DriftSonde System Analysis

Charles Martin

Research Technology Facility

Atmospheric Technology Division

National Center for Atmospheric Research

Boulder, Colorado

ICARUS System Analysis

1 Introduction

This document records the results of the ICARUS system analysis. The goal is to produce a thorough explanation of the ICARUS system requirements. Having this information in print ensures that everybody is trying to build an identical system.

1.1 The analysis phase

We are starting at the very beginning. We have some general (and specific) ideas about what ICARUS is supposed to accomplish[1]. However, we need to rigorously define the system requirements, and devise a system architecture that will fill those requirements.

We can use a modified "Use Case" approach to meet these needs. Use cases are descriptions of small pieces of system behavior. The idea is to write down all of the gross actions of the system. We start out with a very broad brush, without any preconceptions of the system architecture. We begin simply by describing the behavior in terms of the interaction between system entities. We just choose entities that seem reasonable; names that represent concepts we know the system will have such as "ballast controller" or "sounding schedule".

As we write the use cases, we add most of the nouns to the keyword list. Two things will happen as we go along. The first is that we will find ideas that are shared among use cases, and often we will want to rename the object so that it best represents its shared usage. The second is that as we are going through this process, we will begin to understand the interactions among the entities. We will see which ones are shared, and get clues as to the logical grouping of objects that seem to belong together.

As we notice these divisions, we can assign a keyword to a package. Think of a package as a functional sub-system, but be aware that packages can have software and hardware parts to them, and may actually encompass multiple hardware components. For instance, the ballast package could consist of the ballast actuator, the physical ballast itself, a PIC processor which controls the actuator, the software within the PIC processor, and the software running on the controller computer which sends commands to the PIC ballast controller. The point is that at this stage we don't want to be wrapped up in the detail of the ballast package; we want to identify its behavior rather than its implementation.

We will iterate exhaustively through the use cases, making sure that all bases are covered. When we are finished with them, all parties must be able to agree that the system behavior has been completely defined. We will have identified all of the important components in the system, described how they interact, and assigned them to packages.

This will constitute the analysis phase of the project.

1.2 The design phase

After a good first cut at the analysis is complete, we can move into the design phase, where we translate the packages and their internal behaviors into more concrete designs. Whereas the analysis focused on the overall system behavior and requirements, the design phase focuses on the implementation.

The package structure leads us directly to the high level system architecture. The packages represent commonality of responsibility and behavior. The keywords give a first cut at the underlying objects within the system

We can then take each package and its associated behavior to specify requirements for each package. We

---

[1] The currently documented source of these ideas is in the ICARUS proposal.

may decide that a package should be logically split into sub-pacakges.

At this point, we can decide what represents a reasonable fit as far as implementation choices; e.g. how powerful a processor is needed for the system controller, and do we need a sophisticated controller for temperature control? In some cases, the implementation choices are already made, e.g. the telemetry receiver already exists and has a defined command set.

As you can see, this approach leads neatly to the design of the software. The packages are massaged into class libraries, and the keywords lead to classes contained within those libraries. However, it is not the responsibility of the analysis phase to define precise software classes. It is a more broad brush activity. Detailed class definitions will take place only in the design phase.

The output of the design will be the identification of physical sub-systems, either hardware, software, or both. For the software systems, class libraries, individual classes, methods and attributes will be identified. I'm not sure what will be defined for the hardware sub-systems.

1.3 Iteration

As we work through any of these steps, in both analysis and design, we will undoubtedly change our opinions on how things should be done. Therefore, we will go back to the relevant sections, rewriting and moving things around to fit the better ideas. It's important to be thorough, so that we don't introduce errors. By rigorously using a defined set of keywords, when we change one part of the analysis, we should be able to easily see what other areas are affected simply by tracking the common usage of the objects.

This iteration between the analysis and design steps is critical. As we look closely at the design of a package, we may discover that the Use Cases don't really describe the right way to do something, or describe a sequence that is impossible to follow due to some other constraint. At this point we had better figure out an alternative in the system behavior that lets us fulfill the required behavior within the constraint. The affected use cases would be modified, and we check to make sure that interactions between packages are not impacted. New keywords might need to be added, or the meanings changed for existing ones.

