Prediction Module Pitch

Interface Functionality

Vasken Dermardiros

March 14, 2022

Problem

> The raw idea, a use case, or something we've seen that motivates us to work on this

This time around, the problem is to put the meat on the solution rather than building something from the ground up.

Background

Unlike ControlKit 1.0, ControlKit 2.0 is based on FastAPI and running the "main loop" is actually a single call.

Figure 1: FastAPI Endpoints

How does it work? Quite simply, a call is made to the ControlKit endpoint which runs a few sequential processes. It starts by getting and compiling the data, passes it to the ControlModule (algos), get back a sequence of controls, maybe passes that to the PredictionModule, gets predictions, passes it to the SafetyModule which will confirm if all is good and then pushes this to the proposed controls database (we may not write all the commands all the time). The process reads specific settings from the ConfigModule along the way as required.

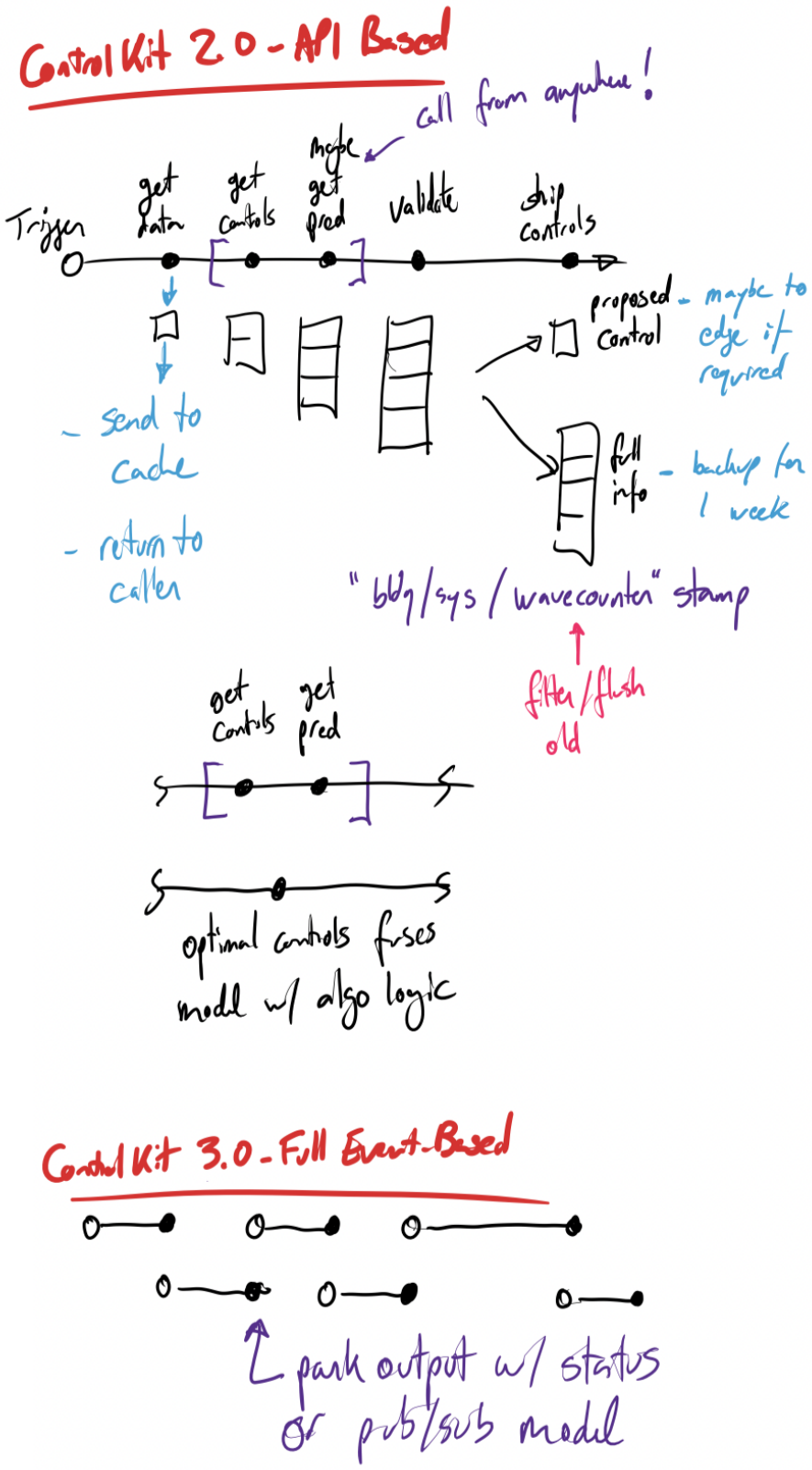

Figure 2: ControlKit 2.0 and 3.0

The difference between ControlKit 2.0 and 3.0 is that 3.0 is event-based where each phase or action is completely independant of the former. Each time an action is complete, the action would broadcast this state where some subscribed module(s) may pick it up from there. This is much trickier to orchestrate and we don't have the infrastructure now to do things like this. Would require a much tighter integration with DataStreams and full Kubernetes (k8s) deployment.

Lastly, so what's the trigger? It is a script that checks the `extraction_cycles` table for new data and once satisfied makes the call. Down the line, we may have the edge device itself make this call when it sees that its `tape` is either running out or the predictions are diverging. The `tape` is the two-or-more-hours of predictions and controls that will be sent to the edge device which will then write to the building one after the other. Alternately, the `tape` may be a decision tree that is updated regularly.

This API and triggered approach solves the data synchonizing issue and allows to have different versions or setups of ControlKit running for a given building in parallel. It allows to run a staging version of ControlKit in "virtual mode" which could be swapped into "control mode" for a short while. This simplifies our rollout strategy. We may also have a certain configuration used for special cases such as demand response or have specific algos running, and so on. This would reduce the need to develop a multi-control module.

