Movie Rating App

Software Engineering Lab WS21/22

Mohamed Abdelmagied , Janis Evers , Abdulrahman Boushi , Mohammed Alshaikh-Khalil, Ahmed Abdellatif, Ahmed Shehata

Supervisor: Jana Friese

Contents

Analysis Phase

A1: Problem elicitation and description

Requirements

- R1: To use the web application, a person has to register first.
- R2: During the registration process, he/she must provide an email address, his/her age and a username.
- R3: The username has to be unique.
- R4: To register, a person must be at least eighteen years old, otherwise the registration fails.
- R5: The user can log in and log out.
- R6: A logged in user can add movies he/she has watched from a database to his/her list and rate them.
- R7: A rating consists out of a mandatory point rating from one to ten (1 = very bad movie, 10 = excellent movie) and optionally a written comment.
- R8: If a value differing from one to ten is entered, the rating process will fail. Movies without any rating are rated as zero.
- R9: If a movie is not yet contained in the database, the user can add it by providing the title, director, main actors (at least one) and original publishing date.
- R10: Each movie can be contained only once in the database.
- R11: A user cannot give more than one rating per movie.
- R12: Users can access a list of all movies in the database sorted by rating in descending order.
- R13: For each movie in the database, the average rating is calculated and shown together with the comments.
- R14: Registered users can add other registered users into a movie discussion group.
- R15: A member of the group can leave it if he/she wants to.
- R16: A group has a unique name (e.g., “What is the best film of actor x y?”) and a list of group members.
- R17: Within a group, members can see the movie lists of other members and can have a discussion in the form of a group chat.
- R18: A chat message contains the username of its creator, a time stamp and the content.
- R19: In the chat, the messages are sorted by the order of their creation.
- R20: The creator of the group is its administrator.
- R21: Administrator can ban members from the group if he/she thinks that the member is misbehaving.
- R22: When the administrator leaves the group, the group is deleted.
- R23: If a group consists only of one member, an automated method will delete it after a certain amount of time.

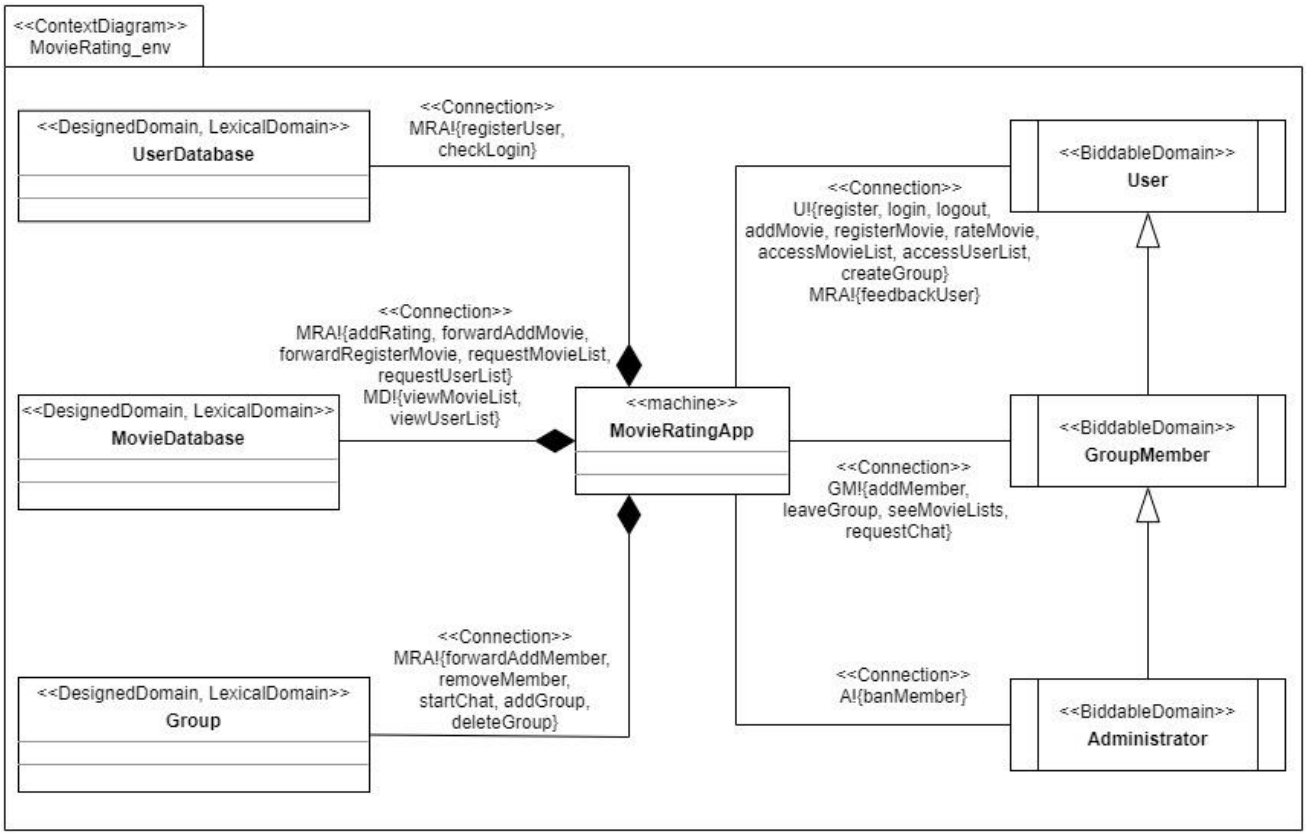
Assumptions

- A1: A web application is suitable to be used on different platforms including mobile devices.
- A2: Users only add new movies that really exist and movie data that is valid.
- A3: Users only rate movies they have really watched, and their rating is based only on their own opinions.
- A4: The administrator's decisions are always fair.
- A5: In groups, members only discuss movie related topics.

Facts

- F1: Each movie has a title, a director, at least one main actor and an original publishing date.

Context Diagram



Validation A1

- Notions mentioned in Requirements and Domain Knowledge are contained in the glossary
- Domains and phenomena of the context diagram must be consistent with Requirements and Domain Knowledge

Notions in CD	Notions in R/D	Type
MovieRatingApp	The software we are going to build.	Domain
register	Has to register	Phenomenon
login	can log in	Phenomenon
logout	can log out	Phenomenon
MovieDatabase	Database R6	Domain
watch	he/she has watched	Phenomenon
addMovie	Can add movie they watched to their list	Phenomenon
forwardAddMovie	Can add movie they watched to their list	Phenomenon
User	user	Domain
rateMovie	and can rate them, rating process	Phenomenon
registerMovie	If a movie is not yet contained in database a user can add it	Phenomenon
forwardRegisterMovie	If a movie is not yet contained in database a user can add it	Phenomenon
accessMovieList	User can access a list of all movies in the database	Phenomenon
accessUserList	User can access a list of his/her movies in the database	Phenomenon
requestMovieList	User can access a list of all movies in the database	Phenomenon
viewMovieList	User can access a list of all movies in the database	Phenomenon
addRating	The average rating is calculated	Phenomenon
addMember	Can add other registered users	Phenomenon
leaveGroup	Can leave group	Phenomenon
forwardAddMember	Can add other registered users	Phenomenon
GroupMember	A member of the group	Domain
Group	group	Domain
seeMovieLists	Can see movie lists of other members	Phenomenon
requestMovieList	Can see movie lists of other members	Phenomenon
requestUserList	Can see movie his list of Movies	Phenomenon

requestChat	Can have discussions in the form of a group chat	Phenomenon
viewUserList	User can access his/her list of movies in the database	Phenomenon
startChat	Can have discussions in the form of a group chat	Phenomenon
Administrator	Administrator of the group	Domain
createGroup	The creator of the group	Phenomenon
addGroup	The creator of the group	Phenomenon
setAdmin	The creator of the group is its administrator	Phenomenon
removeMember	deleting member from the group	Phenomenon
banMember	Can ban members from the group	Phenomenon
adminLeaveGroup	Administrator leaves the group	Phenomenon
deleteGroup	The group is deleted, an automated method will delete it	Phenomenon

Table [1]

- There must be exactly one context diagram.
    - Only one context diagram is provided.
- A context domain has at least one machine domain.
    - MovieRatingApp is one machine domain.

Domain	Domain Type(s)	Connected Domain(s)	Connected Domain Type(s)
MovieRatingApp	machine	User	Biddable Domain
		GroupMember	Biddable Domain
		Administrator	Biddable Domain
		Group	Lexical Domain, Designed Domain
		MovieDatabase	Lexical Domain, Designed Domain
		MovieRatingApp	Machine
GroupMember	Biddable Domain	MovieRatingApp	Machine
Administrator	Biddable Domain	MovieRatingApp	Machine
Group	Lexical Domain, Designed Domain	MovieRatingApp	Machine
MovieDatabase	Lexical Domain, Designed Domain	MovieRatingApp	Machine

Table [2]

- The machine domain must control at least one interface
    - MovieRatingApp controls several interfaces (forwardAddMember, addRating, …)
- Biddable domains cannot be directly connected to lexical domains.
    - No biddable domain is connected to a lexical domain

Domain	Domain Type(s)	Connected Domain(s)	Connected Domain Type(s)
MovieRatingApp	machine	User	Biddable Domain
		GroupMember	Biddable Domain
		Administrator	Biddable Domain
		Group	Lexical Domain, Designed Domain
		MovieDatabase	Lexical Domain, Designed Domain
		MovieRatingApp	Machine
GroupMember	Biddable Domain	MovieRatingApp	Machine
Administrator	Biddable Domain	MovieRatingApp	Machine
Group	Lexical Domain, Designed Domain	MovieRatingApp	Machine
MovieDatabase	Lexical Domain, Designed Domain	MovieRatingApp	Machine

Table [3]

- Causal, designed, lexical, display, machine domain type is not allowed together with biddable domain.
    - User, GroupMember and Administrator are biddable domains only.

Domain	Domain Type(s)	Connected Domain(s)	Connected Domain Type(s)
MovieRatingApp	machine	User	Biddable Domain
		GroupMember	Biddable Domain
		Administrator	Biddable Domain
		Group	Lexical Domain, Designed Domain
		MovieDatabase	Lexical Domain, Designed Domain
		MovieRatingApp	Machine
GroupMember	Biddable Domain	MovieRatingApp	Machine
Administrator	Biddable Domain	MovieRatingApp	Machine
Group	Lexical Domain, Designed Domain	MovieRatingApp	Machine
MovieDatabase	Lexical Domain, Designed Domain	MovieRatingApp	Machine

Table [4]

- Phenomena controlled by a biddable domain must have counterpart phenomena located between machine and causal/lexical/designed domains

	Biddable domain phenomena	counterpart
User	addMovie	forwardAddMovie
	rateMovie	addRating

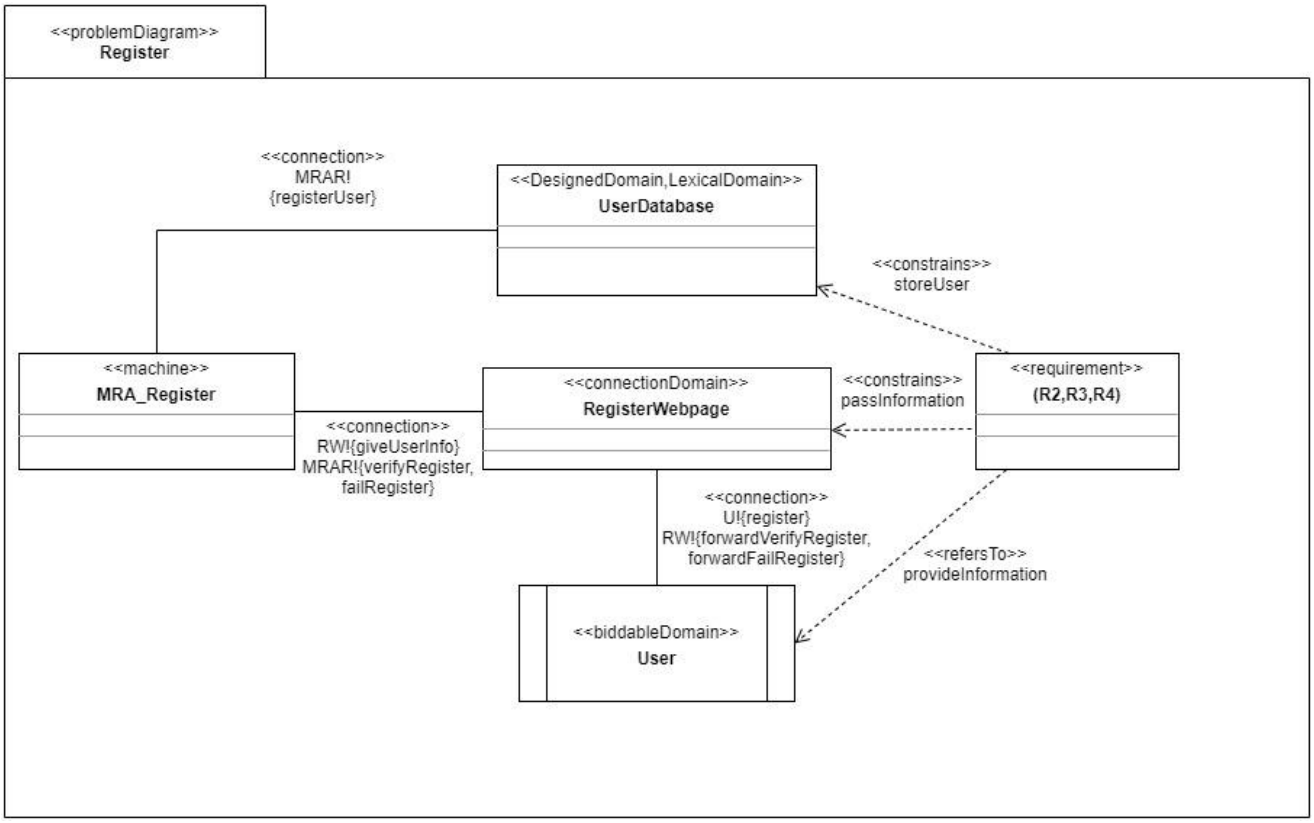
	AccessMovieList	requestMovieList
	createGroup	addGroup
	accessUserList	requestUserList
GroupMember	addMember	forwardAddMember
	leaveGroup	removeMember
	seeMovieList	requestMovieList
	requestChat	startChat
Administrator	banMember	removeMember
	adminLeaveGroup	deleteGroup

Table [5]

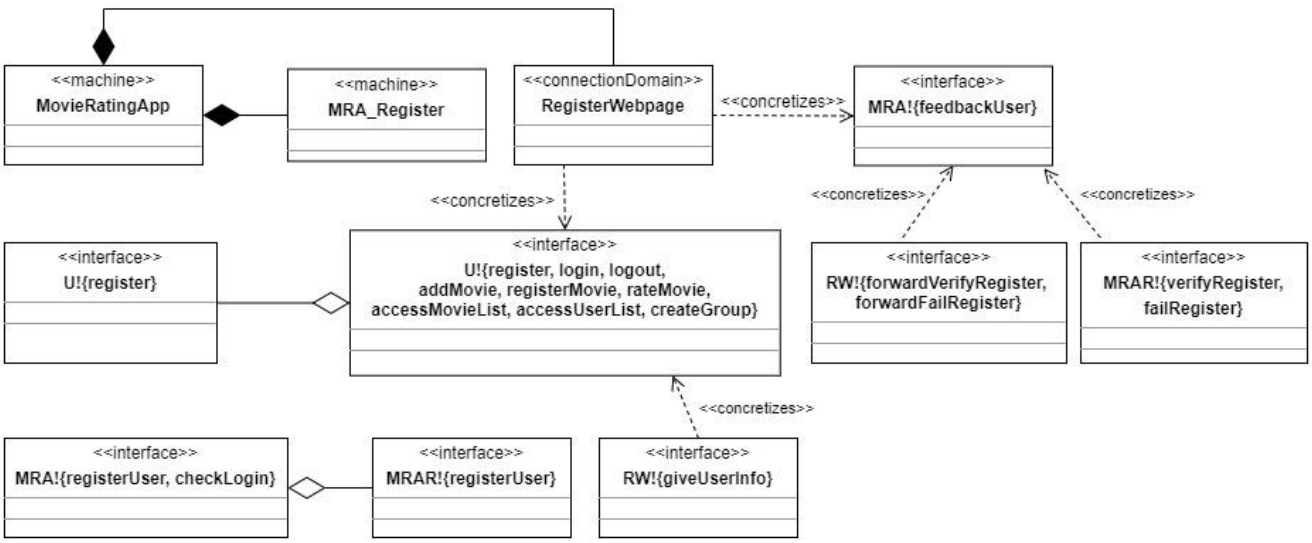
- Connection domains must have at least one observed and one controlled interface.
- For each phenomenon controlled by a connection domain, there must be at least one phenomenon controlled by one of the connected domains, i.e., observed by the connection domain.
- For each phenomenon observed by a connection domain, there must be at least one phenomenon controlled the connection domain, i.e., for each input there is an output.
    - Context diagram contains no connection domain.

A2: Problem decomposition and classification

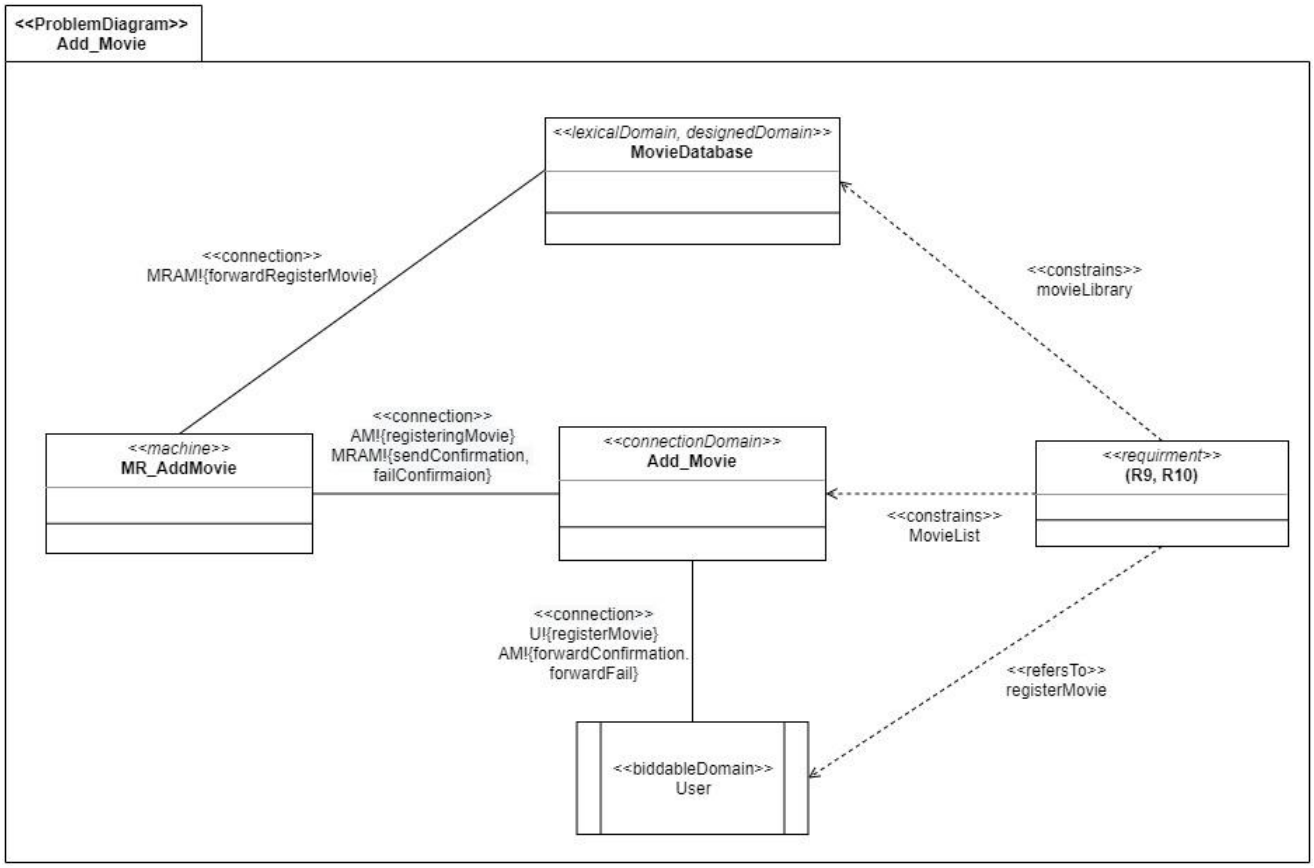
Problem Diagram 1 (Benutzerregistrierung)



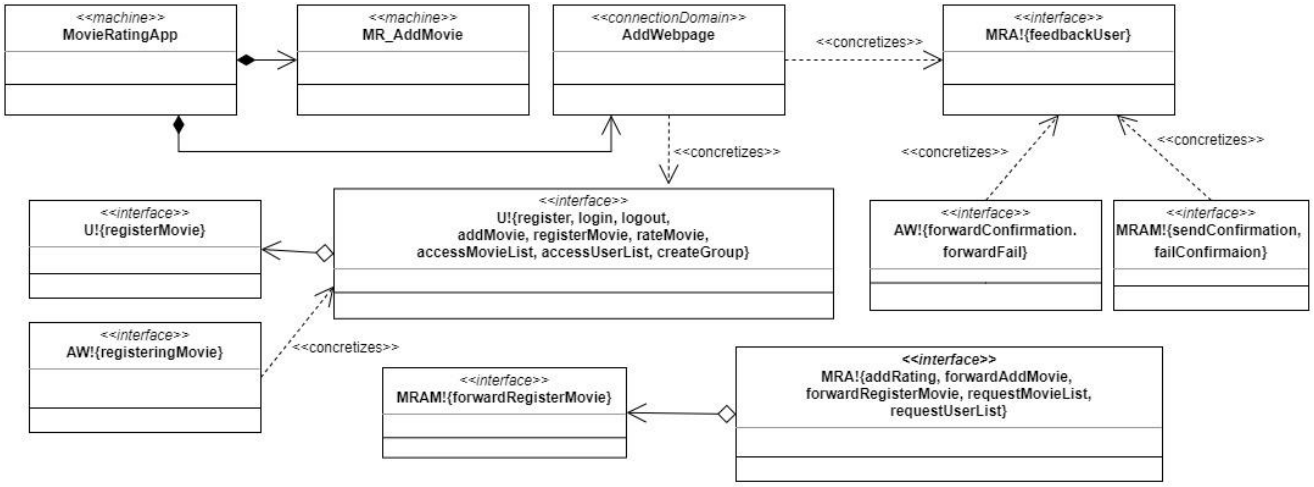
Mapping Diagram 1(Benutzerregistrierung)



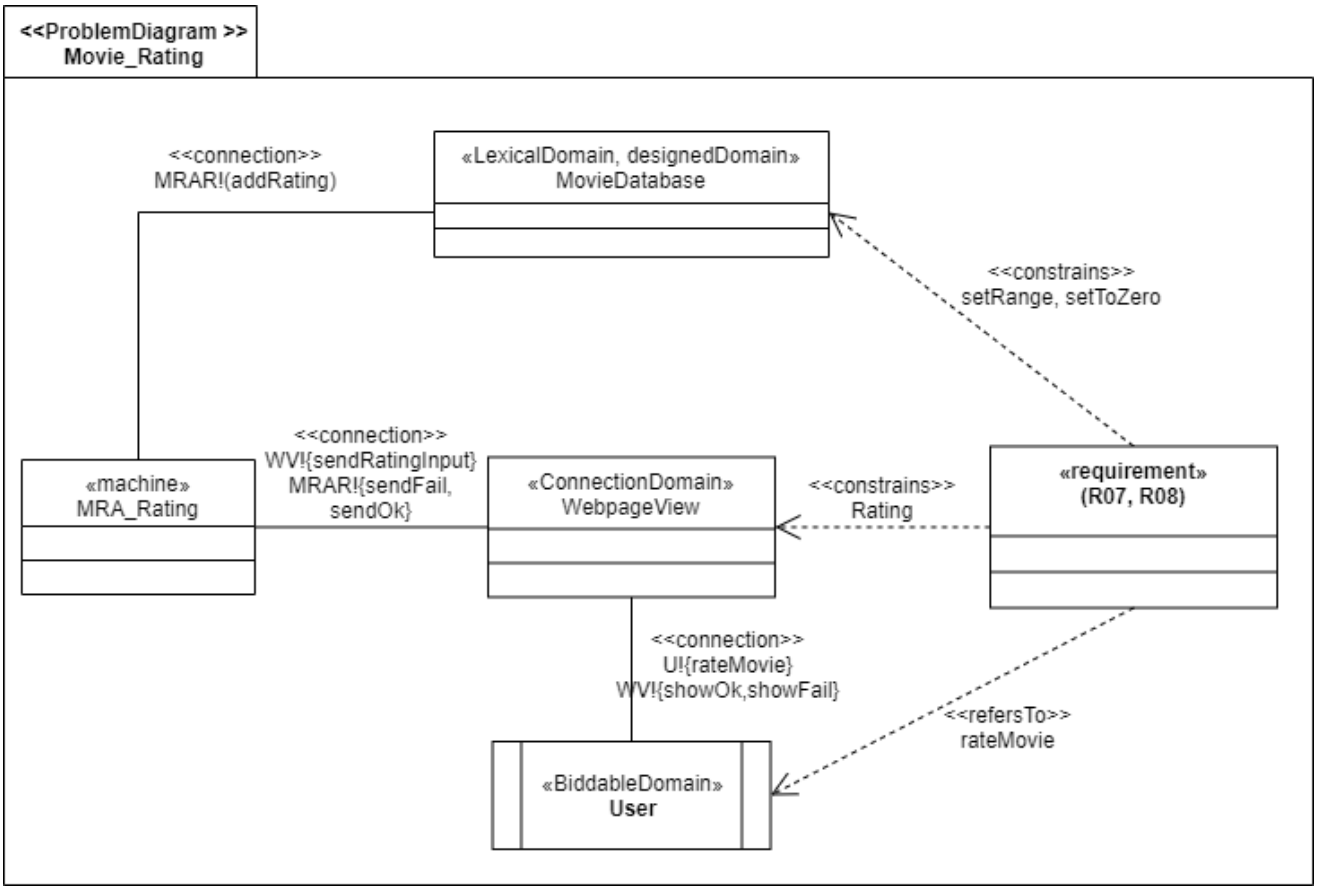
Problem Diagram 2 (Film in Datenbank erstellen)



