Movie Rating App

Software Engineering Lab WS21/22

Mohamed Abokahf (3101712), Islam Abdelghaffar (3102002), Mostafa Mohamed (3101359), Abdelrahman Salam (3100657), Hadeel Almasri (3026936), Walla Alsibai (3060416).

Lab Time: Mo. 12-14

Supervisor: Roman Wirtz

Date of submission: 11.02.2022

Contents

List of figures

1 Analysis	1.1 A1	

1.1.1 Requirements & Domain-Knowledge

Requirements:

- R1: The web application enables persons who are at least 18 years old to register; otherwise, registration will fail.
- R2: The User can add a movie if it is not on the database, by providing the movie's title, director, the main actor (at least one), and original publishing.
- R3: The web application enables the user to rate movies he /she has seen.
- R4: The user can choose movies he/she has seen from a list if it exists in the list and rates it.
- R5: Group members can see the movies' list of other members.
- R6: Group members can discuss a movie in form of a group chat.
- R7: The user can access a list of all movies stored on the webserver and sorted in descending order.
- R8: Any registered user can add other registered users into a movie discussion group.
- R9: The user can leave a movie discussion group if he/she wants to.
- R10: The administrator can ban members from a movie discussion group when they misbehave
- R11: The groups will be deleted if administrators leave them.
- R12: Any group will be deleted automatically after a certain amount of time if it contains only one member.
- R13: The user can login using his/her email address and username.
- R14: The users can end the session by logout.

Assumptions:

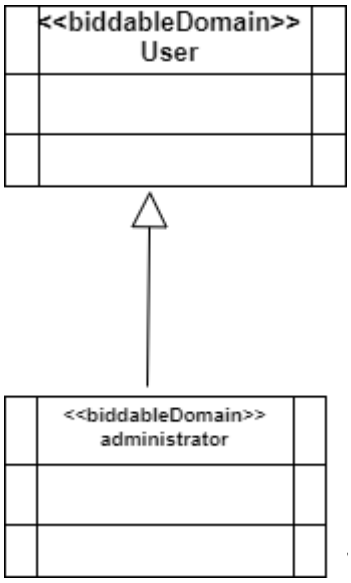
- A1: The users should rate the movies they have already seen.
- A2: The administrators' decisions are always fair.
- A3: In movie discussion groups the members are allowed to discuss movies related topics.

Facts:

- F01: The users must be at least 18 years old, otherwise registration fails.
- F02: The username is unique.
- F03: The rating ranges from 1 to 10, where 1: bad and 10: excellent.
- F04: The movies without any rating are rated as zero.
- F05: Each movie must have title, director, at least one main actor, and the original publishing date.
- F06: Each movie is contained and rated from each user only once.
- F07: Each movie discussion group has a unique name and list of group members.
- F08: Each chat has the user's name of its creator and includes a timestamp and content.
- F09: Each group creator is an administrator.

Generalization / specialization:

Each administrator is considered as a user.



1.1.2 Context diagram:

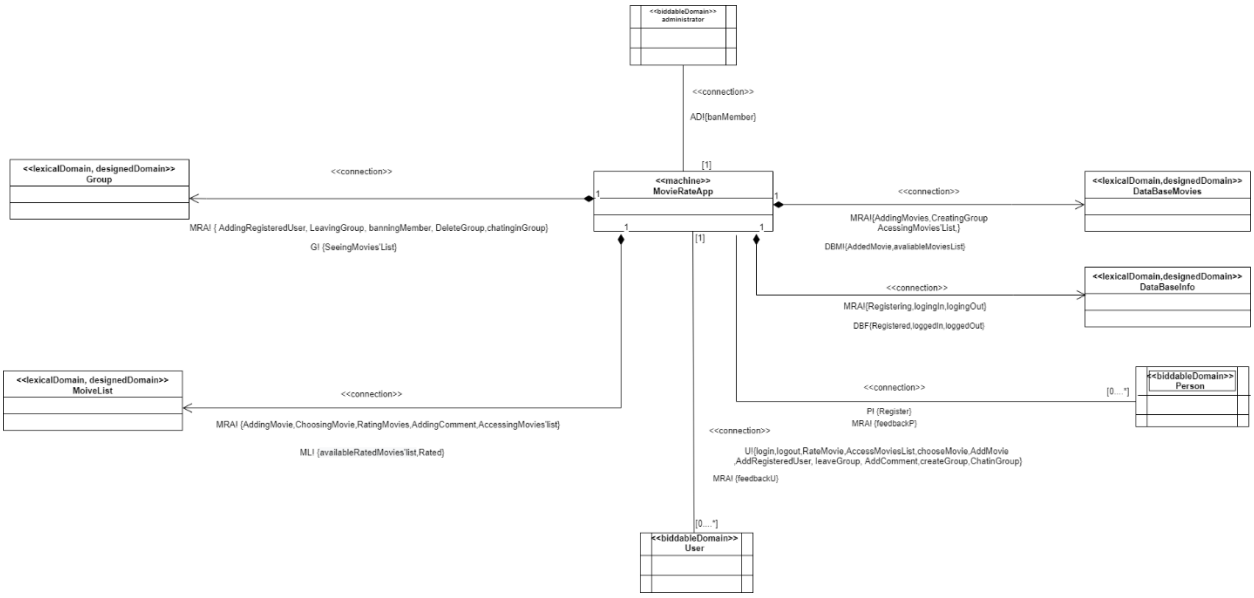


Figure 1.1 Context diagram

1.1.3 validation for A1:

- The glossary contains the notions used in R and D. The notions mentioned in R and D are contained in the glossary.
- Domains and phenomena of the context diagram must be consistent with R and D.

Notion in CD	Notions in R/D	Type
AddingRegisteredUser	Counterpart to AddRegisteredUser	phenomenon
AvailableMoviesList	Counterpart to ChoosingMovie	phenomenon
AddingComment	Counterpart to AddComment	phenomenon

AddComment	A user can add a comment to a movie	phenomenon
LeavingGroup	Counterpart to leaveGroup	phenomenon
banningMember	Counterpart to banMember	phenomenon
DeleteGroup	Can delete Group	phenomenon
Registering	Counterpart to Register	phenomenon
ChoosingMovies	Counterpart to chooseMovie	phenomenon
AddingMovie	Counterpart to AddMovie	phenomenon
AcessingMovies'List	Counterpart to AccessMoviesList	phenomenon
Register	Can register a person	phenomenon
login	Can login a user	phenomenon
logout	Can logout a user	phenomenon
RateMovie	Can rate a movie	phenomenon
AccessMoviesList	Can Access a movies' list	phenomenon
chooseMovie	Can choose a movie	phenomenon
AddMovie	Can add a movie	phenomenon
AddRegisteredUser	Can add a registered user	phenomenon
leaveGroup	Can leave a group	phenomenon
banMember	Can ban a member	phenomenon
logingIn	Counterpart to login	phenomenon
logingOut	Counterpart to logout	phenomenon

RatingMovies	Counterpart to RateMovie	phenomenon
MovieRateApp	The software we are going to build	domain
User	user	domain
Person	Person	domain
administrator	administrator	domain
MoiveList	Movie list	domain
DataBaseInfo	Data base for user's information	domain
DataBaseMovies	Data base for Movie's information	domain
Group	group	domain
CreateGroup	User can create a group	phenomenon
CreatingGroup	Counterpart to CreateGroup	phenomenon
ChatinGroup	User can chat in a group	phenomenon
ChatinginGroup	Counterpart to ChatinGroup	phenomenon
AddedMovie	Successful AddingMovie	phenomenon
avaliableRatedMovies'list	Available Movielist	phenomenon
Registered	successful Registration	phenomenon
loggedIn	successful loggingIn	phenomenon
loggedOut	successful loggingOut	phenomenon
Rated	successful RatingMovie	phenomenon
feedbackU	introduced to provide feedback to Users	phenomenon

feedbackP	Introduced to provide feedback to unregistered users	phenomenon
SeeingMovies'List	Counterpart to AccessMovieList	phenomenon

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

Domain	Domain Type(s)	Connected Domain(s)	Connected Domain(s) Type(s)
MovieRateApp	Machine domain	administrator	biddable domain
		user	biddable domain
		Person	biddable domain
		DatabaseInfo	Lexical Domain, designed Domain
		DatabaseMovie	Lexical Domain, designed Domain
		Group	Lexical Domain, designed Domain
		MovieList	Lexical Domain, designed Domain
User	Biddable Domain	MovieRateApp	Machine Domain
Person	Biddable Domain	MovieRateApp	Machine Domain
administrator	Biddable Domain	MovieRateApp	Machine Domain

MoiveList	Lexical Domain, designed Domain	MovieRateApp	Machine Domain
DataBaseInfo	Lexical Domain, designed Domain	MovieRateApp	Machine Domain
DataBaseMovie	Lexical Domain, designed Domain	MovieRateApp	Machine Domain
Group	Lexical Domain, designed Domain	MovieRateApp	Machine Domain

- The machine domain must control at least one interface. MovieRateApp controls several interfaces (RegisteringUser, AddingMovies, AcessingMovies'List, logingIn, logingOut…….)
- 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(s) Type(s)
MovieRateApp	Machine domain	administrator	biddable domain
		user	biddable domain
		Person	biddable domain
		Database	Lexical Domain, designed Domain
		Group	Lexical Domain, designed Domain
		MovieList	Lexical Domain, designed Domain
User	Biddable Domain	MovieRateApp	Machine domain

Person	Biddable Domain	MovieRateApp	Machine domain
administrator	Biddable Domain	MovieRateApp	Machine domain
MoiveList	Lexical Domain, designed Domain	MovieRateApp	Machine Domain
DataBaseInfo	Lexical Domain, designed Domain	MovieRateApp	Machine Domain
DataBaseMovie	Lexical Domain, designed Domain	MovieRateApp	Machine Domain
Group	Lexical Domain, designed Domain	MovieRateApp	Machine Domain

- Causal, designed, lexical, display, machine domain types are not allowed together with biddable domain. User and administrator are biddable domains only.
- Phenomena controlled by a biddable domain must have counterpart phenomena located between machine and causal/lexical/designed domains.

	Biddable domain phenomena	**counterpart**
user	login	logingIn
	logout	logingOut
	RateMovie	RatingMovies
	AccessMoviesList	AccessMovies'list
	chooseMovie	ChoosingMovie
	AddMovie	AddingMovies

	AddRegisteredUser	AddingRegisteredUser
	leaveGroup	LeavingGroup
	Addcomment	AddingComment
	createGroup	CreatingGroup
	chatInGroup	ChatingInGroup
administrator	banMember	baningMember
Person	Register	Registering

- 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. There are no connection domains.
- 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.

1.2 A2		

1.2.1 Problem Diagram:

R1: The web application enables persons who are at least 18 years old to register; otherwise, registration will fail via registration web page where each person must provide Email, unique name, and age.

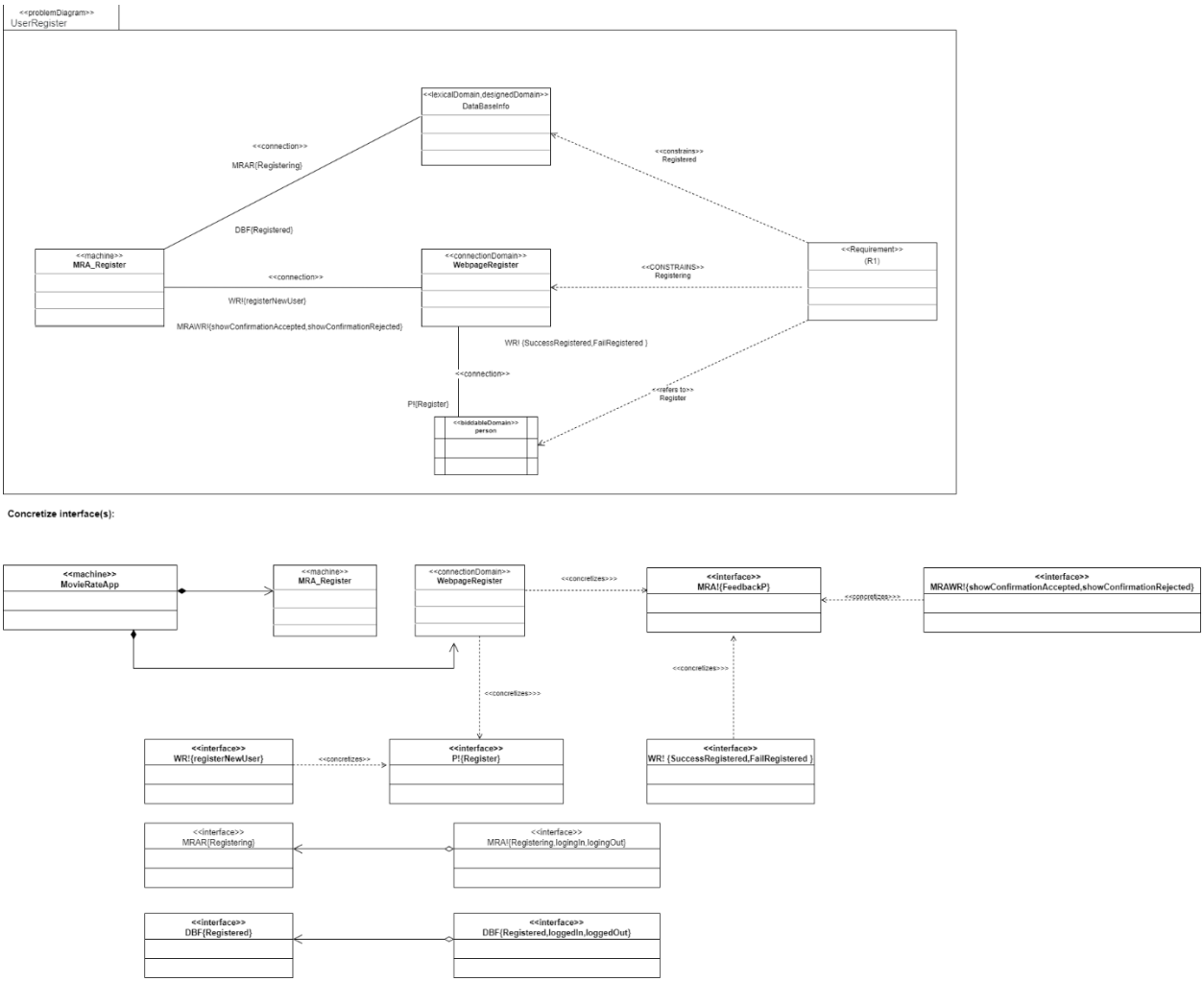
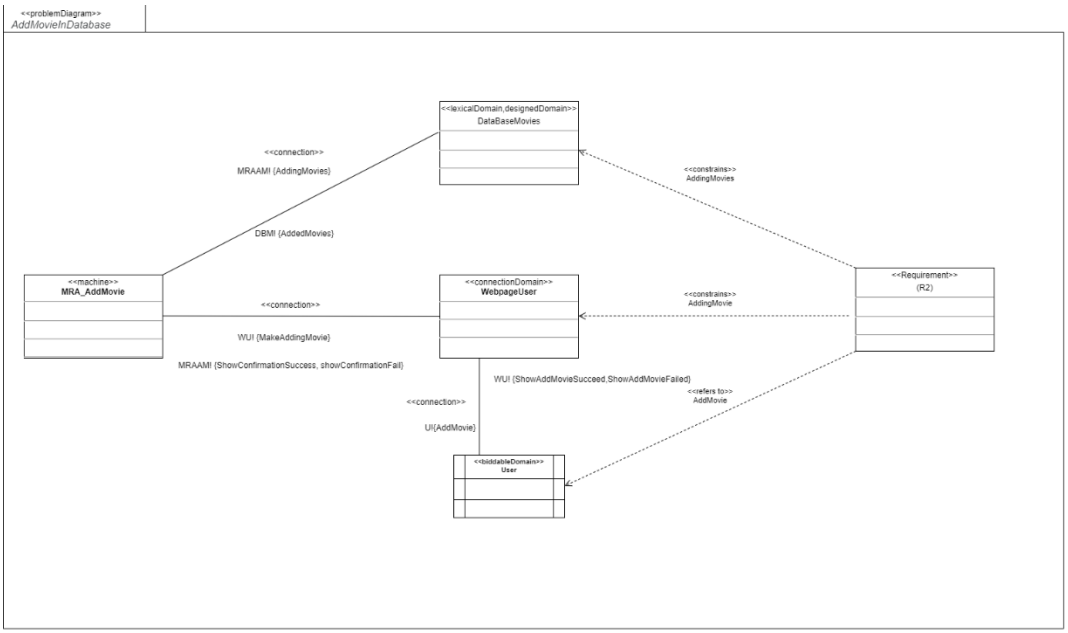


Figure 2Problem diagram and mapping for R1

- (R1) **Fits** to **update** (2), because an operator should be able to update or add information to a database.



Concretize interfaces(s):

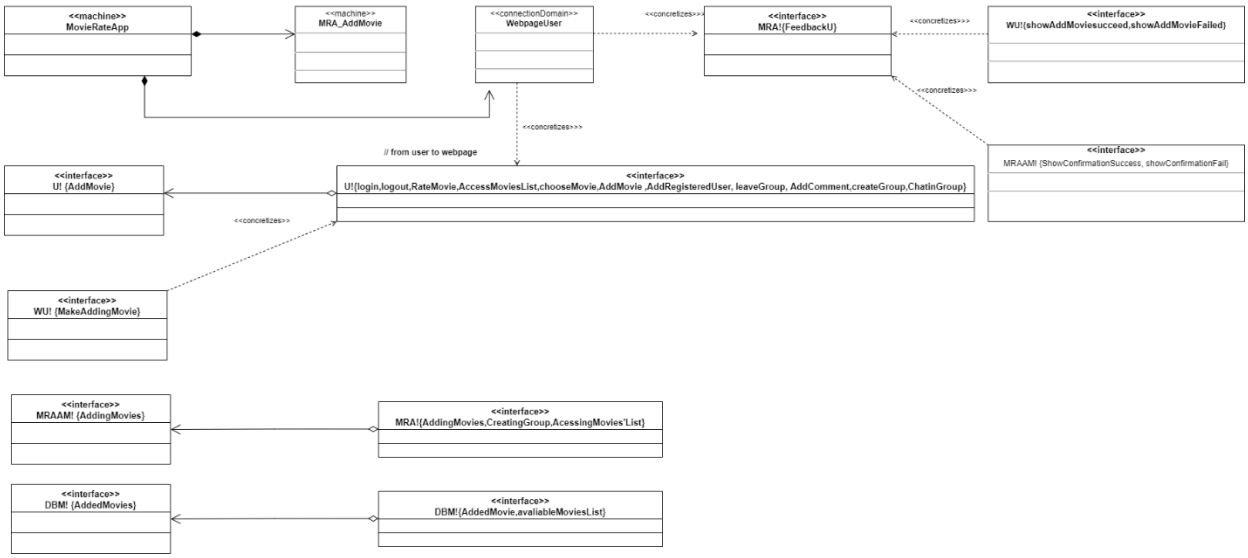


Figure 3 Problem diagram and mapping for R2

R2: The User can add a movie if it is not on the database, by providing the movie's title, director, the main actor (at least one), and original publishing date.

- (R2) **Fits** to **update** (2), because an operator should be able to update or add information to a database.

R3: The web application enables the user to rate movies he /she has seen via user web page by choosing a number from 1 to 10, The movies without any rating are rated as zero.

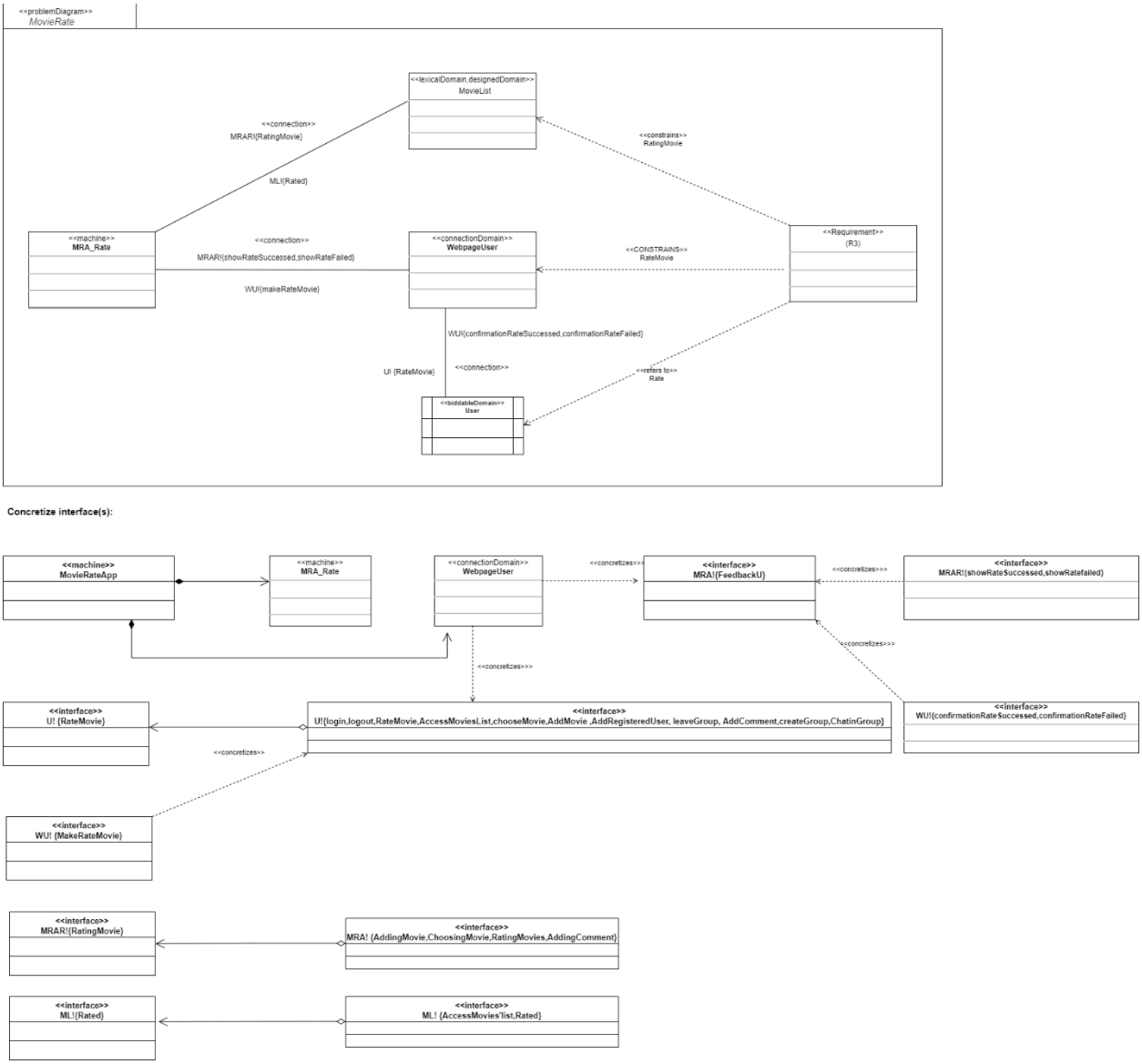


Figure 4 problem diagram and mapping for R3

- **R (3) Fits** to **update** (2), because an operator should be able to update or add information to a database.

R7: The user can access a list of all movies stored on the webserver and sorted in descending order.

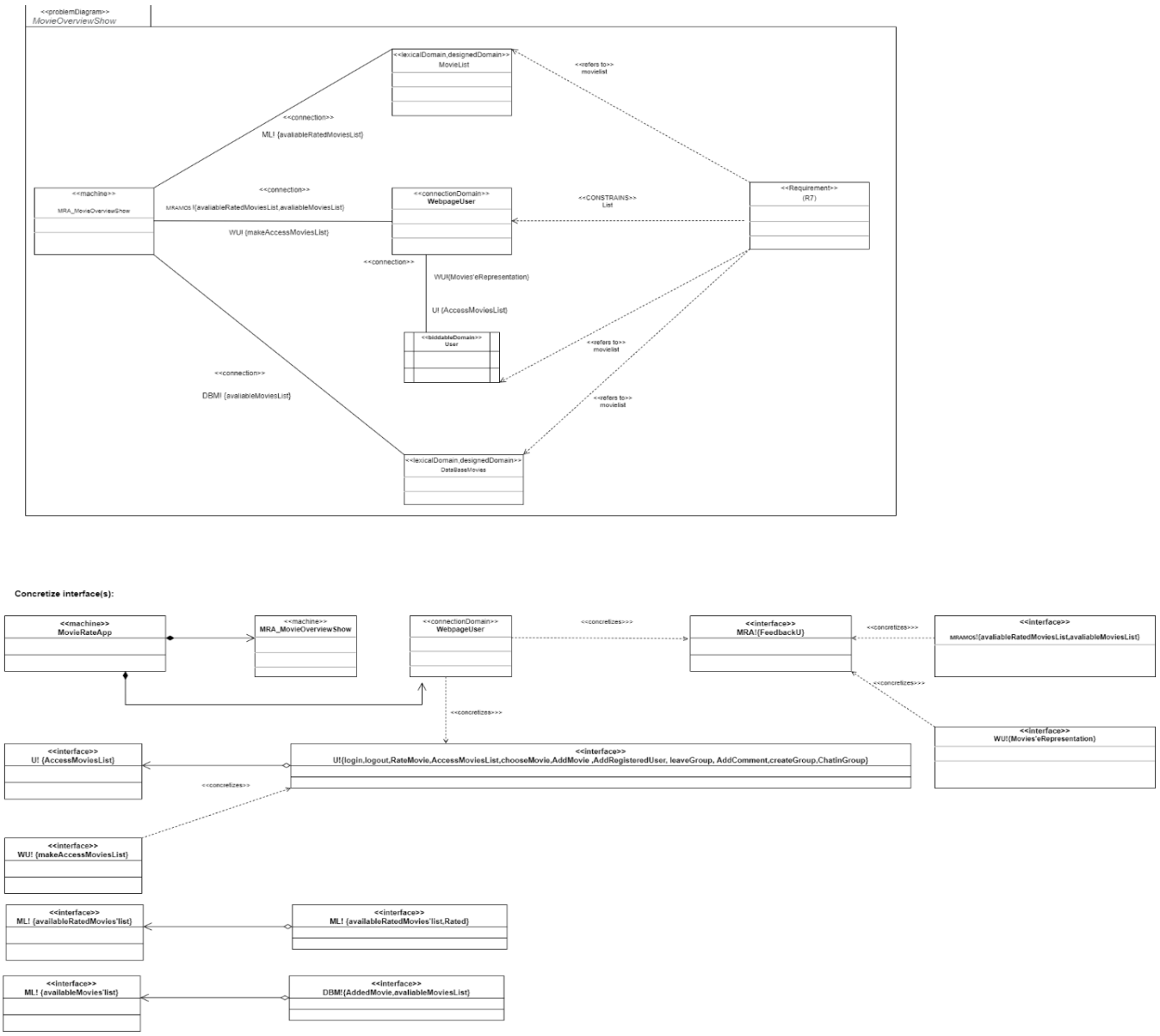


Figure 5 Problem diagram and mapping for R7

- R (7) **Fits** to **query** (2), because an operator queries information from a database.

1.2.2 Validation of A2:

- All requirements R are covered in some subproblem

Requirement	Covered in	Contained domain	Domain type	constrained	Controlled Phenomenon
R1	PD-UserRegister	Person	Biddable domain		Register
		WebpageRegister	Connection Domain	x	SuccessRegistered, FailRegistered, RegisterNewUser
		MRA_Register	Machine		showConfirmationAccepted showConfirmationRejected Registering
		DatabaseInfo	Lexical and designed domain	x	Registered
R2	PD-AddMovieInDatabase	User	Biddable Domain		AddMovie
		WebpageUser	Connection Domain	x	ShowAddMovieSucceed ShowAddMovieFailed MakeAddingMovie
		MRA_AddMovie	Machine		ShowConfirmationSuccess ShowConfirmationFail AddingMovies
		DataBaseMovies	Lexical and designed Domain	x	AddedMovies

R3	PD-MovieRate	User	Biddable Domain		RateMovie
		WebpageUser	Connection Domain	x	confirmationRateSucceed, confirmationRateFailed, makeRateMovie
		MRA_Rate	machine		showRateSuccessed, showRateFailed, RatingMovie
		MovieList	Lexical and designed Domain	x	Rated
R7	PD-MovieOverviewShow	User	Biddable Domain		AccessMoviesList
		WebpageUser	Connection Domain	x	Movies'eRepresentation makeAccessMoviesList
		MRA_MovieOverviewShow	Machine		availableRatedMoviesList, availableMoviesList
		DatabaseMovie	Lexical and designed Domain		availableMoviesList
		MovieList	Lexical and designed Domain		availableRatedMoviesList

- A Problem Diagram has exactly one machine Domain
- A problem Diagram contains at least one requirement. It contains 4.
- The machine domain must control at least one interface.