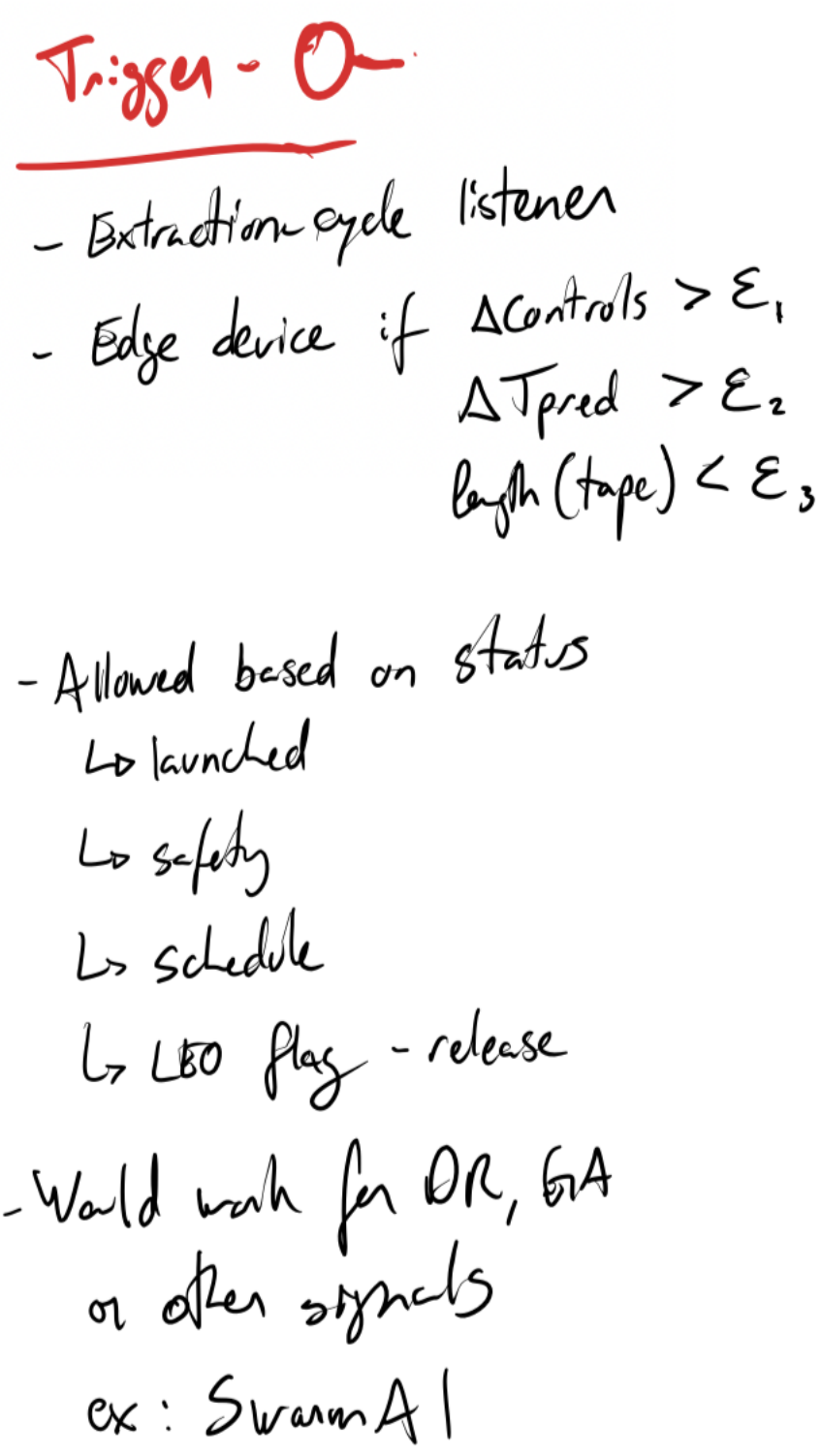

Figure 3: Trigger to call ControlKit endpoint

Main Problem

Which now brings us to the main problem in this pitch: an algo walks into a bar and wants a drink. The cocktail is a prediction. And unlike how we were doing things before, the algo is the first-class citizen here. The algo decides what model it needs (in terms of XUWO points) and the model will reflect the algo. Basically, if a zone goes missing, both the algo and model will be released for that zone or floor **but nothing more than that.** We shouldn't release a whole building if the janitor's closet's temperature sensor in the second basement is broken.

