture  {
    ACTIVATION  {
        CHANGED  {  METRICS.interestingObjects }
            OR
        PERIOD {  60 SEC }
     }
}
```

The four antenna AEs are specified as friends (at the **FRIENDS** sub-tier) of the Voyager AE. According to the ASSL semantics [5] friends can share private interaction protocols. Thus, the antenna AEs can use the *messages* and *channels* specified by the AEIP of the Voyager AE.

Antenna AEs. We specified the four antennas receiving signals from the Voyager II spacecraft as AEs, i.e., we specified AEs termed **Antenna_Australia**, **Antenna_Japan**, **Antenna_California**, and **Antenna_Spain**. Here, the **IMAGE_PROCESSING** self-management policy for these AEs is specified with a few pairs of **inStartingImageSession - inCollectingImagePixels** fluents. A pair of such fluents is specified per image filter and determines states of the antenna AE when an image-receiving session is starting and when the antenna AE is collecting the image pixels.

Because the Voyager AE processes the images by applying different filters and sends each filtered image separately, we specified for each applied filter different fluents in the antenna AEs (see Appendix A for the complete **IMAGE_PROCESSING** specification at the antenna AEs). This allows an antenna AE to process a collection of multiple filtered images simultaneously. Note that according to the ASSL formal semantics, a fluent cannot be re-initiated while it is initiated, thus preventing the same fluent be initiated simultaneously twice or more times [5].

Here, these fluents are initiated and terminated by AE events specified to be prompted by the Voyager AE's messages notifying that an image-receiving session begins or ends. The following partial specification shows two of the **IMAGE_PROCESSING** fluents. These fluents are mapped to AE actions that collect the image pixels per filtered image.

```
FLUENT inStartingGreenImageSession {
    INITIATED_BY { EVENTS.greenImageSessionIsAboutToStart }
    TERMINATED_BY { EVENTS.imageSessionStartedGreen }
}
FLUENT inCollectingImagePixelsBlue {
    INITIATED_BY { EVENTS.imageSessionStartedBlue }
    TERMINATED_BY { EVENTS.imageSessionEndedBlue }
}
```

In addition, an **inSendingImage** fluent is specified. This fluent activates when the antenna AE is done with the image collection work, i.e., all the filtered images (for all the applied filters) have been collected. The fluent is mapped to a **sendImage** action that sends the filtered images as one image to the Voyager Mission base on Earth.

The following listing presents two of the events used to initiate those fluents.

