NTNU – Trondheim
Norwegian University of
Science and Technology

Department of Computer and Information Science

SOLUTION for Examination paper for TDT4240 Software Architecture

The English version is the reference version of the examination paper!

Examination date: Tuesday June 2nd 2015

Solution Problem 1 (20 points)

1.1 The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both.

1.2 Module views: Documents the modules, interfaces and their relationships. Focus on functionality and code-based considerations (component and class diagrams).

1.3 Component-and-connector views: Documents runtime components and connections (Layered, UML component, collaboration, class).

1.4 Allocation views: Documents software elements in one or more external environments which the software is created and executed (UML deployment)

1.5 True: All complex software systems have some internal structure, although it might not be planned or documented.

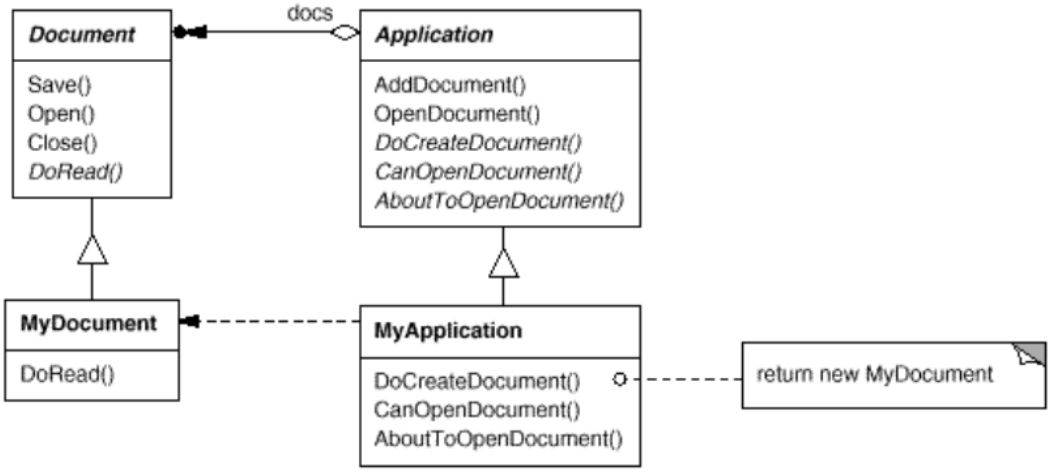
1.6 Development view 4+1 view model: Static organization of software in its development environment to make it easier to distribute work assignments to programmers and coordinate development carried out in parallel.

1.7 Main influences on an architect: Stakeholders (Business, Technical and Project) and Professional.

1.8 Reasons why software architecture is important: Tool to realize a system's quality attributes, reason and manage changes in a system, prediction of a system's qualities, enhances communication among stakeholders, careful planning of most fundamental hardest-to-change design decisions, defines constraints on implementation, dictates structure of an organization and vice versa, basis for evolutionary prototyping, enable reasoning about cost and schedule, enable evaluation of system before implemented, provide a reusable model that can form a product line, reducing design and system complexity, foundation for training...

1.9 Availability tactics: Detect faults (ping/echo, monitor, heartbeat, timestamp, voting, self-test...), Recover from Faults (Active redundancy, spare, exception handling, rollback, software upgrade, shadow...), Prevent Faults (Transactions, Predictive Model, Exception Prevention, Removal from Service...)

1.10 Template method:



1.11 Architecturally Significant Requirement/Architectural Driver: A requirement that will have a profound effect on the architecture.

1.12 Parameters in performance models: arrival rate of events, queuing discipline,

scheduling algorithm, service time for events, network topology, network bandwidth, routing algorithm

1.13 *Security checklists* can be used to analyze the quality attribute security.

1.14 ADD: Two alternatives. Textbook:

1. Choose an element of the system to design
2. Identify the ASRs for the chosen element
3. Generate a design solution for the chosen element
4. Inventory remaining requirements and select the input for next iteration
5. Repeat steps 1-4 until all ASRs have been satisfied

Slide approach:

1. Choose module to decompose
2. Refine module:
    a. Choose architectural drivers
    b. Choose/Create architectural patterns satisfying drivers
    c. Instantiate modules and allocate functionality
    d. Define interfaces of child modules
    e. Verify and refine use cases and quality scenarios
3. Repeat steps above

1.15 Architectural erosion: Gap observed between the planned and the actual architecture as realized in its implementation.

1.16 Techniques to keep code and software architecture consistent: Embed design in code, Use frameworks, Use code templates, Use tools to enforce architectural constraints, Mark documents when out of date, Schedule documentation/code synchronization times

1.17 ATAM: Architecture Trade-off Analysis Method

1.18 Output ATAM: Concise presentation of architecture, Definition of business goals, Utility tree, mapping of architectural decisions to quality requirements, Sensitivity points, tradeoff points, Risk, Non-risks, Risk themes.

