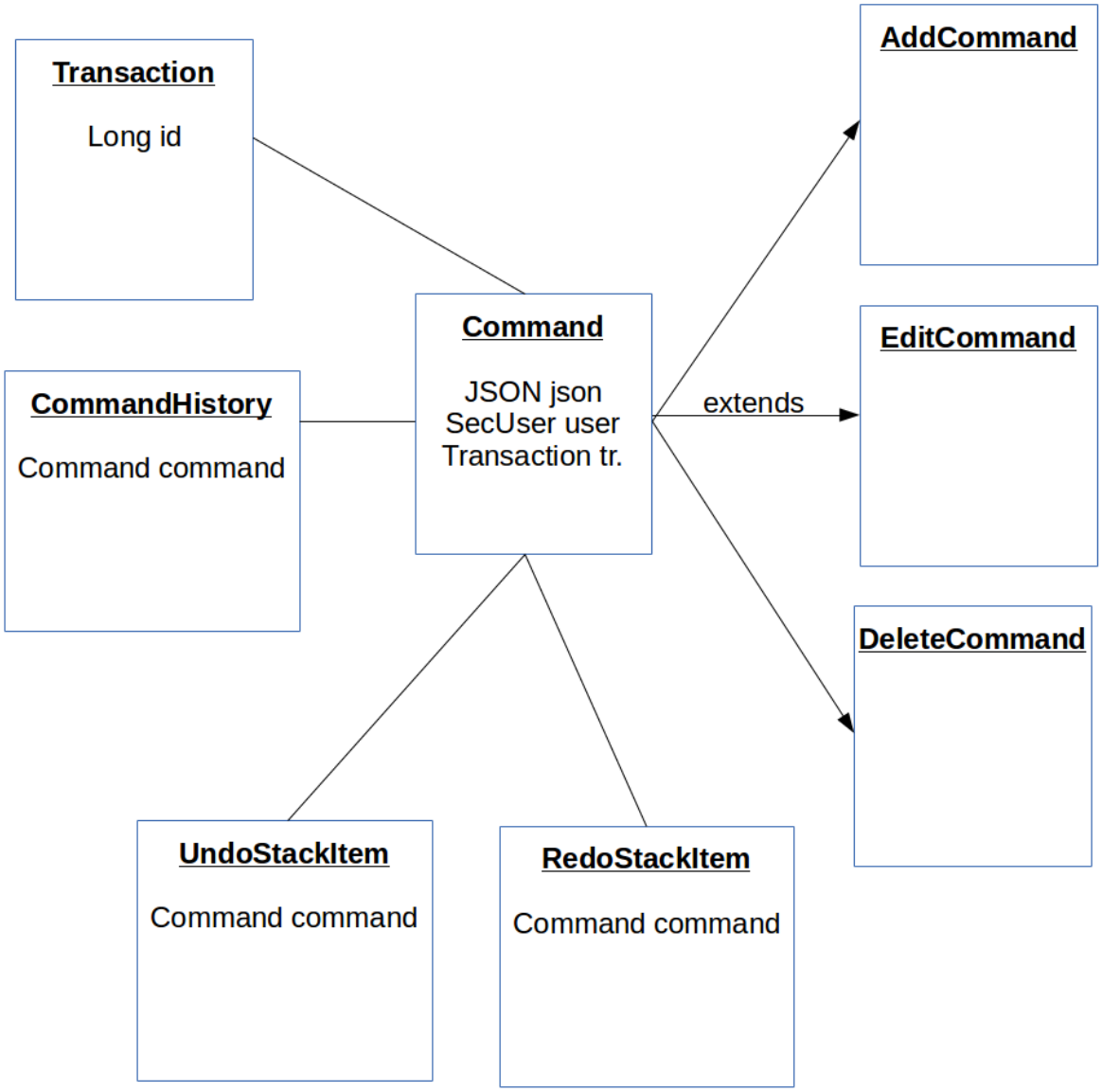
Package Command

Package summary

We store each modification of the database into a command. A modification may be canceled or restored (undo/redo). We are able to provide to the user an activity list based on the command previously done.

