

Software Architecture

Lecture 8 Documenting Software Architecture

João Pedro Sousa

SWE 443, Spring 2008
George Mason University

today's outline

- principles of documentation
 - being precise
 - examples
- documenting interfaces and behavior
- combining views
- going beyond diagrams

Acknowledgment

much of the material presented in this course is adapted from 17655,
taught to the MSE at CMU by David Garlan and Tony Lattanze

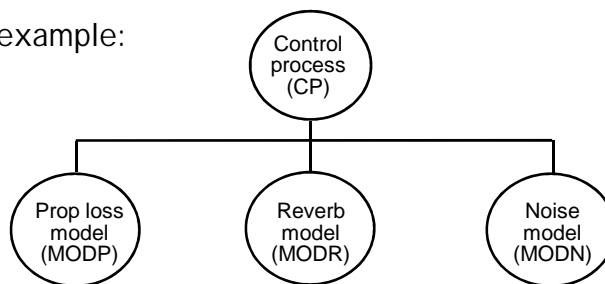
principles that apply to all documentation

1. write from the point of view of the reader
2. avoid unnecessary repetition
3. avoid ambiguity
examples in a bit
4. use a standard organization
5. record rationale
6. keep documentation current but not too current
document fairly stable decisions; have a release strategy
7. review documentation for fitness of purpose

avoid ambiguity

always define what the boxes and lines mean

- analyze this example:

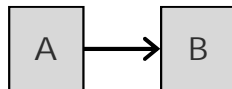


- what do the bubbles mean?
- what do the lines mean?
- what is the significance of the layout?
- why is control process on a higher level?

avoid ambiguity

always define what the boxes and lines mean

- analyze this example:



possible interpretations

- A passes control to B
- A passes data to B
- A gets a value from B
- A streams data to B
- A sends a message to B
- ...

indicators of ambiguity

- no key or legend
- lines all look the same
- arrows mean many things
- confuse architectural views

indicators of preciseness

- lines and boxes have different shapes/colors
- a key explains their meaning
- diagrams do not try to do too much
 - each view of architecture fits on a page
 - separation into views, where necessary
 - use of hierarchy
- clear distinction between viewtypes
 - separation of concerns
 - indicate mappings between views, where appropriate

to improve preciseness

- always include a legend
- be clear what your arrows mean
 - control flow? data flow? service invocation?
 - or ... don't use them at all
- if using a standard notation (e.g., UML) say so
- keep the viewtypes clear
 - recall the viewtypes: module, C&C, allocation
- supplement diagrams with explanation

example: NASA EOSDIS

Earth Observing System Data Information System^(*)

- maintains databases of satellite information
- provides access to this information for scientists throughout the world
- complex system made up of
 - several legacy databases
 - data processing programs
 - scientific applications

^(*) release 6A Segment/Design Specification Section 4.4
NASA Report 305-CD-600-001, March 2001

