<u>Designing a new network protocol</u>

Communication patterns of the network application:

The architecture of our network application will be client | server. Each node (end system) can be either a job-seeker or job-creator but only one at a time. The job-creators will open a server socket connection on a specified port allowing the job-seekers to connect to them. For this they will need the ip as well as the port. This will prevent the job seeker nodes to connecting directly to each other. The nodes will follow the interactive communication method. They will both be prompted for inputs to continue the connection as well as input data for the computational task. We decided to go with the interactive model to fulfil the requirement the creator can have the option to continue the connection or close it as well as give the seeker the option to reject/accept jobs. Since the job seeker can only connect to one job creator at a time, we chose our communication scope to follow 'one-to-one'. Once a connection is established between the creator server and seeker client no other seeker can connect to that creator until the connection is terminated.

Design Goals:

For this network application a reliable message exchange is important as we need to ensure the results or inputs for a job is accurate and uncorrupt. To fulfil this requirement if the connection between a job seeker or job creator is interrupted the process start over again from the beginning once they connect again. Since our tasks do not hold any private information and the seekers are used for computational resources there are not many security measures in place. Our main security is that the specific IPs and ports of the creators must be known to the seekers for them to establish a connection in the first place. The ports can always be subject to change in case of disruption. We decided the job creators will have a persistent connection for as long as the user controlling that node wishes. This means even if a seeker disconnects, the job creator server socket will stay open allowing a different seeker to connect. To handle errors and failures, the application will display to the user an error message of what went wrong. The cause of these errors could include input errors or unavailable servers. Our protocol is as simple as it can be with plain text messages explaining to users what to input with their options. Our application is extremely extensible as we can add/remove requirements and tasks as we please. Regarding scalability, if there are end systems able to run the network application the scalability is endless. The only issue we can foresee is too many end systems running as job seekers and there not being enough job creators to keep up.

