UNIVERSITÄT DUISBURG ESSEN

Open-Minded

Lab for Software Engineering
Winter term 2015/2016

Lab for Software Engineering

Movie Rating App

M. Amir Al Sawah (3086199)
Daniella Deeb (3099596)
Md Nasirul Islam (3098699)
Can Malatyalioglu (3088210)
Evgeny Evgenyevich Fink (2278213)
Oluwadamilare Israel Adegun(3090829)

November 9, 2021
Language: English

University of Duisburg-Essen · Faculty of Engineering Working group Software Engineering – Prof. Dr. M. Heisel
Date: Mo. 10-12- Supervisor: Roman Wirtz

Contents

List of Figures

1. Analysis

1.1. A1

1.1.1. A1.1 Requirements & Domain Knowledge

Requirements

R1: To use the web application, a person has to register first.

R2: During the registration process, he/she must provide an email address, his/her age and a username.

R3: The username has to be unique.

R4: To register, a person must be at least eighteen years old, otherwise the registration fails.

R5: After the registration, the user can log in using his/her email address and username to use the functionality of the app and log out if he/she wants to end the session.

R6: A logged-in user can add movies he/she has watched from a database to his/her list and rate them.

R7: A rating consists of a mandatory point rating from one to ten (1 = very bad movie, 10 = excellent movie) and optionally a written comment.

R8: If a value differing from one to ten is entered, the rating process will fail.

R9: Movies without any rating are rated as zero.

R10: If a movie is not yet contained in the database, the user can add it by providing the title, director, main actors (at least one) and original publishing date.

R11: Each movie can be contained only once in the database.

R12: A user cannot give more than one rating per movie.

R13: Additionally, users can access a list of all movies in the database sorted by rating in descending order.

R14: For each movie in the database, the average rating is calculated and shown together with the comments.

R15: Registered users can add other registered users into a movie discussion group.

R16: A member of the group can leave it if he/she wants to.

R17: Within a group, members can see the movie lists of other members and can have a discussion in the form of a group chat.

R18: In the chat, the messages are sorted by the order of their creation.

R19: The creator of the group is its administrator.

R20: The administrator can ban members from the group if he/she thinks that the member is misbehaving.

R21: When the administrator leaves the group, the group is deleted.

R22: If a group consists only of one member, an automated method will delete it after a certain amount of time.

Facts

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

F2: A group has a unique name (e.g. “What is the best film of an actor xy?”) and a list of group members.

F3: A chat message contains the username of its creator, a time stamp and the content.

Assumptions

A1: A web application is suitable to be used on different platforms, including mobile devices.

A2: Users only add new movies that really exist and movie data that is valid.

A3: Users only rate movies they have really watched.

A4: Users' rating is based only on their own opinions.

A5: The administrator’s decisions are always fair.

A6: In groups, members only discuss movie related topics.

1.1.2. Context Diagram

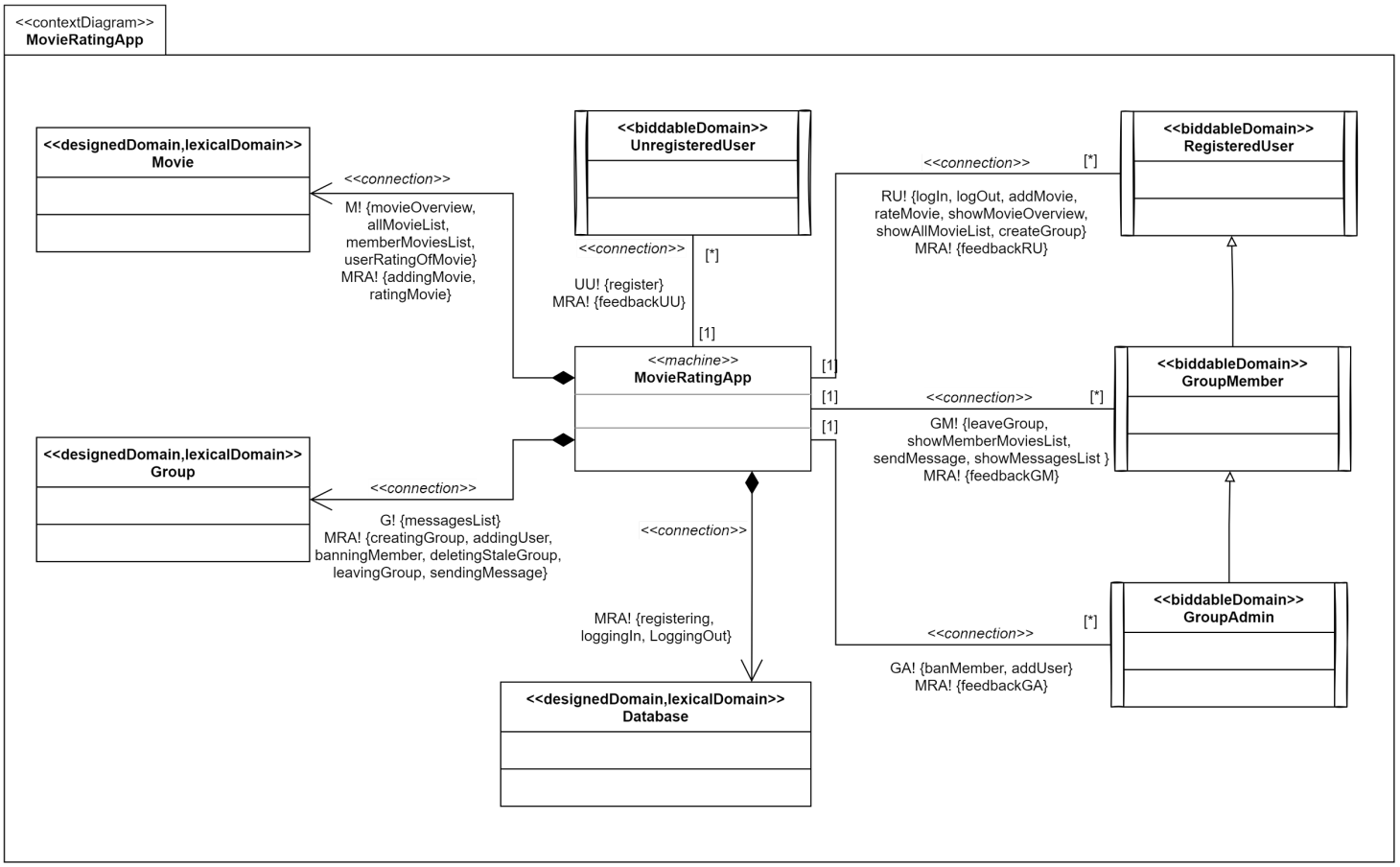


Figure 1.1: Context Diagram

Validation

- 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
logIn	can log in using his/her email address and username	phenomenon
logOut	can log out if he/she wants to end the session	phenomenon
addMovie	can add a movie if a movie is not yet contained in the database	phenomenon
rateMovie	can rate the movies in the database and add to the watched list	phenomenon
showMovieOverview	can show overview of the movie in the database	phenomenon
showAllMovieList	can access a list of all movies in the database	phenomenon
createGroup	can create a movie discussion group	phenomenon
addUser	can add other registered users into a movie discussion group	phenomenon
feedbackRU	introduced to provide feedback to Registered User	phenomenon
leaveGroup	can leave the group	phenomenon
showMemberMoviesList	can see the movie lists of other members	phenomenon
sendMessage	user can send message in the group chat	phenomenon
showMessagesList	user can see messages list in the group chat	phenomenon
feedbackGM	introduced to provide feedback to group Member	phenomenon
banMember	can ban members from the group	phenomenon
feedbackGA	introduced to provide feedback to group Administrator	phenomenon
registering	counterpart to register	phenomenon
loggingIn	counterpart to logIn	phenomenon
loggingOut	counterpart to logOut	phenomenon

creatingGroup	counterpart to createGroup	phenomenon
addingUser	counterpart to addUser	phenomenon
banningMember	counterpart to banMember	phenomenon
deletingStaleGroup	can delete a group with one member after a certain amount of time	phenomenon
leavingGroup	counterpart to leaveGroup	phenomenon
sendingMessage	counterpart to sendMessage	phenomenon
messagesList	show message list	phenomenon
addingMovie	counterpart to addMovie	phenomenon
ratingMovie	counterpart to rateMovie	phenomenon
movieOverview	counterpart to showMovieOverview	phenomenon
allMovieList	show all movies list	phenomenon
memberMoviesList	show member message list	phenomenon
register	can register to the app	phenomenon
feedbackUU	introduced to provide feedback to Unregistered User	phenomenon
MovieRatingApp	the software we are going to build	domain
UnregisteredUser	user who has not registered to the app	domain
RegisteredUser	user who has registered to the app	domain
GroupMember	user who is also a member of a movie discussion group	domain
GroupAdmin	group member who is also the admin of the movie discussion group	domain
Database	Database which holds necessary data for user registration	domain
Group	movie discussion group	domain
Movie	movie in the database of MovieRatingApp	domain

- There must be exactly one Context Diagram.

  Only one context diagram is provided.

- A context diagram has at least one machine domain.

Domain	Domain Type(s)	Connected Domain(s)	Connected Domain(s) Type(s)
MovieRatingApp	Machine Domain	RegisteredUser	Biddable Domain
		UnregisteredUser	Biddable Domain
		GroupAdmin	Biddable Domain
		GroupMember	Biddable Domain
		Group	Designed domain, lexical domain
		DataBase	Designed domain, lexical domain
		Movie	Designed domain, lexical domain
Movie	Designed domain, lexical domain	MovieRatingApp	Machine domain
Group	Designed domain, lexical domain	MovieRatingApp	Machine domain
Database	Designed domain, lexical domain	MovieRatingApp	Machine Domain
GroupAdmin	Biddable Domain	MovieRatingApp	Machine Domain
RegisteredUser	Biddable Domain	MovieRatingApp	Machine Domain
UnregisteredUser	Biddable Domain	MovieRatingApp	Machine Domain
GroupMember	Biddable Domain	MovieRatingApp	Machine Domain

- The machine domain can control at least one interface.

  MovieRatingApp controls several interfaces (feedbackRU, feedbackUU,...)

- 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)
Movie Rating App	Machine Domain	RegisteredUser	Biddable Domain
		UnregisteredUser	Biddable Domain
		GroupAdmin	Biddable Domain
		GroupMember	Biddable Domain
		Group	Designed domain, lexical domain
		DataBase	Designed domain, lexical domain
		Movie	Designed domain, lexical domain
Movie	Designed domain, lexical domain	MovieRatingApp	Machine domain
Group	Designed domain, lexical domain	MovieRatingApp	Machine domain
Database	Designed domain, lexical domain	MovieRatingApp	Machine Domain
GroupAdmin	Biddable Domain	MovieRatingApp	Machine Domain
RegisteredUser	Biddable Domain	MovieRatingApp	Machine Domain
UnregisteredUser	Biddable Domain	MovieRatingApp	Machine Domain
GroupMember	Biddable Domain	MovieRatingApp	Machine Domain

- Causal, designed, lexical, display, machine domain type are not allowed together with biddable domain.

  GroupAdmin, RegisteredUser, UnregisteredUser and GroupMember are biddable domains only.

Domain	Domain Type(s)	Connected Domain(s)	Connected Domain(s) Type(s)
Movie Rating App	Machine Domain	RegisteredUser	Biddable Domain
		UnregisteredUser	Biddable Domain
		GroupAdmin	Biddable Domain
		GroupMember	Biddable Domain

		Group	Design domain, lexical domain
		DataBase	Design domain, lexical domain
		Movie	Design domain, lexical domain
Movie	Design domain, lexical domain	MovieRatingApp	Machine domain
Group	Design domain, lexical domain	MovieRatingApp	Machine domain
Database	Design domain, lexical domain	MovieRatingApp	Machine Domain
GroupAdmin	Biddable Domain	MovieRatingApp	Machine Domain
RegisteredUser	Biddable Domain	MovieRatingApp	Machine Domain
UnregisteredUser	Biddable Domain	MovieRatingApp	Machine Domain
GroupMember	Biddable Domain	MovieRatingApp	Machine Domain

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

	Biddable domain phenomena	**counterpart**
Unregistered User	register	registering
Registered User	logIn	loggingIn
	logOut	LoggingOut
	addMovie	addingMovie
	rateMovie	ratingMovie
	showMovieOverview	MovieOverview
	showAllMovieList	allMovieList
	createGroup	creatingGroup
GroupMember	leaveGroup	leavingGroup
	showMemberMoviesList	memberMoviesList
	sendMessage	sendingMessage
	showMessagesList	messagesList

GroupAdmin	banMember	banningMember
	addUser	addingUser

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

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

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

Context diagram contains no connection domain.