Package main classes

Domain	Description	Main properties
Command • AddCommand • EditCommand • DeleteCommand	An action that modifies the content of the application database. An action may be done, undo or redo. There are 3 subclasses for each action type (Add, Edit and Delete).	• String data • (transient) JSON json • SecUser user • Project project • Transaction transaction
CommandHistory	Provide all command history (command done, command undo, command redo). If a command is done and then undo, there will be two CommandHistory domain.	• Command command
UndoStackItem	An action that has been executed. This action is stores on the undo stack so that it may be undo.	• Command command
RedoStackItem	An Action that has been undo. This action is stored on the redo stack so that it may be restore.	• Command command
Transaction	A transaction is a group of command from the same transaction.	



Package description

Command

Each action modifying the content of the application database is stored as a command. It may be a add, edit or delete command depending of the action. A command mainly store the user executing the action and the JSON content of the domain that has been created, deleted or edited (in the last case: it store the version before update and after update).

TermService.add()

```
def add(def json) {
        securityACLService.check(json.ontology, Ontology,WRITE) //simply check if auth to add a term in this
ontology
    SecUser currentUser = cytomineService.getCurrentUser()
    return executeCommand(new AddCommand(user:currentUser),null,json)
}
```

In this sample from the TermService add method, we simply create a new AddCommand and we give it to a generic executeCommand method with the json from the HTTP POST data.

The executeCommand may have 4 arguments:

1. Command command: the command to be executed,
2. Domain domain: the domain affected by the command (in the add case, we pass null because the domain does not exist yet),
3. JSON json: The JSON of the new, edited or deleted domain,
4. Task task: Task object (see Utils package) that may be used to follow the progress of a big operations (example: a UI progress bar when you delete a project)

The executeCommand do some step:

1. If its a DeleteCommand, we look for all method for deleting dependency (see delete dep),
2. The next step is running the execute method of the domain. This is a generic method for all domains.
    a. AddCommand.execute(): create the domain from the JSON, save the domain into the database and create the response.
    b. DeleteCommand.execute(): create the reponse (when the domain still exist) and destroy the domain from the database.
    c. EditCommand.execute(): Backup the old domain version JSON, update the domain into the database and create the response.
3. Storing the command in the database and creating a CommandHistory instance

Transaction

Each command is linked with a transaction. A transaction may group many commands that needs to be executed atomically.
For example: If a service allows you to add a new ontology with some terms, you will have one AddCommand for the ontology, one AddCommand for the first term, …
All these commands are from the same transaction. If you undo the Ontology creation, you need to the Term commands. It will be done because they have the same transaction.