```
EVENT greenImageSessionIsAboutToStart {
    ACTIVATION { SENT { AES.Voyager.AEIP.MESSAGES.
                        msgGreenSessionBeginAus } }
}
EVENT imageSessionStartedBlue {
    ACTIVATION { RECEIVED { AES.Voyager.AEIP.MESSAGES.
                    msgBlueSessionBeginAus } }
}
```

Note that the **greenImageSessionIsAboutToStart** event is prompted when the Voyager's **msgGreenSessionBeginSpn** message has been sent, and the **imageSessionStartedBlue** event is prompted when the Voyager AE's **msgBlueSessionBeginSpn** message has been received by the antenna.

Moreover, each antenna AE specifies communication functions that allow the AE receives the Voyager AE's messages (see Appendix A). These communication functions are called by the AE actions.

4.2. Structure and Behavior of the Voyager Prototype

In this endeavor, we experimented with the prototype generated from the ASSL specification of the Voyager II Mission. Our goal was to demonstrate that the ASSL-generated prototype is capable of self-managing in respect of the specified with ASSL self-management policies.

4.2.1. Structure

With ASSL we generated a Voyager prototype that is a pure software solution; i.e., the Voyager spacecraft and the four antennas were implemented as interacting components embedded in a Java application. It is important to mention that instead of generating a monolithic application, the ASSL framework strives to organize the generated ASs in a granular fashion. Thus, at runtime, an ASSL-generated prototype has a *multi-granular structure* composed of instances (objects) of the specified tiers in the ASSL specification of the Voyager Mission. Here all tier instances together form the *runtime object model* of the Voyager's prototype (see Figure 3). Similar to the applied ASSL specification model (see Section 3.1), the prototype's runtime object model has a somewhat hierarchical composition where sub-tier instances are grouped around instances of major tiers. Figure 3(a) depicts the runtime object model of a Voyager prototype generated with ASSL and Figure 3(b) presents a runtime object model for an AE generated for that prototype. Note that both Figure 3(a) and Figure 3(b) present generic object models. Thus, concrete models have an arbitrary number and types of nodes derived from their corresponding ASSL specification.

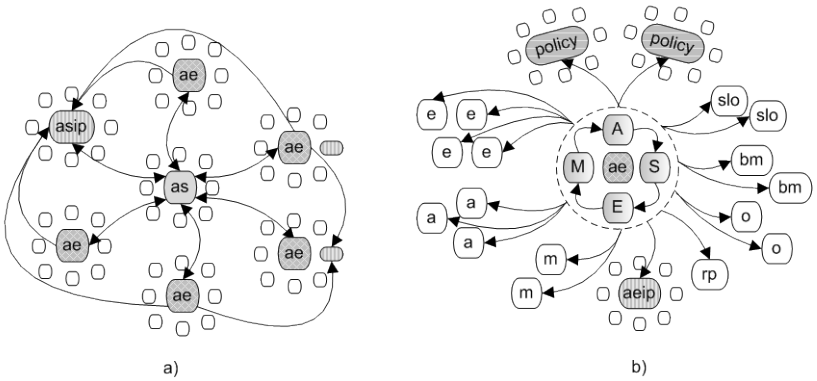

Figure 3. (a) AS Runtime Object Model; (b) AE Runtime Object Model

Figure 3(b) presents the granular structure of an AE object model. Here, at the core of the AE we can see four objects forming a special AE control loop. As depicted, the latter is composed of the objects M (monitor), A (analyzer), S (simulator), and E (executor). ASSL generates these objects to provide a sort of control over the autonomic behavior of the AE [5].

4.2.2. Behavior

Due to specific features, common to all the Java applications generated with ASSL, at runtime, a Voyager prototype produces log records, which show important state-transition operations ongoing in the system [6]. Here, we used these records to trace and evaluate the behavior of the generated prototype model.

In order to perform this exercise, we compiled the generated Java code with Java 1.6.0 first, and then we ran the compiled code. The latter ran smoothly with no errors. First, it started all system threads as it is partially shown in the following log records. Note that starting all system threads first is a standard running procedure applied to all prototype models generated with the ASSL framework.

Log Records "Starting System Threads"

1) EVENT 'as.aes.antenna_california.events.IMAGESESSIONENDEDBLUE': started
2) EVENT 'as.aes.antenna_california.events.IMAGESESSIONSTARTEDGREEN': started
3) EVENT 'as.aes.antenna_california.events.REDIMAGESESSIONISABOUTTOSTART': started
4) EVENT 'as.aes.antenna_california.events.IMAGESESSIONENDEDGREEN': started
5) EVENT 'as.aes.antenna_california.events.IMAGESESSIONSTARTEDRED': started
6) EVENT 'as.aes.antenna_california.events.IMAGESESSIONENDEDRED': started
7) EVENT 'as.aes.antenna_california.events.IMAGEANTCALIFORNIASENT': started
8) EVENT 'as.aes.antenna_california.events.GREENIMAGESESSIONISABOUTTOSTART': started
9) EVENT 'as.aes.antenna_california.events.BLUEIMAGESESSIONISABOUTTOSTART': started
10) EVENT 'as.aes.antenna_california.events.IMAGESESSIONSTARTEDBLUE': started
11) FLUENT 'as.aes.antenna_california.aeself_management.image_processing. INSENDINGIMAGE': started
12) FLUENT 'as.aes.antenna_california.aeself_management.image_processing. INCOLLECTINGIMAGEPIXELSBLUE': started
13) FLUENT 'as.aes.antenna_california.aeself_management.image_processing. INCOLLECTINGIMAGEPIXELSGREEN': started
14) FLUENT 'as.aes.antenna_california.aeself_management.image_processing. INSTARTINGBLUEIMAGESESSION': started
15) FLUENT 'as.aes.antenna_california.aeself_management.image_processing. INCOLLECTINGIMAGEPIXELSRED': started
16) FLUENT 'as.aes.antenna_california.aeself_management.image_processing. INSTARTINGGREENIMAGESESSION': started
17) FLUENT 'as.aes.antenna_california.aeself_management.image_processing. INSTARTINGREDIMAGESESSION': started
18) POLICY 'as.aes.antenna_california.aeself_management.IMAGE_PROCESSING': started
19) AE 'as.aes.ANTENNA_CALIFORNIA': started

Here records 1 through to 19 show the start-up process of the ANTENNA_CALIFORNIA autonomic element. Similar log records notified us that all the threads in all generated AEs started successfully. After starting up all the threads, the system ran in idle mode for 60 seconds, when the TIMETOTAKEPICTURE timed event occurred (see record 99). This event is specified in the Voyager AE to run on regular basis every 60 seconds (see Section 4.1.3) and it triggers a series of system transitions following the specified autonomic behavior. The following log records demonstrate that the runtime image-processing behavior followed correctly the ASSL specification of the **IMAGE_PROCESSING** policy.

Log Records "Voyager Autonomic Behavior"

99) EVENT 'as.aes.voyager.events.TIMETOTAKEPICTURE': has occurred
100) FLUENT 'as.aes.voyager.aeself_management.image_processing.INTAKINGPICTURE': has been initiated
101) ACTION 'as.aes.voyager.actions.TAKEPICTURE': has been performed
102) EVENT 'as.aes.voyager.events.PICTURETAKEN': has occurred
103) FLUENT 'as.aes.voyager.aeself_management.image_processing.INTAKINGPICTURE': has been terminated
104) FLUENT 'as.aes.voyager.aeself_management.image_processing. INPROCESSINGPICTUREPIXELS': has been initiated
105) ACTION 'as.aes.voyager.actions.PROCESSFILTEREDPICTURE': has been performed
106) ACTION 'as.aes.voyager.actions.PROCESSFILTEREDPICTURE': has been performed
107) ACTION 'as.aes.voyager.actions.PROCESSFILTEREDPICTURE': has been performed
108) ACTION 'as.aes.voyager.actions.PROCESSPICTURE': has been performed
109) EVENT 'as.aes.voyager.events.PICTUREPROCESSED': has occurred
110) EVENT 'as.aes.antenna_japan.events.BLUEIMAGESESSIONISABOUTTOSTART': has occurred
111) EVENT 'as.aes.antenna_spain.events.REDIMAGESESSIONISABOUTTOSTART': has occurred
112) FLUENT 'as.aes.antenna_spain.aeself_management.image_processing. INSTARTINGREDIMAGESESSION': has been initiated
113) FLUENT 'as.aes.antenna_japan.aeself_management.image_processing. INSTARTINGBLUEIMAGESESSION': has been initiated
114) EVENT 'as.aes.antenna_spain.events.BLUEIMAGESESSIONISABOUTTOSTART': has occurred
115) FLUENT 'as.aes.voyager.aeself_management.image_processing. INPROCESSINGPICTUREPIXELS': has been terminated

Here, records 99 through to 103 show the initiation and termination of the voyager's INTAKINGPICTURE fluent. This resulted in the execution of the TAKEPICTURE action (see record 101), which triggered the PICTURETAKEN event (see record 102). The latter consecutively initiated the INPROCESSINGPICTUREPIXELS fluent. Records 104 through to 109 and record 115 show the initiation and termination of that fluent. The INPROCESSINGPICTUREPIXELS fluent prompted the execution of the PROCESSPICTURE action (see record 108), which executed the PROCESSFILTEREDPICTURE action three times (see records 105 through to 107). Each time, this action was called to apply a different filter color (*blue*, *red*, or *green*) and sent the filtered image to the antennas on Earth. Note that this action also uses the Voyager AE's AEIP-specified functions (see Appendix A) **sendBeginSessionMsgs** and **sendEndSessionMsgs** to send *begin-session* and *end-session* messages for each applied filter to the antennas on Earth.

Subsequently, these messages prompted three [color]**ImageSessionIsAboutToStart** events for each antenna, one per a filter color (see record 110 for the **BLUE**IMAGESESSIONISABOUTTOSTART event). Next these events initiated in the antenna AEs three **inStarting**[color]**ImageSession** fluents, one per filter color (see record 113 for the INSTARTING**BLUE**IMAGESESSION fluent). Each of these fluents prompted the execution of the STARTIMAGECOLLECTSESSION action (see records 116). Note that this action was executed twelve times (one time for each applied filter per antenna) and it prompted the operation of receiving the *begin-session* messages. Subsequently, the antennas received these messages and corresponding events were prompted to terminate **inStarting**[color]**ImageSession** fluents and initiate fluents to collect the image pixels.

For each antenna AE, the pixel-collection fluent prompted the execution of a special pixel-collection action (see Appendix A). Thus, that action was executed for each antenna three times, one per a filter color. Internally, this action received image messages specified at the ASIP tier (see Section 4.1.2) including special *end-session* messages that terminated the image-transmission sessions (per filter color and per antenna).

Next, every received end-session message terminated the current active fluent for the current antenna AE. In addition, the last end-session message, for every antenna, initiated another fluent (termed **inSendingImage** – see Appendix A) that prompted the execution of a special action (termed **sendImage**; see Appendix A). The latter prepared the collected image and sent it to the Voyager Mission base on Earth. Further, this operation prompted a particular event at each antenna that terminated the **inSendingImage** fluent. Further, the system continued repeating the same steps on a regular basis due to the TIMETOTAKEPICTURE timed event (see record 99), which occurs every 60 seconds (see the **timeToTakePicture** ASSL specification in Section 4.1.3). It is important to mention that the run-time behavior of the generated prototype model for the Voyager II mission strictly followed that specified with the ASSL **IMAGE_PROCESSING** self-management policy.

5. DISCUSSION AND CONCLUSION

In the most basic of terms, experiments are said to be valid if they do what they are supposed to do. In that context, the experiments and evaluation results described here are valid and they demonstrated that the Voyager's prototype developed with ASSL is able to perform image processing as the original mission did. Although programmed as an autonomic policy, the image-processing behavior implanted in our prototype does not extend the original event-driven behavior observed in the Voyager Mission, but rather copies the same. Here, under simulated conditions (the prototype is triggered to take pictures every 60 sec), the prototype successfully transmitted blended images to (virtual) antennas on Earth, where these images were redirected to the mission base for further processing.

It is important to mention though, that in its initial version, the Voyager's prototype abstracts the components of the spacecraft without evaluating their behavior. Hence, the next prototype model will specify the Voyager spacecraft's radio, antenna, and two cameras as distinct managed elements. This will allow the evaluation of their behavior (via metrics and events) and extending the **IMAGE_PROCESSING** policy with other self-management features. For example, fluents that react on malfunction in some of these components can trigger self-healing policies. In such a case, we are planning to implement two scenarios: *remote-assistance self-healing* and *on-board self-healing*. The former will copy the behavior of the original spacecraft, where remote assistance is provided in the form of radio contact and remote control programming. However, the on-board self-healing will add new autonomic features, which do not exist in the original spacecraft. Having the self-healing operations automated will allow us to evaluate to some extent the potential impact of AC on the maintenance required by the Voyager Mission.

5.1. Benefits for Space Systems

As we have stated, both the ASSL specifications of the Voyager Mission and the prototypes of the same can be extremely useful for the design and implementation of future Voyager-like missions. The ability to compare features and issues with the actual mission and with hypothesized possible autonomic approaches gives significant benefit.

In our approach, we develop Voyager prototypes in a series of incremental and iterative steps where each prototype includes new autonomic features. This helps to evaluate the performance of each feature and gradually construct a model of a future Voyager-like system. Here, different prototypes can be tried and tested (and benchmarked as well), and get valuable feedback before we implement the real system.

Moreover, this approach helps to discover eventual design flaws in both the original system and the prototype models. Currently, the features are validated through experiments. However, the new ASSL model checking mechanism currently under development [9] will allow for automatic feature validation and discovery of design flaws. Hence, the Voyager prototypes assist in refining the potential risks in the development and exploitation of future missions, helping in the considerable reduction in development and maintenance costs.

5.2. Autonomy Requirements for Space Missions

An important part of the work on modeling autonomic features for space missions is to identify the autonomy requirements for such features. Although in this first step of building prototypes for autonomic Voyager-like missions we did not really use requirements engineering to capture the needed autonomic features, for autonomic features, which are not that explicitly stated, we will need to go through all the steps of a thorough requirements process. Part of this research, is our work on an approach to Autonomy Requirements Engineering (ARE) [10] intended to address requirements related to adaptation issues, in particular: 1) what adaptations are possible; 2) under what constrains; and 3) how those adaptations are realized. Note that adaptations

arise when a system needs to cope with changes to ensure realization of the system's objectives. The self-adaptation abilities of a system are considered as autonomic features.

The ARE approach combines *generic autonomy requirements* (GAR) [11] with *goal-oriented requirements engineering* (GORE) [12]. Using this approach, software engineers can determine what autonomic features to develop for a particular space mission (e.g., Voyager) as well as what artifacts that process might generate (e.g., goals models, requirements specification, etc.). The inputs required by this approach are the *mission goals* and *domain-specific GAR* reflecting specifics of the mission class (e.g., interplanetary missions).

The first step in developing any new software-intensive system is to determine the system's functional and non-functional requirements. The former requirements define what the system will actually do, while the latter requirements refer to its qualities, such as performance, along with any constraints under which the system must operate. Despite differences in application domain and functionality, all autonomic systems extend upstream the regular software-intensive systems with special *self-management objectives* (self-* objectives). Basically, the self-* objectives provide the system's ability to automatically discover, diagnose, and cope with various problems. This ability depends on the system's degree of autonomicity, quality and quantity of knowledge, awareness and monitoring capabilities, and quality characteristics such as adaptability, dynamicity, robustness, resilience, and mobility [11]. Basically, this is the basis of the ARE approach [10, 11, 12]: autonomy requirements are detected as self-objectives backed up by different capabilities and quality characteristics outlined by the GAR model. Note that the approach targets exclusively the *self-* objectives* as the only means to explicitly determine and define autonomy requirements. Thus, it is not meant to handle the regular functional and non-functional requirements of the systems, presuming that those might by tackled by the traditional requirements engineering approaches, e.g., use case modeling, domain modeling, constraints modeling (OCL), etc. Functional and non-functional requirements might be captured by the ARE approach only as part of the self-* objectives elicitation, i.e., some of the GAR's requirements might be considered as functional and non-functional requirements.

The ARE approach starts with the creation of a *goals model* that represents system objectives and their interrelationships for the mission in question. For this, we use GORE where ARE goals are generally modeled with intrinsic features such as *type*, *actor*, and *target*, with links to other goals and constraints in the requirements model. Goals models might be organized in different ways copying with the mission specifics and engineers' understanding about the mission goals. Thus we may have 1) hierarchical structures where goals reside different level of granularity; 2) concurrent structures where goals are considered as concurrent; etc. The goals models are not formal and we use natural language along with UML-like diagrams to record them.

The next step in the ARE approach is to work on each one of the system goals along with the elicited environmental constraints to come up with the self-* objectives representing autonomic features and providing the autonomy requirements for this particular system's behavior. In this phase, the GAR model is applied to a single mission goal to derive autonomy requirements in the form of goal's supportive and alternative self-* objectives along with the necessary capabilities and quality characteristics. In the first part of this phase, we record the GAR model in natural language. In the second part though, we may use a formal notation (e.g., ASSL) to express this model in a more precise way. Note that, this model carries more details about the autonomy requirements, and can be further used for different analysis activities, including requirements validation and verification. However, the formal model is not mandatory in this approach and we can simply write the requirements details in natural language instead. Of course, a formal model has significant advantages over a natural language, which lay mainly in the ambiguity of the natural language and the mathematical precision provided by the formal notation's semantics.

Although ASSL is not designed to tackle autonomy requirements, it could be extremely powerful when handling the so-called event-based autonomy [13]. ASSL aims at event-driven autonomic behavior. Recall that to specify self-management policies, we need to specify appropriate events (see Section 4.1.1). Here, we rely on the reach set of event types exposed by ASSL. For example, to specify ASSL events, one may use logical expressions over SLOs (service-level objectives), or may relate events with metrics, other events, actions, time, and messages (see Section 3.1). Moreover, ASSL allows for the specification of special conditions that must be stated before an event is prompted. Therefore, events can be constrained by adding such conditions to their specification. Because ASSL is event-driven (exposes a rich set of possible events raised in the environment and the system itself), special competence models can be built by using the self-management policies and relying on ASSL fluents and actions to provide for desired behaviors. Note that ASSL implies layering for structuring functionalities in event-driven autonomy and provides computational structures that can be possibly effective when handling autonomy requirements. Moreover, the platform can effectively handle goals models through the specification of the service-level objectives, i.e., each mission goal might be specified as a distinct SLO and globally-defined SLOs (at the level of the AS Tier) might capture the relationships between those SLOs.

Although very efficient and powerful, ASSL does not provide real reasoning capabilities, which might be required for more complex self-management problems. Actually, *reasoning* along with *knowledge representation* is required for the highest level of autonomy – the so-called *goal-oriented autonomy* [13]. At Lero, we are currently developing a formal

method called KnowLang [14] incorporating the ASSL features along with knowledge representation and reasoning. With KnowLang, the ARE-captured self-* objectives are specified with special policies (similar to the ASSL policies) associated with goals, special situations, actions (eventually identified as system capabilities), metrics, etc. Thus, the self-* objectives are represented as policies describing at an abstract level what the spacecraft will do when particular situations arise. The situations are meant to represent the conditions needed to be met in order for the system to switch to a self-* objective while pursuing a system goal. KnowLang has been successfully used to specify autonomy requirements for the ESA's BepiColombo Mission [12] – a mission similar to Voyager targeting scientific study of Mercury.

ARE could be used at several stages in the work flow from initiating a mission concept through to building and launching a spacecraft:

- High-level mission goals can be used in conjunction with a fairly general GAR model to generate a high level model incorporating the autonomy requirements (self-* objectives). This model could be combined with a reasoning engine to establish whether or not all the requirements are mutually compatible. It could also be used to communicate the requirements as long as the engineers can see what alternative behavior is required when the mission is following a particular goal and under what circumstances.
- The model could be used to assist in the compilation of the Autonomy Requirements (AR) section of the System Requirements Specification document. The goals model along with the autonomy requirements elicited per goal will form such a section. This eventually, will help to easily derive some of the functional and non-functional requirements – related to the monitoring activities, knowledge, and AR (autonomy requirements) quality attributes..
- The process of writing the ARs could also be used to add further details to the ARE model.
- With the necessary tool support it should be possible to formally validate and verify the formally specified ARs (with ASSL or KnowLang).
- Eventually, if both the ARs written in a natural language and the formal model are made available together to the software design engineers, it should help to ensure more accurate implementation of the software with fewer bugs.

5.3. Future Work

Future work is concerned with further prototype development by including new autonomic features. Together with a detailed specification of the Voyager spacecraft components, we intend to build prototypes incorporating self-healing, self-protecting, and self-adapting policies. These will help to construct an intelligent Voyager-like system able to react automatically to hazards in space by finding possible solutions and applying those on-board with no human interaction. ARE shall be applied to capture the right autonomy requirements for those autonomic features. We will continue experimenting with both platforms – ASSL and KnowLang, eventually targeting event-driven and goal-oriented autonomy for Voyager-like missions.

ACKNOWLEDGMENT

This work was supported in part by the Science Foundation Ireland grant 03/CE2/I303_1 to Lero—the Irish Software Engineering Research Centre at University of Limerick, Ireland, the ESTEC ESA (contract No. 4000106016), and by the European Union FP7 Integrated Project on Autonomic Service-Component Ensembles (ASCENS).

REFERENCES

[1] R. Murch, Autonomic Computing: On Demand Series, IBM Press, Prentice Hall, 2004.

[2] IBM Corporation, "An architectural blueprint for autonomic computing", White paper, 4th ed., IBM Press, 2006.

[3] W. Truszkowski, M. Hinchey, J. Rash, and C. Rouff, "NASA's swarm missions: The challenge of building autonomous software", IT Professional, vol. 6(5), 2004, pp. 47-52.

[4] M. Hinchey, J. Rash, W. Truszkowski, C. Rouff, and R. Sterritt, "Autonomous and autonomic swarms", Proc 8th Biennial Conference on Real Time in Sweden (RTiS 2005), Sweden, 2005, pp. 65-73.

[5] E. Vassev, Towards a Framework for Specification and Code Generation of Autonomic Systems - Ph.D. Thesis, Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, November, 2008.

[6] The Planetary Society, "Space topics: Voyager – the story of the mission", http://planetary.org/explore/topics/space_missions/voyager/ objectives.html, 2009.

[7] Jet Propulsion Laboratory, Voyager – The Interstellar Mission, Califoprnia Institute of Technology, http://voyager.jpl.nasa.gov/ mission/interstellar.html, 2009.

[8] W. M. Browne, "Technical 'magic' converts a puny signal into pictures", NY Times, August 26, 1989.

[9] E. Vassev, M. Hinchey, and A. Quigley, "Model Checking for autonomic systems specified with ASSL", Proc. of the First NASA Formal Methods Symposium (NFM 2009), NASA, 2009, pp.16-25.

[10] E. Vassev and M. Hinchey, "Autonomy Requirements Engineering", IEEE Computer, vol. 46 (8), 2013, pp. 82-84.

[11] E. Vassev and M. Hinchey, “Autonomy Requirements Engineering: a case study on the BepiColombo Mission”, Proc. of C* Conference on Computer Science & Software Engineering (C3S2E '13). ACM, 2013, pp. 31-41.

[12] E. Vassev and M. Hinchey, “On the autonomy requirements for space missions”, Proc. of the 16th IEEE International Symposium on Object/Component/Service-oriented Real-time Distributed Computing Workshops (ISCORCW 2013), IEEE Computer Society, Paderborn, Germany, June 19-21, 2013, to appear.

[13] ECSS Secretariat, “Space Engineering: Space Segment Operability”, ESA-ESTEC, Requirements & Standards Division Noordwijk, The Netherlands. ECSS-E-ST-70-11C (2008).

[14] E. Vassev and M. Hinchey, “Knowledge representation for cognitive robotic systems”, Proc. of the 15th IEEE International Symposium on Object/Component/Service-oriented Real-time Distributed Computing Workshops (ISCORCW 2012), 2012, pp. 156–163.

APPENDIX A: ASSL SPECIFICATION MODEL FOR THE NASA VOYAGER MISSION

