Summary

I am doing a research project in collaboration with Total Energies. I am in the process of modelling their scheduling process in a way that will allow us to optimize their operations. My main concerns in this process is how to make the project succeed with the numerous decision that I have to make but which I feel poorly equiped to make. My hope is that an external stay at Decision Brain will help me understand what the best appraoch are to making my project succeed. This includes both the Ph.D. and potentially after my graduation. Below is a list of my main issues so far that I have failed to solve by myself.

Main Issues:

- Developing the API without a frontend to get feedback
  - I is hard to get the Stakeholders that are actually doing the work to understand the idea without developing a frontend.
- Many topics are outside the scope of my research project due to a lack of the essential skills.
  - Stakeholder management, and UX development.
- Gauging the financial vaule of the project; whether I should continue implementing or keep it academic.
  - I think that the team at Decision Brain will be able to gauge this in less than a month.

Goals of the External Stay

For me the three most significant goals of the external stay would be:

- Gauge whether my Ph.D. project can be implemented in practice.
- Integrate my application into a test environment at Decision Brain.
- Get competent feedback on my scheduling approach.

I have a somewhat naive belief that I have found a solid scalable approach to modelling a generic maintenance scheduling system (see section below). I am modelling something that is similar to what is described on **?** which is a source that has a more practical orientation than most academic works.

My code work on backend SAP tables and user-inputs so I believe that there may be a possibility of integrating my code into a system at Decision Brain if this is deemed of value. Integrating the system at Decision Brain would be ideal as I think that it would enable us judge the potential financial value of a full implementation of my project. It would also make it clear if it is a project that is worth chasing further by Decision Brain.

Setup of the External Stay

My initial idea of a setup with Decision Brain, would be that I get a stable contact for the duration of the external stay with which I can discuss my ideas and that I can rely on for help.

- First month: Determine if I can integrate my application in a relevant project at Decision Brain.

- Second month: Work on implementing the scheduling system in Decision Brain's IT infrastructure.
- Third month: Assess the feasibility of the project and possible course corrections.

Roadmap: Technical Parts

- [x] Model the Scheduler stakeholder
- [x] Model the Supervisor stakeholder
- [x] Model the Technician stakeholder
- [x] Determine a software architecture
- [x] Host the API on Total Energies servers
- [x] Read data from SAP
- [ ] Write data directly to SAP
- [x] Test output with scheduler stakeholder
- [ ] Test output with supervisor stakeholder
- [ ] Test output with technicial stakeholder

Personal

Personality Test

Extroversion ——————— Introversion Extroversion ——————— Introversion

Technical

I will provide a high-level overview of what it is that the current application is doing. I believe that this will make it clear how Decision Brain can help to make the project succeed.

Architecture of the Scheduling System

I have spend a significant part of my Ph.D. program refining and testing different architectures to enable meta-heuristics to coordinator state in real-time. The latest version is shown below.

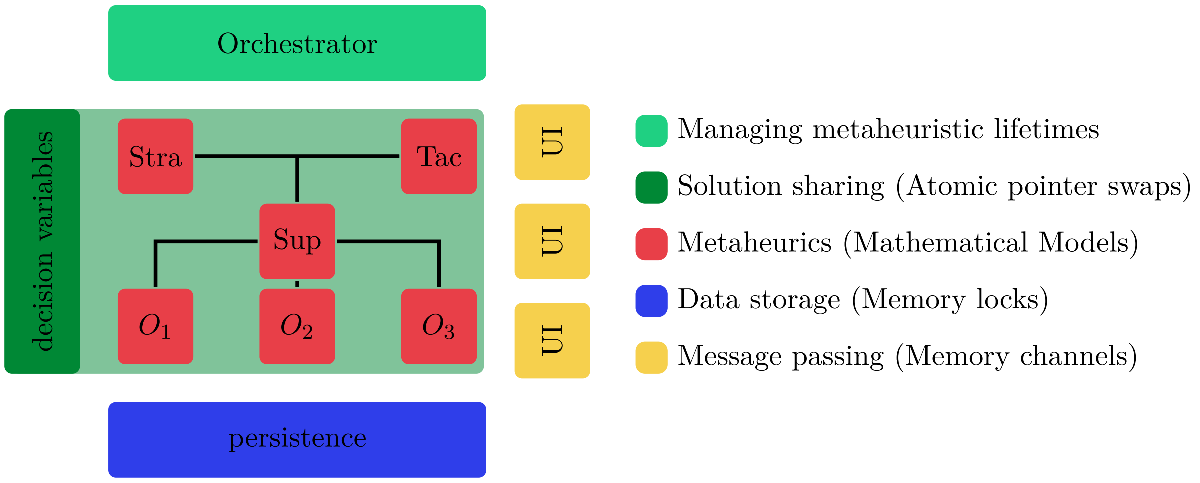

Orchestrator
decision variables
Stra
Tac
Sup
O_1
O_2
O_3
UI
UI
UI
persistence
Managing metaheuristic lifetimes
Solution sharing (Atomic pointer swaps)
Metaheurics (Mathematical Models)
Data storage (Memory locks)
Message passing (Memory channels)

