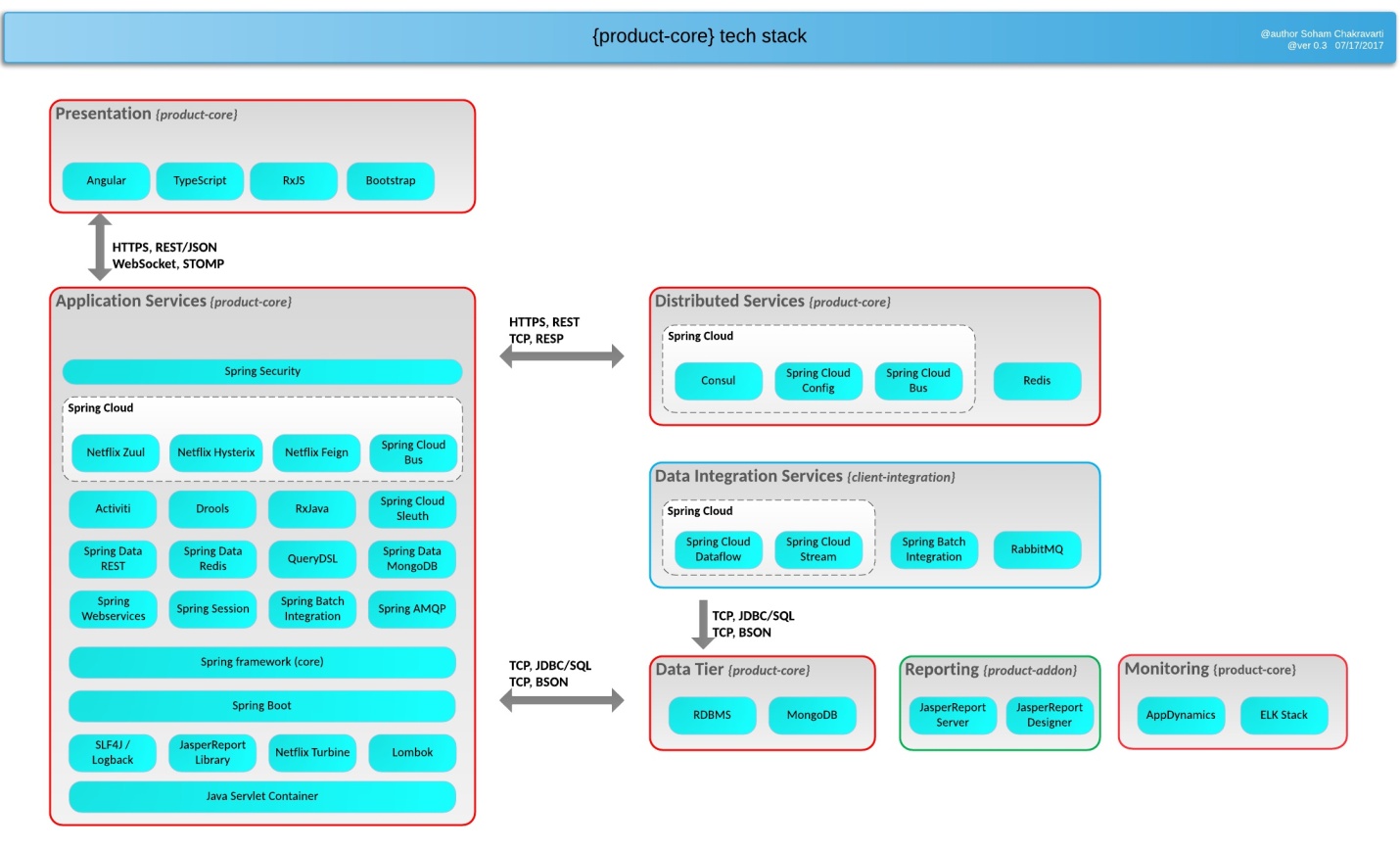
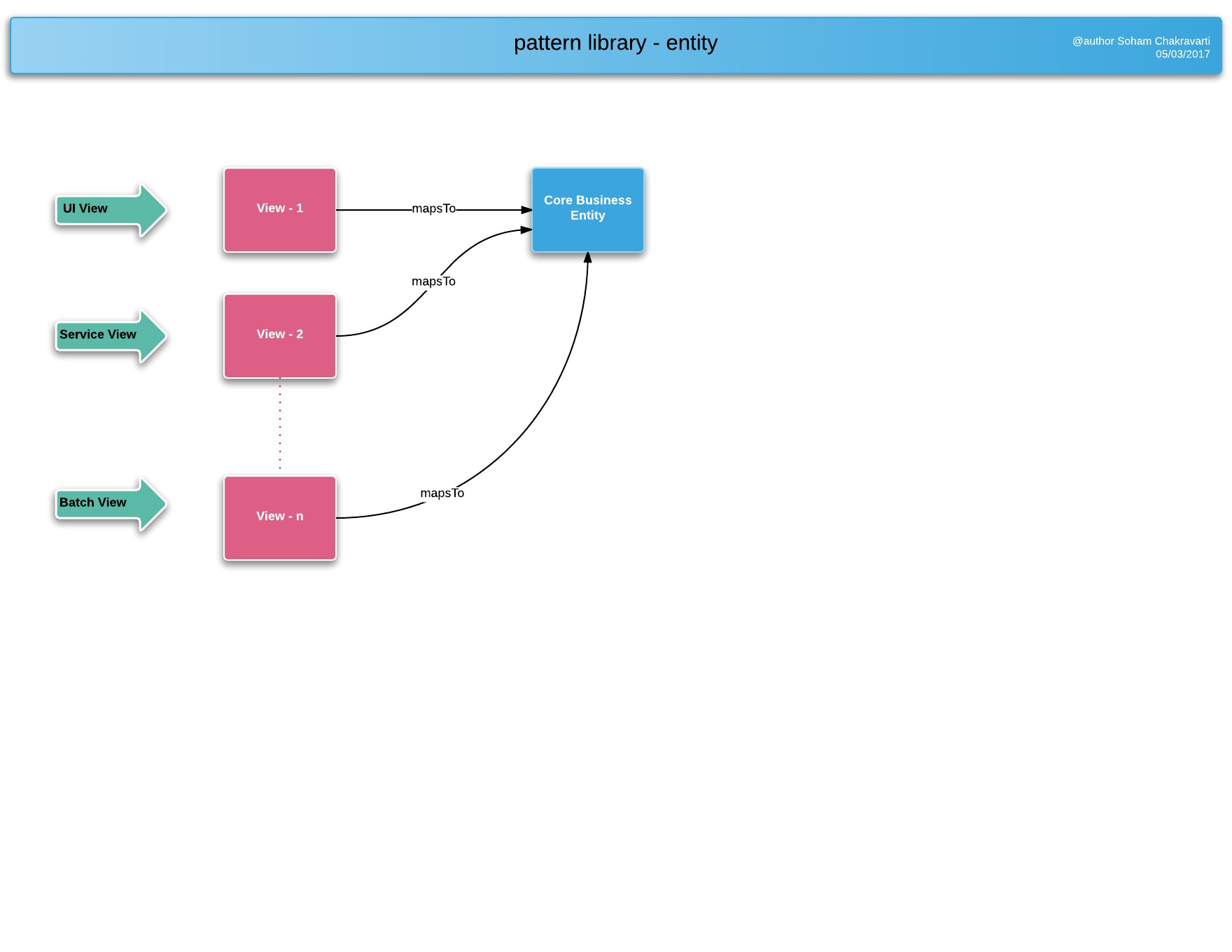
Config session 1: **Objective** Understand the underlying framework architecture and technology stack