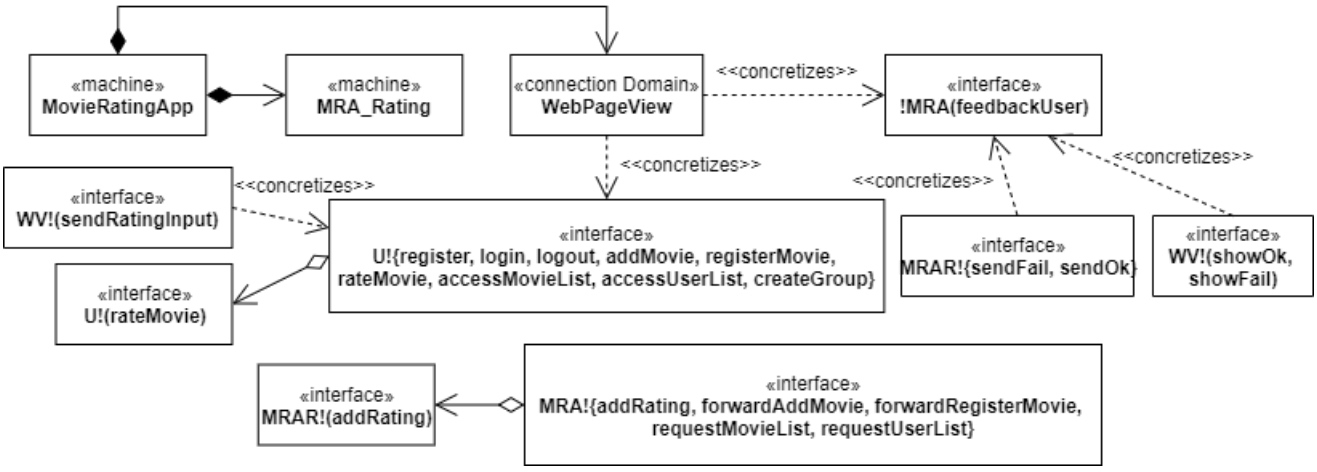
Mapping Diagram 2 (Film in Datenbank erstellen)



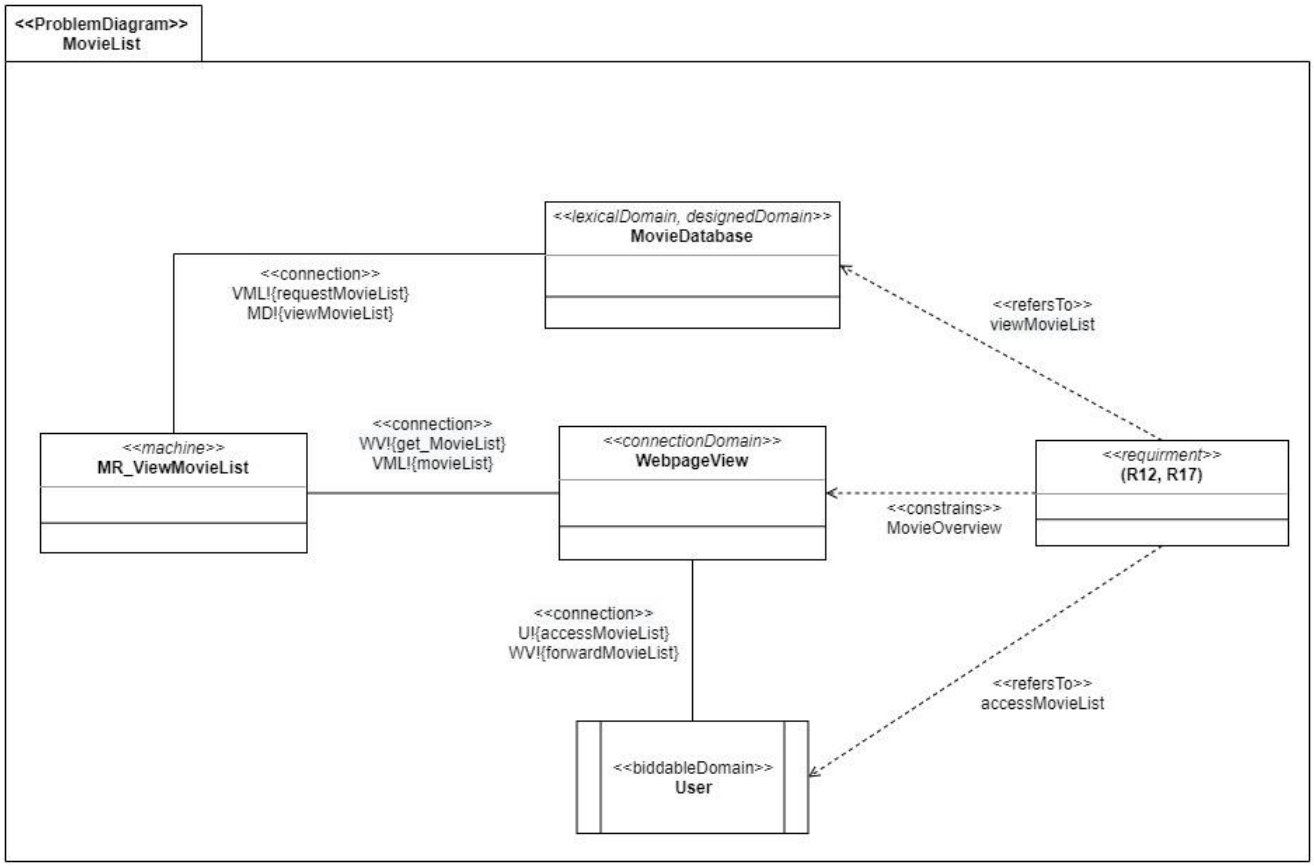
Problem Diagram 3 (Film bewerten)



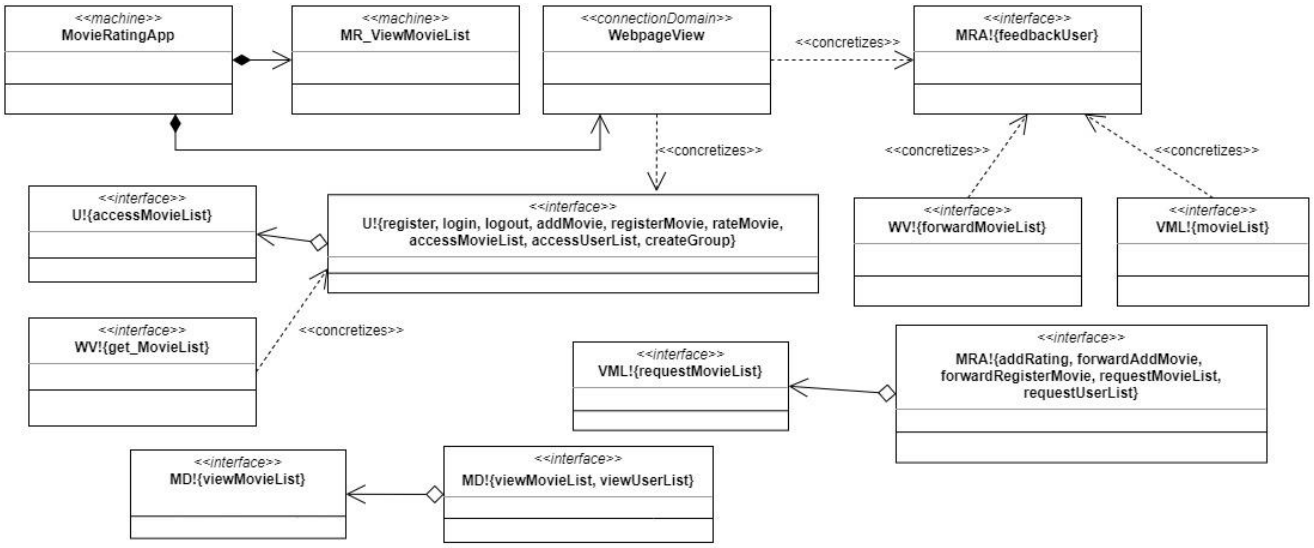
Mapping Diagram 3 (*Film bewerten*)



Problem Diagram 4 (Filmübersicht anzeigen)



Mapping Diagram 4 (Filmübersicht anzeigen)



Validation A2.1.

- A problem diagram has exactly one machine domain

requirements	covered in	contained domain	domain type	constrained	controlled phenomena
R2, R3, R4	pdRegister	MRA_Register	Machine		registerUser, verifyRegister, failRegister
		UserDatabase	Designed, Lexical	X	
		RegisterWebpage	Connection	X	giveUserInfo, forwardVerifyRegister, forwardFailRegister
		User	Biddable		register
R7, R8	pdMovieRating	MRA_Rating	Machine		sendOk, sendFail, addRating
		MovieDatabase	Designed, Lexical	X	
		WebpageView	Connection	X	showOk, showFail, sendRatingInput
		User	Biddable		rateMovie
R9, R10	pdAddMovie	MR_AddMovie	Machine		forwardRegisterMovie, sendConfirmation, failConfirmation
		MovieDatabase	Designed, Lexical	X	
		AddWebpage	Connection	X	registeringMovie, ForwardConfirmation, forwardFail
		User	Biddable		registerMovie
R12, R17	pdViewMovieList	MR_ViewMovieList	Machine		requestMovieList, movieList
		MovieDatabase	Designed, Lexical	X	viewMovieList
		WebpageView	Connection	X	requestMovieList, forwardMovieList
		User	Biddable		accessMovieList

- A problem diagram contains at least one requirement.

requirements	covered in	contained domain	domain type	constrained	controlled phenomena
R2, R3, R4	pdRegister	MRA_Register	Machine		registerUser, verifyRegister, failRegister
		UserDatabase	Designed, Lexical	X	

		RegisterWebpage	Connection	X	giveUserInfo, forwardVerifyRegister, forwardFailRegister
		User	Biddable		register
R7, R8	pdMovieRating	MRA_Rating	Machine		forwardRating, addRating
		MovieDatabase	Designed, Lexical	X	
		WebpageView	Connection	X	showOk, showFail, sendRatingInput
		User	Biddable		rateMovie
R9, R10	pdAddMovie	MR_AddMovie	Machine		forwardRegisterMovie, sendConfirmation, failConfirmation
		MovieDatabase	Designed, Lexical	X	
		AddWebpage	Connection	X	registeringMovie, ForwardConfirmation, forwardFail
		User	Biddable		registerMovie
R12, R17	pdViewMovieList	MR_ViewMovieList	Machine		requestMovieList, movieList
		MovieDatabase	Designed, Lexical	X	viewMovieList
		WebpageView	Connection	X	requestMovieList, forwardMovieList
		User	Biddable		accessMovieList

- The machine domain must control at least one interface.

requirements	covered in	contained domain	domain type	constrained	controlled phenomena
R2, R3, R4	pdRegister	MRA_Register	Machine		registerUser, verifyRegister, failRegister
		UserDatabase	Designed, Lexical	X	
		RegisterWebpage	Connection	X	giveUserInfo, forwardVerifyRegister, forwardFailRegister
		User	Biddable		register
R7, R8	pdMovieRating	MRA_Rating	Machine		sendOk, sendFail, addRating
		MovieDatabase	Designed, Lexical	X	
		WebpageView	Connection	X	showOk, showFail, sendRatingInput
		User	Biddable		rateMovie
R9, R10	pdAddMovie	MR_AddMovie	Machine		forwardRegisterMovie,

					sendConfirmation, failConfirmation
		MovieDatabase	Designed, Lexical	X	
		AddWebpage	Connection	X	registeringMovie, ForwardConfirmation, forwardFail
		User	Biddable		registerMovie
R12, R17	pdViewMovieList	MR_ViewMovieList	Machine		requestMovieList, movieList
		MovieDatabase	Designed, Lexical	X	viewMovieList
		WebpageView	Connection	X	requestMovieList, forwardMovieList
		User	Biddable		accessMovieList

- Requirements constrain at least one domain

requirements	covered in	contained domain	domain type	constrained	controlled phenomena
R2, R3, R4	pdRegister	MRA_Register	Machine		registerUser, verifyRegister, failRegister
		UserDatabase	Designed, Lexical	X	
		RegisterWebpage	Connection	X	giveUserInfo, forwardVerifyRegister, forwardFailRegister
		User	Biddable		register
R7, R8	pdMovieRating	MRA_Rating	Machine		forwardRating, addRating
		MovieDatabase	Designed, Lexical	X	
		WebpageView	Connection	X	showOk, showFail, sendRatingInput
		User	Biddable		rateMovie
R9, R10	pdAddMovie	MR_AddMovie	Machine		forwardRegisterMovie, sendConfirmation, failConfirmation
		MovieDatabase	Designed, Lexical	X	
		AddWebpage	Connection	X	registeringMovie, ForwardConfirmation, forwardFail
		User	Biddable		registerMovie
R12, R17	pdViewMovieList	MR_ViewMovieList	Machine		requestMovieList, movieList

		MovieDatabase	Designed, Lexical		viewMovieList
		WebpageView	Connection	X	requestMovieList, forwardMovieList
		User	Biddable		accessMovieList

- Requirements do not constrain machine(s).

requirements	covered in	contained domain	domain type	constrained	controlled phenomena
R2, R3, R4	pdRegister	MRA_Register	Machine		registerUser, verifyRegister, failRegister
		UserDatabase	Designed, Lexical	X	
		RegisterWebpage	Connection	X	giveUserInfo, forwardVerifyRegister, forwardFailRegister
		User	Biddable		register
R7, R8	pdMovieRating	MRA_Rating	Machine		sendOk, sendFail, addRating
		MovieDatabase	Designed, Lexical	X	
		WebpageView	Connection	X	showOk, showFail, sendRatingInput
		User	Biddable		rateMovie
R9, R10	pdAddMovie	MR_AddMovie	Machine		forwardRegisterMovie, sendConfirmation, failConfirmation
		MovieDatabase	Designed, Lexical	X	
		AddWebpage	Connection	X	registeringMovie, ForwardConfirmation, forwardFail
		User	Biddable		registerMovie
R12, R17	pdViewMovieList	MR_ViewMovieList	Machine		requestMovieList, movieList
		MovieDatabase	Designed, Lexical	X	viewMovieList
		WebpageView	Connection	X	requestMovieList, forwardMovieList
		User	Biddable		accessMovieList

- If requirements do constrain biddable domains, a good argument is given and documented.

requirements	covered in	contained domain	domain type	constrained	controlled phenomena
R2, R3, R4	pdRegister	MRA_Register	Machine		registerUser, verifyRegister, failRegister

		UserDatabase	Designed, Lexical	X	
		RegisterWebpage	Connection	X	giveUserInfo, forwardVerifyRegister, forwardFailRegister
		User	Biddable		register
R7, R8	pdMovieRating	MRA_Rating	Machine		forwardRating, addRating
		MovieDatabase	Designed, Lexical	X	
		WebpageView	Connection	X	showOk, showFail, sendRatingInput
		User	Biddable		rateMovie
R9, R10	pdAddMovie	MR_AddMovie	Machine		forwardRegisterMovie, sendConfirmation, failConfirmation
		MovieDatabase	Designed, Lexical	X	
		AddWebpage	Connection	X	registeringMovie, ForwardConfirmation, forwardFail
		User	Biddable		registerMovie
R12, R17	pdViewMovieList	MR_ViewMovieList	Machine		requestMovieList, movieList
		MovieDatabase	Designed, Lexical	X	viewMovieList
		WebpageView	Connection	X	requestMovieList, forwardMovieList
		User	Biddable		accessMovieList

- Connection domains must have at least one observed and one controlled interface.

Connection Domain	Phenomenon controlled by connection domain	Connected domain	Phenomenon controlled by connected domain
RegisterWebpage	forwardVerifyRegister, forwardFailRegister	User	register
	giveUserinfo	MRA_Register	verifyRegister, failRegister
AddWebpage	ForwardConfirmation, forwardFail	User	registerMovie
	registerMovie	MR_AddMovie	sendConfirmation,failConfirmation
Webpage_Movie	showOk, showFail	User	rateMovie
	sendRatinginput	MRA_Rating	sendOk, sendFail
View	forwardMovieList	User	accessMovieList
	requestMovieList	MR_ViewMovieList	MovieList

- For each phenomenon controlled by a connection domain, there must be at least one phenomenon controlled by one of the connected domains, i.e. observed by the connection domain.

Connection Domain	Phenomenon controlled by connection domain	Connected domain	Phenomenon controlled by connected domain
RegisterWebpage	forwardVerifyRegister, forwardFailRegister	User	register
	giveUserinfo	MRA_Register	verifyRegister, failRegister
AddWebpage	ForwardConfirmation, forwardFail	User	registerMovie
	registerMovie	MR_AddMovie	sendConfirmation,failConfirmation
Webpage_Movie	showOk, showFail	User	rateMovie
	sendRatinginput	MRA_Rating	sendOk, sendFail
View	forwardMovieList	User	accessMovieList
	requestMovieList	MR_ViewMovieList	MovieList

- For each phenomenon observed by a connection domain, there must be at least one phenomenon controlled by the connection domain, i.e. for each input there is an output.

Connection Domain	Phenomenon controlled by connection domain	Phenomenon observed by connection domain
RegisterWebpage	forwardVerifyRegister, forwardFailRegister	register
	giveUserinfo	verifyRegister, failRegister
AddWebpage	ForwardConfirmation, forwardFail	registerMovie
	registerMovie	sendConfirmation,failConfirmation
Webpage_Movie	showOk, showFail	rateMovie
	sendRatinginput	sendOk, sendFail
View	forwardMovieList	accessMovieList
	requestMovieList	MovieList

- The problem diagrams must be consistent to the context diagram, e.g. each machine of the problem diagrams is a part of the context diagram machine.

Provided mapping diagrams

- All subproblems can be derived from the context diagram by means of decomposition operators.

Problem Diagram	operator	Related domains or phenomena
Register	Leave out domain	MovieDatabase, Group, GroupMember, Administrator
	Introduce connection domain	RegisterWebpage
	Split interface	U!{register,…}, MRA{registerUser,checkLogin}
	Concretize interface	U!{register,…}, MRA!{feedbackUser}
AddWebpage	Leave out domain	UserDatabase, Group, GroupMember, Administrator
	Introduce connection domain	AddWebpage
	Split interface	U!{register,…}, MRA!{ addRating,…}
	Concretize interface	U!{register,…}, MRA!{feedbackUser}
Movie_Rating	Leave out domain	UserDatabase, Group, GroupMember, Administrator
	Introduce connection domain	WebpageView
	Split interface	U!{register,…}, MRA!{ addRating,…}
	Concretize interface	U!{register,…}, MRA!{feedbackUser}
R4	Split domain	UserDatabase, Group, GroupMember, Administrator
	Introduce connection domain	View
	Split interface	U!{register,…}, MRA!{ addRating,…}
	Concretize interface	U!{register,…}, MRA!{feedbackUser}

Problem Frames:

R1 & R2 & R3 Problem Frame

(R1), (R3) and (R2) fit to **update II**
Because the operator can change, add or update in the Database and there is a Connection Domain between the Machine and the Operator which forwarding requests and receiving Responses.
(Lexical Domain: constrains, Connection Domain: constrains, Biddable Domain: refers to)
R1: The Operator can register a new User and forward his/her info to the Database.
R2: The Operator can add new movie to the Database, and the user will receive feedback whether the adding process is succeeded or failed.
R3: The Operator can add new rating to the Database, and the user will receive feedback of the movie rating.

R4 Problem Frame

(R4) fits to **query II**
We have our requirement that do not change the database and there is one connection domain for the interaction between the machine and the operator
(Lexical Domain: refers to, Connection Domain: constrains, Biddable Domain: refers to)
Furthermore, we do not change or add something. The user wants to get the Movie list and the machine should just show us the Movie list.

Validation A2.2.

A problem diagram must be consistent to its problem frame.

- All connections in a problem diagram correspond to a connection in the frame diagram (connects same domain types).

Problem Diagram	Problem Frame	Connection in PD	Connection in PF	Domain Type 1	Domin Type 2
Register	Update (2)	U!{register} RW!{feedbackUser}	UO!E6 IOD!C7	biddable domain	Connection domain
		MRAR!{verifyRegister, failRegistration} RW!{giveUserinfp}	UM!E4 IOD!E8	machine	Connection domain
		MRAR!{registerUser}	UM!E2	machine	lexical domain
Add_Movie	Update(2)	U!{registerMovie} AM!{ForwardConfirmation, forwardFail	UO!E6 IOD!C7	Biddable domain	Connection domain
		MRAM!{sendConfirmation, failConfirmation} AM!{registeringMovie}	UM!E4 IOD!E8	machine	Connection domain
		MRAM!{forwardRegisterMovie}	UM!E2	machine	lexical domain
Movie_Rating	Update(2)	U!{rateMovie} WV!{shareRating}	UO!E6 IOD!C7	Biddable doman	Connection domain
		MRAR!{sendOk, sendFail} WV!{sendRatinginput}	UM!E4 IOD!E8	machine	Connection domain
		MRAR!{addRating}	UM!E2	machine	Lexical domain
R4	Query(2)	U!{accessMovieList} WV!{forwardMovieList}	EO!E5 IOD!E7	Biddable domain	Connection domain
		VML!{movieList} WV!{requestMovieList}	QM!Y3 IOD!C6	machine	Connection domain
		MD!{viewMovieList}	DBY1	Lexical domain	machine

- The domain types of constrained domains in the problem diagram are the same as in the frame diagram.

Problem Diagram	Problem Frame	Constrained domains in PD	Constrained domains in PF	Domain Type
Register	Update(2)	UserDatabase	Database	Lexical domain
		RegisterWebpage	Input Output Device	Connection domain

Add_Movie	Update(2)	MovieDatabase	Database	Lexical domain
		AddWebpage	Input Output Device	Connection domain
Movie_Rating	Update(2)	MovieDatabase	Database	Lexical domain
		WebpageView	Input Output Device	Connection domain
MovieList	Query(2)	View	Input Output Device	Connection domain

- Each referred domain in the problem frame corresponds to a domain in the problem diagram.

Problem Diagram	**Problem Frame**	**Referred domains in PD**	**Referred domains in PF**	**Domain Types**
Register	Update(2)	User	UpdateOperator	Biddable domain
Add_Movie	Update(2)	User	UpdateOperator	Biddable domain
Movie_Rating	Update(2)	User	UpdateOperator	Biddable domain
MovieList	Query(2)	User	EnquiryOperator	Biddable domain
		MovieDatabase	Database	Lexical domain

--->All problem diagrams are consistent to their problem frames.

A3: Abstract Software Specification

(R1) Register Movie:

- R02: During the registration process, he/she must provide an email address, his/her age and a username.
- R03: The username has to be unique.
- R04: To register, a person must be at least eighteen years old, otherwise the registration fails.

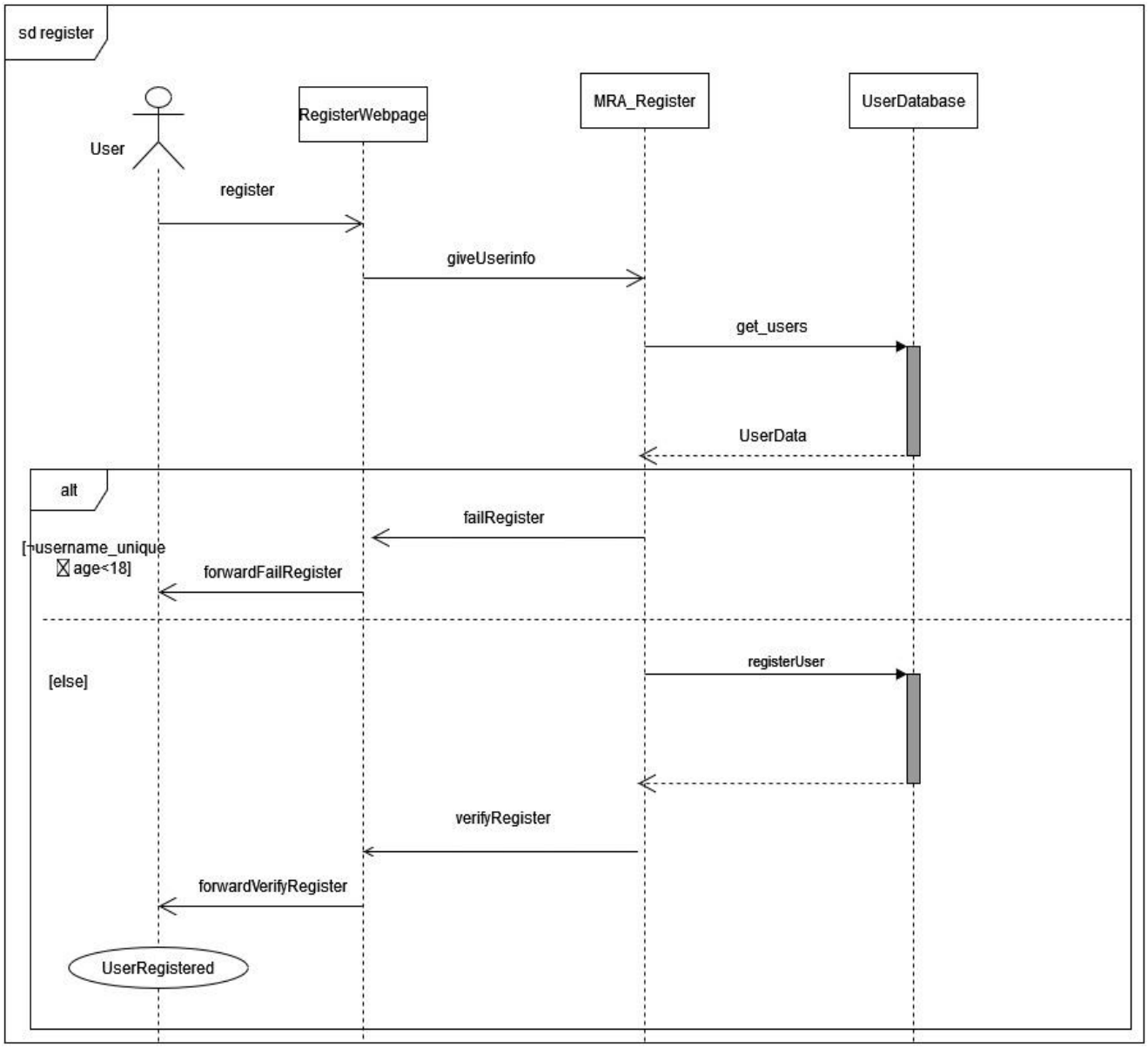
From problem diagram 1, we can derive the following specifications:
RegisterWebpage (S01a): When the webpage receives the command “register”, it forwards this command to MRA_Register with the command “giveUserinfo”. The feedback whether the registration process succeeded or failed is received via the commands “verifyRegister” and “failRegister”. Both feedback types are shown to the User with the commands “forwardVerifyRegister” and “forwardFailRegister”.

MRA_Register (S01b): When the command “giveUserinfo” is received. It is first checked if the username is unique and the age is bigger than or equal to Eighteen. If this is the case, MRA_Register forwards the user information to UserDatabase via the command “registerUser”. Then informs the user with the command “verifyRegister ” sent to RegisterWebpage. Otherwise, it will directly inform the user of the failure of registration by sending command “failRegister” to RegisterWebpage.

UserDatabase (S01c): When the command “registerUser” is received, user data are added to UserDatabase.

Correctness condition: (S01a) ∧ (S01b) ∧(S01c) → (R02/R03/R04)

User Registration Sequence Diagram:



(R2) Adding a Movie in the Database:-

If a movie is not yet contained in the database, the user can add it by providing the title, director, main actors (at least one) and original publishing date. Each movie can be contained only once in the database.