1.2. A2 Problem decomposition and classification

From now on, only the following 4 (concretized) Requirements will be considered:

New R1: Register

- To use the web application, a person has to register first. (Old R1)
- During the registration process, he/she must provide an email address, his/her age and a username. (Old R2)
- The username has to be unique. (Old R3)
- To register, a person must be at least eighteen years old, otherwise the registration fails. (Old R4)

New R2: AddMovie

- If a movie is not yet contained in the database, the user can add it by providing the title, director, main actors (at least one) and original publishing date. (Old R10)
- Each movie can be contained only once in the database. (Old R11)

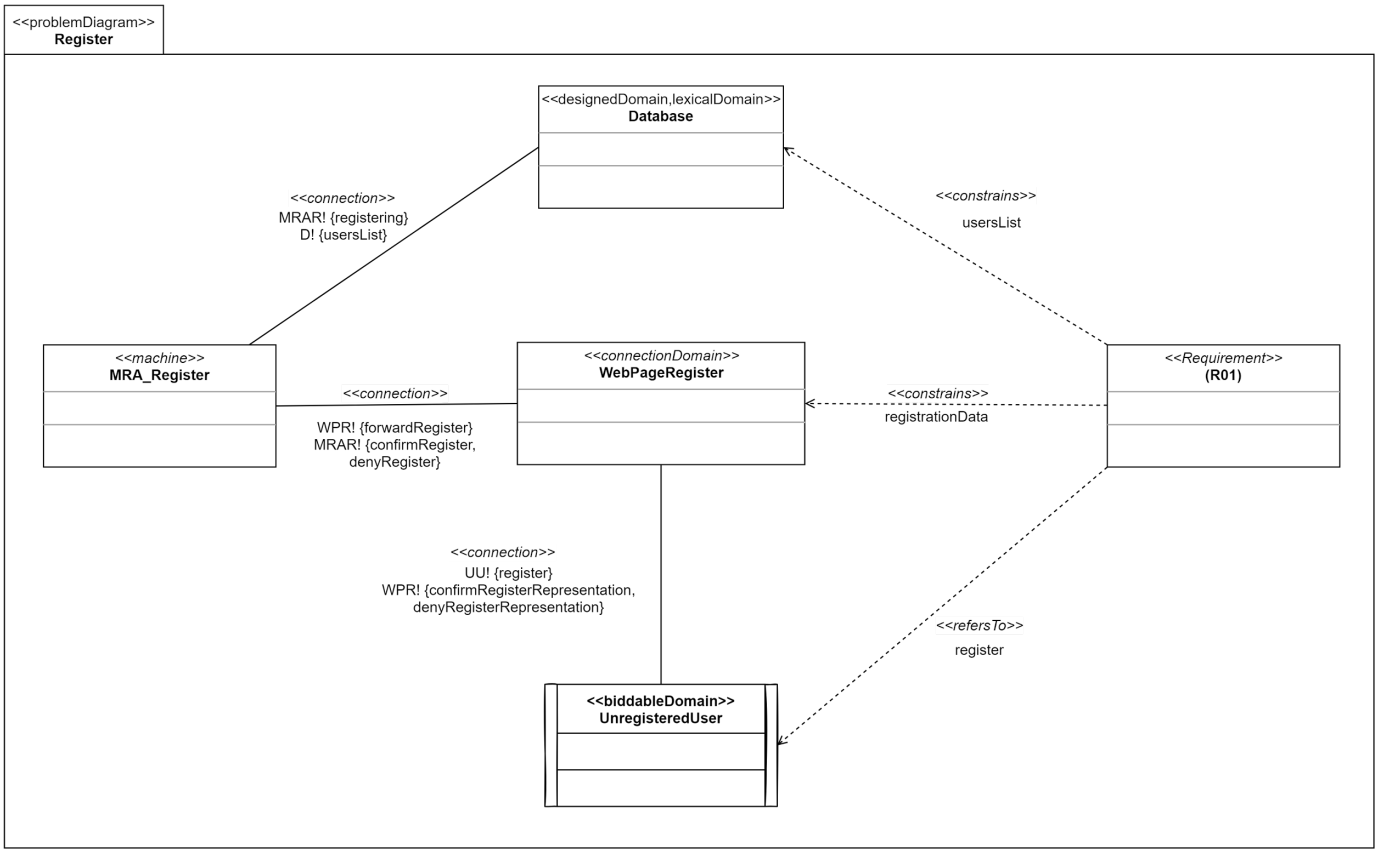
New R3: RateMovie

- A logged-in user can add movies he/she has watched from a database to his/her list and rate them. (Old R6)
- A rating consists of a mandatory point rating from one to ten (1 = very bad movie, 10 = excellent movie) and optionally a written comment. (Old R7)
- If a value differing from one to ten is entered, the rating process will fail. (Old R8)
- A user cannot give more than one rating per movie. (Old R12)

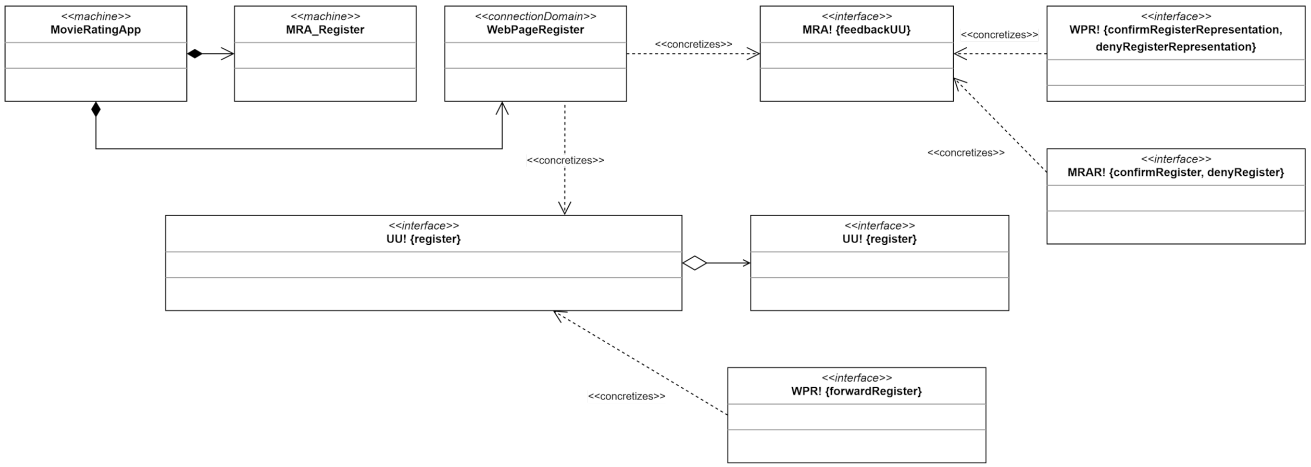
New R4: MovieOverview

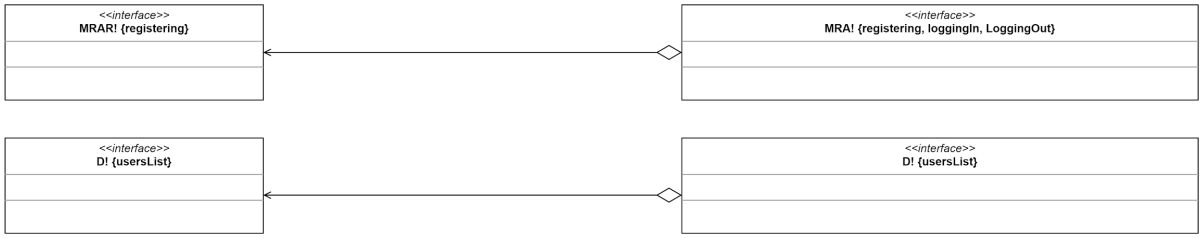
- Movies without any rating are rated as zero. (Old R9)
- For each movie in the database, the average rating is calculated and shown together with the comments. (Old R14)
- Additionally, users can access a list of all movies in the database sorted by rating in descending order. (Old R13)

R1: Problem Diagram (Register)

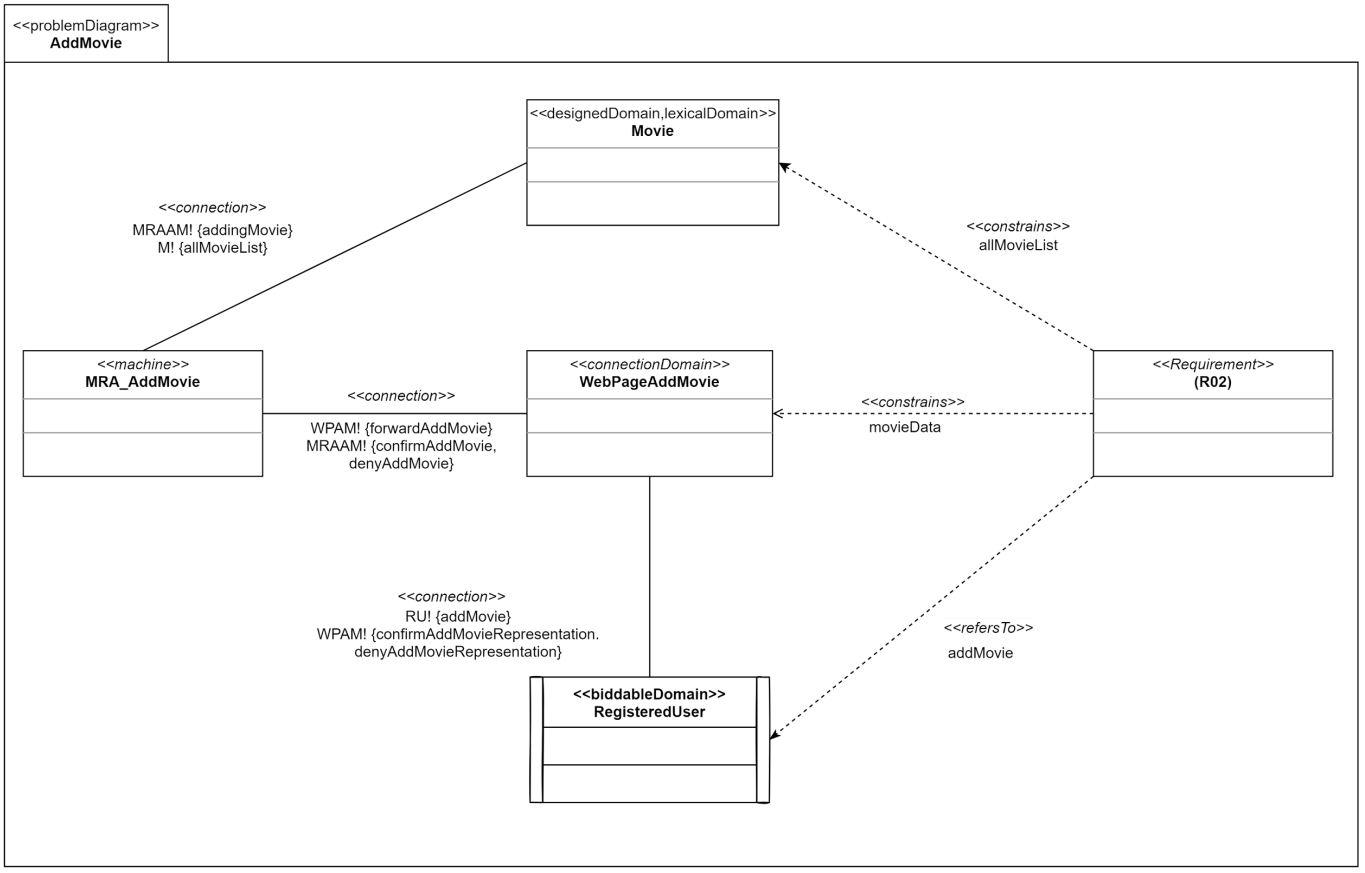


Mapping: R1

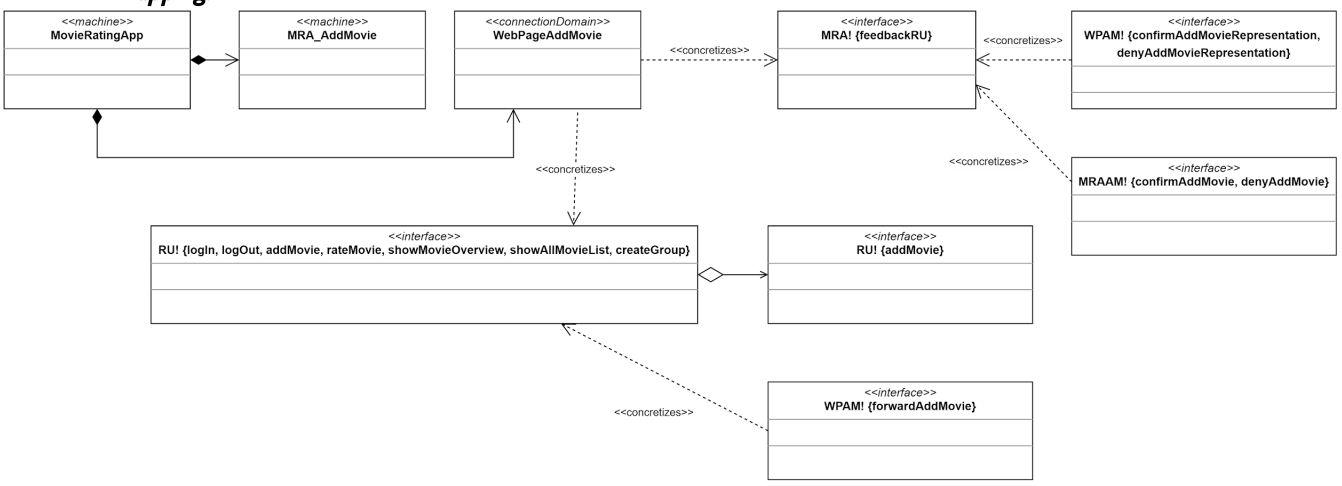


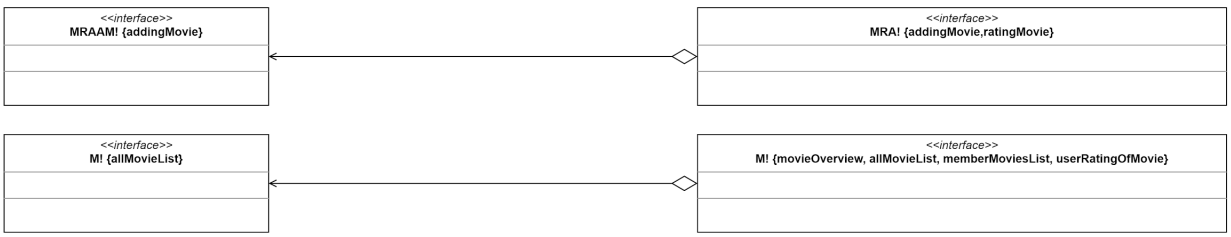


R2 Problem Diagram (AddMovie)

