- implementation in web application pmc-vis that aims for exploring and viewing mdps.
- since it is a webbased application the general project structure consists of two parts: the fronted and the backend.
- the frontend is responsible for the vizualization and provides several options to do so.
- the backend supplies the data for the frontend in json format. Inernally it consists of three parts. The prism model, the java server and several databases. It controls a set of prism models, parses them and creates q database for each of them, containing the MDP structure. Each files contains two tables. One for the states one for transitions, where in both each line represents a state or a transition.
- information such as properties are written to the respective state table. For a new property a new collumn is added assigning each state the value of the property. In the same way the mapped values from the groupingfunction of a view are stored in a dedicated collumn example?
- views are implemented in the server on the backend in the package prism.core.views.
- In general each view is a dedicated java class, but sometimes several similar views are realized in a single java class
- All views are derived from an abstract class View, which contains attributes that shall be available in all views, as well as methods that are needed by all views. Many of these are used for testing and I/O. In Figure an overview of relevant attributes and methods is shown.
- The most important method is buildView() which accomplishes that the mappings of the grouping function are computed and written to the states table in the database of the model it is built on. It consist of 4 parts. 1. Checking if requirements for building a view are met such as if the view is not already built 2.create a new column in the database where the results of the grouping function are to be saved. 3. The actual application of the grouping function, where for every state an SQL Query the grouping function value is added to a list of SQL Query strings. This list is the return value of groupingFunction() 4. all the SQL queries that contain the insertion operation of the grouping function value being executed.
- when implementing a new view only the grouping function, getCollumn (and some I/O) as well as for the view necessary private attributes have to be implemented. This makes sense because the process of actually generating the view is always the same as it was already notable in the definitions of a view, where every view is defined by its grouping function.

- parallel composition is accomplished by simply building another view. This causes the grouping function entries to be written in a new collumn in the datatabse.

- selective composition is achieved by setting a restriction that corresponds to Z from the Definition **??**. This restriction is realized as a Map, that maps collumn names of the database to a list of values (Strings)

- in contrast to the formalization this restriction is the used to compute an SQL Query, that selects states from the database that meet this requirement. This is accomplished by creating a query in which the where clause is boolean formulae in conjunctive normal form for a selection of collumns which values are allowed

- the notation of disregarding views is implemented a little less general as it has been formalized here.

- In general the variable semigrouping says whether or not something is not grouped. For categorizing views this option is fixed to one value. Only states that would otherwise be mapped to the empty set or string can instead not be grouped. An example is the view $\mathcal{M}_{scc}$ with $n = 3$. For this view it can be selected with the variable semigrouping wheter or not states that are in an SCC with less than three states are grouped. For binary views it can be selected with an enumeration value if $\top$ or $\bot$ shall be disregarded.

- to create a view currenty `localhost:8080/<model_id>/view:<viewname>?param=<param_val` has to accessed via a browser, assumed the backend server and the frontend to be running. With <param_val> required values for the view can be provided. Afterwards with localhost:3000/?id=<model_id>/ the graphical representation of view is displayed relying on the created json file from the backend. This has been created with the access of localhost:8080...

- This caused a call of createView() which uses <viewname> to select which view is to be created and hands over the paramaters. After the View-Object has been instanciated it is called buildView() on it.

- Apart from the actual implementation several functionalties have been implemented to allow the following actions at runtime without restarting the servers:

  - Show current Information (Parameter values etc ) about view as well as which views are currently built
  - rebuild view with new parameters
  - remove all views or specific ones by providing id or name

- DATASTRUCTURE/GRAPH

- implementation of views rely on an internal graph structure that was necessary for the views that implement grouping based on structural properties of the

MDP graph. Views based on MDP components firstly where implemented with direct accesses on the database but later on also adopted to the internal graph structure for performance reasons

- internal graph structure the jGraphtT library was used. It supplies graph structures as well as common algorithms performed on them. It was chosen because it is the most common, most up to date java library for graphs with the best documentation and broadest functionality. It is developed by ... has a java style documentation and is open source. The broad functionality assured that when implementing a view it is most certainly assured that if necessary a respective algorithm is available.

- The MDP was realized as a class extending an directed weighted pseudograph. The transition probabilities are stored as weights. A pseudograph has been chosen because it allows double edges as well as loops.

- in order of keeping the graph lean nodes and vertices are long values. The refer to the state id and the transition id respectively. The state id matches the one in the db whereas the transition id is newly generated because in the database a transition is a action with its probability distribution. This difference to the MDP definition is reversed with the internal graph structure for graph algorithms of the jGraphtT library to work properly on the MDP graph.

- In order to access information about states and transitions, the class MdpGraph contains two hashmaps. One that maps longs to MdpState objects and one that maps longs to MdpTransition objects. The classes MdpState and MdpTranstion are inspired from the respective PRISM classes. There contain only as much information as currently neccessary for views to work. That is MdpState contains the variable values and MdpTransition contains the action. Accessing this information via a HashMap has enables a modular design if these information later on were to be rather stored somewhere else.

- example?