Christopher Cargile CSCI656 – G. Rudolph 2/8/14 Homework Exercise

Responses for Chapter 3 Textbook Questions

3.6.1) Software architecture and design are distinctive aspects of software development. Creating the artifacts for software design may take place prior to constructing a piece of or an entire software program. The process of defining a product architecture requires reevaluation of needs and concerns that influence the software system. Changes in need are due to changing desires of the customer or to impediments or alternatives encountered on an ongoing basis throughout the development of a system. Capturing design decisions, in contrast to architectural decisions, may lead to documentation that reflects stakeholder intentions. To provide an illustration of technologically relevant or programmatically feasible options associated with the production of operable systems, an architect may capture the essences of design decisions in a more functional overview of a system that still remains abstract. The architecture may be non-physical as is also possible for a design decision and may be made prior to commencing development or changing throughout the project but will remain more of an idea or intent versus a means for putting an actionable plan to a working effect.

3.6.2) Software components and software connectors are distinguishable by the functions they serve and usage by which they interact in a system. While components may consist of data, connectors are more so a means for conveying that data within the system, acting as a bridge between various components themselves. Connectors have been given a more prominent role in architectures in modern times, becoming more like first-class citizens versus tiny lines connecting boxes, in more traditional designs, such as the desktop PC architectures of late.

3.6.3) The design decisions addressed in 3.1.5 are reflective of those encountered when applying the architectural style familiarly described as "highly distributed." The guidelines provided could be turned into specific design decisions applicable to the application scenario of building a medical appointments scheduler after assumptions concerning the underlying intent of the design decisions as regards the architecture is presumed. Regardless of the mechanism used in the application of the 'highly-distributed' style, specific choices affecting properties associated with individual components and the system as a whole are to be faced.

At a high level, the physical means for separating software components which provide services from those which request them of the application can be implemented in terms of distributing these components amongst distinctive locations. The logical components that provide the algorithm to effect the most efficient scheduling approach may be stored on a machine or environment, separately, in terms of hardware, in this case, in other words. Components that initiate scheduling requests are not a part of the application proper as one might encounter in a web application running on a server that gets all its data from the database connector and database instance on the same machine as the app server, itself, in the physically and logically separated design prototype, for example.

(i) In terms of identity agnosticism, TCP's means are a suitable medium through which to carry out such a decision (in contrast to the web application that accepts and responds to all logic, itself and ties requests to a process specifically associated with each requester) as the REST principle can be prescribed or adhered to using TCP.

(ii) By providing a key or index into the schedulers' queue or results, in this situation the scheduler can maintain an open line for inbound communication from requesters, and perhaps upon receiving the request the receiver can immediately return an isAccepted() handle that allows isScheduledYet() calls to be made by the requester thereafter and a retrieveScheduleDetails() which first validates using an isScheduledYet() call inside it.

(iii) To keep requesters as entities separate from each other, the requesters are insulated by using the 'handler' technique described in (ii) and the data that is generated in association with each request is reflective of this idea in terms of its relational data storage process and structure. The requester thus becomes dependent on the response mechanism that the server delivered to it, which may remain physically separated. The handler technique is implemented using an implementation of connectors, similar to as is described on pg.71, paragraph 2, in association with the 'procedure call' approach.

(iv) Multiple service providers can emerge to service the growing or extant requests to the server(s) by peeking into a queuing server that managers the active handles into the scheduler and can return scheduling details to each (active) requester based on the mapping between the handle-ID's and scheduling objects with which they are associated.