The point is that we will be switching back and forth between these two phases as we progress. Simply remember that in the analysis we are defining system behavior and requirements. In the design phase we are elaborating the system architecture and implementation.

1.4 A comment about "modified use cases"

We are learning about this process as we go along, and it seems that everyone else is also in the same boat. I've read several texts on this process, and I have to admit that they seem vague in many respects. For instance, they all talk about Use Cases as actions that are triggered by an "actor" external to the system. Well, in the case of ICARUS, the main actor-event pair occurs when the operator launches the balloon. After that, the system is mostly on its own. We would only have a couple use cases. The books also want the use cases to be mostly divorced from the realization of the system architecture. I've found it hard to separate the two aspects. When you are describing the behavior of the system, it is natural to consider the interaction of its components. Therefore, the approach I've taken here is my blending of what I've read with what feels more natural to me. For lack of (mostly) my experience, lets try it this way.

2 Nomenclature

References to use cases will be capitalized. References to keywords will be in italics.

3 Use Cases

3.1 ICARUS Load

Preconditions: the sondes have been adapted with the launch plates.

The user loads a sonde into each sonde slot. The corresponding launcher wire harness connector is attached to the launch plate.

3.2 ICARUS Running

The basic operational state, where the system is powered up and all subsystems are functional. ICARUS may be in the Running mode at any time; e.g. in the lab, during Prelaunch, during ascent, or during flight.

1. Power is turned on
2. All sub-systems are booted and running.

3.3 Mission initialization

A mission has a definite life span, starting at launch time. The notion here is to allow ICARUS to continue a mission even after a reboot or reset of any or all of the sub-systems. Thus some artifacts, such as the *Sounding Schedule*, must remain valid even after a **Hard Reset** or **Soft Reset**.

1. Set the *Sounding Schedule* to empty.
2.

3.4 Hard Reset

Power to all sub-systems is held off for 2 seconds, and then turned on.

3.5 Soft Reset

Preconditions: ICARUS Running

All sub-systems are asked to voluntarily reset.

1. For each sub-system, send a **Reset** command.
2. If any of the sub-systems did not acknowledge a successful reset, **Hard Reset**.
3. Send **Engineering Report**?

3.6 ICARUS Prelaunch

Preconditions: ICARUS load

1. The *ground station* is attached to the *controller* via the communications cable.
2. The *gondola* is powered on and is in the **Running State**.
3. The ground station *maintenance program* is run.
4. **Ground Diagnostics** are performed.
5. *Mission parameters* loaded.
6. (Optional) The *gondola* may be powered down.

3.7 ICARUS Ground Diagnostics

Preconditions: ICARUS Running

1. *Power system* check.
2. *Telemetry system* test.
3. *Meteorological system* test.
4. *Sonde* functional check.
5. *Ballast system* test.
6. *Clock system* test.

3.8 ICARUS Launch

Preconditions: ICARUS Prelaunch

1. *Ground system* ready and receiving.
2. *Ground system* connected to Internet, ready for remote command activity.
3. *Balloon* is released.

Note: We are assuming that the launch crew will want the ability to send remote commands to the *controller* as soon as launch, in case a cut-down or other control is needed. This is why the *ground system* is connected to the Internet.

3.9 State Data Access

The state data encompass all global data that are pertinent to the Driftsonde application. Some of this data may be fixed for the duration of a mission; other data will be modified periodically or on an irregular basis. Examples of this include the current GPS readings, or the flight level data. "Global" is key to the definition; the state data only includes information that must be shared between two or more consumers.

The state data will be placed in a well-defined location, so that it can be accessed by any other application. It will include a *timestamp*, which can be compared to the current system clock in order to determine the age of the data. The data format may be sub-system dependent, i.e. a uniform data format is not imposed on the state data. The user is responsible for decoding the data.

A mechanism will be provided to allow write locked access to the data. *(Perhaps using file locking?)*

A naming convention will assign keys to particular state data elements.

