ASSIGNMENT 1 PART 2

Software Engineering Methods

Group 11b

Alexandar Andonov

Bart Coster

Kasper van Duijne

Ariel Potolski Eilat

Alan Kuźnicki

Vladimir Pavlov

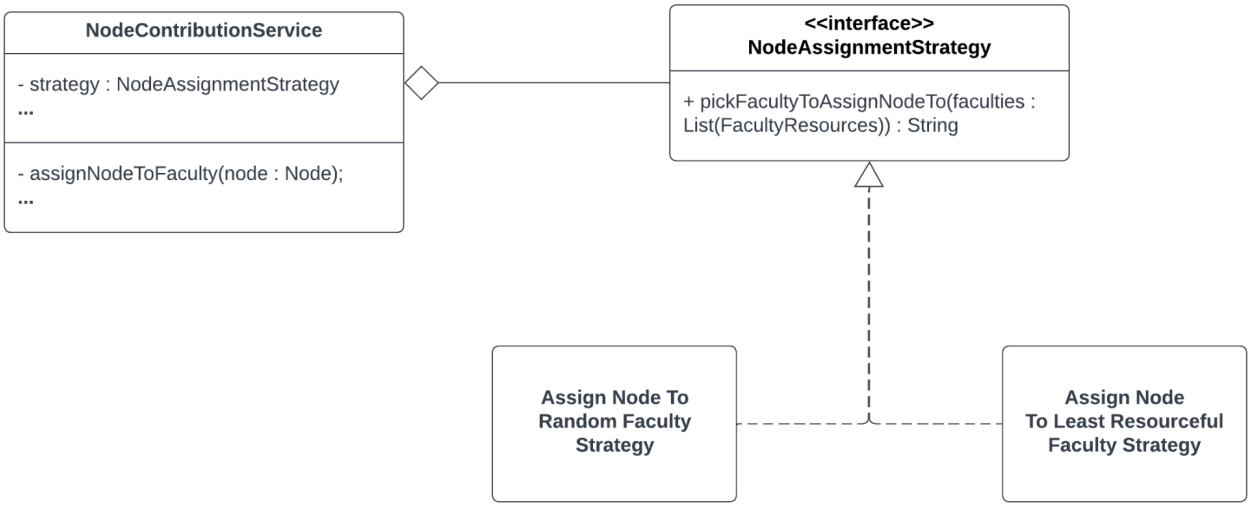
1. Strategy

1.1. Node Assignment Strategy

Why? Together with job scheduling, nodes are the core part of the cluster service, since it is through nodes that the cluster gets the resources needed to perform the jobs scheduled. Each node contributes a certain amount of gpu, cpu and memory resources, where such amount is determined by the user contributing that node to the cluster. However, given that the system has many faculties and each of them have access to a private number of resources, we need to assign nodes contributed to the cluster to a specific faculty. In order to do this division, we implemented two node assignment strategies that assign the node to a faculty; namely, the random faculty strategy, and the least resourceful faculty strategy. In the first, as the name suggests, the cluster randomly picks a faculty to assign the node to; by using the latter, the cluster assigns nodes to the faculty that currently possess the least amount of resources. It is important to note that, when contributing a node to the cluster, the user can specify the faculty that they want to contribute to. In that case, no strategy will be used and the node will be assigned to the chosen faculty.

How? The strategy is used by the NodeAssignmentService to assign an upcoming node without a faculty specified, to one of the faculties recognised by the cluster. We followed the textbook approach of having the service having the interface strategy as a dependency, with several (two in this case) implementations differing in behavior. We provided the service with the ability to change the strategy being used, and set the default to be the random assignment strategy. The reason why we chose the two strategies mentioned earlier is that we believe to insert balance (least resourceful strategy) to the system, but we do not discriminate between faculties (random assignment strategy).

Diagram.



