Third-Level Decomposition: Extended Computer Module

1. data type module
2. data structure module
3. input/output module
4. computer state module
5. parallelism control module
6. sequence control module
7. diagnostics module (R)
8. virtual memory module (H)
9. interupt handler module (H)

Data Type Module

- implements variables and operators for real numbers, time periods, and bit strings
- primary secrets: data representations and data manipulation instructions built into the computer hardware
- secondary secrets:
  - how range and resolution requirements are used to determine representation
  - procedures for performing numeric operations
  - procedures for performing bitstring operations
  - how to compute the memory location of an array index given the array name and the element index

Computer State Module

- keeps track of current state of extended computer (operating / off / failed)
- signals relevant state changes to user programs
  - after extended computer is initialized,
    signals the event that starts initialization of the rest of the software
- primary secret: the way that the hardware detects and causes state changes

Diagnostics Module (R)

- provides diagnostic programs to test
  - the interrupt hardware
  - the I/O hardware
  - the memory
- use is restricted
  because it reveals secrets of the extended computer

Virtual Memory Module (H)

- presents a uniformly addressable virtual memory for use by
  - data type module
  - input/output module
  - sequence control module
- allows using virtual addresses for data and subprograms
- primary secrets:
  - hardware addressing methods for data and instructions in real memory
  - differences in the way that different areas of memory are addressed

- secondary secrets:
  - policy for allocating real memory to virtual addresses
  - programs that translate from virtual address references to real instruction sequences

Third-Level Decomposition: Device Interface Module

1. air data computer
   - how to read barometric altitude, true airspeed, and Mach number
2. angle of attack sensor
   - how to read angle of attack
3. audible signal device
4. computer fail device
5. Doppler radar set
6. flight information displays
7. forward looking radar
8. head-up display (HUD)

9. inertial measurement set (IMS/IMU)
10. panel
11. projected map display set (PMDS)
12. radar altimeter
13. shipboard inertial navigation system (SINS)
14. slew control
15. switch bank
16. TACAN
17. visual indicators
18. waypoint information system
19. weapon characteristics

20. weapon release system
    - how to ascertain weapon release actions the pilot has requested
21. weight on gear

- almost corresponds to hardware structure
    - exceptions are closely linked devices

Third-Level Decomposition: Function Driver Module

1. air data computer functions
2. audible signal functions
3. computer fail signal functions
4. Doppler radar functions
5. flight information display functions
6. forward looking radar functions
7. head-up display (HUD) functions
8. inertial measurement set (IMS/IMU) functions
9. panel functions
10. projected map display set (PMDS) functions

11. ships inertial navigation system (SINS) functions
12. visual indicator functions
13. weapon release functions
14. ground test functions

- input-only modules are missing here:
  - angle of attack sensor
  - radar altimeter
  - . . .
- each module can be divided further

Head-Up Display Functions

- primary secrets:
  - where the movable HUD symbols should be placed
  - whether a HUD symbol should be on, off, or blinking
  - what information should be displayed on the fixed-position displays

Inertial Measurement Set Functions

- primary secrets:
  - rules determining the scale to be used for the IMS velocity measurements
  - when to initialize the velocity measurements
  - how much to rotate the IMS for alignment

Panel Functions

- primary secrets:
  - what information should be displayed on panel window
  - when the enter light should be turned on

Third-Level Decomposition: Shared Services Module

1. mode determination module
2. stage director module
3. shared subroutine module
4. system value module
5. panel I/O support module
6. diagnostic I/O support module
7. event tailoring module

Mode Determination Module

- determines system modes
  (as defined in the requirements document)
- signals the occurence of mode transitions
- makes the identity of the current modes available
- primary secrets:
  the mode transition tables in the requirements document

System Value Module

- has a set of sub-modules
- each sub-module computes a set of values,

  some of which are used by more than one function driver
- primary secrets: the rules in the requirements that define the value that it computes
  - selection among several alternative sources
  - applying filters to values produced by other modules
  - imposing limits on a value calculated elsewhere

Third-Level Decomposition: Application Data Type Module

- examples:
  - angles (several versions)
  - distances
  - temperatures
  - local data types for device modules
  - STE (state transition event) variables