To access state data:

1. obtain lock for the given key
2. read (or write) the data
3. return lock

Obviously, the application must not hold the lock for long periods of time, and must be able to wait on a lock. For robustness, we should have a mechanism that allows for a timeout and failure on a lock request.

The state data may provide a semaphore type of functionality, allowing independent tasks to signal each other via the state data.

3.10 Subsystem Management

Each subsystem that requires interaction has a control task running on the system controller. This task is the only one with a direct communication link to the subsystem. The task is responsible for:

1. The behavioral control of the subsystem. Thus any primary algorithms that are concerned with real-time control of the subsystem reside in this task.
2. The transfer of information from the subsystem to the state data.

The control tasks will take their cues, if required, from the state data.

3.11 Perform Sounding

Preconditions: *ICARUS* at flight level

1. Select next *sonde*.
2. **Sonde Preflight** procedure
3. **Sonde Launch**.
4. While sounding active

Collect and store data
End

5. **Reduce Sounding**.

3.12 Sonde Preflight

Preconditions: System is in sonde preflight mode.

1. The *telemetry receiver* is set to the frequency of the sonde.
2. The *sonde coefficients* are loaded.
3. The *sonde* is powered up and allowed to stabilize for 5 minutes.
4. The data acquisition routine reads the data stream from the telemetry receiver over a certain time. If the received data passes a validity check, the *sonde* is available for launching.

3.13 Sonde Launch

Preconditions: **Sonde Preflight**

1. The data acquisition routine is switched to flight mode.
2. The gondola flight conditions are stored (PTH, GPS, internal temp and battery voltage).
3. The *launcher* is commanded to release the *sonde*.

3.14 Collect and Store Data

Precondition: **Sonde Launch**

The data acquisition routine reads the GPS and PTU data from the telemetry receiver. Conversion to final engineering units occurs as the data are being read. The data are saved as *raw data* .

The end of the sounding is signaled by the detection of at least one of three conditions:

1. No data is received for longer than 2 minutes. (Surface impact or sonde malfunction)
2. The reported pressure does not decrease by more than **X** mb over **Y** seconds (Surface impact – still transmitting, sonde malfunction)
3. The sounding lasts more than **Z** minutes. (Slow descent)

3.15 Reduce Sounding

The raw data will need to be reduced for *SatCom* transmission. The amount of reduction necessary will be mandated by the constraints of the *SatCom* bandwidth available (or affordable). The result of the reduction will be a *data product*. This may be a *decimated data* set, a set of *levels*, or a *WMO message*. There are pros and cons for each of these.

For any scenario however, the data will need to be quality controlled in order to remove erroneous data.

1. Apply Q/C procedures to the *raw data*, to create a *Q/C data* set.
2. Convert the *Q/C data* to the *data product*.

3.16 Transmit Sounding Report

A Data Product is broken into *SatCom messages*. The *SatCom messages* are transferred to the SatCom communications process, to be queued for transmission.

1. Get the *data product* for the *sounding ID*.
2. Convert to a *sounding report*.
3. Code it into *SatCom messages*.

4. Transfer the *SatCom messages* to the SatCom system.

3.17 Download High Resolution Data

1.

3.18 Set Sounding Schedule

The *sounding schedule* can be specified or changed at any time. A freshly initialized system will have a blank schedule. A rebooted system will retain the existing schedule.

3.19 Schedule Sounding

In normal operations, soundings will be launched at scheduled intervals.

1. Every minute: if a sounding is NOT active, check the *sounding schedule*.
2. If the *sounding schedule* calls for a sounding, mark the *schedule entry* as completed, and **Perform Sounding**.
3. Monitor Clock for schedule.

3.20 Remote Schedule Change

3.21 Gondola Cut Down

The *gondola* can be commanded to disconnect itself from the *balloon* and parachute to the surface. The system will remain running during the descent, and will periodically **send engineering reports** in order to track it's descent. It can be also be commanded to **send engineering reports** in order to find it on the ground.

Once on the ground, it will **send engineering reports** less frequently, until it is recovered, or the batteries fail.

1. Activate *cutdown* mechanism.
2. While pressure is decreasing, **send an engineering report** every ten minutes.
3. After pressure becomes constant, **send an engineering report** every 6 hours.