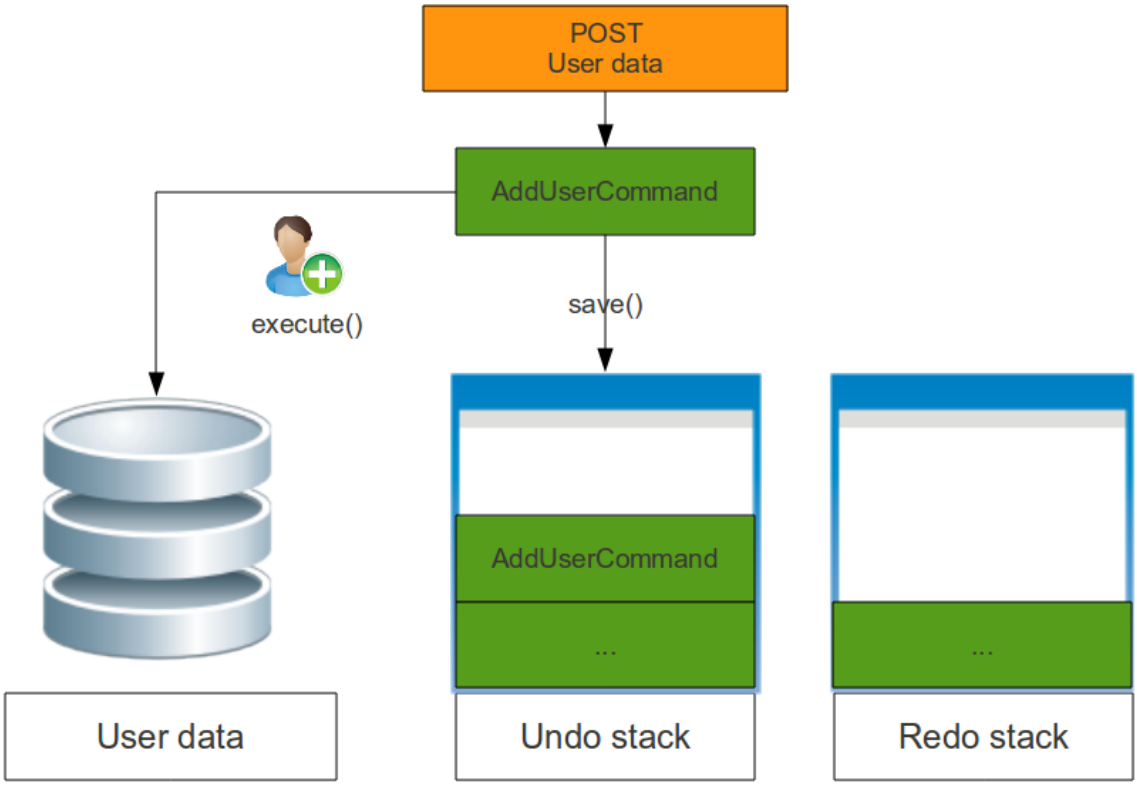
Undo/Redo

A command may be canceled or restored.
For Add and DeleteCommand, an undo is the opposite action. If you undo the creation of a term, you delete the term. The redo is the same action.
For EditCommand, an undo is just the action of restoring the previous version by editing the actual.

	execute()	**undo()**	**redo()**
AddCommand	create	destroy	create
EditCommand	edit	edit	edit
DeleteCommand	destroy	create	destroy

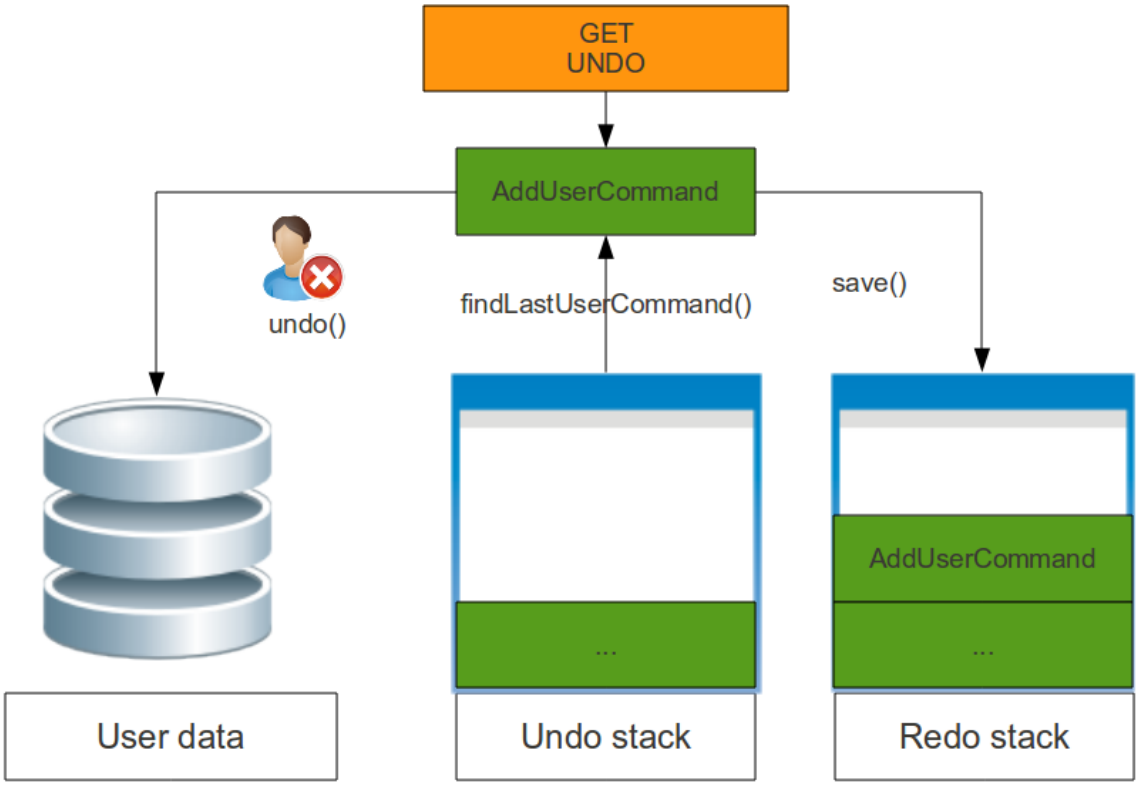
The code the undo/redo mechanism is located in CommandController.undo()/redo().