- Requirements constrain at least one domain. It constrains in our case 7 domains.
- Requirements don't constrain machines and it is our case.
- If Requirements do constrain biddable domain, a good argument is given and documented.

Requirement	Covered in	Contained domain	Domain type	constrained	Controlled Phenomenon
R1	PD-UserRegister	Person	Biddable domain		Register
		WebpageRegister	Connection Domain	x	SuccessRegistered, FailRegistered, RegisterNewUser
		MRA_Register	Machine		showConfirmationAccepted showConfirmationRejected Registering
		DatabaseInfo	Lexical and designed Domain	x	Registered
R2	PD-AddMovieInDatabase	User	Biddable domain		AddMovie
		WebpageUser	Connection Domain	x	ShowAddMovieSucceed ShowAddMovieFailed MakeAddingMovie
		MRA_AddMovie	Machine		ShowConfirmationSuccess ShowConfirmationFail AddingMovie
		DataBaseMovies	Lexical and designed Domain	x	AddedMovies
R3	PD-MovieRate	User	Biddable domain		RateMovie
		WebpageUser	Connection Domain	x	confirmationRateSuccessed, confirmationRateFailed, makeRateMovie
		MRA_Rate	Machine		showRateSuccessed,

					showRateFailed, RatingMovie
		MovieList	Lexical and designed Domain	x	Rated
R7	PD-MovieOverviewShow	User	Biddable domain		AccessMoviesList
		WebpageUser	Connection Domain	x	Movies'eRepresentation, makeAccessMoviesList
		MRA_MovieOverviewShow	Machine		availableRatedMoviesList, availableMoviesList
		DatabaseMovie	Lexical and designed Domain		availableMoviesList
		MovieList	Lexical and designed Domain		availableRatedMoviesList

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

Connection domain	Phenomenon controlled by connection domain	Connected domains	Phenomenon controlled by connected domains
WebpageUser	SuccessRegistered	Person	Register
	FailRegistered	Person	Register
	registerNewUser	MRA_Register	showConfirmationAccepted showConfirmationRejected Registering
WebpageUser	ShowAddMovieSucceed	User	AddMovie

	ShowAddMovieFailed	User	AddMovie
	MakeAddingMovie	MRA_AddMovie	ShowConfirmationSuccess ShowConfirmationFail AddingMovies
WebpageUser	confirmationRateSuccessed	User	RateMovie
	confirmationRateFailed	User	RateMovie
	makeRateMovie	MRA_Rate	showRateSuccessed, showRateFailed, RatingMovie
WebpageUser	Movies'eRepresentation	User	AccessMoviesList
	makeAccessMoviesList	MRA_MovieOverviewShow	availableRatedMoviesList, availableMovieList

- The problem diagram must be consistent to the context diagram. Each machine of the problem diagram is a part of 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 phenomenon
PD-UserRegister	Leave out domain	User, administrator, DatabaseMovies, Group and Movielist
	Introduce connection / display domain	WebpageUser
	Split interface	MRAR!{.....}, DBF!{.....}
	Concretize interface	MRA! {....}, P! {....}
PD-AddMovieInDatabase	Leave out domain	Group, Movielist, DatabaseInfo and Person
	Introduce connection / display domain	WebpageUser
	Split interface	MRAAM! {.....} U! {.....} DBM! {.....}
	Concretize interface	U! {.....} MRA! {.....}

PD-MovieRate	Leave out domain	Group, DatabaseMovies, DatabaseInfo and Person
	Introduce connection / display domain	WebpageUser
	Split interface	MRAR! {.....} ML! {.....} U! {.....}
	Concretize interface	U! {.....} MRA! {.....}
PD-MovieOverviewShow	Leave out domain	Group, Movielist, DatabaseInfo and Person
	Introduce connection / display domain	WebpageUser
	Split interface	ML! {.....} U! {.....}
	Concretize interface	U! {.....} MRA! {.....}

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	Connections in PD	Connections in PF	Domain Type 1	Domain Type 2
UserRegister	Update 2	MRAR{Registering}, DBF{Registered}	DB! Y1, UM! E2	Machine	Lexical Domain
		WR! {registerNewUser} MRAWR! {showConfirmationAccepted, showConfirmation}	UM! E4, IOD! E8	Machine	Connection Domain
		P! {Register} WR! {SuccessRegistered, FailRegistered}	UO! E6, IOD! C7	Biddable Domain	Connection Domain
AddMovieInDatabase	Update 2	MRAAM! {AddingMovies} DBM! {Adde	DB! Y1, UM! E2	Machine	Lexical Domain

		WU! {MakeAddingMovie} MRAAM! {ShowConfirmationSuccess, showConfirmationFail}	UM! E4, IOD! E8	Machine	Connection Domain
		WU! {ShowAddMovieSucceed, ShowAddMovieFailed} U! {AddMovie}	UO! E6, IOD! C7	Biddable Domain	Connection Domain
MovieRate	Update 2	MRAR! {RatingMovie} ML! {Rated}	DB! Y1, UM! E2	Machine	Lexical Domain
		MRAR! {showRateSuccessed, showRateFailed} WU! {makeRateMovie}	UM! E4, IOD! E8	Machine	Connection Domain
		WU! {confirmationRateSuccessed, confirmationRateFailed} U! {RateMovie}	UO! E6, IOD! C7	Biddable Domain	Connection Domain
MovieOverviewShow	Query 2 (MovieList&DatabaseMovies merged)	ML! {avaliableRatedMoviesList} DBM! {avaliableMoviesList}	DB! Y1	Machine	Lexical Domain
		MRAMOS! {avaliableRatedMoviesList, avaliableMoviesList} WU! {makeAccessMoviesList}	QM! Y3, IOD! C6	Machine	Connection Domain
		WU! {Movies'eRepresentation} U! {AccessMoviesList}	IOD! E7, EO! E5	Biddable Domain	Connection Domain

- 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
UserRegister	Update 2	DataBaseInfo	Data Base	Lexical Domain
		WebpageRegister	Input Output Device	Connection Domain

AddMovieInDatabase	Update 2	DataBaseMovies	Data Base	Lexical Domain
		WebpageUser	Input Output Device	Connection Domain
MovieRate	Update 2	MovieList	Data Base	Lexical Domain
		WebpageUser	Input Output Device	Connection Domain
MovieOverviewShow	Query 2	WebpageUser	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 Type
UserRegister	Update 2	Person	Update Operator	Biddable Domain
AddMovieInDatabase	Update 2	User	Update Operator	Biddable Domain
MovieRate	Update 2	User	Update Operator	Biddable Domain
MovieOverviewShow	Query 2	User	Enquiry Operator	Biddable Domain
		MovieList	Data Base	Lexical Domain
		DataBaseMovies	Data Base	Lexical Domain

1.3 A3		

R1: The web application enables persons who are at least 18 years old to register; otherwise, registration will fail via registration web page where each person must provide Email, unique name, and age.

1.3.1 Specifications:

WebpageRegister(S01a)

When the webpage receives the command "Register", then the command is forwarded to the machine with the command "registerNewUser". The feedback whether the request was succeeded or not is received via the commands "showConfirmationAccepted" and "showConfirmationRejected". Both feedback types are shown to the person via the commands "SuccessRegistered" and "FailRegistered".

MRA_Register(S01b)

When the machine receives the command "registerNewUser", it will be checked if the person's name is unique or not with the command "get_Registered" and if the age is above or equal 18. The result is based on the received data "Registered" which includes all registered names. If the person's name is unique and his/her age >=18, then the machine sets it as a new user with command "Registering", the machine informs the user with the command "showConfirmationAccepted" to the Registration's webpage about the successful registering process.

Otherwise, the machine informs the user with the command "showConfirmationRejected" to the Registration's webpage about the unsuccessful registering process.

DataBaseInfo(S01c)

When receiving the command "get_Registered", the Registered users' information's are returned as the data "Registered". When the command "Registering" is received, a new user will be added to DatabaseInfo.

Correctness condition:

(S01a) ∧ (S01b) ∧ (S01c) ⇒ (R01)

1.3.2 Sequence diagram:

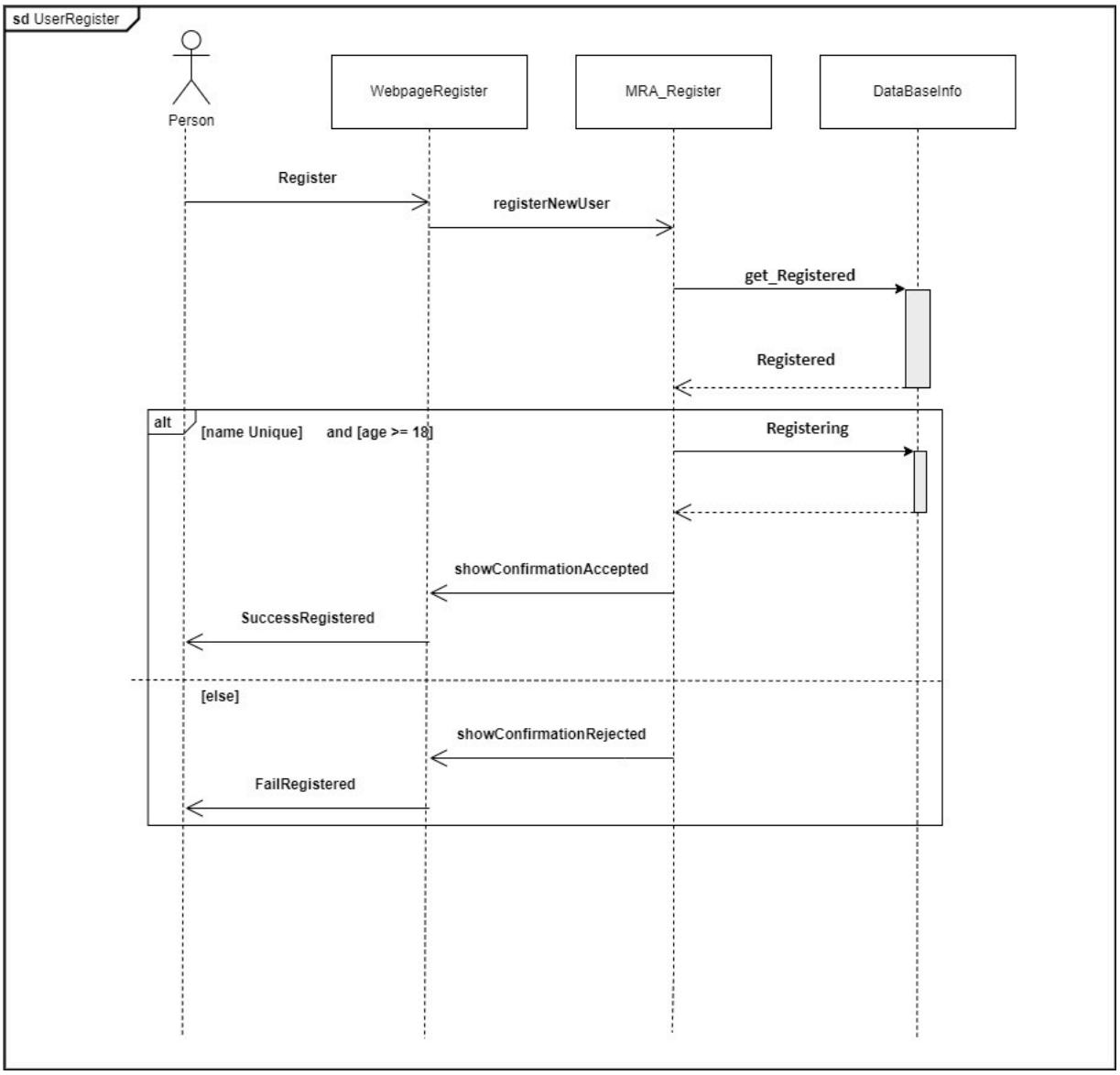


Figure 6 Sequence diagram for R1

R2: The User can add a movie if it is not on the database, by providing the movie's title, director, the main actor (at least one), and original publishing date.

1.3.1 Specifications:

WebpageUser(S02a)

When the webpage receives the command “AddMovie”, then the command is forwarded to the machine with the command “MakeAddingMovie”. The feedback whether the request was succeeded or not is received via the commands “ShowConfirmationSuccess” and “showConfirmationFail”. Both feedback types are shown to the User via the commands “ShowAddMovieSucceed” and “ShowAddMovieFailed”.

MRA_AddMovie(S02b)

When the machine receives the command “MakeAddingMovie”, if the user entered all required data needed to add new movie, it is checked if the movie already exists in Movies’ database or not with the command “get_AddedMovies” and the result is received as the data “AddedMovies”. If the Movie is not in the movies’ database, then the machine sets it as a new movie with command “AddingMovies”, the machine informs the user with the command “ShowConfirmationSuccess” to the User webpage about the successful Adding Movie process.

Otherwise, the machine informs the user with the command “showConfirmationFail” to the User web page about the unsuccessful Adding Movie process.

DataBaseMovies (S02c)

When receiving the command “get_AddedMovies”, the Added Movies information are returned as the data “AddedMovies”. When the command “AddingMovies” is received, a new Movie will be added to

DataBaseMovies.

Correctness condition: (S02a) ∧ (S02b) ∧ (S02c) ⇒ (R02)

1.3.2 Sequence diagram:

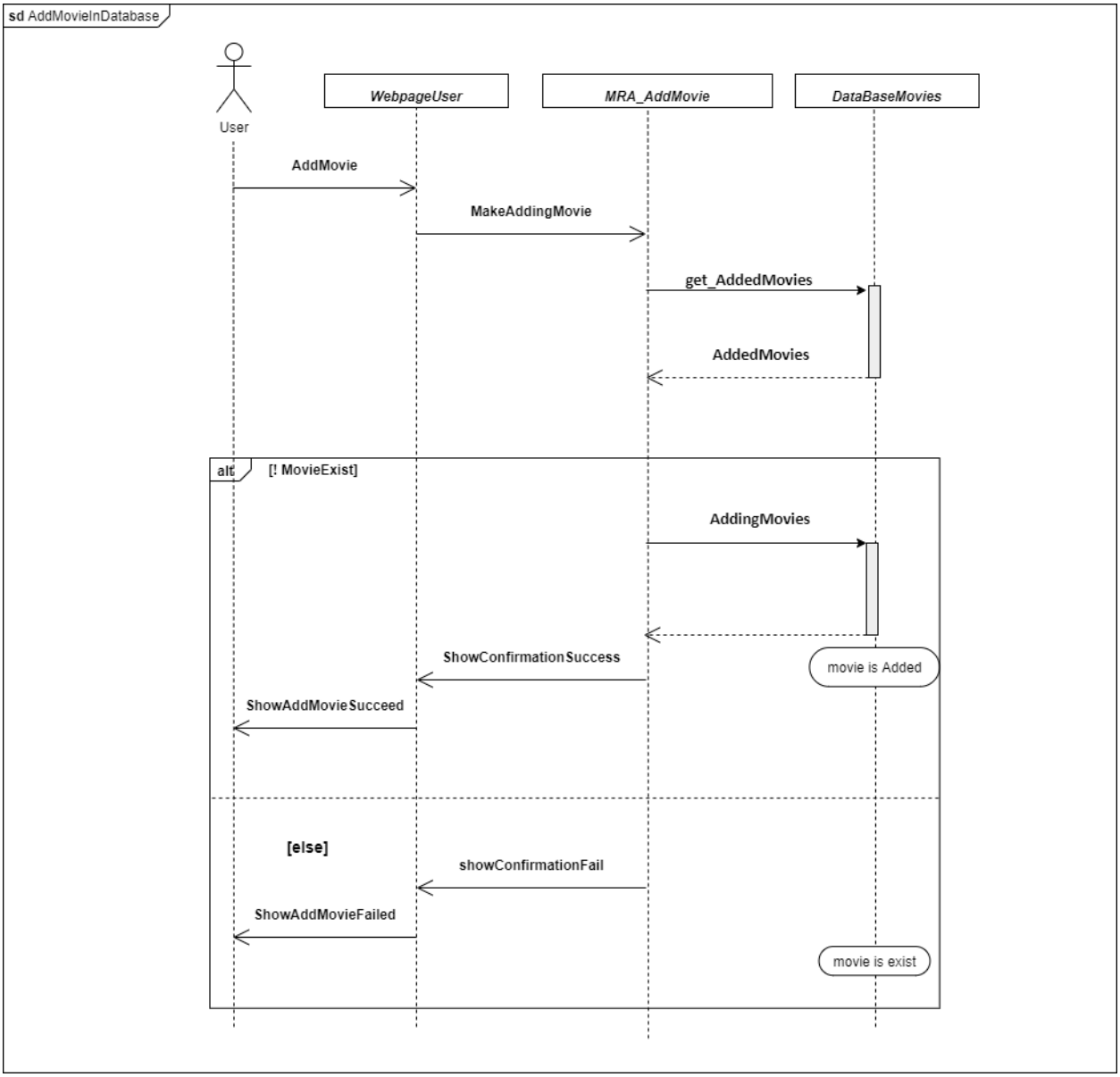


Figure 7 Sequence diagram for R2

R3: The web application enables the user to rate movies he /she has seen via user web page by choosing a number from 1 to 10, The movies without any rating are rated as zero.

1.3.1 Specifications:

WebpageUser(S03a)

When the webpage receives the command "RateMovie", then the command is forwarded to the machine with the command "makeRateMovie". The feedback whether the request was succeeded or not is received via the commands "showRateSuccessed" and "showRateFailed". Both feedback types are shown to the user via the commands "confirmationRateSuccessed" and "confirmationRateFailed".

MRA_Rate(S03b)

When the machine receives the command "makeRateMovie", it is checked if the movie is not already rated from this user, with the command "get_Rated" and by checking if the given Rate between (1 and 10), the result will be based on data "Rated". If the Movie is not rated already from this user, then the machine rates it with command "RatingMovie", the machine informs the user with the command "showRateSuccessed" to the User webpage about the successful Rating Movie process.

Otherwise, the machine informs the user with the command "showRateFailed" to the User webpage about the unsuccessful Rating Movie process.

MovieList (S03c)

When receiving the command "get_Rated", the Rated Movies information's are returned as the data "Rated". When the command "RatingMovie" is received, the movie will be rated in movies' list.

Correctness condition: (S03a) ∧ (S03b) ∧ (S03c) ⇒ (R03).

1.3.2 Sequence diagram:

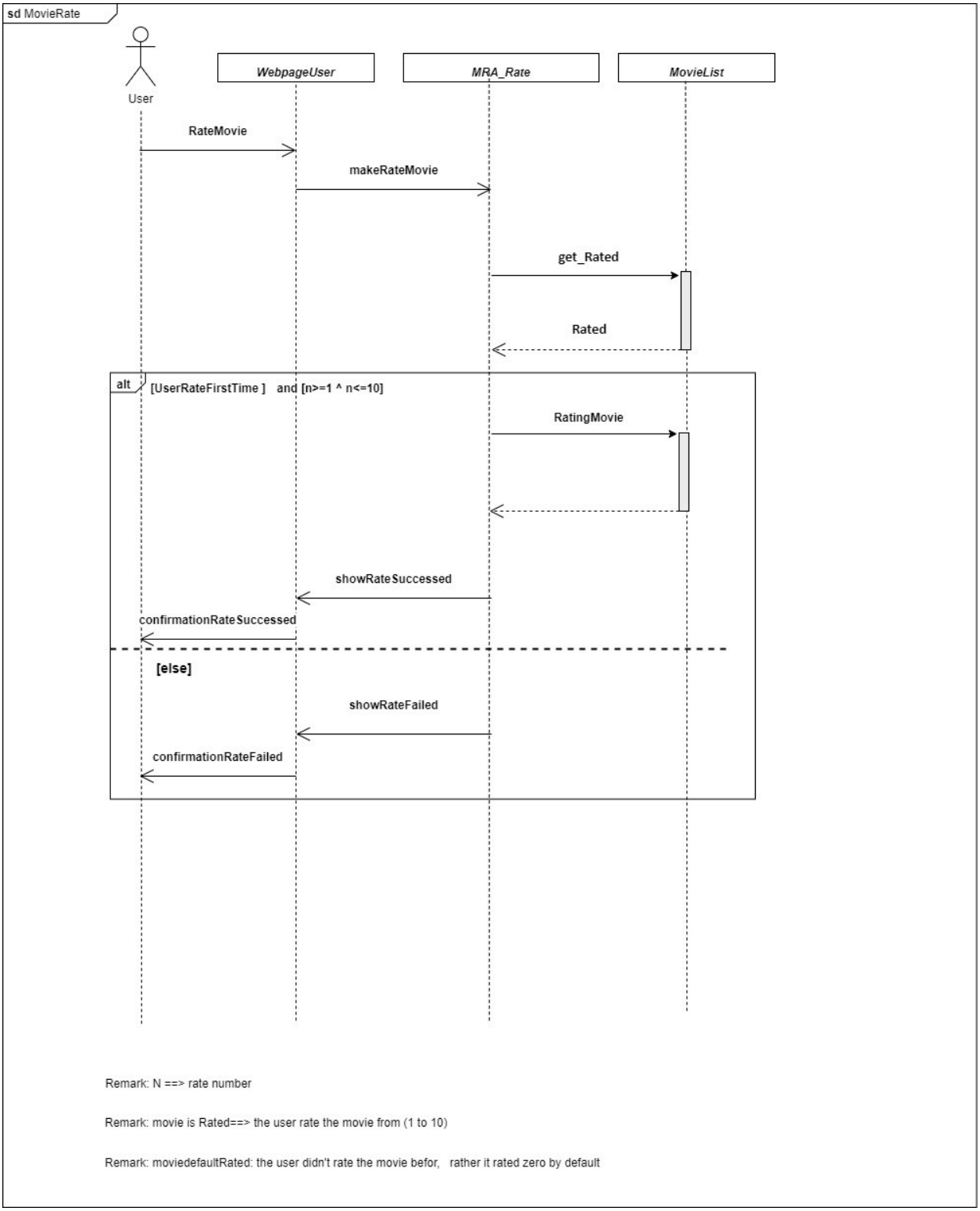


Figure 8 Sequence diagram for R3

R7: The user can access a list of all movies stored on the webserver and sorted in descending order.

1.3.1 Specifications:

WebpageUser(S07a)

When the webpage receives the command "AccessMoviesList", then the command is forwarded to the machine with the command "makeAccessMoviesList". The results are received via the command "avaliableRatedMoviesList" and "avaliableMoviesList" and shown to the guest by "Movies'eRepresentation".

MRA_MovieOverviewShow(S07b)

When the machine receives the command "makeAccessMoviesList", the availableRatedMoviesList and available MoviesList are selected and with the command "get_ availableMoviesList"and command "get_ availableRatedMoviesList" and received as the data "avaliableRatedMoviesList" and "avaliableMoviesList". The results are returned via the command "AvaliableRatedMoviesList" and "AvaliableMoviesList".

MovieList(S07c)

After receiving the command "get_ availableRatedMoviesList" the results are returned as the data "avaliableRatedMoviesList".

DataBaseMovies (S07d)

After receiving the command "get_ availableMoviesList" the results are returned as the data "avaliableMoviesList".

Correctness condition: (S07a) ∧ (S07b) ∧ (S07c) ∧ (S07d) ⇒ (R07)

1.3.2 Sequence diagram:

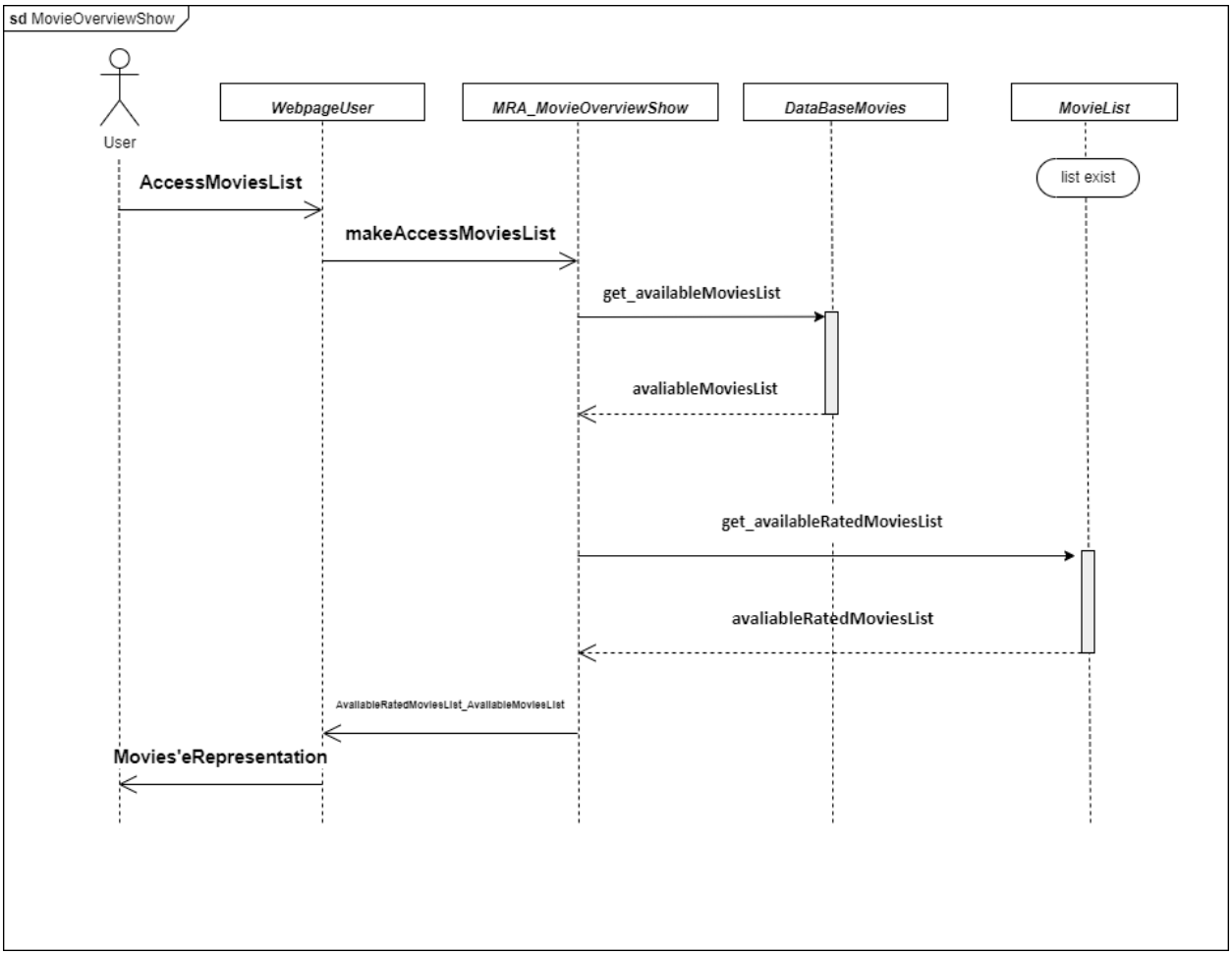


Figure 9 Sequence diagram for R7

1.3.3 Validation for A3:

Validation	S abstract ∧ D are non-contradictory, S abstract ∧ D ==⇒ R.
	Messages and phenomena are consistent.
	Lexical domains are not sources of messages.
	There exists at least one scenario for each subproblem.
	Scenarios cover normal cases and possibly exceptional cases.

- S abstract ∧ D are non-contradictory. No contradictions can be found in S abstract ∧ D.
- S abstract ∧ D ==⇒ R.
  (S01a) ∧ (S01b) ∧ (S01c) ⇒ (R01)
  (S02a) ∧ (S02b) ∧ (S02c) ⇒ (R02)
  (S03a) ∧ (S03b) ∧ (S03c) ⇒ (R03)
  (S07a) ∧ (S07b) ∧ (S07c) ∧ (S07d) ⇒ (R07)
- Messages and Phenomena are consistent.