example: NASA EOSDIS

Earth Observing System Data Information System

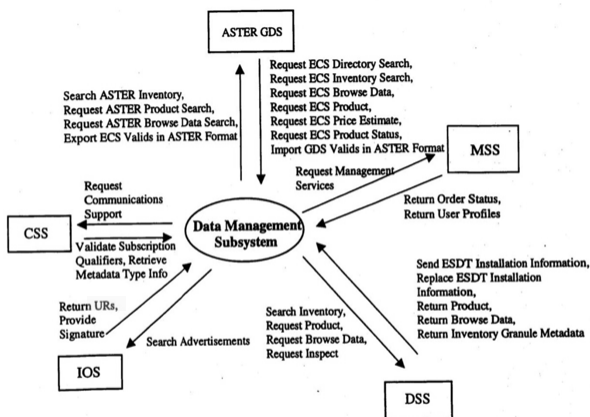


Figure 4.4-1. Data Management Subsystem Context Diagram

example: NASA EOSDIS

Earth Observing System Data Information System

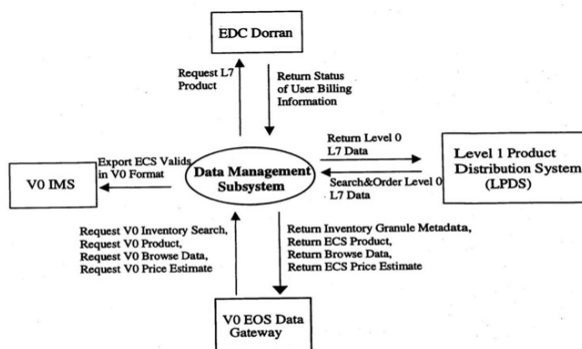


Figure 4.4-2. Data Management Subsystem Context Diagram

example: NASA EOSDIS Earth Observing System Data Information System

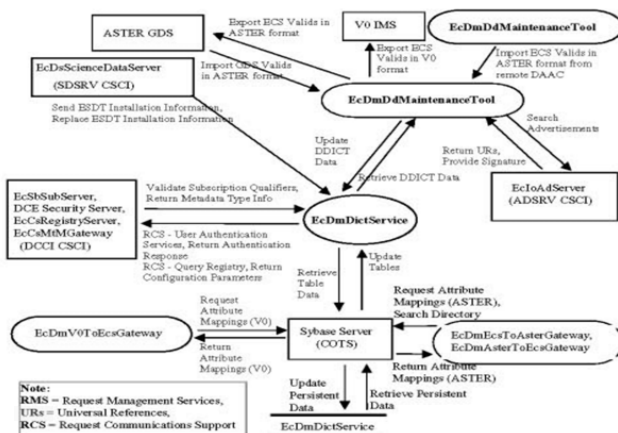


Figure 4.4.1.3-1. Data Dictionary CSCI Architecture Diagram

example: NASA EOSDIS diagram shortcomings

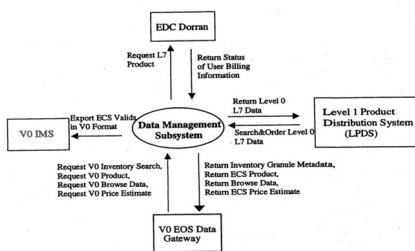


Figure 4.4-2. Data Management Subsystem Context Diagram

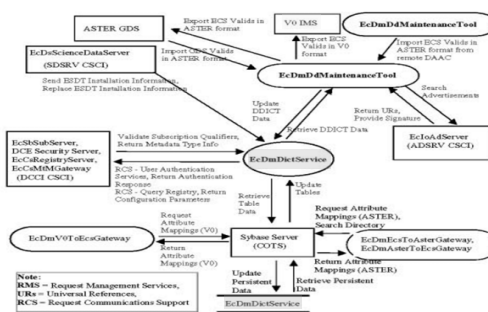


Figure 4.4.1.3-1. Data Dictionary CSCI Architecture Diagram

- not clear what is inside and outside the system
- not clear if there are logical subgroups
- some components appear in multiple diagrams
- some components appear twice in the same diagram
- not clear if notation describes code or run-time structure
- ...