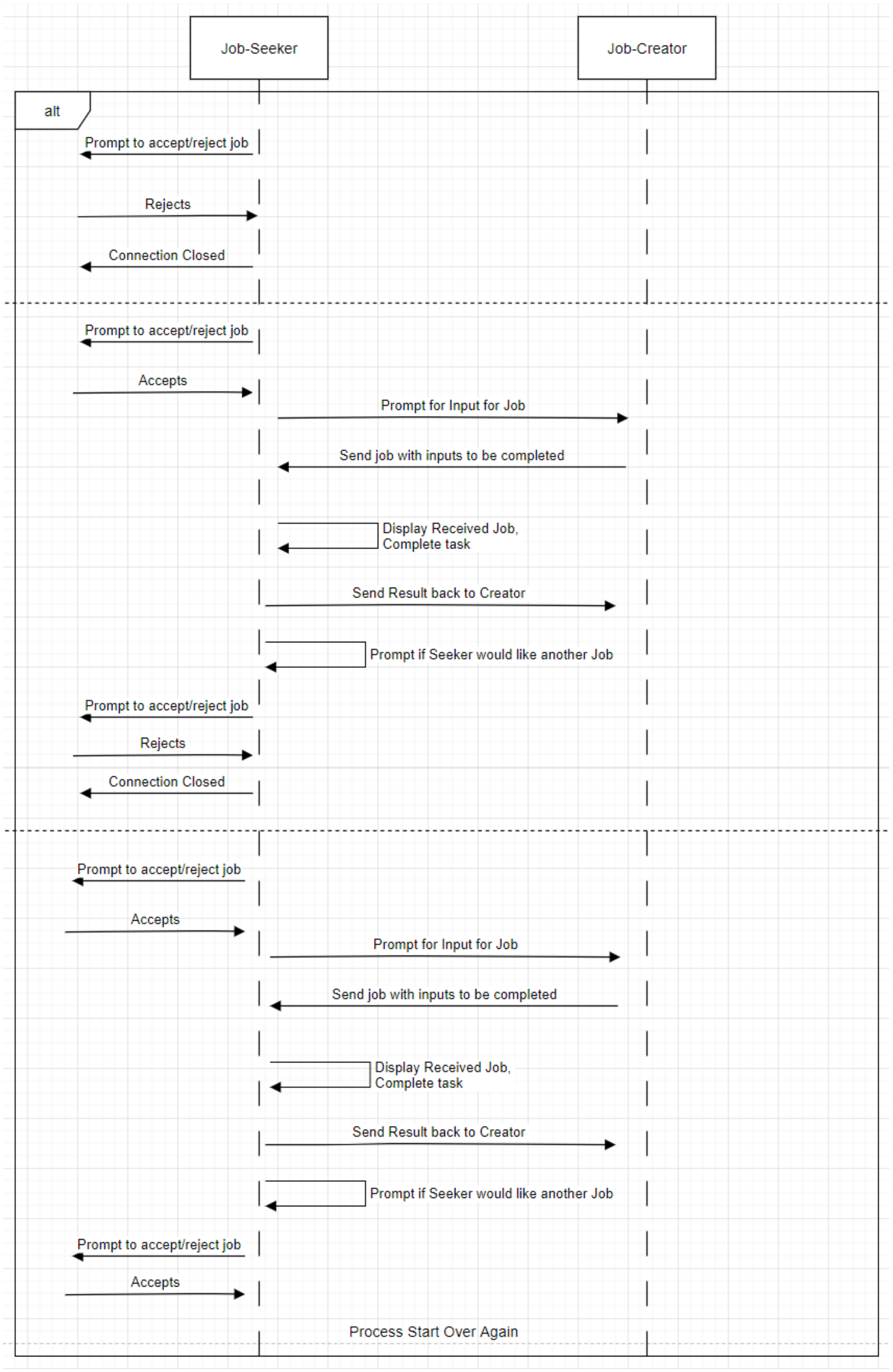
Message Design

Our message format will be text-protocols. We decided to go with text-protocol for the simplicity of debugging as well as being able to communicate to users specific requirements. It will also allow us to display error messages in simple text explaining to the user what went wrong. Most of our command and control messages are requests/prompts for the user. One example is a message will appear asking the job seeker if they want to accept a job from the creator they are connected to. After the control message the stage of the interaction continues to the job itself prompting even more command and control messages. Another example would be the creator asking for inputs regarding an addition job. Our data messages would include updates about the job as well as the results.

Design the communication rules

When a client (job seeker) connects to a server (job creator) a command message will be sent to the seeker requesting an input. This will give the seeker a chance to accept the job or reject the job. The seeker will choose by inputting ‘yes’ or ‘no’. If the seeker chooses to reject, the connection between the two will then be closed, but if the seeker inputs ‘yes’ the next command and control message is sent to the creator. The creator will be prompted a request to enter integers they want to be added. This will create the job for the seeker. After this stage data messages will be sent to both the creator and the seeker updating them on the task. After the task is completed, a data message will be sent to the creator with the result of the task. Finally, the seeker will be given a similar ‘yes’ or ‘no’ request asking if they will like another job. Our application protocol has strict communication rules resulting if a user makes an input error the application will send error messages and close the connection. An example of this would be if a seeker inputs an integer instead of ‘yes’ or ‘no’, an error message “I/O error: (the reason of the error)” will be displayed to them. These strict communication rules are to guarantee the completion of the application.

Below is a rough sequence diagram with a visual representation of a walkthrough of our application. It does not include the use case of the job creator ending the connection as that can be done at anytime they please.


Job-Seeker
Job-Creator
alt
Prompt to accept/reject job
Rejects
Connection Closed
Prompt to accept/reject job
Accepts
Prompt for Input for Job
Send job with inputs to be completed
Display Received Job, Complete task
Send Result back to Creator
Prompt if Seeker would like another Job
Prompt to accept/reject job
Rejects
Connection Closed
Prompt to accept/reject job
Accepts
Prompt for Input for Job
Send job with inputs to be completed
Display Received Job, Complete task
Send Result back to Creator
Prompt if Seeker would like another Job
Prompt to accept/reject job
Accepts
Process Start Over Again