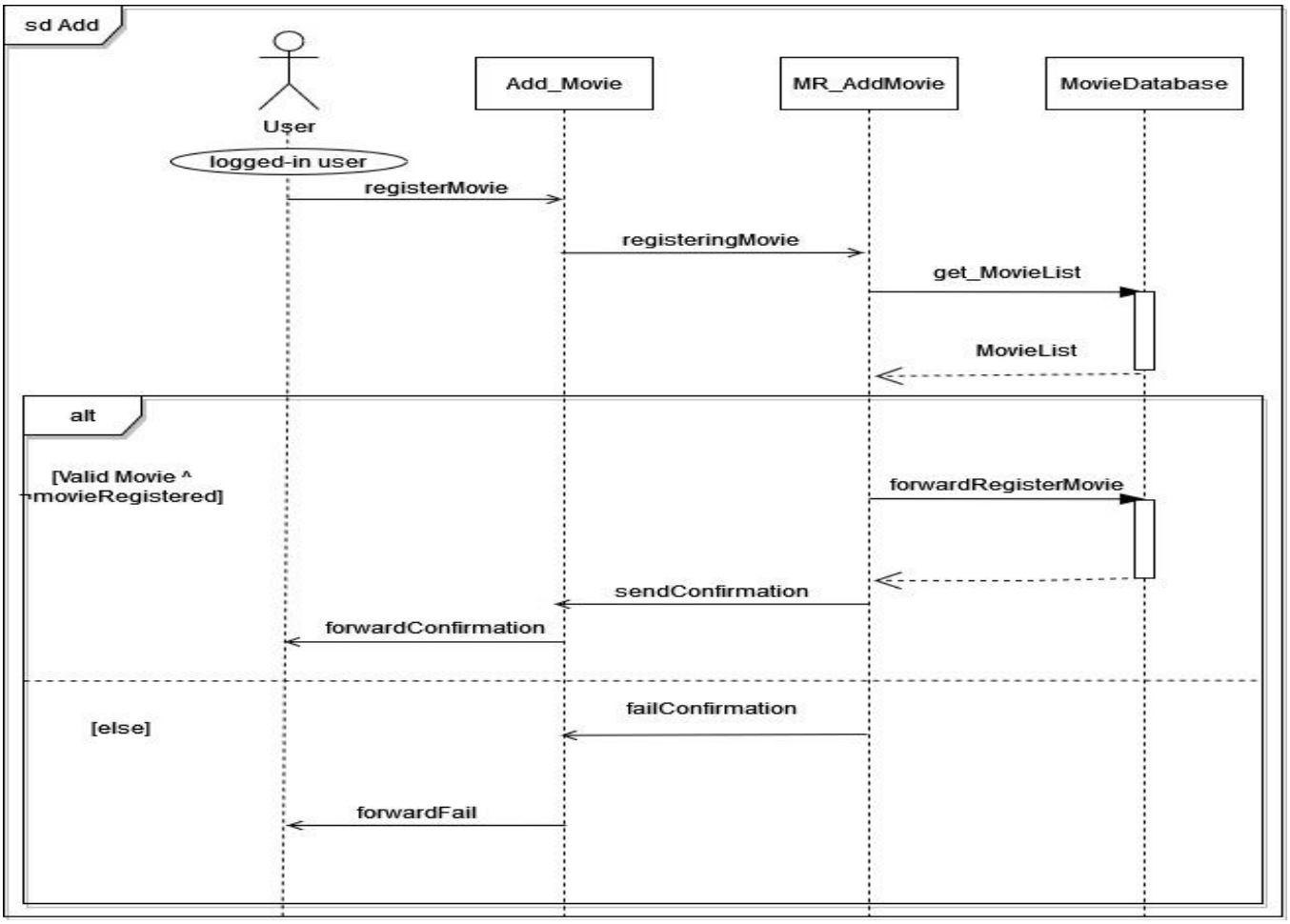
Using the domain knowledge:-
Valid Movie (A2) Users only add new movies that really exist and movie data that is valid.
Movie Data (F1) Each movie has a title, a director, at least one main actor and an original publishing date.

We can derive the specifications:-
Add_Movie (S2a): When the connection Domain receives the command "registerMovie", this command is forwarded to the Machine with the command "registeringMovie". The feedback whether the request was confirmed or not is received via the commands "sendConfirmation" and "failConfirmation". Both feedback types are shown to the User via the commands "forwardConfirmation" and "forwardFail".

MR_AddMovie (S2b): When the machine receives the command "registeringMovie", the selected movie will be added to the Movie Database with the command "forwardRegisterMovie". Then the feedback whether the adding process is succeeded or failed will be forwarded to the user through the connection domain via the commands "sendConfirmation" and "failConfirmation".
MovieDatabase (S2c): When receiving the command "forwardRegisterMovie", the selected movie will be added to the Movie Database.

Correctness condition: (S02a) ∧ (S02b) ∧ (S02c) ∧ (F1) ∧ (A2) ⇒ (R9/R10).

Adding Movie Sequence Diagram:

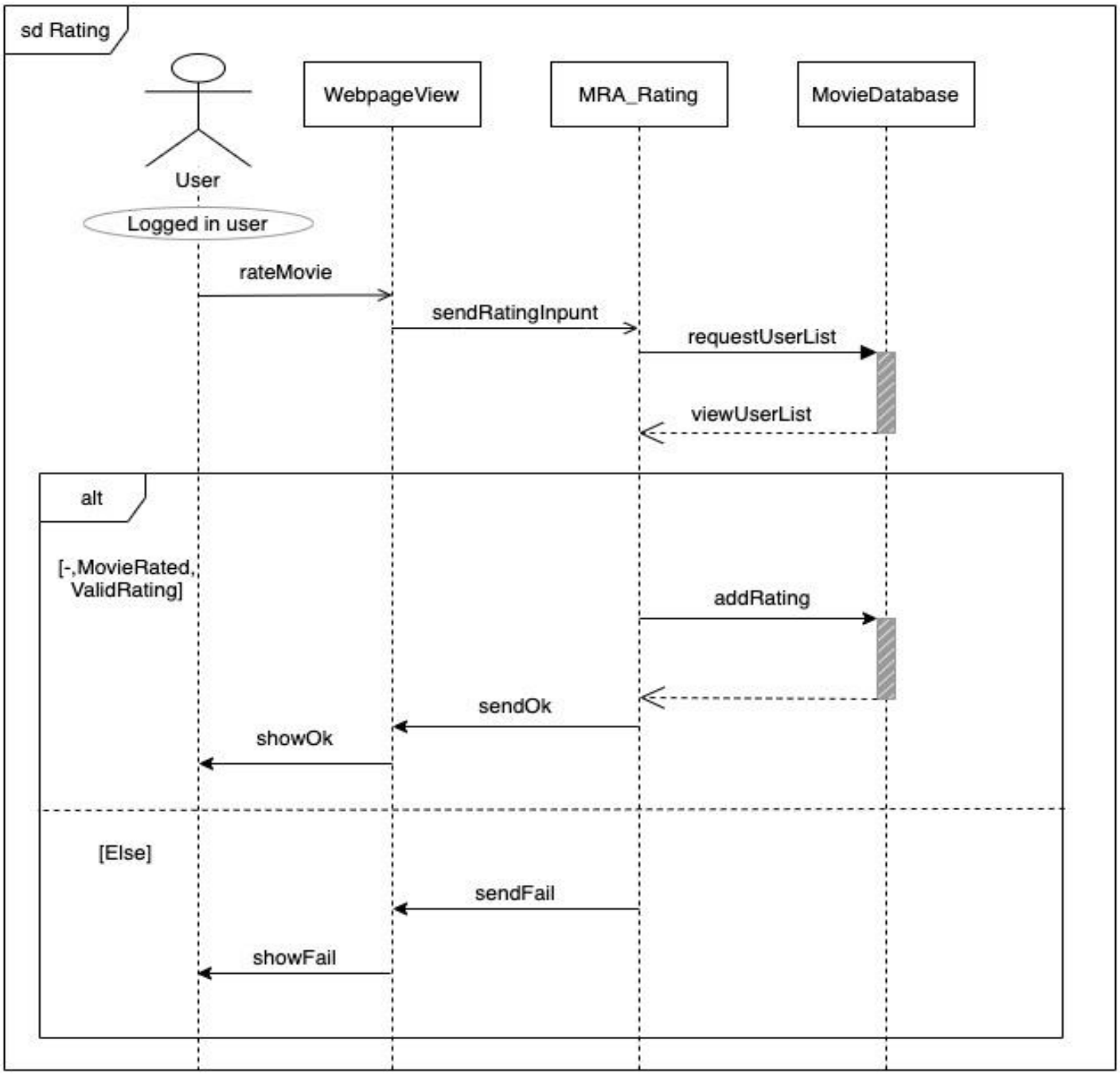


(R3) Rate Movie:

- R7: A rating consists out of a mandatory point rating from one to ten (1 = very bad movie, 10 = excellent movie) and optionally a written comment.
- R8: If a value differing from one to ten is entered, the rating process will fail. Movies without any rating are rated as zero.

From problem diagram 3, we can derive the following specifications:
Specification for Problem Diagram 3 (Movie Rating)
WebpageView (S03a): When the WebpageView receives the command “rateMovie”, then the command is forwarded to the machine with the command “sendRatingInput”. The WebpageView then receives the the commands “sendOk” or “sendFail” and according to that the WebpageView then sends the command “showOk” or “showFail” to the user, so the user can validate if the right rating was entered to the database and if the check according to the stated rules in the requirements were correct.
MRA_Rating (S03b): When the machine receives the command “sendRatingInput” from the WebpageView, the rating will be checked, that is if it’s between 1 to 10 (1 = very bad movie, 10 = excellent movie) and will be handled according as follows: If a value differing from one to ten is entered the rating process will fail. If it was valid the machine then sends a command “addRating” to the database, which adds the validated rating to the MovieDatabase. If it was not nothing will happen. The machine then sends the feedback in form of the command “sendOk” if the rating was valid or “sendFail” if it was not.
MovieDatabase (S03c): A movie without any rating will have a rating zero. This is made sure by initializing the rating of all new movie entries with 0. When MovieDatabase receives addRating it adds the new rating to the movie.
Correctness condition: (S03a) ∧ (S03b) ∧ (S03c) => (R07/R08)

Movie rating Sequence Diagram:



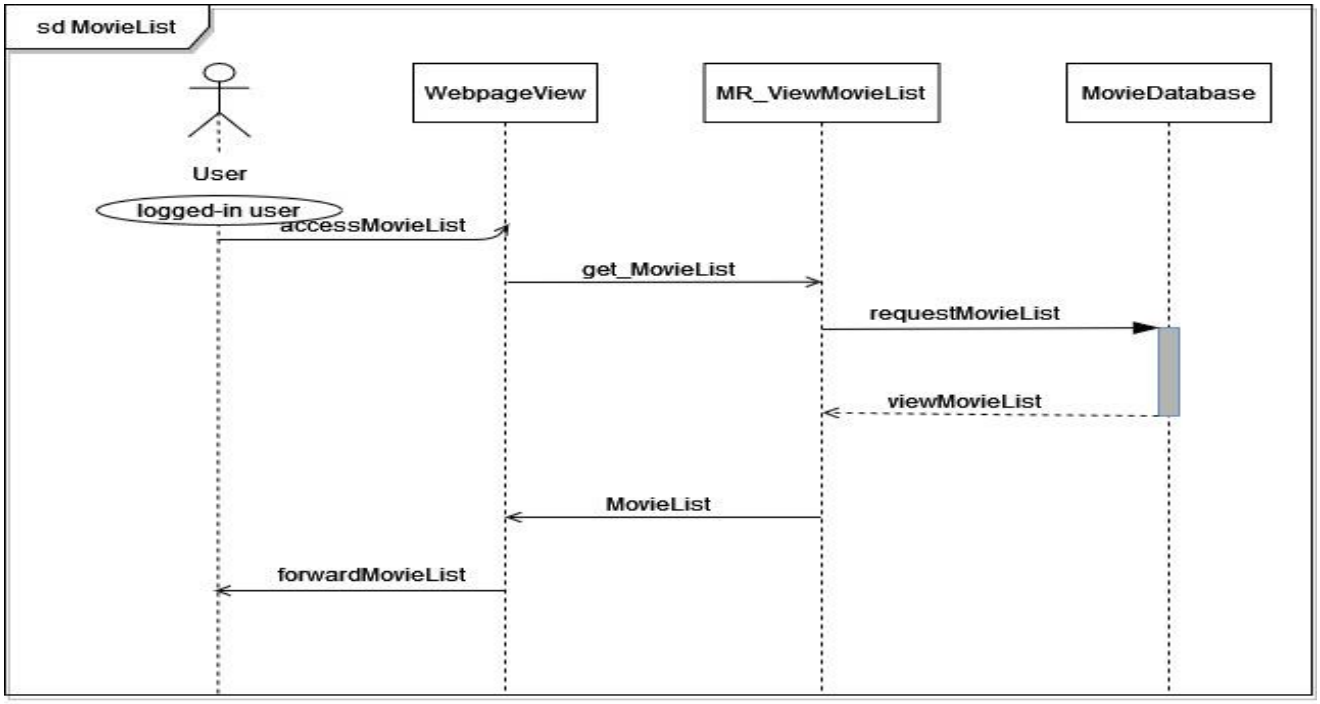
(R4) View the List of Movies to the User:-

Users can access a list of all movies in the database sorted by rating in descending order.

We can derive the specifications:-
View (S04a): when the connection Domain receives the command “accessMovieList” by the User, then the command is forwarded to the machine with the command “get_MovieList”. The results are received via the command “movieList” and shown to the User by “forwardMovieList”.
MR_ViewMovieList (S04b): When the machine receives the command “get_MovieList”, the Movies are selected with the command “requestMovieList” and received as the data “viewMovieList”. The results are returned via the command “movieList”.
MovieDatabase (S04c): After receiving the command “requestMovieList” the results are returned as the data “viewMovieList”.

Correctness condition: (S04a) ∧ (S04b) ∧ (S04c) ∧ (F1) ⇒ (R4).

View Movie List Sequence Diagram:



Remark: Result may be empty.

Validation A3:

- $S_{abstract}$ ∧ D are non-contradictory.
No contradictions can be found in $S_{abstract}$ ∧ D.
- $S_{abstract}$ ∧ D ⇒ R.

R1 (S01a) ∧ (S01b) ∧ (S01c) => (R02/R03/R04)
R2 (S02a) ∧ (S02b) ∧ (S02c) ∧ (F1) ∧ (A2) ⇒ (R9/R10)
R3 (S03a) ∧ (S03b) ∧ (S03c) => (R07/R08)
R4 (S04a) ∧ (S04b) ∧ (S04c) ∧ (F1) ⇒ (R4).

-Messages and phenomena are consistent.

message in scenario	source	target	phenomena in problem diagram
register	User	RegisterWebpage	U!{register}
giveUserinfo	RegisterWebpage	MRA_Register	RW!{giveUserinfo}
failRegister	MRA_Register	RegisterWebpage	MRAR!{failRegister}
forwardFailRegister	RegisterWebpage	User	RW!{forwardFailRegister}
registerUser	MRA_Register	UserDatabase	MRAR!{registerUser}
verifyRegister	MRA_Register	RegisterWebpage	MRAR!{verifyRegister}
forwardVerifyRegister	RegisterWebpage	User	RW!{forwardVerifyRegister}
registerMovie	User	Add_Movie	U!{registerMovie}
registeringMovie	Add_Movie	MR_AddMovie	AM!{registeringMovie}
forwardRegisterMovie	MR_AddMovie	MovieDatabase	MRAM!{forwardRegisterMovie}
sendConfirmation	MR_AddMovie	Add_Movie	MRAM!{sendConfirmation}
forwardConfirmation	Add_Movie	User	AM!{forwardConfirmation}
failConfirmation	MR_AddMovie	Add_Movie	MRAM!{failConfirmation}
forwardFail	Add_Movie	User	AM!{forwardFail}
rateMovie	User	WebpageView	U!{rateMovie}
sendRatingInput	WebpageView	MRA_Rating	WV!{sendRatingInput}
addRating	MRA_Rating	MovieDatabase	MRAR!{addRating}
sendOk	MRA_Rating	WebpageView	MRAR!{sendOk}
showOk	WebpageView	User	WV!{showOk}
sendFail	MRA_Rating	WebpageView	MRAR!{sendFail}
ShowFail	WebpageView	User	WV!{showFail}
accessMovieList	User	WebpageView	U!{accessMovieList}
get_MovieList	WebpageView	MR_ViewMovieList	WV!{get_MovieList}
requestMovieList	MR_ViewMovieList	MovieDatabase	VML!{requestMovieList}
MovieList	MR_ViewMovieList	WebpageView	VML!{movieList}
forwardMovieList	WebpageView	User	WV!{forwardMovieList}

- Lexical domains are not sources of messages.

message in scenario	source	domain type
register	User	biddableDomain
giveUserinfo	RegisterWebpage	connectionDomain
failRegister	MRA_Register	machine
forwardFailRegister	RegisterWebpage	connectionDomain
registerUser	MRA_Register	machine
verifyRegister	MRA_Register	machine

forwardVerifyRegister	RegisterWebpage	connectionDomain
registerMovie registeringMovie forwardRegisterMovie sendConfirmation forwardConfirmation failConfirmation forwardFail	User Add_Movie MR_AddMovie MR_AddMovie Add_Movie MR_AddMovie Add_Movie	biddableDomain connectionDomain machine machine connectionDomain machine connectionDomain
rateMovie sendRatingInput addRating sendOk showOk sendFail ShowFail	User WebpageView MRA_Rating MRA_Rating WebpageView MRA_Rating WebpageView	biddableDomain connectionDomain machine machine connectionDomain machine connectionDomain
accessMovieList get_MovieList requestMovieList MovieList forwardMovieList	User WebpageView MR_ViewMovieList MR_ViewMovieList WebpageView	biddableDomain connectionDomain machine machine connectionDomain

- There exists at least one scenario for each subproblem.
- Scenarios cover normal cases and possibly exceptional cases.

subproblem	normal case	Exceptional case
pdRegister	sdRegister	-
pdAdd	sdAdd	-
pdRating	sdRating	-
pdMovieList	sdMovieList	-

A4: Technical software specification

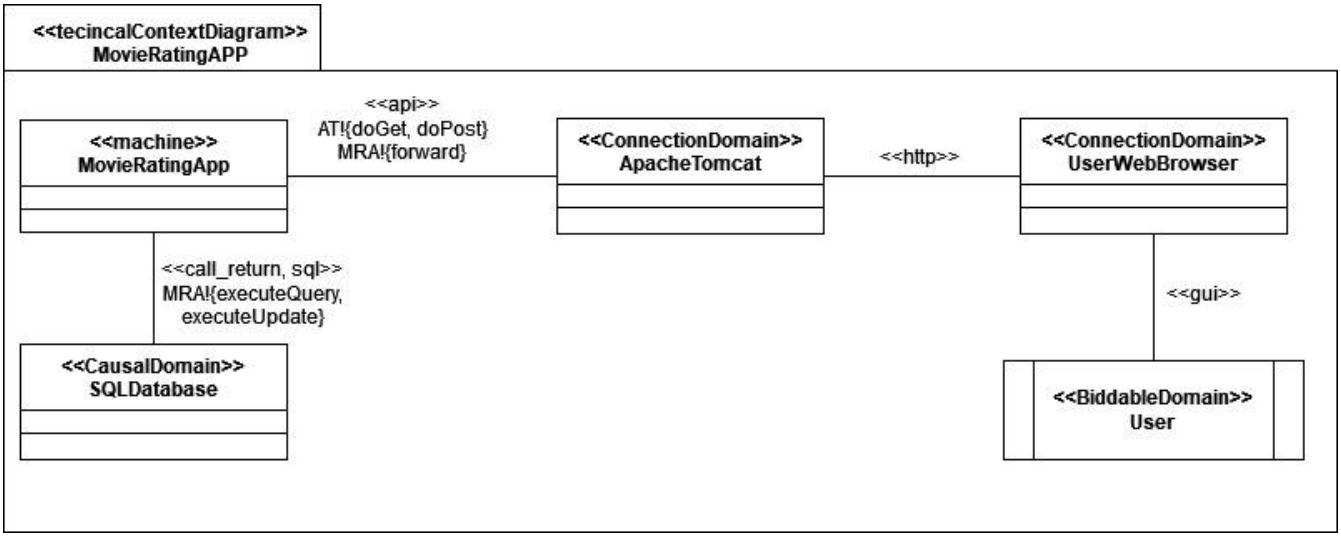
Technical Software Specification

MovieRatingApp: Realized as SQLDatabase on the same computer as the machine. Therefore, the database is connected by a call-and-return interface and used with SQL commands.

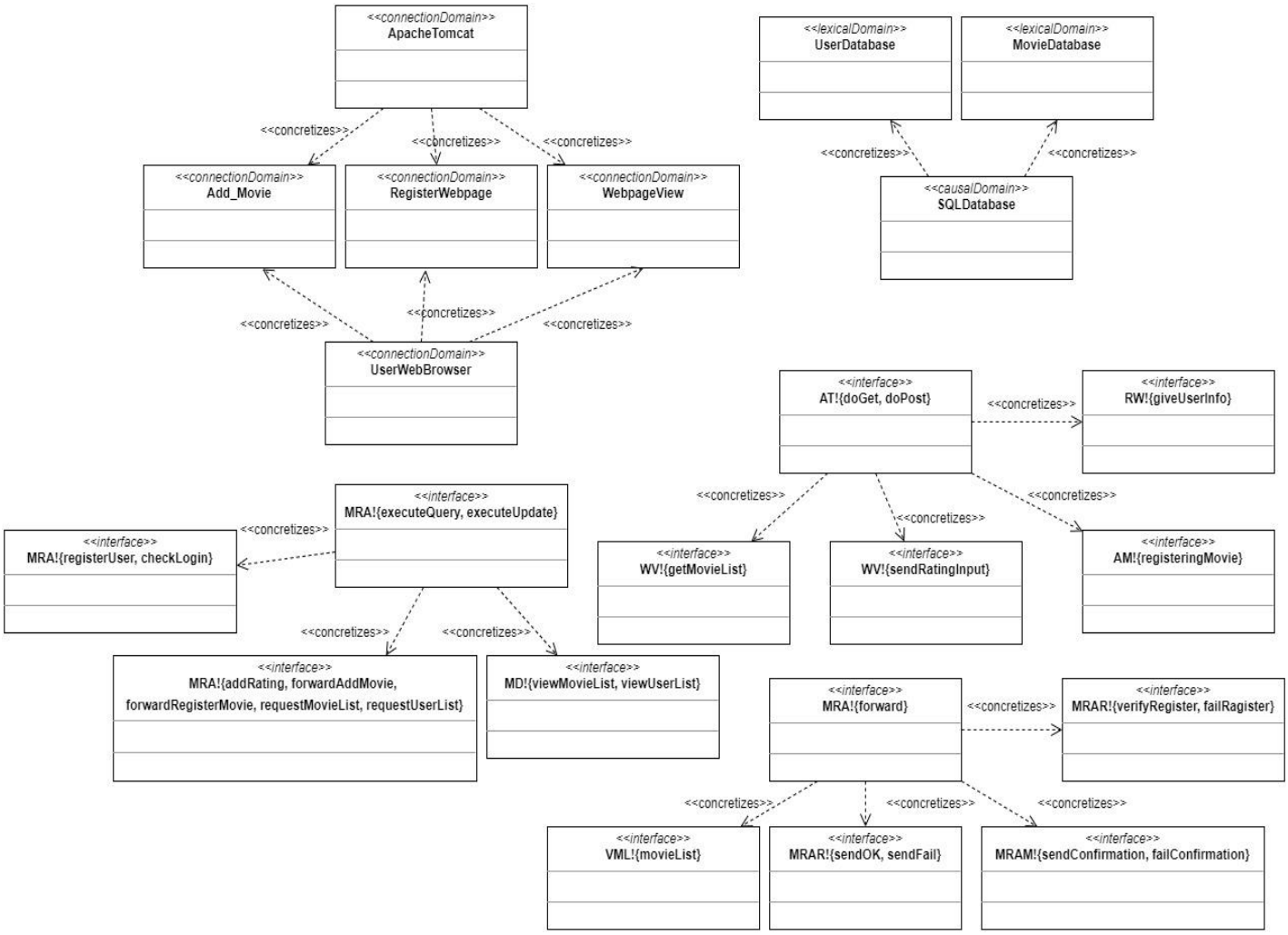
ViewWebpage: Realized using ApacheTomcat and UserWebpage We decide to use ApacheTomcat as a server platform, because the customer realized other projects on this platform and requires a Java implementation.

We decide to use ApacheTomcat as a server platform, because the customer realized other projects on this platform and requires a Java implementation.

Techincal Context Diagram



Mapping Techincal Context Diagram



Validation A4:

- New phenomena and domains are suitable to implement the external messages used in the abstract phenomena:

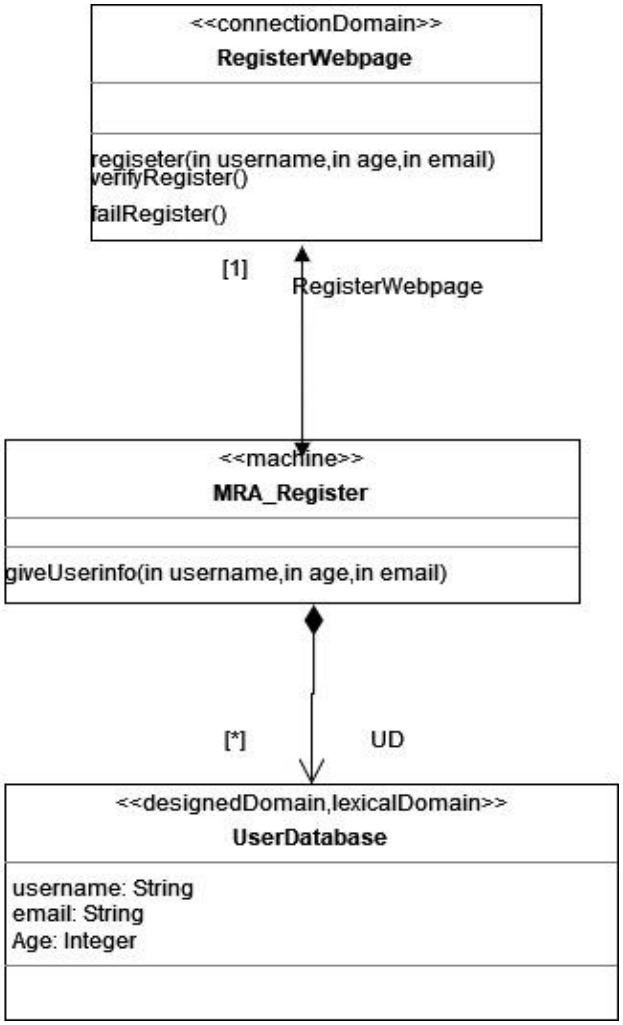
Message	new phenomena and domains
giveUserInfo	ApacheTomcat, HTTP
verifyRegister, failRegister	ApacheTomcat, HTTP
registeringMovie	ApacheTomcat, HTTP
sendConfirmation, failConfirmation	ApacheTomcat, HTTP
sendRatingInput	ApacheTomcat, HTTP
SendFail,sendOk	ApacheTomcat, HTTP
get_MovielIst	ApacheTomcat, HTTP
movieList	ApacheTomcat, HTTP
register	UserWebBrowser,gui
ForwardVerifyRegister,forwardFailRegister	UserWebBrowser,gui
tegisterMovie	UserWebBrowser,gui
ForwardConfirmation,forwardFail	UserWebBrowser,gui
rateMovie	UserWebBrowser,gui
ShowOk,showFail	UserWebBrowser,gui
accessMovieList	UserWebBrowser,gui
forwardMovieList	UserWebBrowser,gui

- All internal messages can be realized using SQL commands.
- All domains of the technical context diagram are related to domains in the problem diagrams:
- All phenomena in the technical context diagram are related to elements in the problem diagrams:
    - Provided mapping diagram
- All domains directly connected with the machine in the problem diagrams are related to elements in the technical context diagram:

Problem Diagram	Domain connected with the machine	Element in the TCD
Register	RegisterWebpage	UserWebpage, ApacheTomcat
	UserDatebase	SQLDatabase
Add_Movie	Add_Movie	UserWebpage, ApacheTomcat
	MovieDatabase	SQLDatabase
Movie_Rating	WebpageView	UserWebpage, ApacheTomcat
	MovieDatabase	SQLDatabase
MR_viewMovieList	WebpageView	UserWebpage, ApacheTomcat
	MovieDatabase	SQLDatabase

A5 Operations and Data Specification:

1. User Registration Class Model:



1.1. Register Operation Specifcation:

Name: register
Description: forwards register request from the user's browser to the machine.