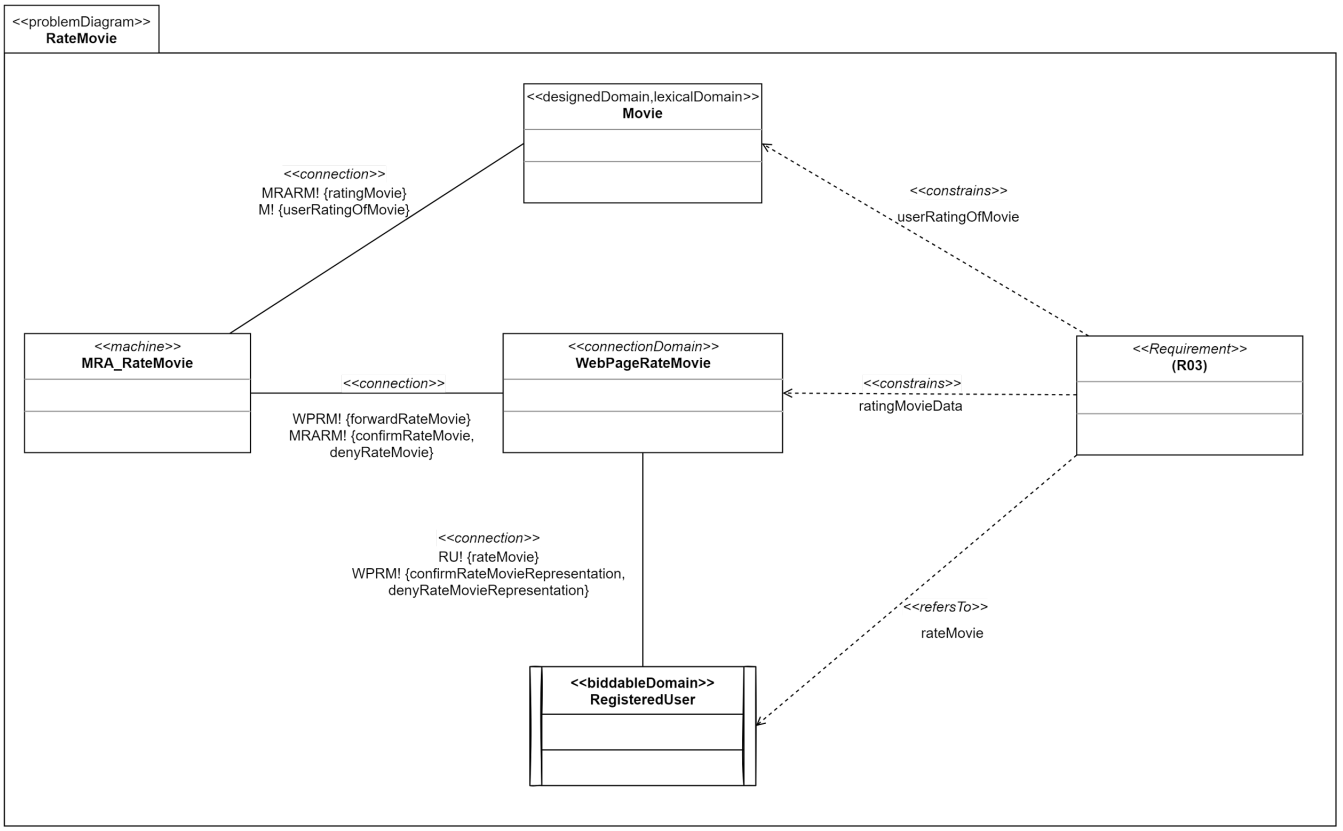


Mapping: R2

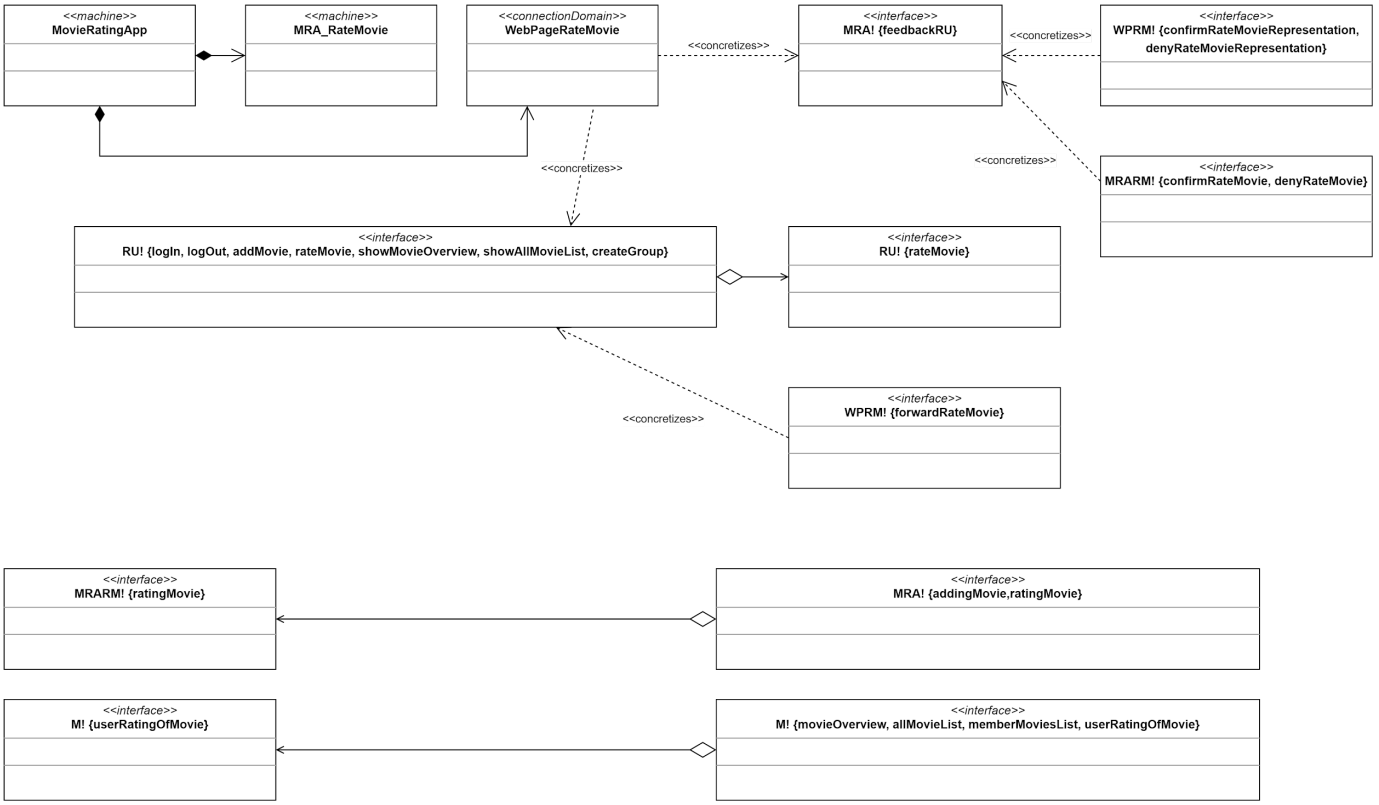




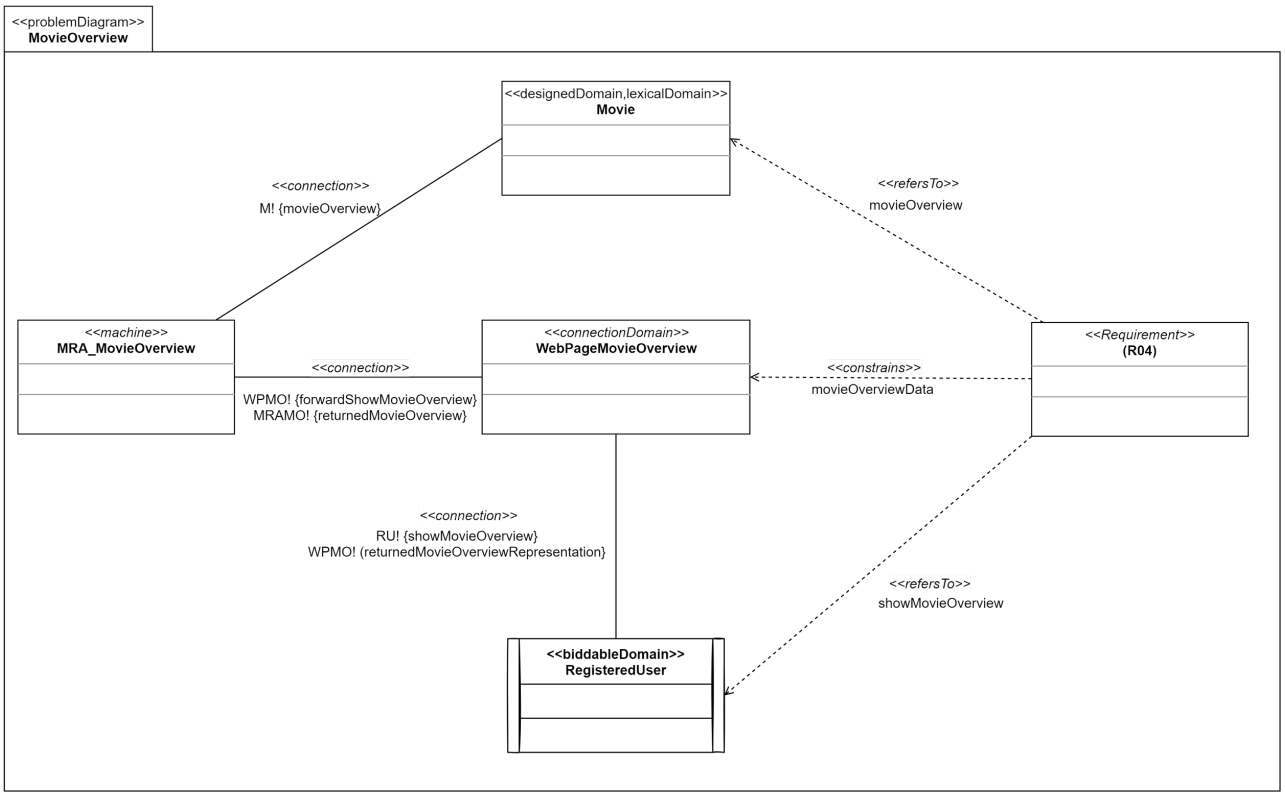
R3 Problem Diagram (RateMovie)



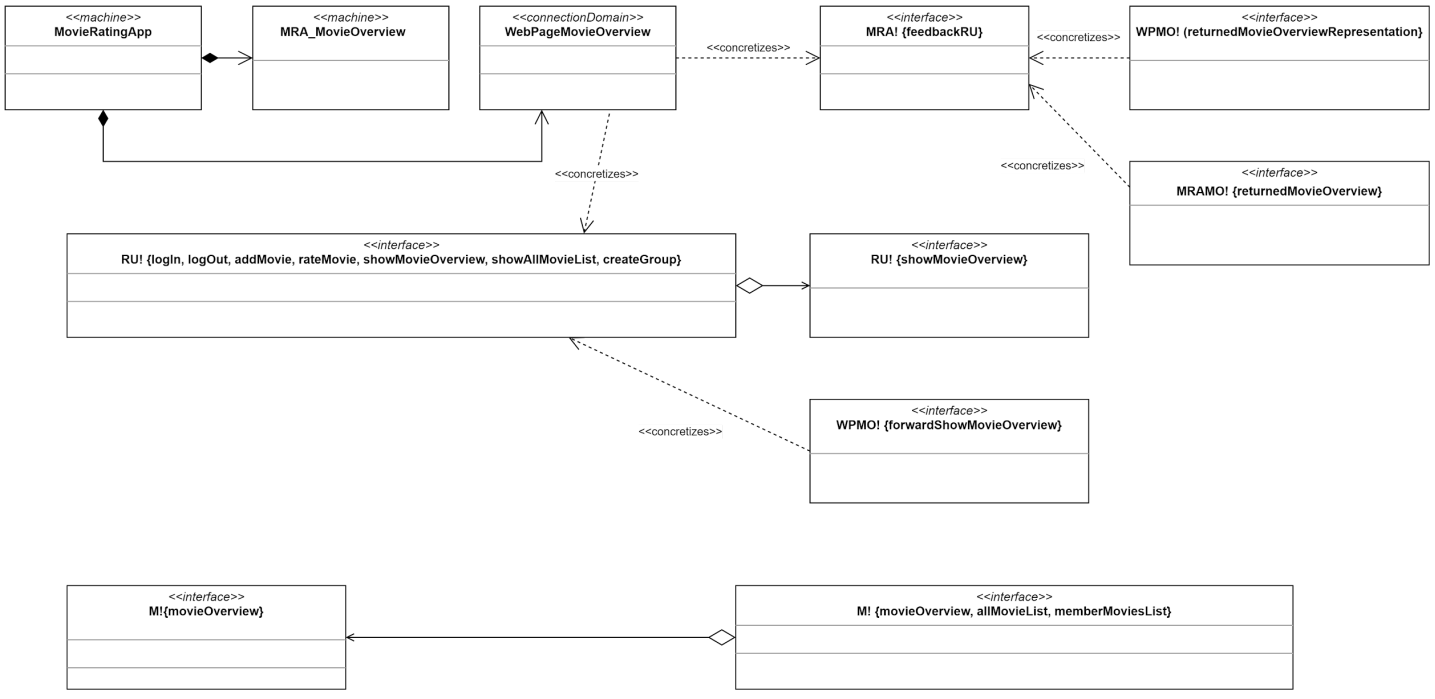
Mapping: R3



R4 Problem Diagram (MovieOverview)



Mapping: R4



Validation

- All requirements R are covered in some subproblem.