```
//==================== autonomic system VOYAGER MISSION ====================
//================ IMAGE-PROCESSING self-management policy =================
//
// This is the full specification of the Voyager2 mission.
// All the four antennas are operational.
//
// Note:
// - the voyager's cameras are set to apply three filters blue, red, and green;
// - the wide-angle camera takes pictures 100 x 50 pixels;
// - the narrow-angle camera takes pictures 50 x 50 pixels.
// - to trigger the self-management mechanism we specified a PERIOD { 60 sec )
//   activation in the Voyager's timeToTakePicture event.
//
//==========================================================================
AS VOYAGER_MISSION {
   TYPES {   Pixel }
      VARS { integer numPixelsPerImage    } // determines the image size in pixels

      ASSELF_MANAGEMENT {
            OTHER_POLICIES {
                  POLICY IMAGE_PROCESSING {
                        FLUENT inProcessingImage_AntAustralia {
                              INITIATED_BY { EVENTS.imageAntAustraliaReceived } TERMINATED_BY { EVENTS.imageAntAustraliaProcessed } }
                        FLUENT inProcessingImage_AntJapan {
                              INITIATED_BY { EVENTS.imageAntJapanReceived }        TERMINATED_BY { EVENTS.imageAntJapanProcessed } }
                        FLUENT inProcessingImage_AntCalifornia {
                              INITIATED_BY { EVENTS.imageAntCaliforniaReceived }  TERMINATED_BY { EVENTS.imageAntCaliforniaProcessed } }
                        FLUENT inProcessingImage_AntSpain {
                              INITIATED_BY { EVENTS.imageAntSpainReceived }        TERMINATED_BY { EVENTS.imageAntSpainProcessed } }
                        MAPPING {
                              CONDITIONS { inProcessingImage_AntAustralia} DO_ACTIONS { ACTIONS.processImage("Antenna_Australia") } }
                        MAPPING {
                              CONDITIONS { inProcessingImage_AntJapan} DO_ACTIONS { ACTIONS.processImage("Antenna_Japan") } }
                        MAPPING {
                              CONDITIONS { inProcessingImage_AntCalifornia} DO_ACTIONS { ACTIONS.processImage("Antenna_California") } }
                        MAPPING {
                              CONDITIONS { inProcessingImage_AntSpain} DO_ACTIONS { ACTIONS.processImage("Antenna_Spain") } }
                  }
            }
      } // ASSELF_MANAGEMENT

      ASARCHITECTURE {
            AELIST {AES.Voyager, AES.Antenna_Australia, AES.Antenna_Japan, AES.Antenna_California, AES.Antenna_Spain}
            DIRECT_DEPENDENCIES {
                  DEPENDENCY AES.Antenna_Australia { AES.Voyager }
                  DEPENDENCY AES.Antenna_Japan { AES.Voyager }
                  DEPENDENCY AES.Antenna_California { AES.Voyager }
                  DEPENDENCY AES.Antenna_Spain { AES.Voyager }
            }
            GROUPS {
                  GROUP VoyagerGroup {
                        MEMBERS { AES.Voyager, AES.Antenna_Australia, AES.Antenna_Japan, AES.Antenna_California, AES.Antenna_Spain } }
            }
      } // ASARCHITECTURE

      ACTIONS {
            ACTION IMPL processImage { // process an image sent by a specific antenna
                  PARAMETERS { string antennaName  }
                  GUARDS { ASSELF_MANAGEMENT.OTHER_POLICIES.IMAGE_PROCESSING.inProcessingImage_AntAustralia OR
                        ASSELF_MANAGEMENT.OTHER_POLICIES.IMAGE_PROCESSING.inProcessingImage_AntJapan OR
                        ASSELF_MANAGEMENT.OTHER_POLICIES.IMAGE_PROCESSING.inProcessingImage_AntCalifornia OR
                         ASSELF_MANAGEMENT.OTHER_POLICIES.IMAGE_PROCESSING.inProcessingImage_AntSpain }
                  TRIGGERS {
                        IF antennaName = "Antenna_Australia" THEN EVENTS.imageAntAustraliaProcessed       END
                        ELSE
                              IF antennaName = "Antenna_Japan" THEN EVENTS.imageAntJapanProcessed END
                              ELSE
                                    IF antennaName = "Antenna_California" THEN EVENTS.imageAntCaliforniaProcessed END
                                    ELSE
                                          IF antennaName = "Antenna_Spain" THEN EVENTS.imageAntSpainProcessed END
                                    END
                              END
                        END
                  }
```

```
            }
        } // ACTIONS
        EVENTS { // these events are used in the fluents specification
            EVENT imageAntAustraliaReceived { ACTIVATION  { RECEIVED { ASIP.MESSAGES.msgImageAntAustralia } } }
            EVENT imageAntJapanReceived { ACTIVATION  { RECEIVED { ASIP.MESSAGES.msgImageAntJapan } } }
            EVENT imageAntCaliforniaReceived { ACTIVATION  { RECEIVED { ASIP.MESSAGES.msgImageAntCalifornia } } }
            EVENT imageAntSpainReceived { ACTIVATION  { RECEIVED { ASIP.MESSAGES.msgImageAntSpain } } }

            EVENT imageAntAustraliaProcessed { }
            EVENT imageAntJapanProcessed { }
            EVENT imageAntCaliforniaProcessed { }
            EVENT imageAntSpainProcessed { }
        } // EVENTS

} // AS VOYAGER_MISSION

//==================== AS interaction protocol ============
ASIP {
        MESSAGES {
            MESSAGE msgImageAntAustralia { SENDER { AES.Antenna_Australia }      RECEIVER { ANY } PRIORITY { 1 } MSG_TYPE { BIN } }
            MESSAGE msgImageAntJapan { SENDER { AES.Antenna_Japan } RECEIVER { ANY } PRIORITY { 1 } MSG_TYPE {BIN } }
            MESSAGE msgImageAntCalifornia { SENDER { AES.Antenna_California } RECEIVER { ANY } PRIORITY { 1 } MSG_TYPE { BIN } }
            MESSAGE msgImageAntSpain { SENDER { AES.Antenna_Spain }  RECEIVER { ANY } PRIORITY { 1 } MSG_TYPE {BIN } }
        } // MESSAGES