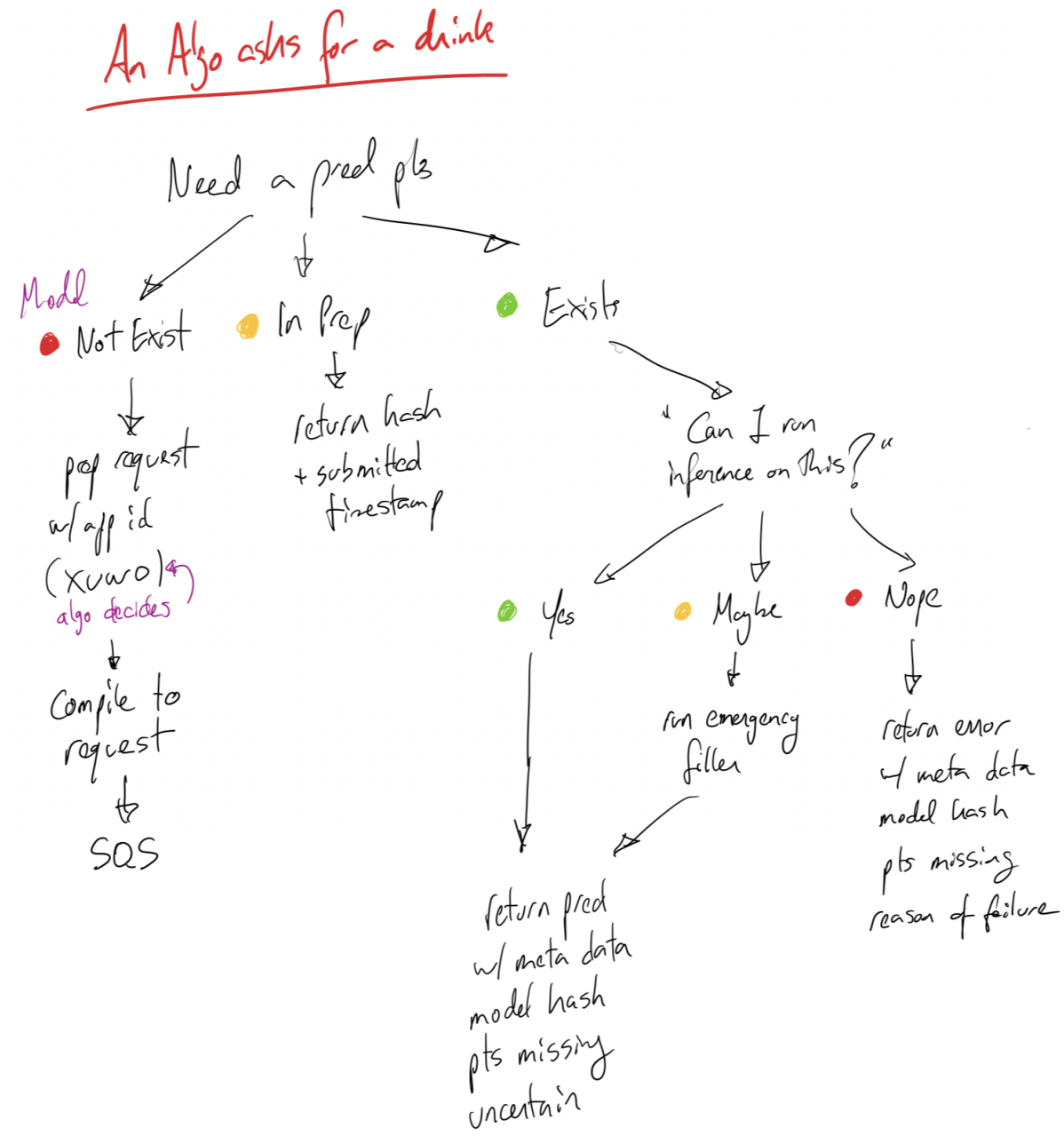

Figure 4: An algo walks into a bar

To obtain a prediction, a few logic steps have to be followed. On one side is how to make the calls and bring the data to the model. On the other is how to manage the "bar" to have the models ready. And these are the two pitches for this build cycle.

This pitch covers the former.

Clarification on Submodels

Since the dawn of time, we've been building a model per building. The whole building. The `ai_config` does include the building hierarchy such that we could break the large model into smaller submodels based on levels, data quality, etc. using the `phd_submodel_generator()` function inside `DataKit.parsers`.

The days of taking the initiative are over. Ok. At least on this topic.

For ControlKit 2.0, the design is around having a parallel instance per each system / application running. This system will start by being the whole building, but as the algos are recoded, the algos will be deployed on a system-level and not a building-level. Being deployed on a system-level, the models that will serve these algos should follow the same structure and contain the same points in them. So, we will have a model per system / application. The extraction list may also follow this logic too, so that everything is broken up based on system / application. Of course, we will have multiple of these per building.

An algo configured for a system / application will know the points associated to itself and can pass it on in its model creation request.

Appetite

> How much time we want to spend and how that constrains the solution

Time budget: six weeks starting March 14, 2022.

Solution is to build the interface to ease interacting with ClearML and to integrate with ControlKit 2.0. The back-end will be built by the parallel pitch while this one allows to get that information out.