/\*\*

\* View is a perspective of Core. It can be used in presentation layer or can be part of web service integration.

\* Relationship of View to Core is many-to-one. View is not mandated to have a core backing.

\* Within the platform, View is associated to an user, while Core is the same across users.

\* Authorization cross-cutting component ensures that access to Core by an user is valid.

\*

\* An user could potentially be logged into the platform with different sessions. Relationship of Session to View is one-to-one.

\* In the scenario, with 2 users (UserA, UserB) with valid access to a domain-entity (id: D100) will have following relation:

\*

\* UserA\_\_\_\_\_\_\_\_SessionA1\_\_\_\_\_\_\_\_\_QuadA1\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_D100

\* |\\_\_\_\_\_\_\_\_SessionA2\_\_\_\_\_\_\_\_\_QuadA2\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/|

\* \\_\_\_\_\_\_\_\_SessionA3\_\_\_\_\_\_\_\_\_QuadA3\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/|

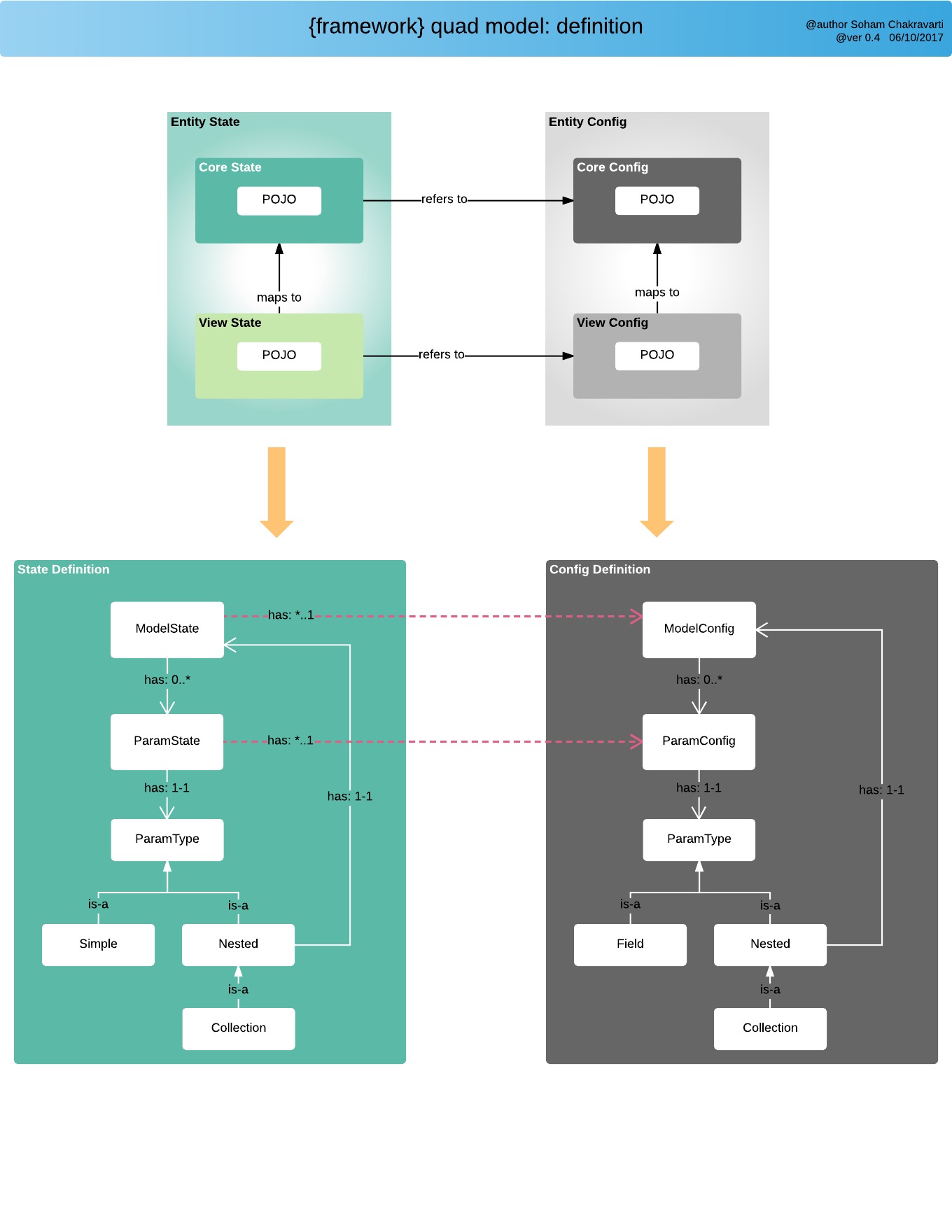
\* | <br>

\* UserB\_\_\_\_\_\_\_\_SessionB1\_\_\_\_\_\_\_\_\_QuadB1\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/|

