Capr Design Guide and Roadmap

Contents


```
library(Capr)
```

1 Executive Summary

Capr, pronounced 'kay-pr' like the edible flower, is an R package that serves as a programmatic interface to defining cohort definitions for data standardized to the OMOP CDM. Capr allows for templating cohort logic, where a user can shuffle through concept sets to initiate variations of a cohort definition. Cohort templating is a critical value add that Capr provides to OHDSI software. The intent of this software is to make the process of defining cohorts less of a clerical task (via point and click) and allow for OHDSI studies to be designed in a single development environment. The value of developing Capr for Odysseus is that it integrates cohort development into the development pipeline, enables templating which can be leveraged in a phenotyping framework.

The mechanics of Capr are based on emulating the representation of a cohort definition as specified by circe-be. Circe-be is the underlying mechanism that builds sql based on input specification defined by a json structure that fills in a java class. Capr can be used now be used without a connection to a CDM, however the user must have knowledge of the OMOP CDM providing concept ids as inputs. Capr returns an R object that can be converted to a json string. Commonly, cohort definitions are distributed within the OHDSI community as json files, therefore it is a natural endpoint for Capr, allowing circe to serialize to ohdsiSql. The functionality of Capr includes: creation of a concept set, specifying the cohort logic, building a cohort definition and coercing it to a json representation. Future functionality will provide print and validity methods for class, importing Capr from json and saving/loading cohort sub-components to disk. Capr also has room to grow in terms of providing functionality to describe a cohort definition both by graph and text, modifying cohort definitions by addition of Capr objects and enumerating sub-components.

Currently, Capr is preparing for a release of v2.0.0 to the OHDSI community github as a HADES R package. The release date is anticipated for early spring 2023. There is a plan to incorporate complimentary changes

to v2.0.0 by late summer 2023 in addition to suggestions based on user feedback.

2 Introduction

2.1 Purpose

Capr is a programmatic interface to defining cohort definition for the OMOP CDM. Capr builds a representation of a cohort through R that can be serialized to json. The json structure can be imported into ATLAS. This structure follows that of circe-be, that object representation of cohort definitions in OHDSI. A valid circe-be json can be serialized into ohdsisql, a parameterized version of sql that builds cohort under the assumptions of the OMOP CDM structure. Capr fills a gap in OHDSI software in providing a programmatic tool for cohort construction as an alternative to ATLAS. A programmatic tool enables us to parameterize definitions. This has a few advantages:

1. *reduce clerical error* – minimize copy/paste activities in ATLAS by instead offering a cohort skeleton and iterating across concept sets.
2. *Focus on concept set development* – our ability to properly enumerate a cohort of patients ultimately relies on whether the correct concepts have been identified for a database. Many cohort definition skeletons are simple where the variation in the definition comes largely from the concepts. Capr allows us to expediate the definition process and focus on identifying the rights concepts.
3. *Provide alternative entry points to OHDSI analysis* – ATLAS installation is not guaranteed in a study site with OMOP data. Further some researchers do not like to use a GUI like ATLAS and prefer to use a programming language like R. Capr builds an equivalent cohort definition to ATLAS meaning there is an alternative entry point to an OHDSI study. Capr allows a researcher to remain in a single interface to complete their entire analysis: from cohort definition to diagnostics to analysis. It is however, suggested the Capr be used in tandem with ATLAS for clinical review.

2.2 User-base

A Capr user is someone who has knowledge of OHDSI and the OMOP CDM and is comfortable programming in R. Capr is intended to be used by data scientists to help create a scripted representation of a cohort definition that has been defined and reviewed by a clinical team. Defining cohorts is a team effort, however Capr gives programmatically inclined users a tool to produce cohort definitions for their clinical team.

3 Roadmap

3.1 Goals

The goal of this software is to provide a programmatic alternative to ATLAS for defining cohort definitions. A programmatic interface provides many advantages to users trying to handle complex studies. Sub-goals are listed as follows:

- **Templating**: Through a programmatic interface, users can create cohort templates. Templates are representations of the cohort logic where the input parameters are concept sets. Templates serve as a solution for minimizing clerical errors when users are trying to create several definitions for a study.
- **Organization**: An issue when running OHDSI studies is organization. There are several components that need to be managed over the course of the study. This becomes even more complicated when using multiple platforms to achieve an analysis (i.e. ATLAS and R). A goal of Capr is provide a tool to allow OHDSI studies to be developed on a single interface.