Context RegisterWebpage::register(username: String, age: Integer ,email: String)

Pre: True

Post: MRA_Register^giveUserinfo(username: String, age: Integer ,email: String)

1.2. giveUserinfo Operation Specification:

Name: giveUserinfo

Description: registers user's information in UserDatabase and send to RegisterWebpage verifyRegister() (success) or it fails and sends to RegisterWebgape failRegister().

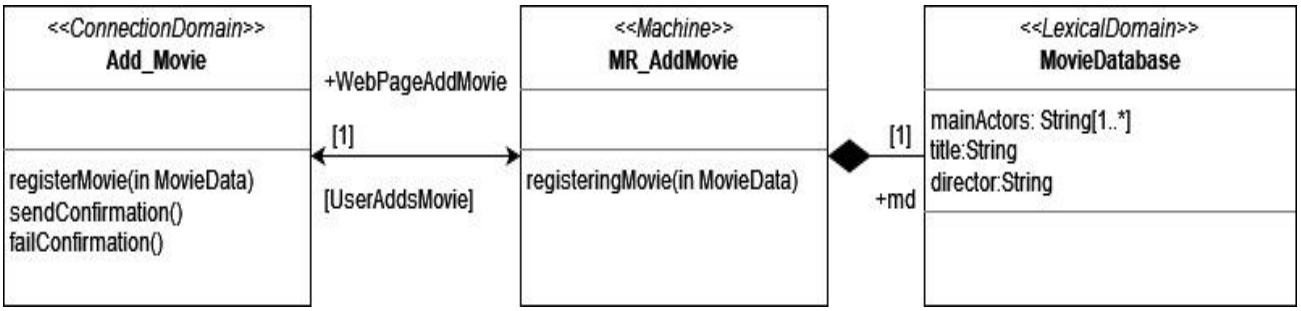
```
Context MRA_Register::giveUserinfo(username: String, age: Integer ,email: String)
Pre:True
Post:
If UD@pre->exists(user: UserDatabase|user.username=username) or age<18
                Then RegisterWebpage^failRegister()
                Else  UD.->one(user: UserDatabase|user.username=username and
user.age=age and user.email=email)  and UD->size()=UD@pre->size()+1
                                and RegisterWebpage^verifyRegister()
```

Remark: verifyRegister() and failRegister() are not specified here since they are feedback functions and therefore considered trivial.

1.3. OCL Constraint:

```
Context UserDatabase
Inv: UserDatabase.allInstances()->isUnique(username) and
UserDatabase.allInstances().age>=18
```

2. Register Movie Class Model



2.1. registerMovie Operation Specification:

Name: registerMovie
Description: forwards the request from the user to the machine.

context: Add_Movie::registerMovie(mainActors: String[0..*], title: String, director: String)

pre: true

post:

MR_AddMovie^registerMovie(mainActors: String[0..*], title: String, director: String)

2.2. registeringMovie Operation Specification:

Name: registeringMovie
Description: User adds a movie with the certain data to the database and an ok or fail message will be returned to the user.

Context:MR_AddMovie::registeringMovie(mainActors: String[0..*], title: String, director: String)

Pre: true

Post: if md@pre->exists(movie: MovieDatabase|movie.title=title and movie.director=director and movie.mainActors=maiActors)

Then WebpageAddMovie^failConfirmation()

Else md.allInstances()->one(movie:
MovieDatabase|movie.mainActors=mainActors and movie.title=title and movie.director=director)

md.allInstance->size()=md.allInstance@pre->size()+1

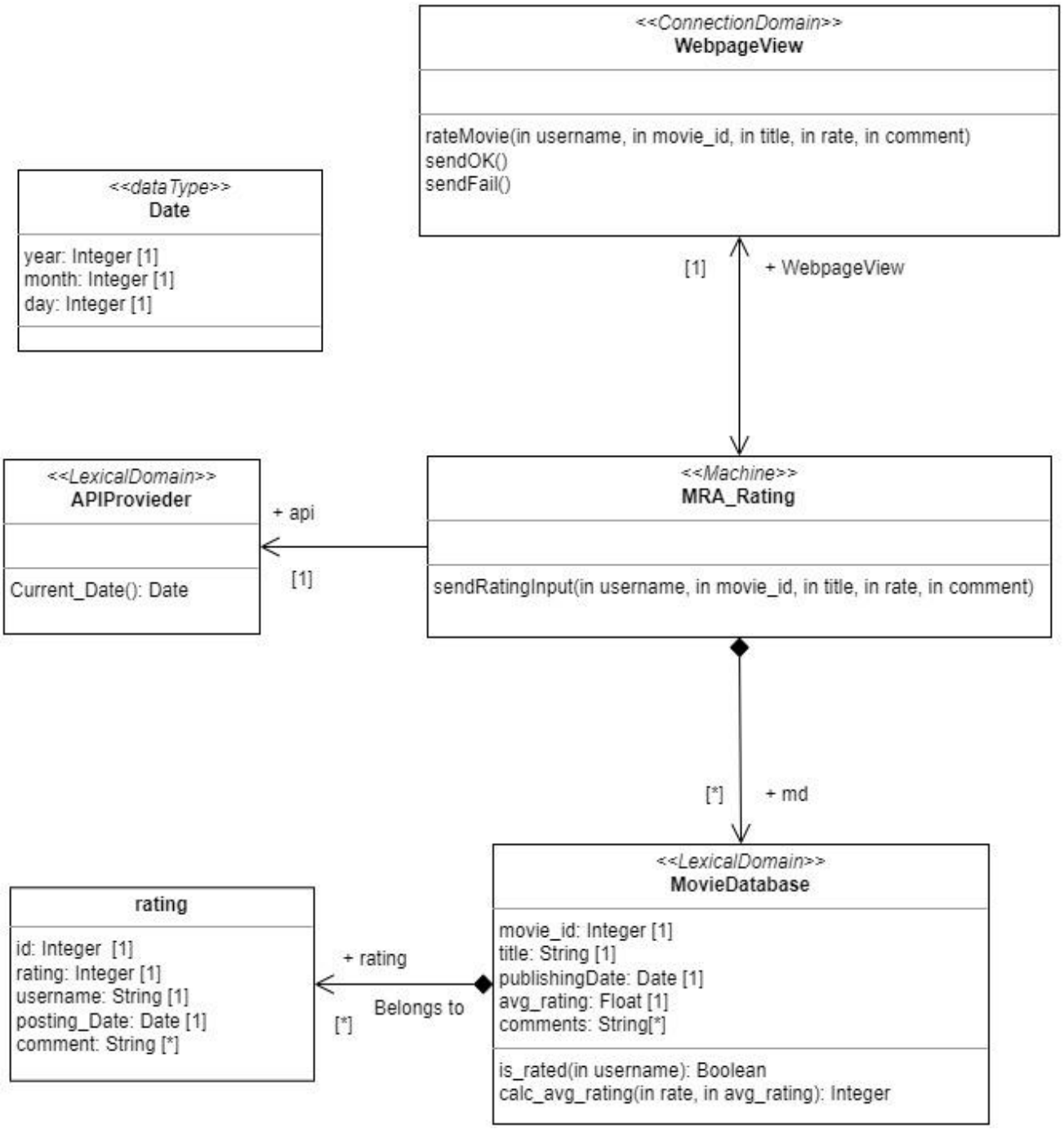
WebpageAddMovie^sendConfirmation()

2.3. OCL constraint:

Context: MovieDatabase

Inv: md.allInstances->isUnique(mainActors and title and director)

3. Movie Rating Movie Class Model



3.1. rateMovie Operation Specification:

Name: rateMovie

Description: Forwards ratings from user to the machine

Context WebpageView::rateMovie(in username, in movie_id, in title, in rate, in comment)
Pre: true
Post: MRA_Rating^sendRatingInput(username, movie_id, title, rate, comment)

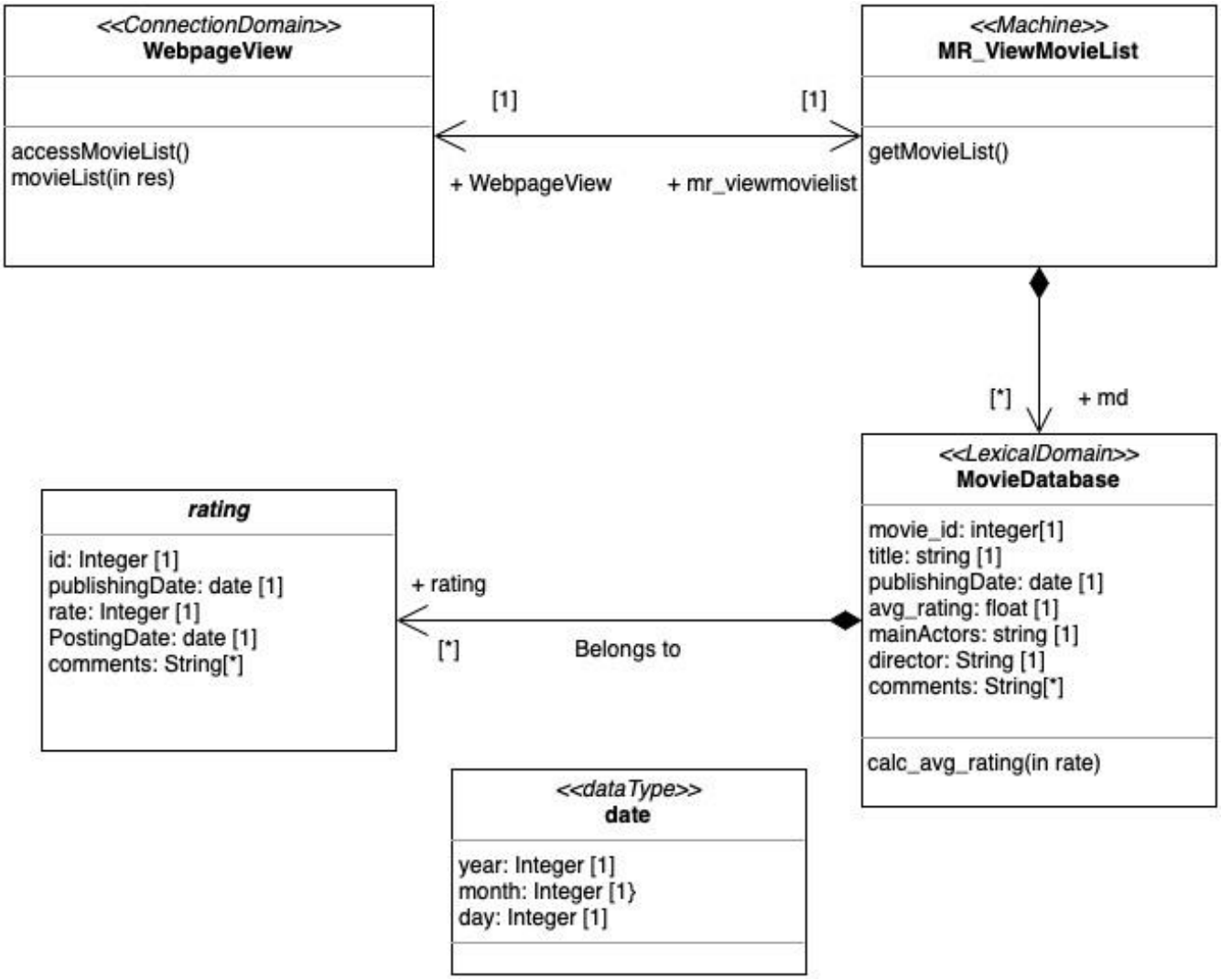
3.2. sendRatingInput Operation Specification:

Name: sendRatingInput

Description: It calculates new average rating for a movie and adds it to the database.

Context MRA_Rating::sendRatingInput(in username, in movie_id, in title, in rate, in comment)
Pre: md−>one (m: MovieDatabase | m.movie_id = movie_id)
Post:
if rate >= 1 and rate <= 10 **then**
    let m: MovieDatabase −> one(mdb: MovieDatabase | mdb.movie_id = movie_id) in
    if m@pre.is_rated(username) **then**
        m.rating → one (r: rating |
        r.rating = rate **and**
        r.username = username **and**
        r.postingDate = api.CurrentDate() **and**
        r.comment = comment) **and**
        m.avg_rating=m@pre.calc_avg_rating(rate)
        WebpageView^sendOK()
    else
        WebpageView^sendFail()
    endif
endif

4. View Movie List Class Model



4.1. accessMovieList Operation Specifcation:

Name: accessMovieList

Description: Forwards the browse request of the movie list from the User to the machine.

Context WebpageView::accessMovieList()

Pre: true

Post: MR_ViewMovieList^getMovieList()

4.2. getMovieList Operation Specifcation:

Name: getMovieList

Description: requests the movielist from the database

context MR_ViewMovieList: :getMovieList(username: String)

pre : true

post : WebpageView^movieList(md.allInstances()-> sortedBy(m: MovieDatabase|10-m.avg_rating))

Validation of Operation Specification A5:

User Registration:

- Operations specification must be consistent with specifications:
    - The specification of operations of registering user is consistent with the abstract specification.
- The postcondition covers all cases exhibited in abstract specification:
    - Both cases covered in the abstract specification (success and failure of registration) are covered in the post condition of giveUserinfo operation specification
- Parameters must be used in the pre- and/or postcondition:
    - All parameters of operations specified are used in the postcondition of these operations
- All parameters of operations must be known by the caller and all parameters of sent messages must be known by the machine:
    - User can input all parameters to RegisterWebpage via their web browser which is then able to forward them to the operation giveUserinfo.
- All classes, associations, and attributes newly introduced in the class model must be motivated by some operation specification:
    - No new classes, associations, or attributes were introduced in the class model during the specification.

Register Movie Validation:

- Operation specifications must be consistent with abstract specifications:
    - The operation specification of registeringMovie is consistent with the abstract specification
- The postcondition covers all cases exhibited in the abstract specification:
    - The normal case behavior described in the abstract specification is covered in the postcondition.
- All parameters of operations must be known by the caller and all parameters of sent messages must be known by the machine:
    - User can input all parameters to Add_Movie, which forwards these to this operation.
- Parameters must be used in the pre- and/or postcondition:
    - The parameters are used in the postcondition.
- All classes, associations, and attributes newly introduced in the class model must be motivated by some operation specification:
    - No new classes, associations, or attributes were introduced in the class model during the specification.

Movie Rating:

- Operations specification must be consistent with specifications:
    - The specification of operations of rating movie is consistent with the abstract specification.
- The postcondition covers all cases exhibited in abstract specification:
    - The normal case behavior described in the abstract specification is covered in the postcondition.
- Parameters must be used in the pre- and/or postcondition:
    - All parameters of operations specified are used in the postcondition of these operations
- All parameters of operations must be known by the caller and all parameters of sent messages must be known by the machine:
    - User can input all parameters to WebpageView via their web browser which is then able to forward them to the operation sendRatingInput.
- All classes, associations, and attributes newly introduced in the class model must be motivated by some operation specification: calcAverage was introduced in order to calculate the new average rating internally, also to compare dates equals method has been introduced

View Movie List:

- Operations specification must be consistent with specifications:
    - The specification of operations of registering user is consistent with the abstract specification.
- The postcondition covers all cases exhibited in abstract specification:
    - The normal case behavior described in the abstract specification is covered in the postcondition.
- Parameters must be used in the pre- and/or postcondition:
    - All parameters of operations specified are used in the postcondition of these operations
- All parameters of operations must be known by the caller and all parameters of sent messages must be known by the machine:
    - User can input all parameters to webpageView via their web browser which is then able to forward them to the operation getMovieList.
- All classes, associations, and attributes newly introduced in the class model must be motivated by some operation specification:
    - No new classes, associations, or attributes were introduced in the class model during the specification.

A6 Software Life-cycle

$\mathrm{LC}_{\mathrm{user}} = \mathrm{register}; (\mathrm{Add}|\mathrm{Rating}|\mathrm{MovieList})^*$
$\mathrm{LC}_{\mathrm{machine}} = ||_{i=1}^{n}(\mathrm{n})\ \mathrm{LC}_{\mathrm{user}}$

Validation A6:

Each sequence diagram of Step A3: Abstract software specification is contained in at least one life-cycle expression

Scenario	Life-cycle expression
sdRegister	$\mathrm{LC}_{\mathrm{user}}$
sdAdd	$\mathrm{LC}_{\mathrm{user}}$
sdRating	$\mathrm{LC}_{\mathrm{user}}$
sdMovieList	$\mathrm{LC}_{\mathrm{user}}$

For the biddable domain user exactly one life-cycle exists, namely

The life-cycles are consistent with the state predicates in Step A3: Abstract software specification:

- sdRegister has no state predicates at the start, therefore it can be executed anytime, but it needs to be executed only once for each user.
- sdAdd, sdRating, and sdMovieList: They all have a state predicate to check if a user is logged-in first, which means a user should register first.

the life-cycles are consistent with the pre- and postconditions in Step A5: Operations and data specification:

- sdRegister contains two operations register and giveUserinfo. They have no preconditions. So they can be executed at the beginning of life-cycle.
- SdRegisterMovie contains two operations registerMovie and registeringMovie. Both have no preconditions, however, the state predicate logged-in user in the abstract specification A3 still holds, since in order for the user to interact with Add_Movie(Webpage as connection domain), they have to be logged in. Therefore, a user should register first and then be able to register movie.
- sdRating contains rateMovie and sendRatingInput. Although it also has no precondition, for the user to interact with WebpageView, they should be logged-in, hence, they should be registered.

- sdMovieList contains getMovieList and accessMovieList Both have no preconditions, but it also holds that for the user to interact with WebpageView, it is already assumed that this user is logged-in, so they need to be registered first.

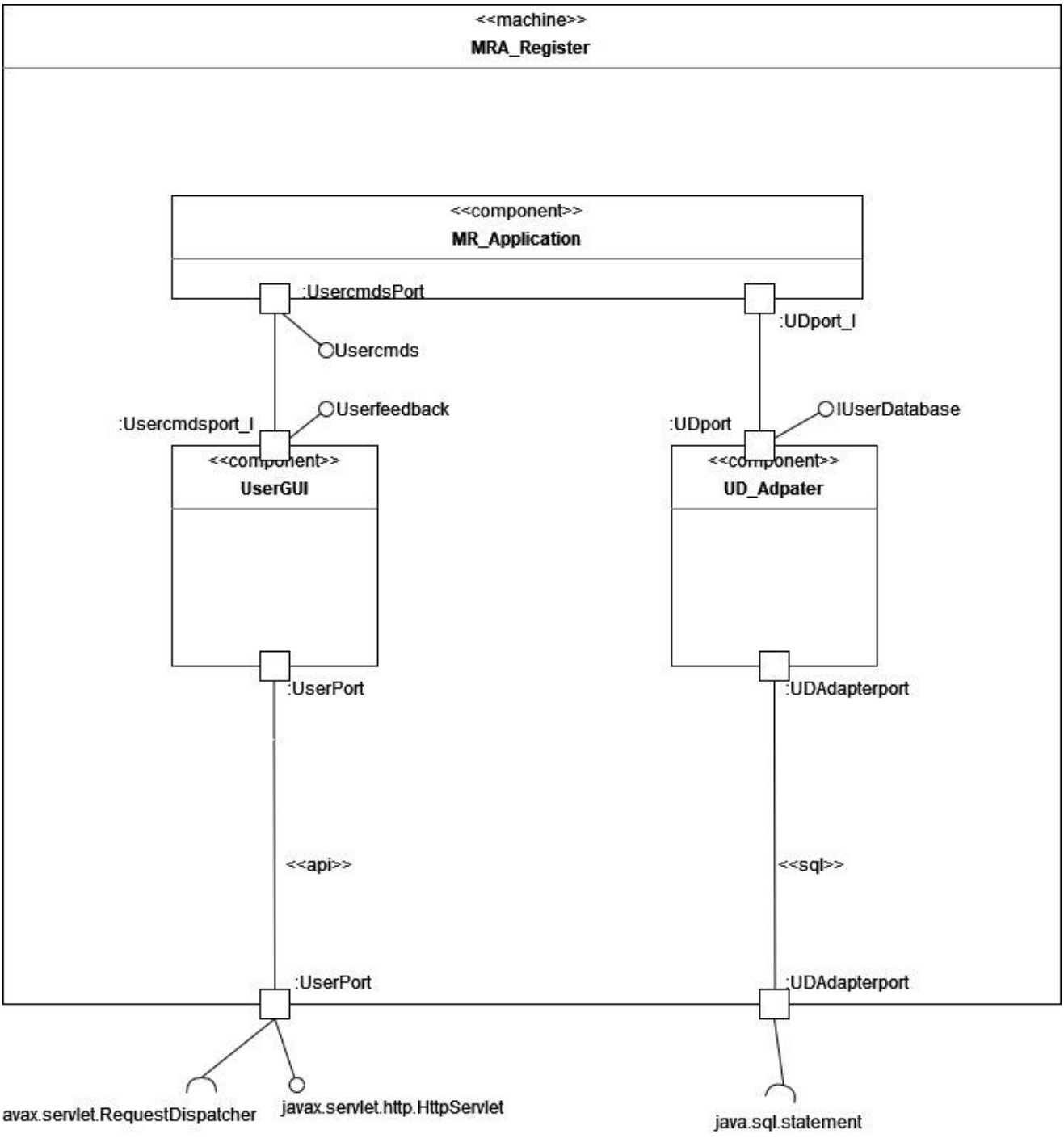
Exactly one life-cycle exists for the machine which have all other life-cycles.
Life-cycle LCmachine exists for the machine and contains LCuser.

Design phase

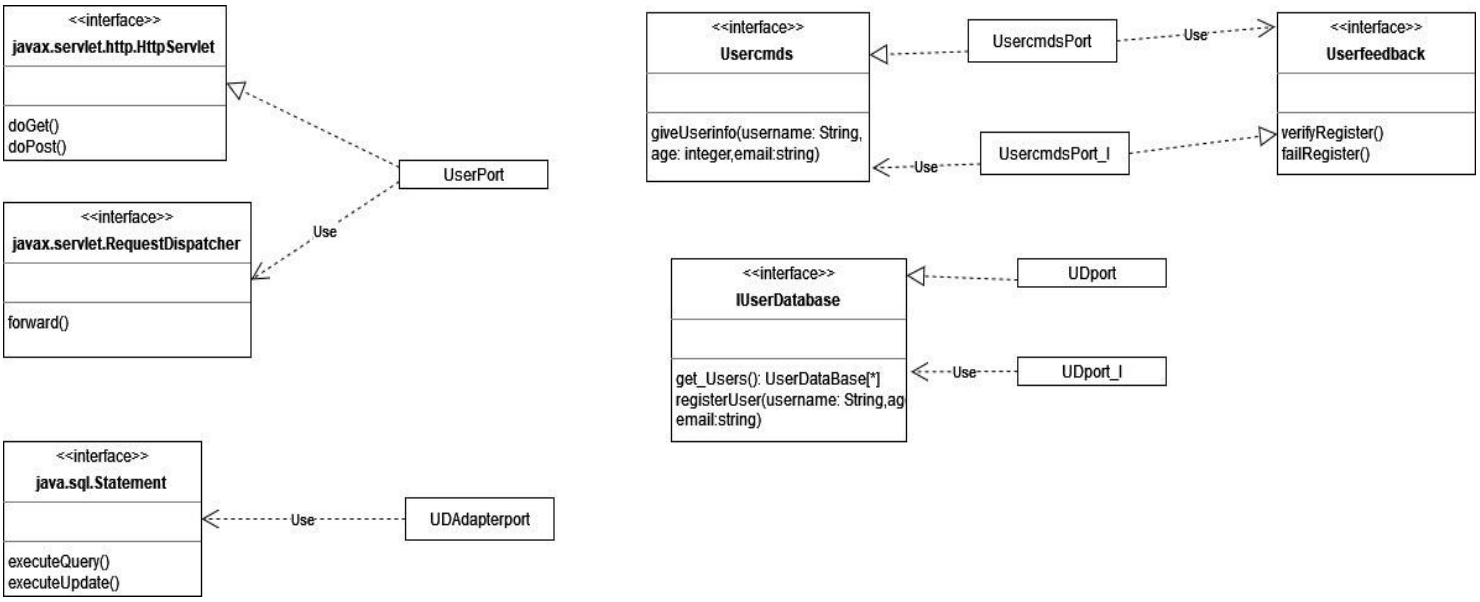
D1: Software Architecture

1. User Registration Architecture Diagram:

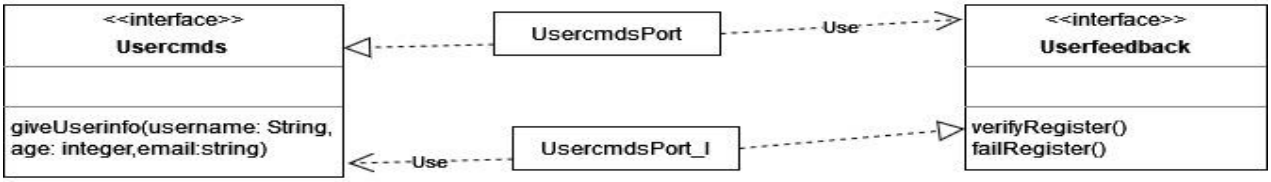
This Architecture Diagram uses the Architecture Pattern corresponding to the problem frame **update(2)** as discussed in the lecture.