        CHANNELS {
            CHANNEL IMG_Link {
                ACCEPTS {  ASIP.MESSAGES.msgImageAntAustralia, ASIP.MESSAGES.msgImageAntJapan,
                           ASIP.MESSAGES.msgImageAntCalifornia, ASIP.MESSAGES.msgImageAntSpain }
                ACCESS { SEQUENTIAL }
                DIRECTION { INOUT } }
        } // CHANNELS

        FUNCTIONS {
            FUNCTION sendImageMsg {
                PARAMETERS { string antennaName }
                DOES {
                    IF antennaName = "Antenna_Australia" THEN ASIP.MESSAGES.msgImageAntAustralia >> ASIP.CHANNELS.IMG_Link END
                    ELSE
                        IF antennaName = "Antenna_Japan" THEN ASIP.MESSAGES.msgImageAntJapan >> ASIP.CHANNELS.IMG_Link END
                        ELSE
                            IF antennaName = "Antenna_California" THEN ASIP.MESSAGES.msgImageAntCalifornia >> ASIP.CHANNELS.IMG_Link END
                            ELSE
                                IF antennaName = "Antenna_Spain" THEN ASIP.MESSAGES.msgImageAntSpain >> ASIP.CHANNELS.IMG_Link END
                            END
                        END
                    END
                }
            }
            FUNCTION receiveImageMsg {
                PARAMETERS { string antennaName }
                DOES {
                    IF antennaName = "Antenna_Australia" THEN ASIP.MESSAGES.msgImageAntAustralia << ASIP.CHANNELS.IMG_Link END
                    ELSE
                        IF antennaName = "Antenna_Japan" THEN ASIP.MESSAGES.msgImageAntJapan << ASIP.CHANNELS.IMG_Link END
                      ELSE
                        IF antennaName = "Antenna_California" THEN ASIP.MESSAGES.msgImageAntCalifornia << ASIP.CHANNELS.IMG_Link END
                            ELSE
                                IF antennaName = "Antenna_Spain" THEN ASIP.MESSAGES.msgImageAntSpain << ASIP.CHANNELS.IMG_Link END
                            END
                        END
                    END
                }
            }
        } // FUNCTIONS
}

//==================== autonomic elements ====================
AES {

        //==================== AE Voyager ====================
        AE Voyager {
            VARS { boolean isWideAngleImage } //determines the type of picture (wide-angle or narrow-angle)

            AESELF_MANAGEMENT {
                OTHER_POLICIES {
                    POLICY IMAGE_PROCESSING {
                        FLUENT inTakingPicture {
                            INITIATED_BY { EVENTS.timeToTakePicture } TERMINATED_BY { EVENTS.pictureTaken } }
                        FLUENT inProcessingPicturePixels {
                            INITIATED_BY { EVENTS.pictureTaken } TERMINATED_BY { EVENTS.pictureProcessed } }
                        MAPPING {
                            CONDITIONS { inTakingPicture }  DO_ACTIONS { ACTIONS.takePicture } }
                        MAPPING {
                            CONDITIONS { inProcessingPicturePixels }  DO_ACTIONS { ACTIONS.processPicture } }
                    }
                }
            } // AESELF_MANAGEMENT

            //====== AEs that can use the messages and channels specified by this AE ======
            FRIENDS {
                AELIST { AES.Antenna_Australia, AES.Antenna_Japan, AES.Antenna_California, AES.Antenna_Spain }
            }
```

```
//==================== AE interaction protocol ====================
AEIP {
    MESSAGES {
        MESSAGE msgImagePixel {
            SENDER { AES.Voyager }
            RECEIVER { AES.Antenna_Australia, AES.Antenna_Japan, AES.Antenna_California, AES.Antenna_Spain }
            MSG_TYPE { BIN }
        }

        // session messages to be received by Antenna_Australia
        MESSAGE msgBlueSessionBeginAus {
            SENDER { AES.Voyager }
            RECEIVER { AES.Antenna_Australia }
            MSG_TYPE { NEGOTIATION }
            BODY { BEGIN }
        }
        MESSAGE msgBlueSessionEndAus {
            SENDER { AES.Voyager }
            RECEIVER { AES.Antenna_Australia }
            MSG_TYPE {  NEGOTIATION }
            BODY { END }
        }
        MESSAGE msgRedSessionBeginAus {
            SENDER { AES.Voyager }
            RECEIVER { AES.Antenna_Australia }
            MSG_TYPE { NEGOTIATION }
            BODY { BEGIN }
        }
        MESSAGE msgRedSessionEndAus {
            SENDER { AES.Voyager }
            RECEIVER { AES.Antenna_Australia }
            MSG_TYPE { NEGOTIATION }
            BODY { END }
        }
        MESSAGE msgGreenSessionBeginAus {
            SENDER { AES.Voyager }
            RECEIVER { AES.Antenna_Australia }
            MSG_TYPE { NEGOTIATION }
            BODY { BEGIN }
        }
        MESSAGE msgGreenSessionEndAus {
            SENDER { AES.Voyager }
            RECEIVER { AES.Antenna_Australia }
            MSG_TYPE { NEGOTIATION }
            BODY { END }
        }

        // session messages to be received by Antenna_Japan
        MESSAGE msgBlueSessionBeginJpn {
            SENDER { AES.Voyager }
            RECEIVER { AES.Antenna_Japan }
            MSG_TYPE { NEGOTIATION }
            BODY { BEGIN }
        }
        MESSAGE msgBlueSessionEndJpn {
            SENDER { AES.Voyager }
            RECEIVER { AES.Antenna_Japan }
            MSG_TYPE { NEGOTIATION }
            BODY { END }
        }
        MESSAGE msgRedSessionBeginJpn {
            SENDER { AES.Voyager }
            RECEIVER { AES.Antenna_Japan }
            MSG_TYPE { NEGOTIATION }
            BODY { BEGIN }
        }
        MESSAGE msgRedSessionEndJpn {
            SENDER { AES.Voyager }
            RECEIVER { AES.Antenna_Japan }
            MSG_TYPE { NEGOTIATION }
            BODY { END }
        }
        MESSAGE msgGreenSessionBeginJpn {
            SENDER { AES.Voyager }
            RECEIVER { AES.Antenna_Japan }
            MSG_TYPE { NEGOTIATION }
            BODY { BEGIN }
        }
        MESSAGE msgGreenSessionEndJpn {
            SENDER { AES.Voyager }
            RECEIVER { AES.Antenna_Japan }
            MSG_TYPE { NEGOTIATION }
            BODY { END }
        }

        // session messages to be received by Antenna_California
        MESSAGE msgBlueSessionBeginCfn {
            SENDER { AES.Voyager }
            RECEIVER { AES.Antenna_California }
            MSG_TYPE { NEGOTIATION }
            BODY { BEGIN }
        }
```

```
        MESSAGE msgBlueSessionEndCfn {
                SENDER { AES.Voyager }
                RECEIVER { AES.Antenna_California }
                MSG_TYPE { NEGOTIATION }
                BODY { END }
        }
        MESSAGE msgRedSessionBeginCfn {
                SENDER { AES.Voyager }
                RECEIVER { AES.Antenna_California }
                MSG_TYPE { NEGOTIATION }
                BODY { BEGIN }
        }
        MESSAGE msgRedSessionEndCfn {
                SENDER { AES.Voyager }
                RECEIVER { AES.Antenna_California }
                MSG_TYPE { NEGOTIATION }
                BODY { END }
        }
        MESSAGE msgGreenSessionBeginCfn {
                SENDER { AES.Voyager }
                RECEIVER { AES.Antenna_California }
                MSG_TYPE { NEGOTIATION }
                BODY { BEGIN }
        }
        MESSAGE msgGreenSessionEndCfn {
                SENDER { AES.Voyager }
                RECEIVER { AES.Antenna_California }
                MSG_TYPE { NEGOTIATION }
                BODY { END }
        }

        // session messages to be received by Antenna_Spain
        MESSAGE msgBlueSessionBeginSpn {
                SENDER { AES.Voyager }
                RECEIVER { AES.Antenna_Spain }
                MSG_TYPE { NEGOTIATION }
                BODY { BEGIN }
        }
        MESSAGE msgBlueSessionEndSpn {
                SENDER { AES.Voyager }
                RECEIVER { AES.Antenna_Spain }
                MSG_TYPE { NEGOTIATION }
                BODY { END }
        }
        MESSAGE msgRedSessionBeginSpn {
                SENDER { AES.Voyager }
                RECEIVER { AES.Antenna_Spain }
                MSG_TYPE { NEGOTIATION }
                BODY { BEGIN }
        }
        MESSAGE msgRedSessionEndSpn {
                SENDER { AES.Voyager }
                RECEIVER { AES.Antenna_Spain }
                MSG_TYPE { NEGOTIATION }
                BODY { END }
        }
        MESSAGE msgGreenSessionBeginSpn {
                SENDER { AES.Voyager }
                RECEIVER { AES.Antenna_Spain }
                MSG_TYPE { NEGOTIATION }
                BODY { BEGIN }
        }
        MESSAGE msgGreenSessionEndSpn {
                SENDER { AES.Voyager }
                RECEIVER { AES.Antenna_Spain }
                MSG_TYPE { NEGOTIATION }
                BODY { END }
        }
} // MESSAGES

CHANNELS {
        CHANNEL VOYAGER_Link {
                ACCEPTS { AEIP.MESSAGES.msgImagePixel,
                        AEIP.MESSAGES.msgBlueSessionBeginAus, AEIP.MESSAGES.msgBlueSessionEndAus,
                        AEIP.MESSAGES.msgRedSessionBeginAus, AEIP.MESSAGES.msgRedSessionEndAus,
                        AEIP.MESSAGES.msgGreenSessionBeginAus, AEIP.MESSAGES.msgGreenSessionEndAus,

                        AEIP.MESSAGES.msgBlueSessionBeginJpn, AEIP.MESSAGES.msgBlueSessionEndJpn,
                        AEIP.MESSAGES.msgRedSessionBeginJpn, AEIP.MESSAGES.msgRedSessionEndJpn,
                        AEIP.MESSAGES.msgGreenSessionBeginJpn, AEIP.MESSAGES.msgGreenSessionEndJpn,

                        AEIP.MESSAGES.msgBlueSessionBeginCfn, AEIP.MESSAGES.msgBlueSessionEndCfn,
                        AEIP.MESSAGES.msgRedSessionBeginCfn, AEIP.MESSAGES.msgRedSessionEndCfn,
                        AEIP.MESSAGES.msgGreenSessionBeginCfn, AEIP.MESSAGES.msgGreenSessionEndCfn,