Requirement	Covered in	Contained Domain	Domain Type	Constrained	Controlled Phenomena
R1	pdRegister	MRA_Register	Machine		registering, confirmRegister, denyRegister
		Database	lexical, designed	x	
		WebPageRegister	connection	x	forwardRegister, confirmRegisterRepresentation, denyRegisterRepresentation
		UnregisteredUser	Biddable		register
R2	pdAddMovie	MRA_Register	Machine		addingMovie, confirmAddMovie, denyAddMovie
		Movie	lexical, designed	x	
		WebPageAddMovie	connection	x	forwardAddMovie, confirmAddMovieRepresentation, denyAddMovieRepresentation
		RegisteredUser	Biddable		addMovie
R3	pdRateMovie	MRA_Register	Machine		ratingMovie, confirmRateMovie, denyRateMovie
		Movie	lexical, designed	x	
		WebPageRateMovie	connection	x	forwardRateMovie, confirmRateMovieRepresentation, denyRateMovieRepresentation
		RegisteredUser	Biddable		rateMovie
R4	pdMovieOverview	MRA_Register	Machine		returnedMovieOverview
		Movie	lexical, designed		movieOverview
		WebPageMovieOverview	connection	x	forwardShowMovieOverview, returnedMovieOverviewRepresentation
		RegisteredUser	Biddable		showMovieOverview

- A problem diagram has exactly one machine domain.

Requirement	Covered in	Contained Domain	Domain Type	Constrained	Controlled Phenomena
R1	pdRegister	MRA_Register	Machine		registering, confirmRegister, denyRegister
		Database	lexical, designed	x	
		WebPageRegister	connection	x	forwardRegister, confirmRegisterRepresentation, denyRegisterRepresentation
		UnregisteredUser	Biddable		register
R2	pdAddMovie	MRA_Register	Machine		addingMovie, confirmAddMovie, denyAddMovie
		Movie	lexical, designed	x	
		WebPageAddMovie	connection	x	forwardAddMovie, confirmAddMovieRepresentation, denyAddMovieRepresentation
		RegisteredUser	Biddable		addMovie
R3	pdRateMovie	MRA_Register	Machine		ratingMovie, confirmRateMovie, denyRateMovie
		Movie	lexical, designed	x	
		WebPageRateMovie	connection	x	forwardRateMovie, confirmRateMovieRepresentation, denyRateMovieRepresentation
		RegisteredUser	Biddable		rateMovie
R4	pdMovieOverview	MRA_Register	Machine		returnedMovieOverview
		Movie	lexical, designed		movieOverview
		WebPageMovieOverview	connection	x	forwardShowMovieOverview, returnedMovieOverviewRepresentation
		RegisteredUser	Biddable		showMovieOverview

- A problem diagram contains at least one requirement.

Requirement	Covered in	Contained Domain	Domain Type	Constrained	Controlled Phenomena
R1	pdRegister	MRA_Register	Machine		registering, confirmRegister, denyRegister
		Database	lexical, designed	x	
		WebPageRegister	connection	x	forwardRegister, confirmRegisterRepresentation, denyRegisterRepresentation
		UnregisteredUser	Biddable		register
R2	pdAddMovie	MRA_Register	Machine		addingMovie, confirmAddMovie, denyAddMovie
		Movie	lexical, designed	x	
		WebPageAddMovie	connection	x	forwardAddMovie, confirmAddMovieRepresentation, denyAddMovieRepresentation
		RegisteredUser	Biddable		addMovie
R3	pdRateMovie	MRA_Register	Machine		ratingMovie, confirmRateMovie, denyRateMovie
		Movie	lexical, designed	x	
		WebPageRateMovie	connection	x	forwardRateMovie, confirmRateMovieRepresentation, denyRateMovieRepresentation
		RegisteredUser	Biddable		rateMovie
R4	pdMovieOverview	MRA_Register	Machine		returnedMovieOverview
		Movie	lexical, designed		movieOverview
		WebPageMovieOverview	connection	x	forwardShowMovieOverview, returnedMovieOverviewRepresentation
		RegisteredUser	Biddable		showMovieOverview

- The machine domain must control at least one interface.

Requirement	Covered in	Contained Domain	Domain Type	Constrained	Controlled Phenomena
R1	pdRegister	MRA_Register	Machine		registering, confirmRegister, denyRegister
		Database	lexical, designed	x	
		WebPageRegister	connection	x	forwardRegister, confirmRegisterRepresentation, denyRegisterRepresentation
		UnregisteredUser	Biddable		register
R2	pdAddMovie	MRA_Register	Machine		addingMovie, confirmAddMovie, denyAddMovie
		Movie	lexical, designed	x	
		WebPageAddMovie	connection	x	forwardAddMovie, confirmAddMovieRepresentation, denyAddMovieRepresentation
		RegisteredUser	Biddable		addMovie
R3	pdRateMovie	MRA_Register	Machine		ratingMovie, confirmRateMovie, denyRateMovie
		Movie	lexical, designed	x	
		WebPageRateMovie	connection	x	forwardRateMovie, confirmRateMovieRepresentation, denyRateMovieRepresentation
		RegisteredUser	Biddable		rateMovie
R4	pdMovieOverview	MRA_Register	Machine		returnedMovieOverview
		Movie	lexical, designed		movieOverview
		WebPageMovieOverview	connection	x	forwardShowMovieOverview, returnedMovieOverviewRepresentation
		RegisteredUser	Biddable		showMovieOverview

- Requirements constraints at least one domain.

Requirement	Covered in	Contained Domain	Domain Type	Constrained	Controlled Phenomena
R1	pdRegister	MRA_Register	Machine		registering, confirmRegister, denyRegister
		Database	lexical, designed	x	
		WebPageRegister	connection	x	forwardRegister, confirmRegisterRepresentation, denyRegisterRepresentation
		UnregisteredUser	Biddable		register
R2	pdAddMovie	MRA_Register	Machine		addingMovie, confirmAddMovie, denyAddMovie
		Movie	lexical, designed	x	
		WebPageAddMovie	connection	x	forwardAddMovie, confirmAddMovieRepresentation, denyAddMovieRepresentation
		RegisteredUser	Biddable		addMovie
R3	pdRateMovie	MRA_Register	Machine		ratingMovie, confirmRateMovie, denyRateMovie
		Movie	lexical, designed	x	
		WebPageRateMovie	connection	x	forwardRateMovie, confirmRateMovieRepresentation, denyRateMovieRepresentation
		RegisteredUser	Biddable		rateMovie
R4	pdMovieOverview	MRA_Register	Machine		returnedMovieOverview
		Movie	lexical, designed		movieOverview
		WebPageMovieOverview	connection	x	forwardShowMovieOverview, returnedMovieOverviewRepresentation
		RegisteredUser	Biddable		showMovieOverview

- Requirements do not constrain machine(s).

Requirement	Covered in	Contained Domain	Domain Type	Constrained	Controlled Phenomena
R1	pdRegister	MRA_Register	Machine		registering, confirmRegister, denyRegister
		Database	lexical, designed	x	
		WebPageRegister	connection	x	forwardRegister, confirmRegisterRepresentation, denyRegisterRepresentation
		UnregisteredUser	Biddable		register
R2	pdAddMovie	MRA_Register	Machine		addingMovie, confirmAddMovie, denyAddMovie
		Movie	lexical, designed	x	
		WebPageAddMovie	connection	x	forwardAddMovie, confirmAddMovieRepresentation, denyAddMovieRepresentation
		RegisteredUser	Biddable		addMovie
R3	pdRateMovie	MRA_Register	Machine		ratingMovie, confirmRateMovie, denyRateMovie
		Movie	lexical, designed	x	
		WebPageRateMovie	connection	x	forwardRateMovie, confirmRateMovieRepresentation, denyRateMovieRepresentation
		RegisteredUser	Biddable		rateMovie
R4	pdMovieOverview	MRA_Register	Machine		returnedMovieOverview
		Movie	lexical, designed		movieOverview
		WebPageMovieOverview	connection	x	forwardShowMovieOverview, returnedMovieOverviewRepresentation
		RegisteredUser	Biddable		showMovieOverview

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

Requirement	Covered in	Contained Domain	Domain Type	Constrained	Controlled Phenomena
R1	pdRegister	MRA_Register	Machine		registering, confirmRegister, denyRegister
		Database	lexical, designed	x	
		WebPageRegister	connection	x	forwardRegister, confirmRegisterRepresentation, denyRegisterRepresentation
		UnregisteredUser	Biddable		register
R2	pdAddMovie	MRA_Register	Machine		addingMovie, confirmAddMovie, denyAddMovie
		Movie	lexical, designed	x	
		WebPageAddMovie	connection	x	forwardAddMovie, confirmAddMovieRepresentation, denyAddMovieRepresentation
		RegisteredUser	Biddable		addMovie
R3	pdRateMovie	MRA_Register	Machine		ratingMovie, confirmRateMovie, denyRateMovie
		Movie	lexical, designed	x	
		WebPageRateMovie	connection	x	forwardRateMovie, confirmRateMovieRepresentation, denyRateMovieRepresentation
		RegisteredUser	Biddable		rateMovie
R4	pdMovieOverview	MRA_Register	Machine		returnedMovieOverview
		Movie	lexical, designed		movieOverview
		WebPageMovieOverview	connection	x	forwardShowMovieOverview, returnedMovieOverviewRepresentation
		RegisteredUser	Biddable		showMovieOverview

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

Connection Domain	Phenomenon Controlled by Connection Domain	Connected Domain	Phenomenon Controlled by Connected Domain
WebPageRegister	forwardRegister	UnregisteredUser	register
	confirmRegisterRepresentation	MRA_Register	confirmRegister
	denyRegisterRepresentation	MRA_Register	denyRegister
WebPageAddMovie	forwardAddMovie	RegisteredUser	addMovie
	confirmAddMovieRepresentation	MRA_AddMovie	confirmAddMovie
	denyAddMovieRepresentation	MRA_AddMovie	denyAddMovie
WebPageRateMovie	forwardRateMovie	RegisteredUser	rateMovie
	confirmRateMovieRepresentation	MRA_RateMovie	confirmRateMovie
	denyRateMovieRepresentation	MRA_RateMovie	denyRateMovie
WebPageMovieOverview	forwardShowMovieOverview	RegisteredUser	showMovieOverview
	returnedMovieOverviewRepresentation	MRA_MovieOverview	returnedMovieOverview

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

Connection Domain	Phenomenon Controlled by Connection Domain	Connected Domain	Phenomenon Controlled by Connected Domain
WebPageRegister	forwardRegister	UnregisteredUser	register
	confirmRegisterRepresentation	MRA_Register	confirmRegister
	denyRegisterRepresentation	MRA_Register	denyRegister
WebPageAddMovie	forwardAddMovie	RegisteredUser	addMovie
	confirmAddMovieRepresentation	MRA_AddMovie	confirmAddMovie
	denyAddMovieRepresentation	MRA_AddMovie	denyAddMovie
WebPageRateMovie	forwardRateMovie	RegisteredUser	rateMovie
	confirmRateMovieRepresentation	MRA_RateMovie	confirmRateMovie
	denyRateMovieRepresentation	MRA_RateMovie	denyRateMovie
WebPageMovieOverview	forwardShowMovieOverview	RegisteredUser	showMovieOverview
	returnedMovieOverviewRepresentation	MRA_MovieOverview	returnedMovieOverview

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

Connection Domain	Phenomenon Observed by Connection Domain	Phenomenon Controlled by Connection Domain
WebPageRegister	register	forwardRegister
	confirmRegister	confirmRegisterRepresentation
	denyRegister	denyRegisterRepresentation
WebPageAddMovie	addMovie	forwardAddMovie
	confirmAddMovie	confirmAddMovieRepresentation
	denyAddMovie	denyAddMovieRepresentation
WebPageRateMovie	rateMovie	forwardRateMovie
	confirmRateMovie	confirmRateMovieRepresentation
	denyRateMovie	denyRateMovieRepresentation
WebPageMovieOverview	showMovieOverview	forwardShowMovieOverview
	returnedMovieOverview	returnedMovieOverviewRepresentation

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

  Provided mapping diagrams

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

Problem diagram	**operator**	**related domains or phenomena**
pdRegister	leave out domain	RegisteredUser, GroupAdmin, GroupMember, Group, Movie
	introduce connection/display domain	WebPageRegister
	split interface	MRA!{...}
	concretize interface	MRA!{...}, UU!{...}
pdAddMovie	leave out domain	UnregisteredUser, GroupAdmin, GroupMember, Group, Database
	introduce connection/display domain	WebPageAddMovie
	split interface	MRA!{...}, RU!{...}
	concretize interface	MRA!{...}, RU!{...}
pdRateMovie	leave out domain	UnregisteredUser, GroupAdmin, GroupMember, Group, Database
	introduce connection/display domain	WebPageRateMovie
	split interface	MRA!{...}, RU!{...}
	concretize interface	MRA!{...}, RU!{...}
pdMovieOverwiew	leave out domain	UnregisteredUser, GroupAdmin, GroupMember, Group, Database
	introduce connection/display domain	WebPageMovieOverview
	split interface	M!{...}, RU!{...}
	concretize interface	MRA!{...}, RU!{...}

+ Provided mapping diagrams

A2.2 Problem Frames

Subproblem for requirements.

- R1 fits to update2.
- R2 fits to update2.
- R3 fits to update2.
- R4 fits to query2.