example: NASA EOSDIS diagram shortcomings

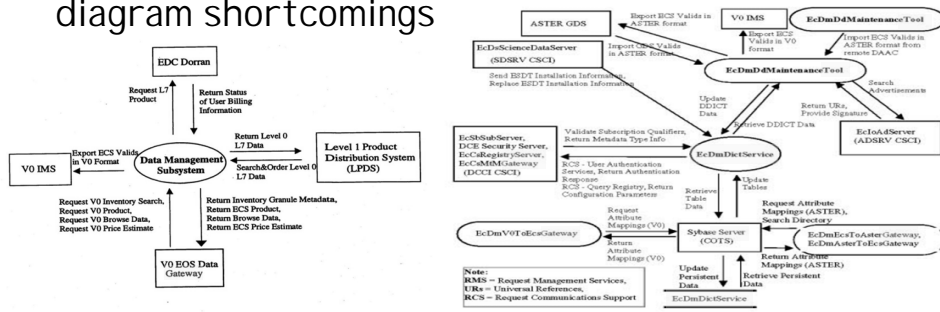


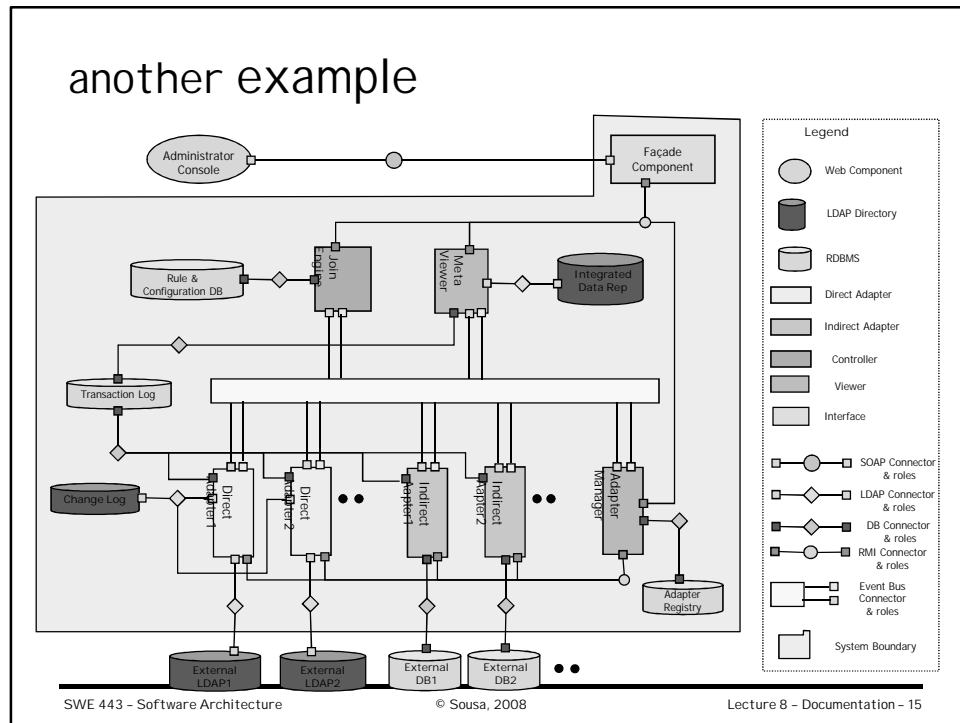
Figure 4.4-2. Data Management Subsystem Context Diagram

Figure 4.4.1-3-1. Data Dictionary CSCI Architecture Diagram

- ...
- not clear what are interfaces and what are components
- not clear what interfaces a component offers
- not clear what is relationship between kinds of components and graphical symbol
- not clear what is the nature of interactions (arrows)

example: NASA EOSDIS documentation could be improved

- step 1: clarify intent of the EOSDIS
 - what is its essential function?
 - what are the main parts?
 - what is inside the system and what is out?
- step 2: using C&C notation
 - what are the main types of runtime elements?
 - what interfaces do they have?
 - what is the nature of interactions between
 - system and outside?
 - different parts of the system?
- step 3: simplify
 - use hierarchy to simplify the description



outline

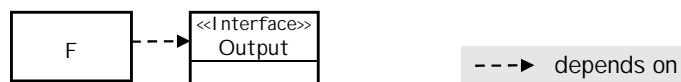
- principles of documentation
 - being precise
 - examples
- documenting interfaces and behavior
- combining views
- going beyond diagrams

documenting interfaces and behavior going from high-level diagrams to detailed ones

- an interface specification is a statement that the architect chooses to make known
 - appears in Module views, and usually
 - defines a set of methods that may be called (aka *provides*)
 - may also define routines required from other parts (aka *requires*)
- an interface is a boundary across which two independent entities interact or communicate
 - appears in C&C views, and usually
 - defines a point of run-time interaction with other parts
 - often called ports to distinguish them from interface specifications

ports and interface specifications are not the same

- consider a filter, *F*, with two outputs both of which write characters to a pipe
 - in the Module view both outputs have the same interface specification

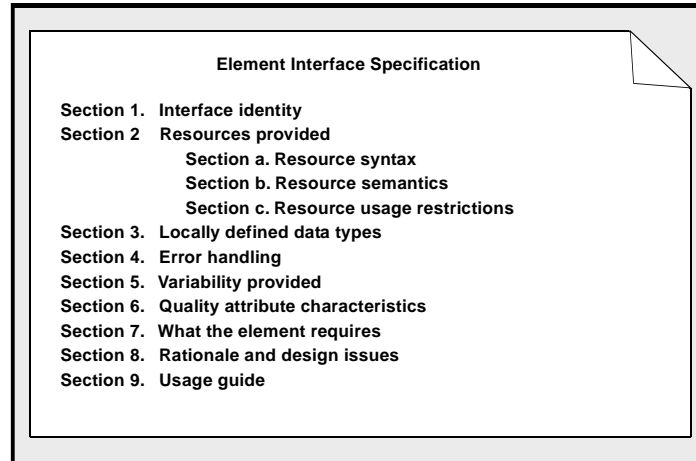


- in the C&C view there are two distinct ports even though their "signatures" are the same