The solution will heavy rely on the FastAPI Helper and FastAPI Tools developed by Maroun for the DataModule and ConfigModule. All containerization and documentation will follow this template. This is our Standard Interface now and will be extended to other teams.

For logging, we would need to standardize this as well and push it to a centralized location; need to see with Marc-Olivier, Stefan, Farzam and/or Saeid. All other config and settings will go in the database too; we can ask to make tables if the info we need isn't available anywhere else. This effort is also a company-wide concern and we should allow other teams to work on it too.

Solution

> The core elements we came up with, presented in a form that's easy for people to immediately understand

The solution contains four parts: (1) drafting all the API endpoints in FastAPI, (2) assuring that this interface can run by itself since there could be cases that we can have models outside ClearML, (3) the raw inference data itself can either be passed through a call or the location of this data can be passed, (4) allow for logging and monitoring useage.

Model Requests from Anywhere!

The submittal of a new model request will be from an API endpoint. Once presented a preliminary request, the back-end will validate the request, check the data quality (if available) and then either (a) reject returning an error with an explanation or (b) accept and place the request in the SQS queue and returning to the client a successful status.

What's in the call: `bldg_id`, `app_id`, `settings`, `[optional] pipeline to use`. `settings` is the `XUWO` list or other argument as per the pipeline requirements.

NOTE: what is the minimal info that can be passed? Can it be just the `bldg_id` and that's it?

API Calls Desired

The purpose here is to expose or forward these. To get the numbers might be part of the parallel Model Management Pitch.