When a command is created, we will store add the command on the Undo stack (UndoStackItem). Here is a sample with the service that allows you to create a user.



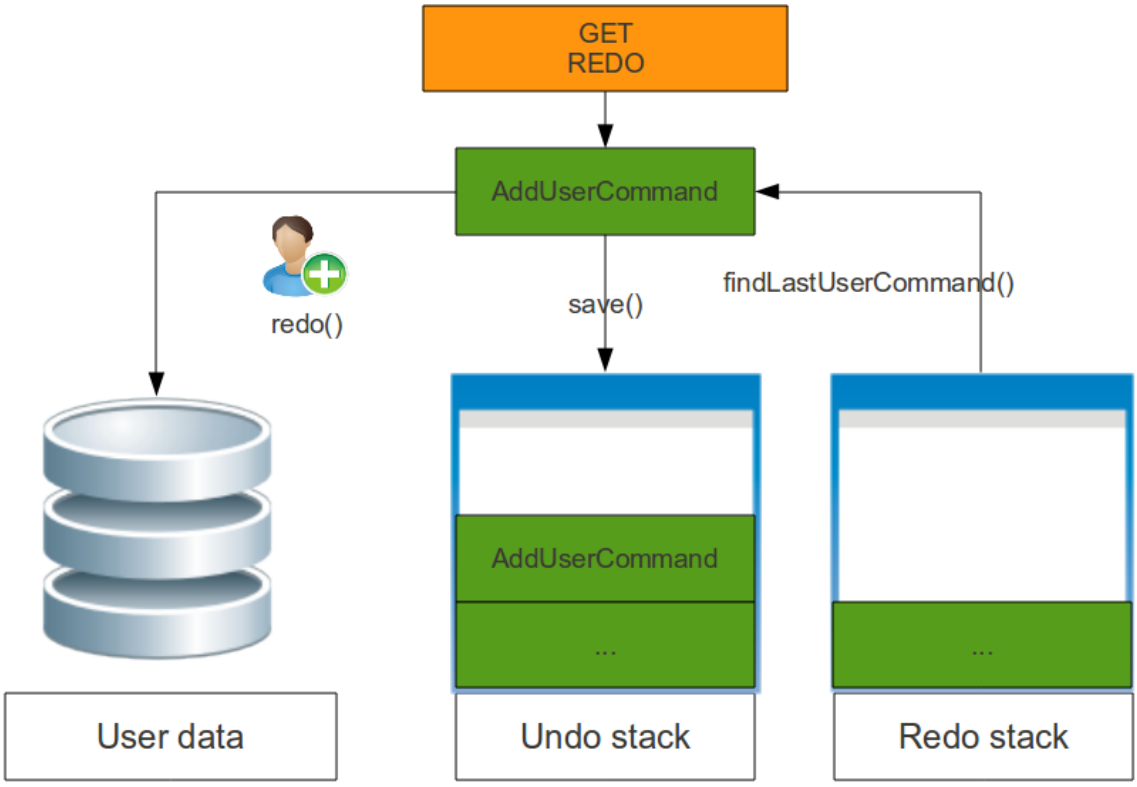
Undo action:

- Get the last command created by the current user on the undo stack,
- If the action is not from a transaction, call the command undo() method and move this command in the redo stack,
- If the action is from a transaction, loop through each command with the same transaction id and do the step 2 for each command.



Redo action:

Same as Undo action but look for command in the redo stack and not in the undo stack. We call the redo() methdod for each command.


GET
REDO
AddUserCommand
redo()
save()
findLastUserCommand()
AddUserCommand
...
...
User data
Undo stack
Redo stack