                        AEIP.MESSAGES.msgBlueSessionBeginSpn, AEIP.MESSAGES.msgBlueSessionEndSpn,
                        AEIP.MESSAGES.msgRedSessionBeginSpn, AEIP.MESSAGES.msgRedSessionEndSpn,
                        AEIP.MESSAGES.msgGreenSessionBeginSpn, AEIP.MESSAGES.msgGreenSessionEndSpn
                }
                ACCESS { DIRECT }
                DIRECTION { INOUT }
        }
}
```

```
        FUNCTIONS {
                FUNCTION sendImagePixelMsg {
                        DOES { AEIP.MESSAGES.msgImagePixel >> AEIP.CHANNELS.VOYAGER_Link     }
                }
                FUNCTION sendBeginSessionMsgs {
                        PARAMETERS { string filterName }
                        DOES {
                                IF filterName = "blue" THEN
                                        AEIP.MESSAGES.msgBlueSessionBeginAus >> AEIP.CHANNELS.VOYAGER_Link;
                                        AEIP.MESSAGES.msgBlueSessionBeginJpn >> AEIP.CHANNELS.VOYAGER_Link;
                                        AEIP.MESSAGES.msgBlueSessionBeginCfn >> AEIP.CHANNELS.VOYAGER_Link;
                                        AEIP.MESSAGES.msgBlueSessionBeginSpn >> AEIP.CHANNELS.VOYAGER_Link
                                END
                                ELSE
                                        IF filterName = "red" THEN
                                                AEIP.MESSAGES.msgRedSessionBeginAus >> AEIP.CHANNELS.VOYAGER_Link;
                                                AEIP.MESSAGES.msgRedSessionBeginJpn >> AEIP.CHANNELS.VOYAGER_Link;
                                                AEIP.MESSAGES.msgRedSessionBeginCfn >> AEIP.CHANNELS.VOYAGER_Link;
                                                AEIP.MESSAGES.msgRedSessionBeginSpn >> AEIP.CHANNELS.VOYAGER_Link
                                        END
                                        ELSE
                                                IF filterName = "green" THEN
                                                        AEIP.MESSAGES.msgGreenSessionBeginAus >> AEIP.CHANNELS.VOYAGER_Link;
                                                        AEIP.MESSAGES.msgGreenSessionBeginJpn >> AEIP.CHANNELS.VOYAGER_Link;
                                                        AEIP.MESSAGES.msgGreenSessionBeginCfn >> AEIP.CHANNELS.VOYAGER_Link;
                                                        AEIP.MESSAGES.msgGreenSessionBeginSpn >> AEIP.CHANNELS.VOYAGER_Link
                                                END
                                        END
                                END
                        }
                }
                FUNCTION sendEndSessionMsgs {
                        PARAMETERS { string filterName }
                        DOES {
                                IF filterName = "blue" THEN
                                        AEIP.MESSAGES.msgBlueSessionEndAus >> AEIP.CHANNELS.VOYAGER_Link;
                                        AEIP.MESSAGES.msgBlueSessionEndJpn >> AEIP.CHANNELS.VOYAGER_Link;
                                        AEIP.MESSAGES.msgBlueSessionEndCfn >> AEIP.CHANNELS.VOYAGER_Link;
                                        AEIP.MESSAGES.msgBlueSessionEndSpn >> AEIP.CHANNELS.VOYAGER_Link
                                END
                                ELSE
                                        IF filterName = "red" THEN
                                                AEIP.MESSAGES.msgRedSessionEndAus >> AEIP.CHANNELS.VOYAGER_Link;
                                                AEIP.MESSAGES.msgRedSessionEndJpn >> AEIP.CHANNELS.VOYAGER_Link;
                                                AEIP.MESSAGES.msgRedSessionEndCfn >> AEIP.CHANNELS.VOYAGER_Link;
                                                AEIP.MESSAGES.msgRedSessionEndSpn >> AEIP.CHANNELS.VOYAGER_Link
                                        END
                                        ELSE
                                                IF filterName = "green" THEN
                                                        AEIP.MESSAGES.msgGreenSessionEndAus >> AEIP.CHANNELS.VOYAGER_Link;
                                                        AEIP.MESSAGES.msgGreenSessionEndJpn >> AEIP.CHANNELS.VOYAGER_Link;
                                                        AEIP.MESSAGES.msgGreenSessionEndCfn >> AEIP.CHANNELS.VOYAGER_Link;
                                                        AEIP.MESSAGES.msgGreenSessionEndSpn >> AEIP.CHANNELS.VOYAGER_Link
                                                END
                                        END
                                END
                        }
                }
        } // FUNCTIONS

        MANAGED_ELEMENTS {
                MANAGED_ELEMENT wideAngleCamera {
                        INTERFACE_FUNCTION takePicture { }
                        INTERFACE_FUNCTION applyFilterBlue { }
                        INTERFACE_FUNCTION applyFilterRed { }
                        INTERFACE_FUNCTION applyFilterGreen { }
                        INTERFACE_FUNCTION getPixel { }
                        INTERFACE_FUNCTION countInterestingObjects { RETURNS { integer } }
                }
                MANAGED_ELEMENT narrowAngleCamera {
                        INTERFACE_FUNCTION takePicture { }
                        INTERFACE_FUNCTION applyFilterBlue { }
                        INTERFACE_FUNCTION applyFilterRed { }
                        INTERFACE_FUNCTION applyFilterGreen { }
                        INTERFACE_FUNCTION getPixel { }
                }
        }
} // AEIP

ACTIONS {
        ACTION takePicture { // take a picture of an interesting spot/object
                GUARDS { AESELF_MANAGEMENT.OTHER_POLICIES.IMAGE_PROCESSING.inTakingPicture }
                DOES {
                        IF AES.Voyager.isWideAngleImage THEN
                                call AEIP.MANAGED_ELEMENTS.wideAngleCamera.takePicture;
                                AES.Voyager.isWideAngleImage = false;
                                AS.numPixelsPerImage = 100*50  // an image has 100 x 50 pixels
                        END
                        ELSE
                                call AEIP.MANAGED_ELEMENTS.narrowAngleCamera.takePicture;
                                AES.Voyager.isWideAngleImage = true;
                                AS.numPixelsPerImage = 50*50  // an image has 50 x 50 pixels
```

```
                        END
                    }
                    TRIGGERS { EVENTS.pictureTaken    }
                }

/*              ACTION prepareImagePixelMsg {       // wraps the pixel into the msgImagePixel message
                    DOES { call AEIP.MANAGED_ELEMENTS.narrowAngleCamera.getPixel }
                }
*/
                ACTION processFilteredPicture {
                    PARAMETERS { string filterName }
                    VARS { integer numPixels }
                    DOES {
                        IF AES.Voyager.isWideAngleImage THEN
                            IF filterName = "blue" THEN
                                call AEIP.MANAGED_ELEMENTS.wideAngleCamera.applyFilterBlue
                            END;
                            IF filterName = "red" THEN
                                call AEIP.MANAGED_ELEMENTS.wideAngleCamera.applyFilterRed
                            END;
                            IF filterName = "green" THEN
                                call AEIP.MANAGED_ELEMENTS.wideAngleCamera.applyFilterGreen
                            END
                        END
                        ELSE
                            IF filterName = "blue" THEN
                                call AEIP.MANAGED_ELEMENTS.narrowAngleCamera.applyFilterBlue
                            END;
                            IF filterName = "red" THEN
                                call AEIP.MANAGED_ELEMENTS.narrowAngleCamera.applyFilterRed
                            END;
                            IF filterName = "green" THEN
                                call AEIP.MANAGED_ELEMENTS.narrowAngleCamera.applyFilterGreen
                            END
                        END;

                        call AEIP.FUNCTIONS.sendBeginSessionMsgs (filterName);

                        numPixels = 0;
                        DO {
                        //      call ACTIONS.prepareImagePixelMsg;
                                call AEIP.MANAGED_ELEMENTS.narrowAngleCamera.getPixel;
                                call AEIP.FUNCTIONS.sendImagePixelMsg;
                                numPixels = numPixels + 1
                        }       WHILE numPixels < AS.numPixelsPerImage;

                        call AEIP.FUNCTIONS.sendEndSessionMsgs (filterName)
                    }
                }

                ACTION processPicture { // process all picture pixels - apply filters and send pixels to Earth
                    GUARDS { AESELF_MANAGEMENT.OTHER_POLICIES.IMAGE_PROCESSING.inProcessingPicturePixels }
                    DOES {
                        call ACTIONS.processFilteredPicture("blue");
                        call ACTIONS.processFilteredPicture("red");
                        call ACTIONS.processFilteredPicture("green")
                    }
                    TRIGGERS { EVENTS.pictureProcessed }
                }
            } // ACTIONS

            EVENTS {
                EVENT timeToTakePicture {
                    ACTIVATION { CHANGED {      METRICS.interestingObjects} OR  PERIOD { 60 SEC } }
                }
                EVENT pictureTaken { }
                EVENT pictureProcessed { }
            } // EVENTS

            METRICS {
                METRIC interestingObjects {     // increments when a new interesting spot or object has been found
                    METRIC_TYPE { RESOURCE    }
                    METRIC_SOURCE {         AEIP.MANAGED_ELEMENTS.wideAngleCamera.countInterestingObjects }
                    DESCRIPTION {"counts the interesting spots and objects to be taken pictures of"}
                    VALUE { 0 }
                    THRESHOLD_CLASS {    integer [0~) }
                }
            }

} // AE Voyager

//==================== AE Antenna_Australia ====================
AE Antenna_Australia {

            AESELF_MANAGEMENT {
                OTHER_POLICIES {
                    POLICY IMAGE_PROCESSING {
                        FLUENT inStartingBlueImageSession {

                            INITIATED_BY { EVENTS.blueImageSessionIsAboutToStart }
                            TERMINATED_BY { EVENTS.imageSessionStartedBlue }
                        }
```

```
                FLUENT inStartingRedImageSession {

                        INITIATED_BY { EVENTS.redImageSessionIsAboutToStart }
                        TERMINATED_BY { EVENTS.imageSessionStartedRed }
                }
                FLUENT inStartingGreenImageSession {

                        INITIATED_BY { EVENTS.greenImageSessionIsAboutToStart }
                        TERMINATED_BY { EVENTS.imageSessionStartedGreen }
                }
                FLUENT inCollectingImagePixelsBlue {

                        INITIATED_BY { EVENTS.imageSessionStartedBlue }
                        TERMINATED_BY { EVENTS.imageSessionEndedBlue }
                }
                FLUENT inCollectingImagePixelsRed {

                        INITIATED_BY { EVENTS.imageSessionStartedRed }
                        TERMINATED_BY { EVENTS.imageSessionEndedRed }
                }
                FLUENT inCollectingImagePixelsGreen {