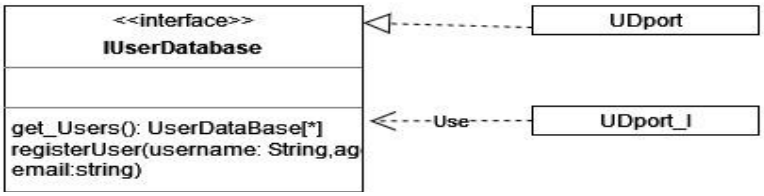


Port Types and Interface Relations in MRA_Register:



Internal Interfaces in MRA_Register:

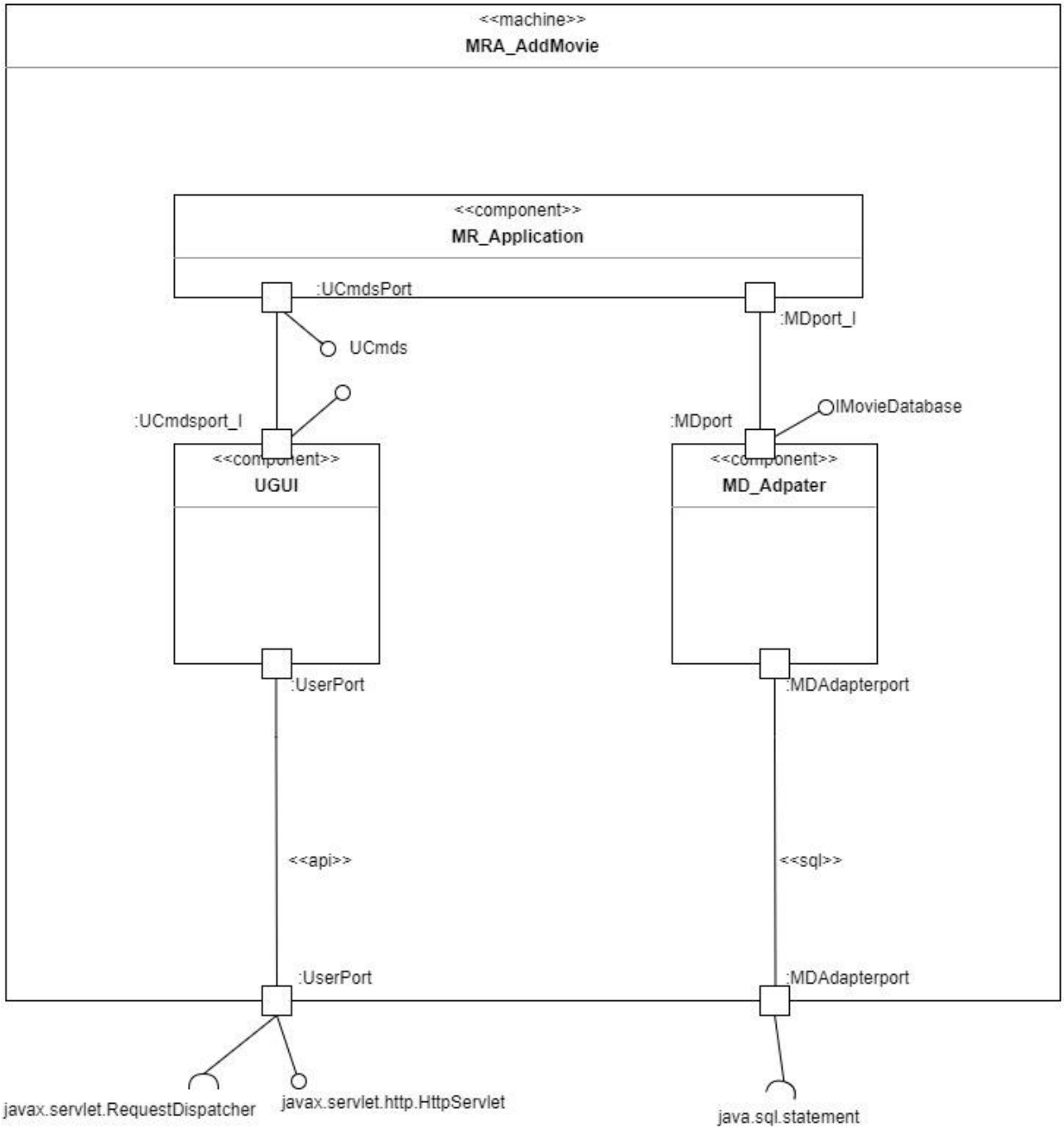




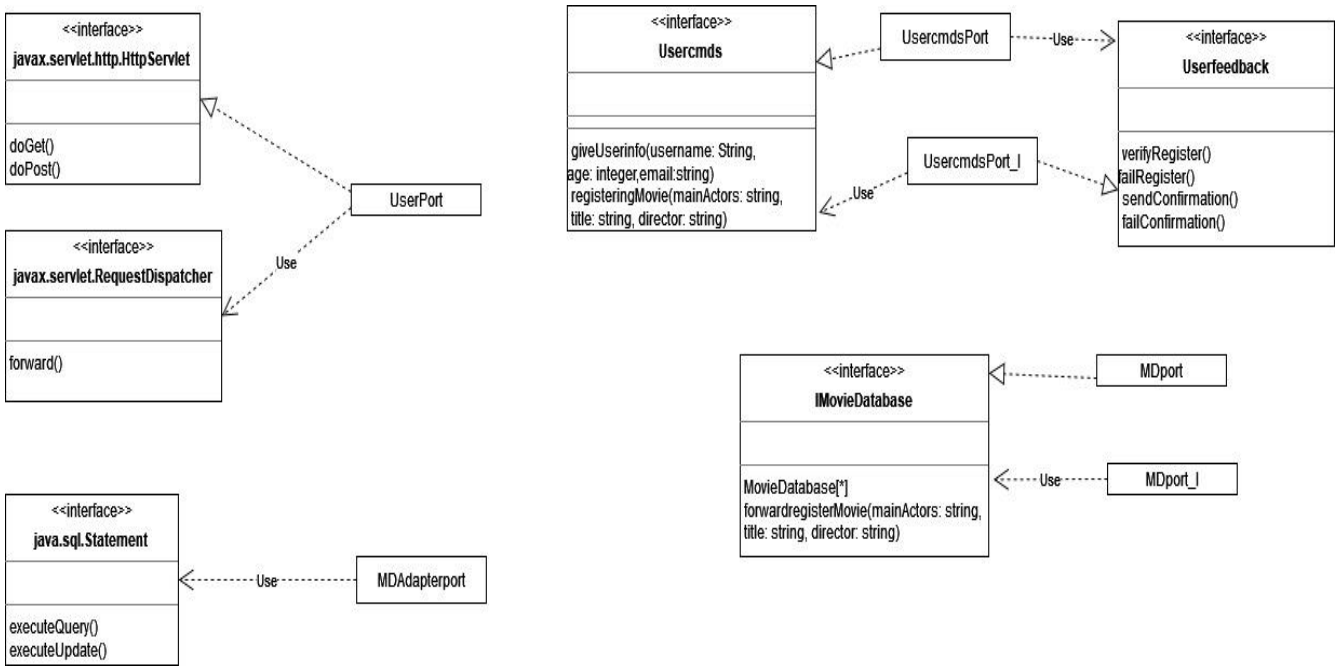
The interfaces withe the stereotype <<api>> and <<sql>> are machine interfaces and will be defined later

2. Register Movie Architecture Diagram:

This Architecture Diagram uses the Architecture Pattern corresponding to the problem frame **update (2)** as discussed in the lecture.

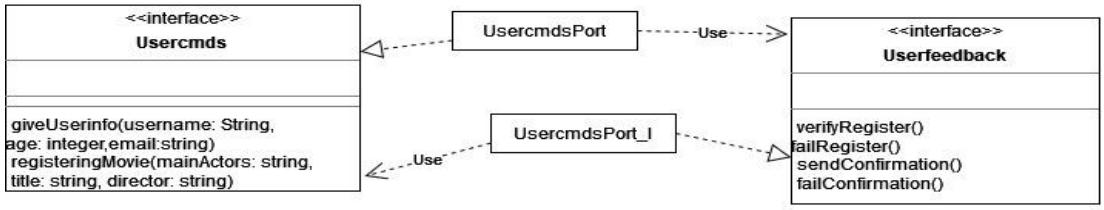


Port Types and Interface Relations in MRA_AddMovie:

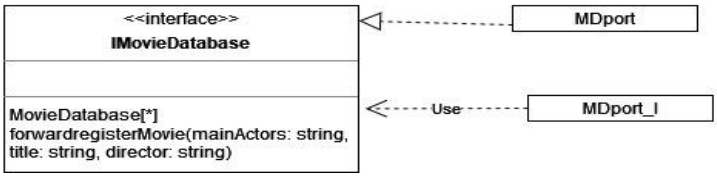


Internal Interfaces in MRA_AddMovie:

Usercmds and Userfeedback are extended as follows:



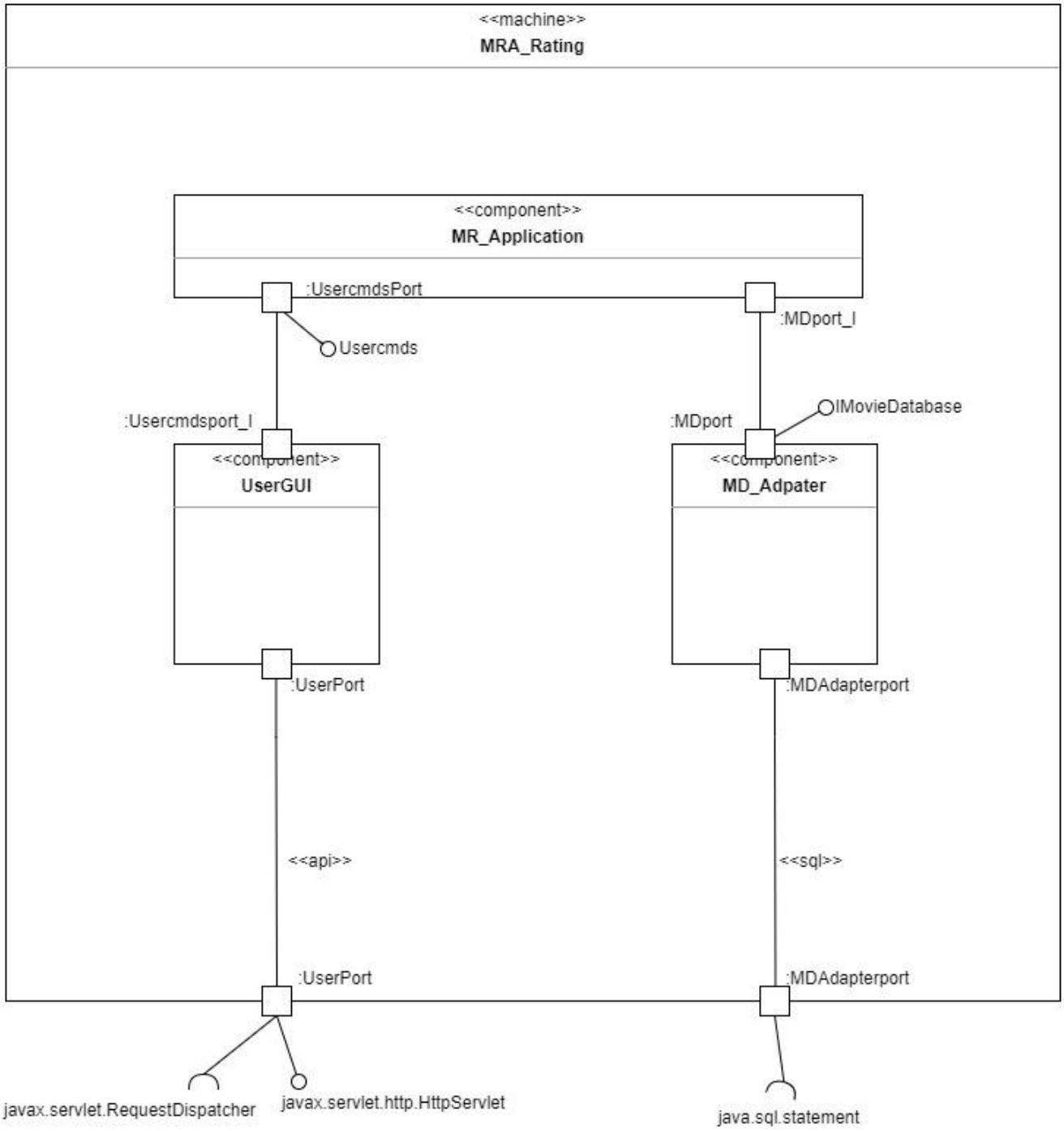
IMovieDatabase interface is added:



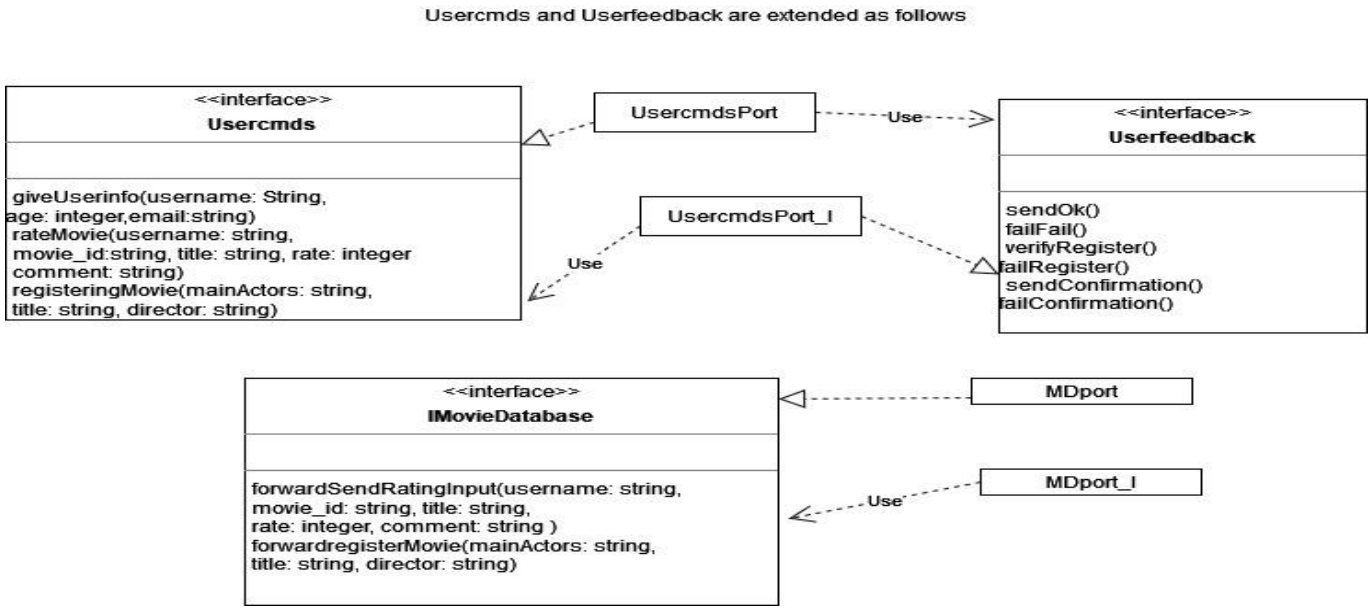
The interfaces withe the stereotype <<api>> and <<sql>> are machine interfaces and will be defined later

3. Rate Movie Architecture Diagram:

This Architecture Diagram uses the Architecture Pattern corresponding to the problem frame **update(2)** as discussed in the lecture.



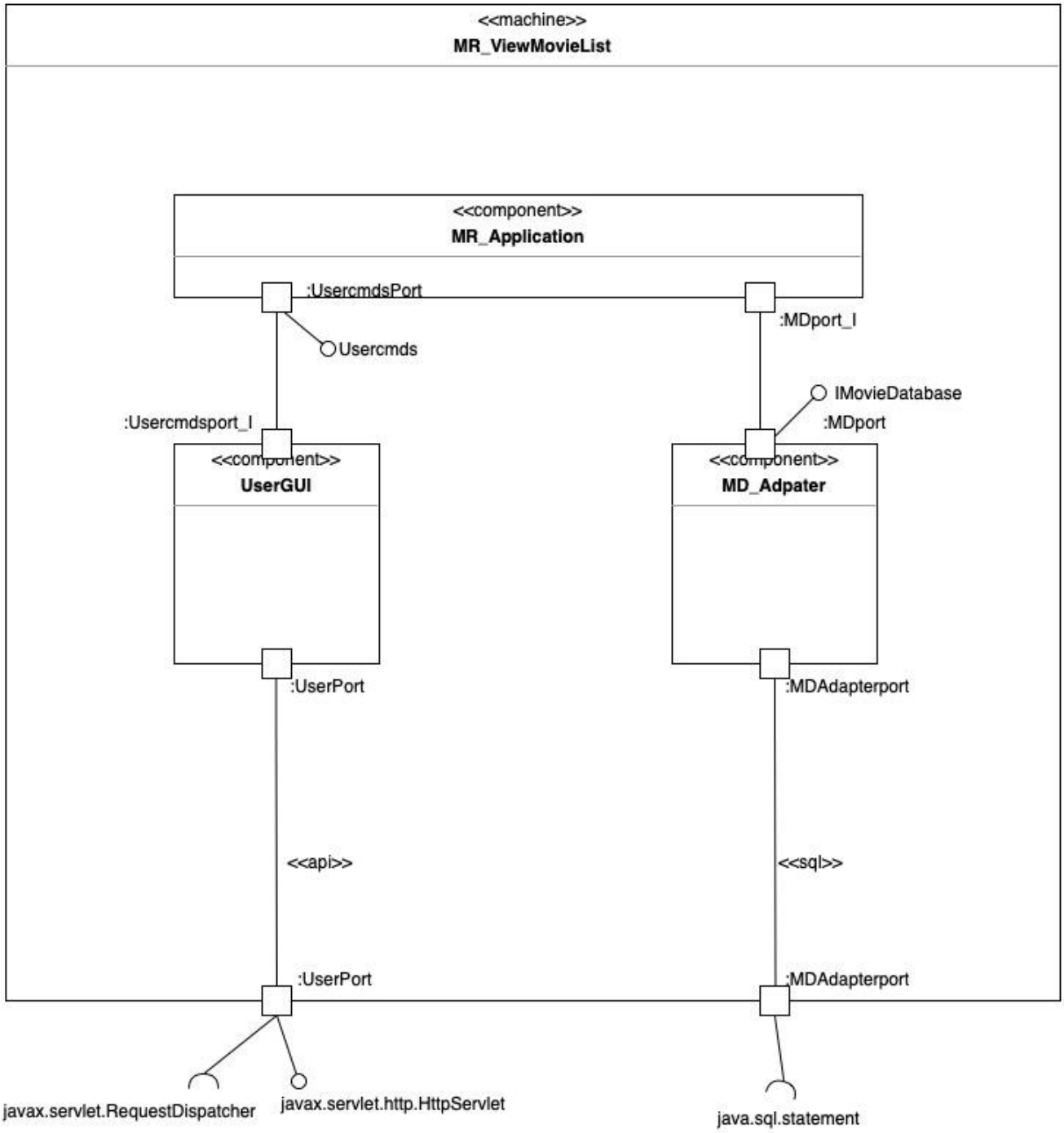
Internal Interfaces in MRA_Rating:



The interfaces withe the stereotype <<api>> and <<sql>> are machine interfaces and will be defined later

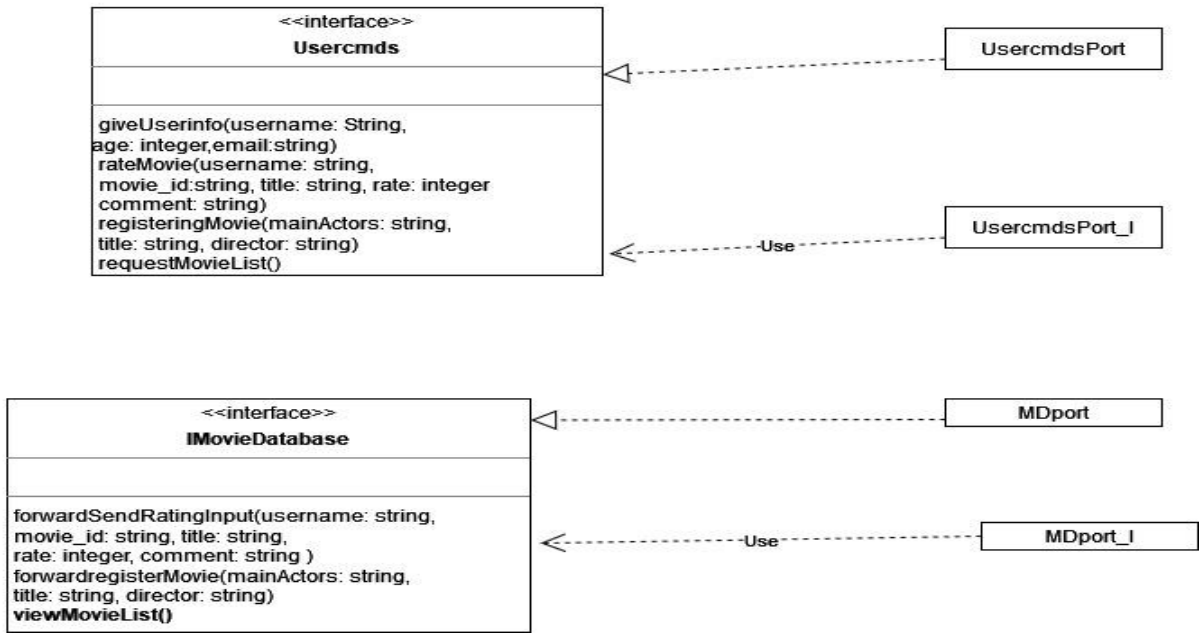
4. Movie List Architecture Diagram:

This Architecture Diagram uses the Architecture Pattern corresponding to the problem frame **Query(2)** as discussed in the lecture.



Internal Interfaces in MR_ViewMovieList:

Usercmds and IMovieDatabase are extended as follows:



The interfaces withe the stereotype <<api>> and <<sql>> are machine interfaces and will be defined later

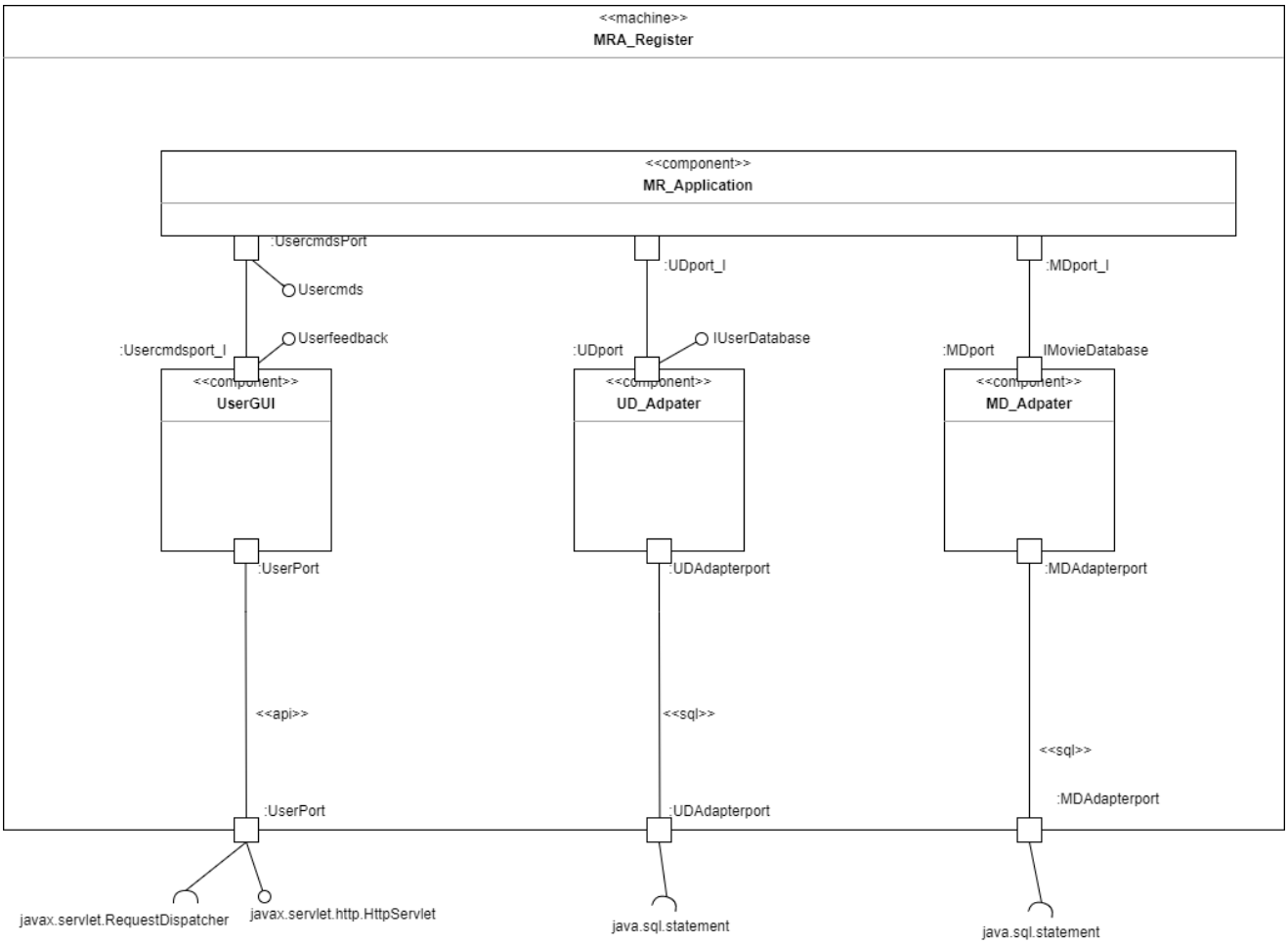
Refining Machine Interfaces using Technical Context Diagram:

Considered interface in subproblem architecture	**technical interface**
<<api>> javax.servlet.http.HttpServlet in MRA_Register	<<api>> AT!{doGet, doPost}
<<api>> javax.servlet.RequestDispatcher in MRA_Register	<<api>> MRA!{forward}
<<sql>> java.sql.Statement in MRA_Register	<<call return, sql>> MRA!{executeQuery, executeUpdate}
<<api>> javax.servlet.http.HttpServlet in MRA_AddMovie	<<api>> AT!{doGet, doPost}
<<api>> javax.servlet.RequestDispatcher in MRA_AddMovie	<<api>> MRA!{forward}
<<sql>> java.sql.Statement in MRA_AddMovie	<<call return, sql>> MRA!{executeQuery, executeUpdate}
<<api>> javax.servlet.http.HttpServlet in MRA_Rating	<<api>> AT!{doGet, doPost}
<<api>> javax.servlet.RequestDispatcher in MRA_Rating	<<api>> MRA!{forward}
<<sql>> java.sql.Statement in MRA_Register	<<call return, sql>> MRA!{executeQuery, executeUpdate}
<<api>> javax.servlet.http.HttpServlet in MR_ViewMovieList	<<api>> AT!{doGet, doPost}
<<api>> javax.servlet.RequestDispatcher in MR_ViewMovieList	<<api>> MRA!{forward}
<<sql>> java.sql.Statement in MR_ViewMovieList	<<call return, sql>> MRA!{executeQuery, executeUpdate}

Merging Subproblem Architectures:

The components can be merged as follows:

- The application components in all architectures for the subproblems should be merged because the subproblems Add, Rating and MovieList are related by alternative. Register is also merged because it is related sequentially to the other subpoblems.
- The MD_Adapter in Add, Rating and MovieList should be merged, because it is an adapter, establishing the connection to the DB and thus fulfills the same task in all subproblems.
- The UD_Adapter in Register is only used in this subproblem and is simply added to the merged architecture.
- The UserGUI for the User in all architectures for the subproblems uses the same technology and should be merged.