Message in scenario	**source**	**target**	**phenomena in problem diagram**
Register	Person	WebpageRegister	P! {Register}
registerNewUser	WebpageRegister	MRA_Register	WR! {RegisterNewUser}
get_Registered	MRA_Register	DataBaseInfo	DBF! {Registered}
showConfirmationRejected	MRA_Register	WebpageRegister	MRAWR! {showConfirmationRejected}
FailRegistered	WebpageRegister	Person	WR! {FailRegistered}
Registering	MRA_Register	DataBaseInfo	MRAR! {Registering}

showConfirmationAccepted	MRA_Register	WebpageRegister	MRAWR! {showConfirmationAccepted}
SuccessRegistered	WebpageRegister	Person	WR! {successRegistered}
AddMovie	User	WebpageUser	U! {AddMovie}
MakeAddingMovie	WebpageUser	MRA_AddMovie	WU! {MakeAddingMovie}
get_AddedMovies	MRA_AddMovie	DataBaseMovies	DBM! {AddedMovies}
showConfirmationFail	MRA_AddMovie	WebpageUser	MRAAM! {showConfirmationFail}
showAddMovieFailed	WebpageUser	User	WU! {showAddMovieFailed}
AddingMovies	MRA_AddMovie	DataBaseMovies	MRAAM! {AddingMovies}
showConfirmationSuccess	MRA_AddMovie	WebpageUser	MRAAM! {showConfirmationSuccess}
showAddMovieSucceed	WebpageUser	User	WU! {showAddMovieSucceed}
RateMovie	User	WebpageUser	U! {RateMovie}
makeRateMovie	WebpageUser	MRA_Rate	WU! {makeRateMovie}
get_Rated	MRA_Rate	MovieList	ML! {Rated}
showRateFailed	MRA_Rate	WebpageUser	MRAR! {showRateFailed}
confirmationRateFailed	WebpageUser	User	WU! {confirmationRateFailed}
RatingMovie	MRA_Rate	MovieList	MRAR! {RatingMovie}
showSuccessed	MRA_Rate	WebpageUser	MRAR! {showSuccessed}
confirmationRateSucceed	WebpageUser	User	WU! {confirmationRateSucceed}
AccessMoviesList	User	WebpageUser	U! {AccessMoviesList}
makeAccessMoviesList	WebpageUser	MRA_MovieOverviewShow	WU! {makeAccessMoviesList}

get_ availableMoviesList	MRA_MovieOverviewShow	DataBaseMovies	DBM! {availableMovieList}
AvailableMovieList	MRA_MovieOverviewShow	WebpageUser	MRAMOS! {AvailableMovieList}
get_availableRatedMovieList	MRA_MovieOverviewShow	MovieList	ML! {availableRatedMovieList}
AvailableRatedMovieList	MRA_MovieOverviewShow	WebpageUser	MRAMOS! {AvailableRatedMovieIst}
Movies'eRepresentation	WebpageUser	User	WU! {Movies'eRepresentation}

- Lexical domains are not sources of messages

Message in scenario	Source	Domain type
Register	Person	BiddableDomain
registerNewUser	WebpageRegister	CausalDomain
get_Registered	MRA_Register	Machine
showConfirmationRejected	MRA_Register	Machine
FailRegistered	WebpageRegister	CausalDomain
Registering	MRA_Register	Machine
showConfirmationAccepted	MRA_Register	Machine
SuccessRegistered	WebpageRegister	CausalDomain
AddMovie	User	BiddableDomain
MakeAddingMovie	WebpageUser	CausalDomain
get_AddedMovies	MRA_AddMovieInDatabase	Machine
showConfirmationFail	MRA_AddMovieInDatabase	Machine
showAddMovieFailed	WebpageUser	CausalDomain
AddingMovies	MRA_AddMovieInDatabase	Machine
showConfirmationSuccess	MRA_AddMovieInDatabase	Machine

showAddMovieSucceed	WebpageUser	CausalDomain
RateMovie	User	BiddableDomain
makeRateMovie	WebpageUser	CausalDomain
Get_Rated	MRA_MovieRate	Machine
showRateFailed	MRA_MovieRate	Machine
confirmationRateFailed	WebpageUser	CausalDomain
RatingMovie	MRA_MovieRate	Machine
showSuccessed	MRA_MovieRate	Machine
confirmationRateSucceed	WebpageUser	CausalDomain
“AccessMoviesList”	User	BiddableDomain
makeAccessMoviesList	WebpageUser	CausalDomain
get_ availableMoviesList	MRA_MovieOverviewShow	Machine
AvailableMovieList	MRA_MovieOverviewShow	Machine
Get_availableRatedMovieList	MRA_MovieOverviewShow	Machine
AvailableRatedMovieList	MRA_MovieOverviewShow	Machine
Movies’eRepresentation	WebpageUser	CausalDomain

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

Subproblem	Normal case	Exceptional case
PD-UserRegister	SD- UserRegister	
PD-AddMovieInDatabase	SD- AddMovieInDatabase	
PD-MovieRate	SD- MovieRate	
PD-MovieOverviewShow	SD-MovieOverviewShow	

1.4 A4		

1.4.1 Technical Context diagram:

Technical realization of domains from context and problem diagrams:

- DataBaseMovies: Realized as SQL DataBase on the same computer as the machine. Therefore, the database Is connected by a call and return interface and used with SQL commands.
- DataBaseInfo: Realized as SQL DataBase on the same computer as the machine. Therefore, the database Is connected by a call and return interface and used with SQL commands.
- MovieList: Realized as SQL DataBase on the same computer as the machine. Therefore, the database Is connected by a call and return interface and used with SQL commands.
- WebpageUser: Realized using Apache Tomcat and UserWebBrowser (Browser of user)
- WebpageRegister**:** Realized using Apache Tomcat and RegisterWebBrowser (Browser of Registration)

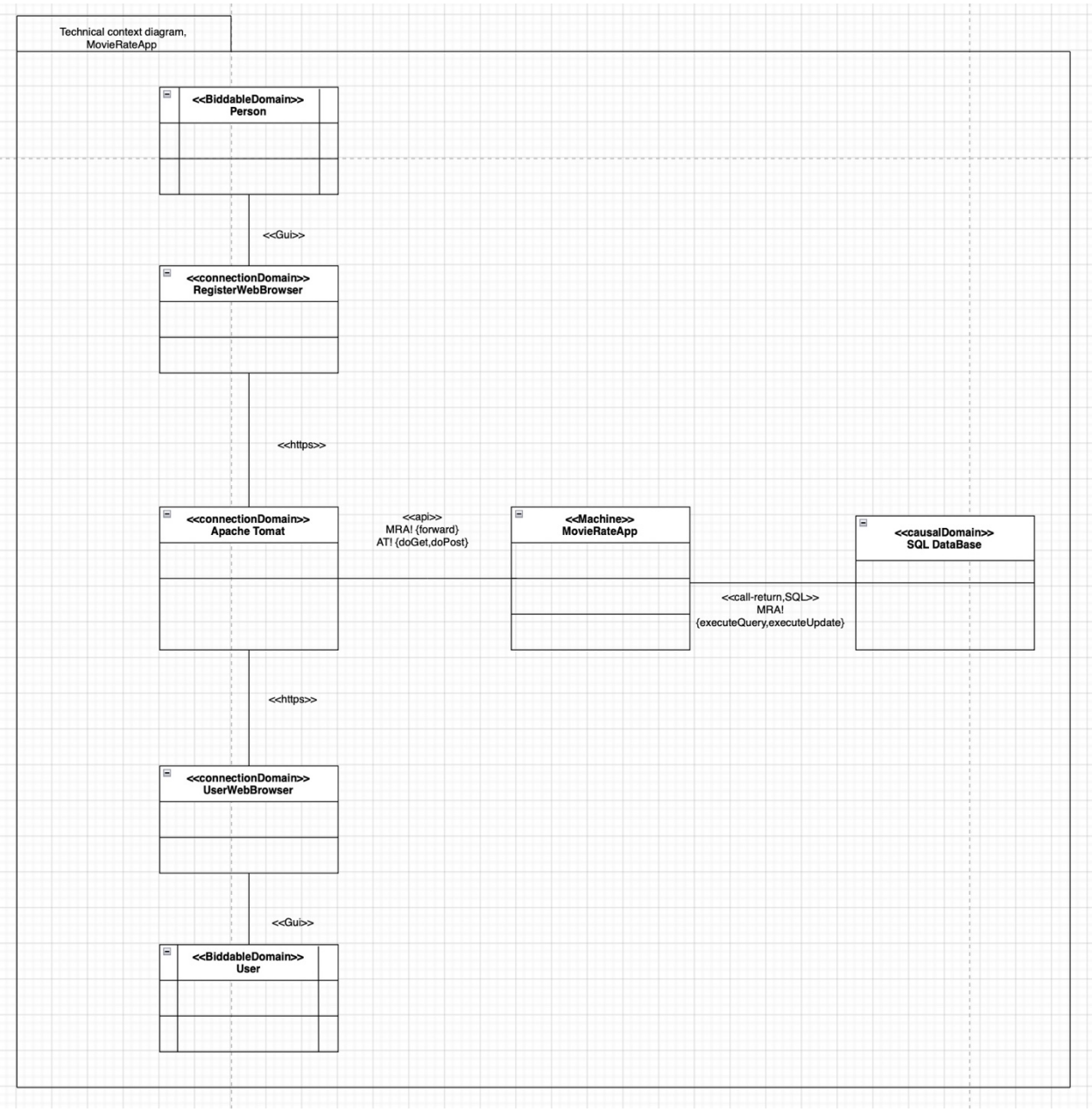


Figure 10 Technical Context diagram

Technical interfaces in the environment:

- HTTP: hypertext transfer Protocol
- GUI: user interfaces presented by RegisterWebBrowser and UserWebBrowser.

- SQL commands: defined in FIPS PUB127-2

Mapping diagram for technical context diagram:

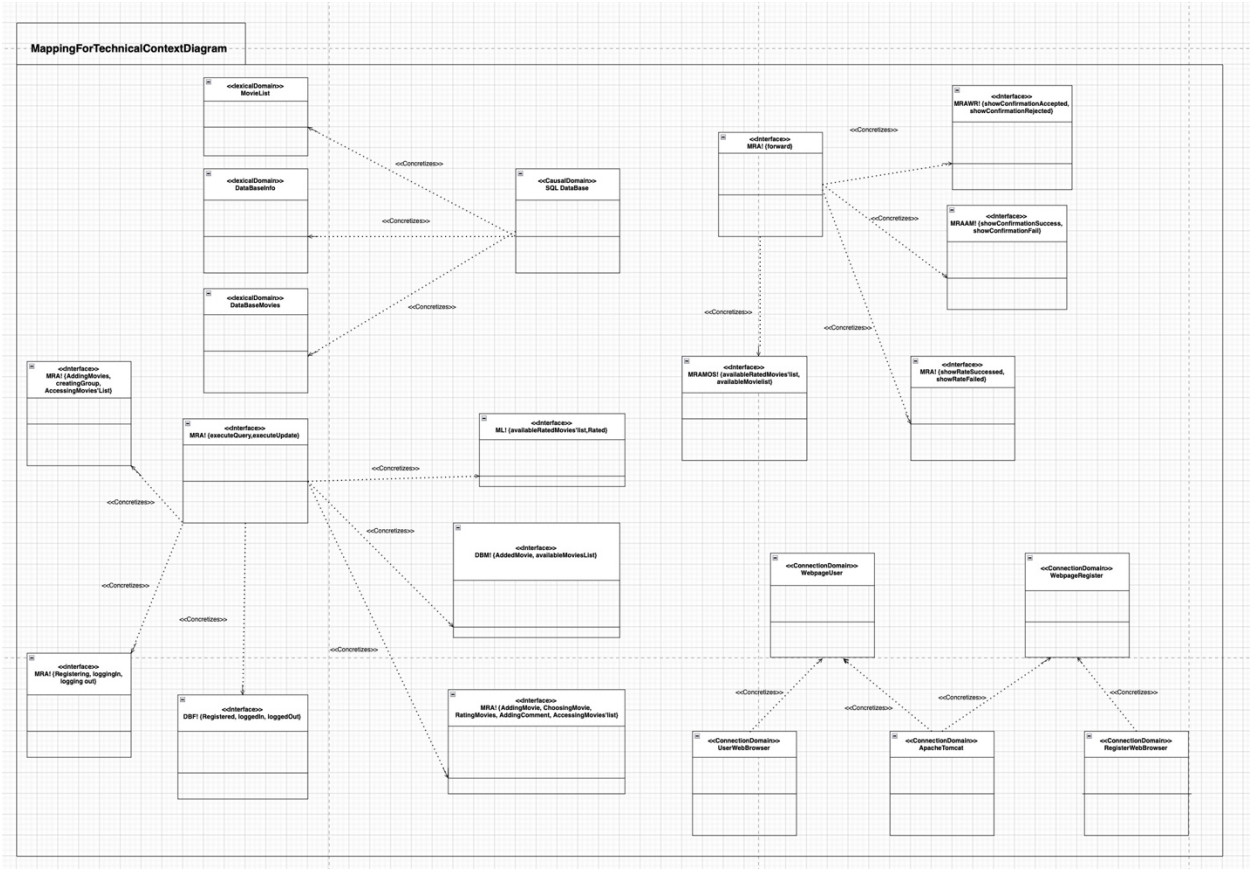


Figure 11 mapping for context diagram

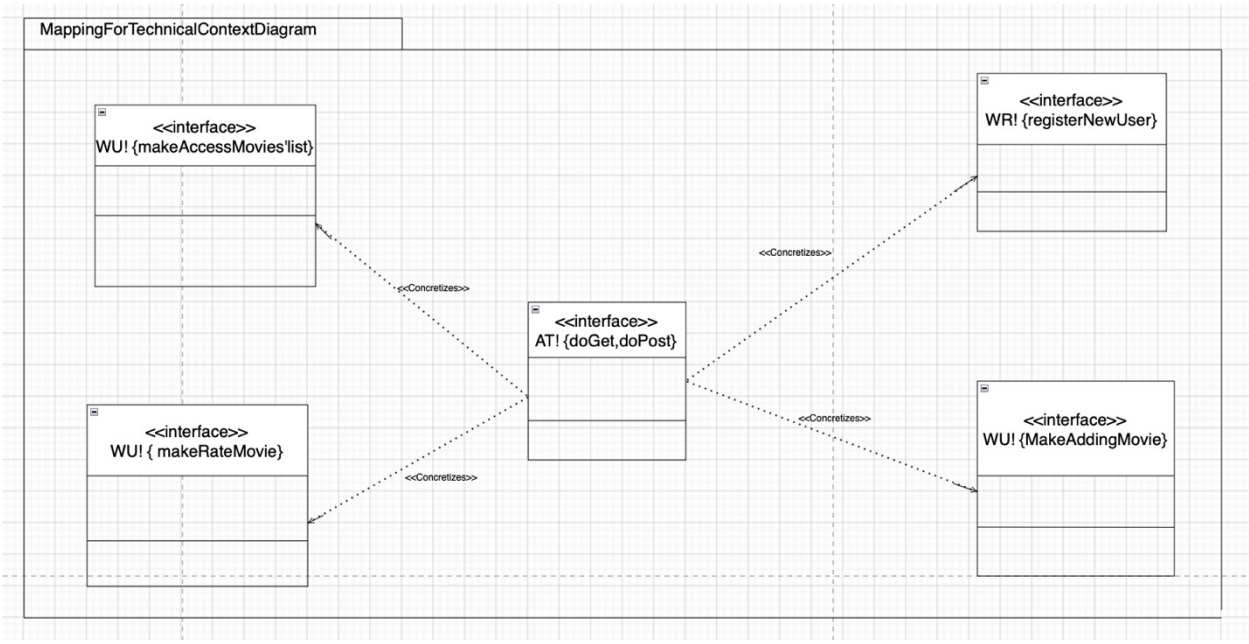


Figure 12 Mapping of context diagram 2

1.4.2 validation for A4:

Validation:	New phenomena and domains are suitable to implement the external messages used in the abstract phenomena
	The technical context diagram must be consistent to the context diagram and problem diagrams: • All domains and phenomena of the technical context diagram are related to elements in the problem diagrams or context diagrams

	• All domains directly connected with the machine in the problem diagrams must be related to an element in the technical context diagram

Message	New phenomenon and domains
Register	Apache Tomcat, http
showConfirmationAccepted	Apache Tomcat, http
showConfirmationRejected	Apache Tomcat, http
AddMovie	Apache Tomcat, http
showConfirmationSuccess	Apache Tomcat, http
showConfirmationFail	Apache Tomcat, http
RateMovie	Apache Tomcat, http
showRateSuccessed	Apache Tomcat, http
showRateFailed	Apache Tomcat, http
AccessMovielist	Apache Tomcat, http
AvailableRatedMovielist-AvailableMovielist	Apache Tomcat, http

Problem diagram	Domain connected with the machine	Element in the TCD
UserRegister	WebpageUser	UserWebBrowser, Apache Tomcat
	DataBaseInfo	SQL DataBase

MovieRate	WebpageUser	UserWebBrowser, Apache Tomcat
	MovieList	SQL MovieList
AddMovieInDatabase	DataBaseMovie	SQL DataBase
	WebpageUser	UserWebBrowser, Apache Tomcat
MovieOverviewShow	WebpageUser	UserWebBrowser, Apache Tomcat
	DataBaseMovies	SQL DataBase
	MovieList	SQL DataBase

1.5 A5		

1.5.1 Class diagrams and OCL Specifications for R1:

- **User Register:**
- **Name**: registerNewUser
- **Description**: register a new user and return confirmation success or fail.
- **OCL constraint**:

Context: MRA_UserRegister:: registerNewUser(name: string, E-mail: string, age: Integer)

Pre: True

Post:

If age >=18 and Info@pre.available(name)=false then

Info->one (User: DatabaseInfo | User.name=name and User.age=age and User.E-mail=E-mail) and

Info->size () =Info@pre->size () +1 and
WebpageRegister^showConfirmationAccepted ()
Else WebpageRegister^showConfirmationRejected ()
Endif

OCL constraint:

Context: MRA_UserRegister
Inv: DatabaseInfo.AllInstances ()-> isUnique(name)

Validation:

- Operation specifications must be consistent with abstract specifications:
  the operation specification of registerNewUser is consistent with the abstract Specification
- The postcondition covers all cases exhibited in the abstract specifications.
  The normal and exceptional case behavior described in the abstract specification are covered in postcondition
- Parameters must be used in pre- and/or postcondition: the parameters are used in the pre and 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 WebpageRegister via his/her Web Browser, which forwards these to this operation.
- All classes, associations and attributes newly introduced in the class model must be motivated by some operation specification: the method Available (name: string) is added to the class DataBaseinfo because we need to know whether the new username is unique or not

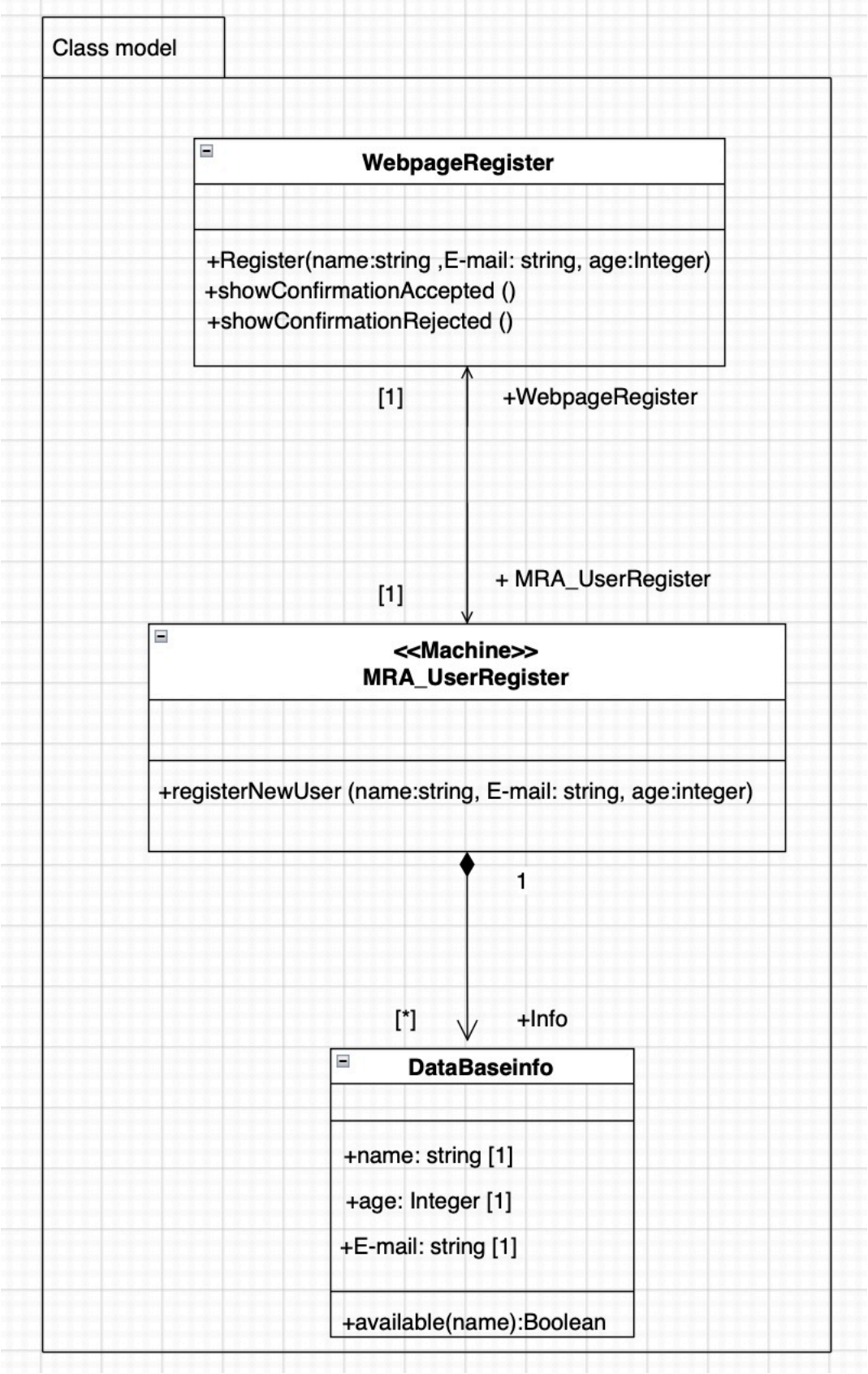

Class model
WebpageRegister
+Register(name:string ,E-mail: string, age:Integer)
+showConfirmationAccepted ()
+showConfirmationRejected ()
[1]
+WebpageRegister
[1]
+ MRA_UserRegister
<<Machine>>
MRA_UserRegister
+registerNewUser (name:string, E-mail: string, age:integer)
1
[*]
+Info
DataBaseinfo
+name: string [1]
+age: Integer [1]
+E-mail: string [1]
+available(name):Boolean

Figure 13 class diagram for R1

1.5.2 class diagram and OCL Specifications for R2:

- **Name:** MakeAddingMovie
- **Description:** add a movie with providing its title, the actors (at least one) and original publishing date and we must consider the movie exist once. So, if we try to add it and it fails it means it is existent already.
- **OCL constraint:**

Context: MRA_AddMovie::MakeAddingMovie(title: MovieData, actors: MovieData, director: MovieData, original Publishing Date: Date)
Pre: True
Post: let
D: DataBaseMovie=DBM->any (M: DataBaseMovie | M. Movietitle=Movietitle) in
If DBM@pre->exists (d: DataBaseMovie | d. MovieTitle = Movietitle)
then WebpageUser^showConfirmationFail ()
else D. DataBaseMovie->one (film: DatabaseMovie |film. MovieID=MovieID) and
D.DBM→ size () = D. DBM @ Pre → size () + 1 and
WebpageUser^showConfirmationSuccess ()
Endif

OCL constraints:
Context: DataBaseMovie
Inv: MakeAddingMovie.AllInstances()->isUnique (Movietitle)

Validation:

- Operation specifications must be consistent with abstract specifications:
  the operation specification of MakeAddingMovie is consistent with the abstract Specification
- The postcondition covers all cases exhibited in the abstract specifications.
  The normal and exceptional case behavior described in the abstract specification are covered in postcondition
- Parameters must be used in pre- and/or postcondition: the parameters are used in the pre and 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 WebpageUser via his/her Web Browser, which forwards these to this operation.
- All classes, associations and attributes newly introduced in the class model must be motivated by some operation specification: the method MakeAddingMovie (with Movietitle: string, with Movieactors: string, with original publishing date: date).

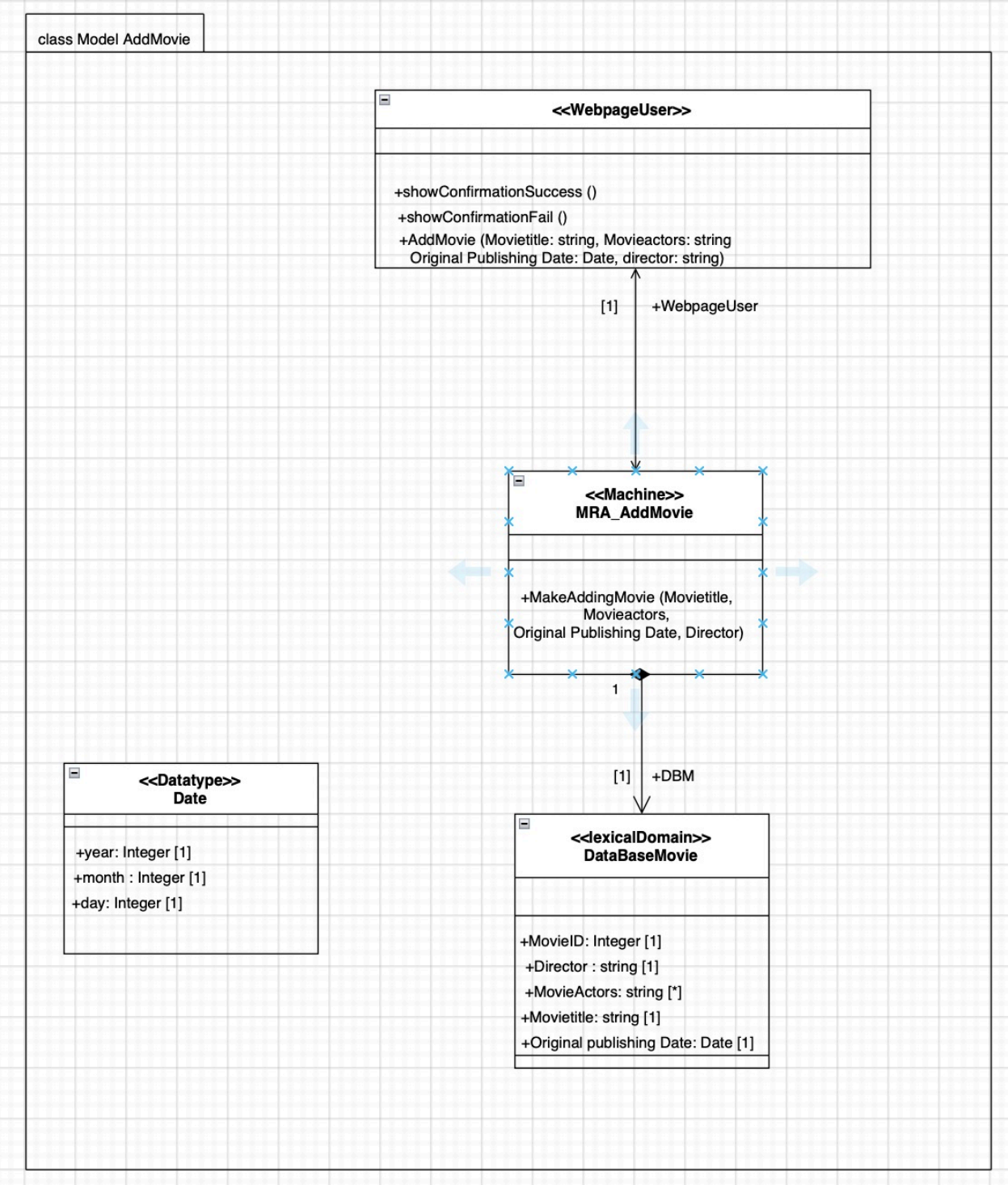


Figure 14 Class diagram for R2

1.5.3 class diagram and OCL Specifications for R3:

- **Name:** makeRateMovie
- **Description:** it is used to rate a movie and the rating is a number in between 1 and 10, where 1 Is very bad and 10 is very good. The movies without rating will be rated with zero and each movie can be rated one time from the user.
- **OCL constraint:**

Context: MRA_Rate::makeRateMovie(MovieID: Integer, Username: string, Rate: Integer)
Pre: True
Post: let L: MovieList| M: MovieID in
If
M: MovieID=ID and
M@pre.UserfirstRate (Username)= true
Then
L->one (M: MovieID | M. Rate= Rate) and
WebpageUser^showRateSuccessed ()
Else WebpageUser^showRateFailed ()
Endif
OCL constraint:
Context: MovieList
Inv: MovieList.AllInstances ()->isUnique (MovieID and Username)

Remarks: UserfirstRate will be number between 1 and 10. It can equal one and can equal 10 and will return true only if it's the first user rate for specific movie

Validation:

- Operation specifications must be consistent with abstract specifications:
  the operation specification of RateMovie is consistent with the abstract Specification
- The postcondition covers all cases exhibited in the abstract specifications.
  The normal and exceptional case behavior described in the abstract specification are covered in postcondition
- Parameters must be used in pre- and/or postcondition: the parameters are used in the pre and 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 WebpageUser via his/her Web Browser, which forwards these to this operation.

- All classes, associations and attributes newly introduced in the class model must be motivated by some operation specification: the method MakeRateMovie (Rate: Integer).

