Contents

1. Language Training Application

Repository: **languageApp & learnServer**

Source code main branch: masterOfPuppets

The main challenge was to accommodate some resource intensive features in a desktop application. It is a live class room application with different real-time activities available to groups and individual persons, and it runs on LAN. The project was initially taken by another company and failed due to complexity in features and resource management. We could achieve reasonable performance with all the features. Some complicated ones are:

1. Teacher could control other computers
2. Teacher could start a live class-room
3. A **live class room** has different individual and group activities like:
    a. content sharing – teacher could share interactive contents
    b. examination and exercise
    c. round-table discussion: where there is a leader who controls who can speak at a given time.
    d. Group discussion: similar to group audio conferencing
    e. Screen broadcasting
    f. Locking computers so that students cannot do anything else other than what is presented to them.

Some of the challenges that I faced

I build a whole desktop application framework to manage different resources. The most crucial thing was to allocate and release resources, and enable real-time communication to realize the live activities. So, I built a framework with following features:

1. Main application class initiates a dependency container, registers all the services, and also resolves all the services which runs through the application lifetime. No feature could create these objects. (**languageApp / languageApp / App.xaml.cs**)
2. Message Queue Library for real-time communication: I wrote the whole stack here and it was a painful job (**languageApp / languageApp / modules / mq**). It was built on Apache NMS for ActiveMQ to solve a few crucial issues which are neither addressed nor documented in the library:
    a. Apache NMS library is not fault tolerant. So, whenever the LAN connection was broken, whole communication system stopped working. I created 1 wrapper called MQService which periodically refreshes the connection if needed. The service also hides the actual connection object so that no other code holds onto an expired connection object.
    b. When a connection was lost, all the producers and consumers broke down along with the session. So, I created wrapper classes, Publisher and Subscriber respectively, for them to hide actual objects and also recreate them when expired.
    c. Lastly, I solved a issue which was really difficult to catch. Whenever a student logged in, he was subscribed to his command channel. Now when the user logged out and logged in again without closing the application, he was missing some commands. Initially I had known idea how he was missing commands because MQ server monitor service showed all the messages coming and going out right. After a few days I discovered that the consumer QUEUE has two sessions and it was sending messages in round-robin. I wrote code to dispose the session and thought it was solved. But the problem persisted. Then I found it was a bug in NMS library, it does not dispose sessions correctly. Then I solved the problem by recreating the whole connection.
3. Core classes to obtain and release resources uniformly: I wrote the whole core package which acts as the base building blocks for the application (**/ languageApp / modules / core /).** The most important class here is the BasePage class which has the **template methods** for acquiring and releasing resources. It also initializes page events which are necessary to process real-time commands.The crucial feature was to unregistering from events to prevent memory leak when the page was unloaded.
4. Command library: (**languageApp / languageApp / modules / commands /)** package has the base command protocol which I wrote. Features communicate through this package

rather than accessing the publishers and subscribers directly. **CommunicationChannelManager** does the dirty job of creating and disposing the channels. So, it's easily monitorable and manageable.

5. Command Processing: This was a challenging job. Apache NMS doesn't intelligently handle message consumer threads. It actually creates threads for each message received. So, when we joined a live class where many features were communicating with each other, the application memory consumption grew very fast (in gigabytes). One of the core reason was the message consumption threads. The final version of the command processing model is here:

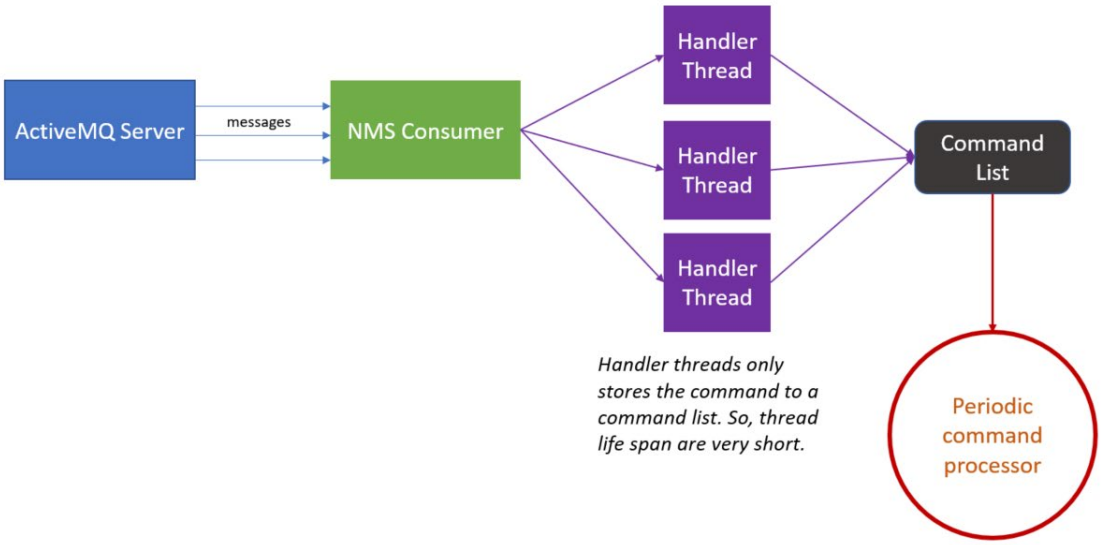
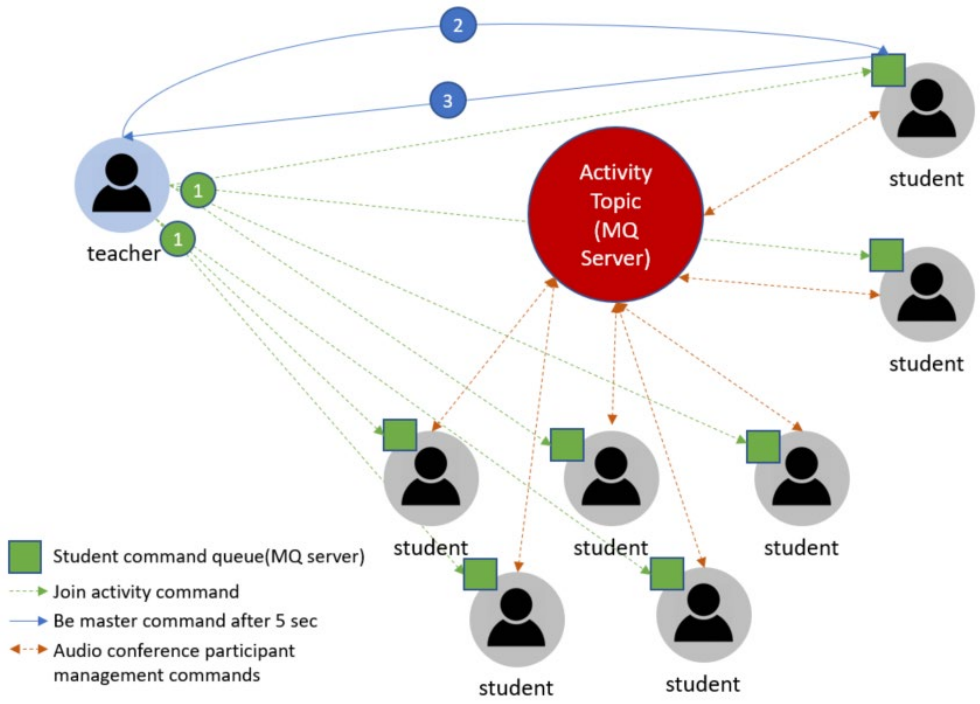


Fig: reducing message event thread pressure on the application

6. Activity Dashboard: the live class room features are located under (**languageApp / languageApp / modules / activityBoard / activity /**) Most of them are highly complex. I made over 80% of this part. It's easier for me to explain it in a conversation.
7. Audio conference networking: (**languageApp / languageApp / modules / audio /)**For audio conference we improve a library from our VOIP dialer product. To manage participants in the conference we made the manager package. The final communication process looks like this:



2. Lawbase

Repository: **lawbase**

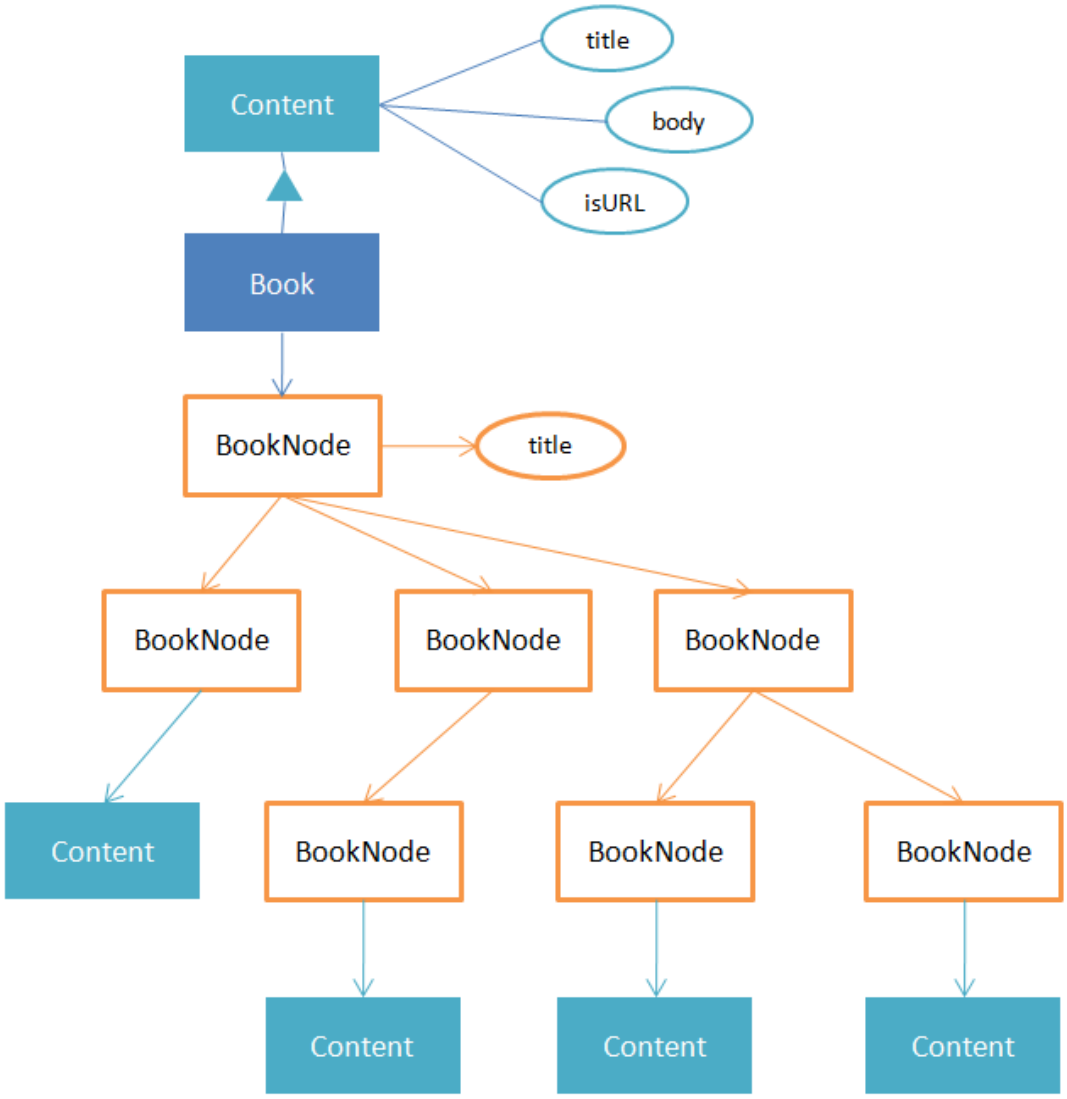
Source code main branch: master

A database and search engine of cases, acts, books, journals for lawyers.

Challenge 1: accommodating different kinds of cases, acts, and books

(lawbase / src / main / java / com / book /)

Acts, cases, and books have different structures. The contents are laid out into sections, pages, parts, volumes, etc. And the client wanted to navigate through the structural parts in front-end. Initially I thought about using mysql as that's our standard in the company. But, after analysis the structures well, I built a schema-less model in MongoDB which can store any kind of book. It is a very simple yet powerful model. In the most basic form the structure of a book is



With this structure we made

1. Courtbooks which has years, volumes in a year, and caseBooks in a volume.
2. Case books holds details of a case. It can have different structures. A general one has parts, sections in parts
3. Acts, journals, and published books.

Challenge 2: Search engine synchronization

(lawbase / src / main / java / com / solr)

I have implemented the search engine with apache solr. Instead of simple MongoDB to Solr synchronization model, I wrote a whole library to fine tune indexing, balancing load, search repositories, updating content through reactive events, etc. I wish to make this library open source. When a record like act is added, updated, or deleted, the ActService publishes an event to a reactive stream. SolrActService consumes that event and updates the solr database. So, ActService do not need to wait for execution of search indexing. I have also wrote a scheduled job to rebuild whole index through admin panel. The reactive library used is RxJava.

3. Bangla Language Projects

Bangla Grammar and Spell Checker:

Repository: **BanglaSpellGrammar & GrammarChecking**

I contributed to all the algorithms and solutions. I also wrote the **BanglaSpellGrammar / src / main / java / com / bangla / DEDM / errorChecker /** package. Here I wrote an algorithm for replacement, deletion, and insertion error which is faster than edit-distance and another algorithm suggested by a paper which have used in writing tools. I will publish it in a paper. I am also trying to define a X-bar tree for Bangla language which will also be published if successful.

NLP Tools:

I am also working on a Bangla Corpus Project for the government. I am building the platform in micro-service architecture and spring webflux (a reactive web framework built on projectreactor.io) which can easily be scaled to serve millions of hits per second. Along with the platform, I am building some phonetics and grammar tools and leading the team to build over 30 tools to gather and analyze text and voice data for corpus. The corpus will also have public services to be integrated with third-party software.

4. Mazefire

I transformed the mazefire.com codebase. It had so many issues. This is a Boston based startup currently Led by Professor Donald O'Malley (Chairman), Northeastern University, and Jay Japra(CEO). One of the core things I did here was to normalize database for maze play history. It is now easy to run statistical operations and show reports to teachers based on student's play moves. As this is a USA-based product, I couldn't give you the access to the code. But both Don and Jay would be happy to share their experience working with me with specific details.