                        INITIATED_BY { EVENTS.imageSessionStartedGreen }
                        TERMINATED_BY { EVENTS.imageSessionEndedGreen }
                }
                FLUENT inSendingImage {
                        INITIATED_BY { EVENTS.imageSessionEndedGreen }
                        TERMINATED_BY { EVENTS.imageAntAustraliaSent }
                }
                MAPPING {
                        CONDITIONS { inStartingBlueImageSession }
                        DO_ACTIONS { ACTIONS.startImageCollectSession ("blue") }
                }
                MAPPING {
                        CONDITIONS { inStartingRedImageSession }
                        DO_ACTIONS { ACTIONS.startImageCollectSession ("red") }
                }
                MAPPING {
                        CONDITIONS { inStartingGreenImageSession }
                        DO_ACTIONS { ACTIONS.startImageCollectSession ("green") }
                }
                MAPPING {
                        CONDITIONS { inCollectingImagePixelsBlue }
                        DO_ACTIONS { ACTIONS.collectImagePixels ("blue") }
                }
                MAPPING {
                        CONDITIONS { inCollectingImagePixelsRed }
                        DO_ACTIONS { ACTIONS.collectImagePixels ("red") }
                }
                MAPPING {
                        CONDITIONS { inCollectingImagePixelsGreen }
                        DO_ACTIONS { ACTIONS.collectImagePixels ("green") }
                }
                MAPPING {
                        CONDITIONS { inSendingImage }
                        DO_ACTIONS { ACTIONS.sendImage }
                }
        }
    }
} // AESELF_MANAGEMENT

//====== AEIP for this AE ======
AEIP {
    FUNCTIONS {
        FUNCTION receiveImagePixelMsg {
            DOES { AES.Voyager.AEIP.MESSAGES.msgImagePixel << AES.Voyager.AEIP.CHANNELS.VOYAGER_Link  }
        }
        FUNCTION receiveSessionBeginMsg {
            PARAMETERS {    string filterName }
            DOES {
                IF filterName = "blue" THEN
                    AES.Voyager.AEIP.MESSAGES.msgBlueSessionBeginAus << AES.Voyager.AEIP.CHANNELS.VOYAGER_Link
                END
                ELSE
                    IF filterName = "red" THEN
                        AES.Voyager.AEIP.MESSAGES.msgRedSessionBeginAus << AES.Voyager.AEIP.CHANNELS.VOYAGER_Link
                    END
                    ELSE
                        IF filterName = "green" THEN
                            AES.Voyager.AEIP.MESSAGES.msgGreenSessionBeginAus << AES.Voyager.AEIP.CHANNELS.VOYAGER_Link
                        END
                    END
                END
            }
        }
        FUNCTION receiveSessionEndMsg {
            PARAMETERS {    string filterName }
            DOES {
                IF filterName = "blue" THEN
                    AES.Voyager.AEIP.MESSAGES.msgBlueSessionEndAus << AES.Voyager.AEIP.CHANNELS.VOYAGER_Link
                END
                ELSE
                    IF filterName = "red" THEN
```

```
                                        AES.Voyager.AEIP.MESSAGES.msgRedSessionEndAus << AES.Voyager.AEIP.CHANNELS.VOYAGER_Link
                                    END
                                    ELSE
                                        IF filterName = "green" THEN
                                            AES.Voyager.AEIP.MESSAGES.msgGreenSessionEndAus << AES.Voyager.AEIP.CHANNELS.VOYAGER_Link
                                        END
                                    END
                                END
                            }
                        }
                    }
                    MANAGED_ELEMENTS { }
            }

            ACTIONS {
                    ACTION startImageCollectSession {
                        PARAMETERS {    string filterName }
                        GUARDS { AESELF_MANAGEMENT.OTHER_POLICIES.IMAGE_PROCESSING.inStartingBlueImageSession OR
                                     AESELF_MANAGEMENT.OTHER_POLICIES.IMAGE_PROCESSING.inStartingRedImageSession OR
                                     AESELF_MANAGEMENT.OTHER_POLICIES.IMAGE_PROCESSING.inStartingGreenImageSession }
                        DOES {
                            CALL AEIP.FUNCTIONS.receiveSessionBeginMsg (filterName)
                        }
                    }
                    ACTION collectImagePixels {
                        PARAMETERS {    string filterName }
                        GUARDS { AESELF_MANAGEMENT.OTHER_POLICIES.IMAGE_PROCESSING.inCollectingImagePixelsBlue OR
                                     AESELF_MANAGEMENT.OTHER_POLICIES.IMAGE_PROCESSING.inCollectingImagePixelsRed OR
                                     AESELF_MANAGEMENT.OTHER_POLICIES.IMAGE_PROCESSING.inCollectingImagePixelsGreen          }

                        VARS { integer numPixels }
                        DOES {
                            numPixels = 0;
                            DO {
                                CALL AEIP.FUNCTIONS.receiveImagePixelMsg;
                                numPixels = numPixels + 1
                            } WHILE numPixels < AS.numPixelsPerImage ;

                            CALL AEIP.FUNCTIONS.receiveSessionEndMsg (filterName)
                        }
                    }
                    ACTION IMPL prepareImage { }
                    ACTION sendImage {
                        GUARDS { AESELF_MANAGEMENT.OTHER_POLICIES.IMAGE_PROCESSING.inSendingImage }
                        DOES {
                            CALL IMPL ACTIONS.prepareImage;
                            CALL ASIP.FUNCTIONS.sendImageMsg("Antenna_Australia");
                            CALL ASIP.FUNCTIONS.receiveImageMsg("Antenna_Australia")
                        }
                    }
            } // ACTIONS

            EVENTS {
                    EVENT blueImageSessionIsAboutToStart { ACTIVATION { SENT { AES.Voyager.AEIP.MESSAGES.msgBlueSessionBeginAus } } }
                    EVENT redImageSessionIsAboutToStart { ACTIVATION { SENT { AES.Voyager.AEIP.MESSAGES.msgRedSessionBeginAus } } }
                    EVENT greenImageSessionIsAboutToStart { ACTIVATION { SENT { AES.Voyager.AEIP.MESSAGES.msgGreenSessionBeginAus } } }
                    EVENT imageSessionStartedBlue { ACTIVATION { RECEIVED { AES.Voyager.AEIP.MESSAGES.msgBlueSessionBeginAus } } }
                    EVENT imageSessionEndedBlue { ACTIVATION { RECEIVED { AES.Voyager.AEIP.MESSAGES.msgBlueSessionEndAus } } }
                    EVENT imageSessionStartedRed { ACTIVATION { RECEIVED { AES.Voyager.AEIP.MESSAGES.msgRedSessionBeginAus } } }
                    EVENT imageSessionEndedRed { ACTIVATION { RECEIVED { AES.Voyager.AEIP.MESSAGES.msgRedSessionEndAus } } }
                    EVENT imageSessionStartedGreen { ACTIVATION { RECEIVED { AES.Voyager.AEIP.MESSAGES.msgGreenSessionBeginAus } } }
                    EVENT imageSessionEndedGreen { ACTIVATION { RECEIVED { AES.Voyager.AEIP.MESSAGES.msgGreenSessionEndAus } } }
                    EVENT imageAntAustraliaSent { ACTIVATION { SENT { ASIP.MESSAGES.msgImageAntAustralia } } }
            } // EVENTS

    }

    //=================== AE Antenna_Japan ===================
    AE Antenna_Japan {

            AESELF_MANAGEMENT {
                    OTHER_POLICIES {
                        POLICY IMAGE_PROCESSING {
                            FLUENT inStartingBlueImageSession {

                                INITIATED_BY { EVENTS.blueImageSessionIsAboutToStart }
                                TERMINATED_BY { EVENTS.imageSessionStartedBlue }
                            }
                            FLUENT inStartingRedImageSession {

                                INITIATED_BY { EVENTS.redImageSessionIsAboutToStart }
                                TERMINATED_BY { EVENTS.imageSessionStartedRed }
                            }
                            FLUENT inStartingGreenImageSession {

                                INITIATED_BY { EVENTS.greenImageSessionIsAboutToStart }
                                TERMINATED_BY { EVENTS.imageSessionStartedGreen }
                            }
                            FLUENT inCollectingImagePixelsBlue {

                                INITIATED_BY { EVENTS.imageSessionStartedBlue }
                                TERMINATED_BY { EVENTS.imageSessionEndedBlue }
                            }
```

```
                FLUENT inCollectingImagePixelsRed {

                        INITIATED_BY { EVENTS.imageSessionStartedRed }
                        TERMINATED_BY { EVENTS.imageSessionEndedRed }
                }
                FLUENT inCollectingImagePixelsGreen {

                        INITIATED_BY { EVENTS.imageSessionStartedGreen }
                        TERMINATED_BY { EVENTS.imageSessionEndedGreen }
                }
                FLUENT inSendingImage {
                        INITIATED_BY { EVENTS.imageSessionEndedGreen }
                        TERMINATED_BY { EVENTS.imageAntJapanSent }
                }
                MAPPING {
                        CONDITIONS { inStartingBlueImageSession }
                        DO_ACTIONS { ACTIONS.startImageCollectSession ("blue") }
                }
                MAPPING {
                        CONDITIONS { inStartingRedImageSession }
                        DO_ACTIONS { ACTIONS.startImageCollectSession ("red") }
                }
                MAPPING {
                        CONDITIONS { inStartingGreenImageSession }
                        DO_ACTIONS { ACTIONS.startImageCollectSession ("green") }
                }
                MAPPING {
                        CONDITIONS { inCollectingImagePixelsBlue }
                        DO_ACTIONS { ACTIONS.collectImagePixels ("blue") }
                }
                MAPPING {
                        CONDITIONS { inCollectingImagePixelsRed }
                        DO_ACTIONS { ACTIONS.collectImagePixels ("red") }
                }
                MAPPING {
                        CONDITIONS { inCollectingImagePixelsGreen }
                        DO_ACTIONS { ACTIONS.collectImagePixels ("green") }
                }
                MAPPING {
                        CONDITIONS { inSendingImage }
                        DO_ACTIONS { ACTIONS.sendImage }
                }
        }
    }
} // AESELF_MANAGEMENT