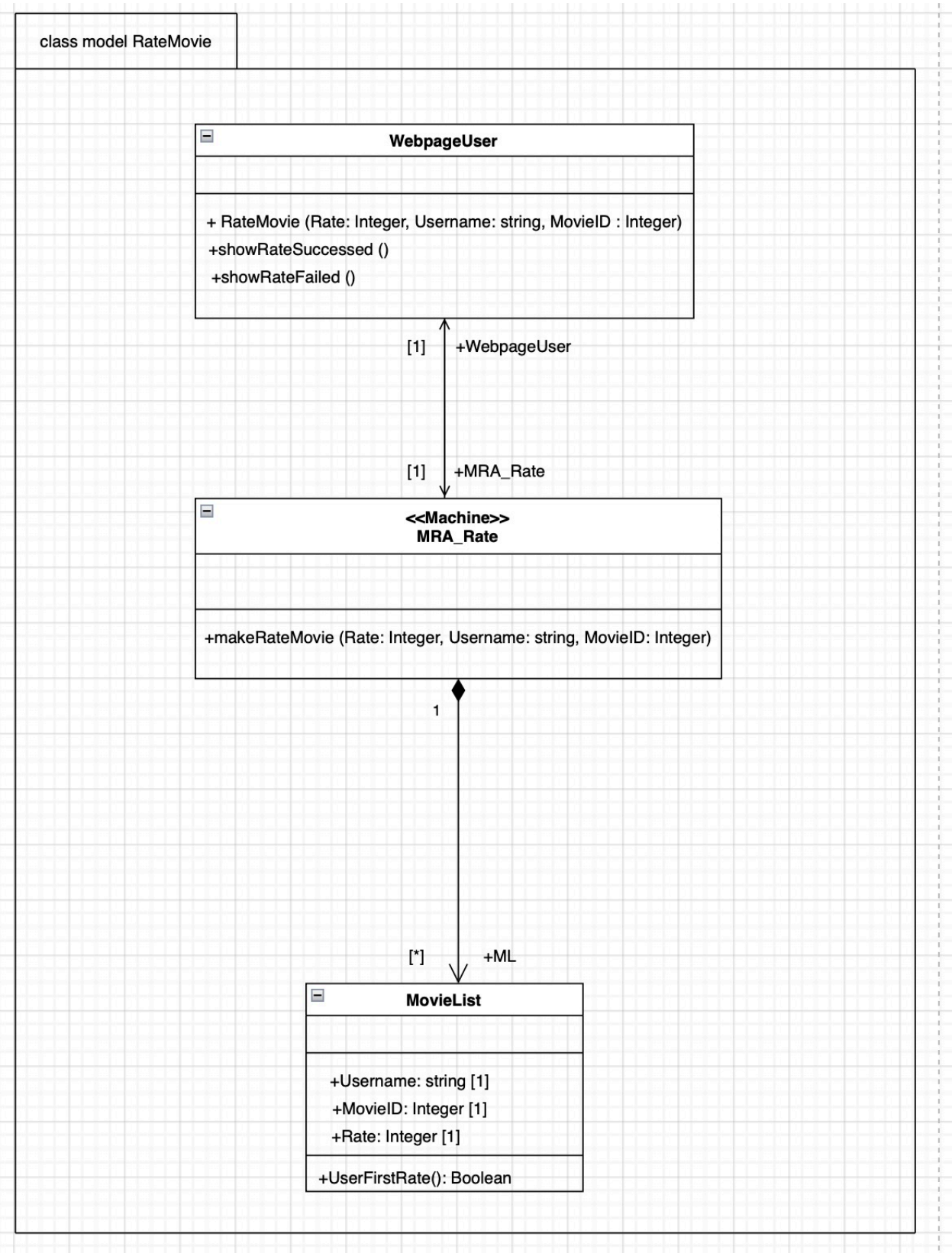


Figure 15 class diagram for R3

1.5.4 Class diagram and OCL Specifications for R7:

- **Name:** makeAccessMovieList
- **Description:** it is used to able the user to make overview of films and their ratings and return the movies list
- **OCL constraint:**

Context: MRA_MovieOverviewShow:: makeAccessMovieList ()
Pre: True
Post: Let
Let DBM: moviedatabase= MovieDataBase in
Let MovieList: Set (MovieData)= DBM. MovieData.allInstances()->asSet () in
WebpageUser ^ AvailableRatedMovielist_AvailableMovieList (MovieList)

Validation:

- Operation specifications must be consistent with abstract specifications:
  the operation specification of MovieOverviewShow is consistent with the abstract Specification
- The postcondition covers all cases exhibited in the abstract specifications.
  The normal and exceptional case behavior described in the abstract specification are covered in postcondition
- Parameters must be used in pre- and/or postcondition: the parameters are used in the pre and 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 WebpageUser via his/her Web Browser, which forwards these to this operation.
- All classes, associations and attributes newly introduced in the class model must be motivated by some operation specification: the method makeAccessMovieList ().

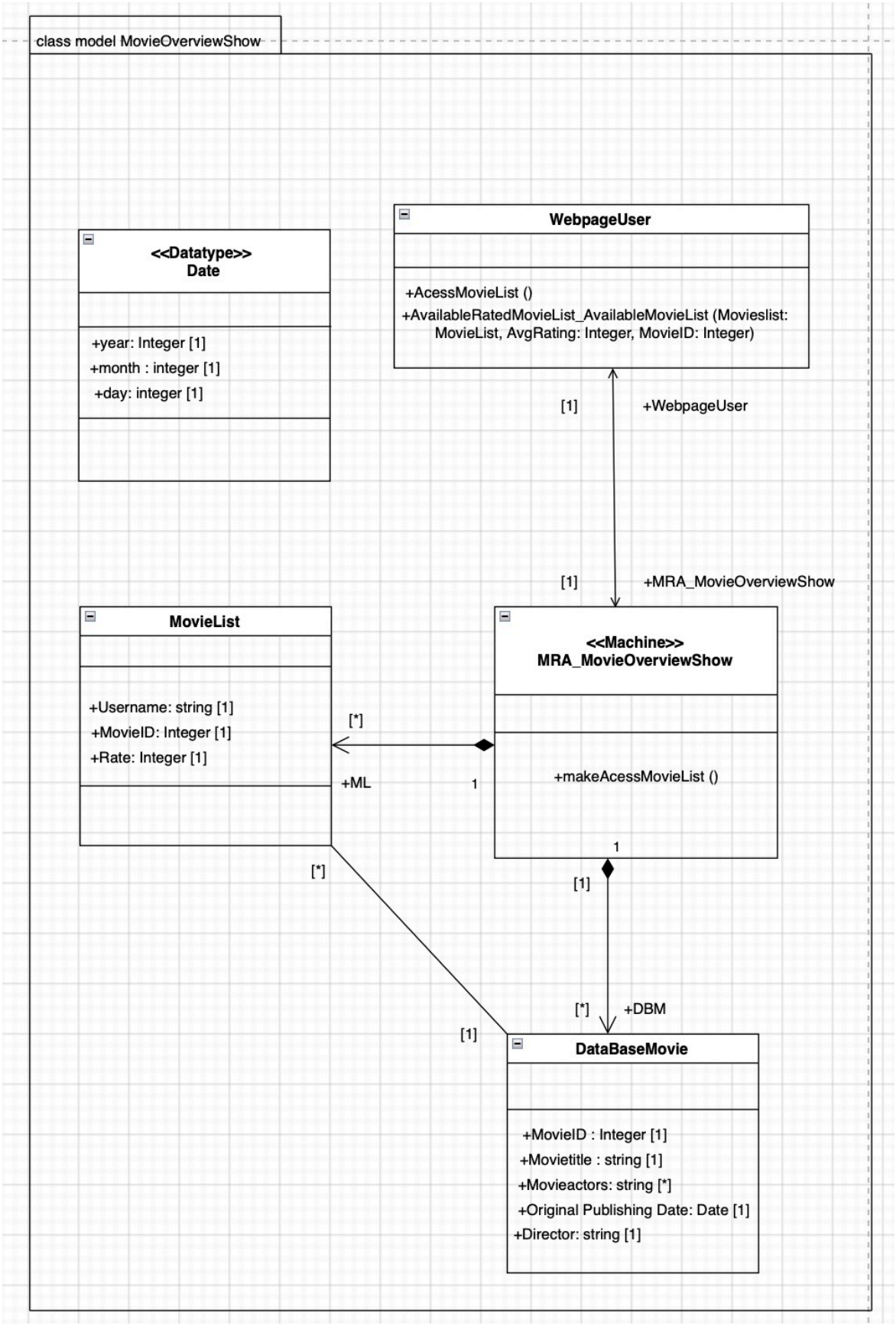

class model MovieOverviewShow
<<Datatype>>
Date
+year: Integer [1]
+month : integer [1]
+day: integer [1]
WebpageUser
+AcessMovieList ()
+AvailableRatedMovieList_AvailableMovieList (Movieslist: MovieList, AvgRating: Integer, MovieID: Integer)
[1]
+WebpageUser
[1]
+MRA_MovieOverviewShow
MovieList
+Username: string [1]
+MovieID: Integer [1]
+Rate: Integer [1]
[*]
+ML
1
<<Machine>>
MRA_MovieOverviewShow
+makeAcessMovieList ()
1
[*]
[1]
[*]
+DBM
[1]
DataBaseMovie
+MovieID : Integer [1]
+Movietitle : string [1]
+Movieactors: string [*]
+Original Publishing Date: Date [1]
+Director: string [1]

Figure 16 class diagram for R7

1.6 A6		

Movie Rating App life cycle:

LC_{Person}=[UserRegister]

Any person has an option [] to register him/herself.

LC_{user}= ((MovieOverviewShow)+; AddMovieinDatabase| MovieOverviewShow |MovieRate) *

The User can AddMovieinDatabase or to make MovieOverviewShow at least once to be able to Rate a Movie and this all can take place 0 times or more than one time.

$LC_{MovieRateApp}$= $(||_{i=1}^{n} LC_{Person\ i})$ || $(||_{i=1}^{n} LC_{User\ i})$
Where || denotes the parallel composition of n copies of Life cycle LC.

Validation:

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

Scenario	Life-cycle expression
Sd UserRegister	LC_{Person}
Sd AddMovieinDatabase	LC_{User}
Sd MovieRate	LC_{User}
Sd MovieOverviewShow	LC_{User}

For the biddable Domain **Person** exactly one Lifecycle exists, namely LC_{Person}
For the biddable Domain **User** exactly one Lifecycle exists, namely LC_{User}

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

- UserRegister has no state predicates at the beginning and end. Hence, it can be executed an arbitrary number of times.
- AddMovieinDatabase has no state predicates at the beginning. Hence, it can be executed an arbitrary number of times but at the end there is state Predicate "movie is added" in case of successful adding a new Movie or " movie is exist" in case of the existence of movie.
- MovieRate has no state predicates at the beginning and end. Hence, it can be executed an arbitrary number of times.
- MovieOverviewShow can be executed if the MovieList is existent

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

- The sequence diagram UserRegister contains the operation registerNewUser. It has no precondition. Hence, it can be executed at any position of the life-cycle.
- The sequence diagram AddMovieinDatabase contains the operation MakeAddingMovie.it has no precondition. Hence, it can be executed at any position of the life-cycle.
- The sequence diagram MovieRate contains the operation MakeRateMovie. MakeRateMovie requires that there exist already a MovieList, this is ensured by the postcondition "makeAccessMoviesList", that returns the "availableMoviesList". Hence, MovieOverviewShow must be executed before MovieRate
- The sequence diagram MovieOverviewShow contains the operation MakeAccessMoviesList. MakeAccessMoviesList requires that a list exists already.

Exactly one life-cycle exists for the machine domain, that combines all life-cycles
The life-cycle $LC_{MovieRateApp}$ exists for the machine domain. It combines all life-cycles.

2 Design	2.1 D1	

2.1.1 Architectural pattern for R1:

- MRA_UserRegister fits to update 2. instantiated architectural pattern for MRA_UserRegister

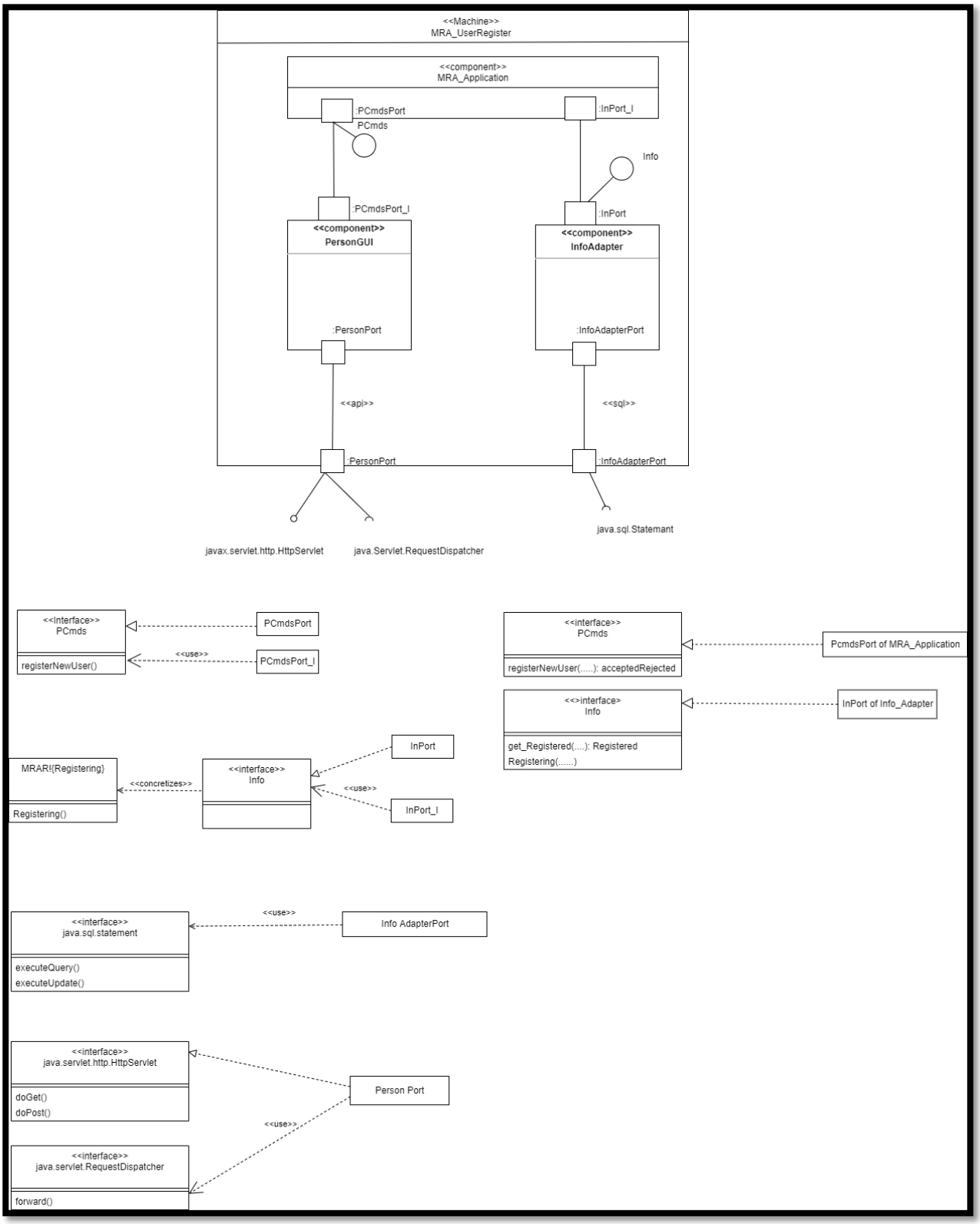


Figure 17 architectural pattern for R1

2.1.2 Architectural pattern for R2:

- MRA_AddMovieinDatabase fits to update 2. instantiated architectural pattern for MRA_AddMovieinDatabase

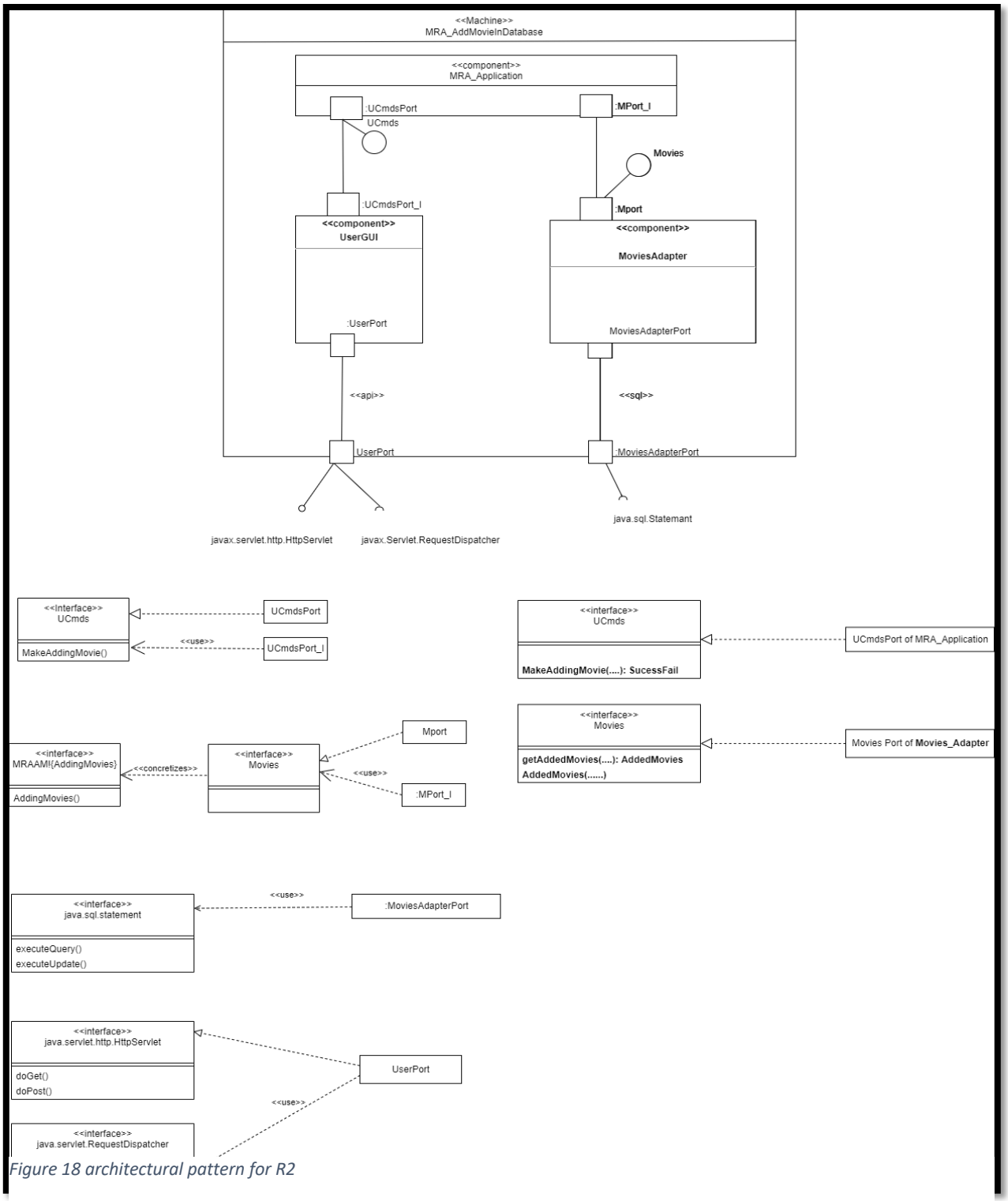


Figure 18 architectural pattern for R2

2.1.3 Architectural pattern for R3:

- MRA_MovieRate fits to update 2. instantiated architectural pattern for MRA_MovieRate

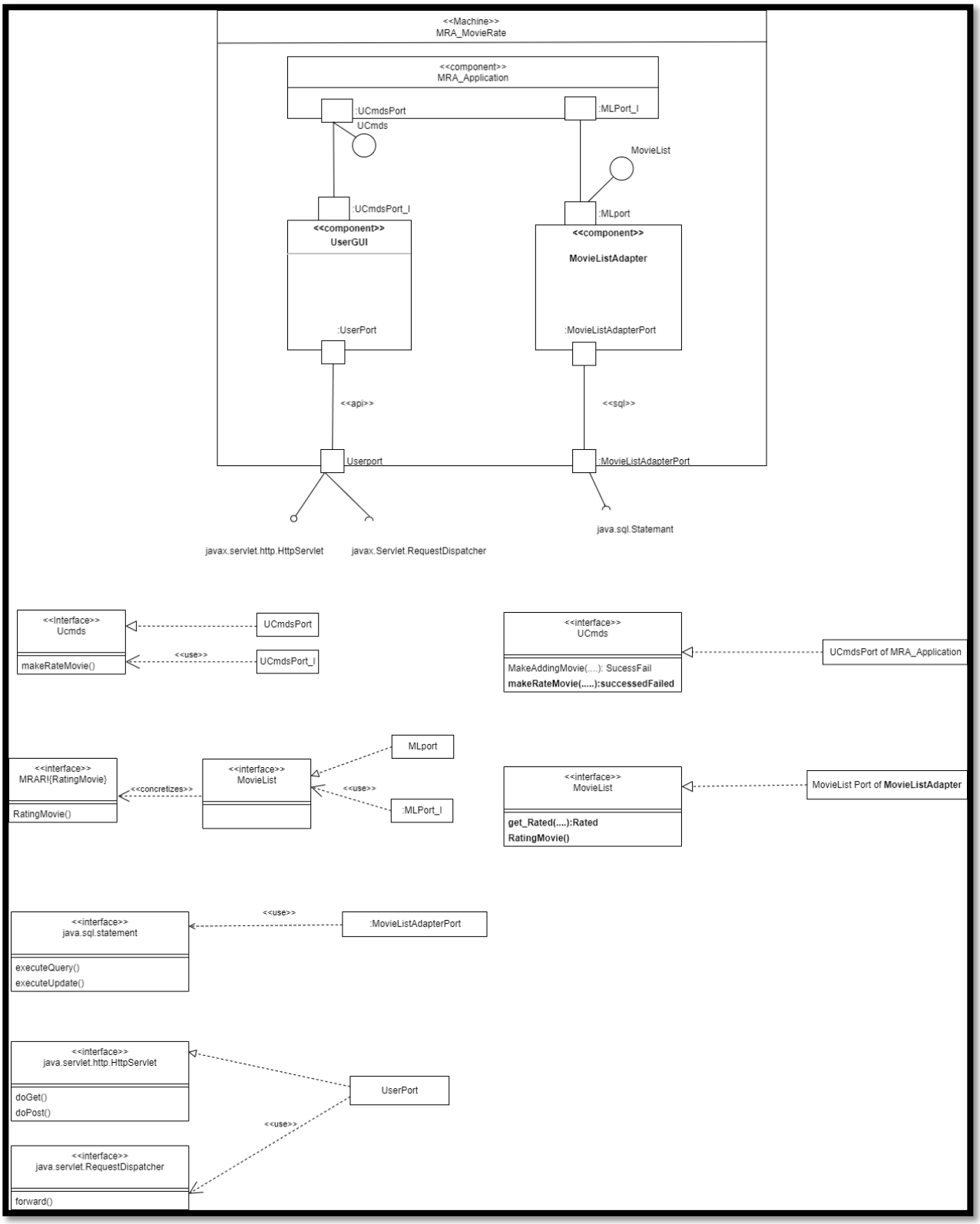


Figure 19 architectural pattern for R3

2.1.4 Architectural pattern for R7:

- MRA_MovieOverviewShow fits to Query 2. instantiated architectural pattern for MRA_MovieOverviewShow

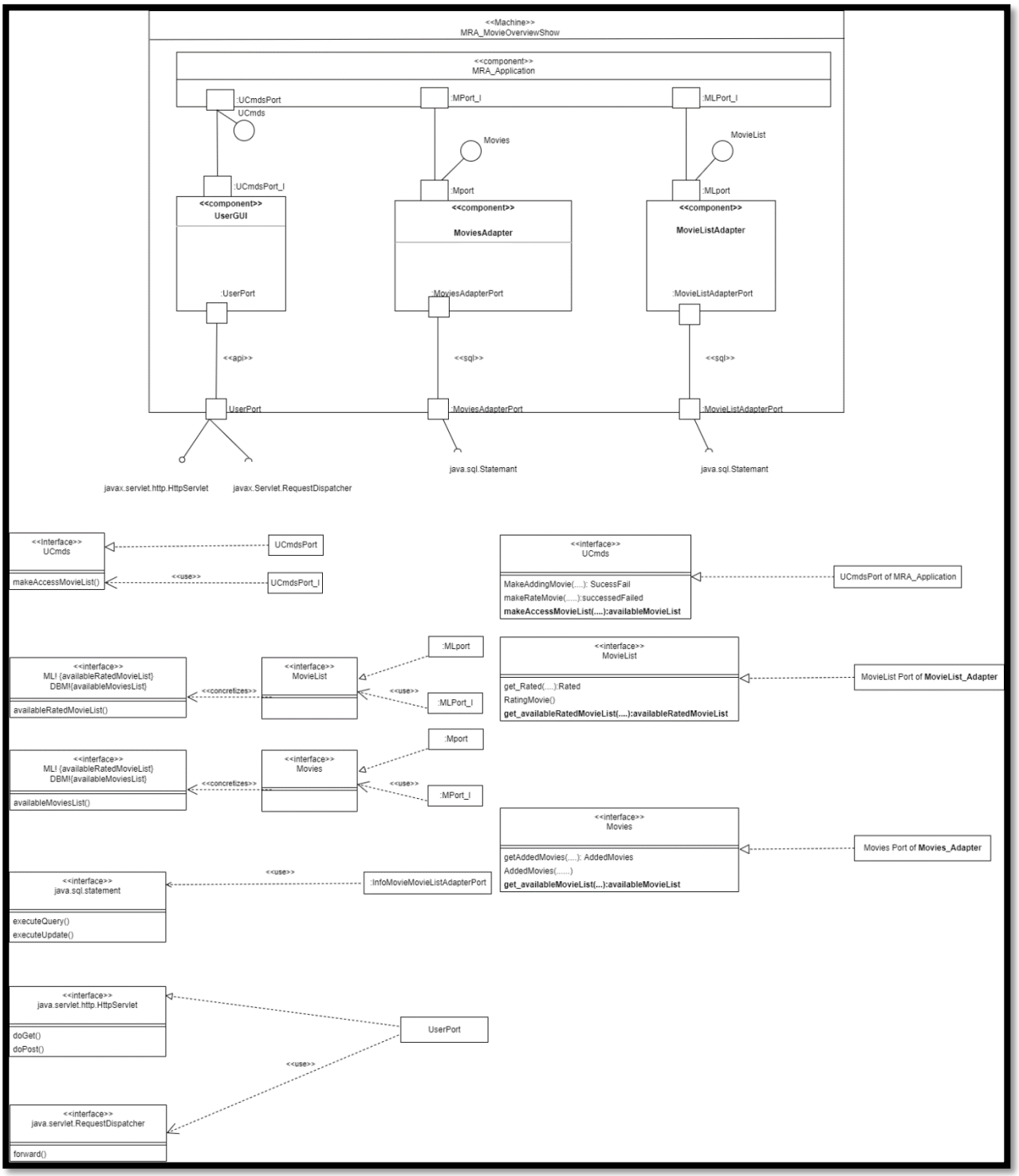
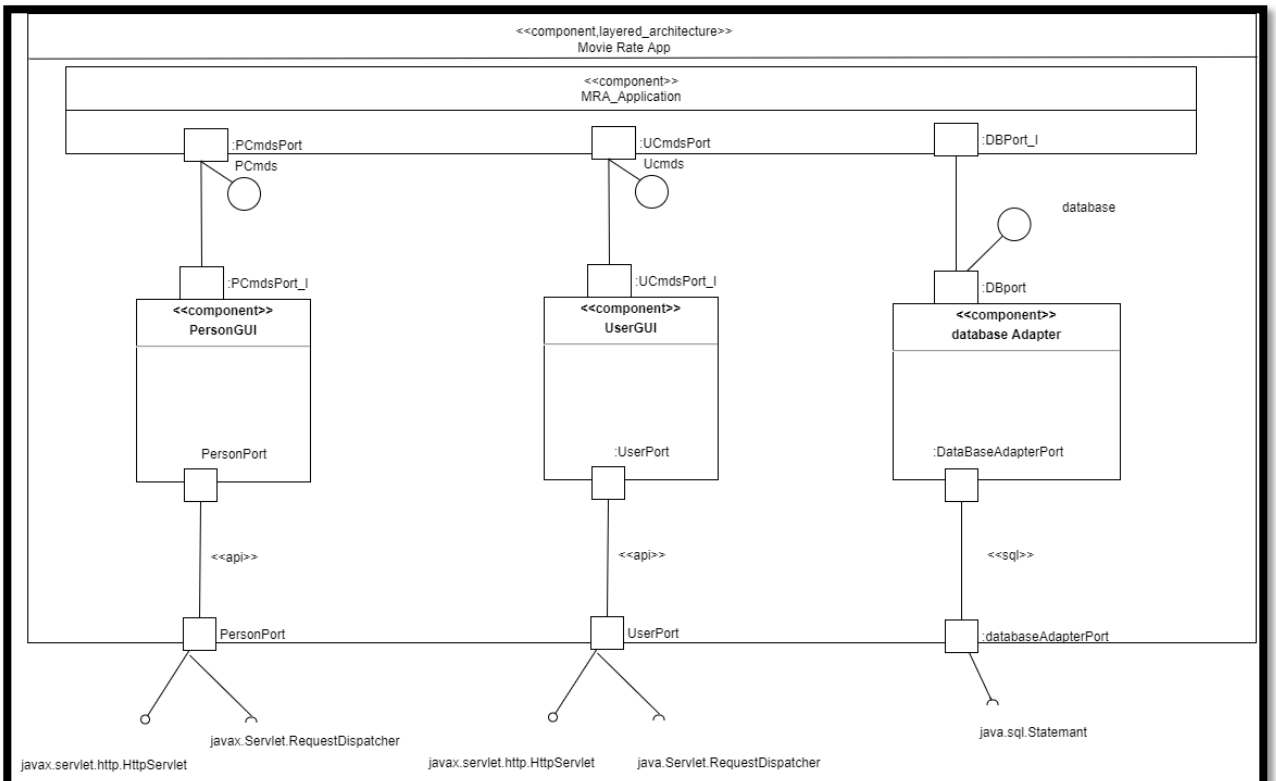
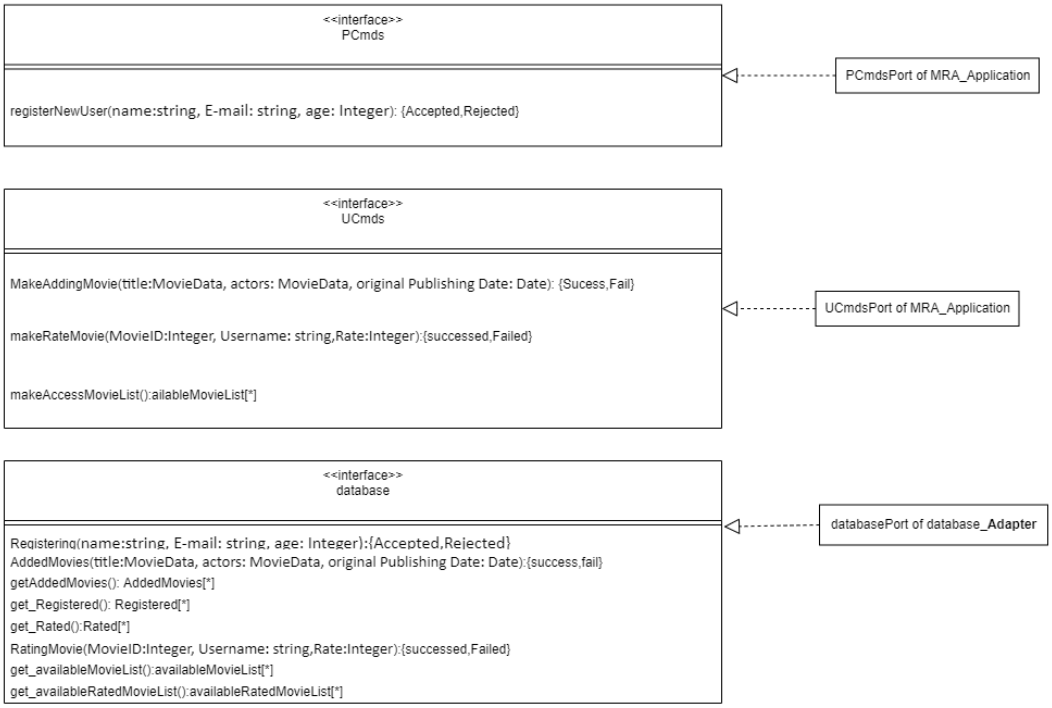


Figure 20 architectural pattern for R7

2.1.5 Refining app_if interfaces class & 2.1.8 Merging Subproblem Architectures:

2.1.6 Refining adapter_if" Interfaces classes: There are no adapter_if interface classes





Remark: All the adapters have been merged in just one Adapter "database Adapter" ,
because all these adapters establishing the connection to the same hardware device.