Validation D1:

All messages of Step A3: Abstract software specification are interfaces of the application layer

Sequence Diagram	Message	in/ out	Application Layer Interface	required/ provided
register	giveUserinfo	in	UserCmds::giveUserinfo	provided
	get_users	out	IuserDatabase::get_users	required
	UserData	in	return value of IuserDatabase::get_Users	required
	failRegister	out	return value of UserCmds::giveUserinfo	provided
	registerUser	out	IuserDatabase::registerUser	required
	verifyRegister	out	Return value UserCmds::giveUserinfo	provided
Add	registeringMovie	in	UserCmds::registeringMovie	provided
	get_MovieList	out	ImovieDatabase::get_MovieList	required
	MovieList	in	Return value of IMovieDatabase::get_MovieList	required
	forwardRegisterMovie	out	ImovieDatabase::forwardRegisterMovie	required
	sendConfirmation	out	Return value of UserCmds::registeringMovie	provided
	failConfirmation	out	Return value of UserCmds::registeringMovie	provided
Ratting	sendRatingInput	in	UserCmds::sendRatingInput	provided
	requestUserList	out	ImovieDatabase::requestUserList	required
	viewUserList	in	return value of ImovieDatabase::requestUserList	required
	addRating	out	ImovieDatabase::addRating	required
	sendOk	out	Return value of UserCmds::sendRatingInput	provided
	sendFail	out	Return value of UserCmds::sendRatingInput	provided
MovieList	get_MovieList	in	UserCmds::get_MovieList	provided
	requestMovieList	out	ImovieDatabase::requestMovieList	required
	viewMovieList	in	Return value of ImovieDatabase::requestMovieList	required

	MovieList	out	Return value of Usercmds::get_MovieList	provided

For global architecture: direction of all messages consistent to each other and input.

Provided by machine	required by adapter / provided by app
javax.servlet.http.HttpServlet	UserCmds

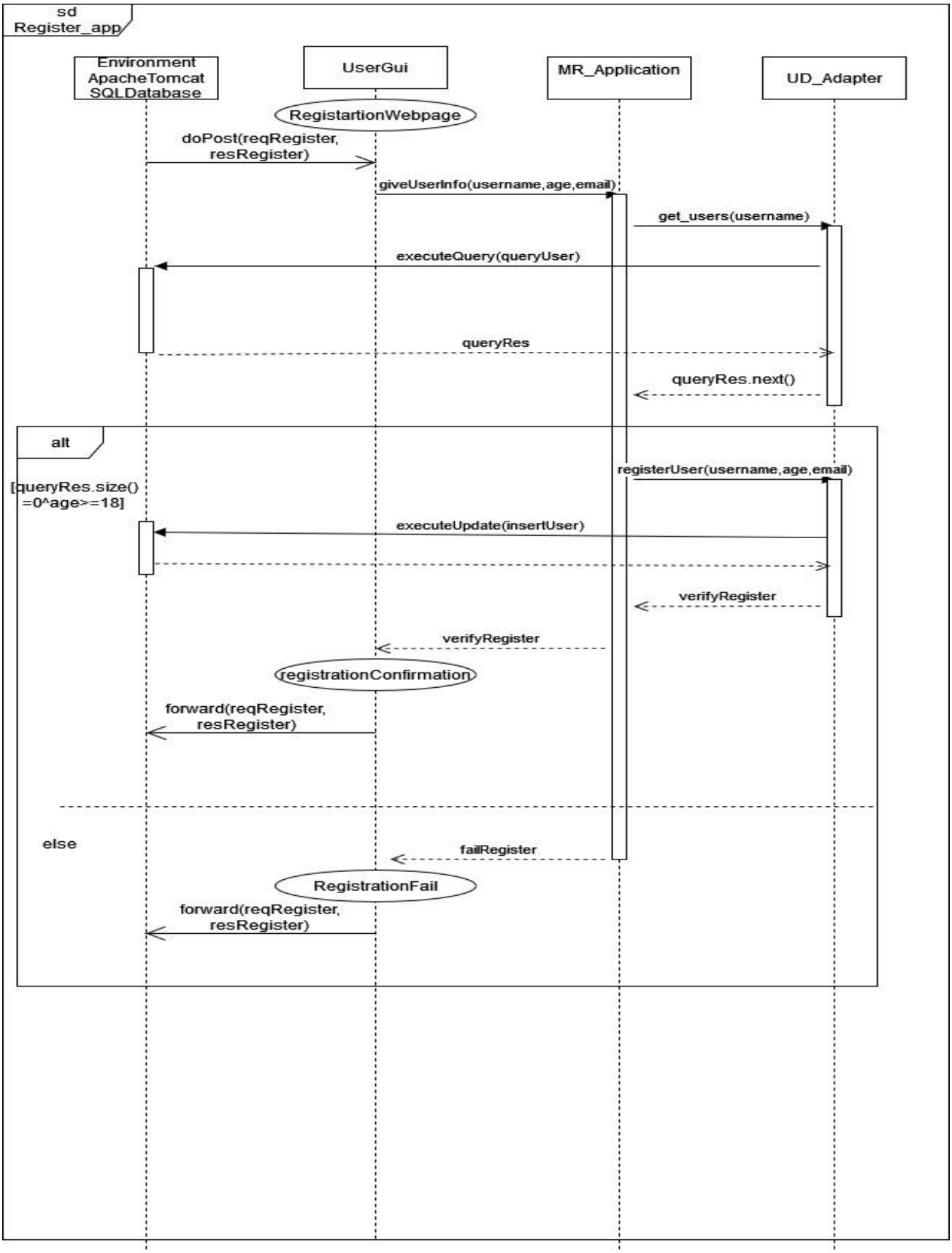
required by machine	Provided by adapter / required by app
javax.servlet.RequestDispatcher	return values in UserCmds
java.sql.Statement	IMovieDatabase
java.sql.Statement	IUserDatabase

The external ports of the subproblem architectures and the global architecture correspond to the interfaces and connection types in the technical context diagram.

external port type	interface in architecture	required/ provided	interface in technical context diagram
UserPort	javax.servlet.http.HttpServlet	provided	AT!{doGet, doPost}
	javax.servlet.RequestDispatcer	required	MRA!{forward}
UDAdapterPort	java.sql.Statement	required	MRA!{ executeQuery,executeUpdate }
MDAapterPort	java.sql.Statement	required	MRA!{ executeQuery,executeUpdate }

D2: Inter-Component Interaction

1. User Registration Intercomponent Interaction:



Register Intercomponent interaction remarks:

- reqRegister represents HTTPServletRequest object containing the required input.
- resRegister represents HTTPServletResponse object as the counterpart for the request.
- The state predicate registrationWebpage represents that the input form for registration is show.
- The state predicate RegistrationConfirmation represents that the registration confirmation is shown.
- The state predicate RegistartionFail represents that an error message is shown in case of registration failure.
- Forward(reqRegister,resRegister) sends the request and response back to the server to generate HTML webpage.
- Since we use a MySQL database, we do not need to specify the interfaces to lexical domains in more detail. We use standardized SQL statements to access the database.

SQL

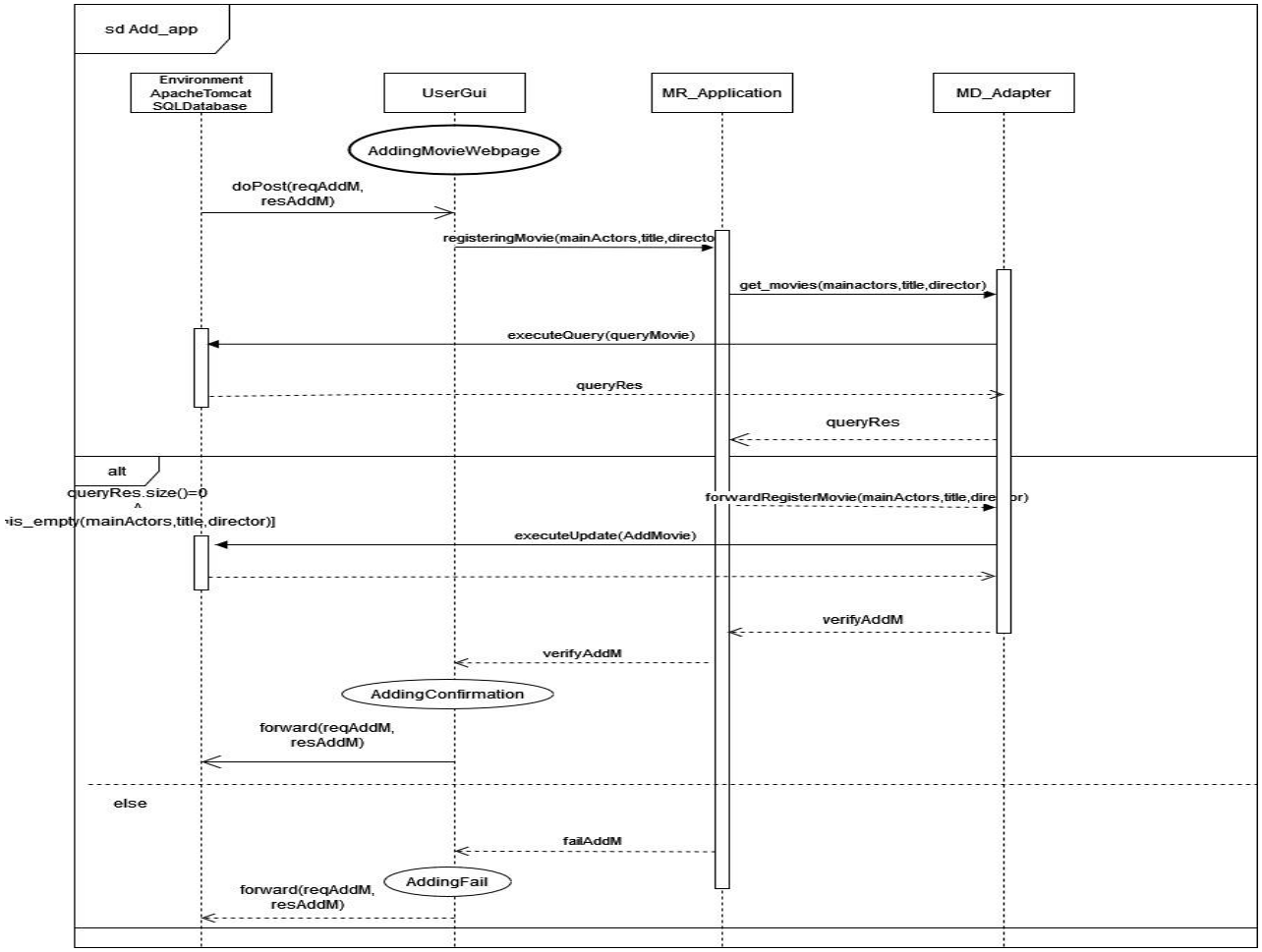
Query User:

```
SELECT * FROM UserDatabse WHERE ("username"=username)
```

Insert User:

```
INSERT INTO UserDatabase (username, age, email) VALUES
("username","age","email")
```

2. Adding Movie Intercomponent Interaction:



Add Intercomponent interaction remarks:

- reqAddM represents HTTPServletRequest object containing the required input.
- resAddM represents HTTPServletResponse object as the counterpart for the request.
- The state predicate AddingMovieWebpage represents that the input form for registration is show.
- The state predicate AddingConfirmation represents that the Adding confirmation is shown.
- The state predicate AddingFail represents that an error message is shown in case of Adding failure.
- Forward(reqAddM ,resAddM) sends the request and response back to the server to generate HTML webpage.

- Since we use a MySQL database, we do not need to specify the interfaces to lexical domains in more detail. We use standardized SQL statements to access the database.

SQL

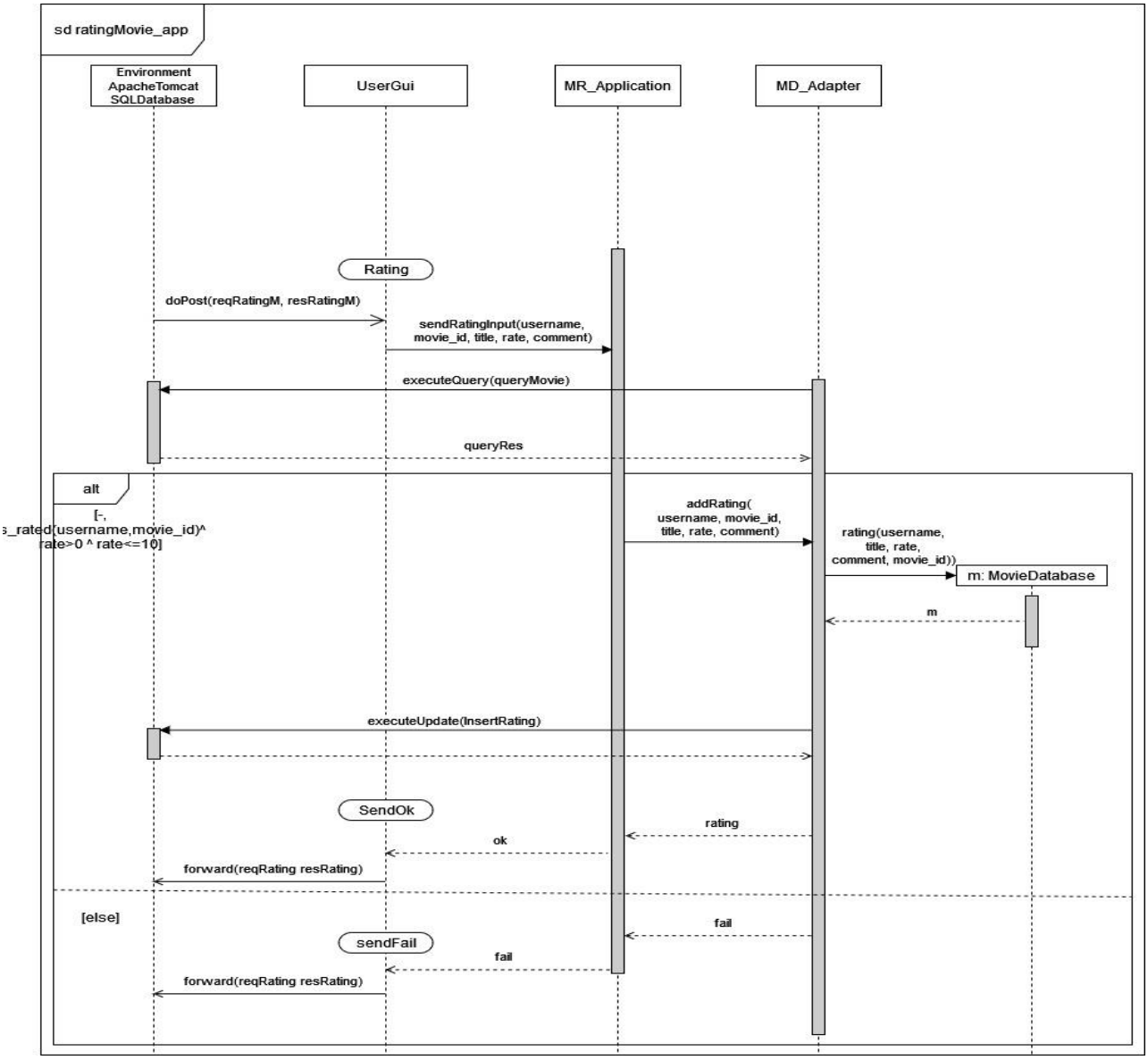
Query Movie:

```
SELECT * FROM MovieDatabase WHERE
(“mainActor”=mainActor AND ”title”=title AND ”director”=director)
```

Insert Movie:

```
INSERT INTO MovieDatabase (mainActors, title, director) VALUES
(“mainActor”, ”title”, ”director”)
```

3. Rating Movie Intercomponent Interaction:



Rate Intercomponent interaction remarks:

- **reqSelect** and **reqRatingM** represent HTTPServletRequest objectscontaining the required input.
- **resSelect** and **resRatingM** represent HTTPServletResponse objects as the counterpart for the request.
- The state predicate "**Rating**" represents that the input form for booking the selected offer is shown.
- The state predicate **sendOk** represents that the confirmation is shown.
- The state predicate **sendFail** represents that an error message is shown.
- **forward(reqRatingM, resRatingM)** sends the request and response back to the server to generate the HTML webpage.
- Since we use a MySQL database, we do not need to specify the interfaces to lexical domains in more detail. We use standardized SQL statements to access the database.

SQL

queryMovie:

```
SELECT * FROM MovieDatabase WHERE (Movie_id = "Movie_id")
```

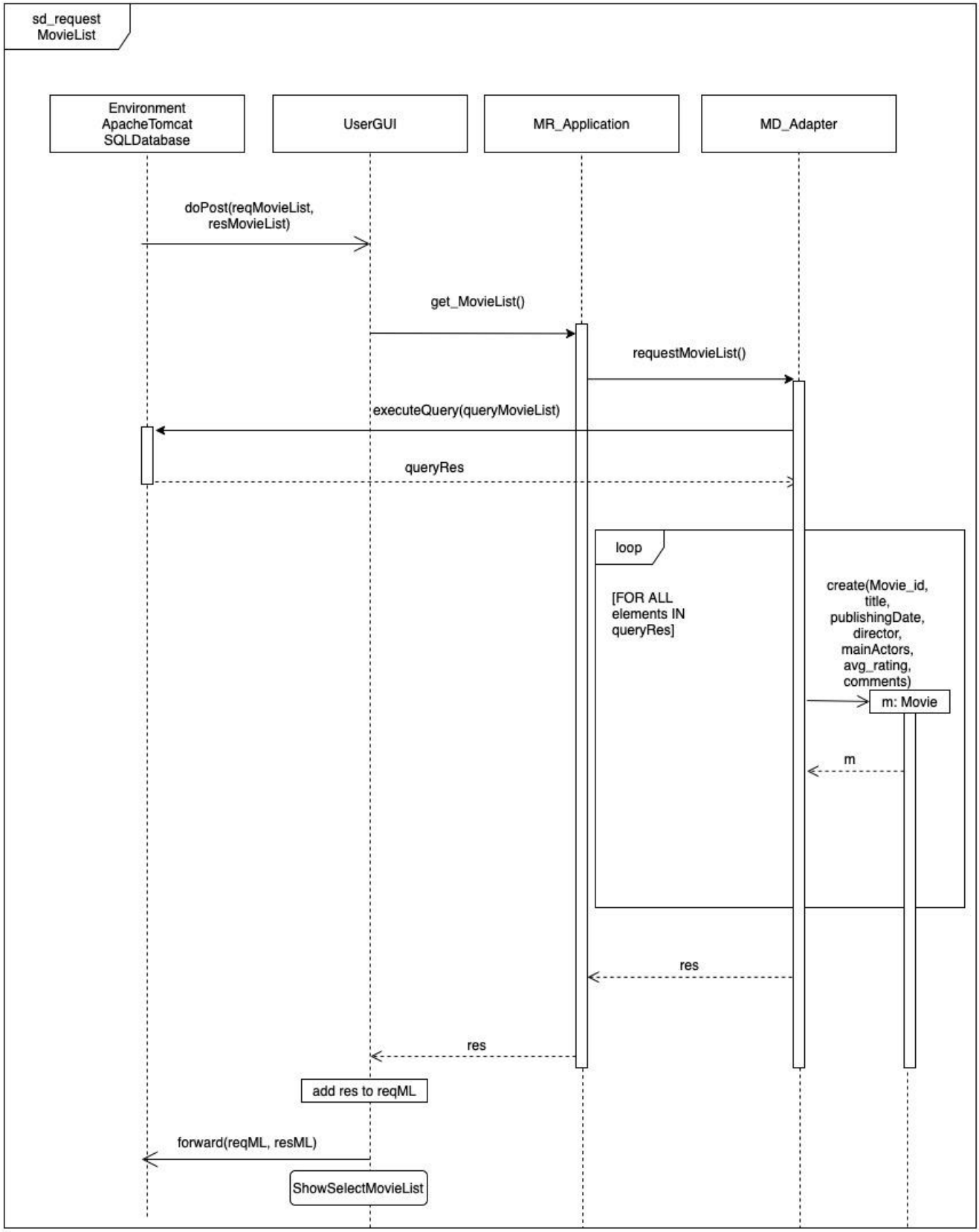
Add Rating:

```
INSERT INTO Ratings(Movie_id, title, username, rate, comment, Posting_date)
VALUES  ("Movie_id","title","username","5","comment","ddmmyyyy")
```

UpdateAvgRating:

```
UPDATE MovieDatabase
SET avg_rating=(SELECT avg(r.rating) FROM ratings r GROUP BY Movie_id)
WHERE Movie_id = "Movie_id"
```

4. MovieList Intercomponent Interaction:



View MovieList Intercomponent interaction remarks:

- **reqMovie** represents HTTPServletRequest objectscontaining the required input.
- **resMovieList** represents HTTPServletResponse objects as the counterpart for the request.
- The state predicate **ShowSelectMovieList** represents that the the list of available offers is shown.
- **res** is an ordered set Movies in a list that fit to the selection criteria. The set will be added to the request object to be processed by the server.
- m refers to an object of class MovieDatabase.
- **forward(reqML, resML)** sends the request and response back to the server to generate the HTML webpage.
- Since we use a MySQL database, we do not need to specify the interfaces to lexical domains in more detail. We use standardized SQL statements to access the database.

SQL

queryMovie:

```
SELECT * FROM MovieDatabase Where movie_id = “movie_id”
ORDER BY avg_rating DESC
```

Validation D2:

The sequence diagrams must be consistent with the behavior described in Step A3: Abstract software specification and in Step A6: Software lifecycle.
Consistency of sdRegister_app and sdregister.

Message in D2	Corresponding message in A3
doPost(reqRegister, resRegister)	Refines register
giveUserInfo(..)	giveUserinfo
get_users(..)	get_users
executeQuery(..)	Refines get_users
registerUser(..)	registerUser
executeUpdate(..)	Refines registerUser
Forward(reqRegister, res Register)	Refines forwardVerifyRegister

Consistency of sdAdd_app and sdAdd

Message in D2	Corresponding message in A3
doPost(reqAddM, resAddM)	Refines registerMovie
registeringMovie(MovieData)	registeringMovie
forwardregisterMovie(...)	get_MovieList
executeQuery(queryMovie)	Refines get_MovieList
executeUpdate(AddMovie)	Refines get_MovieList
Forward(reqAddM, resAddM)	Refines forwardConfirmation
Forward(reqAddM, resAddM)	Refines forwardFail

Consistency of sdratingMovie_app app and sdRating.

Message in D2	Corresponding message in A3
doGet(reqSelect, resSelect)	Refines rateMovie
forward(reqSelect, resSelect)	Refines rateMovie
doPost(reqRatingM, resRatingM)	Refines rateMovie
rateMovie(...)	sendRatingInput
sendRatingInput(...)	requestUserList
excuteQuery(queryMovie)	Refines sendRatingInput
excuteUpdate(InsertRating)	Refines sendRatingInput
forward(reqRating, resRating)	Refines showOk
forward(reqRating, resRating)	Refines showFail

Consistency of sdaccessMovieList app and sdMovieList.

Message in D2	Corresponding message in A3
doPost(reqMovieList, resMovie)	Refines accessMovieList
get_MovieList(userName)	get_MovieList
requestMovieList(userName)	requestMovieList
executeQuery(...)	Refines requestMovieList
forward(reqML, resML)	forwardMovieList

Consistency with life-cycle:
Register; (Add|Rating|MovieList)*.

- sdRegister has no state predicates at the start, therefore it can be executed anytime, but it needs to be executed only once for each user. sdregisterMovie, sdrateMovie, and sdaccessMovieList: They all have a state predicate to check if a user is loggedin first, which means a user should register first.

- The sequence diagrams must realize the operations described in Step A5: Operations and data specification.
  - giveUserinfo is realized in sdregister app:
    - Precondition does not have to be established, because it is true.
    - Postcondition MRA application delegate the message to UD adapter. Using the sql commands queryRegister, UD adapter check if the parameters which have inputted in RegisterWebpage are corrected then RegisterWebpage send verifyRegister otherwise failRegister.
  - registeringMovie is realized in sdregisterMovie app:
    - Precondition does not have to be established, because it is true.
    - Postcondition MD adapter check if the movie data are already in MovieDatabase , GUI in WebpageAddMovie show failConfirmation, else show sendConfirmation.
  - sendRatingInput is realized in sdrateMovie app:
    - Precondition user can rate any movie just one time.
    - Postcondition just the registered users can rate movies which they watched in their watch list, the rating must be between 1 and 10.
  - getMovieList is realized in sdaccessMovieList app:
    - Precondition does not have to be established, because it is true.
    - Postcondition MR Application received the message from MD adapter, then show movieList

All messages in the application interface classes of Step D1: Software architecture must be used in some sequence diagram.

Interface	Message	Used in sequence diagram
UserCmds	giveUserinfo registeringMovie sendRatingInput getMovieList	sdregister app sdregisterMovie app sdrateMovie app sdaccessMovieList app
IuserDatabase	get_users registerUser	sdregister app sdregister app
ImovieDatabase	get_MovieList forwardRegisterMovie requestUserList addRating requestMovieList	sdregisterMovie app sdregisterMovie app sdrateMovie app sdrateMovie app sdaccessMovieList app

The directions of messages must be consistent with the required and provided interfaces of Step D1: Software architecture.

Interface Message	Provided by Recipient	Required by Sender
UserCmds: giveUserinfo registeringMovie sendRatingInput getMovieList	MRA Application MRA Application MRA Application MRA Application MRA Application	GuestGUI GuestGUI GuestGUI GuestGUI GuestGUI
IuserDatabase: get_users registerUser	UD_Adapter UD_Adapter UD_Adapter	MRA Application MRA Application MRA Application
ImovieDatabase: get_MovieList forwardRegisterMovie requestUserList addRating requestMovieList	MD_Adapter MD_Adapter MD_Adapter MD_Adapter MD_Adapter MD_Adapter	GuestGUI GuestGUI GuestGUI GuestGUI GuestGUI GuestGUI

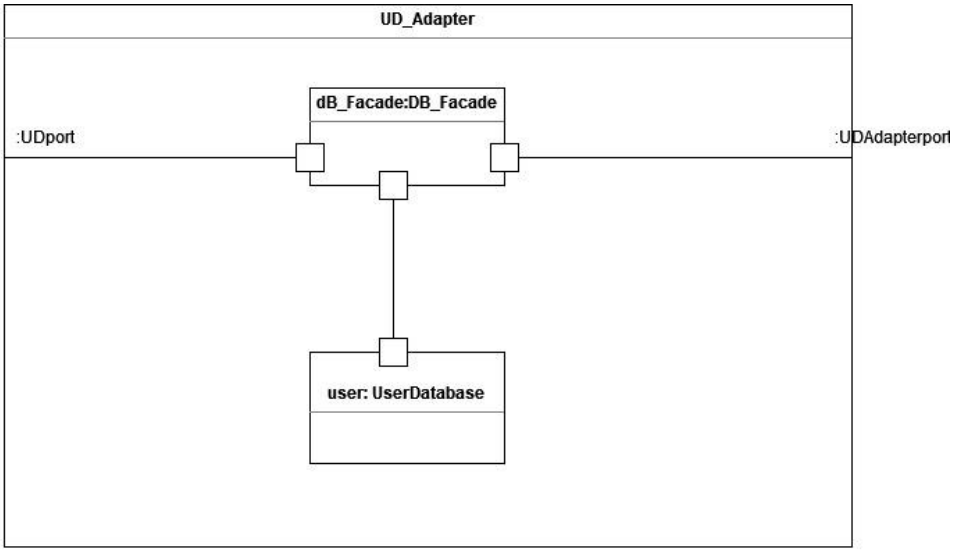
Messages must connect components as connected in the software architecture of Step D1: Software architecture.