Validation

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
Register	update2	MRAR!{registering}	DB!Y1, UM!E2	Machine	LexicalDomain
		MRAR!{confirmRegister, denyRegister} WPR!{forwardRegister}	UM!E4, IOD!E8	Machine	ConnectionDomain
		WPR!{confirmRegisterRepresentation, denyRegisterRepresentation} UU!{Register}	UO!E6, IOD!C7	BiddableDomain	ConnectionDomain
AddMovie	update2	MRAAM!{addingMovie}	DB!Y1, UM!E2	Machine	Lexical Domain
		MRAAM!{confirmAddMovie, denyAddMovie} WPAM!{forwardAddMovie}	UM!E4, IOD!E8	Machine	ConnectionDomain
		WPAM!{confirmAddMovieRepresentation, denyAddMovieRepresentation} RU!{addMovie}	UO!E6, IOD!C7	BiddableDomain	ConnectionDomain
RateMovie	update2	MRARM!{ratingMovie}	DB!Y1, UM!E2	Machine	Lexical Domain
		MRARM!{confirmRateMovie, denyRateMovie} WPRM!{forwardRateMovie}	UM!E4, IOD!E8	Machine	ConnectionDomain
		WPRM!{confirmRateMovieRepresentation, denyRateMovieRepresentation} RU!{rateMovie}	UO!E6, IOD!C7	BiddableDomain	ConnectionDomain
MovieOverview	query2	M!{movieOverview}	DB!Y1	Machine	Lexical Domain
		MRAMO!{returnedMovieOverview} WPMO!{forwardShowMovieOverview}	QM!Y3, IOD!C6	Machine	ConnectionDomain
		WPMO!{returnedMovieOverviewRepresentation} RU!{showMovieOverview}	IOD!E7, EO!E5	BiddableDomain	ConnectionDomain

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

Problem Diagram	Problem Frame	Constrained Domains in PD	Constrained Domains in PF	Domain Type
Register	update2	DataBase	Data Base	Lexical Domain
		WebPageRegister	Input Output Device	Connection Domain
AddMovie	update2	Movie	Data Base	Lexical Domain
		WebPageAddMovie	Input Output Device	Connection Domain
RateMovie	update2	Movie	Data Base	Lexical Domain
		WebPageRateMovie	Input Output Device	Connection Domain
MovieOverview	query2	WebPageMovieOverview	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
Register	update2	UnregisteredUser	Update Operator	Biddable Domain
AddMovie	update2	RegisteredUser	Update Operator	Biddable Domain
RateMovie	update2	RegisteredUser	Update Operator	Biddable Domain
MovieOverview	query2	RegisteredUser	Enquiry Operator	Biddable Domain
		Movie	Data Base	Lexical Domain

All problem diagrams are consistent to their problem frame.

2. A3 Abstract software specification

Sequence Diagram R1

Specifications R1

The following specifications can be derived:

WebPageRegister (S01a) When the WebPageRegister receives the command "*register*", then the command is forwarded to the machine with the command "*forwardRegister*". If the unregistered user is less than 18 years old and username is not unique then Machine will command "*denyRegister*" to WebPageRegister for registration deny and the WebPageRegister will forward this command to the unregistered user by "*denyRegisterRepresentation*". If the unregistered user is more than 18 years old and they have a unique username, then Machine will command "*confirmRegister*" to WebPageRegister for registration success and the WebPageRegister will forward this command "*confirmRegisterRepresentation*" to the unregistered user.

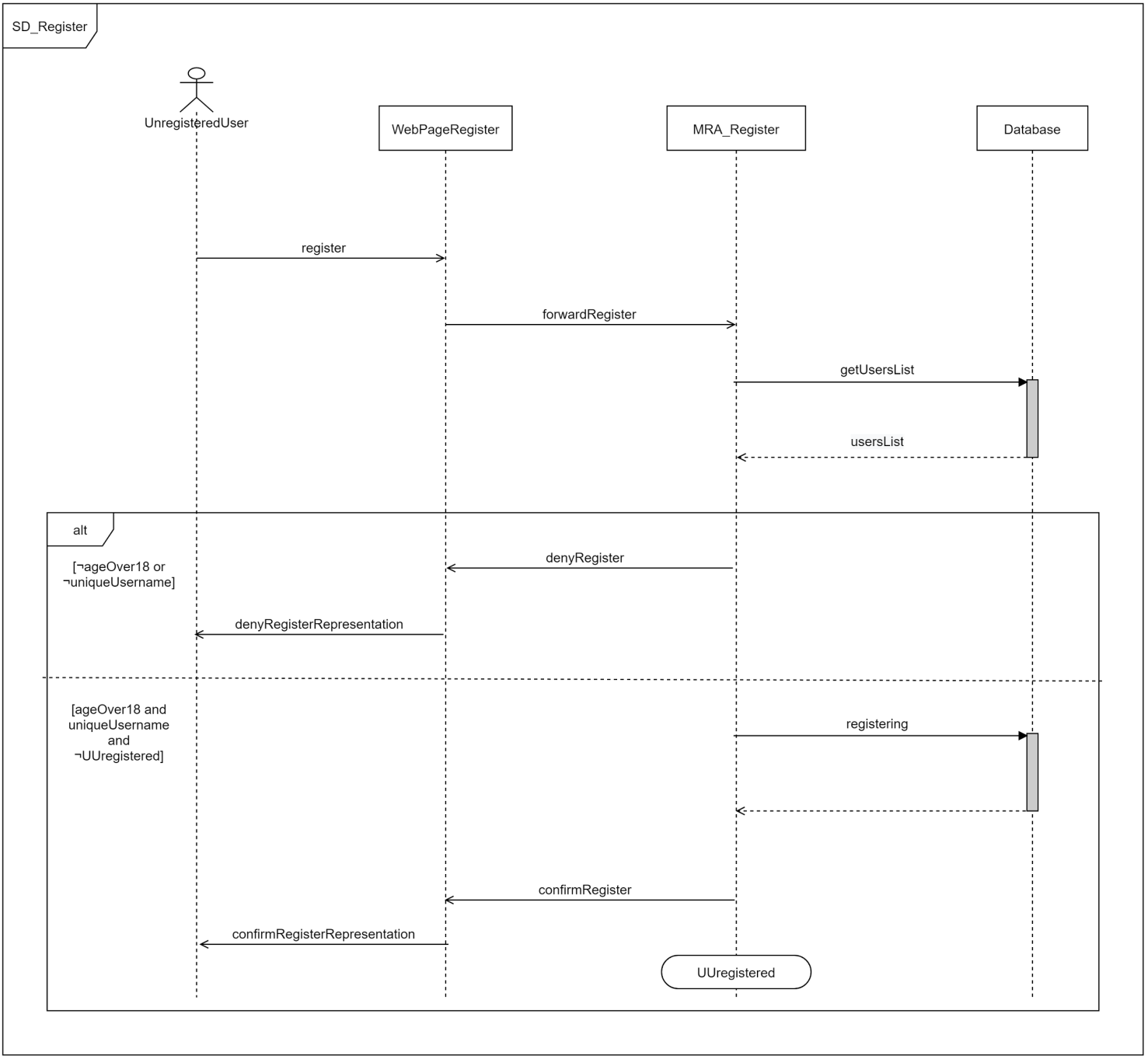
MRA_Register (S01b) When the machine receives the command "*forwardRegister*", the machine will command "*getUsersList*" to send a query to the Database in order to get every users' usernames, which will be received as "*UsersList*", and it will be used to check if the new user's username is unique or not. If the age of the unregistered user is less than 18, or their username is not unique, then the machine will deny the registration request with the message "*denyRegister*". If the age of the unregistered user is 18 or more and their username is unique, then the registration data of the unregistered user will be stored in the Database with the command "*registering*". The result of the operation is returned via the command "*confirmRegister*", and the unregistered user will be registered.

Database (S01c) When the Database will receive the command "*getUsersList*" then the Database will synchronously provide the users' usernames "*UsersList*" to the machine. If the Database receives the command "*registering*", then the Database will store the data of the newly registered user, and the result of the operation is returned back to the machine MRA_Register.

Cross-Platform Compatibility (A1) A web application is suitable to be used on different platforms, including mobile devices.

Correctness condition: (S01a) $\wedge$ (S01b) $\wedge$ (S01c) $\wedge$ (A1) $\Rightarrow$ (R1)

Sequence Diagram of R1



Remark: -

Sequence Diagram R2

Specifications R2

The following specifications can be derived:

WebPageAddMovie (S02a) When the WebPageAddMovie receives the command "*addMovie*", then the command is forwarded to the machine with the command "*forwardAddMovie*". If the movie already exists in the movie database then the machine will command "*denyAddMovie*" to WebPageAddMovie and then WebPageAddMovie will forward this command to the registered users by "*denyAddMovieRepresentation*". If the movie does not exist in the movie database the machine will command "*confirmAddMovie*" to WebPageAddMovie and then WebPageAddMovie will forward this command to registered users by "*confirmAddMovieRepresentation*".

MRA_AddMovie (S02b) When the machine receives the command "*forwardAddMovie*", the machine will command "*getAllMovieList*" to send a query to the database "Movie" in order to get every users' usernames, which will be received as "allMovieList", and it will be used to check if the new movie already exists in database "Movie". If the movie already exists in the database "Movie", then the machine will deny the request with the message "*denyAddMovie*". If the movie does not already exist, then the new movie's data will be stored in the database "Movie" with the command "*addingMovie*". The result of the operation is returned via the command "*confirmAddMovie*".

Movie (S02c) When the database "Movie" receives the command "*getAllMovieList*" then the database "Movie" will synchronously provide every movie in the movie database with "*allMovieList*" to the machine. If "Movie" receives the command "*addingMovie*", then the database "Movie" will store the data of the newly added movie, and the result of the operation is returned back to the machine MRA_AddMovie.

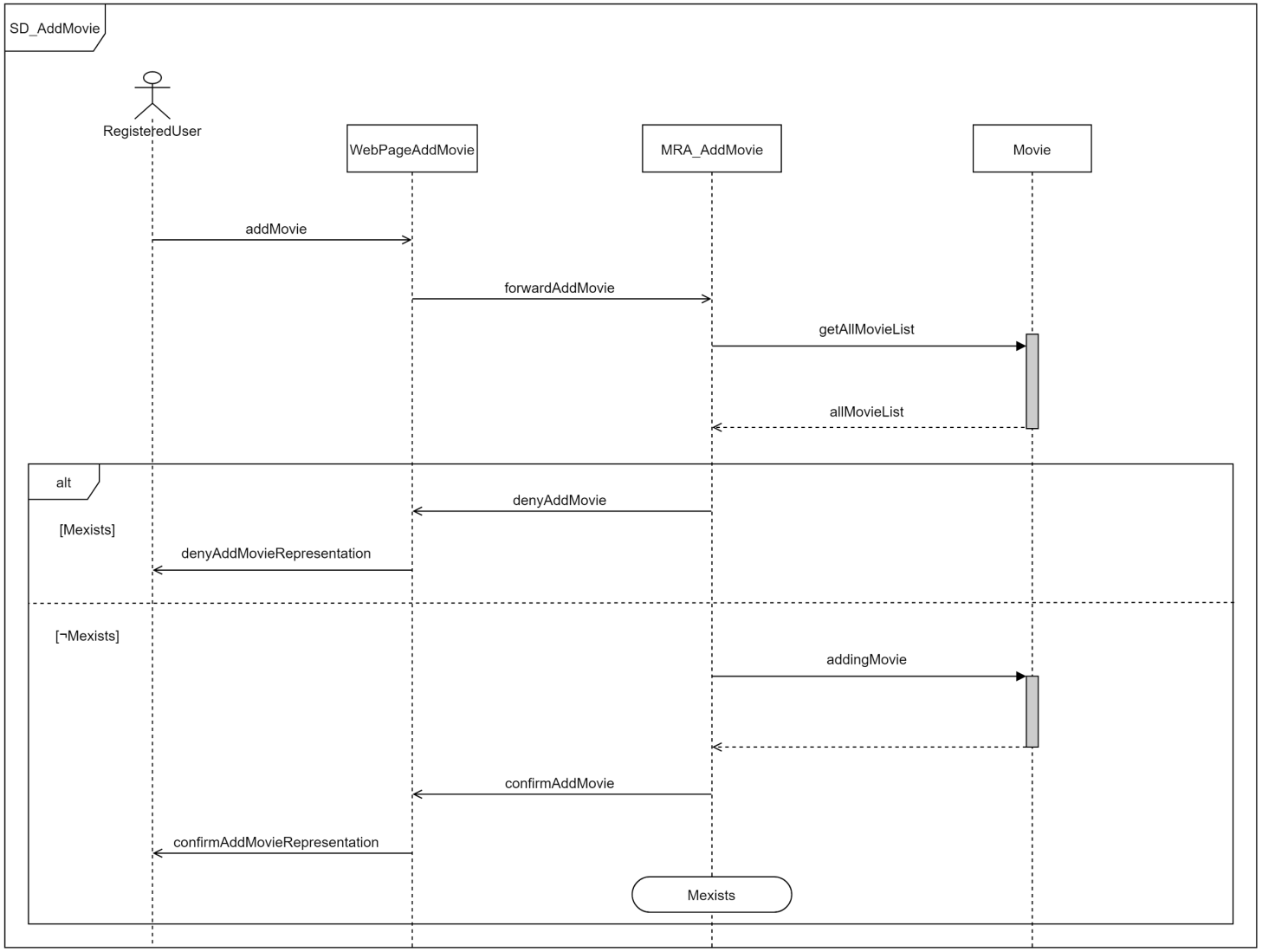
Original Movie (F1) Each movie has a title, a director, at least one main actor and an original publishing date.

Cross-Platform Compatibility (A1) A web application is suitable to be used on different platforms, including mobile devices.

Valid Movies (A2) Users only add new movies that really exist and movie data that is valid.

Correctness condition: (S02a) ∧ (S02b) ∧ (S02c) ∧ (F1) ∧ (A1) ∧ (A2) ⇒ (R2)

Sequence Diagram of R2



Remark: -

Sequence Diagram R3

Specifications R3

The following specifications can be derived:

WebPageRateMovie (S03a) When the WebPageRateMovie receives the command "*rateMovie*", then the command is forwarded to the machine with the command "*forwardRateMovie*". If the movie is already rated by this user then the machine will command "*denyRateMovie*" to WebPageRateMovie and then WebPageRateMovie will forward this command to registered users by "*denyRateMovieRepresentation*". If rating for this movie does not exist then the machine will command "*confirmRateMovie*" to WebPageRateMovie and then WebPageRateMovie will forward this command to registered users by "*confirmRateMovieRepresentation*".

MRA_RateMovie (S03b) When the machine receives the command "*forwardRateMovie*", the machine will command "*getUserRatingOfMovie*" to send a query to the database "Movie" in order to get the rating of the movie that is given by the user, which will be received as "*UserRatingOfMovie*", and it will be used to check if the user has already rated for the movie. If the user has already rated for the movie or the rating is invalid (not a number between one to ten), then the machine will deny the rating request with the message "*denyRateMovie*". If the user has not already rated for the movie and the rating is not invalid, then the new movie's rating data will be stored in the database "Movie" with the command "*ratingMovie*". The result of the operation is returned via the command "*confirmRateMovie*".