Figure 21 Merged architecture and Refining app_if

2.1.7 Refining tech_if "interface classes:

Considered interface in subproblem architecture	Technical interfaces
<<api>> javax.servlet.httpServlet in UserRegister Maschine	<<api>> AT!doGet,doPost
<<api>> javax.servlet.RequestDispatcher in UserRegister machine	<<api>> MRA!forward
<<api>> org.apache.commons.mail.Email in UserRegister machine	<<api>> MRA!send
<<sql>> java.sql.Statement in UserRegister machine	<<callreturn,sql>> MRA!executequery,executeUpdate
<<api>>javax.servlet.http.Httpservlet in AddMovieinDatabase machine	<<api>>AT!doGet,doPost
<<api>> javax.servlet.RequestDispatcher in AddMovieinDatabase machine	<<api>> MRA!forward
<<api>> org.apache.commons.mail.Email in AddMovieinDatabase machine	<<api>> MRA!send
<<sql>> java.sql.Statement in AddMovieinDatabase machine	<<callreturn,sql>> MRA!executequery,executeUpdate
<<api>>javax.servlet.http.Httpservlet in RateMovie machine	<<api>>AT!doGet,doPost
<<api>> javax.servlet.RequestDispatcher in RateMovie machine	<<api>> MRA!forward
<<api>> org.apache.commons.mail.Email in RateMovie machine	<<api>> MRA!send
<<sql>> java.sql.Statement in RateMovie machine	<<callreturn,sql>> MRA!executequery,executeUpdate
<<api>>javax.servlet.http.Httpservlet in MovieOverviewShow machine	<<api>>AT!doGet,doPost
<<api>> javax.servlet.RequestDispatcher in MovieOverviewShow machine	<<api>> MRA!forward
<<api>> org.apache.commons.mail.Email in MovieOverviewShow machine	<<api>> MRA!send

<<sql>> java.sql.Statement in MovieOverviewShow machine	<<callreturn,sql>> MRA!executequery,executeUpdate

2.1.9 Validation for 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
UserRegister	registerNewUser	in	PCmds::registerNewUser	Provided
	get_Registered	out	Info::get_Registered	required
	Registered	in	Return value of Info::get_Registered	required
	Registering	out	Info::Registering	required
	showConfirmationAccepted	out	Return value of PCmds::regiserNewUser	Provided
	showConfirmationRejected	out	Return value of PCmds::regiserNewUser	Provided
AddMovieIn DataBase	MakeAddingMovie	in	UCmds::MakeAddingMovie	provided
	get_AddedMovies	out	Movies::get_AddedMovies	required
	AddedMovies	in	Return value of Movies::get_AddedMovies	required
	AddingMovies	out	Movies:: AddingMovies	required
	ShowConfirmationSuccess	out	Return value of UCmds::MakeAddingMovie	provided

	ShowConfirmationFail	out	Return value of UCmds::MakeAddingMovie	provided
RateMovie	makeRateMovie	in	UCmds:: makeRateMovie	provided
	get_Rated	Out	MovieList::get_Rated	required
	Rated	In	Return value of MovieList:: get_Rated	required
	RatingMovie	Out	MovieList:: RatingMovie	required
	showRateSuccessed	Out	Return value of UCmds:: makeRateMovie	provided
	showRateFailed	Out	Return value of UCmds:: makeRateMovie	provided
MovieOverviewShow	makeAccessMoviesList	in	UCmds:: makeAccessMoviesList	provided
	get_availableMovieslist	Out	Movies:: get_availableMovieslist	required
	availableMovieslist	In	Return value of Movies:: get_availableMovieslist	required
	get_availableRatedMovieslist	Out	Info_MovieMovieList:: get_availableRatedMovieslist	required
	availableRatedMovieslist	In	Return value of Movie:: get_availableRatedMovieslist	required
	availableRatedMovieslist_ availableMovieslist	Out	Return value of UCmds:: makeAccessMoviesList	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	PCmds/UCmds

provided by machine	required by adapter / provided by app
javax.servlet.RequestDispatcher	return values in PCmds / UCmds
java.sql.Statement	Info_MovieMovieList

- 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.RequestDispatcher	required	VR!{forward}
PersonPort	javax.servlet.http.HttpServlet	provided	AT!{doGet, doPost}
	javax.servlet.RequestDispatcher	required	VR!{forward}
Info_MovieMovieList Port	java.sql.Statement	required	VR!{executeQuery, executeUpdate}

2.2 D2		

2.2.1 Sequence diagram and SQL for R1:

- **Inter-component interaction: UserRegister**

  In our example we will use SQL as Data base system

  The relational data base system requires a server software, i.e., MySQL server

- **User Register:**
- **Name**: registerNewUser
- **Description**: register a new user and return confirmation success or fail.
- **OCL constraint**:

Context: MRA_UserRegister:: registerNewUser(name: string, E-mail: string, age: Integer)

Pre: True

Post:

If age >=18 and Info@pre.available(name)=false then

Info->one (User: DatabaseInfo | User.name=name and User.age=age and User.E-mail=E-mail) and

Info->size () =Info@pre->size () +1 and

WebpageRegister^showConfirmationAccepted ()

Figure 22 OCL for R1

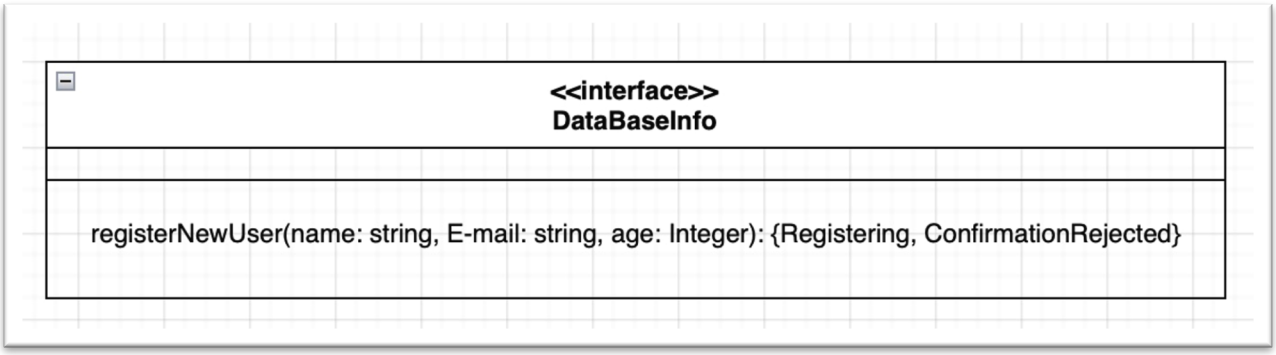


Figure 23 function for R1

The interface behavior between all components will be shown on the next two diagrams:

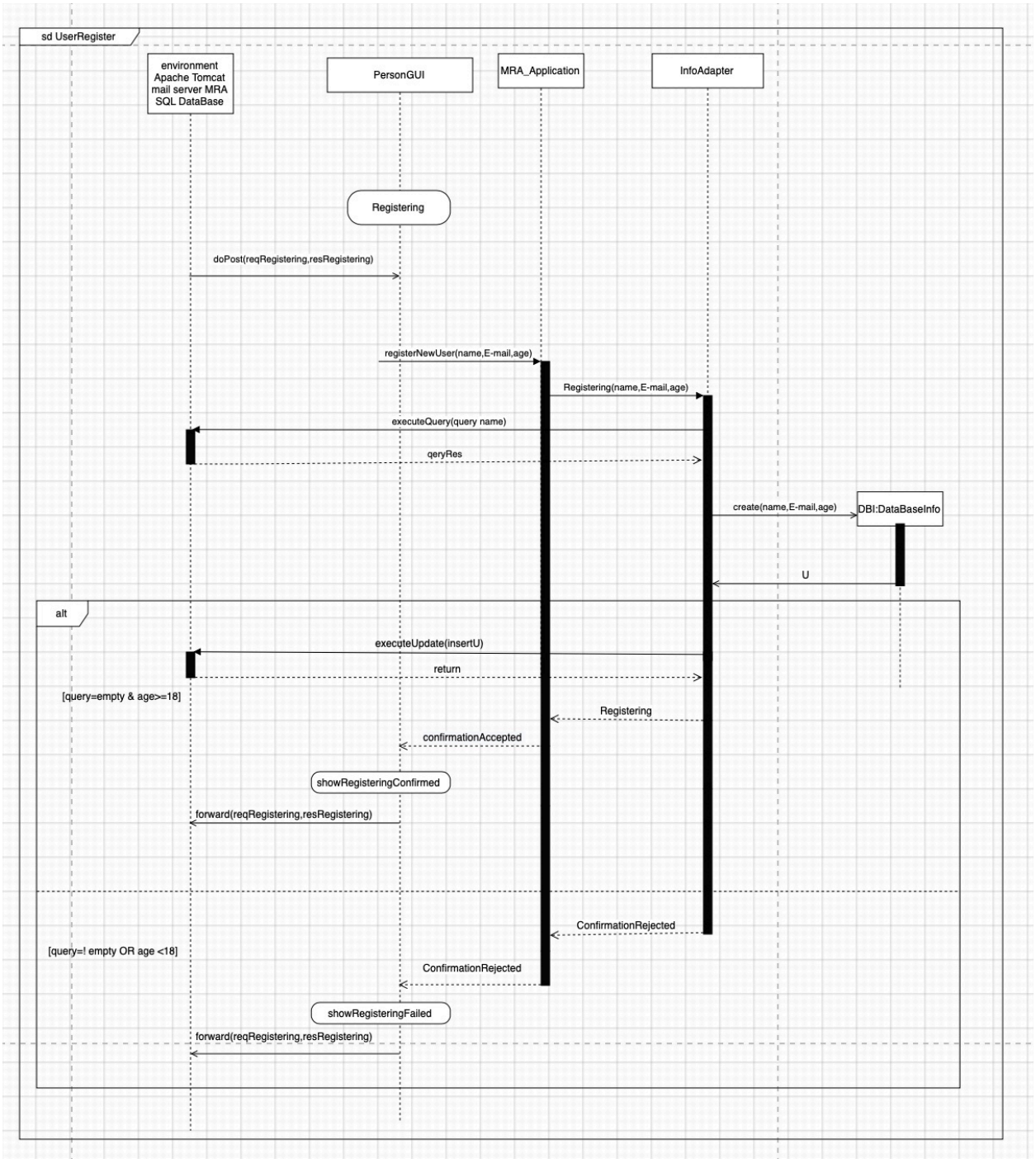


Figure 24 sequence diagram for R1

Inter-component interaction UserRegister- Remarks

1. reqSelect and reqRegistering represents HTTPServletRequest objects containing the required user input
2. reselect and resRegistering represent HTTPServletResponse objects as the counterpart for the request
3. The state Predicate Registering represents that the input form for Registering the selected offer is shown
4. The state Predicate ShowRegisteringConfirmed represents that the confirmation is shown
5. The state Predicate of ShowRegisteringFailed represents that an Error message is shown
6. Forward (….) sends the request and the response back to the server to generate HTML webpage
7. Since we use a MySQL database, we don't need to specify the interfaces to lexical domains in more detail. We use standardized SQL statements to access the database

Queryname:

SELECT * FROM DataBaseinfo DBI WHERE (DBI.name="name")

insertU:

INSERT INTO DataBaseinfo (name, age, E-mail) VALUES ("name"," age"," E-mail")

2.2.2 Sequence diagram and SQL for R3:

- **Inter-component interaction: MovieRate**

- **Name:** makeRateMovie
- **Description:** it is used to rate a movie and the rating is a number in between 1 and 10, where 1 Is very bad and 10 is very good. The movies without rating will be rated with zero and each movie can be rated one time from the user.
- **OCL constraint:**

Context: MRA_Rate::makeRateMovie(MovieID: Integer, Username: string, Rate: Integer)
Pre: True
Post: let L: MovieList| M: MovieID in
If
M: MovieID=ID and
M@pre.UserfirstRate (Username)= true
Then
L->one (M: MovieID | M. Rate= Rate) and
WebpageUser^showRateSuccessed ()
Else WebpageUser^showRateFailed ()
Endif
OCL constraint:
Context: MovieList
Inv: MovieList.AllInstances ()->isUnique (MovieID and Username)

Figure 25 OCL for R3

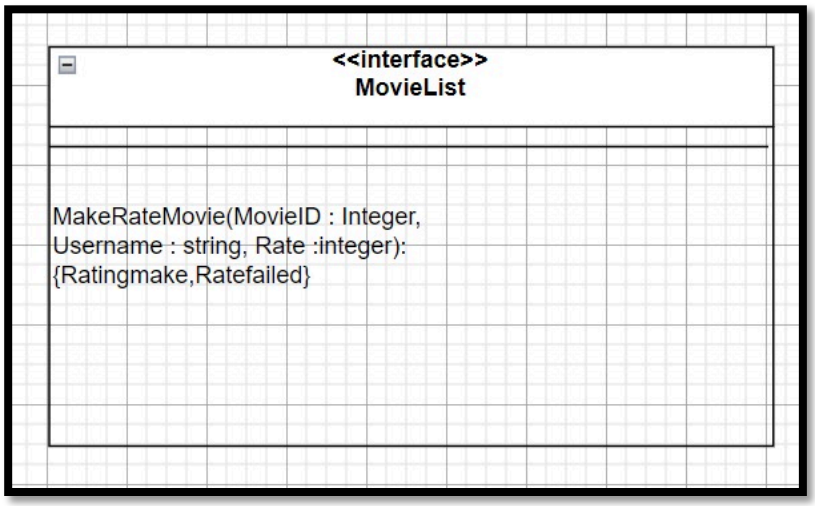


Figure 26 function for R3

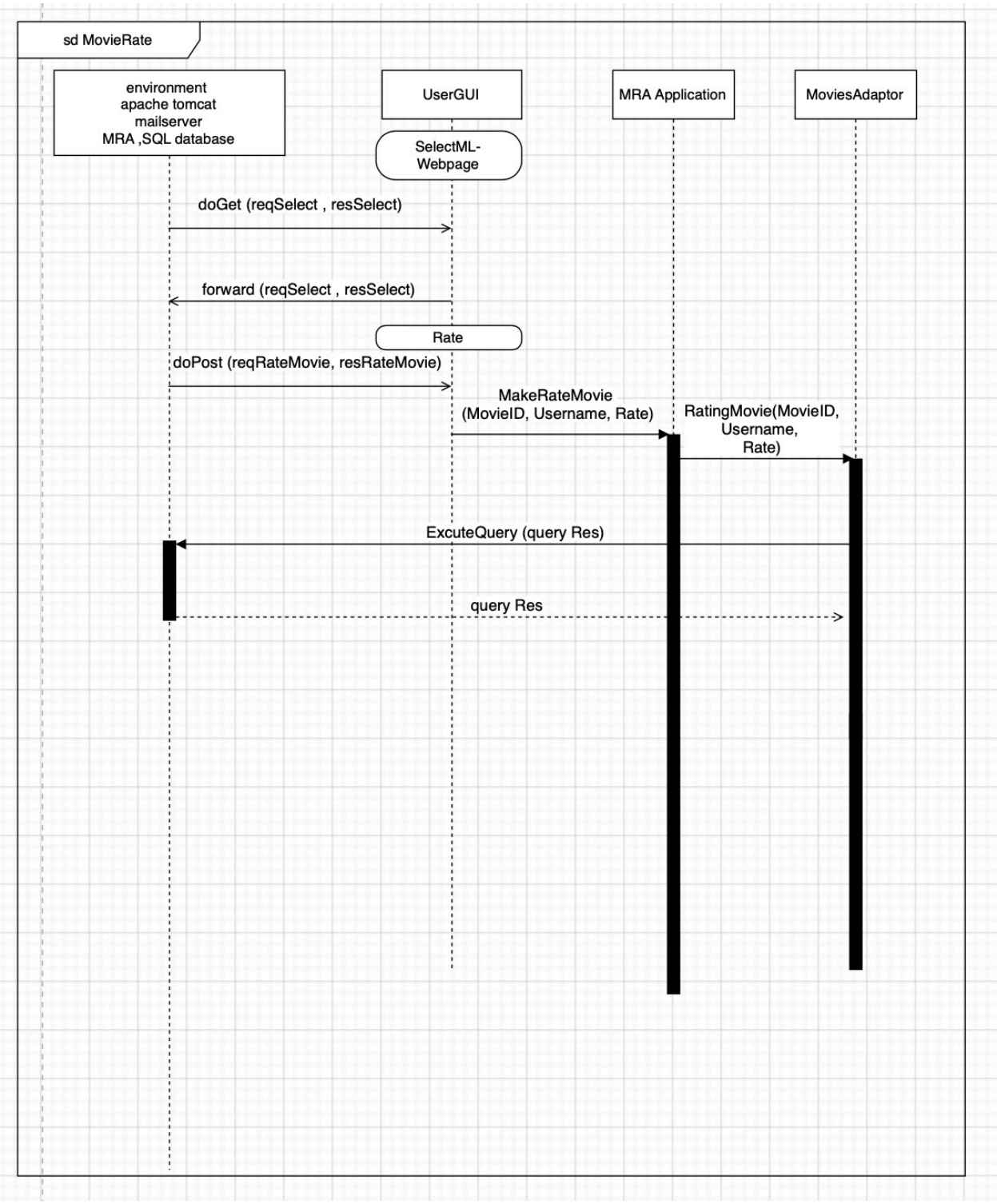


Figure 27 sequence diagram for R3

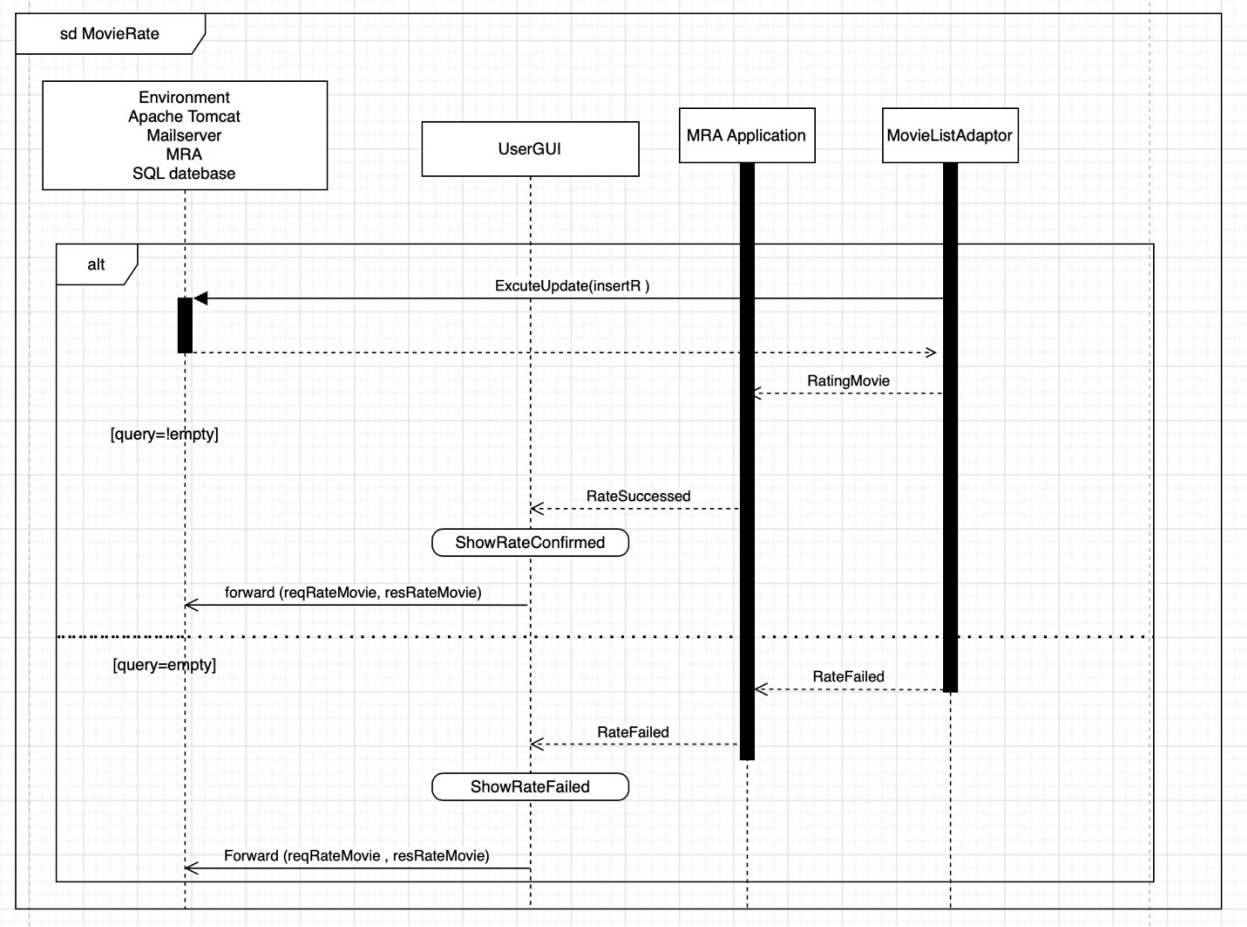


Figure 28 sequence diagram for R3'

Inter-component interactions MovieRate-Remarks

1. reqSelect and reqRateMovie represent HTTPServletRequest objects containing the required user input
2. resSelect and resRateMovie represent HTTPServletResponse objects as the Counterpart for the request
3. The state predicate Rate represents that the input form for Rating a Movie the selected offer is shown
4. The state predicate Show Rate Confirmed represents that the confirmation is shown
5. The state predicate Show Rate Failed represents that an Error message is shown
6. Forward (….) sends the request and response back to the server to generate the HTML webpage.
7. 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.