\* \\_\_\_\_\_\_\_\_SessionB2\_\_\_\_\_\_\_\_\_QuadB2\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/

\*

Quad Model



\* Domain state is always retrieved directly from the Repository API.

\* Quad State is always obtained from User Session and persisted back to DB on explicit \_save or $save

\*

\*

\* Within a Session, each Flow for a given domain-alias can point to multiple Quad models.

\*

\* Example:

\* flow\_car has two pages. First page is for searching existing cars and second is to display details of any one unique car.

\*

\* flow\_car/\_new - creates new entity and assigns unique persistence id, if configured with {@linkplain Repo}

\* flow\_car:100/\_get - checks if entity exists in session, others retrieves & puts entity in session, if configured as such in {@linkplain Repo}

\* flow\_car/\_search - creates {@linkplain QuadModel} in transient mode and doesn't interact with session

\*

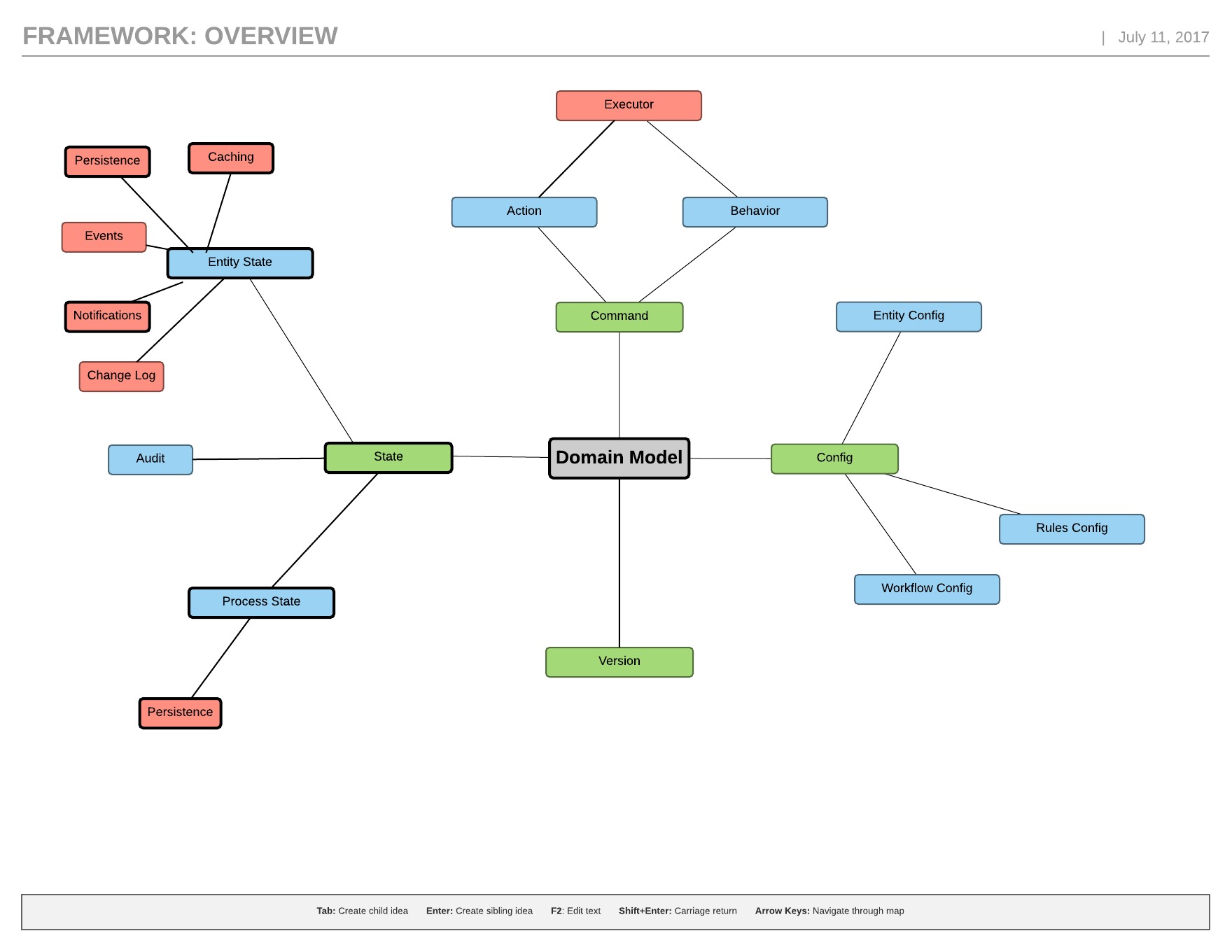
\* Possible URLs:

\* /flow\_car/\_new\_\_\_\_\_Quad1\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_new Car()

\* /flow\_car/search \_\_\_\_\_Quad2\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Car[instance level of Quad1]

\* /flow\_car/details:100\_\_\_\_\_Quad3\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Car[100]

\* /flow\_car/details:200\_\_\_\_\_Quad4\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Car[200]



**Domain Model** - For any application we first have to define the business entity/entities. This would be the first step in the process of building the product.

Domain

Core Config configuration @Domain annotation persists data.

Core config @Domain will always be followed by @Repo that will specify the way data is persisted.

