Software crítico para sistemas espaciales

2024/25

Master Universitario en Enxeñaría Aeroespacial

Arno Formella
Miguel Ramón González Castro
Manuel Pérez Cota

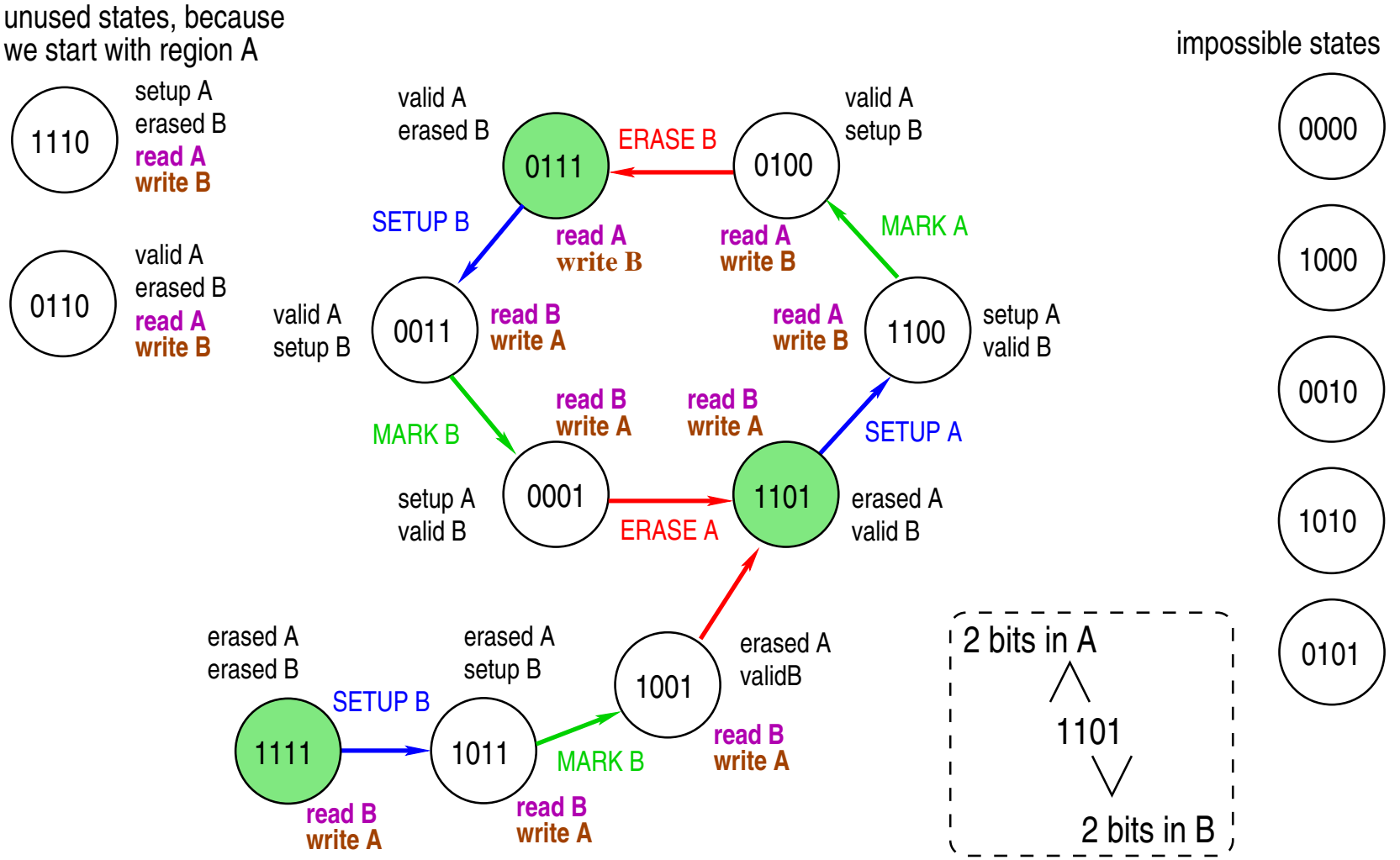
Departamento de Informática
Universidade de Vigo

24/25

doble búfer para estado del sistema

- satélite se apaga siempre que sea posible
- sino, se apaga por el perro de guardia (*watchdog*)
- pueden existir apagones no esperados
- por eso:
  - el estado del sistema se mantiene en dos búferes
  - uno para constantes modificables desde tierra
  - uno para variables del estado
  - los dos búferes se cargan y se almacenan de forma alterna
  - una máquina de estados de 4 bits (bytes seguros en realidad) controla el proceso
- la máquina de estados puede vivir con una flash borrada

máquina de estados del doble búfer



planificadores de tares

- planificadores

  tareas vitales: comprobación de eclipse, resetear perro de guardia, apagado periódico