3.22 Send Engineering Report

This is a compact but complete report containing information that documents the health of the system. The message must be kept compact since it is sent frequently and must make efficient use of the satellite. It would be nice to send a key along with each data field so that decoding will not need to be changed with software revisions. Perhaps start each field with a letter identifier, and end with a semi-colon (except for last field). A numeric message sequence number begins the report.

1. Retrieve system state from the global data base, and format fields:

A	GMT date and time	ddmmyyhhmmss	
B	Valid GPS indicator	A(=valid) or V(=invalid)	
C	Position	LLLL.LLLLH,LLLLL.LLLLH	Latitude, hemisphere (N or S), longitude, hemisphere (E or W)
D	Uptime	TTT.TTTU	Time followed by units (s, m, h, or d)
E	Processes	N	Number of processes

F	Disk space used	R,T,V	/, /tmp, /var
G	GPS velocity	U,V, W	
H	Internal temperature		
I	Ambient Pressure		
J	Ambient Temperature		
K	Ambient RH		
L	Battery voltage1		
M	Battery voltage2		
N	Gas temperature		
O			
P			
Q			

2. Transmit *satcom message*.
3. Send distress signal if gondola prematurely falls prior to timer cutdown or commanded cutdown.

3.23 Control Altitude

Preconditions: **ICARUS Launch**

Need altitude control algorithm. Perhaps there is some literature on this, or information from GSSL? Some relevant factors:

Is ICARUS in the initial ascent?

The altitude is a function of gas temperature and the system weight.

The gas temperature is a function of the solar insolation and the ambient temperature.

The system weight is a function of the number of sondes dropped, the ballast weight, the fixed mass (gondola, balloon, and other), and the gas weight.

When should altitude be actively adjusted?

3.24 Control Internal Temperature

Preconditions:

1. If the *internal temperature* drops below the *internal temperature low set point*, turn the *heater* on.
2. If the *internal temperature* goes above the *internal temperature high set point*, turn the *heater* off.

3.25 Gondola Power Management

1. Control (on/off) of DC power to each module, based upon needs of functionality of task required. This is to conserve battery power.

3.26 Remote Command

Preconditions: **ICARUS Running**

A small set of commands is available to be sent to ICARUS in order to initiate actions on the *controller*. These commands are transmitted via the *satcom* link.

1. Wait for incoming message from the *satcom system.*
2. For each message type:
   - 2.1. *Hard Reset*: **Hard Reset**
   - 2.2. *Soft Reset*: **Soft Reset**
   - 2.3. *Cutdown*: **Gondola Cutdown**
   - 2.4. *Sounding*: **Perform Sounding**
   - 2.5. *Sounding Schedule*: **Set Sounding Schedule**
   - 2.6. *Report Schedule*: Set the *engineering report interval.*
   - 2.7. *Report*: **Send Engineering Report**
   - 2.8. *Download*: **Download High Resolution Data**
   - 2.9. *Retransmit*: **Transmit Sounding Report** for the requested *sounding ID*. (Should we be able to ask ICARUS to retransmit the Sounding Data Product for a given sounding ID? This is only needed if we did not receive a particular sounding. Need for this depends on the reliability of the SatCom link.)
   - 2.10. *Reset cutdown timer*: Reset the independent cut time timer for the gondola package.
   - 2.11. *Ballast Control*: Release ballast on command.
3. Send acknowledgement *satcom message* back to *earth station*. The acknowledgement should contain status information specific to the command. Perhaps an acknowledgement is not needed for commands that send an immediate response, such as *Report*. All messages should have a time tag for when the messages were received and time the messages were sent back.

3.27 Code WMO Message

1. Create *levels* from the *Q/C data*.
2. Create *WMO message* from the *levels*.

3.28 Transmit Satcom Message

3.29 Receive Satcom Message

Is some form of authentication required? Perhaps Orbcomm access control is good enough. Are arbitrary users prevented from ending email to our OrbComm untis?

3.30 Forward WMO message to GTS

3.31 Direct Telemetry Link to ground

1. Send Engineering reports via direct RF link to ground.
2. Send temp drop messages via direct RF link to ground.

4 Package Physical Interfaces

4.1 Ballast

The ballast dump valve is energized by a solenoid.