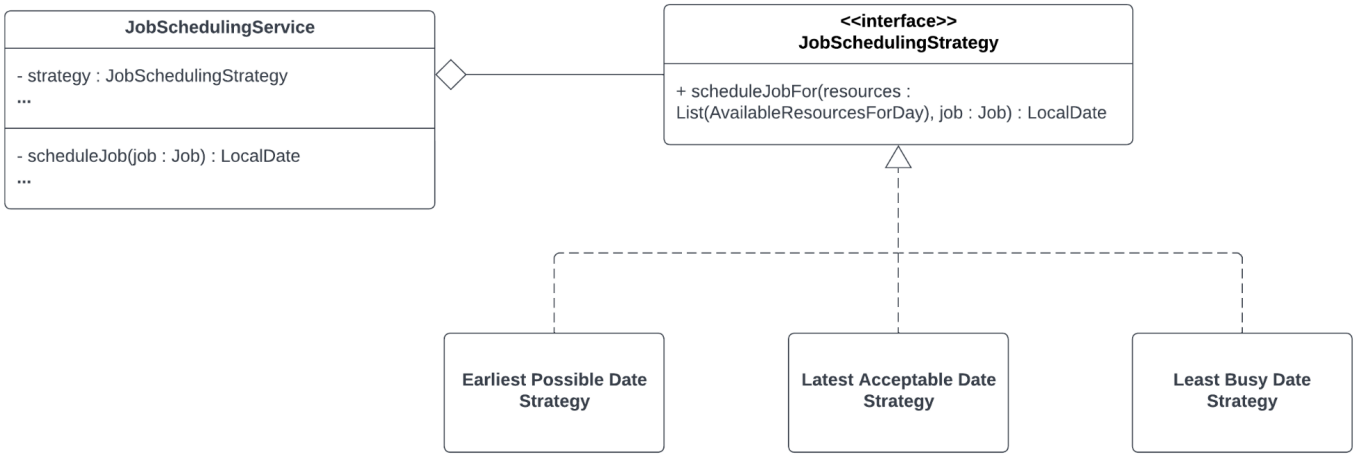
Implementation. Our strategies have been grouped together in the “strategies” package in “domain” in the root folder of the cluster microservice. We chose to do so to have clear separation from the rest of the codebase and easy access to the strategies when necessary.

1.2. Job Scheduling Strategy

Why? Accepting and completing jobs is the absolute core of the cluster service in particular, and the Delft Blue project in general. The cluster does not have unlimited resources, though; it is bounded by the capacity of its nodes. Because of that, requested jobs have to be scheduled; sometimes within desired timeframes, sometimes outside of them. For scheduling, we decided that it should be the cluster itself making the decisions instead of any specific user, so as to avoid potential issues with favoritism. We independently came up with several ideas to “optimize” the cluster: a greedy approach of scheduling a job as early as possible considering the available resources, a “safer” way of trying to schedule them as late as possible, bounded by the preferred completion date (so that the requester is not unsatisfied,) and finally, of putting a job on the least busy day within the preferred completion timeframe. We were quick to recognize the strong and weak sides of all of these approaches, and therefore chose to apply the strategy design pattern for scheduling with the three aforementioned implementations. Thanks to that, the cluster can be set up to suit the needs of different clients and situations.

How? The strategy is used by the JobSchedulingService to produce a date on which a given job is to be scheduled and save that in the relevant repository. Again, we followed the textbook approach of having the service having the interface strategy as a dependency, with several (three in this case) implementations differing in behavior. We provided the service with the ability to change the strategy, to ensure that it is always best suited to the users’ needs. We implemented the strategy by developing three algorithms performing the task of selecting a date to schedule a job on, returning either the earliest possible, the latest acceptable, or the least busy date within the desired timeframe. It is important to note that if the last two strategies fail to produce a date within the “acceptable” time period, they default to the earliest possible to minimize dissatisfaction due to lateness. The default strategy used by the cluster is the Least Busy Date.

Diagram.



Implementation. Our strategies have been grouped together in the “strategies” package in “domain” in the root folder of the cluster microservice.