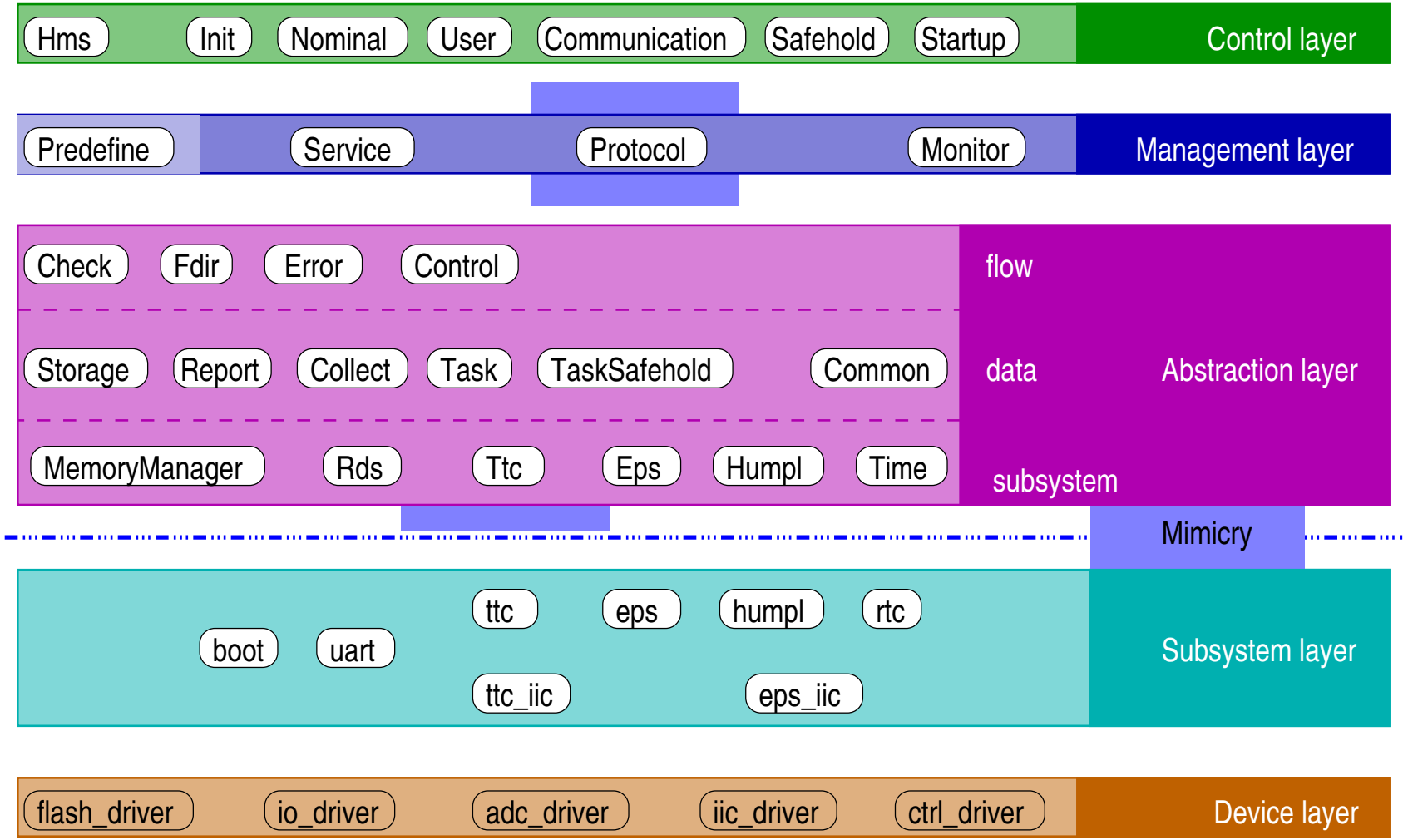
  tareas nominales: pedidos de datos por servicio, operaciones periódicas de cargas útiles, colección (y monitorización) de datos de gestión interna

  tareas seguras: alternancia entre fase de descubrimiento y de comunicación más colección (y monitorización) de datos de diagnosis

- cola de prioridad de tamaño fijo con búsqueda lineal
- cola de prioridad como lista ordenada embebido en un array

Software design

layered structure



entorno de simulación: *gemelo digital*

en aquellos tiempos sin saberlo, se implementó un *gemelo digital* del satélite (por lo menos a capa de lógica)

consiste de tres programas:

- HDS: HumSAT servicio de depuración (*debug service*)
- HMS: HumSAT software de misión (*mission software*)
- HGS: HumSAT simulación terreno (*ground simulation*)

HDS como parte del entorno de simulación

HumSAT servicio de depuración

- emula la conección via UART mediante sockets TCP
- solamente salida implementado

HMS como parte del entorno de simulación

HumSAT software de misión

- código idéntico al código de vuelo
  (con excepciones como el apagado)
- mimetismo lógico del comportamiento de los subsistemas incluidos son: FLASH, EPS, TTC, RTC, HUMPL, UART, watchdog, powerdown
- conectable al servidor host (con o sin protocolo de comunicación) basado en protocolo UDP
- se puede usar para comprobar estación terrena y entrenar operaciones

HMS como parte del entorno de simulación

HumSAT simulación de estación terrena

- emula o bien RX o bien TX de una estación terrena via sockets UDP
- estación terrena TX con interfaz de línea de comando para mandar telecomandos
- estación terrena RX con visualización de informes y señal de faro (*beacon*)
- simulación como en vuelo (pedidos/informes, TC/TM)

entorno de simulación: *gemelo digital* ejemplos

```
HumSAT ground simulation software

hgs: HumSAT epoch Sun Jan  1 00:00:00 2012
hgs: epoch offset 52796523.000000 seconds
hgs: starting humsat ground simulation software
hgs: sending to hms at localhost
hgs: sizeof xtc_frame_t: 40

hgs: use ?? for help

hgs>
```

entorno de simulación: *gemelo digital* ejemplos

```
hgs> ??

  ad(d)          an(tenna)      ch(eck)          cl(ear)
  co(py)         cr(itical)     def(ine)         del(ete)
  du(mp)         ec(lipse)      ep(s)            er(ase)
  ev(ent)        g(et)          he(ater)         hu(mpl)
  i(nterval)     l(oad)         mod(e)           mon(itor)
  n(ame)         pa(rameter)    pi(ng)           po(werdown)
  pr(otocol)     ra(te)         rd(s)            rep(ort)
  rese(t)        rest(ore)      se(t)            shi(ft)
  sho(w)         ta(sk)         te(lemetry)      ti(me)
  tr(ansmit)     tt(c)          u(nknown)        w(rite)
```

entorno de simulación: *gemelo digital* ejemplos

```
hgs: use  ?command  for more help on command

    !q(uit)                  quit

    !c(lear)                 clear history
    !e(xecute) filename      execute script/history file
    !r(ead) filename         read history file
    !s(et) wait_time         set script wait time
    !w(rite) filename        write history file

hgs>
```

entorno de simulación: *gemelo digital* ejemplos

```
hgs> ?mon
use: mon(itor) c(lear)|l(ist)|r(eport)
     mon(itor) d(elete) <id>:1

hgs> ?pa
use: pa(rameter) <id>:1 <interval>:2
                 <rep_limit>:1 <rep_delta>:1
       [ l(imit) <low>:2 <rid>:1 <high>:2 <rid>:1 ]
       [ d(elta) <low>:2 <rid>:1 <high>:2 <rid>:1 ]
       [ e(xpected) <value>:2 <rid>:1 ]

hgs> ?ad
use: ad(d) <when>:4 r(elative)|a(bsolute) command
```