1. TTL line to relay

4.2 Controller

4.3 Earth Station

Satcom messages are sent and received via email. GTS messages are lodged via FTP.

1. Ethernet to the Internet

4.4 Engineering

4.5 Gondola

4.6 Ground Station

Gondola checkout, loading and initialization.

1. Ethernet directly to controller, or via network.
2. USB could be an alternative.
3. RS-232 could be an alternative.

4.7 Heater

Heater must be powered on and off.

1. Controller receives commands via rs-232. (If there is a separate controller)
2. Heater relay (transistor?) is activated by TTL signal.

4.8 Met

Ambient and internal variables are measured.

1. Standard Vaisala PTH module produces 3 frequencies? To be measured by

4.9 Power

4.10 Satcom

4.11 Telemetry

4.12 Launcher

5 Keywords

Keywords represent identifiable entities. The entity has a primary owner, or package, with which it is associated. Sometimes the entities are shared between systems, and so co-package can be identified. Co-packeges are only noted for resources that are shared. For instance, even though the controller will be issuing commands to the *ballast controller*, it does not deal directly with *ballast*, and so controller is not shown as a co-system for *ballast*.

package	co-package	keyword	description
ballast			The subsystem responsible for managing ballast releases.
ballast		*ballast*	The mass that can be dropped from the gondola in metered amounts. The smallest discrete amount required will depend on the altitude control algorithm.
ballast		*ballast controller*	The device that accepts commands to drop a specific amount of ballast.
controller			The subsystem that performs high level control and communication functions for ICARUS.
controller		*ascent phase*	The period between the *gondola* launch and when it first reaches the *mission altitude*.
controller	sonde	*coefficients*	The *sonde* sensor coefficients. They are stored in the *sonde*, and must be read from the *sonde* via the umbilical cable.
controller	many	*state data*	Global data that is used by two or more consumers. This is outside of the normal Unix system artifacts; i.e. it is specifically related to the Driftsonde application.
controller		*data product*	The final reduced data product for a single *sounding*. it is currently undefined, and in the end it may become either *decimated data*, *levels* or *wmo message*.
controller		*data stream*	The serial data stream contain all information sent by the *sonde* via the *sonde* telemetry link.
controller		*decimated data*	A compressed form of the *q/c data*, where a certain percentage of the data have been removed simply to reduce the volume of transmitted data. It is vying to ultimately become

package	co-package	keyword	description
			the *data product*.
controller	earth station	*engineering report*	A report containing system health and diagnostic information. It needs to be small enough to afford to send frequently via the *satcom*.
controller		*levels*	Levels extracted from *q/c data*, according to fmh-3. It is vying to ultimately become the *data product*.
controller		*mission*	One deployment of ICARUS, such as a single flight of 6 days, a simulation test in the lab, or a system check out (without launch).
controller		*mission parameters*	The parameters that apply to a given *mission*, such as the nominal *flight altitude*, *engineering report* interval, etc. The *sounding schedule* will be part of the *mission parameters*.
controller		*q/c data*	The data resulting from application of quality control algorithms to the *raw data*. The *q/c data* is still at high resolution.
controller		*raw data*	The time series of data received from a single *sonde* drop. The data have been converted into to engineering units.
controller		*schedule*	The schedule for automatic launch of *sondes* during a *mission*.
controller		*sounding*	A container for all artifacts of a single sounding; i.e. the *raw data*, the *q/c data* and the *data product*. It's not clear that we actually need to identify this as an individual object.
controller		*sounding id*	An identifier that is unique for any *sounding*, from any ICARUS *mission*. The *sounding id* will unequivocally identify any ICARUS *sounding*. We might as well make this system wide, from the very beginning. It will make the data management easier in the long run. It suggests that each ICARUS *gondola* should have its own serial number.
controller		*sounding report*	A pairing of a data product and metadata. For instance, the metadata might contain performance indicators, such as telemetry recovery rates, Q/C error detection rates, etc.

