Why We Use SPARK/Ada

ELaNa IV lessons for CubeSat software:

• NASA’s 2010 CubeSat Launch Initiative (ELaNa)

• Our project was in the first group selected for launch

• Our single-unit CubeSat was launched as part of NASA’s ELaNa IV on an Air Force ORS-3 Minotaur 1 flight November 19, 2013 to a 500 km altitude, 40.5° inclination orbit and remained in orbit until reentry over the central Pacific Ocean, November 21, 2016, after two years and two days. Eight others were never heard from, two had partial contact for a few days, and one worked for 4 months.

• The Vermont Lunar CubeSat tested components of a Lunar navigation system in Low Earth Orbit
It worked until our reentry on November 21, 2015:

• We completed 11,071 orbits.

• We travelled about 293,000,000 miles, equivalent to over 3/4 the distance to Jupiter.

• Our single-unit CubeSat was launched as part of NASA’s ELaNa IV on an Air Force ORS-3 Minotaur 1 flight November 19, 2013 to a 500 km altitude, 40.5° inclination orbit and remained in orbit until November 21, 2016. It is the only one of the 12 ELaNa IV university CubeSats that operated until reentry, the last one quit 19 months earlier.

• We communicated with it the day before reentry

• We are the only successful university satellite on the east coast

• Follow our project at cubesatlab.org
Vermont Lunar CubeSat *SPARK 2005* software

- 5991 lines of code
- 4095 lines of comments (2843 are SPARK annotations)
- A total of 10,086 lines (not including blank lines)
- The Examiner generated 4542 verification conditions
- All but 102 were proved automatically (98%)
- We attempted to prove the program free of runtime errors
- Which allowed us to suppress all checks
- The C portion consisted of 2239 lines (including blank lines)
- Additional provers in SPARK 2014 would improve this
Vermont Lunar CubeSat (10 cm cube, 1 kg)
ELaNa IV Launch Minotaur 1 – Wallops Island
November 19, 2013, 8:15 PM

First two stages are Minuteman II, third and fourth stages are Pegasus second and third stages

Brandon & Chapin- NASA Academy of Aerospace Quality 2017
Vermont Lunar CubeSat

Our first picture of Earth, The North coast of Western Australia
Clouds over the ocean, June 2015, 19 months after launch.
Software Development Comments
for our first CubeSat

- SPARK caught errors as we refactored the software as we developed greater understanding of the hardware

- SPARK helped the discipline of the software during turnover as some students graduated and were replaced

- Although we did not have a formal development process, without SPARK we probably would not have completed the project with the limited personnel resources and tight time constraint

- Although the CubeSat is limited to 1.3kg, the paperwork is unlimited 😊
Four aerospace software failures that would have been prevented with SPARK/Ada:

- Mars Science Laboratory Sol-200 Memory Anomaly
- Ariane 5 initial flight failure
- Boeing 787 generator control computer shutdown
- Boeing 787 avionics reset
Language Comparison

UK Ministry of Defense C-130J software study:
The anomalies per 1,000 lines of code (average):
• for C was 97
• for Ada 95 was 25
• for SPARK/Ada 95 was 4

Newer Tokeneer project (for NSA)
• For SPARK/Ada 2005 was 0.4

Productivity of 38 lines of code per programmer day, compared with 10 to 12 lines of code when using C.

We are now using the even newer SPARK/Ada 2014
Mars Science Laboratory
Sol-200 Memory Anomaly

• Six months after landing on Mars, uncorrectable errors in the NAND flash memory led to an inability of the Mars Science Laboratory (MSL) prime computer to turn off for its normal recharge session.

• This potentially fatal error was apparently due to two pieces of its C software having pointers which pointed to the same memory. Curiosity has about 3.5 MLOC written in C. (One would expect about 35,000 errors, they have corrected about 1,500 so far)

• SPARK/Ada would have prevented this almost fatal error in a 2.5 billion dollar spacecraft.
Ariane 5 initial flight failure:

Good

Bad, 37 seconds later
Ariane 5 initial flight failure:
- Software reused from Ariane 4, written in Ada
- The greater horizontal acceleration caused a data conversion from a 64-bit floating point number to a 16-bit signed integer value to overflow and cause a hardware exception.
- “Efficiency” considerations had omitted range checks for this particular variable, though conversions of other variables in the code were protected.
- The exception halted the reference platforms, resulting in the destruction of the flight.
- Financial loss over $500,000,000.
- SPARK/Ada would have prevented this failure
Boeing 787 generator control computer:

- There are two generators for each of two engines, each with its own control computer programmed in Ada.
- The computer keeps count of power on time in centiseconds in a 32 bit register.
- Just after 8 months elapses, the register overflows.
- Each computer goes into “safe” mode shutting down its generator resulting in a complete power failure, causing loss of control of the aircraft.
- The FAA Airworthiness Directive says to shut off the power before 8 months as the solution.
- There is now a second 787 reset problem.
- SPARK/Ada would have prevented both.
Deep Space Application

6U CubeSat with ion thruster
Asteroid flyby mission
Busek Ion Thruster

BIT-3 Iodine Propellant

75W, 1.24 mN, 2.5 cm beam width, $I_{sp} = 2,640$

For a 6U, 14 kg spacecraft with 1.5 kg iodine:

$\Delta V = 2,900$ m/s
Flight Software based on CubedOS

- **Intended to be a general purpose framework for CubeSat flight software**
- Written in **SPARK**; proven free from runtime errors
- Provides inter-module message passing framework
- Provides services of interest to flight software
- Can integrate existing Ada or C runtime libraries
- Uses a Low Level Abstraction Layer (LLAL) to abstract OS/hardware
- **Conceptually similar to NASA’s cFE/CFS except written in SPARK (not C).**
Current Software Environment

- VxWorks 6.9 on PowerPC
- SPARK 2014 with Ravenscar runtime

Current Development Team

- VTC: 2 faculty, 5 students (2 MS, 3 BS)
- Grant applied for to increase to 4 faculty and 7 students (3 MS, 4 BS)
CubedOS Verification Goals

• No flow errors
• Show freedom from runtime error
• Other correctness properties as time allows

CubedOS Testing

• Unit tests with AUnit (x86)
• Some additional test programs (x86)
• Hardware development system (PowerPC)
• Hardware “FlatSat” (PowerPC)
Continuous Integration

• We use Jenkins-CI ([https://jenkins.io/](https://jenkins.io/))

• Every night...
  – ... builds & executes unit test programs
  – ... does SPARK flow analysis
  – ... does SPARK proofs

• Build considered to have failed if unit tests fail
  – Requiring successful proofs for “successful” build too high a bar
Software Architecture

• Collection of “modules” that pass messages
  – Each module reads messages from exactly one mailbox
  – Each module contains a message processing task
  – Modules all execute concurrently

• Collection of libraries
  – Passively called from multiple modules
Software Architecture

• CubedOS comes out-of-the-box with:
  – A set of standard server modules
    • Timing services
    • Publish/Subscribe services
    • File system interface
    • Communication protocols (e.g., CFDP)
    • ... etc
  – A set of library facilities
    • CRC, Packet encoding/decoding, data compression
Small Spacecraft Flight Software

• A CubedOS application
  – Application modules for:
    • Device drivers for subsystem hardware
    • Spacecraft state manager (“main” module that initiates and coordinates other activity)
    • Command scheduler
    • Implementation of CubedOS standard file system interface
Software Stack (Spacecraft Modules)

“Main” Module

State Manager
Storage Manager
Schedule

Control Modules
Spiral Thruster
Logger

CubedOS

Driver Modules
Iris
UHF
BIT3
ADACS
EPS
Instrument

Brandon & Chapin- NASA Academy of Aerospace Quality 2017
CubedOS Mailboxes

generic
    Module_Count : Positive;
    Mailbox_Size : Positive;
    Maximum_Message_Size : Positive;
package CubedOS.Generic_Message_Manager is
    type Message_Record is
        record
            Sender     : Module_ID_Type;
            Receiver   : Module_ID_Type;
            Message_ID : Message_ID_Type;
            Priority   : System.Priority;
            Size       : XDR_Size_Type;
            Payload    : XDR_Array;
        end record;
    type Message_Array is array(Message_Index_Type) of Message_Record;
protected type Mailbox is ... end Mailbox;
Mailboxes : array(Module_ID_Type) of Mailbox;
end CubedOS.Generic_Message_Manager;

Mostly for future expansion
XDR encoded message parameters
CubedOS Mailboxes

– Each instantiation of the message manager creates a “communication domain”
– Multiple communication domains possible
– Each module has unique ID within its domain
– Each module has a single task that reads its mailbox and handles/dispatches messages
– Message parameters are encoded/decoded at runtime into octet streams and installed into the receiver’s mailbox
CubedOS Modules

– Each module is a hierarchy of packages
  • Complex modules might have multiple private child packages to support implementation

– Some_Module.API
  • Contains subprograms for encoding/decoding messages
  • Generated automatically by the XDR2OS3 tool (under development) from a high level message specification