**includeListeners={ListenerType.persistence, ListenerType.update}** of @Domain specifies that the data will be persisted.  
**value=Database.rep\_mongodb** of @Repo specifies that a class with @Domain annotation will use MongoDb for persistence.

Repo :

/\* 2rd level repository: persistent stores \*/

Database {

rep\_none,

rep\_mongodb,

rep\_rdbms,

rep\_ws;

/\* 1nd level repository: cache (distributed session or sticky) \*/

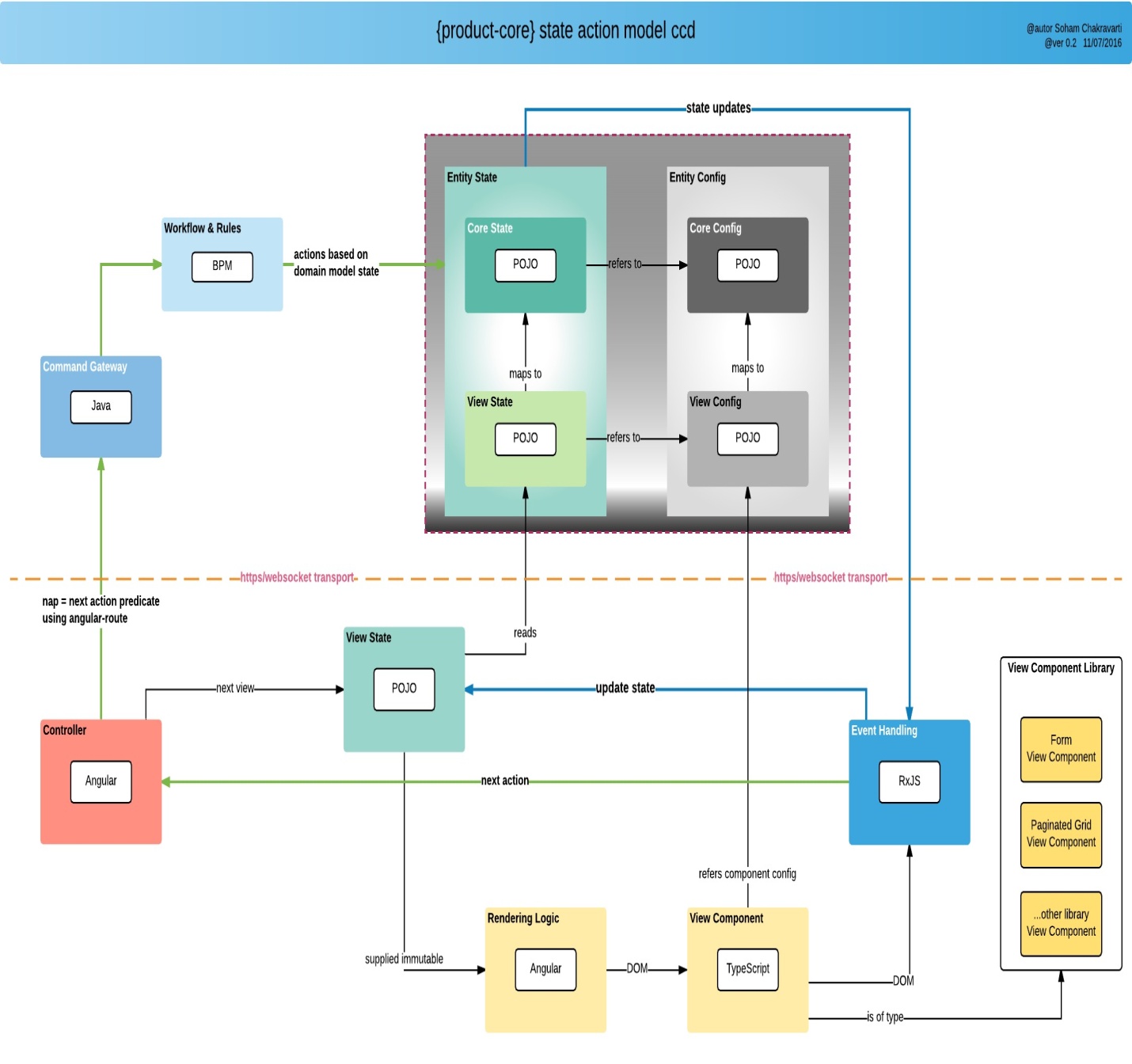
Cache {

rep\_none,

rep\_device;

// rep\_user,

// rep\_entity;



**State** - The value of every entity and its corresponding attributes is referred to as state by the framework. There could various events for example generated based on the state and certain other things associated with the state. To get the history of changes that happened on an entity attribute, we would need audit to be enabled.

StateType

isTransient()

isCollection()

isNested()

isArray()

Nested<P> findIfNested()

NestedCollection<P> findIfCollection()

MappedTransient<P> findIfTransient()

Nimbus frame work Default StateEventHandler for ViewConfig.Modal that sets default contextual values for enabled and visible to the value in the provided ParamContext field context

#### State Event Handlers

The role of a State Event Handler is to instruct the framework on how to process a conditional annotation. Therefore, it is responsible for maintaining the logic to be applied when a conditional annotation is present on a param. State Event Handlers are registered with the framework through the use of @EventHandler on startup as Spring beans. The logic is then executed during specific event hooks defined within the framework.

use the Action output from the setState to check if the action performed is \_update to return true

Stateholder:

It holds all details of state , by using this nimbus fw will take care which actions are allowed at that particular state

1.e Statelife cycle

Rules.

Events.

Notifications. and vise versa.

Param state:

\* Local is always kept, but follows behind cache if configured.

