2QueueOOL4u

October 31, 2017

Architecture description and System design



Metadata:

Project name: NextText

Team name: 2QueueOOL4U

Team members:

James Mulvenna/100965629

Yue Zhang/100980408

Devon Plouffe/100715712

Architecture Description:

Over the course of our application advancement, our strategy was frequently modified due to our repeatedly changing conditions. We ran into multiple architectural complications. One being Android mobile application research with regards to our proposed product. As it stands, we have used a collection of architectural styles as no one style best suited our system. Diverse non-functional/functional properties apply to our product, and for this reason particular imperative styles were used respectively.

During the D0 phase 2QueueOOL4U was most concerned with the practicability of the system, we assessed the inter-dependent components and conducted research to determine how achievable our proposed system was. Fortunately, we found in the investigation period that designing our proposed application on iOS would be near unattainable by cause of Apples restrictions on message automation. Despite running into arguably our most important issue, we made the decision as a team to change platforms, as Androids development guidelines solved this obstacle.

In totality, at this moment our product consists of numerous architectural styles including but not limited to: object-oriented, client-server, mobile code, and finally event-driven. Although at this moment these architectural styles apply to our system, it would be naive of 2QueueOOL4U to declare these as the only styles. Realizing the remaining term of development, perhaps a new style might be used to account for outstanding system requirements.

**Object-oriented**

By reason of 2QueueOOL4U’s system utilizing many independent/dependent objects, using an object-oriented design gave us the ability to administer proper data abstraction. As it goes, our system employs several data objects including: Message, Time, Location, Weather, and more in which are all wrapped in an object which serves our backend database. Considering objects such as Location, and Weather service external API processes, encapsulation was used by wrapping the respective objects in an object which suppresses these background methods. All user input is recorded in the objects listed above and compressed into the final wrapped object called MessageWrapper. Proceeding, our database retrieves this information to be called upon when applicable. In light of the authentic definition of the Object-oriented architectural description, it is clear to notice NextText’s division of responsibility as well as its’ collaborative objects communicating through interfaces leading to the final result.

**Client-server**

NextText exploits its client-server composition constantly when operating. Any user information retrieved, is stored on the backend server. When required, the unified trigger imposed by the client retrieves the respective data from the server to be dispatched accordingly. The server utilizes SQL to effectively compose transactions such as, get all, get sms, get email, add, update, delete, delete all, and many more useful methods.

As a result of the server being accessed indefinitely by the client, it serves the most important role in the system. In order to access/store data as per client, the server must ensure effortless transactions, integrity, scalability, synchronization, and more mated traits.

**Mobile code**

For the sake of our weather trigger adopting the Yahoo weather API, we must make use of requests to JSON data on the Yahoo servers by cause of the client assigning a constraint based on their weather expectations. These requests are done in intervals of time, consisting of mandatory information such as the city and country to perform the remote function and obtain the information needed to inquire the clients’ constraint.

**Event Driven**

The fundamental objective of NextText is to send scheduled messages such as SMS or Email, based on time, location, or weather. In other words, every message is coupled with an event in which fires action. Various senders and receivers are in place to handle methodology implicitly. Our system makes use of the Event Driven Architecture at every moment. Concealed methods are called for each user-system interaction in order to accomplish prescribed events.

System Design:

* Document why you selected your design
* Describe the design of your system, such that a programmer could implement some subset of the system and integrate it
* Discuss how your design would support or inhibit changing requirements
* Identify a way you think your system may need to evolve in the future and describe how your project’s design would support these changes
* Analysis of how your design minimizes coupling and accommodates changing requirements is required

**Composite**

**Façade**

**Template Method**

**Iterator**

**Chain of responsibility?**

**Active Object?!**

**Balking?!**

**Scheduler?!**

https://pmsware.wordpress.com/tag/computational-design-patterns/

* Clear description of the structure of the components and its externally visible interfaces
* Compare the applicability of your design compared to alternative designs in this discussion
* Reference descriptions in architecture of important patterns, classes abstractions, and data structures/algorithms that are critical to the successful implementation of your system
* At minimum, include a class diagram that shows all classes and public API
* Include a sequence diagram (optional) that captures how your system behaves for each scenario from the initial proposal
* Clarify where classes reside (client, server) as well as any external API your system will use

Task Overview:

For the reason that initial complications emerged, system reconstructions were necessary. As a result of this, following early preparations and studies, roles were set. Roles were appointed with relation to outstanding development time, particular developer strengths, and developer interests. Subsequently, divisions of tasks were split in this way:

**James**

James takes control of the entire backend, consisting of the Database and Models. He administers the event handlers as well as takes care of the Weather event trigger. Finally, he yields the documentation including D0, D1, D2, D3, as well as the presentation slides for the proposal, and the demonstration script.

**Yue**

Yue manages the entire frontend user-interface. He is to some extent familiar with the backend as James, and Devon supply him with the functionality he demands. Yue develops client initiated event triggers and finally contributes to most documentation. By cause of Yue requiring the backend interface yet not vice versa, the backend developers have a general understanding of the front-end, however not enough to develop upon, but rather to document its purpose.

**Devon**

Devon is in charge of the Location event trigger. Devon also contributed to the component diagram in D2, as well as the presentation slides for the proposal.

Oral component (Prepared to defend system design, asked how design could adapt to specific constraint, “you must now support X, how would you do that?)

**Page overview:**

1. Page 2; metadata
2. Pages 3-6; architectural description
3. Pages 7-10; system design
4. Page 11; assignment of tasks for each team member

Worth 30% of final grade, written is worth 60% and oral is worth 40%.