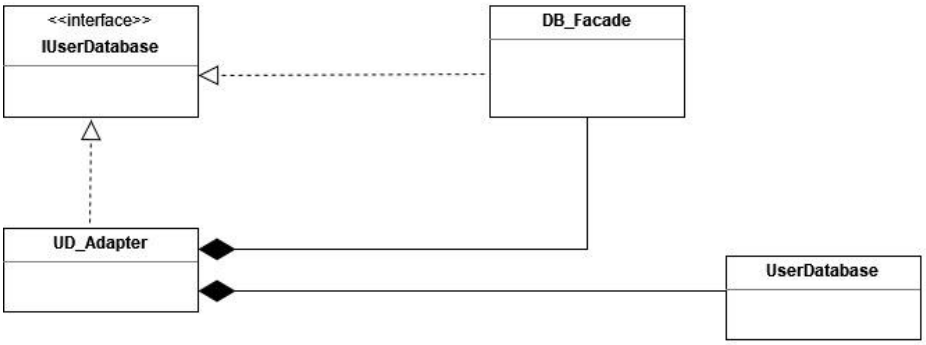
Component	Connected components in architecture	Connected components in sequence diagram
MR_Application	UserGUI, UD_Adapter, MD_Adapter	UserGUI, UD_Adapter, MD_Adapter

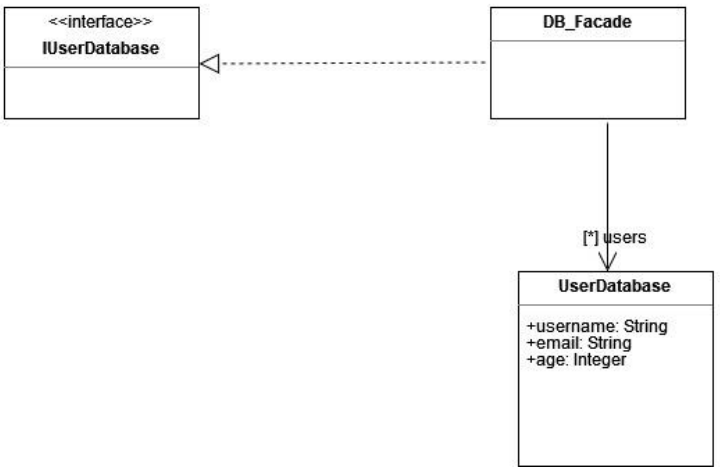
D3: Intra-Component Interaction

1. User registeration:

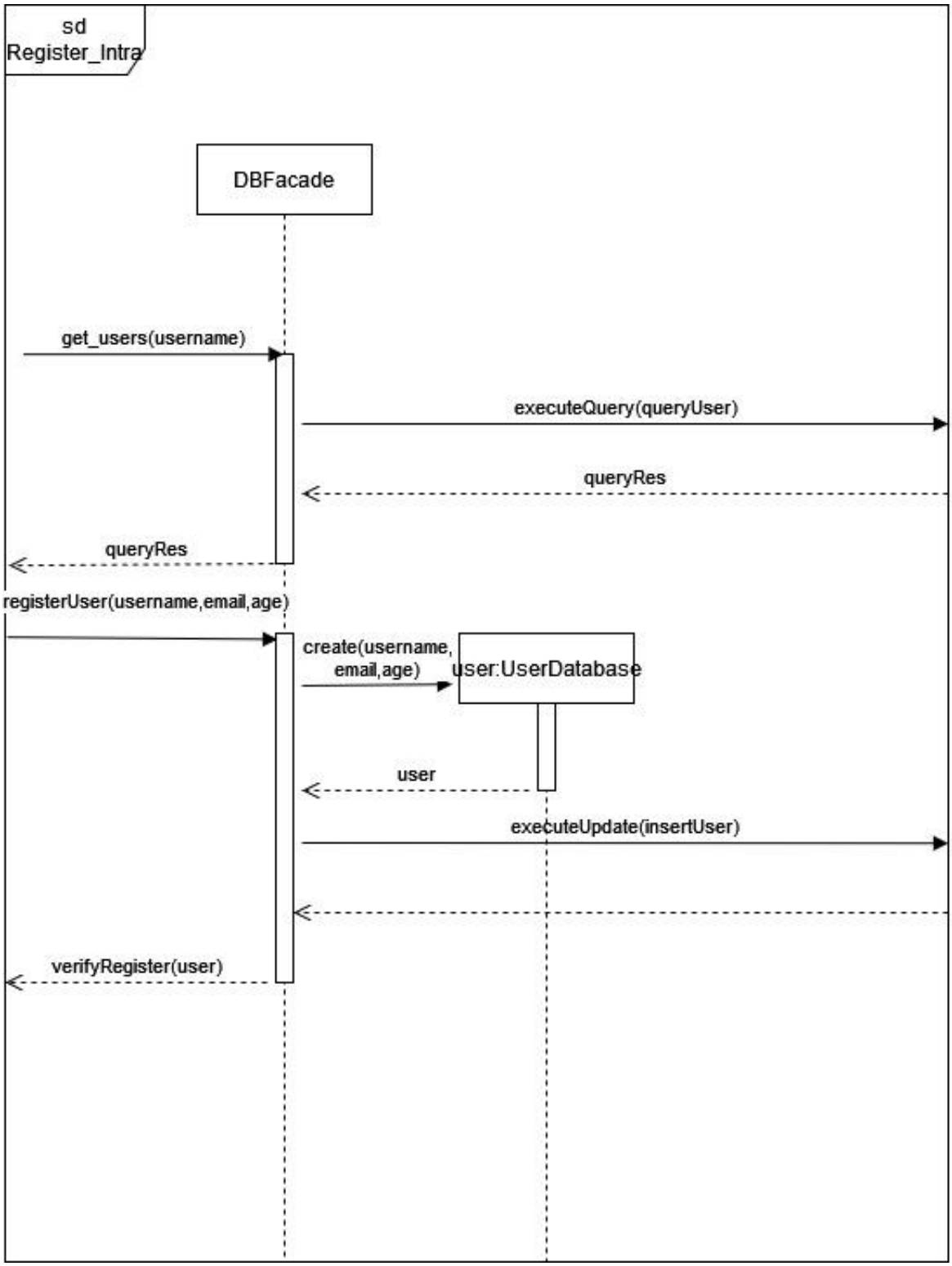
1.1. Preliminary architectural description





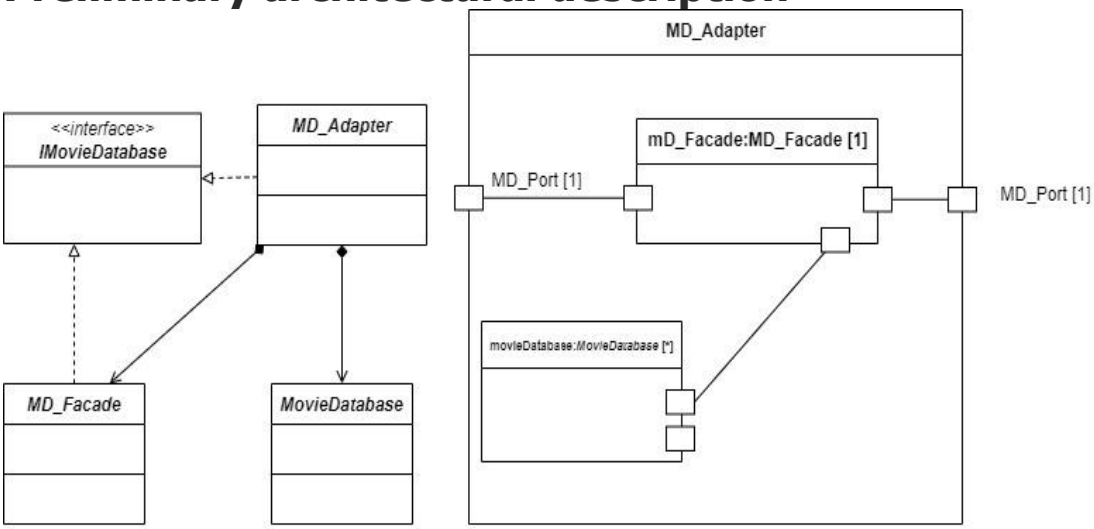


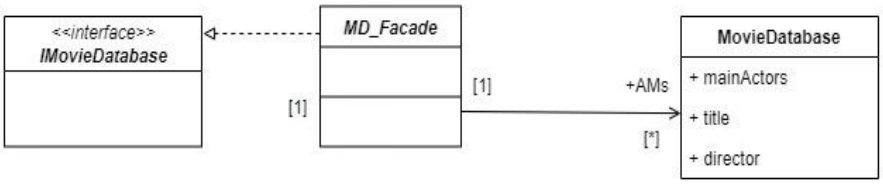
1.2. User Registration Intra-Component Interaction



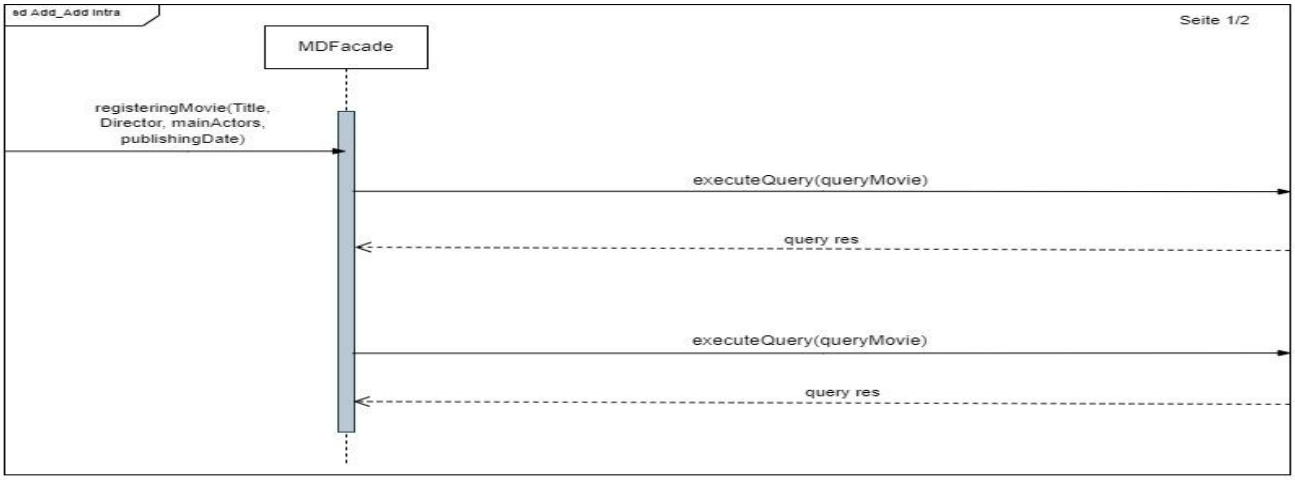
2. Add Movie

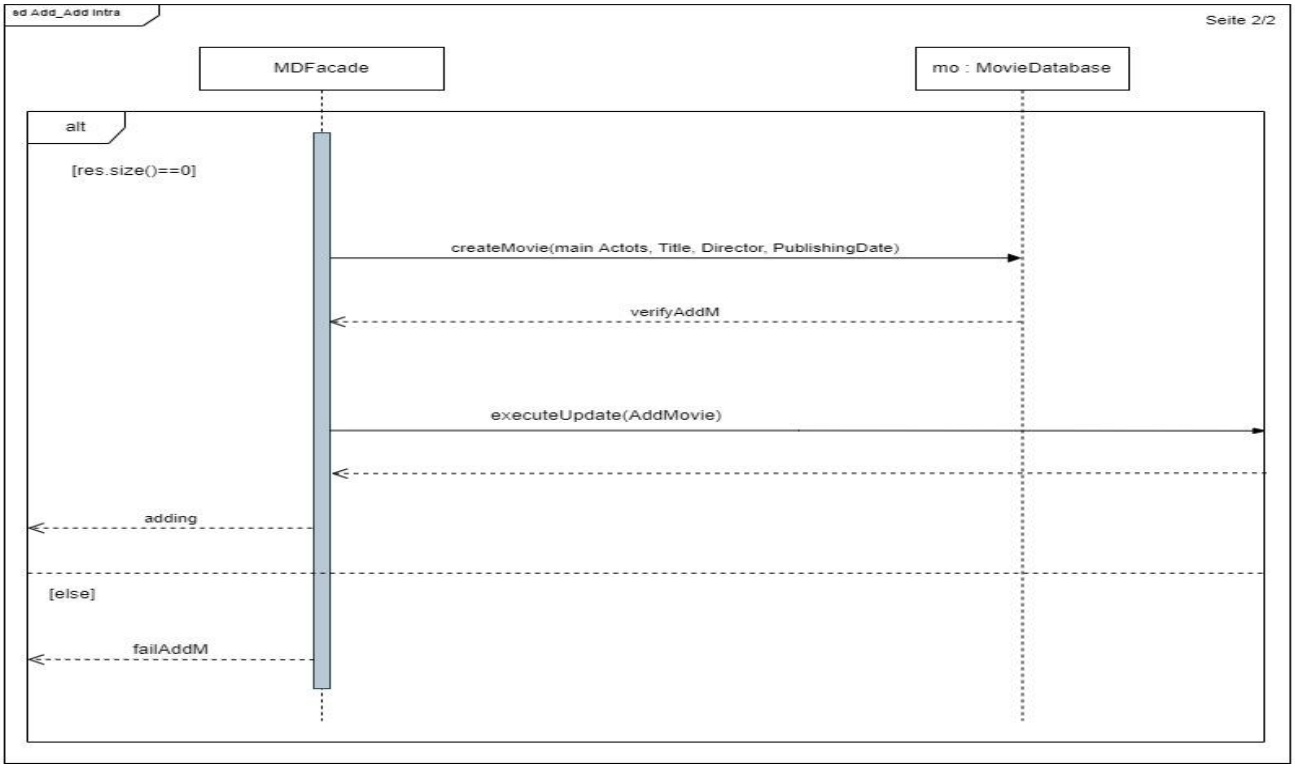
2.1. Preliminary architectural description





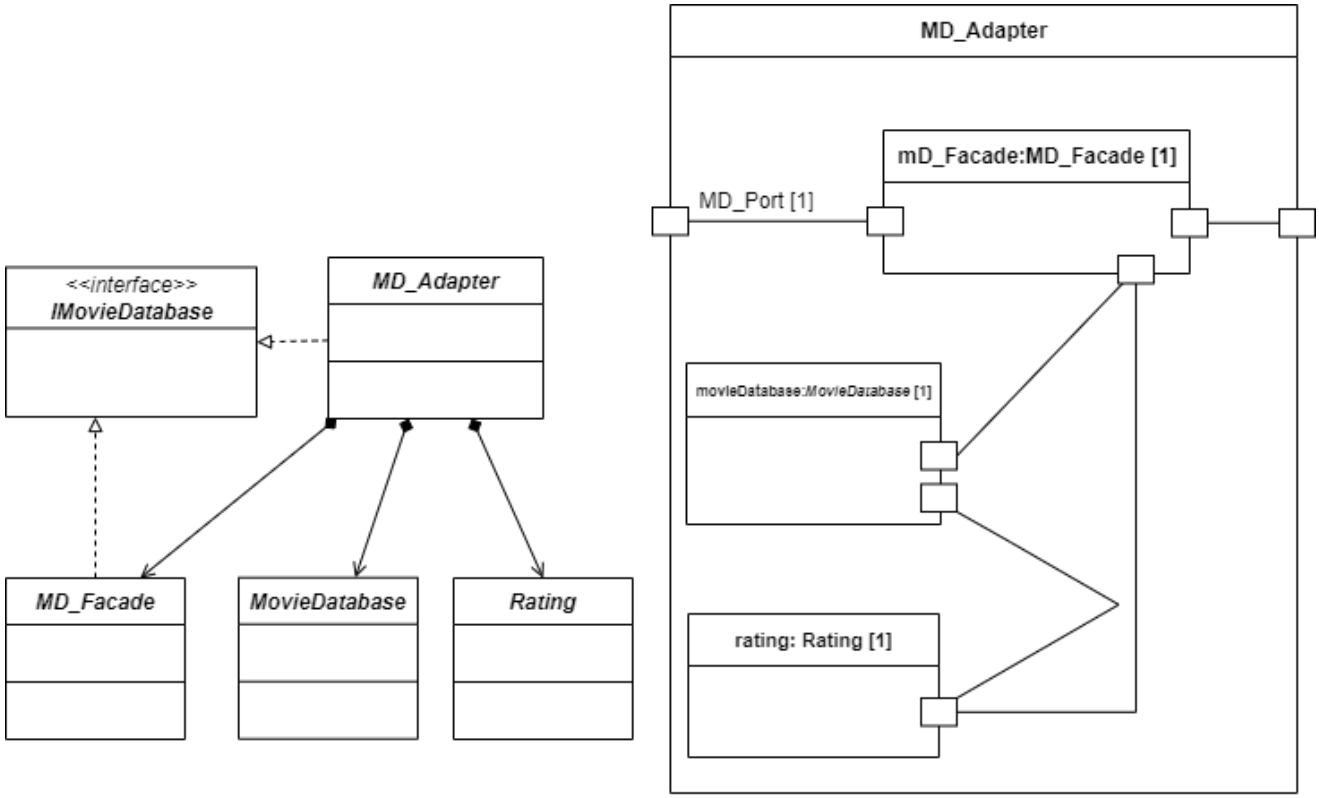
2.2. Add Movie Intra-Component Interaction

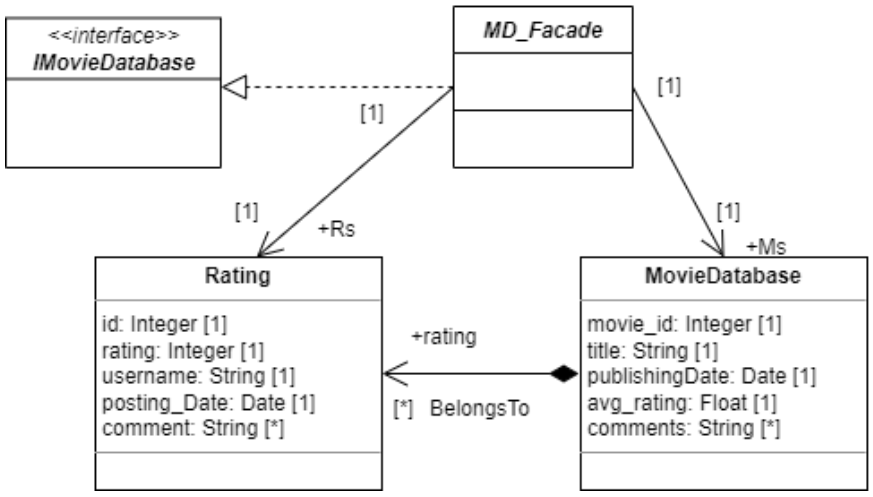

sd Add_Add intra
Seite 1/2
MDFacade
registeringMovie(Title, Director, mainActors, publishingDate)
executeQuery(queryMovie)
query res
executeQuery(queryMovie)
query res


sd Add_Add intra
Seite 2/2
MDFacade
mo : MovieDatabase
alt
[res.size()==0]
createMovie(main Actots, Title, Director, PublishingDate)
verifyAddM
executeUpdate(AddMovie)
adding
[else]
failAddM

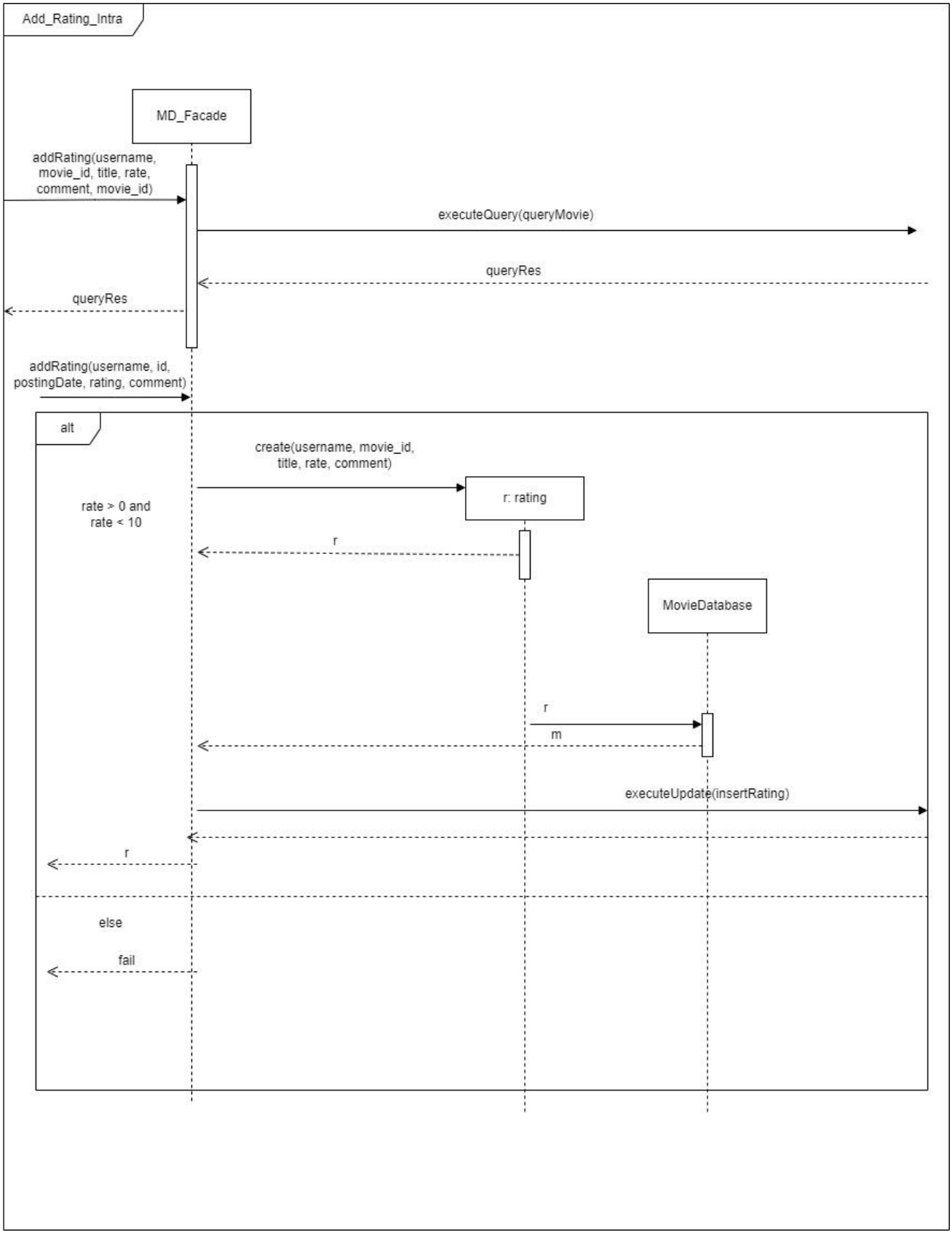
3. Rate Movie

3.1. Preliminary architectural description



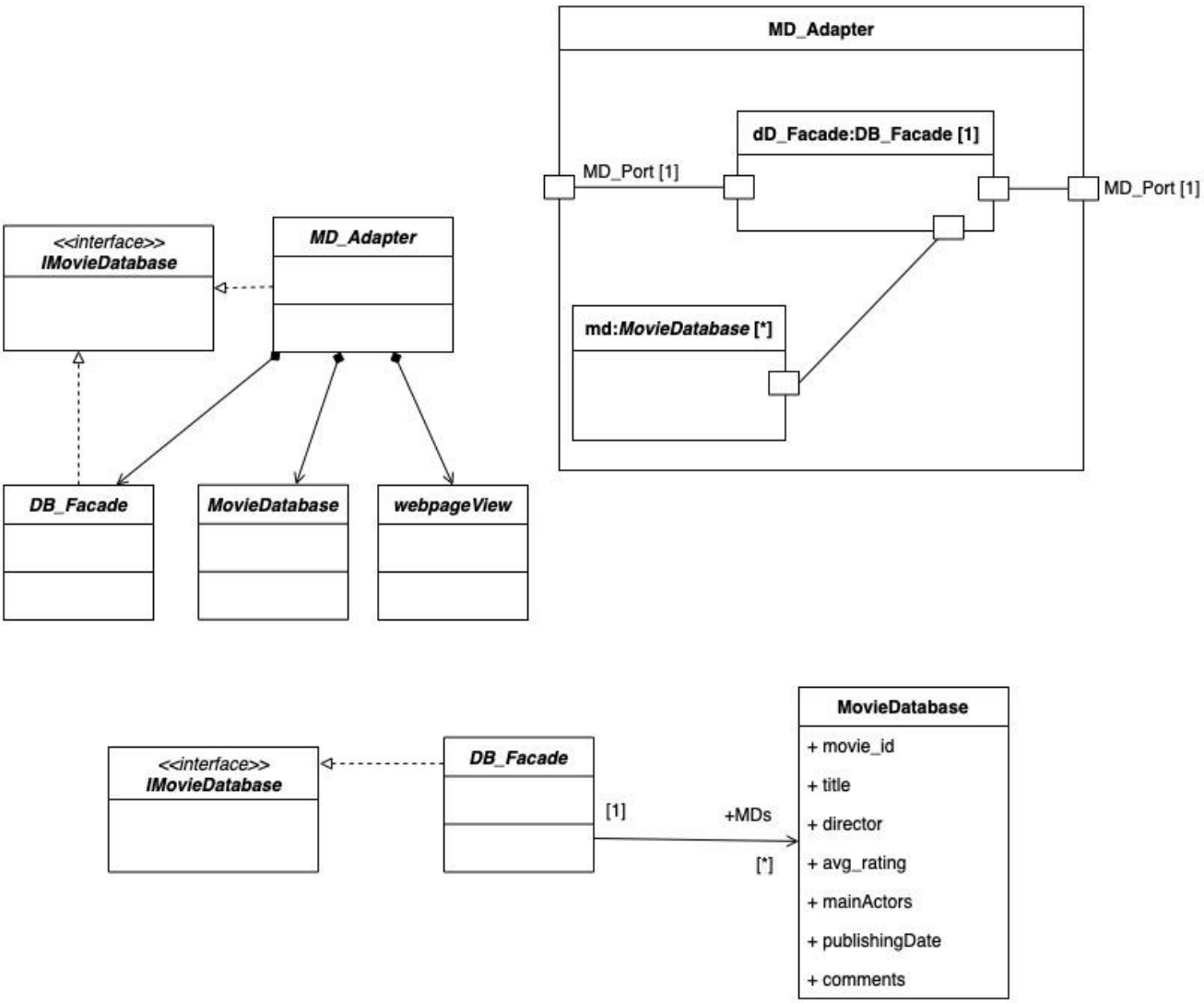


3.2. Rate Movie Intra-Component Interaction

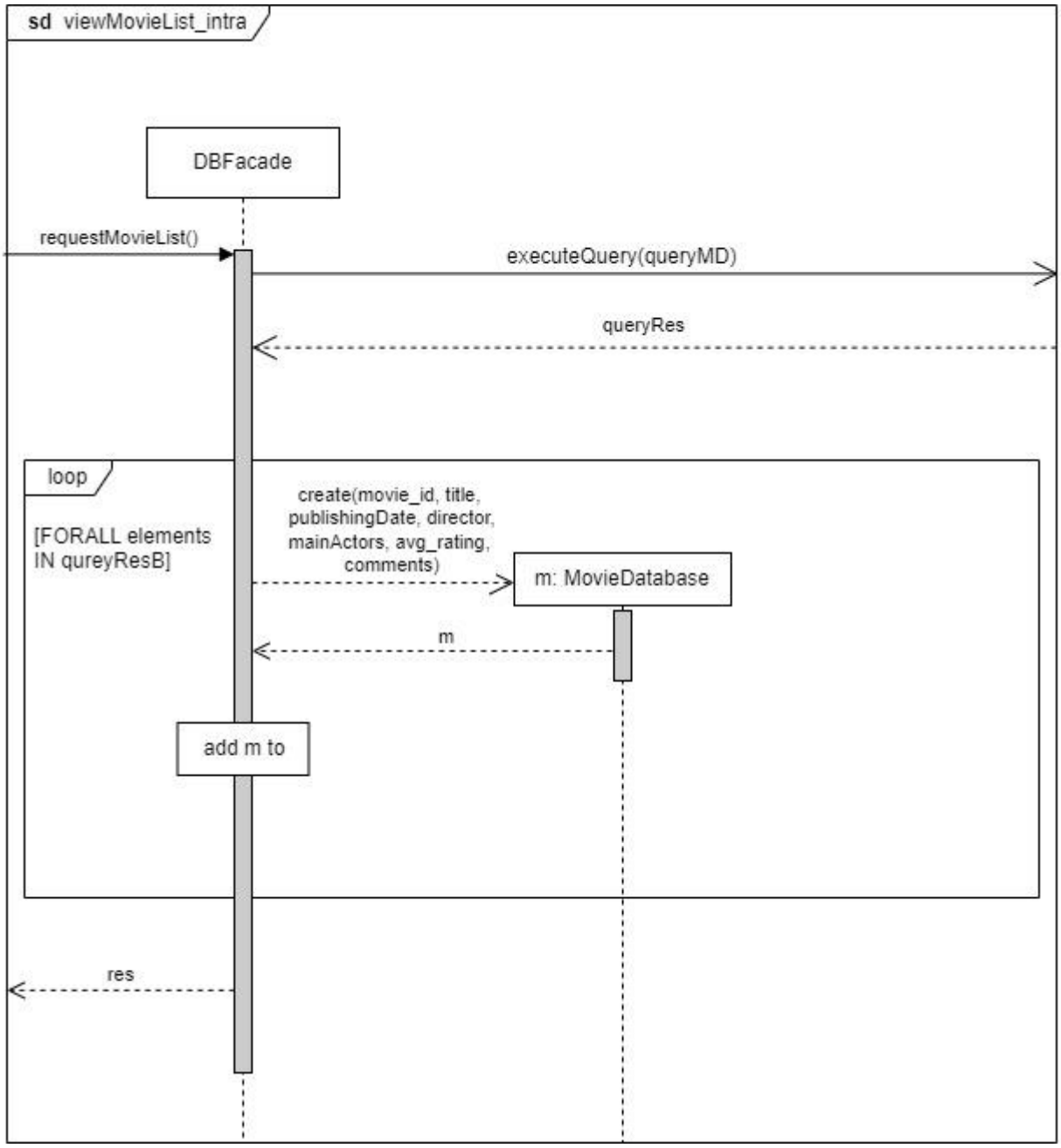


4. View Movie

4.1. Preliminary architectural description



4.2. View Movie List Intra-Component Interaction



Validation D3:

- Sequence diagrams of one component must be consistent with the corresponding interface behavior in step D2:
Inter-Component Interaction.