Query Res:

SELECT * FROM MovieList ML WHERE (ML. MovieID = “MovieID” AND ML. Username=” Username” AND ML. Rate = 0)

insertR:

UPDATE MOVIELIST SET Rate = “new Value” WHERE (MovieID=” MovieID”)

2.2.3 Sequence diagram and SQL for R2:

- **Inter-components interaction: AddMovieinDataBase**

The operation add movie:

- **Name:** MakeAddingMovie
- **Description:** add a movie with providing its title, the actors (at least one) and original publishing date and we must consider the movie exist once. So, if we try to add it and it fails it means it is existent already.
- **OCL constraint:**

Context: MRA_AddMovie::MakeAddingMovie(title: MovieData, actors: MovieData, director: MovieData, original Publishing Date: Date)
Pre: True
Post: let
D: DataBaseMovie=DBM->any (M: DataBaseMovie | M. Movietitle=Movietitle) in
If DBM@pre->exists (d: DataBaseMovie | d. MovieTitle = Movietitle)
then WebpageUser^showConfirmationFail ()
else D. DataBaseMovie->one (film: DatabaseMovie |film. MovieID=MovieID) and
D.DBM→ size () = D. DBM @ Pre → size () + 1 and
WebpageUser^showConfirmationSuccess ()
Endif

Figure 29 OCL for R2

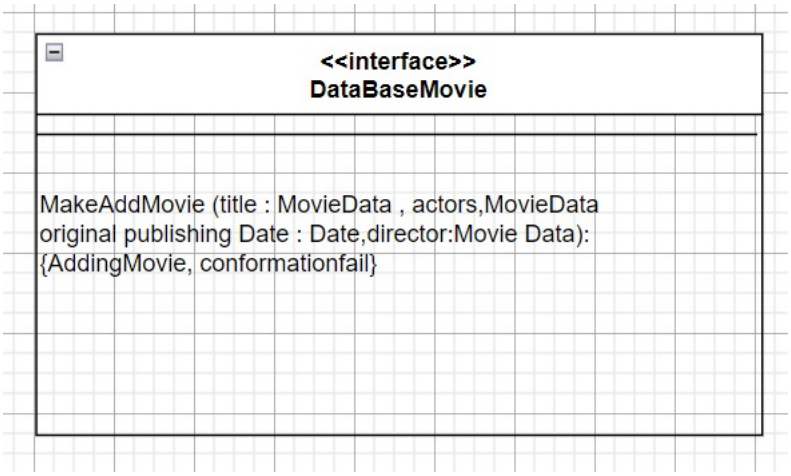


Figure 30 function for R2

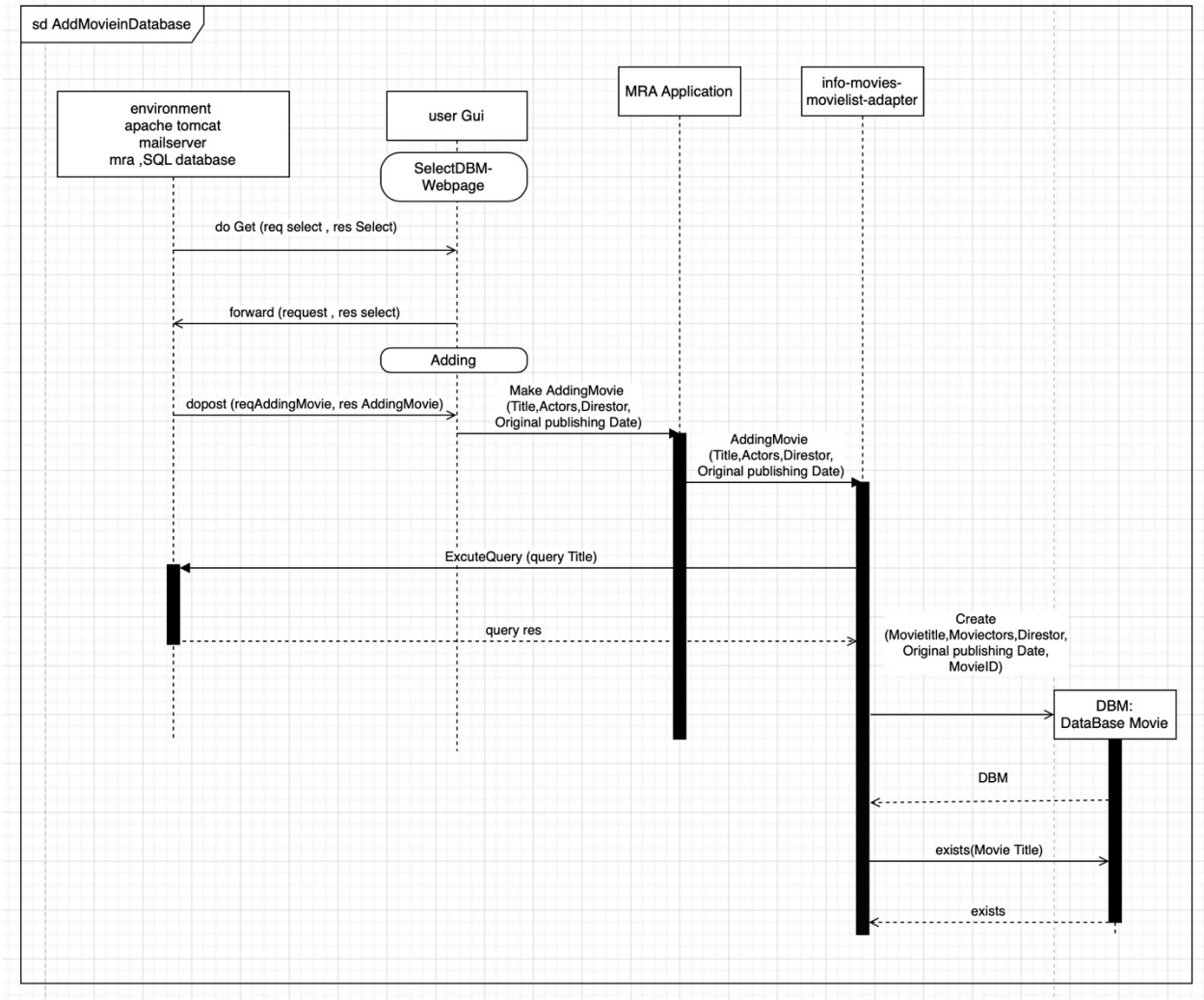


Figure 31 Sequence diagram for R2

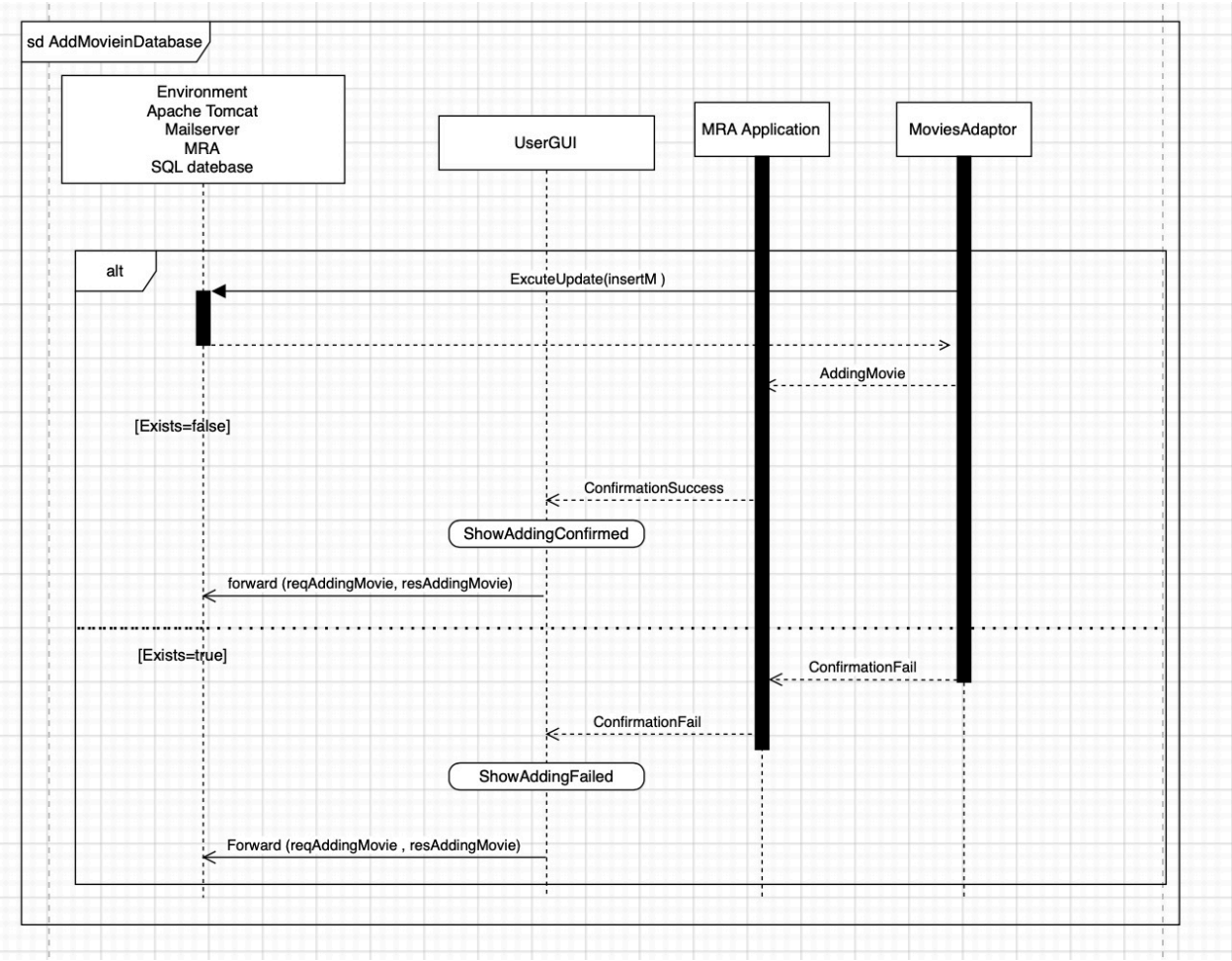


Figure 32 sequence diagram for R2'

Inter-component interaction AddMovieinDatabase-Remarks

1. reqSelect and reqAddingMovie represent HTTPServletRequest objects containing the required user input
2. resSelect and resAddingMovie represent HTTPServletResponse objects as the Counterpart for the request
3. The state predicate Adding represents that the input form for Adding a Movie the selected offer is shown
4. The state predicate ShowAddingConfirmed represents that the confirmation is shown
5. The state predicate ShowAddingFailed represents that an Error message is shown
6. Forward (….) sends the request and response back to the server to generate the HTML webpage.
7. 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.

queryTitle:

SELECT * FROM DataBaseMovie DBM WHERE (DBM.getMovietitle =" Movietitle")

insertMj:

INSERT INTO DataBaseMovie (Movietitle, Movieactors, Director, Original Publishing Date) VALUES ("Movietitle"," Movieactors"," Director"," Original Publishing Date")

Hint:

For the MovieID it will be automatically incremented for each added movie, and this will be defined in MySQL as Following:

CREATE TABLE DataBaseMovie (

MovieID INT AUTOINCREMENT UNIQUE,

……);

- **2.2.4 Sequence diagram and SQL for R7:**
- **Inter-components interaction: MovieOverviewShow**

The operation of making a movie overview

Name: makeAccessMovieList

Description: it is used to able the user to make overview of films and their ratings and return the movies list

OCL constraint:

Context: MRA_MovieOverviewShow:: makeAccessMovieList ()

Pre: True

Post: Let

Let DBM: moviedatabase= MovieDataBase in

Let MovieList: Set (MovieData)= DBM. MovieData.allInstances()->asSet () in

WebpageUser ^ AvailableRatedMovielist_AvailableMovieList (MovieList)

Figure 33 OCL for R7

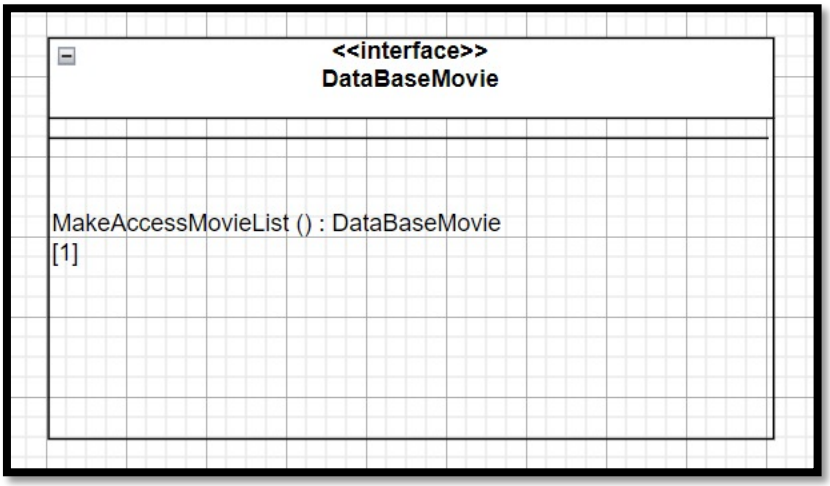


Figure 34 function for R7

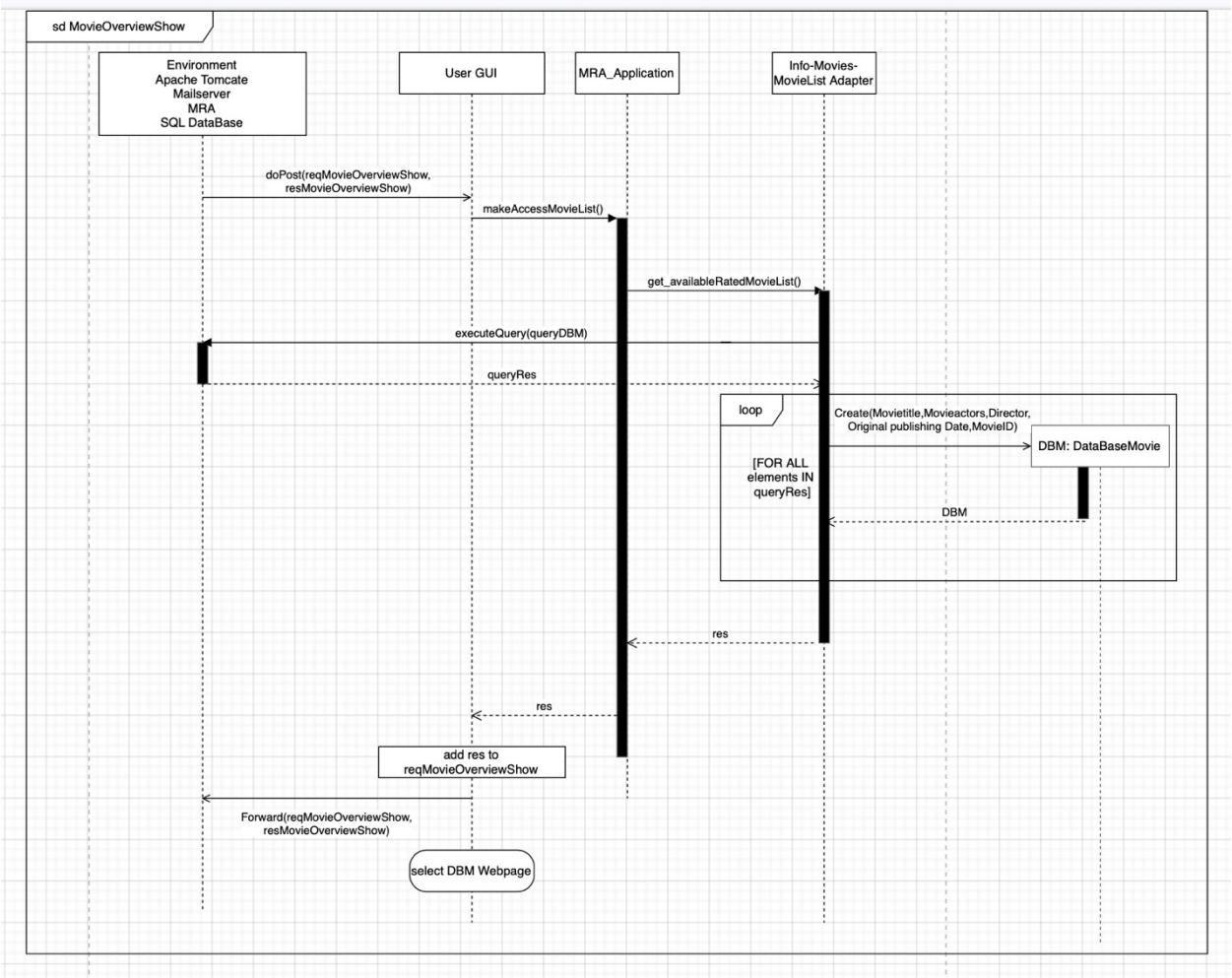


Figure 35 sequence diagram for R7

Inter-component interaction MovieOverviewShow-Remarks

1. reqMovieOverviewShow represents a HTTPServletRequest object containing the required user input
2. resMovieOverviewShow represents a HTTPServletResponse objects as the counterpart for the request
3. The state predicate SelectDBMWebpage represents that the available rated movie list offer is shown
4. DBM refers to an object of class DataBaseMovie
5. Exists (…) checks whether the existence of a movieslist or not
6. res. Is the available Movieslist. that will be added to the request object to be processed by the server
7. forward (…) sends the request and response back to the server to generate the HTML webpage
8. 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.

queryDBM

SELECT *FROM DataBaseMovie DBM LEFT JOIN (SELECT MovieID, Avg (Rate) FROM MovieList GROUP BY MovieID) ML ON (DBM.MovieID=ML.MovieID) ORDER BY Rate

2.2.5 Validation:

The sequence diagrams must be consistent with the behavior described in step A3 and in step A6

- consistency of sd UserRegister and sd UserRegister(A3)

Message in D2	Corresponding message in A3
doGet (reqSelect, resSelect)	Refines registerNewUser
Forward (reqSelect, resSelect)	Refines registerNewUser
doPost (reqRegistering, resRegistering)	Refines registerNewUser
registerNewUser (…)	registerNewUser

Registering (…)	Registering
ExecuteQuery(queryname)	Refines Registering
ExecuteUpdate(insertU)	Refines Registering
Forward (reqRegistering, resRegistering)	Refines FailRegistered
Forward (reqRegistering, resRegistering)	Refines successRegistered

- consistency of sd MovieRate and sd MovieRate(A3)

Message in D2	Corresponding message in A3
doGet (reqSelect, resSelect)	Refines makeRateMovie
Forward (reqSelect, resSelect)	Refines makeRateMovie
doPost (reqRateMovie, resRateMovie)	Refines makeRateMovie
MakeRateMovie (…)	makeRateMovie
RatingMovie (…)	RatingMovie
ExecuteQuery(queryMovieID)	Refines RatingMovie
ExecuteUpdate(insertR)	Refines RatingMovie
Forward (reqRateMovie, resRateMovie)	Refines confirmationRateFailed
Forward (reqRateMovie, resRateMovie)	Refines confirmationRateSuccessed

- consistency of sd AddMovieinDatabase and sd AddMovieinDatabase (A3)

Message in D2	Corresponding message in A3
doGet (reqSelect, resSelect)	Refines makeAddingMovie
Forward (reqSelect, resSelect)	Refines makeAddingMovie

doPost (reqAddingMovie, resAddingMovie)	Refines makeAddingMovie
MakeAddingMovie (…)	makeAddingMovie
AddingMovie (…)	AddingMovie
ExecuteQuery(queryTitle)	Refines AddingMovie
ExecuteUpdate(insertM)	Refines AddingMovie
Forward(reqAddingMovie, resAddingMovie)	Refines ShowAddMovieFailed
Forward(reqAddingMovie, resAddingMovie)	Refines ShowAddMovieSucceed

- consistency of sd MovieOverviewShow and sd MovieOverviewShow (A3)

Message in D2	**Corresponding message in A3**
doPost(reqMovieOverviewShow, resMovieOverviewShow)	Refines AccessMovieList
makeAccessMovieList ()	makeAccessMoviesList
Get_availableRatedMovieList ()	Get_availableRatedMovieList
ExecuteQuery (…)	Refines get_availableRatedMoviesList
forward(reqMovieOverviewShow, resMovieOverviewShow)	Refines Movies'eRepresentation

- consistency with life cycle
    1. [UserRegister]:
       Sd UserRegister can be executed optionally without precondition
    2. AddMovieinDataBase | MovieOverviewShow | MovieRate:
       Sd AddMovieinDataBase or sd MovieOverviewShow or sd MovieRate
       Can be executed an arbitrary number of times without a precondition
    3. [UserRegister] || (AddMovieinDataBase | MovieOverviewShow | MovieRate) *:

The sd UserRegister can be executed concurrently with sd AddMovieinDataBase or sd MovieOverviewShow or sd MovieRate without unwanted side effects.

Validation |

The sequence diagrams must realize the operations described in step A5

- registerNewUser (…) is realized in sd UserRegister:
  - precondition does not have to be established because it is true
  - postcondition is established, if the entered age of the user and the name is unique and does not exist in the DataBaseinfo that will take place by queryname and it is checked if this name exists, if it does not and the age more or equal 18 then the corresponding Registering is made using insertU. The PersonGUI is instructed to show the confirmation by the return value of ConfirmationAccepted. If the name exists already then the PersonGUI is instructed to show the fail information by the return value of ConfirmationRejected
- makeRateMovie (…) is realized in sd MovieRate:
  - precondition does not have to be established because it is true
  - postcondition is established, if the entered MovieID using queryMovieID exists and the function UserFirstRate returns true, so the corresponding RatingMovie is made using insertR. The UserGUI is instructed to show the confirmation by the return value of RateSuccessed. If the function UserFirstRate returns false, so the UserGUI is instructed to show the fail information by the return value of RateFailed
- MakeAddingMovie (…) is realized in sd AddMovieinDataBase:
  - Precondition does not have to be established because it is true
  - Postcondition is established, it will be checked if the movie already exists or not using queryTitle, in case the function exists with input Movietitle returns false, so the UserGUI is instructed to show the confirmation by the return value of ConfirmationSuccess and in case it returns true, so the UserGUI is instructed to show the fail information by the return value of ConfirmationFail
- makeAccessMovieList () is realized in sd MovieOverviewShow:
  - precondition does not have to be established because it is true
  - postcondition is established, using the SQL command queryDBM, ML_Adapter selects the existent Movieslist with a unique ID for each Movie with an Average Rating. MRA_Application forwards the result to UserGUI. That realizes WebpageUser^AvailableRatedMovieList_AvailableMovieList (MovieData)

Validation: all messages in the application interface classes of step D1 must be used in some sequence diagram

interface	Message	Used in sequence diagram
PCmds	registerNewUser	Sd UserRegister
Info	Registering	Sd UserRegister
UCmds	MakeAddingMovies makeRateMovie makeAccessMovieList	Sd AddMovieinDatabase Sd MovieRate Sd MovieOverviewShow
Movies	AddingMovies availableMovieList	Sd AddMovieinDataBase Sd MovieOverviewShow
Movielist	RatingMovie availableRatedMovieList	Sd MovieRate Sd MovieOverviewShow

Validation: the directions of messages must be consistent with the required and provided interfaces of step D1

Interface	Provided by	Required by
Message	**Recipient**	**Sender**
PCmds	MRA_Application	PersonGUI
registerNewUser	MRA_Application	PersonGUI
Info	InfoAdapter	MRA_Application
Registering	InfoAdapter	MRA_Application
UCmds	MRA_Application	UserGUI
MakeAddingMovies	MRA_Application	UserGUI
makeRateMovie	MRA_Application	UserGUI
makeAccessMovieList	MRA_Application	UserGUI
Movies	MoviesAdapter	MRA_Application

AddingMovies	MoviesAdapter	MRA_Application
availableMovieList	MoviesAdapter	MRA_Application
Movielist	MovieListAdapter	MRA_Application
RatingMovie	MovieListAdapter	MRA_Application
availableRatedMovieList	MovieListAdapter	MRA_Application

Validation: messages must connect components as connected in the software architecture of step D1

Component	Connected components in architecture	Connected components in sequence diagrams
MRA_Application	InfoAdapter, MoviesAdapter, PersonGUI, UserGUI	InfoAdapter, MoviesAdapter, PersonGUI, UserGUI
InfoAdapter	MRA_Application, Environment	MRA_Application, Environment
MoviesAdapter	MRA_Application, Environment	MRA_Application, Environment
MovieListAdapter	MRA_Application, Environment	MRA_Application, Environment
PersonGUI	MRA_Application, Environment	MRA_Application, Environment
UserGUI	MRA_Application, Environment	MRA_Application, Environment

Hence, for all components the sets of connected components in the architecture and in the sequence, diagrams are the same.

2.3 D3		

2.3.1 Inter-Component interaction for R1:

UserRegister:

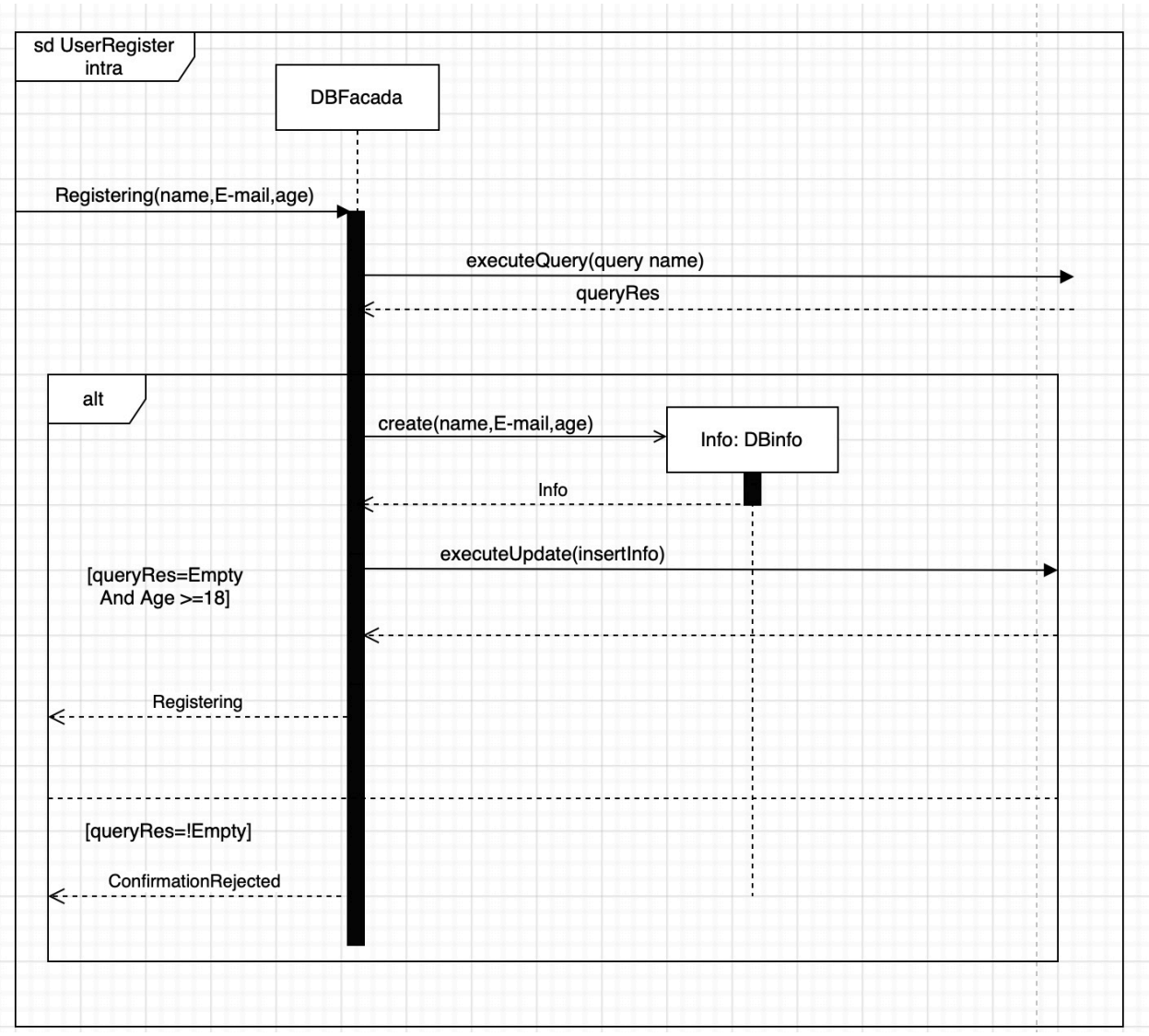


Figure 36 Inter-Component interaction for R1

Query name: SELECT * FROM DataBaseinfo DBI WHERE (DBI.name="name")

InsertInfo: INSERT INTO DataBaseinfo (name, age, E-mail) VALUES ("name"," age"," E-mail")

2.3.2 Inter-Component interaction for R2:

- **AddMovieinDatabase:**

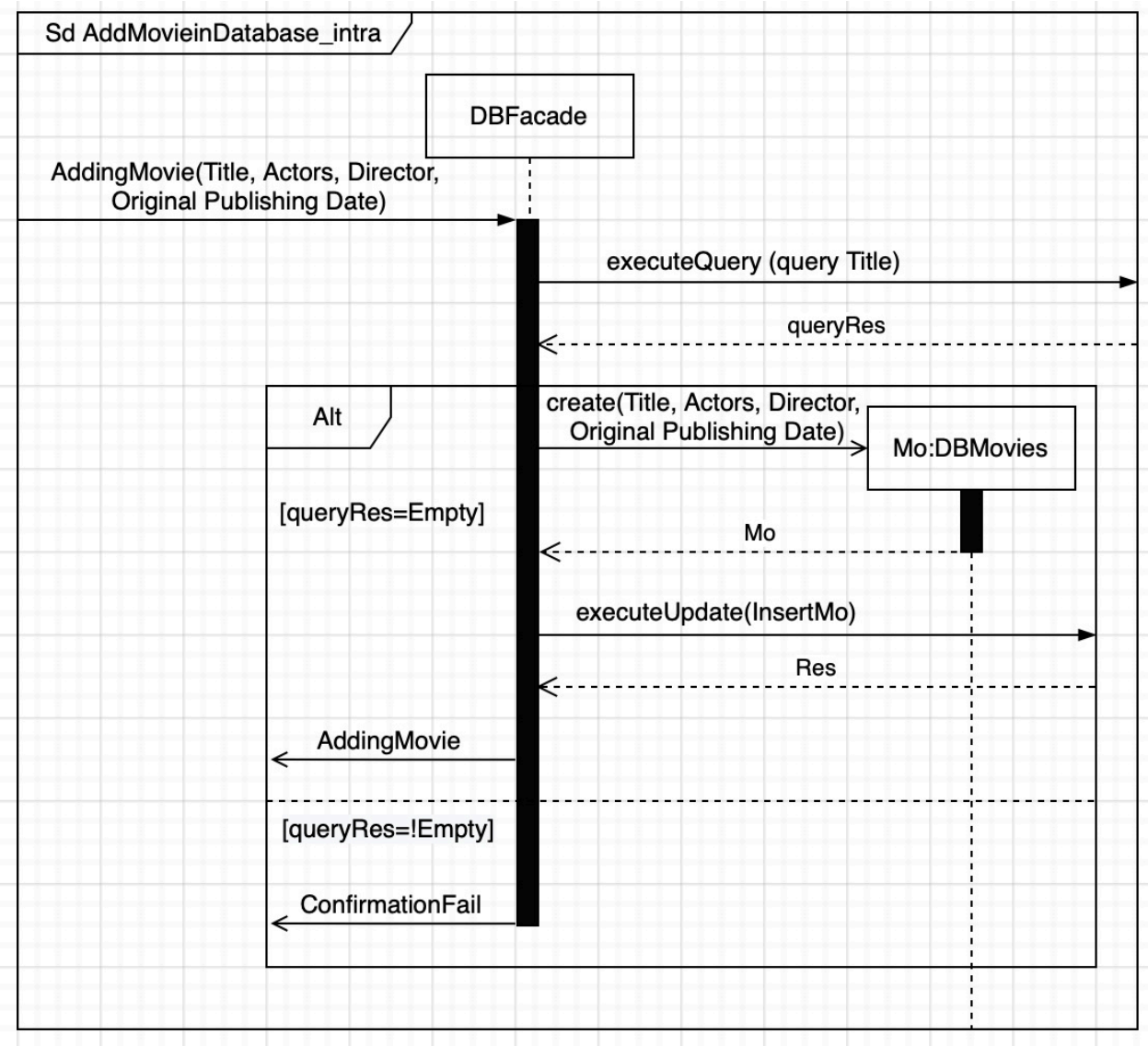


Figure 37 inter-component interaction for R2

Query title:

SELECT * FROM DataBaseMovie DBM WHERE (DBM. getMovietitle =" Movietitle")

InsertMo:

INSERT INTO DataBaseMovie (Movietitle, Movieactors, Director, Original Publishing Date)

VALUES ("Movietitle"," Movieactors"," Director"," Original Publishing Date")

2.3.3 Inter-Component interaction for R3:

MovieRate:

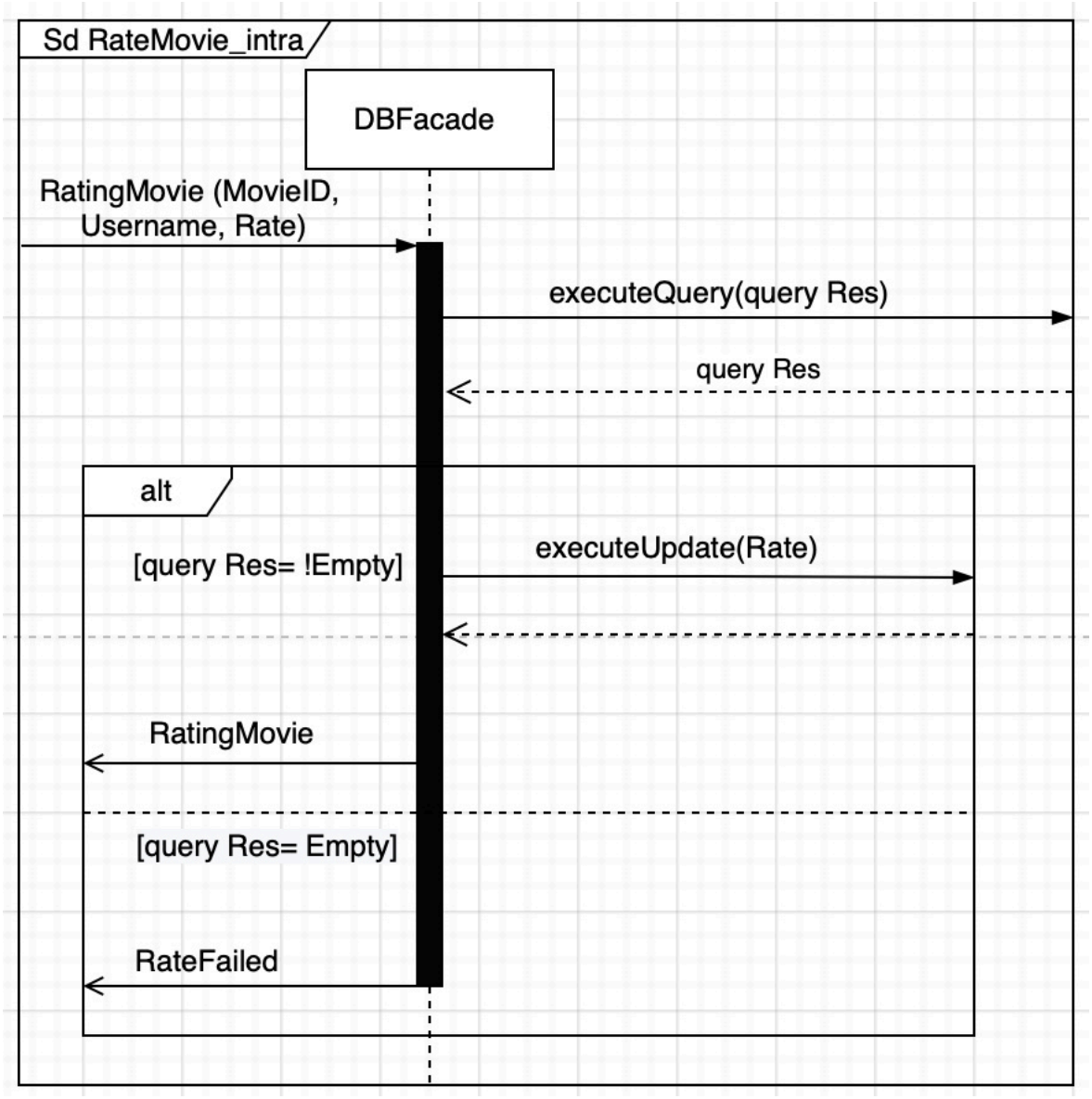


Figure 38 Inter-Component interaction for R3:

Query Res:

SELECT * FROM MovieList ML WHERE (ML. MovieID = “MovieID” AND ML. Username=” Username” AND ML. Rate = 0)

ExecuteUpdate:

UPDATE MOVIELIST SET Rate = “new Value” WHERE (MovieID=” MovieID”)

2.3.4 Preliminary architectural description:

- **DataBase_Adapter is combination of MovieAdapter, MovieListAdapter and InfoAdapter**

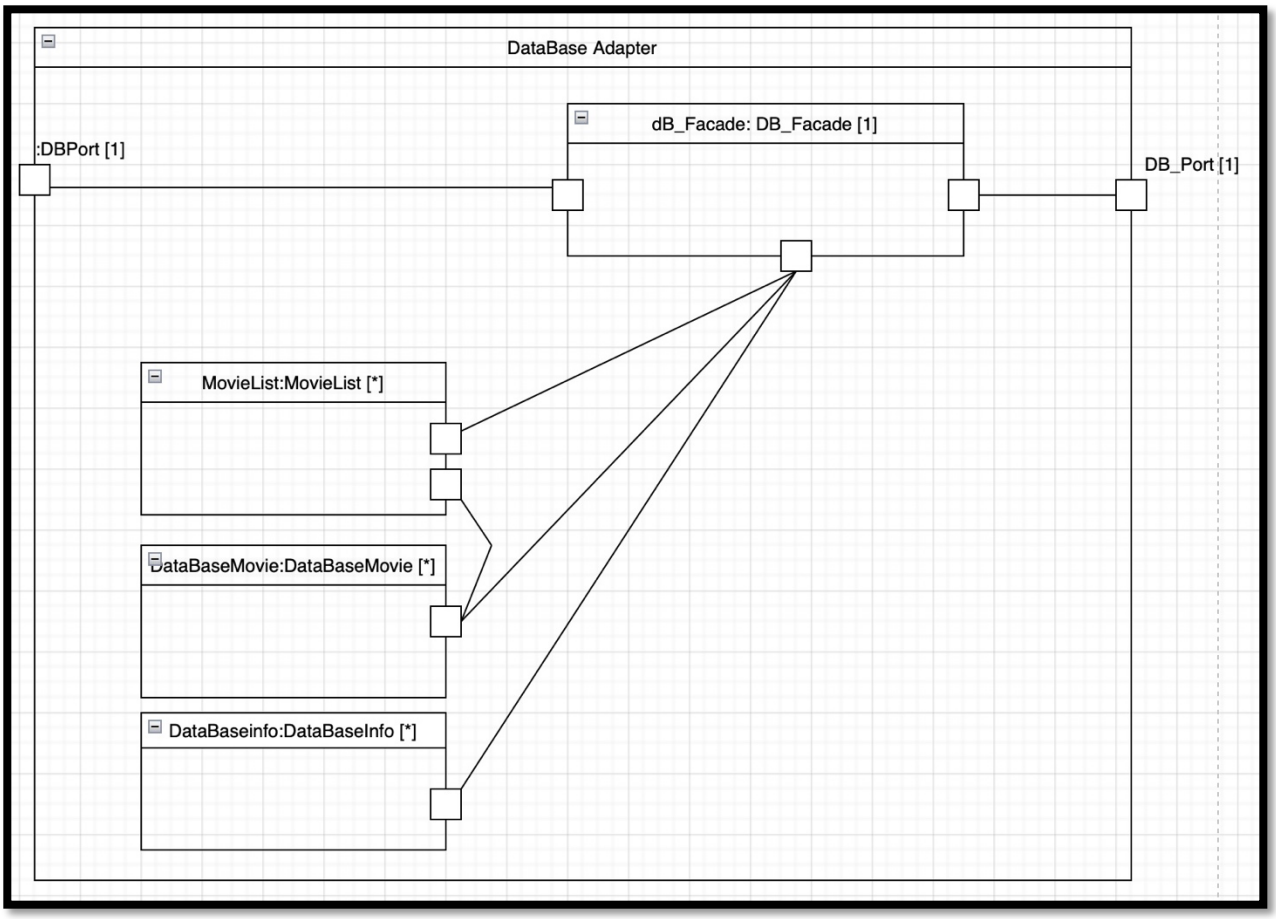


Figure 39 Preliminary architectural description

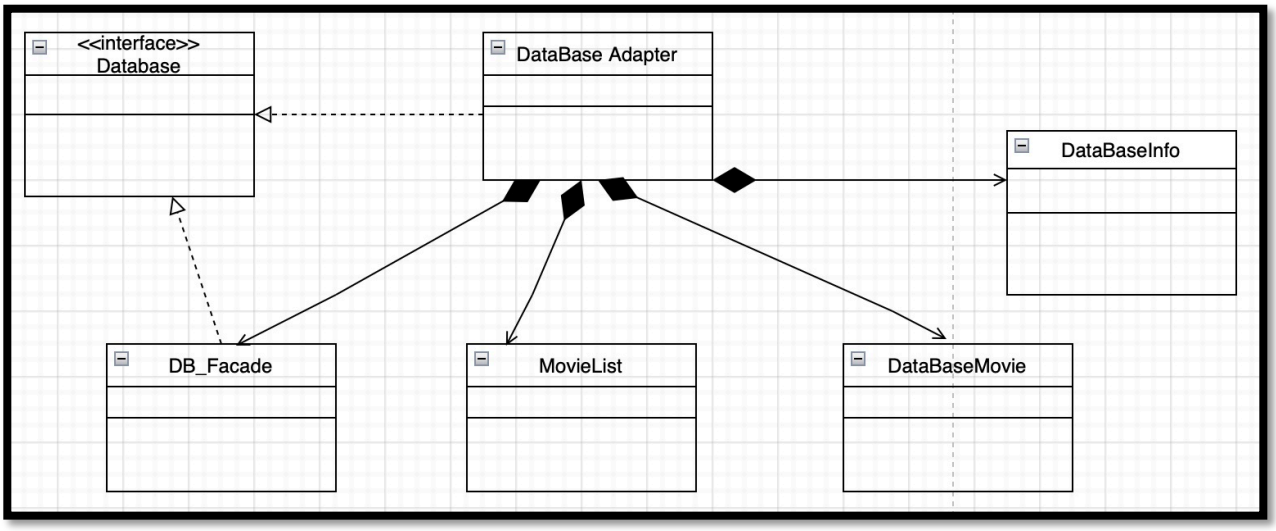


Figure 40 Preliminary architectural description2

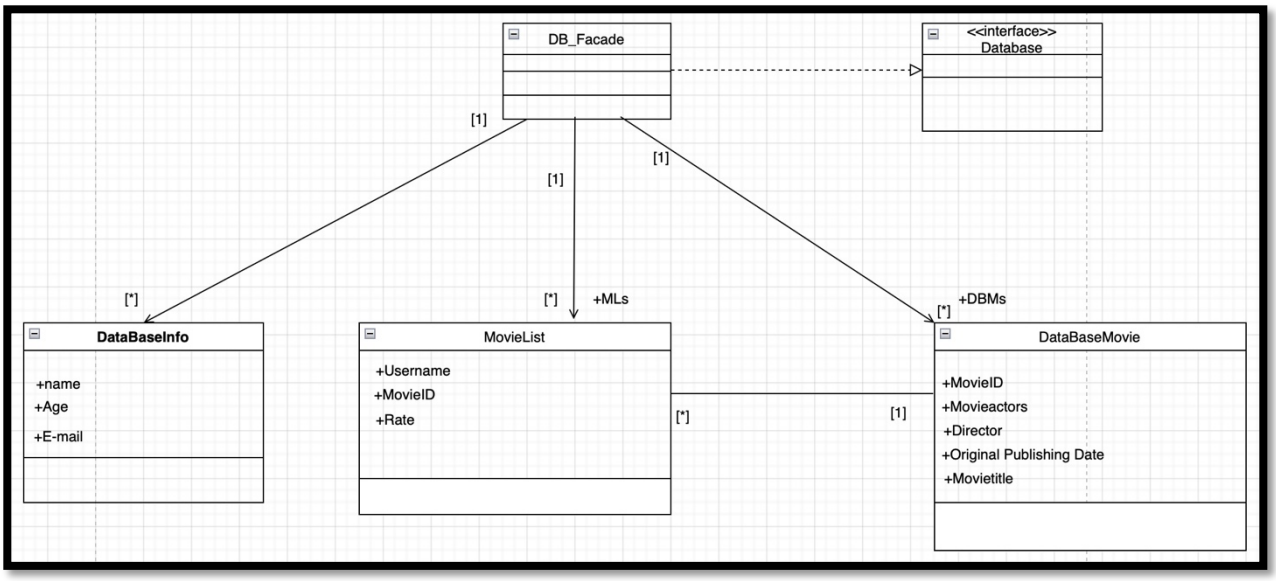


Figure 41 Preliminary architectural description3

2.3.5 Inter-Component interaction for R7:

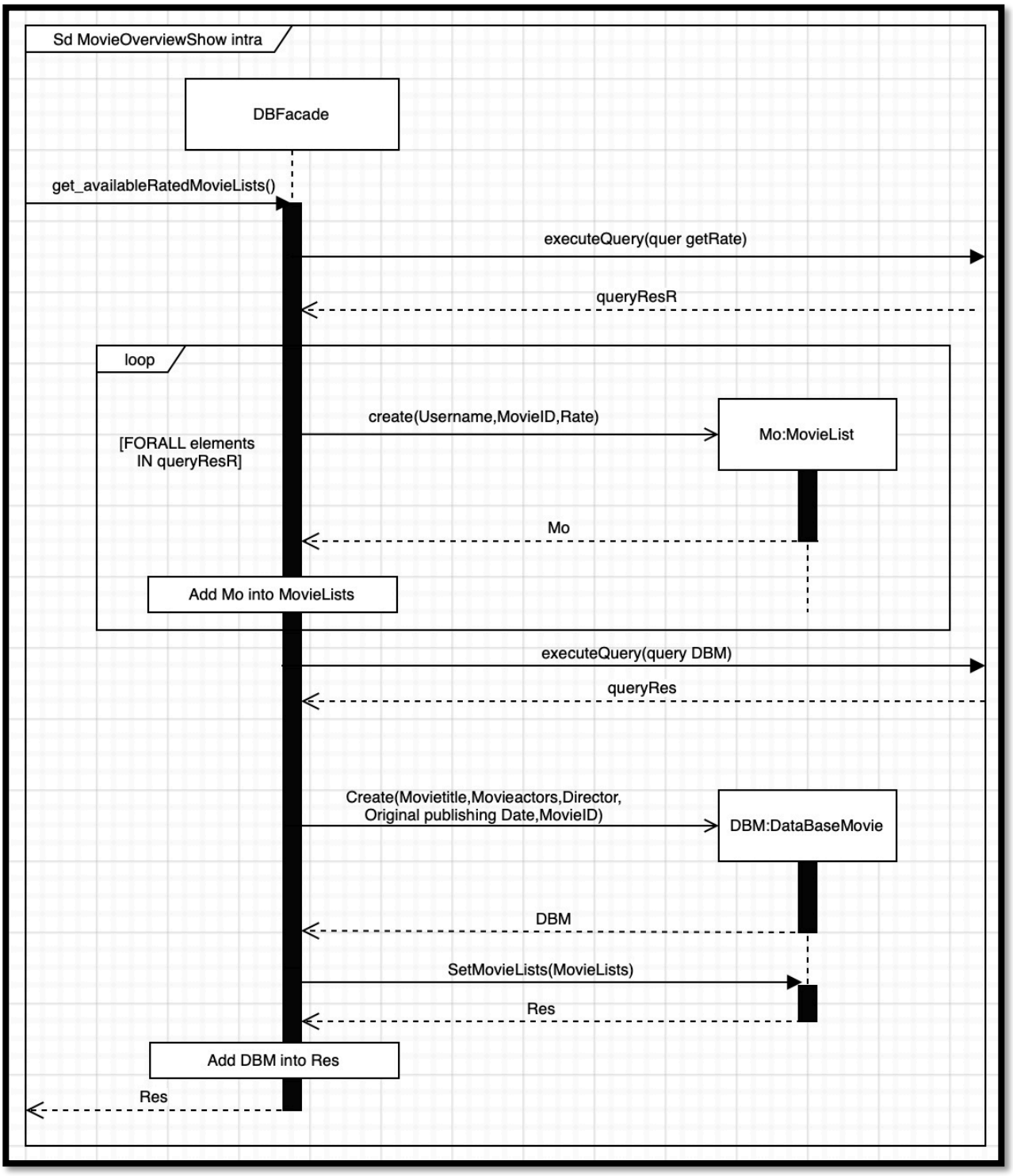


Figure 42 Inter-Component interaction for R7

In order to get the average Rating for each Movie the corresponding query must be made:

Query getRate:

SELECT LS. MovieID, avg (LS. Rate) FROM MovieList LS GROUP BY LS. MovieID

Query DBM:

SELECT *FROM DataBaseMovie DBM LEFT JOIN (SELECT MovieID, Avg (Rate) FROM MovieList GROUP BY MovieID) ML ON (DBM.MovieID=ML.MovieID) ORDER BY Rate

- setMovieLists (MovieLists: MovieList [*]) sets the association between the collection of MovieLists objects and the DataBaseMovie.
- Task add Mo into MovieLists represents a function that inserts Mo into an array named MovieList.

Mo returns the following:

Username	
MovieID	**Rate**
---	---

The query getRate will return the following:

MovieID	AvgRating
---	---

The DBM will return the following:

MovieID	Movietitle	Movieactors	Original publishing Date	Director	Avg

| ---- | ------- | ------ | ------ | ------ | ---- |
| | | | | | |

the above query will be ordered by Rate ascendingly.

2.3.6 Final architectural description:

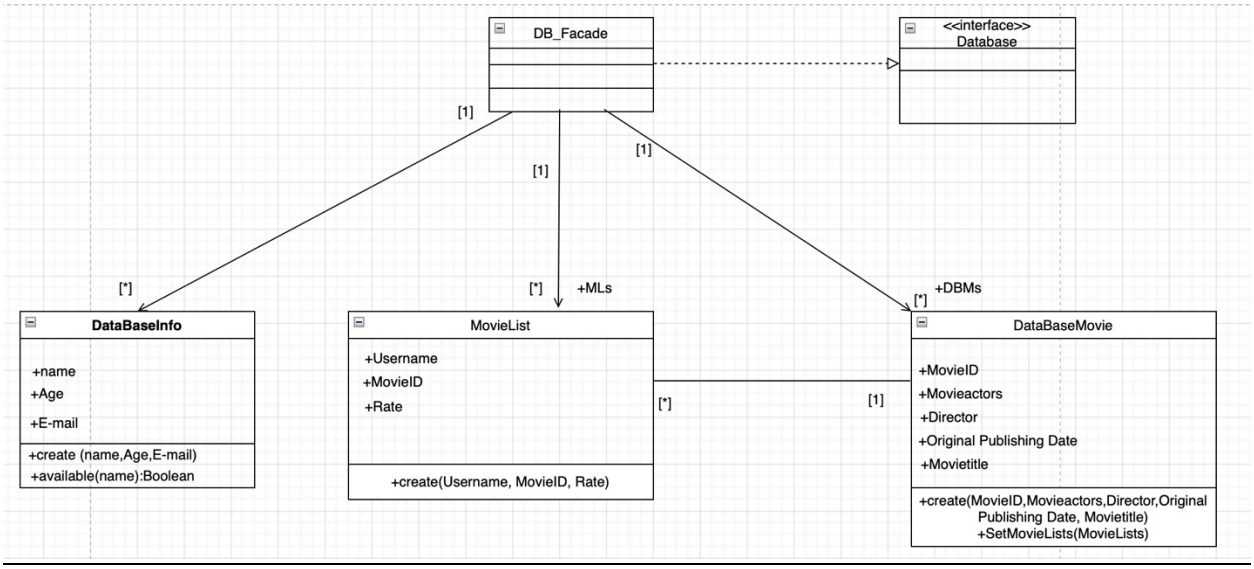


Figure 43 final architectural description

2.3.67 Validation:

Sequence diagrams of one component must be consistent with the corresponding interface behavior in Step D2:

Inter-Component Interaction:

Messages of sequence diagram Sd UserRegister- intra in Step D3: Intra-Component interaction	**Messages of sequence diagram Sd UserRegister in Step D2: Inter-Component interaction**
Registering (…….)	Registering (….)
ExecuteQuery (query name)	ExecuteQuery (query name)
Create (……..)	Refinement

ExecuteUpdate (insertInfo)	Refinement

Validation 2:

Messages of sequence diagram Sd-AddMovieinDatabase intra in Step D3: Intra-Component interaction	**Messages of sequence diagram Sd - AddMovieinDatabase in Step D2: Inter-Component interaction**
AddingMovie (……)	AddingMovie (….)
ExecuteQuery (query title)	ExecuteQuery (query title)
Create (……)	Refinement
ExecuteUpdate (InsertMo)	Refinement

Validation 3:

Messages of sequence diagram Sd-MovieRate intra in Step D3: Intra-Component interaction	**Messages of sequence diagram Sd-MovieRate in Step D3: Inter-Component interaction**
RateMovie (….)	RateMovie (….)
ExecuteQuery (query Res)	ExecuteQuery (query Res)
ExecuteUpdate (Rate)	ExecuteUpdate (InsertR)

Validation 4:

Messages of sequence diagram Sd-MovieOverviewShow intra in Step D3: Intra-Component interaction	**Messages of sequence diagram Sd-MovieOverviewShow in Step D3: Inter-Component interaction**
getAvailableRatedMovieList ()	makeAccessMovieList ()
ExecuteQuery (queryDBM)	ExecuteQuery (queryDBM)
Create (….)	Refinement
ExecuteQuery (query getRate)	Refinement

Create (….)	Refinement

it must be possible to relate any new state (predicates) to the state predicates of step D2: Inter-Component Interaction

New state (predicates) in step D3: Intra-Component Interaction	**New state (predicates) in step D2: Inter-Component Interaction**
----	---

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

2.4 D4		

Inter-Component interaction: We have done above.

2.4.1 Complete Component or Class Behavior:

Check whether a state machine is necessary:

- The component **PersonGUI**: no refinement exists in step D3: Intra-Component Interaction; Continue with looking at step D2: Inter-Component Interaction: There are two states. Therefore, a state machine is required.
- The Component **UserGUI**: no refinement exists in step D3: Intra-Component Interaction; Continue with looking at step D2: Inter-Component Interaction: There are more than two states. Therefore, a state machine is required.
- The Component **Database Adapter**: There exists a refinement in Step D3: Intra-Component Interaction. Hence, we have to look at the DBFacade, no state machine is necessary for this component.
- The component **MRA_Application**: there is no refinement of this component in Step D3: Intra-Component Interaction; continue looking at Step D2: Inter-Component Interaction. Most of the time, the machine gets an input message that is passed on. The machine then waits for the results. Furthermore, the life cycle is ensured via the User It is not necessary to create a state machine for this component.

For the necessary state machines, perform the following according to the corresponding sequence diagrams: considering UserGUI:

I. Messages doGet (…) and doPost (…) must be triggers at transitions
II. Messages makeAccessMovieList (), makeAddMovie (…), makeRateMovie (), registerNewUser () and forward must be output signals or actions
III. State predicate SelectDBM-Webpage, SelectML-Webpage, Rate, Registering, Adding, ShowAddingConfirmed, ShowAddingFailed, ShowRateConfirmed, ShowRateFailed, ShowRegisteringConfirmed and ShowRegisteringFailed must be states
IV. Idle and DefaultWebpage are new states required to represent the functionalities of the web application

2.4.2 state machine:

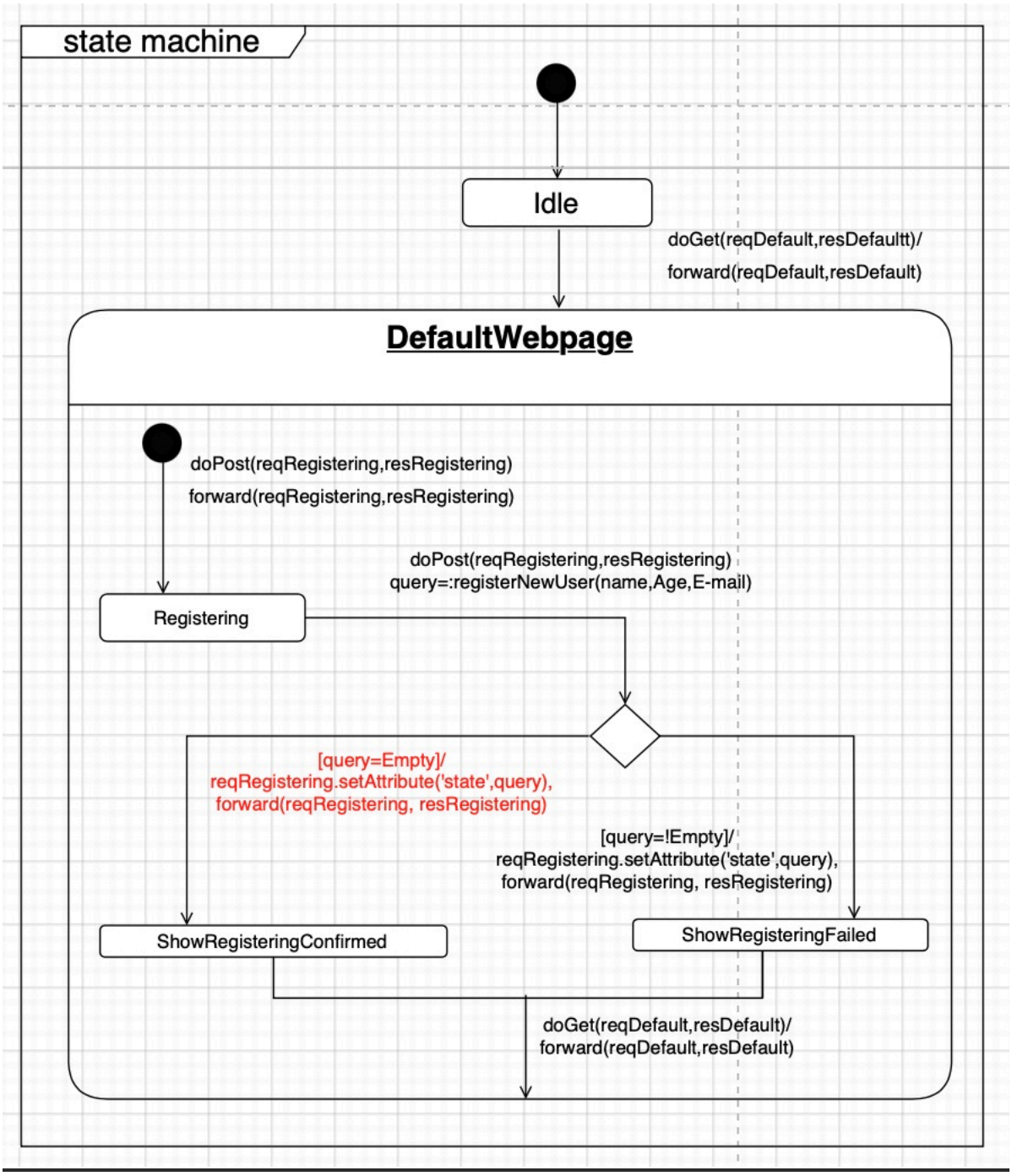


Figure 44 state machine for R1

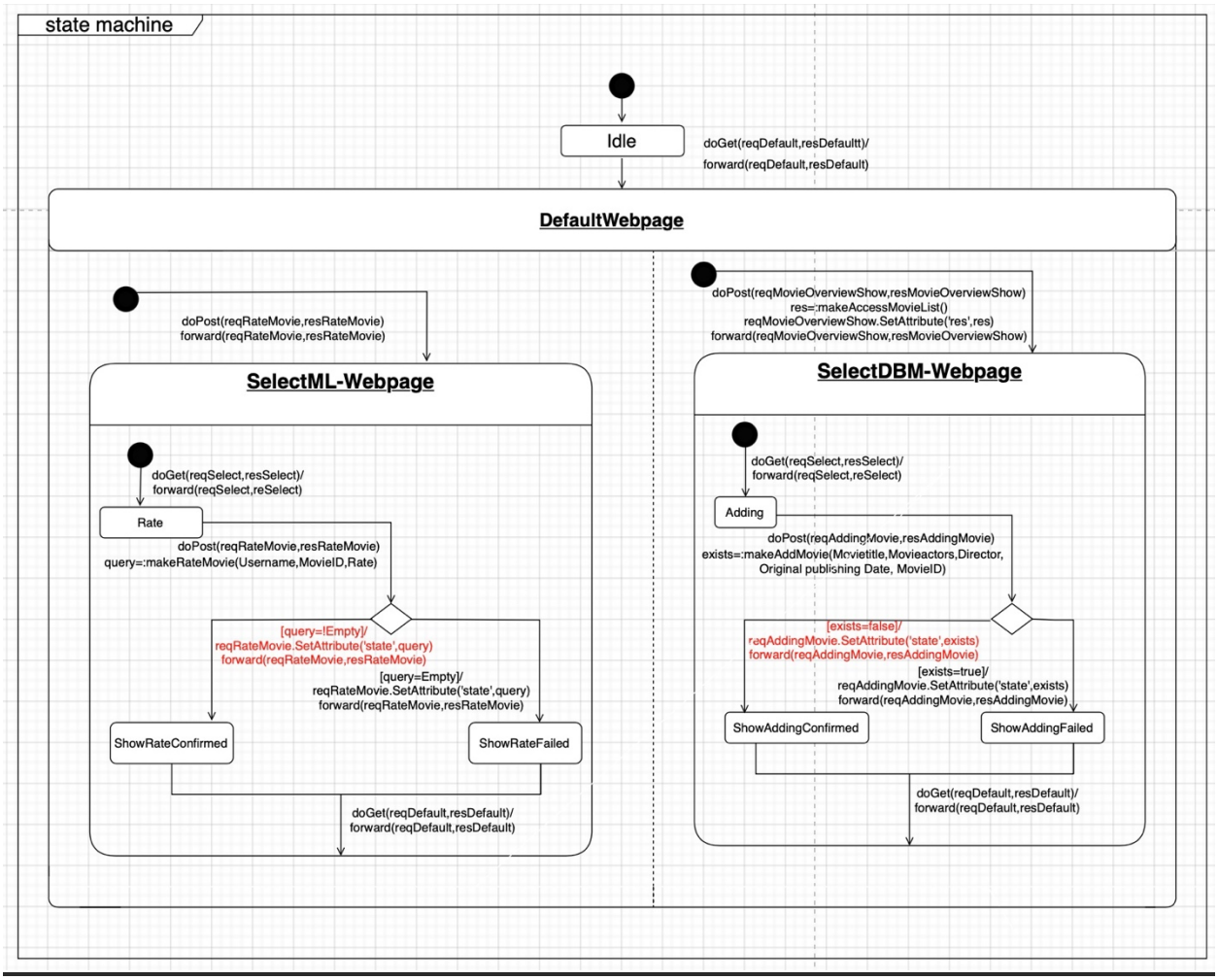


Figure 45 state machine for R2,R3,R7

MovieOverviewShow:

Can be performed starting from idle by sending a signal doPost (reqMovieOverviewShow, resMovieOverviewShow) and if someone wants to add a movie, we should access to the Movie list at first by making MovieOverviewShow then transition to state Adding to decide if the film can be added or not. Rating a Movie can occur concurrently, with MovieOverviewShow and AddMovieinDatabase.