\*

\* 1. If cache=true, then retrieve state from cache AND set to local before returning if local state is different

\* 2. If cache=false, then

\*/

fire rules at root level upon completion of all set actions

// evaluate BPM

1)private String unmapped\_String; // unmapped

@Path

2)private Integer audit\_Integer; // mapped view to a core

3) @Path(value="/a/b/c/action", linked=false)

private Integer audit\_Integer;  // mapped detached and determine if the path has no links

If any thing goes wrong throws errors like:

"MapsTo param must not be null

MapsTo transient param must not be null ….. etc

### Conditional Config Annotations

This section covers the conditional behavior that can be applied to params. In this context, conditional behavior refers to performing a set of actions based on a given condition. The condition is evaluated by using the powerful capabilities of SpEL (Spring Expression Language).

Typically this conditional behavior is evaluated from the context of the conditionally decorated param (the param that is annotated with a conditional annotation), meaning that the condition will inspect the decorated param object to infer if the condition should be true or false.

##### AccessConditional

##### ActivateConditional

##### ConfigConditional

##### EnableConditional

##### ExpressionConditional

##### ValidateConditional

##### The annotation is used to conditionally activate/deactivate the param based on a SpEL condition.  Framework provides default event handling for this annotation on StateChange and StateLoad.

* Some  affects both "enabled" and "visible" state.

Events:

1) StateEvent

T @interface OnStateLoad

T @interface OnStateChange

T @interface OnTxnExecute

2) CommandEvent

T @interface OnRootExecute

T @interface OnSelfExecute

3) ConfigEvent

T @interface OnParamCreate

4) ExecutionRuntimeEvent

T @interface OnRuntimeStart

T @interface OnRuntimeStop

**Rule**

@Rule allows its decorated field a mechanism for triggering one or more rule definitions during its OnStateLoad and OnStateChange events.

### Rule Configuration

Framework comes with support for Drools Rules. The rules can be defined at multiple levels.

**Where can rules be configured?**

* **Entity Level**:
* **Function Handler Level**:
* **Attribute Level**:
* **Configuration Level:**

/\*\*

\* Rule State Event handler for triggering one or more rule definitions during its

\* OnStateLoad and OnStateChange events.

/\*\* Retrieves the RulesConfig for ruleAlias. If it has already been created,

\* the returned value will be retrieved from a local cache relative to this StateEventHandler.

\*

\*ruleAlias- The rule file alias

\***return** the RulesConfig for ruleAlias

**(annotations)Config** - Once we have the domain model, we can define the configuration for the view, workflow and the rules. The view definition configs, the mapping to the domain model, the workflow(if any) and the corresponding view and core domain rules can be written.

#### Configuration

* Business Entity configuration
* View configuration
* Business Rule configuration
* Workflow Configuration

### Process Configuration

#### Business Process Configuration

All standard BPM functions are available for creating business processes. Business process can be defined to manage business entity lifecycle and for creation of stateless processes that executes complex business functions.

#### Entity Lifecyle Management using BPM

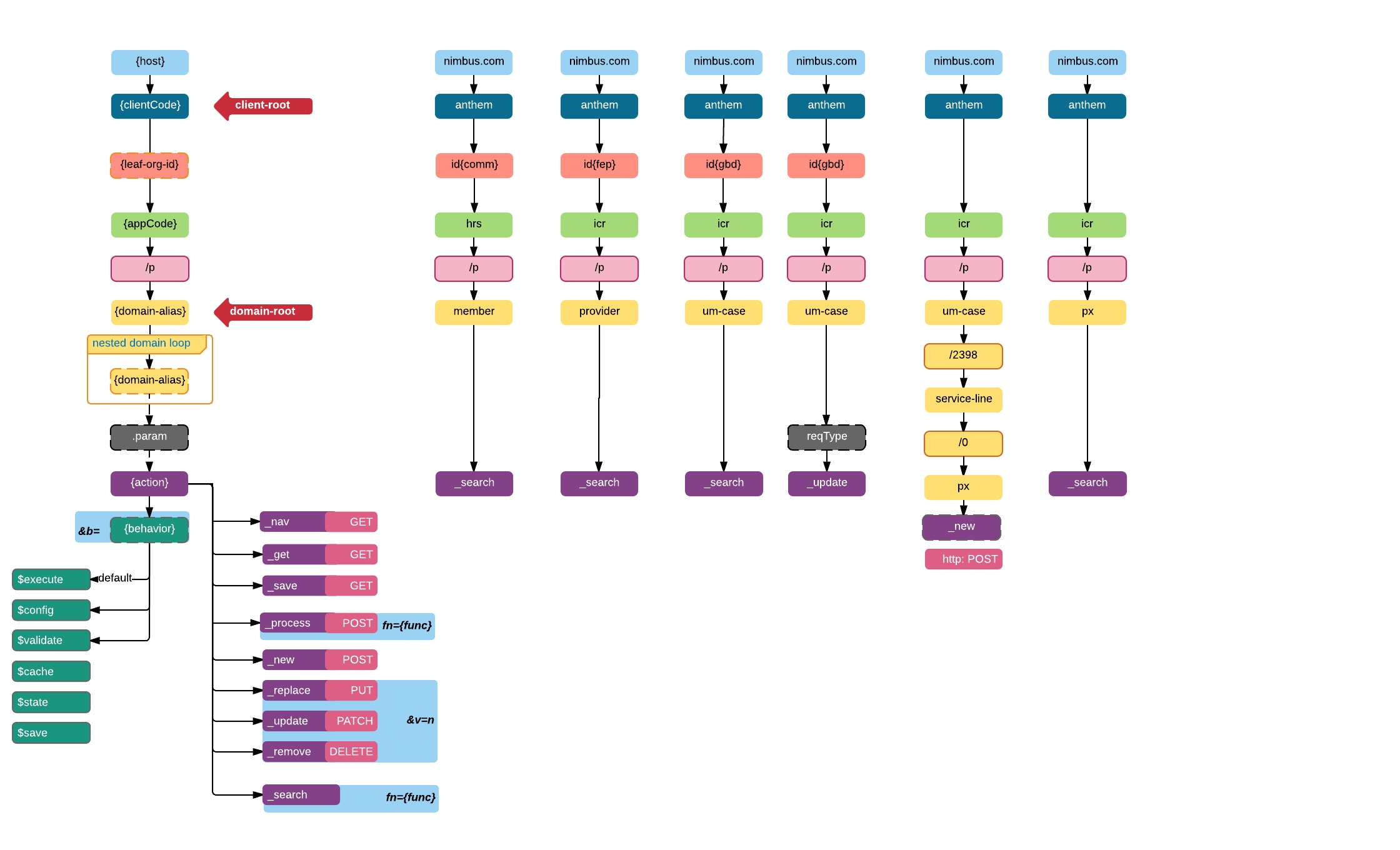
The framework provides the ability to back a business or view entity using a workflow. There might be scenarios where an entity needs to traverse through a series of business steps. Having a BPM defined through standard BPMN construct can help with configuring the business steps and will also provide a visual depiction of the business configuration.

