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

Emil Vassev
Lero–the Irish Software Engineering Research Center
University College Dublin
Dublin, Ireland
emil.vassev@lero.ie

Mike Hinchey
Lero–the Irish Software Engineering Research Center
University of Limerick
Limerick, Ireland
mike.hinchey@lero.ie

***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.**

Keywords - Voyager Mission, ASSL, code generation

I. 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.

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 research characteristics in terms of applying ASSL to the problem of specification and prototype generation of the NASA Voyager Mission. Section 4 presents experimental results, and finally, Section 5 concludes the paper with a description of benefits for space systems, concluding remarks, and future work.

II. 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.

A. 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 (cf. 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 xtended 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 (cf. 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.

B. *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.

III. 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.

A. *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 (cf. 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*.

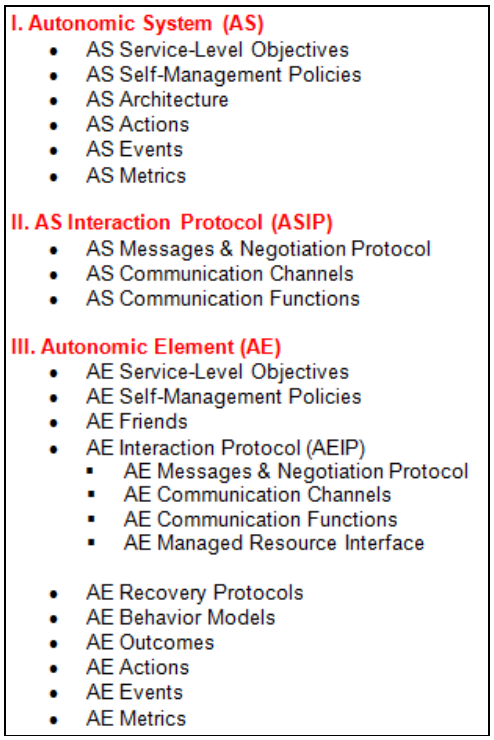

Figure 2. ASSL multi-tier specification model.

B. *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.

C. *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 (cf. Section II.A). 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.

IV. 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.

A. *Specifying Voyager Mission with ASSL*

In order to specify the algorithm described in Section III.C, 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 (cf. Section III.A). 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). A report [9] issued at Lero (the Irish Software Engineering Research Center) contains both the complete specification and evaluation results.

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 [9]. 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.

2) ASIP Tier specification: At this tier, we specified the AS-level communication protocol – the autonomic system interaction protocol (ASIP) (cf. Section III.A). 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** ; cf. [9]) 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 [9] for the ASSL specification of the Voyager ASIP.

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 [9] for the complete specification of the AE Tier.

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) (cf. III.A), 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 (cf. [9]):

- 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 (cf. [9] 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 [9].

```
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 [9]. These communication functions are called by the AE actions.

B. 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.

1) Prototype's 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 (cf. Figure 3). Similar to the applied ASSL specification model (cf. Section III.A), 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.

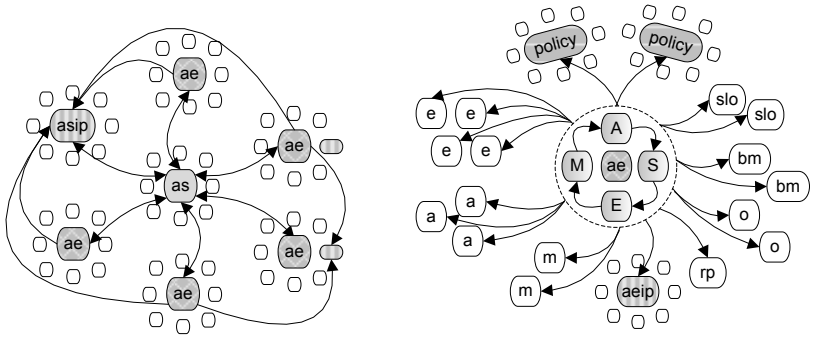

Figure 3. (a) AS Runtime Object Model; (b) AE Runtime Object Model

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].

2) Prototype 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 (cf. record 99). This event is specified in the Voyager AE to run on regular basis every 60 seconds (cf. Section IV.A.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 (cf. record 101), which triggered the PICTURETAKEN event (cf. 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 (cf. record 108), which executed the PROCESSFILTEREDPICTURE action three times (cf. 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 [9] **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 (cf. record 110 for the **BLUE**IMAGESESSIONISABOUTTOSTART event). Next these events initiated in the antenna AEs three **inStarting**[color]**ImageSession** fluents, one per filter color (cf. record 113 for the INSTARTING**BLUE**IMAGESESSION fluent).

Each of these fluents prompted the execution of the STARTIMAGECOLLECTSESSION action (cf. 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 [9]. 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 (cf. Section IV.A.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** – cf. [9]) that prompted the execution of a special action (termed **sendImage**; cf. [9]). 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 (cf. record 99), which occurs every 60 seconds (cf. the **timeToTakePicture** ASSL specification in Section IV.A.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.

V. 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.

A. 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 [10] 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.

B. 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.

Acknowledgment

This work was supported in part by an IRCSET postdoctoral fellowship grant (now termed EMPOWER) at University College Dublin, Ireland, and by the Science Foundation Ireland grant 03/CE2/I303_1 to Lero–the Irish Software Engineering Research Centre.

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 strory 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 and M. Hinchey, "ASSL specification model for the mage-processing behavior in the NASA Voyager Mission, Technical Report Lero-2009-1, Lero—the Irish Software Engineering Research Center, 2009.

[10] 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.