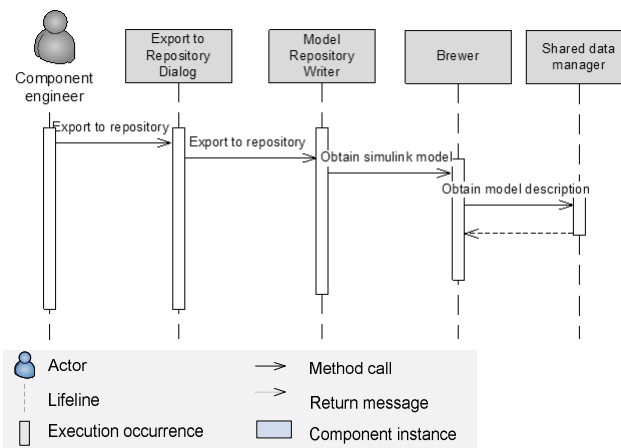
further reading on documenting interfaces

- Clements book, Chapter 7



behavior documentation clarifies sequencing of interactions

- by example: event sequence diagrams



limitations?

outline

- principles of documentation
 - being precise
 - examples
- documenting interfaces and behavior
- combining views
- going beyond diagrams

separate views enable divide and conquer but lead to the problem of relating them

- for example, it is often useful to relate

modules and
components and connectors

module decomposition and
layers

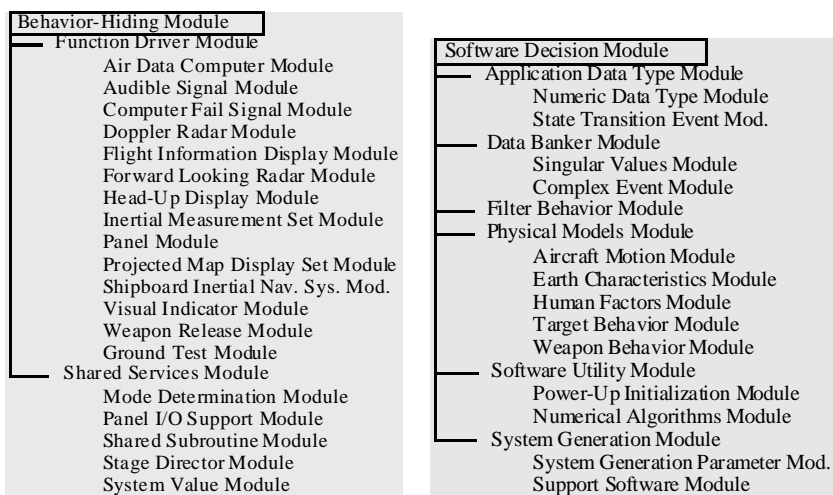
components and connectors and
deployment decisions

layers and
work assignments

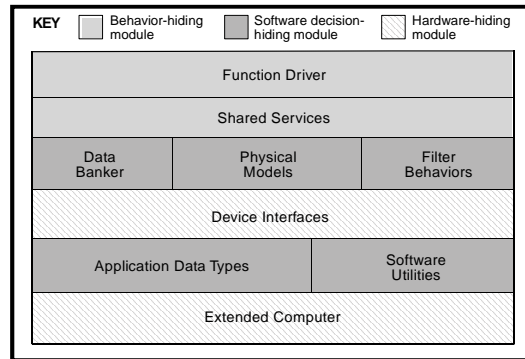
two approaches to relate views

1. build a combined view that shows elements and relations of both constituent views, e.g. overlays
2. create a bridging document that relates elements and relations in one view to elements and relations in another view