Third-Level Decomposition: Physical Model Module

1. earth model module
2. aircraft motion module
3. spatial relations module
4. human factors module
5. weapon behaviour module
6. target behaviour module
7. filter behaviour module

Earth Model Module

- primary secrets: models of the earth and its atmosphere
  - local gravity
  - curvature of the earth
  - pressure at sea level
  - magnetic variation
  - local terrain
  - rotation of the earth
  - coriolis force
  - atmospheric density

Aircraft Motion Module

- primary secrets: models of the aircraft's motion
- used to calculate aircraft position, velocity, attitude from observable inputs

Spatial Relations Module

- primary secrets: models of three-dimensional space
- used to perform coordinate transformations, angle calculations, distance calculations

Human Factors Module

- primary secrets: models of pilot reaction time and perception of simulated continuous motion
- determines the update frequency for symbols on a display

Weapon Behaviour Module

- primary secrets: models used to predict weapon behaviour after release

Third-Level Decomposition: Data Banker Module

- one for each real-time data item
- value always up-to-date
- secret: when to compute up-to-date value

Third-Level Decomposition: System Generation Module

- . . .
  - (these programs do not run on on-board computer)

Third-Level Decomposition: Software Utility Module

- resource monitor module
- other shared resources
  - square root
  - logarithm
  - ...

Results of the A-7E Module Guide

- module guide is $<$ 30 pages
  - every project member must and can read it
- experience:
  - important to organize the guide by secrets, not by interfaces or by roles
  - software requirements document was essential for disambiguating choices in the guide's structure

- implementation of several subsets on a flight simulator
- integration testing of the first "minimal useful subset":
  - took a week only
  - nine bugs found
    - each in a single module only
    - each quickly fixed

  Dave Weiss: "like a breeze!"

- guide often used as a *document template* for other projects applying the method

3.3 Hierarchical Software Structures

Text for Chapter 3.3

[Par74] Parnas, D. *On a 'buzzword': Hierarchical structure*. In "IFIP Congress 74", pp. 336–339. North-Holland (1974). Reprinted in [HoWe01].

[HoWe01] Hoffman, D. M. and Weiss, D. M., editors. *Software Fundamentals – Collected Papers by David L. Parnas*. Addison-Wesley (Mar. 2001).

Additional Background for Chapter 3.3

[Cou85] Courtois, P.-J. *On time an space decomposition of complex structures*. Commun. ACM **28**(6), 590–603 (June 1985).

"Courtois hierarchy" of structures which are complex in time and space.

Structure

- partial description of a system, showing
  - a division into *parts*
  - a *relation* between the parts

- graphs can describe a structure

Hierarchical Structure

- a structure with no loops in its relation's graph:
  - $P_0 = \{\alpha \in P \mid \neg\exists\, \beta \in P \,.\, R(\alpha, \beta)\}$
  - $P_i = \{\alpha \in P \mid \exists\, \beta \in P_{i-1} \,.\, R(\alpha, \beta) \wedge$
    $\neg\exists\, j \in \mathbb{N},\ \gamma \in P_j \,.\, R(\alpha, \gamma)\ \wedge\ j \geq i\}$

- note: hierarchy $\neq$ tree
- meaning of "hierarchical structure"?
  - meaning of parts?
  - meaning of relation?

Different Kinds of Software Hierarchies

- module decomposition hierarchy
- calls hierarchy
- uses hierarchy
- Courtois hierarchy

- gives-work-to hierarchy
- created hierarchy
- resource allocation hierarchy
- can-be-accessed-by hierarchy

Module Decomposition Hierarchy

- kind of structure:
  - parts: write-time modules
  - relation: part-of
- time: early design time
- this structure is always a hierarchy
  - never loop in “part-of”

Calls Hierarchy

- kind of structure:
  - parts: programs
  - relation: calls
- time: design time
- hierarchical relation forbids recursion
  - usually not a useful hierarchy

Uses Hierarchy

- kind of structure:
  - parts: programs
  - relation: uses (i.e., requires-the-presence-of)
- time: design time
- definition of "uses":

  Given a program A with specification S and a program B,
  A uses B iff
  A cannot satisfy S unless B is present and functioning correctly