Movie (S03c) When the database "Movie" receives the command "*getUserRatingOfMovie*" then the database "Movie" will synchronously provide the rating the user has given to the movie with "*UserRatingOfMovie*" to the machine. If "Movie" receives the command "*ratingMovie*", then the database "Movie" will store the rating data of the newly rated movie, and the result of the operation is returned back to the machine MRA_AddMovie.

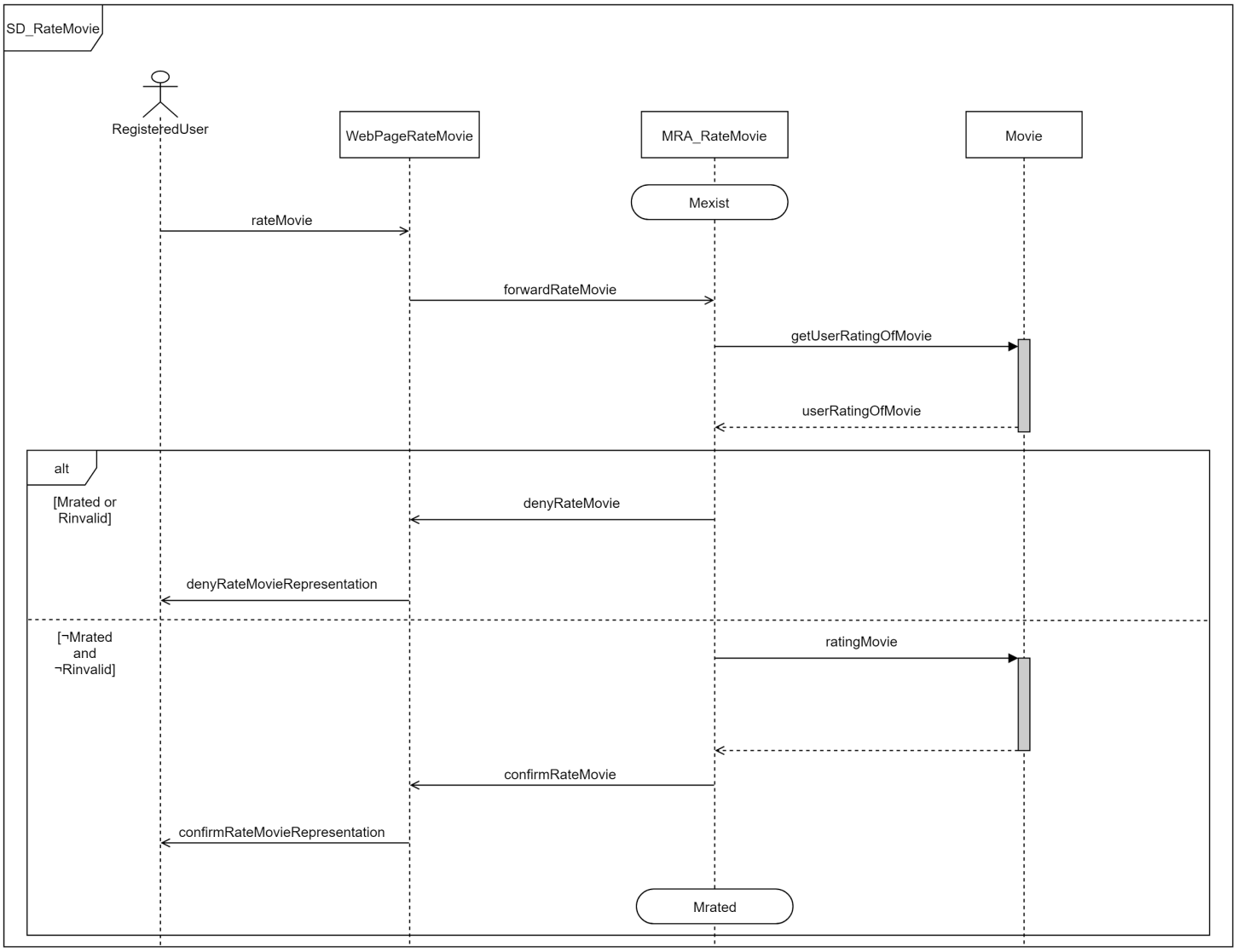
Cross-Platform Compatibility (A1) A web application is suitable to be used on different platforms, including mobile devices.

Really watched Movie (A3) Users only rate movies they have really watched.

Own Opinions (A4) Users' rating is based only on their own opinions.

Correctness condition: (S03a) ∧ (S03b) ∧ (S03c) ∧ (A1) ∧ (A3) ∧ (A4) ⇒ (R3)

Sequence Diagram of R3



Remark: Mrated ⇒ Mexist

Sequence Diagram R4

Specifications R4

The following specifications can be derived:

WebPageMovieOverview (S04a) When the webPageMovieOverview receives the command "*showMovieOverview*", then the command is forwarded to the machine with the command "*forwardShowMovieOverview*". The overview of the movie will be received via the command "*returnedMovieOverview*" from the machine and then webPageMovieOverview will forward this command to the registered users by "*returnedMovieOverviewRepresentation*".

MRA_RateMovie (S04b) When the machine receives the command "*forwardShowMovieOverview*", an overview of the movie is selected with the command "*getMovieOverview*" and received as the data "*movieOverview*". The results are returned via the command "*returnedMovieOverview*".

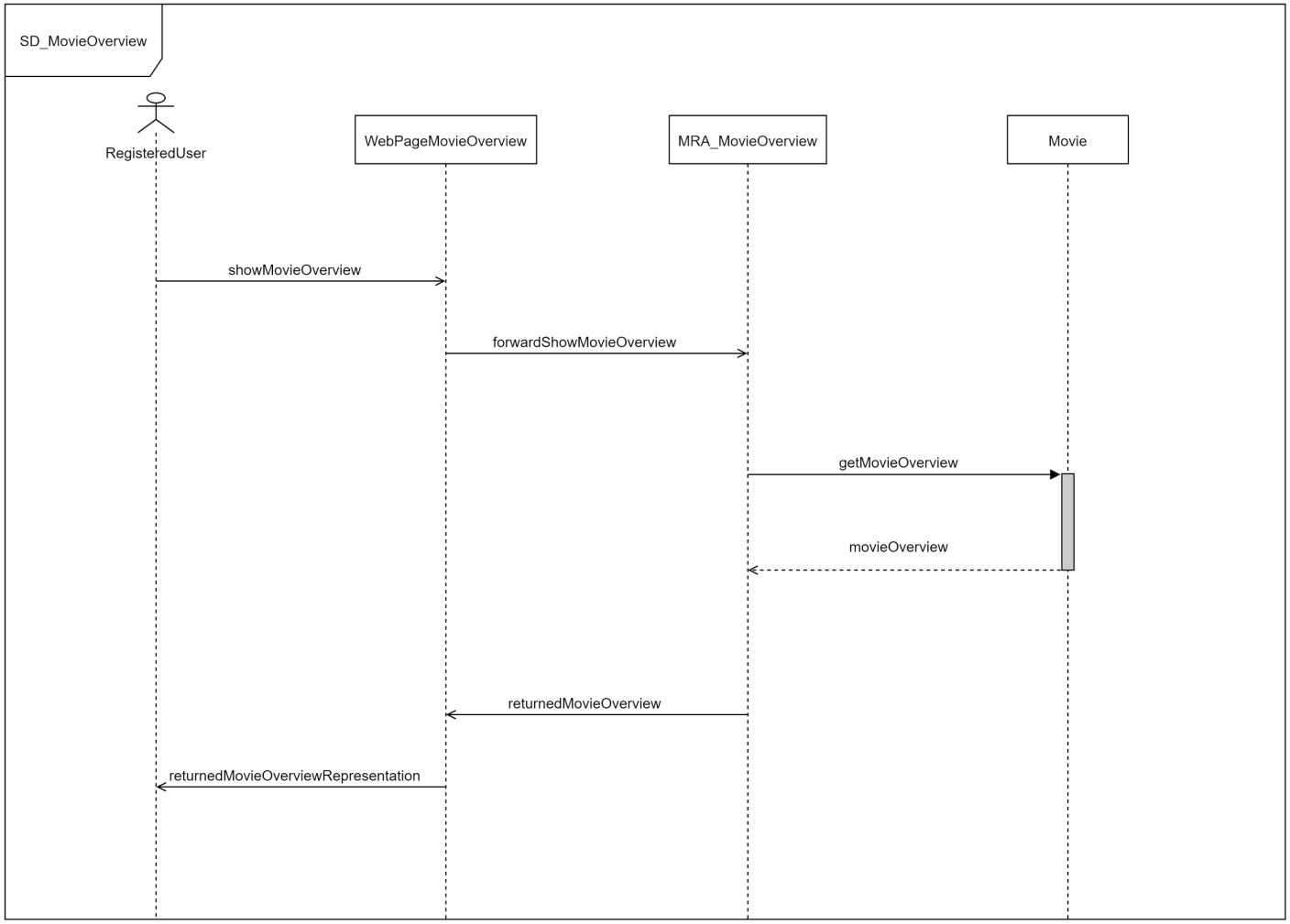
Movie (S04c) After receiving the command "*getMovieOverview*" the results are returned as the data "*movieOverview*".

Original Movie (F1) Each movie has a title, a director, at least one main actor and an original publishing date.

Cross-Platform Compatibility (A1) A web application is suitable to be used on different platforms, including mobile devices.

Correctness condition: (S04a) ∧ (S04b) ∧ (S04c) ∧ (F1) ∧ (A1) ⇒ (R4)

Sequence Diagram of R4



Remark: Result may be empty.

Validation

- $S_{abstract} \wedge D$ are non-concontradictory.
  No contradictions can be found in $S_{abstract} \wedge D$.
- $S_{abstract} \wedge D \Rightarrow R$.

$$(S01a) \wedge (S01b) \wedge (S01c) \wedge (A1) \Rightarrow (R01)$$
$$(S02a) \wedge (S02b) \wedge (S02c) \wedge (F1) \wedge (A1) \wedge (A2) \Rightarrow (R02)$$
$$(S03a) \wedge (S03b) \wedge (S03c) \wedge (A1) \wedge (A3) \wedge (A4) \Rightarrow (R03)$$
$$(S04a) \wedge (S04b) \wedge (S04c) \wedge (F1) \wedge (A1) \Rightarrow (R04)$$

- Messages and phenomena are consistent.

message in scenario	source	target	phenomena in problem diagram
register	UnregisteredUser	WebPageRegister	UU!{register}
forwardRegister	WebPageRegister	MRA_Register	WPR!{forwardRegister}
getUserList	MRA_Register	Database	DB!{usersList}
denyRegister	MRA_Register	WebPageRegister	MRAR!{denyRegister}
denyRegisterRepresentation	WebPageRegister	UnregisteredUser	WPR!{denyRegisterRepresentation}
registering	MRA_Register	Database	MRAR!{registering}
confirmRegister	MRA_Register	WebPageRegister	MRAR!{confirmRegister}
confirmRegisterRepresentation	WebPageRegister	UnregisteredUser	WPR!{confirmRegisterRepresentation}
addMovie	RegisteredUser	WebPageAddMovie	RU!{addMovie}
forwardAddMovie	WebPageAddMovie	MRA_AddMovie	WPAM!{forwardAddMovie}
getUserRatingOfMovie	MRA_AddMovie	Movie	M!{allMovieList}
denyAddMovie	MRA_AddMovie	WebPageAddMovie	MRAAM!{denyAddMovie}
denyAddMovieRepresentation	WebPageAddMovie	RegisteredUser	WPAM!{denyAddMovieRepresentation}
addingMovie	MRA_AddMovie	Movie	MRAAM!{addingMovie}
confirmAddMovie	MRA_AddMovie	WebPageAddMovie	MRAAM!{confirmAddMovie}
confirmAddMovieRepresentation	WebPageAddMovie	RegisteredUser	WPAM!{confirmAddMovieRepresentation}
rateMovie	RegisteredUser	WebPageRateMovie	RU!{rateMovie}
forwardRateMovie	WebPageRateMovie	MRA_RateMovie	WPRM!{forwardRateMovie}
getUserRatingOfMovie	MRA_RateMovie	Movie	M!{userRatingOfMovie}
denyRateMovie	MRA_RateMovie	WebPageRateMovie	MRARM!{denyRateMovie}
denyRateMovieRepresentation	WebPageRateMovie	RegisteredUser	WPRM!{denyRateMovieRepresentation}
ratingMovie	MRA_RateMovie	Movie	MRARM!{ratingMovie}
confirmRateMovie	MRA_RateMovie	WebPageRateMovie	MRARM!{confirmRateMovie}
confirmRateMovieRepresentation	WebPageRateMovie	RegisteredUser	WPRM!{confirmRateMovieRepresentation}
showMovieOverview	RegisteredUser	WebPageMovieOverview	RU!{showMovieOverview}
forwardShowMovieOverview	WebPageMovieOverview	MRA_MovieOverview	WPMO!{forwardShowMovieOverview}
getMovieOverview	MRA_MovieOverview	Movie	M!{movieOverview}
returnedMovieOverview	MRA_MovieOverview	WebPageMovieOverview	MRAMO!{returnedMovieOverview}
returnedMovieOverviewRepresentation	WebPageMovieOverview	RegisteredUser	WPMO!{returnedMovieOverviewRepresentation}

- Lexical domains are not sources of messages.