1.19 Reconstruction process:

1. Information extraction
2. Database construction
3. View fusion
4. Reconstruction of architecture

1.20 High priority when designing software architecture for a software product line: Identify the stable parts and the variation parts in the product line. Support the variation points. Variation may vary in behavior, quality attributes, platform, network, physical configuration, middleware, scale factors etc.

Solution Problem 2 (5 points)

1. d) Pipe-and-Filter: The example consists of several ways transforming an input to a different output, and the need to combine these different encoding filters.
2. c) Model-View-Controller: The same type of data should be shown on various types of display as well as various types of graphical design. MVC makes it easy to map different types of views to the same model.
3. f) Peer-to-Peer: As the smart phones need communicate directly to each other without any specific sever through the internet, the Peer-to-Peer patters would fit well here where all devices will both have the role as a master and a slave.
4. g) Service-Oriented: This is a typical description of a service-oriented architecture where other applications can access data and services through standard APIs.
5. i) Map-Reduce: The challenge here is to be able to reduce the computation to a number of similar parallel computations where the results later can be combined. The example describe fits this description.

Solution Problem 3: Edge-dominant system (5 points)

a) The core must be highly modular, layered, highly robust with respect to errors, well documented APIs, high security, provide discovery service, efficient processing, constructed by a small tight-knit team, etc.
b) In open content systems the users can provide content and services related to the system but they do not contribute to the core. In open source software, the users can provide code including code for the core.
c) Open content systems: Wikipedia, YouTube, Kahoot!, Facebook. Open source: Apache web server, Eclipse, Firefox, Thunderbird, Linux

Solution Problem 4: Cloud Architecture (4 points)

a) Private cloud: Cloud infrastructure owned/operated by a single organization, Public cloud: Cloud infrastructure available to the general public, Community cloud: Cloud infrastructure shared/supported by several organizations, Hybrid cloud: Combinations of the above.
b) Hypervisor (OS to create and manage virtual machines), Virtual Machine (software abstraction of hardware), File system (multiple users, multiple storage), Network (IP address handling).

Solution Problem 5: Quality Attribute Scenario (6 points)

a) Quality attribute scenario on usability:

Source: Tax payer
Stimulus: Complete the tax return form (Selvangivelsen)
Environment: Runtime
Artifacts: The whole system for tax return (Selvangivelsen)
Response: Provide help to guide the user through the form
Response Measure: 90% of users give the system the score high in user satisfaction

b) Quality attribute scenario on modifiability:

Source: Developer
Stimulus: Change the forms for tax deduction
Environment: design time
Artifacts: Deduction GUI component and deduction server component
Response: Make and test modification without any side effects
Response Measure: 8 man hours

c) Quality attribute scenario on security:

Source: Human attacker
Stimulus: Unauthorized attempt to fill in the tax deduction for another person
Environment: Online connected system where the server is protected behind a firewall
Artifacts: The whole system
Response: Data and services are protected from unauthorized access, and the attempt of attack is logged.
Response Measure: 99.9999% of the attacks were resisted.

Problem 6 Design a software architecture (30 points)

a) Architectural Significant Requirements/Architectural Drivers – 2 points

- ASR1: The papers in the system must be protected from unauthorized access (security).
- ASR2: The system must provide various user interfaces including web on PC, tablets and smart phones, as well as an app for tablets and smart phones (modifiability).
- ASR4: The system should be easy to modify and expand, as there might be changes to the conference process or new features are wanted in the future (modifiability).

b) Architectural tactics and patterns – 3 points

Security:

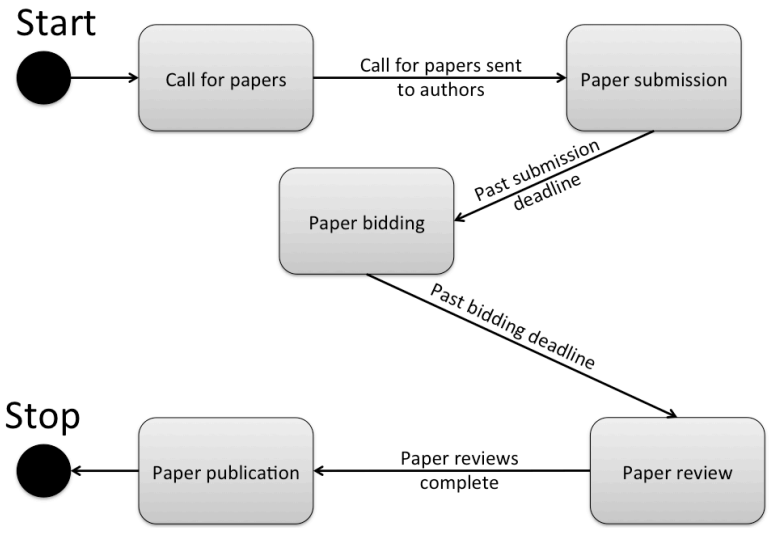
- Authenticate
- Authorize
- Encrypt Data