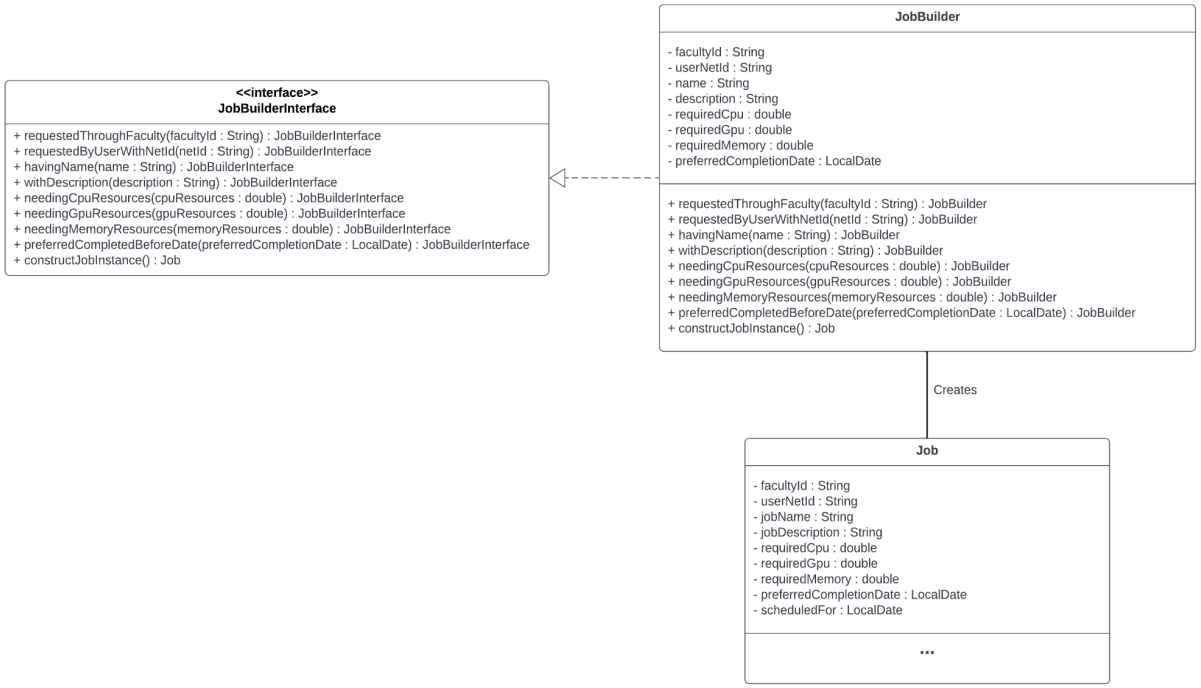
2. Builder

2.1. JobBuilder

Why? A builder is meant to separate the construction of a complex object from its representation. We decided that Job was a class complicated enough that it would benefit from splitting its construction process into several steps which we have names using the ubiquitous language principle. Thanks to this, we believe that the construction of a Job is now more easily understandable. Furthermore, Jobs varies significantly from one another; the usage of Builder allows us to exercise greater control over their creation and makes it possible to vary Jobs' internal representation.

How? We chose to omit the director, as we decided that it introduces an unnecessary level of abstraction and makes the code too convoluted for negligible benefit. With scalability and expandability in mind, we chose to include an interface for the job builder as well as one implementation. This is because we decided that, if the system were to be expanded, other types of Jobs might be introduced. Additionally, fields could be changed to become more complex objects, which could mean that different Builders are necessary for different combinations of components. Unlike the lecture slides, we chose to have all our builder methods return the builder itself, to allow us to chain method names one after another when creating a Job with the Builder (e.g., builder.set().unset().reset()...).

Diagram.