message in scenario	source	domain type
register	UnregisteredUser	BiddableDomain
forwardRegister	WebPageRegister	ConnectionDomain
getUserList	MRA_Register	Machine
denyRegister	MRA_Register	Machine
denyRegisterRepresentation	WebPageRegister	ConnectionDomain
registering	MRA_Register	Machine
confirmRegister	MRA_Register	Machine
confirmRegisterRepresentation	WebPageRegister	ConnectionDomain
addMovie	RegisteredUser	BiddableDomain
forwardAddMovie	WebPageAddMovie	ConnectionDomain
getUserRatingOfMovie	MRA_AddMovie	Machine
denyAddMovie	MRA_AddMovie	Machine
denyAddMovieRepresentation	WebPageAddMovie	ConnectionDomain
addingMovie	MRA_AddMovie	Machine
confirmAddMovie	MRA_AddMovie	Machine
confirmAddMovieRepresentation	WebPageAddMovie	ConnectionDomain
rateMovie	RegisteredUser	BiddableDomain
forwardRateMovie	WebPageRateMovie	ConnectionDomain
getUserRatingOfMovie	MRA_RateMovie	Machine
denyRateMovie	MRA_RateMovie	Machine
denyRateMovieRepresentation	WebPageRateMovie	ConnectionDomain
ratingMovie	MRA_RateMovie	Machine
confirmRateMovie	MRA_RateMovie	Machine
confirmRateMovieRepresentation	WebPageRateMovie	ConnectionDomain
showMovieOverview	RegisteredUser	BiddableDomain
forwardShowMovieOverview	WebPageMovieOverview	ConnectionDomain
getMovieOverview	MRA_MovieOverview	Machine
returnedMovieOverview	MRA_MovieOverview	Machine
returnedMovieOverviewRepresentation	WebPageMovieOverview	ConnectionDomain

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

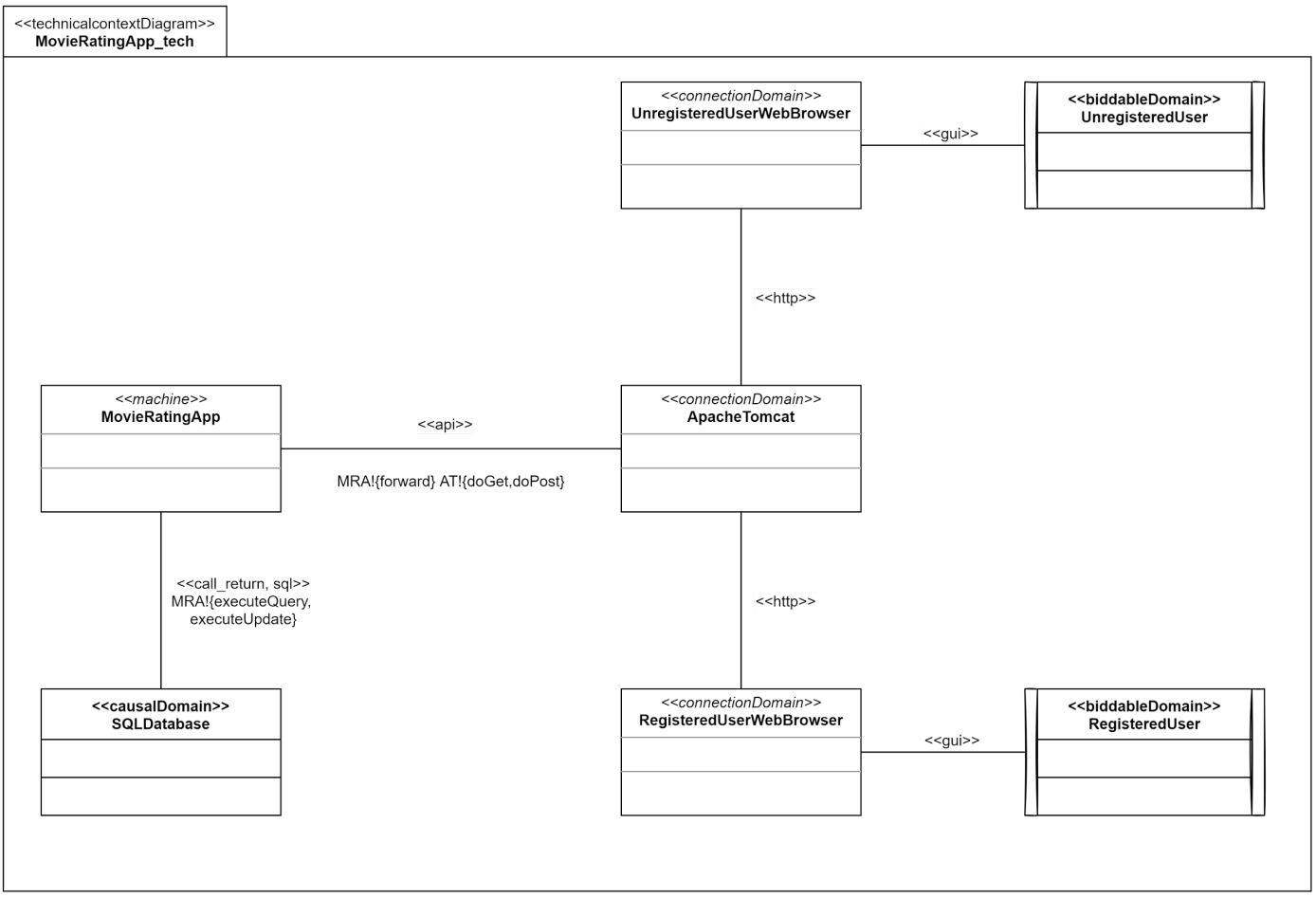
subproblem	normal case	exceptional case
pdRegister	sdRegister	sdRegister
pdAddMovie	sdAddMovie	sdAddMovie
pdRateMovie	sdRateMovie	sdRateMovie
pdMovieOverview	sdMovieOverview	

For pdRegister, pdAddMovie, pdRateMovie an exceptional case should be considered.

3. A4 Technical software specification

3.1. Technical Context Diagram

Technical realization of domains from context and problem diagrams.
Movie and Database: Realized as SQLDatabase on the same computer as the machine. Therefore, the database is connected by a call-and-return interface and used with SQL commands.
WebPageRegister, WebPageAddMovie, WebPageRateMovie and WebPageMovieOverview: Realized using ApacheTomcat and RegisteredUserWebBrowser and UnregisteredUserWebBrowser (browser of User)
We decided to use ApacheTomcat as a server platform, because the user realized other projects on this platform require a Java implementation.



Technical interfaces of the machine:

- SQL Commands: defined in FIPS PUB 127-2,

  (U.S.DEPARTMENT OF COMMERCE/National Institute of

  Standards and Technology, 1993)

  - Operations executeQuery and executeUpdate are defined in interface java.sql.Statement https://docs.oracle.com/javase/8/docs/api/index.html?java/sql/Statement.html

- API for ApacheTomcat (http://tomcat.apache.org/tomcat-9.0-doc/index.html)

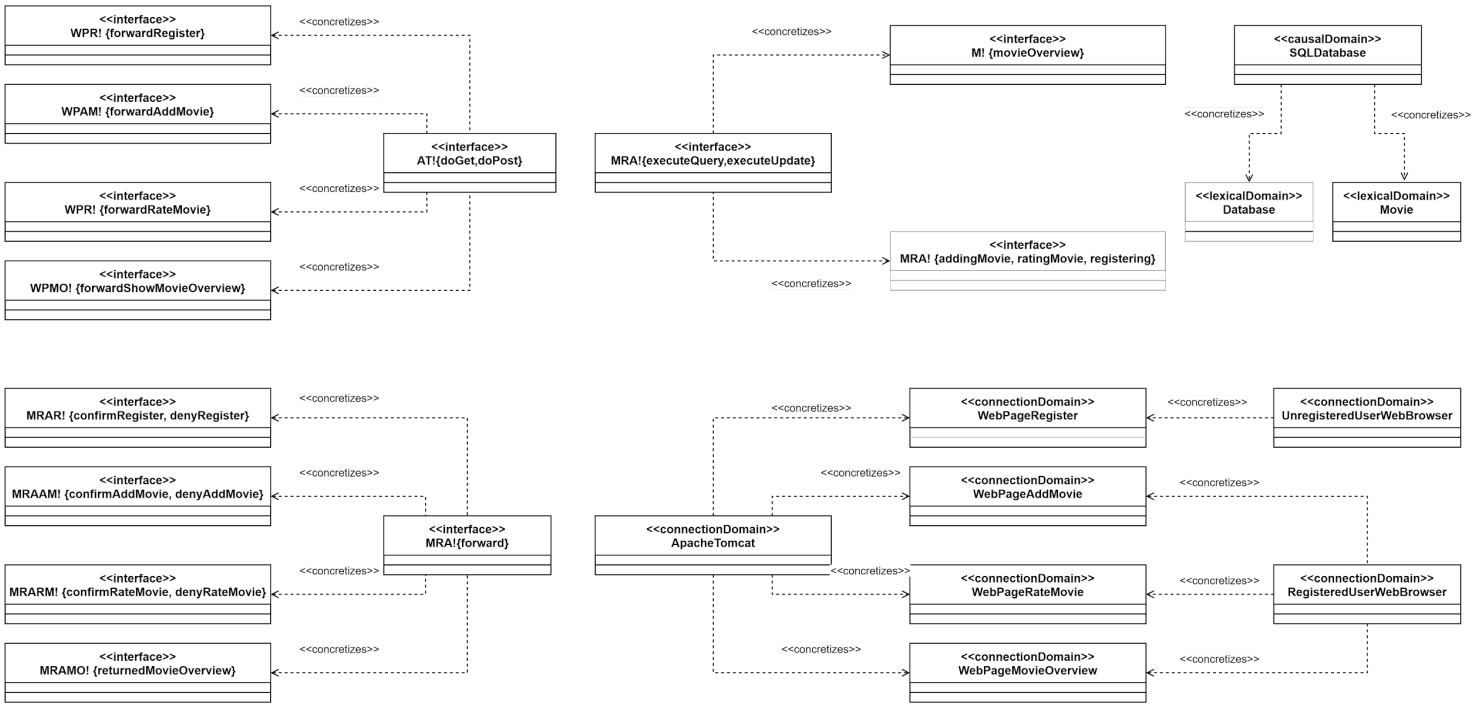
  - Operations doGet and doPost are defined in abstract class javax.servlet.http.HttpServlet (https://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServlet.html)
  - Operation forward defined in interface javax.servlet.RequestDispatcher (http://docs.oracle.com/javaee/7/api/javax/servlet/RequestDispatcher.html)

Technical interfaces in the environment:

- HTTP (Hypertext Transfer Protocol): defined in RFC 2616, (Network Working Group, 1999)

- GUI: User interfaces of MailClient and HTML webpages(dened by https://www.w3.org/TR/html5/) presented by RegisteredUserWebBrowser and UnregisteredUserWebBrowser.

Mapping



Validation for A4

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

Message	new phenomena and domains
register	ApacheTomcat, HTTP
addMovie	ApacheTomcat, HTTP
rateMovie	ApacheTomcat, HTTP
showMovieOverview	ApacheTomcat, HTTP

All internal messages can be realized using SQL commands.

All domains and phenomena of the technical context diagram are related to elements in the problem diagrams or context diagrams.

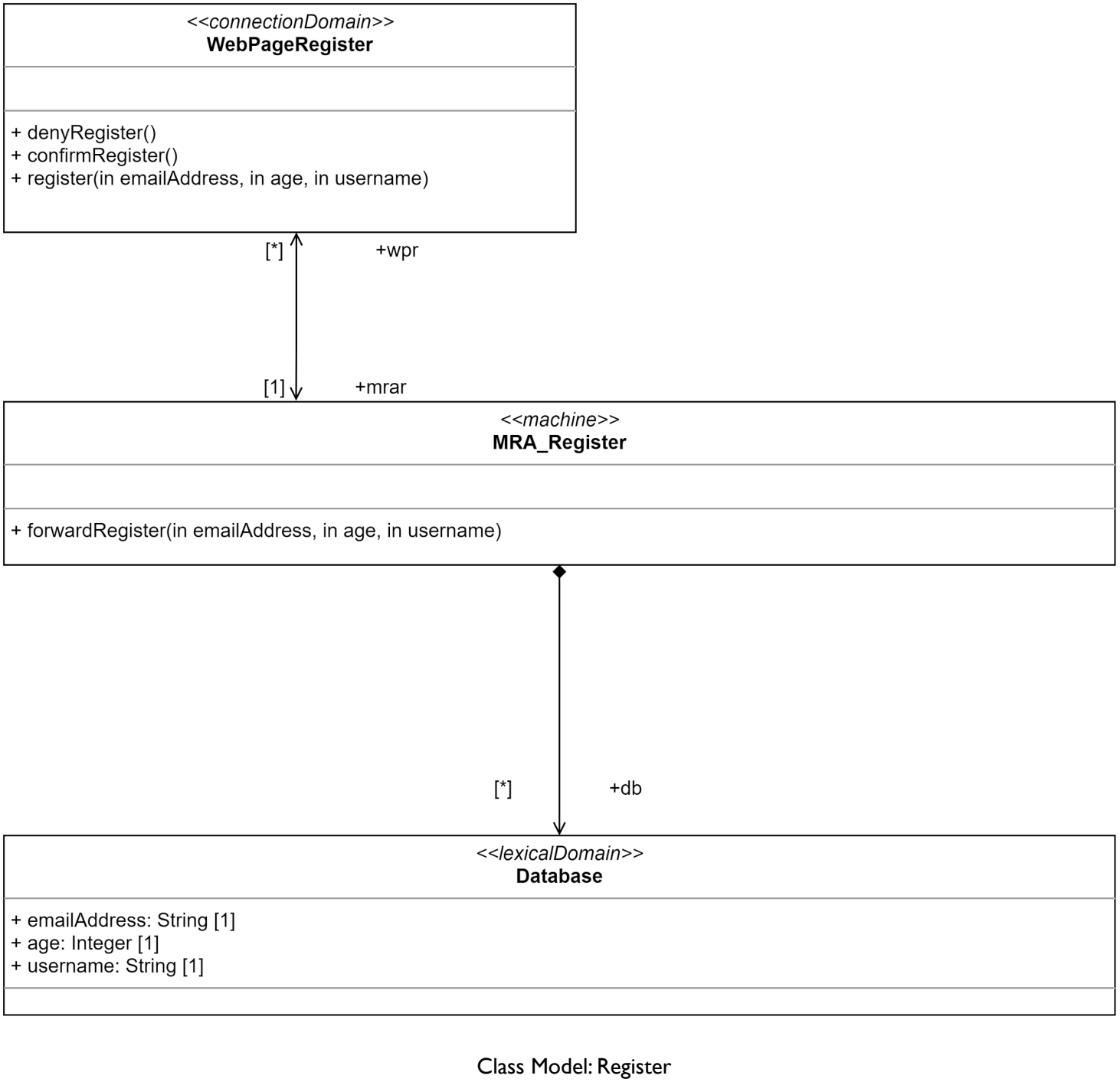
- *Mapping diagrams provided*

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

Problem Diagram	Domain connected with the machine	Element in the TCD
Register	Database	SQLDatabase
	WebPageRegister	UnregisteredUserWebBrowser, ApacheTomcat
AddMovie	Movie	SQLDatabase
	WebPageAddMovie	RegisteredUserWebBrowser, ApacheTomcat
RateMovie	Movie	SQLDatabase
	WebPageRateMovie	RegisteredUserWebBrowser, ApacheTomcat
MovieOverview	Movie	SQLDatabase
	WebPageMovieOverview	RegisteredUserWebBrowser, ApacheTomcat