- example: list insert routine
  - uses getNextElem, setNextElem routines
  - calls nullPointerException routine
  - does *not* “use” nullPointerException routine
- example: window manager with call-backs
  - application passes address of draw() program to window manager
  - application responsible for drawing sub-area when draw() called
  - window manager calls draw()
  - window manager does *not* “use” draw()
- example: layers of communication services
  - the higher layer uses the services of the lower layer
  - messages are passed in both directions
    (reqest, indication, response, confirm)

- if a structure is a uses hierarchy:
  levels define virtual machines
- useful for “ease of subsetting” (see later)

Courtois Hierarchy

- kind of structure:
  - parts: operations
  - relation: takes more time and occurs less frequently than
- time: run time

Courtois: Decomposition of Complex Structures

- domains with complex structures:
  - physics
  - social science
  - economy
  - computer science
- sometimes easily decomposable in time and space
  - concentrations in chemical reactions
    - ▷ differential equation suitable
    - ▷ large number of molecules allows to assume continuum

- hierarchical decomposition difficult when
  - time or size scales are not far apart
  - interesting behavioural properties are related to rare events caused by weak interactions within the system
  - events at many scales of time or size from each other nevertheless have a non-negligible influence on each other
- a hierarchical decomposition should ideally have:
  - *time and size scales far apart between levels*
  - . . .
- (Courtois describes how one can model structures even when they are not easily decomposable)

Some More Kinds of Software Hierarchies

- module decomposition hierarchy
- calls hierarchy
- uses hierarchy
- Courtois hierarchy

some more kinds:

- gives-work-to hierarchy
- created hierarchy
- resource allocation hierarchy
- can-be-accessed-by hierarchy

Gives-Work-To Hierarchy

- kind of structure:
  - parts: processes
  - relation: gives an assignment to
- time: run time
- found in T.H.E. operating system
  - organized as set of parallel sequential processes
  - processes exchange work assignments and information by message passing
  - processes are in hierarchical gives-work-to relation
- useful for guaranteeing termination, but neither necessary nor sufficient for this

Created Hierarchy

- kind of structure:
  - parts: processes
  - relation: created
- time: run time
- must be a hierarchy (parent is older than child)
- is a tree
  - why? (team work in creating progeny is accepted practice)
- sometimes implies unnecessary restrictions
  - example: parent cannot die until all progeny die

Resource Allocation Hierarchy

- kind of structure:
  - parts: processes
  - relation: allocate-a-resource-to or
owns-the-resources-of
- time: run time
- applicable with dynamic resource administration only
- "allocate to" vs. "controls": the question of pre-emption
- example: hierarchical money budgets for country, state, university, department, . . .

- advantages:
  - interference reduced or eliminated
  - deadlock possibilities reduced
- disadvantages:
  - poor utilization when load unbalanced
  - high overhead when resources are tight (especially with many levels)

Can-Be-Accessed-By Hierarchy

- kind of structure:
  - parts: programs
  - relation: can-be-accessed-by
- time: design time
- important to security and reliability
- example: the "rings" of Multics
  - generalization of supervisor/user level of CPU execution
  - is even complete ordering
- a hierarchy prevents some useful accessability patterns

Many Kinds of Software Hierarchies Possible

- not all of these relations must form a hierarchy!
- you may choose some of these relations
  to form a hierarchy
- *if you confuse these relations,*
  you will mess up your design
  - you then force a hierarchy on a relation
    that should not be a hierarchy
    - ▷ T.H.E.: uses hierarchy and gives-work-to hierarchy coincided
    - ▷ write-time module hierarchy and uses hierarchy
      of course should not coincide

- ▷ write-time module hierarchy and created hierarchy should not coincide if the latter imposes constraints (object creation in OO!)

Example: ISO OSI Basic Reference Model

- basic reference model for communication systems
  - 7 layers
- is a uses hierarchy
- should not be implemented as a gives-work-to hierarchy
  - then lots of message passing between layers
  - much too inefficient

Uses Hierarchy and Courtois Hierarchy

- in practice they usually coincide
  - programs that require few or no other programs to function run short and are executed often
  - programs that run long and only a few times require many other programs to function
- except: the handling of exceptions
  - interrupts
  - reboot (seldom, needed by all programs)
  - . . .
- *if the above is not the case then usually something is wrong!*