#### Associating a workflow with an entity

Steps for configuring a BPMN process as a function

* Create a BPMN process with a unique process id
* Define the config url with action as **\_process**,**fn** as **\_bpm** and **processId** as the name of the process id. Example **@Config(url="/p/patient:<!/.m/id!>/\_process?fn=\_bpm&processId=createcaseforpatient**

**Command** - The command is the instruction that the framework understands to execute and come back with an output. It is similar to writing the traditional method calls for button click to do some business logic but just that we have standardized the process of writing such to the domain specific language that the framework understands.



PLATFORM\_BASE\_URL:

‘Protocol+HOST+PORT+'/Anthem/ltss/p'

client-code + app-code + /p + domain-alias + action + behavior + RequestMethod.Type

ShowCustomer

@Config(url="/rootpageAddEditGoal/tileEditGoal/sectionEditGoal/goalDetailsForm/\_nav?pageId=pageCarePlanSummary")

@Button(type = Button.Type.PLAIN)

private String showCustomer;

A class with @Config annotation is used to perform an action on button click. In most cases, the action is to retrieve values via HTTP Rest calls from database (MongoDB), and display on the web page.

In the example shown above, when the button is clicked, the control will be navigated to the specified url. nav is the http call for navigation.

@Configs({

@Config(url="~/client/org\_name/\_update"),

@Config(url="~/client/org\_name/\_process?fn=\_set&amp;url=/p/cmcase/\_search?fn=query&amp;where=cmcase.patientReferred.firstName.eq('<!/.m/patientReferred/firstName!>').and(cmcase.patientReferred.lastName.eq('<!/.m/patientReferred/lastName!>'))"),

@Config(url="~/client/org\_name/\_nav?pageId=vpAdvancedCaseSearch")

        })

@Button(type=Button.Type.PRIMARY, formReset=false)

private String submit;

@Configs is a collection of configuration calls. Multiple calls to the database can be made simultaneously using @Configs, where each call will be represented by @Config.

As shown in the example, when the submit button is clicked, three http calls that are being made viz. update, process, and navigation.

The possible Actions are: -

Action :

/\* CRUD \*/

\_get, //HTTP GET - defaults to \_detail

\_save, //HTTP GET

\_new, //HTTP POST

\_replace, //HTTP PUT - full update

\_update, //HTTP PATCH- partial update

\_delete, //HTTP DELETE

/\* transient state \*/

\_search,

\_config,

/\* process \*/

\_process, //Allows for custom process/work-flow definitions

/\* navigation \*/

\_nav

**Note:** Action DEFAULT = \_get;

pts : remember and consider all the defaults

Command

* Target application identifier: Anything prior to /p identifies the application associated with the request
* Domain identifier: Anything post /p identifies the domain for which the request is to be processed.
  + Action
* \_new: Creates a new instance for the model
* \_get: Fetches the instance of the model referenced by the Id
* \_save: Saves the model into the database
* \_replace: Replaces the model state
* \_update: Updates the model state
* \_remove: Removes the model from the database
* \_search: Searches the model based on a search criteria
* \_process: Executes asigned workflow process or custom handlers

**Function Handlers** are an abstraction within the framework to execute/handle a common set of f/w instructions for a given **Action**.

Given an action, a particular function handler can be executed by specifying a value for the query parameter **fn**. Ex: fn=\_set

**Where , set** is the Function Handler we have configured to execute as a part of this **@Config**configuration.

@Config(url="/p/patient:<!/.m/id!>/\_process?fn=\_set")

**@Config(url="/vpOwners/vtOwners/vsSearchOwnerCriteria/vfSearchOwnerCriteria/vbgSearchOwner/\\_process?fn=\\_custom")**

Using:

\_new?fn=\_initEntity

\_get?fn=param

\_nav?fn=default

\_process?fn=\_set

\_process?fn=\_update

\_process?fn=\_setByRule

\_process?fn=\_add

\_process?fn=\_bpm

\_search?fn=lookup

\_search?fn=example

\_search?fn=query

\_process?fn=\_eval

Behaviours:

// .../p/flow\_um-case/\_process?b=$save

// .../p/flow\_um-case/\_findPatient/\_process?b=$execute

// .../p/flow\_um-case/\_findPatient/\_process?b=$executeAndSave

// .../p/flow\_um-case/requestType/\_update?b=$executeAndSave

--

// To set multiple values values in target since the new command does not have handle of the old object

// Ex : @Config(url="/p/queue/\_new?fn=\_initEntity&target=/entityId&json=\"<!/.m/id!>\"&target=/name&json=\"<!/.m/name!>\"")

**Rule**

@Rule allows its decorated field a mechanism for triggering one or more rule definitions during its OnStateLoad and OnStateChange events.

@Config(url="/vpAdvancedMemberSearch/vtAdvancedMemberSearch/vsMemberSearchCriteria/vfAdvMemberSearch/\_process?fn=\_setByRule&amp;rule=updateadvmbrsearchcriteria")

private VSMemberSearchCriteria vsMemberSearchCriteria;

### Rule Configuration

Framework comes with support for Drools Rules. The rules can be defined at multiple levels.

