Architectural tactics or strategies - Notes

Keorapetse Shiko

09 March 2016

1 Architectural Tactics

1.1 Introduction

Quality requirements have a significant influence on the software architecture of a system. Architectural tactics are techniques that an architect may use to comprehensively achieve the quality requirements. For each quality requirement listed, potential tactics will be mentioned.

1.2 Quality requirements

- Performance
  - The system is fairly small and simple. Any form or action of enhancing performance may be relatively negligible but a couple of tactics can be implemented.
  - Tactics
    - Spread load across resources
      - Concurrency can be introduced. Processing different streams of events on different threads and creating additional threads to process different sets of activities can be done by processing requests in parallel. Appropriately allocating threads to resources can maximally exploit concurrency.
    - Resource re-use, caching.
      - Considering the Readers/Writers problem, multiple readers will have access to view the resource while only one writer can have access to edit the resource. Multiple copies of data can be maintained. Caching is a tactic that replicates data which would be on the same repository to reduce contention.
- Reliability
  - Tactics
    - Resource locking
      - A resource or a part/section of the resource will be locked, the correct resource will be returned by making sure that the correct resource is requested to the database and only valid requests will be approved. After the resource is returned, only one user will be able to edit that resource.

The other users will be notified about the lock by getting conflicts when they try to pull the resource.

    * Maintain backup
        · The database will have a back up of all of the information that it stores.

- Scalability
    - Tactics
        * Increasing capacity of communication channels
            · Users will access the system through a website front-end on a web server maintained by the department. The site will be browser independent in order to increase communication channels. The Android system will also be used as a replacement for the website ans vice versa which allow use of mobile phones or tablets.

- Security
    - Tactics
        * Authentication
            · Users will be authenticated via their usernames and passwords by making use of server side validation.
        * Encryption(hashing passwords)
            · Passwords are to be hashed using at least sha256 and should be stored as such within the database along with a salt.
        * Authorisation
            · Users will be grouped by user classes to ensure access control. Users rights to access and modification of data will be determined by the class the user is in.
        * Drop connection
            · Sessions will be checked periodically(every 10 minutes). Should a user be inactive for more than the stipulated period, the session will be terminated after the state of the system gets stored for possible retrieval.

- Flexibility
    - Tactics

        * Contract based
            · The use of the MVC architecture allows the system to be fairly flexible. The model will be the data(database) itself or actions on the database, the view will be the interface as experienced by the user and the controller will be the facilitator of the program flow logic. This modular format will allow changes to one aspect without greatly affecting the other aspects.

- Maintainability
    - Tactics
        * Localise changes
            · Using MVC architecture, changes are localised. Service contracts will allow semantic coherence, the model component contains and encapsulates the functional core of the application, intermediaries in the form of the controller and the view, and runtime binding so that the view can be opened and closed dynamically.
        * Prevention of ripple effect
            · Intermediaries will be used to prevent ripple effects. The controller will realize the interface if the view and respond to accordingly, it will be the intermediary between the view and the model. The view will thus realize the model's interface and produce the output, making it an intermediary between the model and the controller.

- Auditability/monitorability
    - Tactics
        * Logging
            · An audit trail will be maintained. This will be done by having a copy of each transaction applied to the data in the system along with the identifying information of the user.

- Integrability
    - Tactics
        * Support communication channels

            - The system will be made web browser independent to support the usual communication channels. Java will be used to integrate Android and the API.

- Cost
  - Tactics
    - Open source software
      - The software used will be open software. Programming languages may include Java and python, frameworks like bootstrap and django and so on.

- Usability
  - The interface should be lightweight.
  - The interface should be intuitive to use as well as obey Human Computer Interaction guidelines so that it is efficient and easy to use.
  - Tactics
    - Separate user interface
    - User initiative(Cancel, undo)
    - System initiative(Maintain user,system and task models)
      - Logging