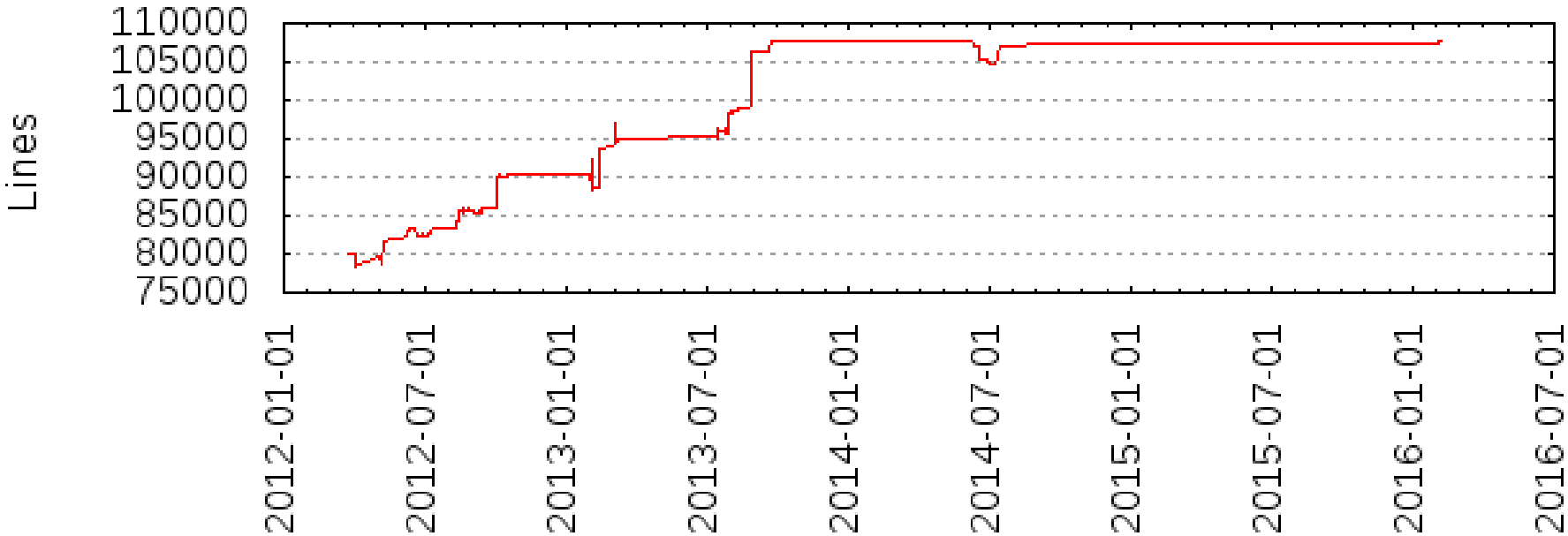
gestión del software

- control de versiones con git
- documentación embebida con doxygen y LaTeX

queda por hacer:

- versiones para 32 y 64 bit
- versiones para orden de bytes de red o nativo en las comunicaciones

líneas de código



recomendaciones desde XatCobeo

- código de asignación única (*single assignment code*)
- comprobación exhaustiva de pre- y pos-condiciones
- documentación del estado del código
  (*note*, *todo*, *warn* etc. con disciplina estricta)

recomendaciones desde XatCobeo

- separa tareas
  - pruebas por otro equipo
  - integración por otro equipo
  - comprobación de calidad por otro equipo
- uso de estándares

recomendaciones desde XatCobeo

- procesos son importantes
- rutinas son importantes
- disciplina es importante
- automatización es importante
- experiencia es importante

recomendaciones desde XatCobeo

- verificación de la documentación es parte imprecindible del desarrollo
  - un lemma mio:
    Si la documentación no coincide con el código,
    ¡ambos son incorrectos!
- especificación formal del formato de bitácoras para su procesamiento automática

ingeniería con vista a una posible malfunción

- un usuario nunca entra algo incorrecto,
  hay que esperar cualquier entrada
- en caso de defecto hay que actuar como consecuencia al defecto no de cualquier manera
- si un usuario no puede realizar cierto paso, sugierele una alternativa segura
- piensa sobre el compromiso por realizar: ir adelante con el fallo o terminar el programa rápido (especialmente en sistemas complejos)
- usa siempre que sea posible una bitácora de datos para una recuperación después de una fallo

ingeniería con vista a una posible malfunción

- comprueba el estado del sistema con frecuencia para detectar posibles fallos aprovechando de información redundante en el sistema (paridad, *checksums*, marcas, cabeceras, autoidentificación etc.)
- piensa sobre aquello que puede fallar y qué se puede hacer a nivel software
- todos los posibles errores deben tener su propio nombre único
- tiene que ser muy claro si una función pasa errores detectados a otra función

ingeniería con vista a una posible malfunción

- comenta y sé preciso si se puede tolerar cierta inconsistencia dentro del sistema especialmente durante qué tiempo y hasta qué extensión
- distingue entre errores que se han anticipado y errores no-esperados
- mensajes de error deben ser legible para quien los lee y ayudar en croncretizar el problema posiblemente con una recomendación de ayuda
- la actuación en caso de fallo debe cumplir con las especificaciones del sistema (por ejemplo cumplir con plazos de tiempo de ejecución (*worst case deadlines*), y no llevar a cabo acciones no permitidas etc.)

ingeniería con el foco en el error

- pensar: ¿qué puede fallar?
- pensar: ¿cuáles son las probabilidades?
- pensar: ¿cuáles son posibles remedios?
- pensar: ¿cuáles son posibles consecuencias?

revisión y trazada

- revisiones frecuentes y a todos los niveles
- implementación de pruebas automáticas y su ejecución frecuente
- precisión (trazable) en requisitos y especificaciones
- incluir datos, configuraciones, despliegue y operación en proceso de verificación y validación
- estas acciones deben estar embebidos en los planes del desarrollo