**Where can rules be configured?**

* **Entity Level**:
* **Function Handler Level**:
* **Attribute Level**:
* **Configuration Level:**

SampleRuleEntity.java

|  |
| --- |
| @Domain(value="sample\_rule\_entity", includeListeners={ListenerType.persistence})  @Repo(Database.rep\_mongodb)  @Getter @Setter  public class SampleRuleEntity {        // Execute the rule at "rules/sample\_increment" during the OnStateLoad and      // OnStateChange events of ruleParam.      @Rule("rules/sample\_increment")      private String rule\_param;  } |

By default, the framework provides support for firing all rules for a given domain entity. That is, for the **SampleRuleEntity.java** above we might have a rule file defined as **sample\_rule\_entity.drl** which will be automatically fired by naming convention.

For cases where additional configuration for other rules is needed, @Rule can be used.

@Values(url="CLIENT\_ID/ORG/p/staticCodeValue/\_search?fn=lookup&amp;where=staticCodeValue.paramCode.eq('/petType')")

@CheckBoxGroup

private String petTypes;

##### Nimbus Search Filtering

Nimbus uses query dsl syntax to construct filtering queries.

###### Criteria

If the requirement is to search for all records that have age between 18 and 21, the criteria will be as below

&criteria=testdsl.age.between(1,11)

client-code + app-code + /p + domain-alias + action + behavior + RequestMethod.Type

|  |
| --- |
| STEPS to follow with examples. |
|  |
| EXAMPLES: |
| Ex 1: /anthem/ltss/member/\_search - POST operation which contains criteria |
| Ex 2: /anthem/ltss/member/{id} - GET operation which has member id as path variable |
| Ex 3: /anthem/ltss/member/{id}/\_info - GET operation which contains member id as path variable, but also contains an action 'info' to return less data |
| Ex 4: /anthem/ltss/um/case/{cid}/member/address/{aid} - GET operation to return one address from the collection |
| Ex 5: /anthem/ltss/um/case/{cid}/member/address - POST operation to create new address |
| Ex 6: /anthem/ltss/um/case/{cid}/member/address/{aid}?version=n - PUT operation to update a given address. This would be a complete model update |
| Ex 7: /anthem/ltss/um/case/{cid}/member/address/{aid}?version=n - PATCH operation to update a given address. This would be a partial model update |
| Ex 8: /anthem/ltss/member/\_search$validate - POST operation which contains criteria which will be validated and response provided per field |
| Ex 9: /anthem/ltss/member/\_search$config - GET operation that returns static meta data created for 'member/\_search' process |
| Ex 10: /anthem/ltss/um/case/{cid}/member/address$validate - POST operation that contains address field populated which will be validated at each field level |
| Ex 11: /anthem/ltss/address$validate - POST operation |
| STEPS: |
| 1. Inject a list of available domain-aliases to consider |
| 2. Inject a list of available actions to consider |
| 3. Inject any rule to explicitly allow or prohibit a certain combination of: client-code + app-code + /p + domain-alias + action + behavior + RequestMethod.Type |
| 4. Inject patterns to follow |
| /{app}/{domain-alias}/{id} |
| a. domain-alias would be identified from injected list</li> |
| b. {id} would be unrecognized and considered as refId for identified domain-alias |
| c. If the HTTP method is GET, then action would be "\_get" |
| d. If the HTTP method is POST, then action would be "\_create" and any RequestBody JSON object would be used |
| e. For DELETE - '\_delete', PUT - '\_replace', PATCH - '\_update'. All these operations require mandatory 'version' parameter |
|  |
| /{app}/{domain-alias-A}/{domain-alias-B}/{B\_id}/.../{domain-alias-N}/{N\_id} |
| a. Detect all domain-aliases from the url and corresponding refId, if available |
| b. Follow same steps as above in '/{domain-alias}/{id}' |
|  |
| /{app}/{domain-alias}/{\_action} -OR- /{app}/{domain=alias}/{refId}/{\_action} |
| a. domain-alias would be identified from injected list |
| b. action would be identified from injected list |
| c. Follow same steps as above for recurring domain-alias identification |
|  |
| /{app}/{domain-alias}/{\_action}.{$behavior} OR /{app}/{domain-alias}{$behavior} |
|  |
| 5. Action can appear only once and at the end of the url but prior to Behavior, if any |
| 6. Action if explicitly provided, overrides inferred action. Ex: /{app}/{domain-alias}/{id}/\_delete GET : would be treated as '\_delete' action and not '\_get' inferred from GET methods |
| 7. GET/\_get and DELETE/\_delete only accepts {refId} as argument and not any RequestBody |
| 8. Behavior can appear only once at the end or url prefixed by '.' Ex: .../../../\_search$config OR ../../\_search$validate |

--

there is a .d that signifies the mapping to the core domain model i.e. it will point to the root which is CarEntity.

then switch from view config to core config using .m to assign a new car to attr\_list\_2\_CarNestedEntity list.