We add the following additional transitions to model the complete behavior of the Web application:

- doGet (reqDefault, resDefault) is a trigger that represents the initial request for the webpage when entering the URL

- forward (reqDefault, resDefault) is the corresponding action to generate the starting page
- setAttribute ('name', value) is a method to use results in a generated webpage.

2.4.3 Validation:

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

Component UserGUI & PersonGUI					
Source state	**Target state**	**Input signal**	**Mapped to Message(s)**	**Output signal**	**Mapped to message(s)**
Init	DefaultWebpage	doPost (reqRegistering,)	doPost (reqRegistering,)	Res=: registerNewUser (name, Age, E-Mail) reqRegistering. SetAttribute (‚res ', res) forward (reqRegistering, resRegistering)	Res=: registerNewUser (name, Age, E-Mail) forward (reqRegistering, resRegistering)
Init	SelectDBM-Webpage	doPost (reqMovieOverviewShow,)	doPost (reqMovieOverviewShow,)	Res=: makeAccessMovieList () reqMovieOverviewShow. SetAttribute ('res', res) forward (reqMovieOverviewShow,)	Res=: makeAccessMovieList () forward (reqMovieOverviewShow,)
Init	SelectML-Webpage	doPost (reqRateMovie,)	doPost (reqRateMovie,)	Res=: makeRateMovie (Username,	Res=: makeRateMovie (Username,

				MovieID, Rate) reqRateMovie. SetAttribute ('res', res) forward (reqRateMovie,)	MovieID, Rate) forward (reqRateMovie,)
SelectML-Webpage	Rate	doGet (reqSelect,)	doGet (reqSelect,)	Forward (reqSelect,)	Forward (reqSelect,)
Rate	ShowRateConfirmed	doPost (reqRateMovie,)	doPost (reqRateMovie,)	Query=: makeRateMovie (Username, MovieID, Rate) reqRateMovie. SetAttribute ('state', query) Forward (reqRateMovie,)	Query=: makeRateMovie (Username, MovieID, Rate) Forward (reqRateMovie,)
Rate	ShowRateFailed	doPost (reqRateMovie,)	doPost (reqRateMovie,)	Query=: makeRateMovie (Username, MovieID, Rate) reqRateMovie. SetAttribute ('state', query) Forward (reqRateMovie,)	Query=: makeRateMovie (Username, MovieID, Rate) Forward (reqRateMovie,)
ShowRateConfirmed	SelectML-Webpage	doGet (reqDefault,)	-----	Forward (reqDefault,)	-----
ShowRateFailed	SelectML-Webpage	doGet (reqDefault,)	-----	Forward (reqDefault,)	-----

SelectDBM-Webpage	Adding	doGet (reqSelect,)	doGet (reqSelect,)	Forward (reqSelect,)	Forward (reqSelect,)
Adding	ShowAddingConfirmed	doPost (reqAddingMovie,)	doPost (reqAddingMovie,)	Exists=: makeAddMovie (Movietitle, Movieactors, Director, Original publishing Date, MovieID) reqAddingMovie. SetAttribute ('state', exists) forward (reqAddingMovie,)	Exists=: makeAddMovie (Movietitle, Movieactors, Director, Original publishing Date, MovieID) forward (reqAddingMovie,)
Adding	ShowAddingFailed	doPost (reqAddingMovie,)	doPost (reqAddingMovie,)	Exists=: makeAddMovie (Movietitle, Movieactors, Director, Original publishing Date, MovieID) reqAddingMovie. SetAttribute ('state', exists) forward (reqAddingMovie,)	Exists=: makeAddMovie (Movietitle, Movieactors, Director, Original publishing Date, MovieID) forward (reqAddingMovie,)
ShowAddingFailed	SelectDBM-Webpage	doGet (reqDefault,)	------	Forward (reqDefault,)	------

ShowAddingConfirmed	SelectDBM-Webpage	doGet (reqDefault,)	------	Forward (reqDefault,)	------
DefaultWebpage	Registering	doGet (reqSelect,)	doGet (reqSelect,)	Forward (reqSelect,)	Forward (reqSelect,)
Registering	ShowRegisteringConfirmed	doPost (reqRegistering,)	doPost (reqRegistering,)	Available=: select * from DataBaseinfo DBI where (DBI.name=" name") reqRegistering. SetAttribute ('state', available) Forward (reqRegistering,)	Available=: select * from DataBaseinfo DBI where (DBI.name=" name") Forward (reqRegistering,)
Registering	ShowRegisteringFailed	doPost (reqRegistering,)	doPost (reqRegistering,)	Available=: select * from DataBaseinfo DBI where (DBI.name=" name") reqRegistering. SetAttribute ('state', available) Forward (reqRegistering,)	Available=: select * from DataBaseinfo DBI where (DBI.name=" name") Forward (reqRegistering,)
ShowRegisteringConfirmed	DefaultWebpage	doGet (reqDefault,)	------	Forward (reqDefault,)	------
ShowRegisteringFailed	DefaultWebpage	doGet (reqDefault,)	------	Forward (reqDefault,)	-----

- 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 & PersonGUI	
LC_{user}= ((MovieOverviewShow)+; AddMovieinDatabase | MovieOverviewShow | MovieRate)* LP_{person}=[UserRegister]	
state	**Covered by life cycle part**
Init	MovieOverviewShow
DefaultWebpage	MovieOverviewShow
SelectDBM-Webpage	MovieOverviewShow
SelectML-Webpage	MovieRate
Rate	MovieRate
Registering	UserRegister
Adding	AddMovieinDatabase
ShowAddingFailed	AddMovieinDatabase
ShowAddingConfirmed	AddMovieinDatabase
ShowRateConfirmed	MovieRate
ShowRateFailed	MovieRate
ShowRegisterationConfirmed	UserRegister
ShowRegisterationFailed	UserRegister

- All transitions are covered by a life cycle:

Component UserGUI & PersonGUI
LC_{user}= ((MovieOverviewShow)+; AddMovieinDatabase | MovieOverviewShow | MovieRate)*

LP_{person}=[UserRegister]				
Source state	**Target state**	**Input signal**	**Output signal**	**Life cycle part**
Init	DefaultWebpage	doPost (reqRegistering,)	Res=: registerNewUser (name, Age, E-Mail) reqRegistering. SetAttribute (‚res ‘, res) forward (reqRegistering, resRegistering)	(MovieOverview Show
Init	SelectDBM-Webpage	doPost (reqMovieOverviewShow,)	Res=: makeAccessMovieList () reqMovieOverviewShow. SetAttribute (‘res’, res) forward (reqMovieOverviewShow,)	MovieOverview Show
Init	SelectML-Webpage	doPost (reqRateMovie,)	Res=: makeRateMovie (Username, MovieID, Rate) reqRateMovie. SetAttribute (‘res’, res) forward (reqRateMovie,)	MovieRate
SelectML-Webpage	Rate	doGet (reqSelect,)	Forward (reqSelect,)	MovieRate
Rate	ShowRateConfirmed	doPost (reqRateMovie,)	Query=: makeRateMovie	MovieRate) *

			(Username, MovieID, Rate) reqRateMovie. SetAttribute ('state', query) Forward (reqRateMovie,)	
Rate	ShowRateFailed	doPost (reqRateMovie,)	Query=: makeRateMovie (Username, MovieID, Rate) reqRateMovie. SetAttribute ('state', query) Forward (reqRateMovie,)	MovieRate) *
ShowRateConfirmed	SelectML-Webpage	doGet (reqDefault,)	Forward (reqDefault,)	MovieRate) *
ShowRateFailed	SelectML-Webpage	doGet (reqDefault,)	Forward (reqDefault,)	MovieRate) *
SelectDBM-Webpage	Adding	doGet (reqSelect,)	Forward (reqSelect,)	(MovieOverview Show)+
Adding	ShowAddingConfirmed	doPost (reqAddingMovie,)	Exists=: makeAddMovie (Movietitle, Movieactors, Director, Original publishing Date, MovieID) reqAddingMovie . SetAttribute ('state', exists) forward (reqAddingMovie,)	(MovieOverview Show)+; AddMovieinDatabase

Adding	ShowAddingFailed	doPost (reqAddingMovie,)	Exists=: makeAddMovie (Movietitle, Movieactors, Director, Original publishing Date, MovieID) reqAddingMovie. SetAttribute ('state', exists) forward (reqAddingMovie,)	(MovieOverviewShow)+; AddMovieinDatabase
ShowAddingFailed	SelectDBM-Webpage	doGet (reqDefault,)	Forward (reqDefault,)	((MovieOverviewShow)+; AddMovieinDatabase) *
ShowAddingConfirmed	SelectDBM-Webpage	doGet (reqDefault,)	Forward (reqDefault,)	((MovieOverviewShow)+; AddMovieinDatabase) *
DefaultWebpage	Registering	doGet (reqSelect,)	Forward (reqSelect,)	[UserRegister]
Registering	ShowRegisteringConfirmed	doPost (reqRegistering,)	Available=: select * from DataBaseinfo DBI where (DBI.name="name") reqRegistering. SetAttribute ('state', available) Forward (reqRegistering,)	[UserRegister]

Registering	ShowRegistering Failed	doPost (reqRegistering,)	Available=: select * from DataBaseinfo DBI where (DBI.name=”name”) reqRegistering. SetAttribute (‘state’, available) Forward (reqRegistering,)	[UserRegister]
ShowRegistering Confirmed	DefaultWebpage	doGet (reqDefault,)	Forward (reqDefault,)	[UserRegister]
ShowRegistering Failed	DefaultWebpage	doGet (reqDefault,)	Forward (reqDefault,)	[UserRegister]

Glossary		

Name	Type	Description	Source
A			
AD		Abbreviation for administrator	CD
Add a member	Phenomenon	Add a member to group	Requirement
AddingRegisteredUser	Phenomenon	Adding a registered user to a group by adding it in data base	CD
AddingMovies	Phenomenon	Adding a new movie to data base	CD, PD-AddMovieInDatabase
AcessingMovies'List	Phenomenon	Machine give permission to user to access movies’ list stored on the webserver	CD PD-MovieOverviewShow

		and sorted in descending order.	
AccessMoviesList	Phenomenon	the user can access a list of all movies	CD, PD-MovieOverviewShow
AddMovie	Phenomenon	the User can add a movie if it is not on the list, by providing the movie's title, director, the main actor (at least one), and original publishing.	CD,PD-AddMovieInDatabase
AddRegisteredUser	Phenomenon	Any registered user can add other registered users into a movie discussion group.	CD
administrator	Biddable domain	A user can be an administrator if he creates a group	CD
AddedMovie	Phenomenon	Indicate that a movie is added	CD
avaliableMovieList	Phenomenon	Database show a movie list	CD, PD-MovieOverviewShow
Ad		Aberration for banning the member	CD
availableRatedMovies'list	Phenomenon	Showing the available rated movie list	CD, PD-MovieOverviewShow
Apache Tomat	Connection Domain	An open source JSP and servlet container from the apache foundation	TCD
Age	Attribute	The age of the person who wants to register him/herself	Class diagram
Available ()	Message	Checks if the name of the person already exists or not	Sd UserRegister (D2)
AvgRating	Attribute	Average rating of a movie in Data base	Class Model

Add res to reqMovieOverviewShow	State predicate	Response to the request of overview	Sd MovieOverviewShow (D2)
Adding	State	Indicates adding a movie takes place	State machine
B			
Ban a member	Phenomenon	Admin can block members	Requirement
banningMember	Phenomenon	Banning a user from a group in data base	CD
banMember	Phenomenon	An administrator can ban a user from a group if he misbehaves	CD
C			
Chat	Phenomenon	Members can chat together in group	Requirement and Domain
ChoosingMovies	Phenomenon	choosing a movie from a Movies' list	CD
chooseMovie	Phenomenon	A user can choose a movie from a Movies' list	CD
chatinginGroup	Phenomenon	A user can chat in group	CD
chatingroup	Phenomenon	Allow to user to chat in group	CD
creategroup	Phenomenon	Allowing a user to create group	CD
confirmationRateSuccessed	Phenomenon	The rating of movie is confirmed	PD-MovieRate

confirmationRateFaild	Phenomenon	The rating of movie is failed	PD-MovieRate
Comment	Attribute	Comment on the Movie from the users can be done	Class diagram MovieOverviewShow
D			
Discussion	Phenomenon	Forum	Requirement and Domain
DeleteGroup	Phenomenon	A group will be deleted if the administrator leaves it, or if it contains just one member	CD
DataBaseMovies	Lexical Domain, designed Domain	Data base contains movies list with an average rating for each movie	CD, PD-AddMovieInDatabase PD-MovieOverviewShow
DataBaseInfo	Lexical Domain, designed Domain	Data base contains every user information	CD
DBM!		Abbreviation for DataBaseMovie	CD, PD-AddMovieInDatabase
DBF!		Abbreviation for DataBaseInfo	CD
doGet	Technical phenomena	Function between Apache Tomcat and MRA to forward the request of UserWebBrowser and RegisterWebBrowser to the machine	TCD
doPost	Technical phenomena	Function between Apache Tomcat and MRA to forward the request of UserWebBrowser and RegisterWebBrowser to the machine	TCD
Date	Class	New data type in class model of Addmovie	Class Diagram AddMovie
Date	Class	New data type in class model of Addmovie	Class Diagram RateMovie

Day	Attribute	The day the Movie published on	Class Diagram AddMovie
Day	Attribute	The day the Movie published on	Class Diagram RateMovie and MovieOverviewShow
Director	Attribute	Director of a movie	Class model
DefaultWebpage	State	Indicates the starting page	State Machine UserGUI
E			
E-mail	Attribute	E-mail of the person who wants to register him/herself	Class diagram
ExecuteQuery	Technical Phenomena	Phenomena between the machine and the Database. In which the machine executes some query in database.	TCD
ExecuteUpdate	Technical Phenomena	Phenomena between the machine and the Database. In which the machine executes Update in database.	TCD
Exists ()	message	Checks if the movie already exists or not in the database	Sd AddMovieinDatabase (D2)
Exists ()	Message	Checks if the movieslist exists in the database or not	Sd MovieOverviewShow (D2)
ExecuteUpdate ()	message	Java API function to send an SQL update command to MySQL Database	Sd UserRegister (D2) Sd MovieRate (D2) Sd AddMovieinDatabase (D2) Sd MovieOverviewShow (D2)
F			
feedbackU	Phenomenon	MovieRateApp send the feedback to the user	CD

FailRegistered	Phenomenon	The Registration has failed	PD-UserRegister
G			
G		Abbreviation for Group	CD
Group	Lexical Domain, designed Domain	A chat which users can discuss about movies	CD
Get_Registered	Message	Return the already registered users	SD UserRegister
Get_AddedMovies	Message	Return the already added movies	SD AddMovieInDatabase
Get rated	Message	Return the already existent rated movies	SD MovieRate
Get_availableMovieList	Message	Return the available Movie List	SD MovieOverviewShow
Get_availableRatedMovieList	Message	Return the available rated Movie List	SD MovieOverviewShow
GUI	Technical phenomena	Presented by RegisterWebBrowser and UserWebBrowser	TCD
H			
HTTP	Technical phenomena	Defined in RFC 2616	TCD
I			
Info_Movies_MovieList_Adapter	Component	Responsible to create and maintain tables for all persistent classes	subArchBrowse, SubArchBook, subArchRest, globalArch
Info_MovieMovieList	Provided Interface	This interface concretizes "Registering" , "AdingMovies",	Composite Structure Diagram

		” RatingMovie”, ” availableRatedMovieList”,” availableMoviesList” phenomena.	
InMMLPort_I	Port	Info_Movies_MovieList database interface port, the application uses this port to communicate with Info_Movies_MovieList database, which contains all users’ and Movies’ information	Composite Structure Diagram
InfoMovieMovieListAdapterPort	Port	Outside Required port	Composite Structure Diagram
InMMLport	port	Info_Movies_MovieList database port, the Info_Movies_MovieList database receives the commands form app through this port	Composite Structure Diagram
Idle	State	Indicates that the server waits for incoming requests	State Machine UserGUI
J			
K			
L			
LeavingGroup	Phenomenon		CD
login	Phenomenon	A user can start a session on web Rate app by log in	CD
logout	Phenomenon	A user can end a session on web Rate app by log out	CD
leaveGroup	Phenomenon	A user can leave a group if he wants	CD
logingIn	Phenomenon	Machine allows user to start a session if the user is registered.	CD

logingOut	Phenomenon	Machine allows user to end a session.	CD
Login	Phenomenon	Users can login	Requirement
Logout	Phenomenon	Users can logout	Requirement
List exist	State predicate	Condition assures that the list exists already	SD MovieOverviewShow
LC_{Person}	Life-Cycle	Life-Cycle for one person	LC
LC_{User}	Life-Cycle	Life-Cycle for one User	LC
$LC_{MovieRateApp}$	Life-Cycle	Combined Life-Cycle (all persons and all users)	LC
M			
MRA		Abbreviation for MovieRateApp	CD
ML		Abbreviation for MovieList	CD, PD-MovieRate PD-MovieOverviewShow
MovieRateApp	machine	The program	Requirement
Misbehave	Phenomenon	Impolite acting	Requirement and Domain
MovieRateApp	Machine		CD
MRA_Register	Machine		PD-UserRegister
MRA_MovieOverviewShow	Machine		PD-MovieOverviewShow
MRA_Rate	Machine		PD-MovieRate
MRA_AddMovie	Machine		PD-AddMovieInDatabase
MRAAM!		Abbreviation for Movie Rating App AddMovie	PD-AddMovieInDatabase

MoiveList	Lexical Domain, designed Domain	A movies' list, where all the movies stored on data base is represented	CD, PD-MovieRate
MRAR!		Abbreviation for Movie Rating App Register	PD-UserRegister PD-MovieRate
MakeAddingMovie	Phenomenon	Adding a new movie request from the webpage to the machine	PD-AddMovieInDatabase
makeRateMovie	Phenomenon	Rating a movie request from the webpageUser to the machine	PD-MovieRate
makeAccessMovieList	Phenomenon	Request to access to the movie list	PD-MovieOverviewShow
MRAMOS!		Abbreviation for Movie Rating App Movie Overview Show	PD-MovieOverviewShow
Movies'eRepresentation	Phenomenon	Represent the movies	PD-MovieOverviewShow
Movie is added	State Predicate	A new film has been added to the data base by a user	SD AddMovieInDatabase
Movie is exist	State Predicate	The film already exists in the movie list so it cannot be added	SD AddMovieInDatabase
Moviedefaultrated	State Predicate	The film is rated by default. i.e., its rating is zero.	SD-MovieRate
Movie is rated	State Predicate	The movie is rated by user in between 1 and 10	SD-MovieRate
Movietitle	Attribute	The title of the movie	Class diagram
MovieActors	Attribute	Actors of the movie	Class diagram
Month	Attribute	Month the Movie Rated on	Class diagram AddMovie

MovieID	Attribute	Each movie has unique ID	Class diagram RateMovie
Month	Attribute	Month the Movie Rated on	Class diagram RateMovie and MovieOverviewShow
Movieslist	Attribute	One movie list contains all movies	Class model
N			
Name is unique	State Predicate	The entered name is unique	SD UserRegister
Name	Attribute	Name of the person who wants to register him/herself	Class diagram
O			
Original publishing date	Attribute	The date of publishing the Movie	Class diagram
P			
P		Abbreviation for Person	CD
Person	Biddable Domain	Someone who has not registered himself / herself yet	CD, PD-UserRegister
Person >= 18	State Predicate	The age of person is at least 18 years old	SD UserRegister
PersonGUI	component	Web interface for Persons	subArchBrowse,SubArchBook, globalArch
PCmdsPort	Port	The App receives commands from PersonGUI through this port	Composite Structure Diagram
PCmds	Provided Interface	It's an Provided Interface responsible for register new user	Composite Structure Diagram

PCmdsPort_I	Port	PersonGUI uses this Port to sends commands to the APP	Composite Structure Diagram
PersonPort	Port	Outside Required port and provided	Composite Structure Diagram
Q			
R			
Registering	Phenomenon	Machine register user and adding the user to database if he satisfies registration condition	CD, PD-UserRegister
Register	Phenomenon	A person can register on web rate app	CD, PD-UserRegister
RateMovie	Phenomenon	A user can rate a movie by giving a number from 0 to 10, 1: bad and 10: Excellent.	CD PD-MovieRate
RatingMovie	Phenomenon	Adding rate of movies to movies stored on database. And if a movie did not get a rate, it will be rated 0 automatically	CD, PD-MovieRate
Rating	Phenomenon	Evaluation	Requirement and Domain
Register	Phenomenon	Registration process	Requirement
Rated	Phenomenon	Showing the Rated movie through movie list	CD, PD-MovieRate
registerNewUser	Phenomenon	to register a new user	PD-UserRegister
Registered	Phenomenon	Act to confirm that a new user is registered	PD-UserRegister
RegisterWebBrowser	Connection Domain	Web Browser used for Registration	TCD

Rate	Attribute	The Rate of a movie	Class Model
Rate	State	Indicates rating a movie takes place	State machine
Registering	State	Indicates registering a new person takes place	State machine
S			
SeeingMovies'List	Phenomenon	Ability to see movies list of other persons	CD
ShowConfirmationAccepted	Phenomenon	to show the confirmation of accepting a new Registered User	PD-UserRegister
showConfirmationRejected	Phenomenon	to show the confirmation of Rejecting a new Registered User	PD-UserRegister
SuccessRegistered	Phenomenon	The Registration is confirmed	PD-UserRegister
ShowAddMovieSucceed	Phenomenon	Adding a new Movie has succeed	PD-AddMovieInDatabase
ShowAddMovieFailed	Phenomenon	Adding a new Movie has failed	PD-AddMovieInDatabase
ShowConfirmationSuccess	Phenomenon	To confirm success of adding new movie or reviewing a movie from the machine to the webpage	PD-AddMovieInDatabase PD-MovieOverviewShow
ShowConfirmationFail	Phenomenon	To confirm the failure of adding new movie from the machine to the webpage	PD-AddMovieInDatabase
showRateSuccessed	Phenomenon	Showing that movie rating has been done successfully	PD-MovieRate
showRateFailed	Phenomenon	Showing that movie rating has been failed	PD-MovieRate

showSuccess	Phenomenon	Showing that movie overview has been done successfully	PD-MovieOverviewShow
SQL DataBase	Causal Domain	Database for users, Movies and MovieList	TCD
SelectDBM-Webpage	State predicate	The existent Movieslist is displayed	Sd MovieOverviewShow (D2)
SelectDBI-Webpage	State predicate	The existed registered users are displayed	Sd UserRegister(D2)
SelectML-Webpage	State predicate	Movie list is displayed	Sd MovieRate(D2)
ShowRegisteringConfirmed	State predicate	Confirmation of registration	Sd UserRegister (D2)
ShowRegisteringFailed	State predicate	Failure of Registration	Sd UserRegister (D2)
ShowRateConfirmed	State predicate	Confirmation of Rating	Sd MovieRate (D2)
ShowRateFailed	State predicate	Failure of rating	Sd MovieRate (D2)
ShowAddingConfirmed	State predicate	Confirmation of adding a movie	Sd AddMovieinDatabase (D2)
ShowAddingFailed	State predicate	Failure of adding a movie	Sd AddMovieinDatabase (D2)
ShowAddingConfirmed	State	Indicates that adding a movie took place	State Machine UserGUI
ShowAddingFailed	State	Indicates that adding a movie did not take place	State Machine UserGUI
SetMovieLists (…)	message	Set the association between the collection of MovieLists objects and DataBaseMovie	Sd MovieOverviewShow intra
SelectML-Webpage	State	The Movie List webpage	State Machine
SelectDBM-Webpage	State	Data base movie webpage	State Machine
ShowAddingConfirmed	State	The adding of movie took place successfully	State Machine

ShowAddingFailed	State	The adding of movie did not take place	State Machine
ShowRateConfirmed	State	The rating of a movie took place successfully	State Machine
ShowRateFailed	State	The rating of a movie did not take place	State Machine
ShowRegisteringConfirmed	State	The Registering took place	State Machine
ShowRegisteringFailed	State	The Registering did not take Place	State Machine
T			
U			
U!		Abbreviation for User	CD, PD-AddMovieInDatabase, PD-MovieRate PD-MovieOverviewShow
User	Biddable domain	A user able to use Movie Rate App	CD, PD-AddMovieInDatabase, PD-MovieRate PD-MovieOverviewShow
UserWebBrowser	Connection Domain	Web Browser used by users	TCD
Username	Attribute	The name of the user	Class Model
UserFirstRate	Function	Returns true In case it was the first rate of a movie from a user, otherwise false	Class Model
UserGUI	component	Web interface for Users	subArchBrowse,SubArchBook, globalArch
UCmdsPort	Port	The App receives commands from UserGUI through this port	Composite Structure Diagram

UCmds	Provided Interface	It's an Provided Interface responsible for register new user	Composite Structure Diagram
UCmdsPort_I	Port	UserGUI uses this Port to sends commands to the APP	Composite Structure Diagram
UserPort	Port	Outside Required port and provided	Composite Structure Diagram
UserFirstRate ()	Message	Checks if this is the first rate of the user or not	Sd MovieRate(D2)
V			
W			
WebpageUser	Connection domain	The connection domain between the user and the machine. Forwarding the inputs of the machine to the user	PD-AddMovieInDatabase, PD-MovieRate, PD-MovieOverviewShow
WU!		Abbreviation of Webpage User	PD-AddMovieInDatabase PD-MovieRate PD-MovieOverviewShow
WebpageRegister	Connection domain	The connection domain between the person and the machine. Forwarding the inputs of the machine to the person	PD-UserRegister
WR!		Abbreviation for WebpageRegister	PD
X			
Y			
year	Attribute	The year the movie Published in	Class diagram AddMovie
year	Attribute	The year the movie Published in	Class diagram RateMovie and MovieOverviewShow

z			