3.2 Development Timelines

3.2.1 Anticipated release cycle

Capr v2 will be gradually released in order to receive feedback from the OHDSI community and identify areas of improvement. The official v2 release on HADES is anticipated for early spring. Complementary to this release will be v2.1 which will contain additions that were not essential for the 2.0 release. This includes methods for the Capr objects including validity checks and print statements. We will also add functionality to import json into R as a Capr object.

Release	Date	Notes
v2 alpha	01/19/23	Proof of concept demo on HADES call
v2 beta	02/13/23	Soft release to specific developers for feedback
v2 HADES	03/15/23	HADES release of v2 on OHDSI github
v2 CRAN	03/17/23	Submission of Capr to CRAN
v2.0.x	04/23 - 7/23	Period of observation, do patches and bug fixes
v2.1	~08/23	Complimentary functions to v2 (show, import etc)

3.2.2 Feature Proposals

The following is a list of proposed future features for Capr that have been discussed during development. They are listed in order of priority.

3.2.2.1 1) Previewing the cohort definition There is a desire to preview a cohort definition both textually and pictographically to help the user understand the cohort definition in development. The cohort preview functions will yield: 1) a markdown text description of the cohort definition (similar to CirceR) and 2) a diagram of the definition as a png. The preview functions will be rendered using the proposed signatures: `jot` for the text preview and `draw` for the graphical preview. This feature is intended to be used for protocol and report development since clients desire both a text and graphical representation of a cohort definition. The text preview will use simple markdown and the graphical preview will use the grid package to build a diagram.

3.2.2.2 2) Cohort Algebra Often times, users want to add aspects of two cohort definitions together. A future feature would be to construct cohort algebra where the addition of a cohort (or a component) would add the detail into the cohort definition. For example, if we wanted to add another set of rules to a cohort this would be expressed as:

```
newCohort <- cohort + inclusionRule
cohortAB <- cohortA + cohortB
```

Line 1 of code snippet 7 demonstrates how one would add the inclusion rule Capr object to the cohort definition. Line 2 of code snippet 7 demonstrates how a user would add the components of two cohorts together to build a new cohort. These functions would only impact the primary criteria, additional criteria and censoring criteria. The end strategy and cohort eras would be adopted from the left-hand side. The value add of this feature would be to simplify cohort modification during development. It has been a common suggested request from other users.

3.2.2.3 3) Enumeration of Sub-components When developing a cohort definition, we often need to know how much information is available in the database before adding it to our definition. A future feature for Capr would be a series of functions for each class that would run a query based on the specification on the class. This would be done through a CDMConnector connection. The proposed signature would be `inquire` which would retrieve the amount available from a component of the cohort definition. We could also use this as feasibility for building towards a ohdsiSql serialization directly from the Capr object. This would also provide feasibility in terms of building cohort attrition as a return of a Capr object. The value add of this feature would be to help users navigate the database during cohort construction.

3.3 Maintenance Plan

Maintenance of Capr will be done by Martin Lavallee and Adam Black. We plan to respond to bug fixes and feature requests via github issues. Between v2.0.0 release and v2.1.0 release, Capr maintenance team will source feature requests but will delay production implementation until v2.1 release on the main branch. New features will be add to the develop branch, which will remain stable, as a experimental release.

4 Design

4.1 A. Orientation

4.1.1 i. Entry Point

Ideally, Capr should be used in tandem with a connection to an OMOP vocabulary. Connection to an OMOP vocabulary can be done through DatabaseConnector or CDMConnector. This allows users to lookup concepts before applying them in Capr objects. Capr does not provide functionality to optimize concept lookup; this is a feature passed to other OHDSI software. Capr assumes the list of concepts as inputs.

4.1.2 ii. Exit Point

Capr stops upon development of the json structure, which is typically saved as a file in a directory. The previous version of Capr allowed for the user to create the ohdsiSql via CirceR. However, this was deemed an overlap between other OHDSI R packages (CirceR, CohortGenerator etc.) and it felt more appropriate for the Capr process to end at building the json file. Ultimately the item that is passed across the OHDSI network is the cohort definition json. The json file can be upload into ATLAS or into R where the ohdsiSql can be built using CirceR. Json is a straight input to build the SQL, so it was deemed the minimally sufficient output needed. We intend to draw distinct lines between OHDSI software in order to improve standard workflows, that are either site internal or for network studies.

4.2 B. Functionality

Capr provides the programmatic interface for constructing cohort definitions that maps to the circe-be logic. In order to maintain this correspondence, the functionality of Capr covers: building concept sets, composing and building the cohort definition, and coercing Capr to json. Future functionality will be added into the section once they are released.