recall example decomposition style

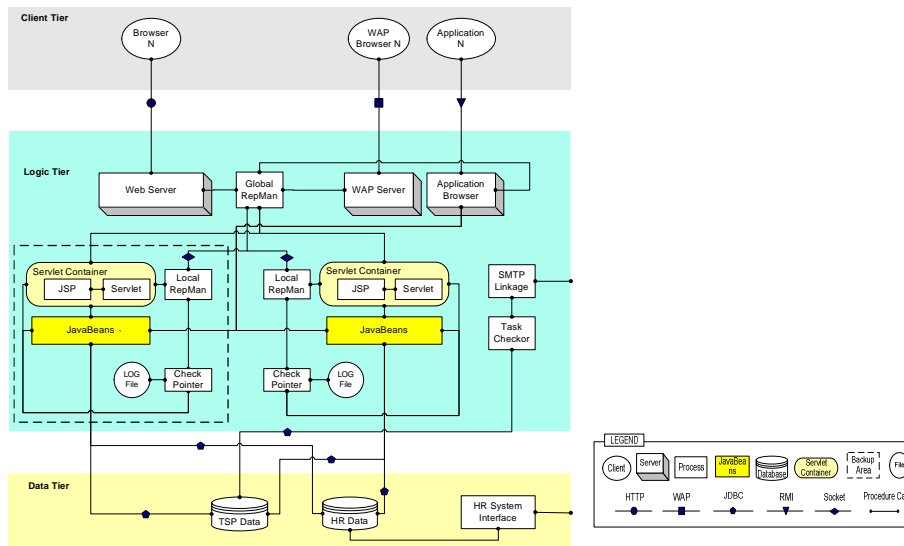


example overlay decomposition and layered views

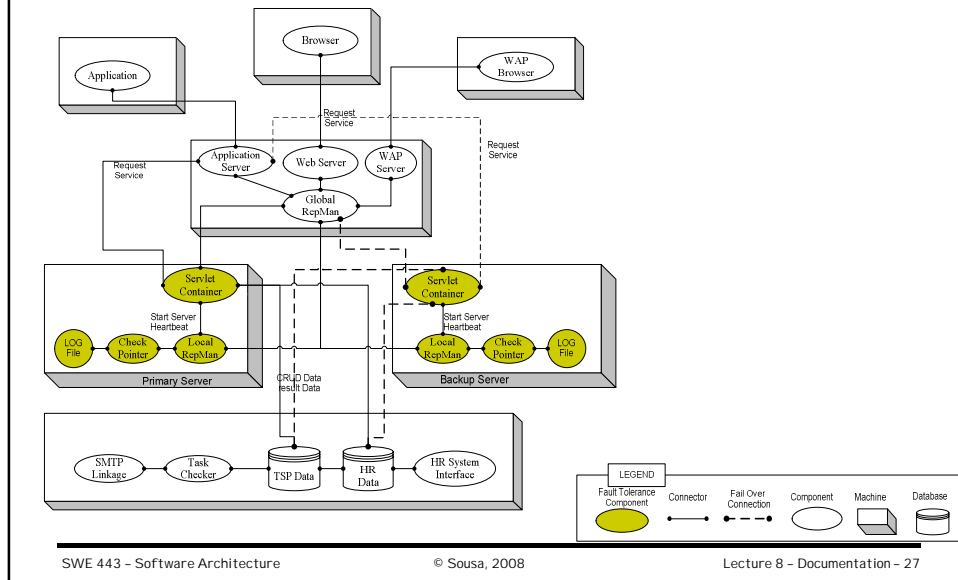


- simple, customized view that combines information from two styles of the same viewtype (module)

example overlay C&C and Tiers



example overlay C&C and Allocation



architecture documentation includes more than diagrams

- statement of requirements
 - business context, rationale for product, domain
 - quality attributes, preferably in terms of scenarios
- statement of constraints
 - business, implementation, deployment
- description of context
 - systems with which it must interact, external interfaces
- architectural diagrams
 - with suitable discrimination of boxes and lines
 - explanatory prose
 - include relevant views

architecture documentation includes more than diagrams

- rationale for the architectural design
 - how it addresses requirements & constraints
 - alternatives considered
- style/product-line issues
 - what are the expected dimensions of variability?
 - what aspects must/should not change?
- management issues
 - implications on organizational structure of development team
 - use of architectural reviews
- other issues
 - operations, administration, and maintenance

in summary

- precise documentation is worth the effort
- level of detail depends on purpose and audience
- different views require different notations
- views may be combined, carefully
- documentation is more than boxes and lines