//====== AEIP for this AE ======
AEIP {
    FUNCTIONS {
        FUNCTION receiveImagePixelMsg {
            DOES { AES.Voyager.AEIP.MESSAGES.msgImagePixel << AES.Voyager.AEIP.CHANNELS.VOYAGER_Link  }
        }
        FUNCTION receiveSessionBeginMsg {
            PARAMETERS {    string filterName }
            DOES {
                IF filterName = "blue" THEN
                    AES.Voyager.AEIP.MESSAGES.msgBlueSessionBeginJpn << AES.Voyager.AEIP.CHANNELS.VOYAGER_Link
                END
                ELSE
                    IF filterName = "red" THEN
                        AES.Voyager.AEIP.MESSAGES.msgRedSessionBeginJpn << AES.Voyager.AEIP.CHANNELS.VOYAGER_Link
                    END
                    ELSE
                        IF filterName = "green" THEN
                            AES.Voyager.AEIP.MESSAGES.msgGreenSessionBeginJpn << AES.Voyager.AEIP.CHANNELS.VOYAGER_Link
                        END
                    END
                END
            }
        }
        FUNCTION receiveSessionEndMsg {
            PARAMETERS {    string filterName }
            DOES {
                IF filterName = "blue" THEN
                    AES.Voyager.AEIP.MESSAGES.msgBlueSessionEndJpn << AES.Voyager.AEIP.CHANNELS.VOYAGER_Link
                END
                ELSE
                    IF filterName = "red" THEN
                        AES.Voyager.AEIP.MESSAGES.msgRedSessionEndJpn << AES.Voyager.AEIP.CHANNELS.VOYAGER_Link
                    END
                    ELSE
                        IF filterName = "green" THEN
                            AES.Voyager.AEIP.MESSAGES.msgGreenSessionEndJpn << AES.Voyager.AEIP.CHANNELS.VOYAGER_Link
                        END
                    END
                END
            }
        }
    }
    MANAGED_ELEMENTS { }
}

ACTIONS {
```

```
            ACTION startImageCollectSession {
                PARAMETERS {    string filterName }
                GUARDS { AESELF_MANAGEMENT.OTHER_POLICIES.IMAGE_PROCESSING.inStartingBlueImageSession OR
                                AESELF_MANAGEMENT.OTHER_POLICIES.IMAGE_PROCESSING.inStartingRedImageSession OR
                                AESELF_MANAGEMENT.OTHER_POLICIES.IMAGE_PROCESSING.inStartingGreenImageSession }
                DOES {
                    CALL AEIP.FUNCTIONS.receiveSessionBeginMsg (filterName)
                }
            }
            ACTION collectImagePixels {
                PARAMETERS {    string filterName }
                GUARDS { AESELF_MANAGEMENT.OTHER_POLICIES.IMAGE_PROCESSING.inCollectingImagePixelsBlue OR
                                AESELF_MANAGEMENT.OTHER_POLICIES.IMAGE_PROCESSING.inCollectingImagePixelsRed OR
                                AESELF_MANAGEMENT.OTHER_POLICIES.IMAGE_PROCESSING.inCollectingImagePixelsGreen          }

                VARS { integer numPixels }
                DOES {
                    numPixels = 0;
                    DO {
                        CALL AEIP.FUNCTIONS.receiveImagePixelMsg;
                        numPixels = numPixels + 1
                    } WHILE numPixels < AS.numPixelsPerImage ;
                    CALL AEIP.FUNCTIONS.receiveSessionEndMsg (filterName)
                }
            }
            ACTION IMPL prepareImage { }
            ACTION sendImage {
                GUARDS { AESELF_MANAGEMENT.OTHER_POLICIES.IMAGE_PROCESSING.inSendingImage }
                DOES {
                    CALL IMPL ACTIONS.prepareImage;
                    CALL ASIP.FUNCTIONS.sendImageMsg("Antenna_Japan");
                    CALL ASIP.FUNCTIONS.receiveImageMsg("Antenna_Japan")
                }
            }
        } // ACTIONS

        EVENTS {
            EVENT blueImageSessionIsAboutToStart { ACTIVATION { SENT { AES.Voyager.AEIP.MESSAGES.msgBlueSessionBeginJpn } } }
            EVENT redImageSessionIsAboutToStart { ACTIVATION { SENT { AES.Voyager.AEIP.MESSAGES.msgRedSessionBeginJpn } } }
            EVENT greenImageSessionIsAboutToStart { ACTIVATION { SENT { AES.Voyager.AEIP.MESSAGES.msgGreenSessionBeginJpn } } }
            EVENT imageSessionStartedBlue { ACTIVATION { RECEIVED { AES.Voyager.AEIP.MESSAGES.msgBlueSessionBeginJpn } } }
            EVENT imageSessionEndedBlue { ACTIVATION { RECEIVED { AES.Voyager.AEIP.MESSAGES.msgBlueSessionEndJpn } } }
            EVENT imageSessionStartedRed { ACTIVATION { RECEIVED { AES.Voyager.AEIP.MESSAGES.msgRedSessionBeginJpn } } }
            EVENT imageSessionEndedRed { ACTIVATION { RECEIVED { AES.Voyager.AEIP.MESSAGES.msgRedSessionEndJpn } } }
            EVENT imageSessionStartedGreen { ACTIVATION { RECEIVED { AES.Voyager.AEIP.MESSAGES.msgGreenSessionBeginJpn } } }
            EVENT imageSessionEndedGreen { ACTIVATION { RECEIVED { AES.Voyager.AEIP.MESSAGES.msgGreenSessionEndJpn } } }
            EVENT imageAntJapanSent { ACTIVATION { SENT {       ASIP.MESSAGES.msgImageAntJapan } } }
        } // EVENTS
}

//==================== AE Antenna_California ====================
AE Antenna_California {

        AESELF_MANAGEMENT {
            OTHER_POLICIES {
                POLICY IMAGE_PROCESSING {
                    FLUENT inStartingBlueImageSession {

                        INITIATED_BY { EVENTS.blueImageSessionIsAboutToStart }
                        TERMINATED_BY { EVENTS.imageSessionStartedBlue }
                    }
                    FLUENT inStartingRedImageSession {

                        INITIATED_BY { EVENTS.redImageSessionIsAboutToStart }
                        TERMINATED_BY { EVENTS.imageSessionStartedRed }
                    }
                    FLUENT inStartingGreenImageSession {

                        INITIATED_BY { EVENTS.greenImageSessionIsAboutToStart }
                        TERMINATED_BY { EVENTS.imageSessionStartedGreen }
                    }
                    FLUENT inCollectingImagePixelsBlue {

                        INITIATED_BY { EVENTS.imageSessionStartedBlue }
                        TERMINATED_BY { EVENTS.imageSessionEndedBlue }
                    }
                    FLUENT inCollectingImagePixelsRed {

                        INITIATED_BY { EVENTS.imageSessionStartedRed }
                        TERMINATED_BY { EVENTS.imageSessionEndedRed }
                    }
                    FLUENT inCollectingImagePixelsGreen {

                        INITIATED_BY { EVENTS.imageSessionStartedGreen }
                        TERMINATED_BY { EVENTS.imageSessionEndedGreen }
                    }
                    FLUENT inSendingImage {
                        INITIATED_BY { EVENTS.imageSessionEndedGreen }
                        TERMINATED_BY { EVENTS.imageAntCaliforniaSent }
                    }
                    MAPPING {
                        CONDITIONS { inStartingBlueImageSession }
                        DO_ACTIONS { ACTIONS.startImageCollectSession ("blue") }
```

```
                }
                MAPPING {
                        CONDITIONS { inStartingRedImageSession }
                        DO_ACTIONS { ACTIONS.startImageCollectSession ("red") }
                }
                MAPPING {
                        CONDITIONS { inStartingGreenImageSession }
                        DO_ACTIONS { ACTIONS.startImageCollectSession ("green") }
                }
                MAPPING {
                        CONDITIONS { inCollectingImagePixelsBlue }
                        DO_ACTIONS { ACTIONS.collectImagePixels ("blue") }
                }
                MAPPING {
                        CONDITIONS { inCollectingImagePixelsRed }
                        DO_ACTIONS { ACTIONS.collectImagePixels ("red") }
                }
                MAPPING {
                        CONDITIONS { inCollectingImagePixelsGreen }
                        DO_ACTIONS { ACTIONS.collectImagePixels ("green") }
                }
                MAPPING {
                        CONDITIONS { inSendingImage }
                        DO_ACTIONS { ACTIONS.sendImage }
                }
        }
    }
} // AESELF_MANAGEMENT

//====== AEIP for this AE ======
AEIP {
    FUNCTIONS {
        FUNCTION receiveImagePixelMsg {
            DOES { AES.Voyager.AEIP.MESSAGES.msgImagePixel << AES.Voyager.AEIP.CHANNELS.VOYAGER_Link  }
        }
        FUNCTION receiveSessionBeginMsg {
            PARAMETERS {    string filterName }
            DOES {
                IF filterName = "blue" THEN
                    AES.Voyager.AEIP.MESSAGES.msgBlueSessionBeginCfn << AES.Voyager.AEIP.CHANNELS.VOYAGER_Link
                END
                ELSE
                    IF filterName = "red" THEN
                        AES.Voyager.AEIP.MESSAGES.msgRedSessionBeginCfn << AES.Voyager.AEIP.CHANNELS.VOYAGER_Link
                    END
                    ELSE
                        IF filterName = "green" THEN
                            AES.Voyager.AEIP.MESSAGES.msgGreenSessionBeginCfn << AES.Voyager.AEIP.CHANNELS.VOYAGER_Link
                        END
                    END
                END
            }
        }
        FUNCTION receiveSessionEndMsg {
            PARAMETERS {    string filterName }
            DOES {
                IF filterName = "blue" THEN
                    AES.Voyager.AEIP.MESSAGES.msgBlueSessionEndCfn << AES.Voyager.AEIP.CHANNELS.VOYAGER_Link
                END
                ELSE
                    IF filterName = "red" THEN
                        AES.Voyager.AEIP.MESSAGES.msgRedSessionEndCfn << AES.Voyager.AEIP.CHANNELS.VOYAGER_Link
                    END
                    ELSE
                        IF filterName = "green" THEN
                            AES.Voyager.AEIP.MESSAGES.msgGreenSessionEndCfn << AES.Voyager.AEIP.CHANNELS.VOYAGER_Link
                        END
                    END
                END
            }
        }
    }
    MANAGED_ELEMENTS { }
}

ACTIONS {
    ACTION startImageCollectSession {
        PARAMETERS {    string filterName }
        GUARDS { AESELF_MANAGEMENT.OTHER_POLICIES.IMAGE_PROCESSING.inStartingBlueImageSession OR
                 AESELF_MANAGEMENT.OTHER_POLICIES.IMAGE_PROCESSING.inStartingRedImageSession OR
                 AESELF_MANAGEMENT.OTHER_POLICIES.IMAGE_PROCESSING.inStartingGreenImageSession }
        DOES {
            CALL AEIP.FUNCTIONS.receiveSessionBeginMsg (filterName)
        }
    }
    ACTION collectImagePixels {
        PARAMETERS {    string filterName }
        GUARDS { AESELF_MANAGEMENT.OTHER_POLICIES.IMAGE_PROCESSING.inCollectingImagePixelsBlue OR
                 AESELF_MANAGEMENT.OTHER_POLICIES.IMAGE_PROCESSING.inCollectingImagePixelsRed OR
                 AESELF_MANAGEMENT.OTHER_POLICIES.IMAGE_PROCESSING.inCollectingImagePixelsGreen          }