4.2.1 i. Concept Set Builder

Before defining cohort logic, the user must specify concept sets. Capr provides functionality to build concept sets off standardized OMOP concept IDs. How the user finds these IDs is left to other tools. A concept set is built in Capr as follows:

```
cs(descendants(201826L), name = "T2D")
cs(201826L, 201254L, name = "Diabetes")
```

The first example in code snippet 1 provides the concept set and specifies that it includes all descendants. Capr supplies a command for the three types of concept mapping: include descendants, exclude, and include mapped. The second example in code snippet 1 provides builds a concept set based on a vector of concept IDs. The first argument of the cs command is an ellipse capturing a variety of input giving the user flexibility. The second argument gives the concept set a name, which in general should be supplied for readability.

When a concept set is created in Capr, it is tagged with a global unique identifier (GUID) using the R package digest. The purpose of this is to ensure that the id of concept set is unique to its construction. In ATLAS, the concept id attached to the concept set is a local id. This means that the id is an integer starting from 0 that identifies the concept set in the definition. This creates an issue if we were to parameterize the concept set, therefore the tag must be unique to the concept set. Since circe-be uses integers, Capr must resolve this issue on the backend, something described in the coercion section.

4.2.2 ii. Cohort Builder

Next, a user must be able to build a cohort definition in R, therefore Capr provides functionality to define a cohort definition and sub-components of the cohort definition. Capr provides a constructor for each circe-be element (i.e. query, count, group, windows). A cohort is defined using the function:

```
cohort(entry, attrition, exit, era)
```

The inputs for the cohort function are the four components for defining a cohort: entry which accrues the initial set of patients, attrition which defines the logic that filters the initial set of patients based on inclusion and exclusion criteria, exit which determines how the person exits the cohort and era which defines the span of time in which events are considered to have a cohort event. Each of these four elements are constructed based on circe-be subcomponents including:

1) *Query*: an object that defines which concepts to extract from a domain table in the CDM;
2) *Attribute*: an object that specifies a subpopulation of the query;
3) *Criteria*: an object that temporally enumerates the presence or absence of an event relative to a point in time; and
4) *Group*: an object that binds multiple criteria and sub-groups as a single expression.

4.2.2.1 1) Query A query is constructed via the name of the domain table in the OMOP CDM. For example:

```
visit(conceptSet, ...)
condition(concept, ...)
```

The conceptSet input takes on a a concept set built from the cs command. The dots argument is where the user can supply attributes that contextualize the query to a specific subpopulation.

4.2.2.2 2) Attribute An attribute has its own set of constructor functions that are variations on four types of attributes that may be used:

- **op**: a boolean operator matched with either a numeric, integer or date value
- **concept**: attributed built on an OMOP concept id
- **logic** an attribute defined by a `T/F` toggle
- **nested**: an attribute based on a Capr group object

An example of using an attribute within a query is shown in code snippet 4:

```
condition(cs(descendants(201826L), name = "T2D"), male())
condition(cs(descendants(201826L), name = "T2D"), age(gte(18)))
condition(cs(descendants(201826L), name = "T2D"), first())
```

The first example in code snippet 4 utilizes a concept attribute. The `male()` function automatically maps the concept to the male concept id. Other concept attribute functions will have mappers and helpers to assist the user in selecting the correct concept to add as an attribute. The second example of an attribute in code snippet 4 is for an op attribute. Any op attribute begins with the domain (i.e. age, startDate etc) and then takes an input of an op. An op is a function with the following signatures:

Operator	Name
gt	Greater than
gte	Greater than or equal to
lt	Less than
lte	Less than or equal to
eq	Equal to
bt	Between
nbt	Not between

Note that with the bt and nbt functions the input requires two values, a start and stop, specifying a bounded range of values. The last example in code snippet 4 is for a logic attribute. The logic attribute is simple in that if it is added to the query it will toggle TRUE for a the specification of the domain. In the example we omit one for a nested attribute because it is more complex and requires explanation of a criteria and group, which are stated in the upcoming sections. A full list of available attributes will be included in a future appendix.