– Some_Module.Messages
  • Contains the message loop and message handling
CubedOS Modules

– Module communication is point-to-point
  • Sender names receiver explicitly
  • Receiver learns sender ID from message header
  • Replies returned via (dynamically specified) ID

– Server modules
  • Can be written without knowledge of clients
  • Provided by third party libraries
Advantages

– Lots of behavior deferred to runtime
  • Flexible and dynamic communication patterns
  • Easily extensible via module libraries
  • OOP-like behavior
    – Many different implementations of the same module API are possible; clients need not know which implementation they are using
Disadvantages

– *Lots of behavior deferred to runtime!*
  
  • Message encoding/decoding overhead (space and time)
  
  • Loss of type safety (compare with well-typed protected object entry calls)

– *Not the SPARK way!*
  
  • But... type safety issue mitigated somewhat by XDR2OS3
Modified XDR Message Specification

typedef unsigned int Channel_ID_Type range 1 .. 16;
typedef enum { Success, Failure } Status_Type;
constant Max_Data_Size = 1024;

message struct {
    Channel_ID_Type Channel;
} Subscribe_Request;

message struct {
    Channel_ID_Type Channel;
    Status_Type Status;
} Subscribe_Reply;

message struct {
    Channel_ID_Type Channel;
    opaque Data<Max_Data_Size>;
} Publish_Request;
package CubedOS.Publish_Subscribe.API is
    type Channel_ID_Type is range 1 .. 16
    type Status_Type is (Success, Failure);
    Max_Data_Size : constant := 1024;

    type Message_Type is
        (Subscribe_Request,
         Subscribe_Reply,
         Publish_Request);

function Subscribe_Request_Encode
    (Sender : Module_ID_Type;
     Channel : Channel_ID_Type;
     Priority : System.Priority) return Message_Record
with Global => null;
...
end CubedOS.Publish_Subscribe.API;
XDR2OS3 Specification Output

package CubedOS.Publish_Subscribe.API is
  ...
  procedure Subscribe_Request_Decode
    (Message : in Message_Record;
     Channel : out Channel_ID_Type;
     Status  : out Status_Type)
  with
    Global => null,
    Pre => Is_Subscribe_Request(Message),
    Depends => ((Channel, Status) => Message);

end CubedOS.Publish_Subscribe.API;
package body CubedOS.Publish_Subscribe.API is

function Subscribe_Request_Encode
  (Sender : Module_ID_Type;
   Channel : Channel_ID_Type;
   Priority : System.Priority) return Message_Record

is
  Message : Message_Record :=
    Make_Empty_Message
      (Sender, ID, Message_Type'Pos(Subscribe_Request), Priority);
  Position : XDR_Index_Type;
  Last : XDR_Index_Type;

begin
  Position := 0;
  XDR.Encode
    (XDR.XDRUnsigned(Channel), Message.Payload, Position, Last);
  Message.Size := Last + 1;
  return Message;
end Subscribe_Request_Encode;
Problem with Mailboxes

• SPARK won’t track information flow through arrays
  • “high: multiple tasks might queue on protected entry "message_manager.mailboxes.receive"”
• We suppress this message!
• Can’t track flow between modules
  • We must take responsibility for initialization, etc.
  • But... this allows flexible communication
• Full strength of SPARK within modules
• NOTE: Must ensure modules have unique IDs!
Why not NASA’s cFE/CFS?

• “cFE/CFS” = “Core Flight Executive / Core Flight System”

• Similar architecture
  – Uses publish/subscribe (not point-to-point)
  – Uses CCSDS space packets for messages

• cFE written in C. Not verified

• We hope to eventually offer CubedOS as a competing SPARK platform for spacecraft software
One of Our Ground Stations

The 70m Dish at Goldstone, California
A SPARK 2014 Book is Available

Building High Integrity Applications with SPARK

John W. McCormick
Peter C. Chapin
Acknowledgements

• NASA Vermont Space Grant Consortium

• Vermont Technical College

• AdaCore, Inc. (GNAT Pro, SPARK Pro)

• Applied Graphics, Inc. (STK)

• Busek (BIT-3 Iodine ion drive)

• NASA Jet Propulsion Lab (Iris-2 Radio)
CubedOS: A SPARK Message Passing Framework for CubeSat Flight Software

NASA Academy of Aerospace Quality 2017 - Copyright 2017 Carl Brandon & Peter Chapin

Dr. Carl Brandon & Dr. Peter Chapin   carl.brandon@vtc.edu  peter.chapin@vtc.edu
Vermont Technical College   +1-802-356-2822 (Brandon), +1-802-522-6763 (Chapin)