Modifiability:

- Increase Cohesion
- Reduce Coupling
- Model-View Controller
- Observer pattern
- Client-Server

c) Process view – 8 points

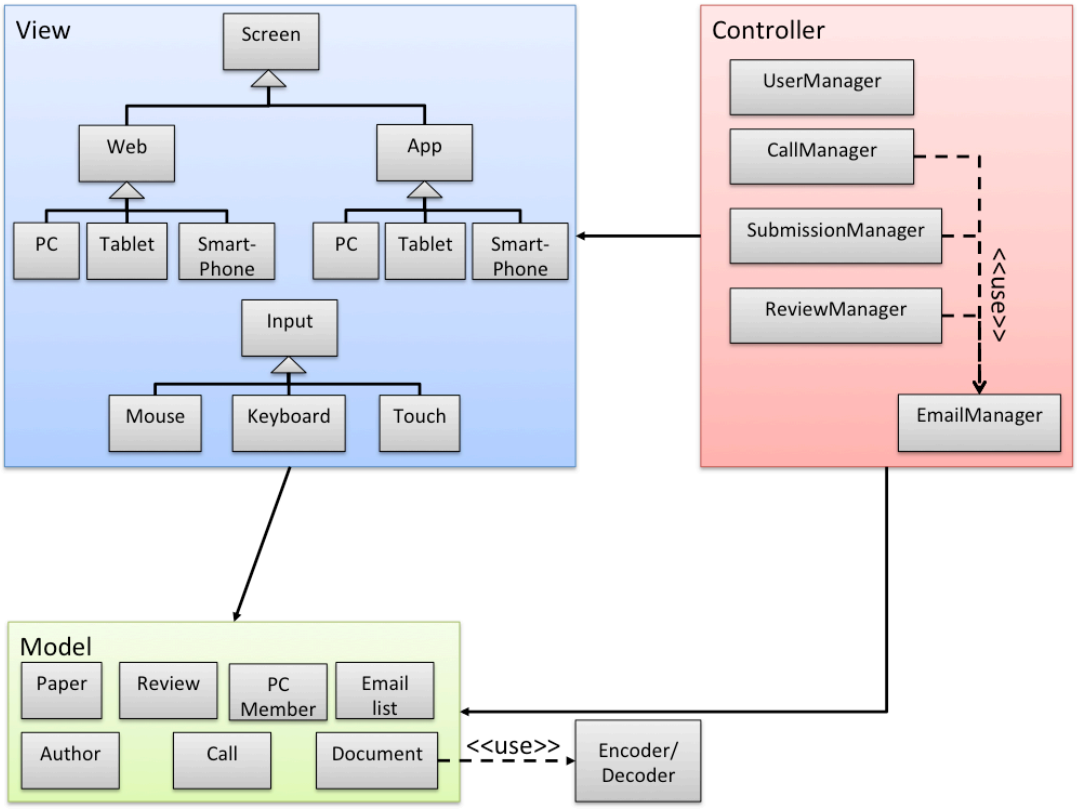
The process view shows the states of the system:



The review process starts with processing the call for papers and sending out call for papers to all authors. The next step of the process is for the authors to submit their papers to the conference system within a specified submission deadline. The next step is for the reviewers (the program committee to bid on what papers they should review within a bidding deadline. The next step is then to review the papers (carried out by the reviewers) which include giving the papers scores, ranking the papers and

choosing which papers should be accepted or not. The final step is to notify the authors about the state of the papers and then publish them.

d) Logical view – 14 points



This local view is design around the model view controller pattern to make it flexible to adapt to various devices and APIs.
The *view module* deals with producing the visuals on the screen as well as receiving input from the user.
The *controller* will listen to changes in the view using the observer pattern and execute changes in the model according to user actions. The UserManager deals with login, user authentication and authorization through the two different roles PC Member and Author. The CallManager deals with issuing and managing call for papers. The SubmissionManager is responsible for all the tasks related to submitting a paper including asking for the necessary information from the user. The ReviewManager provide functionality for doing bidding on papers, for doing reviewing of papers, for ranking papers, and for marking which papers to accept or reject. The EmailManager is used by the Call-, Submission-, and ReviewManager for handling communication between PC Members and Authors.
The *Model* contains typical information objects required in the review process: Paper, Review, PC Member, Email list, Author, Call and Document. Document can use an Encoder/Decoder to encrypt the PDF document whenever needed.
The client-server pattern would have been documented in the physical view.

e) Architectural rationale – 3 points

The emphasis of the design was on modifiability and security. The architecture makes it easy to change support for various user interfaces, as well as supporting changes in

the review process by adding or modifying managers in the controller. The security aspects are handled by the UserManager in addition to Encoder/Decoder.