package	co-package	keyword	description
controller		*system computer*	The main control computer system. It can communicate with the *ground station* via a direct cable. It can communicate with the *earth station* via the *satcom* link.
controller		*wmo message*	*Levels* coded according to WMO 206. It is vying to ultimately become the *data product*.
controller		*Timestamp*	A time record, formatted as the Unix time stamp, in UTC.
earth station		*earth station*	A headquarters or field based ground station which tracks and manages ICARUS missions. At the least, it must have an Internet link, in order to send and receive *satcom* messages. It will have the responsibility of forwarding *WMO messages* to the GTS. Note the distinction between the *ground station* and the *earth station*. These items need better names.
engineering			The subsystem that provides high time resolution diagnostic data. It will probably be a simple RF downlink, which would be active during the ascent phase and early in the traverse phase. Could it be used also for high-resolution data download?
engineering		*engineering transmitter*	Transmitter for sending high resolution engineering data over short distances.
gondola			The combination of the electronics housing and the sonde launcher.
gondola		*cut down device*	The device that will separate the balloon and the *gondola*.
ground			
ground		*ground station*	The computer system that is used for ICARUS initialization, maintenance and launch functions. The implication is that it is able to physically connect to the gondola, and so accompanies ICARUS to the launch site. It may however be useful to provide Internet communications between the *gondola* and the *ground station*. Note the distinction between the *ground station* and the *earth station*. These items need better names.

package	co-package	keyword	description
ground		*maintenance program*	The software used on the *ground station*.
heater			The subsystem responsible for controlling the temperature of the *gondola* electronics compartment
heater		*internal heater controller*	Device that performs the temperature control. May or may not be “smart”; i.e. able to receive commands and issue status.
heater		*internal temperature*	The temperature of the *gondola* electronics compartment.
heater		*internal temperature high set point*	When the internal temperature gets above this value, turn off the heater.
heater		*internal temperature low set point*	When the internal temperature gets below this value, turn on the heater.
launcher			The subsystem that manages sonde dispensing.
launcher		*launch plate*	The small plate that a sonde is attached to in the launcher, and which is activated to launch the sonde.
launcher		*launcher*	The device which can be commanded to drop a single sonde.
launcher		*wire harness*	A wire set that electrically connects the sonde to the rest of the system.
met			The subsystem responsible for measuring and reporting the physical state.
met		*ambient air temperature*	The current ambient air temperture.
met		*ambient humidity*	The current ambient humidity.

package	co-package	keyword	description
met		*ambient pressure*	The current ambient pressure.
met		*ambient radiation*	The current ambient shortwave radiation.
met		*flight pressure*	The desired mission pressure altitude.
met		*gps altitude*	The GPS reported altitude.
met		*gps position*	The GPS reported position.
met		*solar cell*	The *ambient radiation* sensor.
miscellaneous			A catchall package, used to hold items which do not need to be assigned to specific subsystems.
miscellaneous		*balloon*	The balloon. Does not include the balloon volume adjustment control.
miscellaneous		*sonde*	The reason ICARUS exists
power			The subsystem that supplies power for heating and electronics within the *gondola*.
power		*battery pack*	The combined power source.
power		*battery voltage*	The current voltage of the *battery pack*.
satcom			The subsystem responsible for satellite communications from the *gondola* to the *earth station*.
satcom		*orbcom transceiver*	The device that manages the RF side of the satcom communications. It is intelligent in that very little external control is required; i.e give it a message and the rest is handled here. It also can report status information.
satcom		*satcom message*	A single message to be sent to the earth station. A *satcom message* is the smallest unit that can be individually managed by the *orbcom transceiver*. Higher level protocols, needing

package	co-package	keyword	description
			to send larger data objects, would be built on top of *satcom messages*.
telemetry			The subsystem responsible for RF interface to a single *sonde* and for delivering the *data stream* extracted from the *sonde* RF link.
telemetry		*gps-ptu demodulator*	The device that delivers the *data stream* that contains GPS and PTU data from the *sonde*.
telemetry		*telemetry receiver*	The RF receiver that feeds the *gps-ptu demodulator*.
watchdog		*wake-up timer*	A timer that “wakes up” another subsystem at a predetermined time.