P - URI\_PLATFORM\_MARKER

b- URI\_BEHAVIOR\_MARKER

{

param\_path\_identifier - ("<! !>"), ex:

or

("<!")- PLATFROM\_EXPR\_PREFIX\_MARKER,

("!>")- PLATFROM\_EXPR\_SUFFIX\_ MARKER

}

URI\_PLATFORM("p"),

MARKER\_URI\_BEHAVIOR("b"),

MARKER\_COLLECTION\_ELEM\_INDEX("{index}"),

MARKER\_PLATFROM\_EXPR\_PREFIX("<!"),

MARKER\_PLATFROM\_EXPR\_SUFFIX("!>"),

MARKER\_SESSION\_SELF("#self"),

MARKER\_COMMAND\_PARAM\_CURRENT\_SELF("#this"),

MARKER\_REF\_ID("#refId"),

MARKER\_ELEM\_ID("#elemId"),

MARKER\_COL\_PARAM("col"),

MARKER\_COL\_PARAM\_EXPR("<!col!>"),

MARKER\_URI\_PAGE\_EXPR("page=y"),

SEPARATOR\_URI("/"),

SEPARATOR\_URI\_PLATFORM(SEPARATOR\_URI.code + MARKER\_URI\_PLATFORM.code), /\* /p \*/

SEGMENT\_PLATFORM\_MARKER(SEPARATOR\_URI\_PLATFORM.code + SEPARATOR\_URI.code), /\* /p/ \*/

SEPARATOR\_URI\_VALUE(":"),

SEPARATOR\_URI\_PARENT(".."),

SEPARATOR\_URI\_ROOT\_DOMAIN(".d"),

SEPARATOR\_URI\_ROOT\_EXEC(".e"),

SEPARATOR\_CONFIG\_ATTRIB("#"),

SEPARATOR\_UNIQUE\_KEYGEN("^"),

SEPARATOR\_BEHAVIOR\_START("$"),

SEPARATOR\_AND("And"),

SEPARATOR\_MAPSTO(".m"),

PREFIX\_FLOW("flow\_"),

PREFIX\_DEFAULT("default."),

PREFIX\_EVENT("e"),

PREFIX\_EVENT\_URI("e"+"\_"),

SUFFIX\_PROPERTY\_STATE("State"),

CODE\_VALUE\_CONFIG\_DELIMITER("-"),

PARAM\_VALUES\_URI\_PREFIX("\*/\*/\*/p/"),

PARAM\_VALUES\_URI\_SUFFIX("/\_lookup"),

KEY\_FUNCTION("fn"),

KEY\_FUNCTION\_NAME("name"),

KEY\_NAV\_ARG\_PAGE\_ID("pageId"),

KEY\_FN\_INITSTATE\_ARG\_TARGET\_PATH("target"),

KEY\_FN\_INITSTATE\_ARG\_JSON("json"),

KEY\_FN\_PARAM\_ARG\_EXPR("expr"),

KEY\_EXECUTE\_PROCESS\_CTX("processContext"),

KEY\_EXECUTE\_EVAL\_ARG("eval"),

KEY\_EXECUTE\_PROCESS\_ID("processId"),

REQUEST\_PARAMETER\_MARKER("?"),

CLIENT\_USER\_KEY("client-user-key"),

REQUEST\_PARAMETER\_URL\_MARKER("url"),

REQUEST\_PARAMETER\_DELIMITER("&"),

PARAM\_ASSIGNMENT\_MARKER("="),

/\* search request param constants \*/

SEARCH\_REQ\_PROJECT\_ALIAS\_MARKER("projection.alias"),

SEARCH\_REQ\_PROJECT\_MAPING\_MARKER("projection.mapsTo"),

SEARCH\_REQ\_AGGREGATE\_MARKER("aggregate"),

SEARCH\_REQ\_AGGREGATE\_COUNT("count"),

SEARCH\_REQ\_FETCH\_MARKER("fetch"),

SEARCH\_REQ\_ORDERBY\_MARKER("orderby"),

SEARCH\_REQ\_WHERE\_MARKER("where"),

SEARCH\_REQ\_PAGINATION\_SIZE("pageSize"),

SEARCH\_REQ\_PAGINATION\_PAGE\_NUM("page"),

SEARCH\_REQ\_PAGINATION\_SORT\_PROPERTY("sortBy"),

SEARCH\_NAMED\_QUERY\_DELIMTER("~~"),

SEARCH\_NAMED\_QUERY\_RESULT("result");

\* 1. If the command is domain root only, then create new instance

\* 1.1. Check if payload contains json for initial object to be inserted; convert if available

\* 1.2. Else, create new instance and call rep to persist

\* 1.3. Update command with domain root refId

\* 2. Else, use the payload of command message json to convert & instantiate desired object

\* 2.1. Traverse object model path using command domain uri

\* 2.2.

-- this document is in pending still need to update a lot

// Executes a request against \_subscribe, and sets the image

@Link(url="/notifications/\_subscribe:{id}/\_process", b=" And$configAnd$nav" , method="POST")

 @Configs({ @Config(url = "/p/cmcaseview:<!/.m/id!>/\_get?b= "),

        @Config(url = "/p/cmcaseview:<!/.m/id!>/\_nav?pageId=pageCaseInfo")

    })

    @Link(imgSrc = "task.svg")

    @Label(value = "View Case")

    private String viewCase;

    @Configs({

        @Config(url = "/p/caseassignmentwithgridview/\_new?fn=\_initEntity&amp;target=/vpAssignmentTask/vtTaskDetails/vsShowMycases/associatedCaseId&amp;json=<!/.m/id!>")

    })

    @Link(imgSrc = "task.svg")

    @Label(value = "Assign Case Owner")

    private String assignCase;

pts : remember and consider all the defaults

ViewRoot:

@Domain(value = "cmcaseview", includeListeners={ListenerType.websocket}, lifecycle="cmcaseview")

@MapsTo.Type(CMCase.class)

@Repo(value=Database.rep\_none, cache=Cache.rep\_device)

@ViewRoot(layout = "caseoverviewlayout")

@Getter @Setter

public class VRCmCase {

}

[//@Model.Param.Values(url="Anthem/ltss/p/staticCodeValue/\_search?fn=lookup&where=staticCodeValue.paramCode.eq('/orgType')")](mailto://@Model.Param.Values(url=%22Anthem/icr/p/staticCodeValue/_search?fn=lookup&where=staticCodeValue.paramCode.eq('/orgType')%22))

paramcodeValue provider:

/\*\*

\* Search will be in the order:

\* 1. static code values (in below order):

\* 1.1 config server, if not found

\* 1.2 DB

\* 2. Model as code values (in below order)

\* 2.1 config server, if not found

\* 2.2 DB

\*

\*/

--