Figure 1: High level overview of the architecture of the scheduling system. Persistance holds all data whether from SAP, user-input, or other systems; the Orchestrator is the part of the system that manages the lifetimes of the metaheuristics that does the actual optimization; the decision variable are all store together are are shared among all optimization meta-heuristics, each algorithm can write to its own state but only read the state of its neighbors; the UI components each communicate with the algorithms that correspond to the individual stakeholder

Key Lessons:

- Message passing between metaheuristics are unworkable, e.g. a microservice architecture is bad approach
- Optimization problems are difficult due to large and complex solution spaces. Allowing models/meta-heuristics to use each others solutions as parameters allows you to keep solution spaces small while preserving the ability to model the system.
- The operational setting is much more complex than you think and changes faster than think. Developing large integrated models is a nightmare as changes become more and more difficult the larger a model gets.

The key feature that this architecture enables it that we can move away from hierarchical approachs and instead model each stakeholder individually with the responsibilities that exactly that person is responsible for.

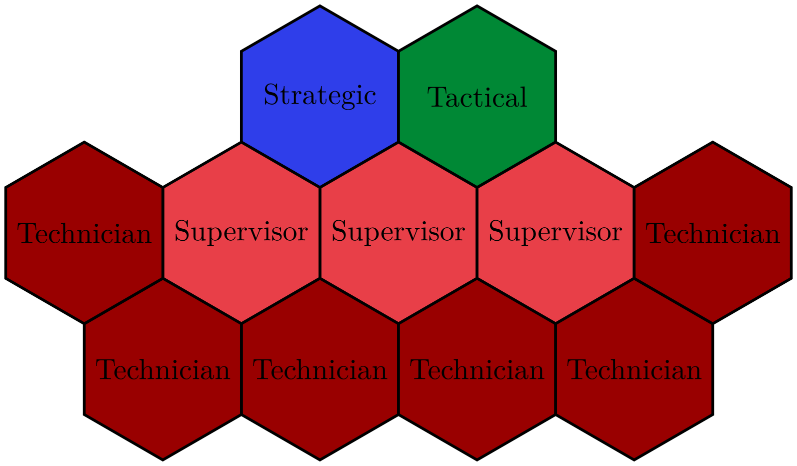
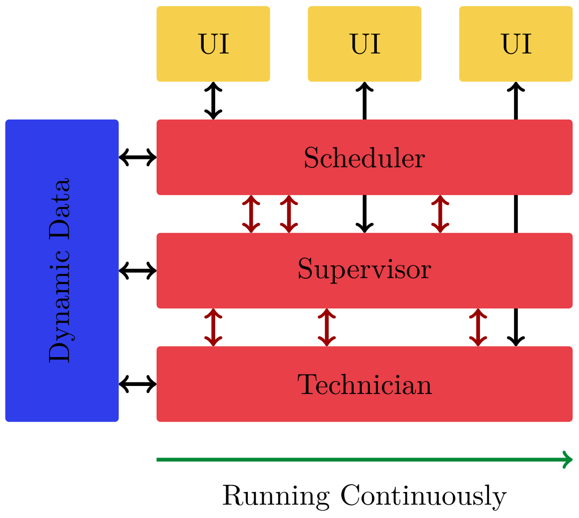


Figure 2: Each meta-heuristic (Actor-based Large Neighborhood Search, see algorithm 1 on page 5) is here shown as a hexagon. Each meta-heuristic is based on a mathematical model each of which are found in the appendix. **Notice**: this system is not hierarchical each metaheuristics reads the solutions of the other metaheuristic but they are not dependent on them for their function.

Key Lessons:

-
-
-

Pertually Running Optimization



Pseudo Code

Academic

Algorithm 1 Actor-based Large Neighborhood Search

```
1: Input $Q$ = message queue
2: Input $P$ = problem instance
3: Input $X$ = initial schedule
4: Input $S$ = SharedSolution
5: repeat
6:     $X^t = clone(X)$
7:     while $Q.has\_message()$ do
8:         $P.update(S, m)$
9:         $X^t.destruct(S, m)$
10:    end while
11:    $X^t.repair(S)$
12:    if accept$(X^t, X)$ then
13:        $X.update(X^t)$
14:    end if
15:    if $c(X^t) < c(X)$ then
16:        $X.update(X^t)$
17:        $S.atomic\_pointer\_swap(X)$
18:    end if
19:    $Q.push(m)$
20: until
```