Messages of the sequence diagram sdRegister_intra in step D3: Intra-Component Interaction.	Messages of the sequence diagram sdRegister_app in step D2: Inter-Component Interaction
get_users(…)	get_users(…)
executeQuery(queryUser)	executeQuery(…)
registerUser(…)	registerUser(…)
create(…)	Refinement
executeUpdate(…)	executeUpdate(…)

Messages of the sequence diagram Sd Add_Add Intra in step D3: Intra-Component Interaction.	Messages of the sequence diagram sdAdd_app in step D2: Inter-Component Interaction
registeringMovie(…)	registeringMovie(…)
executeQuery(…)	executeQuery(…)
executeQuery(…)	Refinement
forwardRegisterMovie(…)	forwardRegisterMovie(…)
executeUpdate(…)	executeUpdate(…)

Messages of the sequence diagram Add_Rating_Intra in step D3: Intra-Component Interaction.	Messages of the sequence diagram sd rating_Movie_app in step D2: Inter-Component Interaction
executeQuery(…)	executeQuery(…)
addRating(…)	addRating(…)
create (…)	Refinement
executeUpdate(…)	executeUpdate(…)

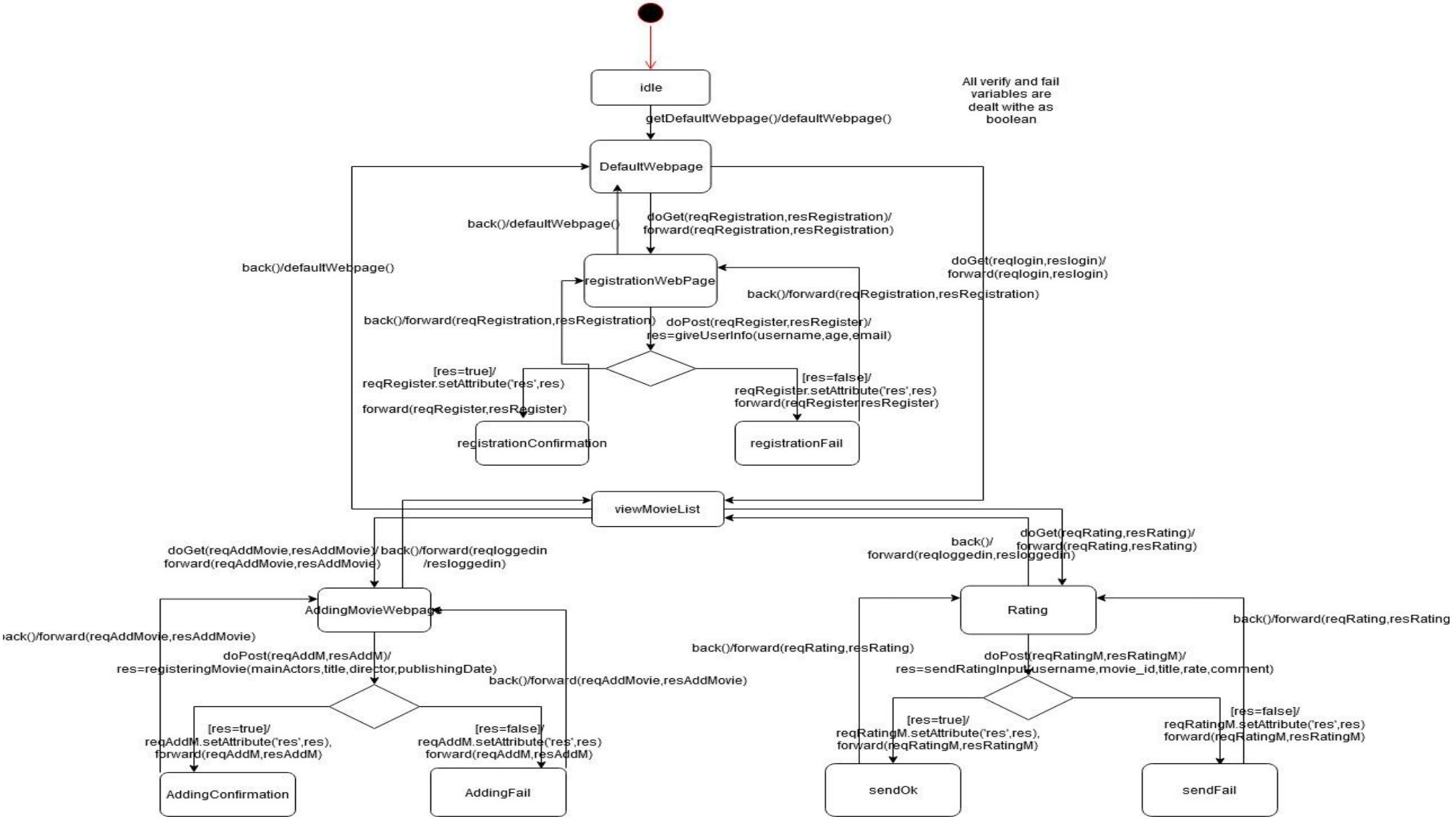
Messages of the sequence diagram sd viewMovieList_intra in step D3: Intra-Component Interaction.	Messages of the sequence diagram sd_requestMovieList in step D2: Inter-Component Interaction
requestMovieList()	requestMovieList()
executeQuery(…)	executeQuery(…)
create (…)	create (…)
executeQuery(…)	Refinement

- -It must be possible to relate any new state (predicates) to the state predicates of Step D2: Inter-Component Interaction.

No new state (predicates) is introduced in step D3: Intra-Component Interaction.

D4: Complete component or class behavior

State Machine UserGUI


idle
All verify and fail variables are dealt withe as boolean
getDefaultWebpage()/defaultWebpage()
DefaultWebpage
back()/defaultWebpage()
doGet(reqRegistration,resRegistration)/ forward(reqRegistration,resRegistration)
back()/defaultWebpage()
registrationWebPage
doGet(reqlogin,reslogin)/ forward(reqlogin,reslogin)
back()/forward(reqRegistration,resRegistration)
back()/forward(reqRegistration,resRegistration)
doPost(reqRegister,resRegister)/ res=giveUserInfo(username,age,email)
[res=true]/ reqRegister.setAttribute('res',res)
forward(reqRegister,resRegister)
[res=false]/ reqRegister.setAttribute('res',res) forward(reqRegister,resRegister)
registrationConfirmation
registrationFail
viewMovieList
doGet(reqAddMovie,resAddMovie)/ forward(reqAddMovie,resAddMovie)
back()/forward(reqloggedin /resloggedin)
back()/ forward(reqloggedin,resloggedin)
doGet(reqRating,resRating)/ forward(reqRating,resRating)
AddingMovieWebpage
Rating
back()/forward(reqAddMovie,resAddMovie)
back()/forward(reqRating,resRating
doPost(reqAddM,resAddM)/ res=registeringMovie(mainActors,title,director,publishingDate)
back()/forward(reqAddMovie,resAddMovie)
back()/forward(reqRating,resRating)
doPost(reqRatingM,resRatingM)/ res=sendRatingInput(username,movie_id,title,rate,comment)
[res=true]/ reqAddM.setAttribute('res',res), forward(reqAddM,resAddM)
[res=false]/ reqAddM.setAttribute('res',res) forward(reqAddM,resAddM)
[res=true]/ reqRatingM.setAttribute('res',res), forward(reqRatingM,resRatingM)
[res=false]/ reqRatingM.setAttribute('res',res) forward(reqRatingM,resRatingM)
AddingConfirmation
AddingFail
sendOk
sendFail

Validation D4:

- The state machines describe the same behavior as in Step D2: Inter-Component Interaction or Step D3: Intra-Component Interaction.

Component UserGUI

Source State	Target State	Input Signal	Mapped to Message(s)	Output Signal	Mapped to Message(s)
idle	DefaultWebpage	doGet(reqDefault,resDefault)	-	forward(reqDefault,resDefault)	-
DefaultWebpage	loggedIn	doGet(reqlogin,..)	doGet(reqlogin,..)	Forward(reqlogin,..)	Forward(reqlogin,..)
DefaultWebpage	registerationWebpage	doGet(reqRegistration,..)	doGet(reqRegistration,..)	Forward(reqRegistration,…)	Forward(reqRegistration,…)
registerationWebpage	DefaultWebpage	back()	-	defaultWebpage()	-
registerationWebpage	registerationFail	doPost(reqRegister,..)	doPost(reqRegister,..)	res=false reqRegister.setAttribute(´res´,res) forward(reqRegister,..)	res=false reqRegister.setAttribute(´res´,res) forward(reqRegister,..)
registerationWebpage	registerationConfirmation	doPost(reqRegister,..)	doPost(reqRegister,..)	res=true reqRegister.setAttribute(´res´,res) forward(reqRegister,..)	res=true reqRegister.setAttribute(´res´,res) forward(reqRegister,..)
registerationFail	registerationWebpage	Back()	-	Forward(reqRegisteration,..)	-

registeration Confirmation	registeration Webpage	Back()	-	Forward(reqRegisteration,..)	-
loggedin	Rating	doGet(reqRating, ..)	doGet(reqRating, ..)	Forward(reqRating,.)	Forward(reqRating,.)
loggedin	DefaultWebpage	Back()	-	defaultWebpage()	-
loggedin	AddingMovieWebpage	doGet(reqAddMovie,..)	doGet(reqAddMovie,..)	Forward(reqAddMovie,.)	Forward(reqAddMovie,.)
Ratting	loggedin	Back()	-	Forward(reqloggedin,..)	-
Rating	sendFail	doPost(reqRatingM,..)	doPost(reqRatingM,..)	Res=sendRatingInput(..) reqRatingM.SetAttribute(…) forward(reqRatingM,..)	Res=sendRatingInput(..) reqRatingM.SetAttribute(…) forward(reqRatingM,..)
Rating	sendOk	doPost(reqRatingM,..)	doPost(reqRatingM,..)	Res=sendRatingInput(..) reqRatingM.SetAttribute(…) forward(reqRatingM,..)	Res=sendRatingInput(..) reqRatingM.SetAttribute(…) forward(reqRatingM,..)
sendFail	Rating	Back()	-	Forward(reqRating,..)	-
sendOk	Rating	Back()	-	Forward(reqRating,..)	-
MovieList	loggedIn	Back()	-	Forward(reqloggedin,..)	-
AddingMovieWebpage	loggedIn	Back()	-	Forward(reqloggedin,…)	-

AddingMovieWebpage	AddingFail	doPost(reqAddM,..)	doPost(reqAddM,..)	Res=registeringMovie(mainActors,..) reqAddM.setAttribute(…) forward(reqAddM,..)	Res=registeringMovie(mainActors,..) reqAddM.setAttribute(…) forward(reqAddM,..)
AddingMovieWebpage	AddingConfirmation	doPost(reqAddM,..)	doPost(reqAddM,..)	Res=registeringMovie(mainActors,..) reqAddM.setAttribute(…) forward(reqAddM,..)	Res=registeringMovie(mainActors,..) reqAddM.setAttribute(…) forward(reqAddM,..)
AddingFail	AddingMovieWebpage	Back()	-	Forward(reqAddMovie,…)	-
AddingConfirmation	AddingMovieWebpage	Back()	-	Forward(reqAddMovie,…)	-

- The state machines are consistent with the life-cycle model of Step A6: Software lifecycle. All states are covered by a life-cycle.

Component UserGUI	
$LC_{user} = sdRegister; (Add \mid Rating \mid MovieList) *$	
State	**Covered by Life Cycle Part**
Init	Register
DefaultWebpage	Register
loggedIn	Add, Ratting, MovieList
Ratting	Ratting
sendFail	Ratting
sendOk	Ratting
MovieList	MovieList
AddMovieWebpage	Add
AddingFail	Add
AddingConfirmation	Add
RegisterationWebpage	Register
RegisterationFail	Register
RegisterationConfirmation	Register

- All transitions are covered by a life-cycle.

Component UserGUI

Source State	Target State	Input Signal	Output Signal	life cycle part
idle	DefaultWebpage	doGet(reqDefault,resDefault)	forward(reqDefault,resDefault)	Register
DefaultWebpage	loggedIn	doGet(reqlogin,..)	Forward(reqlogin,..)	Register
DefaultWebpage	registerationWebpage	doGet(reqRegistration,..)	Forward(reqRegistration, …)	Register
registerationWebpage	DefaultWebpage	back()	defaultWebpage()	Register
registerationWebpage	registerationFail	doPost(reqRegister,..)	res=false reqRegister.setAttribute(´res´,res) forward(reqRegister,..)	Register
registerationWebpage	registerationConfirmation	doPost(reqRegister,..)	res=true reqRegister.setAttribute(´res´,res) forward(reqRegister,..)	Register
registerationFail	registerationWebpage	Back()	Forward(reqRegisteration,..)	Register
registerationConfirmation	registerationWebpage	Back()	Forward(reqRegisteration,..)	Register
loggedin	Rating	doGet(reqRating, ..)	Forward(reqRating,.)	Rating
loggedin	DefaultWebpage	Back()	defaultWebpage()	Rating
loggedin	AddingMovieWebpage	doGet(reqAddMovie,..)	Forward(reqAddMovie,.)	Add

loggedin	MovieList	doPost(reqMovieList,..)	Forward(reqMovieList,..)	Movie List
Ratting	loggedin	Back()	Forward(reqloggedin,..)	Rating
Rating	sendFail	doPost(reqRatingM,..)	Res=sendRatingInput(..) reqRatingM.SetAttribute(…) forward(reqRatingM,..)	Rating
Rating	sendOk	doPost(reqRatingM,..)	Res=sendRatingInput(..) reqRatingM.SetAttribute(…) forward(reqRatingM,..)	Rating
sendFail	Rating	Back()	Forward(reqRating,..)	Rating
sendOk	Rating	Back()	Forward(reqRating,..)	Rating
MovieList	loggedIn	Back()	Forward(reqloggedin,..)	Movie List
AddingMovieWebpage	loggedIn	Back()	Forward(reqloggedin,…)	Add
AddingMovieWebpage	AddingFail	doPost(reqAddM,..)	Res=registeringMovie(mainActors,..) reqAddM.setAttribute(…) forward(reqAddM,..)	Add
AddingMovieWebpage	AddingConfirmation	doPost(reqAddM,..)	Res=registeringMovie(mainActors,..) reqAddM.setAttribute(…) forward(reqAddM,..)	Add
AddingFail	AddingMovieWebpage	Back()	Forward(reqAddMovie,…)	Add
AddingConfirmation	AddingMovieWebpage	Back()	Forward(reqAddMovie,…)	Add

Implementation and Testing

T1

Specifications for the test of UD_Adapter:

For the sdRegister app

- The Component DBFacade accesses the user database.
- Therefore, we decide to use the alternative approach for testing the component.
- We set the database to a defined state to test the operation giveUserInfo(...) which is called by the application.

For the sdRegister intra

- The intra-component specification supposes to distinguish between different objects, but we have in our case only one object which is User Database.
- Since the component requests the object from the database, we have to consider it for preparing the defined state.

Specifications for the test of MD_Adapter:

For the sd app

- The Component MDFacade accesses the user database.
- Therefore, we decide to use the alternative approach for testing the component.
- We set the database to a defined state to test the operations registeringMovie (...), sendRatingInfo(...) and get_MovieList(...) which are called by the application.

For the sd intra

- The intra-component specification supposes to distinguish between different objects, but we have in our case only one object which is Movie Database.
- Since the component requests the object from the database, we have to consider it for preparing the defined state.

Glossary

Name	Type	Description	Source
A			
accesMovieList	Phenomenon, auxiliary function	The user accesses his movie list.	CD, PD, Class Model
accessUserList	phenomenon	It's to access a user list into our database	CD, PD
Add_Movie	class	A connection domain for adding movie	Class Model
Add_Webpage	connectionDomain	It’s connection domain between the user (Biddable domain) and the machine to add a movie into our Moviedatabase.	PD
AddedToDB	State predicate	Film added to database	sd Add
addGroup	phenomenon	The App forwards the request of the user to the group domain to create a new group.	CD
addMember	phenomenon	Group members can add new members to their group.	CD
addMovie	phenomenon	The user adds a new movie to their list in the database.	CD
addRating	phenomenon	The App forwards the request from the user to rate a movie to the domain MovieDatabase.	CD
addRating	phenomenon	The machine adds a new rate to the MovieDatabase.	PD
Administrator	Biddable Domain	Special group members with more rights who administer a group	R16, R18, CD
adminLeaveGroup	phenomenon	The admin leaves the group he has created before.	CD
Age	attribute	Age of the User	Class Model

Name	Type	Description	Source
ApacheTomcat	connection domain	An Open Source JSP and Servlet Container from the Apache Foundation.	TCD
Available	Message, auxiliary function	Checks whether the moviesList exists	Sdrequest_movieList
B			
banMember	phenomenon	An administrator bans a group member from his group.	CD
C			
calcAverage	auxiliary function	Calculate the Average	Class Model
checkLogin	phenomenon	The app checks if the user logs in or out from the user database.	CD
checkRating	auxiliary function	Check the Movie rating	Class Model
createGroup	phenomenon	The user creates a new group for discussion.	CD
D			
DefaultWebpage	state	Indicates the starting page	State Machine UserGUI
Date	Class	Class machine of the Movie date	Class Model
Day	Attribute	Day of the publish	Class Model
DBexists	State predicate	The movie is existed	sd Add
Director	Attribute	Director of the Movie	Class Model
E			
Email	Attribute	Email of the User	Class Model
excuteUpdate()	Message	Java API function to send SQL update command to mySQL database	sd Add_App, sdregister_App
F			
failConfirmation	auxiliary function	Failure of registering movie	Class Model, State machine

Name	Type	Description	Source
failRegister	Phenomenon, auxiliary function	Failure from the the machine to the connection domain	PD, Class Model
feedbackMRAregisterd	phenomenon	UserDatabase sends feedback about the registration.	PD
feedbackUser	phenomenon	The machine gives feedback that the MovieDatabase already forwarded.	CD
forwardAddMember	phenomenon	The web app forwards the request made by the group member to add another user to the group.	CD
forwardAddMovie	phenomenon	The web app forwards the request made by the user to add a movie to their list in database.	CD
ForwardConfirmation	phenomenon	Confirmation from the connection domain about adding new movie	PD
forwardFail	phenomenon	Failure from the connection domain about adding new movie	PD
forwardRegisterMovie	phenomenon	The App forwards the request made by the user to add a movie not in the database to Movie database.	CD, PD
forwardVerifyRegister, forwardFailRegister	phenomenon	The connection domain gives feedback about user-registerarion.	PD
G			
get_MovieList	phenomenon	It's a request to get the movie list from the machine through the connection domain	PD
getMovieList	auxiliary function	To get the Movie List	Class Model
gFeedback	phenomenon	It’s the feedback from the group to the machine to forward it to the GroupMember.	CD

Name	Type	Description	Source
giveUserInfo	Phenomenon, auxiliary function	A RegisterWebpage gives the info that the user already given to the machine	PD, Class Model
Group	DesignedDomain, LexicalDomain	Structure and information about the groups on the website.	CD
GroupMember	biddableDomain	A user that is in a group.	CD
H			
I			
Idle	state	Indicates that the server waits for incoming requests	State Machine UserGUI
J			
K			
L			
LCmachine	Life-Cycle expression		Software Life-cycle
LCuser	Life-cycle expression		Software Life-cycle
leaveGroup	phenomenon		CD
login	phenomenon		CD, PD
logut	phenomenon		CD, PD
M			
mainActors	Attribute	The main actors of the Movie	Class Model
MD_Adapter	component	Responsible to add movie	subArchRegisterMovie subArchRateMovie subArchMovieList globalArch
mdFeedback	phenomenon	It's the feedback from the MovieDatabase to the machine to forward it to the user.	CD

Name	Type	Description	Source
Month	Attribute	Month of the publish of the movie	Class Model
MovieDatabase	DesignedDomain, LexicalDomain, class	Physical representation of the movies, where infromations about movies is stored.	R7, R9, R11, R12, CD, Class Model
movieList	Phenomenon, auxiliary function	The connection domain sends the movie list that the user already accessed.	PD, Class Model
MovieRatingApp	machine		CD, PD, TCD
MR_AddMovie	Machine, class	It's a machine to add a movie into our database	PD, Class Model
MR_ViewMovieList	Machine, class	It is a machine to view a movie list to a user	PD, Class Model
MRA_Rating	Class	Machine Class of rating movies	Class Model
MRA_Register	Machine, class	The machine to register a user.	PD, Class Model
N			
O			
P			
PublishingDate	Attribute	Date of the publish of the movie	Class Model
Q			
R			
rateMovie	Phenomenon, auxiliary function	The user rate a movie	CD, PD, Class Model
Rating	Attribute	Rating of the movie	Class Model
Register	Phenomenon, auxiliary function	A new user registers to use the webapp.	CD, PD, Class Model
RegisteringMovie	Phenomenon, auxiliary function	Register a movie into a machine	CD, PD, Class Model
registerMovie	Phenomenon, auxiliary function	The user adds a movie not in the database to movie database	CD, PD, Class Model

Name	Type	Description	Source
registerUser	phenomenon	The web app registers a new user to the user database.	CD, PD
RegisterationCon-firmation	state	Send a register confirmation	PD, state machine UserGUI
RegisterWebpage	connectionDomain, class	It is a connection between the biddable domain and the machine because the biddable cannot be directly connected to the machine	PD, Class Model
removeMember	phenomenon	The web app removes a member from the group.	CD
requestChat	phenomenon	The user wants to use the chat in the group.	CD
requestMovieList	phenomenon	The web app requests a movie list from the group.	CD, PD
S			
SelectWebpageVie w	State predicate	A list of the Movies is displayed	sdratingMovie_App
sendConfirmation	Phenomenon, auxiliary function	The machine sent confirmation about adding a movie.	PD, Class Model
SendRating	phenomenon	Moviedatabase sends the rating of the movie to the machine to show it to the user	PD
SendRatingInput	Phenomenon, auxiliary function	The connection domain sends the rating that user did to machine to save it in the movieDatabase.	PD, Class Model
setAdmin	phenomenon	The web app sets a group member to an admin.	CD
showFail	phenomenon	The connection domain shows an error occurred while rating the movie.	PD
showOk	phenomenon	The connection domain shows a message that a correct rating was entered and saved to the MoveDatabase.	PD

Name	Type	Description	Source
SQLDatabse	DesignedDomain, LexicalDomain	Database for user and movie registeration	TCD
startChat	phenomenon	The app forwards the request to use the chat functionality in the group.	CD
T			
Title	Attribute	Title of the Movie	Class Model
U			
UD_Adapter	component	Responsible to create user account	subArchRegisterUser globalArch
udFeedback	phenomenon	It's the feedback from the UserDatabase to the machine to forward it to the user	CD
User	Biddable Domain	User of the software	CD, PD, TCD
UserDatabase	DesignedDomain, LexicalDomain, class	Physical representation of the user, where information about user stored	CD, Class Model
UserGUI	component	Web interface for Users	subArchRegisterUser subArchRegisterMovie subArchRateMovie subArchMovieList globalArch
Username	Attribute	Name of the User	Class Model
UserRegistered	State perdicate	The user is registered	Sd register
UserWebBrowser	ConnectionDomain	Web browser used by user	TCD
V			
ValidRating	State predicate	The movie is rated	sd Rating
verifyRegister	Phenomenon, auxiliary function	The machine sends a verification to the connectionDomain to send it to the user	PD, Class Model

Name	Type	Description	Source
view	connetionDomain	It's a connection between the machine to user to view movie list.	PD
viewMovieList	phenomenon	The group shows the movieList of other members in the group.	CD, PD
viewMovieList	phenomenon	The movie database returns a list of movies.	CD
W			
watch	phenomenon	The user watches a movie.	CD
Web Application	Machine	Software to be developed	R3
WebpageView	connectionDomain, class	Connection domain between the user and the machine to show the rating.	PD, Class Model
X			
Y			
Year	Attribute	Year of the publish of the movie	Class Model
Z			