- given bldg_id or bldg_id/app_id:
  - given data, prediction
  - model status, all
  - model status, current published
  - model metrics, all
  - model metrics, current published
  - models available: return model hashes + meta data
  - associated points: XUWO lists that goes into the model
  - request JSON
  - ability to archive a model
  - ability to archive an application / equipment
  - ability to archive a building
  - link to data report in s3bucket
  - KPIs: number of calls, response time in ms, number of failed calls

- given model_hash:
  - model status
  - model metrics
  - model meta data
  - associated points: XUWO lists that goes into the model
  - ability to archive a model
  - model weights location in s3bucket
  - link to model report in s3bucket
  - TODO link to ClearML if there's something worthwhile to show
- generally
  - info on SQS queue
  - info on the health of ClearML: queue status, number of workers, uptime, disk usage and other metrics worth monitoring -> will assure these metrics are monitored in Icinga
  - inspirational: number of active projects, number of total projects, number of tasks, number of models, number of buildings

Services

We are starting to have a lot of models and would need to tidy things up.

Model Monitor Service relies on Task 4 and consumes the results of the daily evaluations. In this framework, the PredictionModule is responsible for model management and will need to train models anew as performance diverges and/or if points go missing or return. Once a new model is trained for a given request or application, it becomes the active one. What would be *nice-to-have* is to monitor the model in real time and warn if a zone's prediction is way off. What we have in place now is to do this on a daily fashion.

Archiver Service will archive this superceded model and will eventually be deleted. For the time being, we will monitor which model is used and can think of ways to retire models that are not used (note that some models, although seldom used, must remain active since they may serve for special purposes, eg. demand response).

The services can utilize the triggers already in ClearML instead of running an infinite loop with a sleep.

Model Meta Data

Generally speaking, these are the model properties:

- underlying recipe / pipeline

- request submitted date
- model created date
- model initial train date
- model retrain train date
- request submitted by
- very high-level metrics, one number
- very high-level data quality metrics, one number

Rabbit Holes

> Details about the solution worth calling out to avoid problems

Oh jeez. We won't know all the endpoints we would need for every case. It's good to start with these and see as we go.

User management can be handled by an AWS gateway or something similar. Don't work on user or group management. Assume anyone and everyone has access to all the endpoints.

No-Gos

> Anything specifically excluded from the concept: functionality or use cases we intentionally aren't covering to fit the appetite or make the problem tractable

Don't reinvent the ClearML API endpoints. See if there's something already in their documentation.

NOTE Project is quite large as-is. Need to constrain it more.

Notes that helped shape this pitch

> These are notes and not deliverables!

```
# Dewarkar
+ We don't know which points are missing
  + In time range: 120 days and 7 days
  + Missing: not in database but in configuration
  + Heatmap
+ We don't choose a certain system

"With this config you've given us -> this is the clean one" do you want
    to proceed
```

```
9
10 Always better to have the option to do it manually. Automation is nice,
       but we want people to become comfortable.
11
12 # Summit View
13 Can we have Mission Control tabs inside Summit View? Have only the ones
       we need.
14
15 ConfigModule needed!
16
17 Also for BuildingKitConfig
18
19
20 # David
21 Sort of needs for v1.5 or maybe before..
22
23 Control Kit General:
24 - Need to have monitoring of the service directly on Icinga. Saeid
25 - Able to schedule downtime (feature in Asana).
26 - Able to recover from Data Gap if less then X hours.
27 - Not completely crash when data is missing.
28 - Not stop the control module if data is missing and LEO is in safe
      mode.
29 - Stop the auto-restart. David
30 - Push to alert hub table with accurate issues. Table in
      monitoring_team; report as link to a file put in s3bucket **standard
       way of doing this to replace `report_issue()`**
31
32 Building & Data Module:
33 - Add water side algorithms into Control Kit. David -> extraction cycle
       != 5 minutes but flexible
34 - Create a getAIConfig function for Water Side algorithms. David
35 - Query 7 days of data only when needed at the cost of needing to query
       it when required or have the option to do it. -> **causing memory
      issues**
36 - Getting rid of load part in the beginning.
37
38 Control Module:
39 - Need to fix false alarms coming from Control Module being stopped and
       algorithms being on.
40 - Add Water Side Algorithms. David
41 - Flexible 1 minute to 5 minutes extraction
42 - Re-vamp Control module to have it in one execution instead of two.
43
44 Time Module:
45 - Remove the time module and time dependencies. -> trigger to come from
       new data
46
47 Prediction Module:
48 - Generate alarm & send alarm when “qa” to “prod” fails.
```