        VARS { integer numPixels }
        DOES {
```

```
                        numPixels = 0;
                        DO {
                                CALL AEIP.FUNCTIONS.receiveImagePixelMsg;
                                numPixels = numPixels + 1
                        } WHILE numPixels < AS.numPixelsPerImage ;
                        CALL AEIP.FUNCTIONS.receiveSessionEndMsg (filterName)
                }
        }
        ACTION IMPL prepareImage { }
        ACTION sendImage {
                GUARDS { AESELF_MANAGEMENT.OTHER_POLICIES.IMAGE_PROCESSING.inSendingImage }
                DOES {
                        CALL IMPL ACTIONS.prepareImage;
                        CALL ASIP.FUNCTIONS.sendImageMsg("Antenna_California");
                        CALL ASIP.FUNCTIONS.receiveImageMsg("Antenna_California")
                }
        }
    } // ACTIONS

    EVENTS {
        EVENT blueImageSessionIsAboutToStart { ACTIVATION { SENT { AES.Voyager.AEIP.MESSAGES.msgBlueSessionBeginCfn } } }
        EVENT redImageSessionIsAboutToStart { ACTIVATION { SENT { AES.Voyager.AEIP.MESSAGES.msgRedSessionBeginCfn } } }
        EVENT greenImageSessionIsAboutToStart { ACTIVATION { SENT { AES.Voyager.AEIP.MESSAGES.msgGreenSessionBeginCfn } } }
        EVENT imageSessionStartedBlue { ACTIVATION { RECEIVED { AES.Voyager.AEIP.MESSAGES.msgBlueSessionBeginCfn } } }
        EVENT imageSessionEndedBlue { ACTIVATION { RECEIVED { AES.Voyager.AEIP.MESSAGES.msgBlueSessionEndCfn } } }
        EVENT imageSessionStartedRed { ACTIVATION { RECEIVED { AES.Voyager.AEIP.MESSAGES.msgRedSessionBeginCfn } } }
        EVENT imageSessionEndedRed { ACTIVATION { RECEIVED { AES.Voyager.AEIP.MESSAGES.msgRedSessionEndCfn } } }
        EVENT imageSessionStartedGreen { ACTIVATION { RECEIVED { AES.Voyager.AEIP.MESSAGES.msgGreenSessionBeginCfn } } }
        EVENT imageSessionEndedGreen { ACTIVATION { RECEIVED { AES.Voyager.AEIP.MESSAGES.msgGreenSessionEndCfn } } }
        EVENT imageAntCaliforniaSent { ACTIVATION { SENT {   ASIP.MESSAGES.msgImageAntCalifornia } } }
    } // EVENTS
}

//==================== AE Antenna_Spain ====================
AE Antenna_Spain {

    AESELF_MANAGEMENT {
        OTHER_POLICIES {
                POLICY IMAGE_PROCESSING {
                        FLUENT inStartingBlueImageSession {

                                INITIATED_BY { EVENTS.blueImageSessionIsAboutToStart }
                                TERMINATED_BY { EVENTS.imageSessionStartedBlue }
                        }
                        FLUENT inStartingRedImageSession {

                                INITIATED_BY { EVENTS.redImageSessionIsAboutToStart }
                                TERMINATED_BY { EVENTS.imageSessionStartedRed }
                        }
                        FLUENT inStartingGreenImageSession {

                                INITIATED_BY { EVENTS.greenImageSessionIsAboutToStart }
                                TERMINATED_BY { EVENTS.imageSessionStartedGreen }
                        }
                        FLUENT inCollectingImagePixelsBlue {

                                INITIATED_BY { EVENTS.imageSessionStartedBlue }
                                TERMINATED_BY { EVENTS.imageSessionEndedBlue }
                        }
                        FLUENT inCollectingImagePixelsRed {

                                INITIATED_BY { EVENTS.imageSessionStartedRed }
                                TERMINATED_BY { EVENTS.imageSessionEndedRed }
                        }
                        FLUENT inCollectingImagePixelsGreen {

                                INITIATED_BY { EVENTS.imageSessionStartedGreen }
                                TERMINATED_BY { EVENTS.imageSessionEndedGreen }
                        }
                        FLUENT inSendingImage {
                                INITIATED_BY { EVENTS.imageSessionEndedGreen }
                                TERMINATED_BY { EVENTS.imageAntSpainSent }
                        }
                        MAPPING {
                                CONDITIONS { inStartingBlueImageSession }
                                DO_ACTIONS { ACTIONS.startImageCollectSession ("blue") }
                        }
                        MAPPING {
                                CONDITIONS { inStartingRedImageSession }
                                DO_ACTIONS { ACTIONS.startImageCollectSession ("red") }
                        }
                        MAPPING {
                                CONDITIONS { inStartingGreenImageSession }
                                DO_ACTIONS { ACTIONS.startImageCollectSession ("green") }
                        }
                        MAPPING {
                                CONDITIONS { inCollectingImagePixelsBlue }
                                DO_ACTIONS { ACTIONS.collectImagePixels ("blue") }
                        }
                        MAPPING {
                                CONDITIONS { inCollectingImagePixelsRed }
                                DO_ACTIONS { ACTIONS.collectImagePixels ("red") }
                        }
```

```
                MAPPING {
                        CONDITIONS { inCollectingImagePixelsGreen }
                        DO_ACTIONS { ACTIONS.collectImagePixels ("green") }
                }
                MAPPING {
                        CONDITIONS { inSendingImage }
                        DO_ACTIONS { ACTIONS.sendImage }
                }
        }
    }
} // AESELF_MANAGEMENT


//====== AEIP for this AE ======
AEIP {
    FUNCTIONS {
        FUNCTION receiveImagePixelMsg {
                DOES { AES.Voyager.AEIP.MESSAGES.msgImagePixel << AES.Voyager.AEIP.CHANNELS.VOYAGER_Link  }
        }
        FUNCTION receiveSessionBeginMsg {
                PARAMETERS {    string filterName }
                DOES {
                        IF filterName = "blue" THEN
                                AES.Voyager.AEIP.MESSAGES.msgBlueSessionBeginSpn << AES.Voyager.AEIP.CHANNELS.VOYAGER_Link
                        END
                        ELSE
                                IF filterName = "red" THEN
                                        AES.Voyager.AEIP.MESSAGES.msgRedSessionBeginSpn << AES.Voyager.AEIP.CHANNELS.VOYAGER_Link
                                END
                                ELSE
                                        IF filterName = "green" THEN
                                                AES.Voyager.AEIP.MESSAGES.msgGreenSessionBeginSpn << AES.Voyager.AEIP.CHANNELS.VOYAGER_Link
                                        END
                                END
                        END
                }
        }
        FUNCTION receiveSessionEndMsg {
                PARAMETERS {    string filterName }
                DOES {
                        IF filterName = "blue" THEN
                                AES.Voyager.AEIP.MESSAGES.msgBlueSessionEndSpn << AES.Voyager.AEIP.CHANNELS.VOYAGER_Link
                        END
                        ELSE
                                IF filterName = "red" THEN
                                        AES.Voyager.AEIP.MESSAGES.msgRedSessionEndSpn << AES.Voyager.AEIP.CHANNELS.VOYAGER_Link
                                END
                                ELSE
                                        IF filterName = "green" THEN
                                                AES.Voyager.AEIP.MESSAGES.msgGreenSessionEndSpn << AES.Voyager.AEIP.CHANNELS.VOYAGER_Link
                                        END
                                END
                        END
                }
        }
    }
    MANAGED_ELEMENTS { }
} // AEIP

ACTIONS {
    ACTION startImageCollectSession {
        PARAMETERS {    string filterName }
        GUARDS { AESELF_MANAGEMENT.OTHER_POLICIES.IMAGE_PROCESSING.inStartingBlueImageSession OR
                 AESELF_MANAGEMENT.OTHER_POLICIES.IMAGE_PROCESSING.inStartingRedImageSession OR
                 AESELF_MANAGEMENT.OTHER_POLICIES.IMAGE_PROCESSING.inStartingGreenImageSession }
        DOES {
                CALL AEIP.FUNCTIONS.receiveSessionBeginMsg (filterName)
        }
    }
    ACTION collectImagePixels {
        PARAMETERS {    string filterName }
        GUARDS { AESELF_MANAGEMENT.OTHER_POLICIES.IMAGE_PROCESSING.inCollectingImagePixelsBlue OR
                 AESELF_MANAGEMENT.OTHER_POLICIES.IMAGE_PROCESSING.inCollectingImagePixelsRed OR
                 AESELF_MANAGEMENT.OTHER_POLICIES.IMAGE_PROCESSING.inCollectingImagePixelsGreen          }

        VARS { integer numPixels }
        DOES {
                numPixels = 0;
                DO {
                        CALL AEIP.FUNCTIONS.receiveImagePixelMsg;
                        numPixels = numPixels + 1
                } WHILE numPixels < AS.numPixelsPerImage ;
                CALL AEIP.FUNCTIONS.receiveSessionEndMsg (filterName)
        }
    }
    ACTION IMPL prepareImage { }
    ACTION sendImage {
        GUARDS { AESELF_MANAGEMENT.OTHER_POLICIES.IMAGE_PROCESSING.inSendingImage }
        DOES {
                CALL IMPL ACTIONS.prepareImage;
                CALL ASIP.FUNCTIONS.sendImageMsg("Antenna_Spain");
                CALL ASIP.FUNCTIONS.receiveImageMsg("Antenna_Spain")
        }
```

```
            }
        } // ACTIONS

        EVENTS {
            EVENT blueImageSessionIsAboutToStart { ACTIVATION { SENT { AES.Voyager.AEIP.MESSAGES.msgBlueSessionBeginSpn } } }
            EVENT redImageSessionIsAboutToStart { ACTIVATION { SENT { AES.Voyager.AEIP.MESSAGES.msgRedSessionBeginSpn } } }
            EVENT greenImageSessionIsAboutToStart { ACTIVATION { SENT { AES.Voyager.AEIP.MESSAGES.msgGreenSessionBeginSpn } } }
            EVENT imageSessionStartedBlue { ACTIVATION { RECEIVED { AES.Voyager.AEIP.MESSAGES.msgBlueSessionBeginSpn } } }
            EVENT imageSessionEndedBlue { ACTIVATION { RECEIVED { AES.Voyager.AEIP.MESSAGES.msgBlueSessionEndSpn } } }
            EVENT imageSessionStartedRed { ACTIVATION { RECEIVED { AES.Voyager.AEIP.MESSAGES.msgRedSessionBeginSpn } } }
            EVENT imageSessionEndedRed { ACTIVATION { RECEIVED { AES.Voyager.AEIP.MESSAGES.msgRedSessionEndSpn } } }
            EVENT imageSessionStartedGreen { ACTIVATION { RECEIVED { AES.Voyager.AEIP.MESSAGES.msgGreenSessionBeginSpn } } }
            EVENT imageSessionEndedGreen { ACTIVATION { RECEIVED { AES.Voyager.AEIP.MESSAGES.msgGreenSessionEndSpn } } }
            EVENT imageAntSpainSent { ACTIVATION { SENT {        ASIP.MESSAGES.msgImageAntSpain } } }
        } // EVENTS
    }
} // AES
```