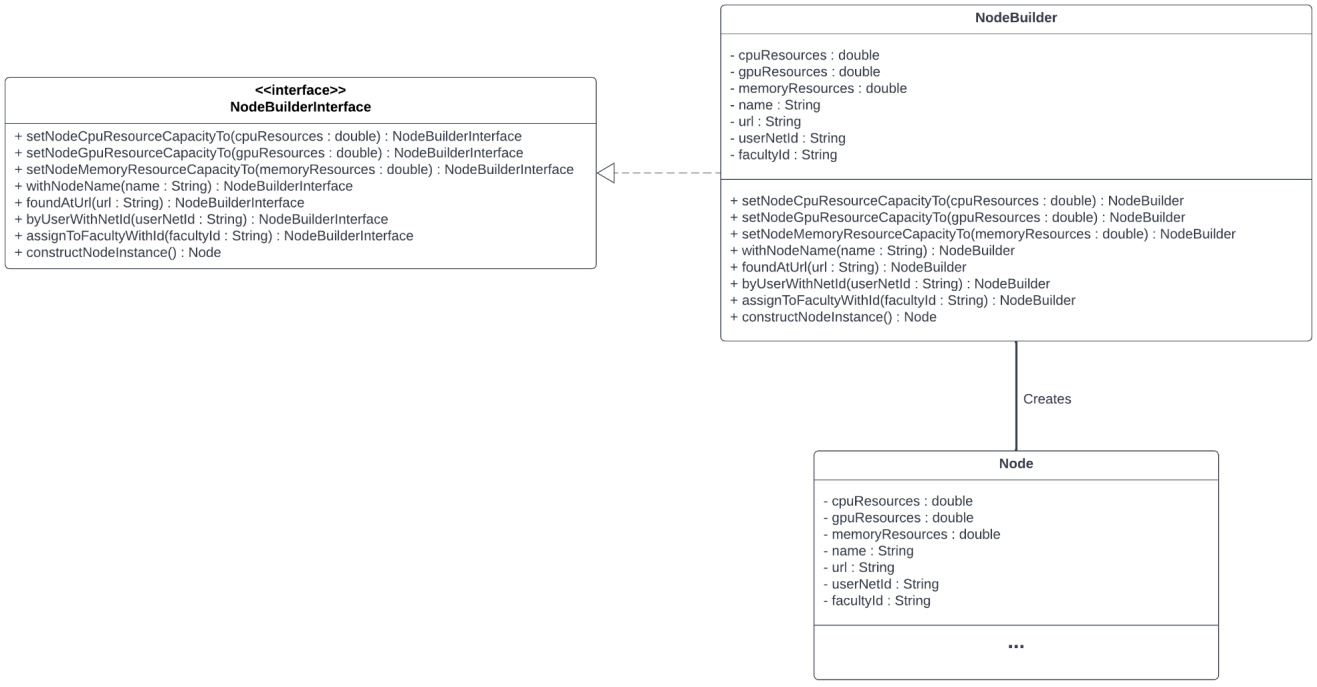
Implementation. We put both the interface and the builder in the "builders" package in "domain" in the root file of the cluster microservice. We chose to do that to have clearer separation between classes of different purposes and to be able to quickly locate these important parts of our code.

2.2. NodeBuilder

Why? Nodes are fairly complex objects already in our project, and, keeping in mind the extensibility and scalability aspect, could quickly balloon into very elaborate and mutually distinct entities, which would benefit from a compartmentalized, Builder-based mode of construction. Furthermore, depending on the scenario, different fields of a node are provided to the constructor - for example, a facultyId can be immediately provided or not. Finally, we believe that methods named using ubiquitous language and clearly explaining what they contribute to the forming node are much more clear and understandable than a Node with 10 words in parentheses. Altogether, it was clear to us that we needed a Builder for nodes.

How? We again did not include the director for reasons of nil benefit and pointless cluttering of the already extensive codebase. Furthermore, as with Jobs, we decided that it is very possible that a potential extension of the system would make individual nodes more distinct from each other. Maybe a node for each resource type will become the norm? With that in mind, we created an interface builder, along with one concrete implementation. Finally, we once more opted to have builder methods return the builder itself, to allow for chaining of method calls, creating easily understandable and natural language-like descriptions of the entire process of creating a Node object.

Diagram.



Implementation. This builder and its interface can be found in the “builders” package of “domain” of the source root in the cluster microservice. It is important to note that, like with the previous builder, the actual constructed class (Node/Job) is defined in “cluster” in “domain”.