4.2.2.3 3) Criteria A criteria is constructed first on an operator that specifies the number of times the criteria is observed, second the query of which the criteria is based on and finally an aperture (i.e. a time window) specifying the timing relative to the index event at which we are to observe the specified criteria. Code snippet 5 shows an example of a criteria call:

```
exactly(0,
        query = condition(cs(descendants(201826L), name = "T2D")),
        aperture = duringInterval(eventStarts(-Inf, -1))
        )
```

A note for the criteria is that its meaning is tethered to an event specified in another part of a cohort definition. This means that a criteria object applies either for cohort attrition (inclusion rules), additional criteria for the primary criteria or in a nested attribute where the index event is the query to which the attribute is attached. The aperture is what restricts the criteria to certain situations. An aperture is built using the `duringInterval` command, where the first option is the start of the window and second is the end of a window. The temporal deployment of the query makes this object complex.

4.2.2.4 4) Group The final sub-component binds multiple criteria and sub-groups in a single expression. In Capr, this is done using the term with, to which we attach the options all, any, at most or at least. This specifies for the candidate to enter the cohort they have to satisfy the specifications of the group, whether that is at least 1 criteria or all. In the example below we assume that the query and aperture have been specified elsewhere in a script.

```
withAny(
  atLeast(1, query = abLabHb, aperture = ap1),
  atLeast(1, query = abLabRan, aperture = ap1),
  atLeast(1, query = abLabFast, aperture = ap1)
)
```

In code snippet 6, we start the group by specifying what needs to be satisfied in order for the candidate to enter the cohort, in this case they must satisfy all the queries. If the group is based on `atLeast` or `atMost` then we must also specify an integer stating how many components need to be satisfied. Groups make complexity possible in a cohort definition where there are multiple factors that specify qualification of remaining in a cohort. Groups can also take on subgroups, for example we could say `withAll(withAny(..))`. The group is also the basis of the nested attribute. Specification that a group is for a nested attribute is as simple as adding `nestedWith` to the function signature.

4.2.3 iii. Coercion/Mapping

For OHDSI studies, the key representation of a cohort definition from a software perspective is via a json file. Thus, we need to coerce the Capr object into a json file so that it can be used within studies or exchanged with other researchers. Previously we had constructed a cohort definition via an R representation supported by Capr. Now we must turn this R object into a json file to save to disk. Coercion is a complex task because Capr must track two aspects of the definition: 1) the concept sets indicating the codes to search within the CDM and 2) the logic specifying how to deploy the codes on the CDM. The following explains how this process is achieved in an abstract sense.

Step 1: Collect and replace concept identifier

Recall that a concept set is initialized with a GUID from the digest R package. While this identifier is useful in Capr, in circe the id must be an integer starting from 0. This ID is an internal reference for the cohort definition. When coercing the Capr structure to a json we need to collect all the GUIDs in the definition, remove duplicates and enumerate each unique concept set for the internal reference. Capr does this in the background using the internal functions `collectGuid` and `replaceCodeSetId`. The first function builds the unique list and the second goes into each Capr object and replace the id slot with the new integer. After this process the object remains a Capr S4 cohort structure but the ids within each query have changed from GUID (character) to integer.

Step 2: Coerce concept set from S4 Capr to S4 list

Once the concept sets have been set within the query object, Capr will then go through the entire definition and extract just the concept set objects and turn them into an S3 list. The internal function `listConceptSets` completes this task, returning a list of all the concept sets used in the definition and removing duplicates.

Step 3: Coerce cohort logic from S4 Capr to S3 list

The cohort logic is now coerced separately from the concept sets. In the circe-be json the concept sets are a different slot the remainder of the definition. Capr handles this the same using the internal `as.list` to coerce the Capr S4 object to an S3 list. It binds the concept set list with the cohort logic list.

Step 4: Convert S3 list to json

Once the object is coerced into an S3 list it can be converted into json using the `jsonlite` R package. The function `jsonlite::toJSON` only works on an S3 list. Capr is designed in S4 because of the complexity of the system, making it easier to prevent mistakes by the user.

4.3 Object Representation

Capr uses S4 classes in order to define the underpinnings of the cohort definition. These S4 classes are closely related to the circe-be java classes that specify the structure of the cohort definition. The Capr classes assume a few differences in order to allow for parameterization of the cohort definition via templating and provided suggested organizational improvements to the structure. The decision to use S4 classes was made because of the need to have a strict interface to adhere to the circe-be class. S4 classes are well suited for very complex codebases, according to Advanced R. Capr's relation to circe-be has the sort of complexity, particularly for the coercion aspect of Capr where the same method is applied slightly differently to each class. Classes are an important aspect of Capr and S4 provides a good balance between functional and object-oriented programming.

5 Appendix