4. A5 Operations & data specification

4.1. R1: The operation Register



Class Model: Register

Class Diagram R1

Name: forwardRegister
Description: Try to register the user and return confirm or deny.
OCL constraint:

```
context MRA_Register::forwardRegister(emailAddress: String, age:
Integer, username: String)
pre: true
post: if not db@pre->one (u:Database | u.username = username)
and age >= 18
      then
           db->one(user : Database |
           user.emailAddress = emailAddress and
           user.age = age and
           user.username = username) and
           db->size() = db@pre->size() + 1 and
           wpr^confirmRegister()
     else wpr^denyRegister()
     endif
```

Remark: We want to be able to identify users by a unique username.
OCL constraint:

```
context Database
inv : Database.allInstances()->isUnique(username) and
      self.age >= 18
```

Remarks:

- The *username* is uniquely identify a user in the database, as a user can only register with a unique username (Old R3)
- The user must be *18 years old* or older.

Validation

Exactly the operations occurring in the abstract specification must be described

- The operations in the abstract specification are described

Operation specifications must be consistent with abstract specifications:

- The operation specification of *register* is consistent with the abstract specification.

For each described operation, a pre- and postcondition must exist

- Each operation has pre and postconditions

The pre- and postconditions expressed in OCL must be syntactically correct

- Pre and postconditions are syntactically correct

The postcondition covers all cases exhibited in the abstract specification:

- The normal and exceptional case behaviors described in the abstract specification is covered in the postcondition.

All parameters of operations must be known by the caller and all parameters of sent messages must be known by the machine:

- Unregistered users can input all parameters to WebpageRegister, which forwards these to this operation.

Parameters must be used in the pre- and/or postcondition:

- The parameters are used in the postcondition.

Operation specification must be consistent with class model of the machine state

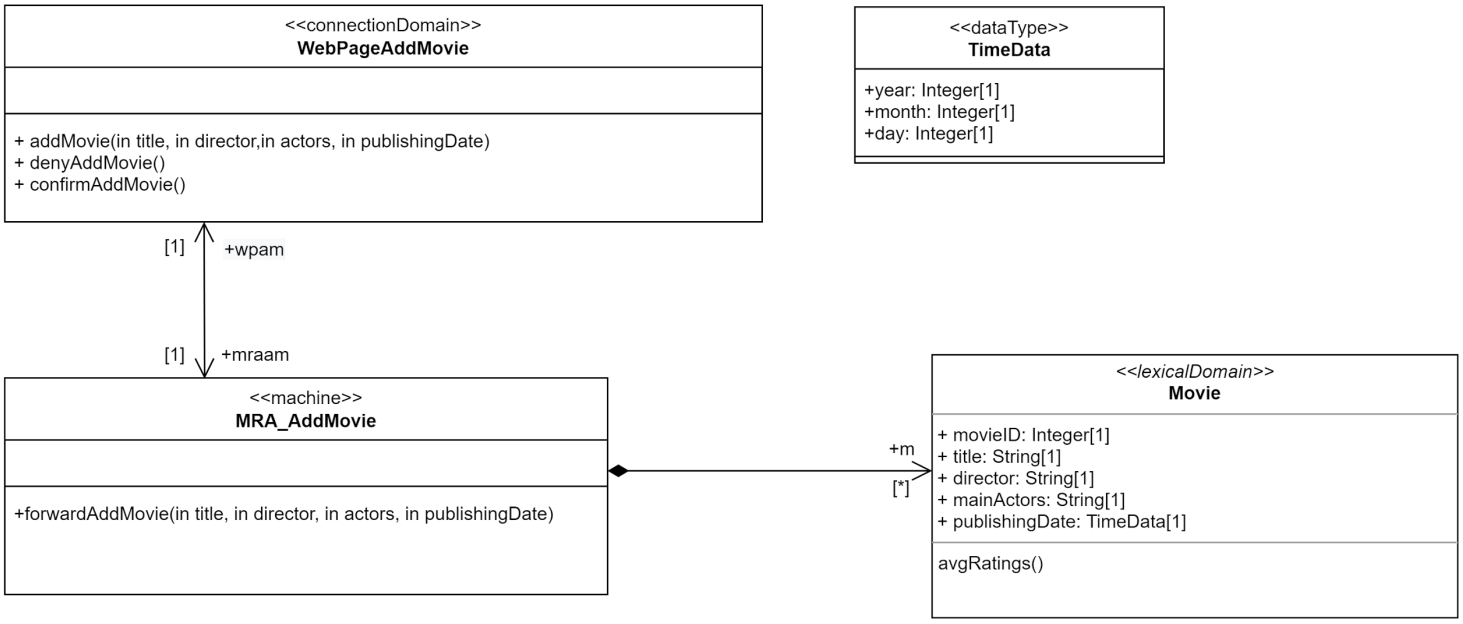
- The operation specification is consistent with its class model.

All classes, associations, and attributes newly introduced in the class model must be motivated by some operation specification:

- We do not have a new class.

4.2. R2: The operation AddMovie

Class Diagram R2



Class Model: AddMovie

Class Diagram R2

Name: forwardAddMovie
Description: Add a new Movie to the Movie Database.

OCL constraint:

```
context MRA_AddMovie::forwardAddMovie(title: String, director:
String, mainActors: String, publishingDate: TimeData )
pre: true
post: if not m@pre->one (m: Movie | m.title = title)
      then
           m->one( a : Movie |
           a.title = title and
           a.director = director and
           a.mainActors = mainActors and
           a.publishingDate = publishingDate) and
           m->size() =  m@pre->size() + 1 and
           wpr^confirmAddMovie()
     else wpr^denyAddMovie()
     endif
```

Remark: -

OCL constraint:

```
context Movie
inv : Movie.allInstances()->isUnique(title) and
      not self.mainActors = ""
```

Remarks: The combination of Title, Director, mainActors and PublishingDate uniquely identifies a movie.

Validation

Exactly the operations occurring in the abstract specification must be described

- The operations in the abstract specification are described

Operation specifications must be consistent with abstract specifications:

- The operation specification of addMovie is consistent with the abstract specification.

For each described operation, a pre- and postcondition must exist

- Each operation has pre and postconditions

The pre- and postconditions expressed in OCL must be syntactically correct

- Pre and postconditions are syntactically correct

The postcondition covers all cases exhibited in the abstract specification:

- The normal and exceptional case behaviors described in the abstract specification is covered in the postcondition.

All parameters of operations must be known by the caller and all parameters of sent messages must be known by the machine:

- Registered users can input all parameters to WebpageAddMovie, which forwards these to this operation.

Parameters must be used in the pre- and/or postcondition:

- The parameters are used in the postcondition.

Operation specification must be consistent with class model of the machine state

- The operation specification is consistent with its class model.

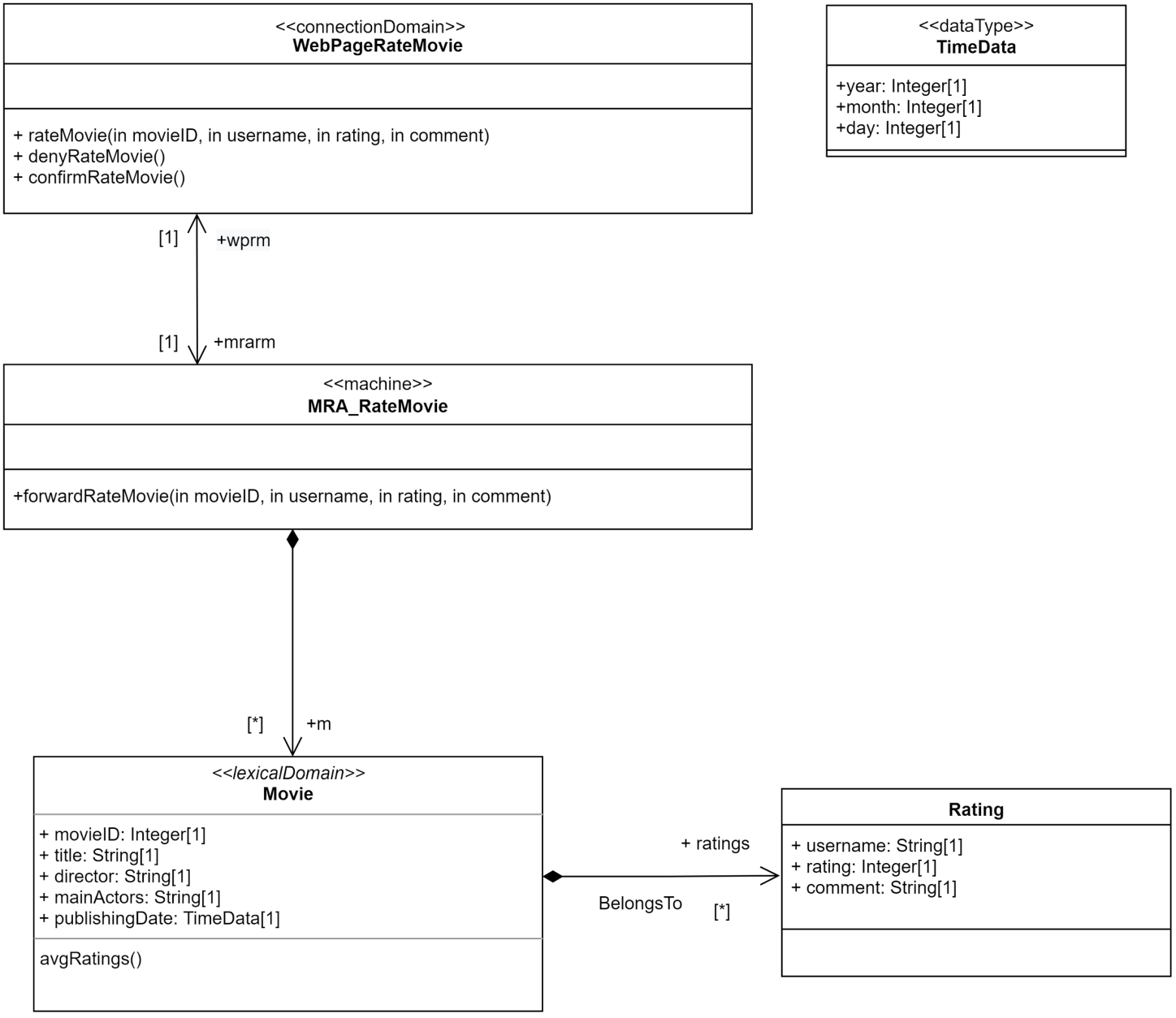
All classes, associations, and attributes newly introduced in the class model must be motivated by some operation specification:

- A new class *TimeData* is added to represent the dates for the movie publishing date.

- The attribute moveID was introduced for practical reasons.

4.3. R3: The operation RateMovie

Class Diagram R3



Class Model: RateMovie

Name: forwardRateMovie
Description: Add a new rating for a movie.

OCL constraint:

```
context MRA_RateMovie::forwardRateMovie(movieID: Integer,
username: String, rating: Integer, comment: String)
pre: true
post: let movie: Movie = m->any (m: Movie | m.movieID = movieID) in
      if not movie.oclIsUndefined and
      not movie@pre.ratings->one(r:Rating|r.username = username)and
      rating <= 10 and
      rating >= 1
      then
          movie.ratings->one( ra : Rating|
          ra.username = username and
          ra.rate = rate and
          ra.comment = comment) and
          movie.ratings->size() = movie@pre.ratings->size() + 1 and
          wprm^confirmRateMovie()
      else wprm^denyRateMovie()
      endif
```

Remark: -

OCL constraint:

```
context Rating
inv : Movie.allInstances()->forAll(mo : Movie |
          mo.ratings->isUnique( username )) and
      self.rating <= 10 and self.rating > 0
```

Remarks: The combination of user rate uniquely identifies rating a movie.

OCL constraint:

```
context Movie
inv : Movie.allInstances()->isUnique(title) and
      not self.mainActors = ""
```

Remarks: The combination of Title, Director, mainActors and PublishingDate uniquely identifies a movie.

Validation

Exactly the operations occurring in the abstract specification must be described

- The operations in the abstract specification are described

Operation specifications must be consistent with abstract specifications:

- The operation specification of rateMovie is consistent with the abstract specification.

For each described operation, a pre- and postcondition must exist

- Each operation has pre and postconditions

The pre- and postconditions expressed in OCL must be syntactically correct

- Pre and postconditions are syntactically correct

The postcondition covers all cases exhibited in the abstract specification:

- The normal and exceptional case behaviors described in the abstract specification is covered in the postcondition.

All parameters of operations must be known by the caller and all parameters of sent messages must be known by the machine:

- Registered users can input all parameters to WebpageRateMovie, which forwards these to this operation.

Parameters must be used in the pre- and/or postcondition:

- The parameters are used in the postcondition.

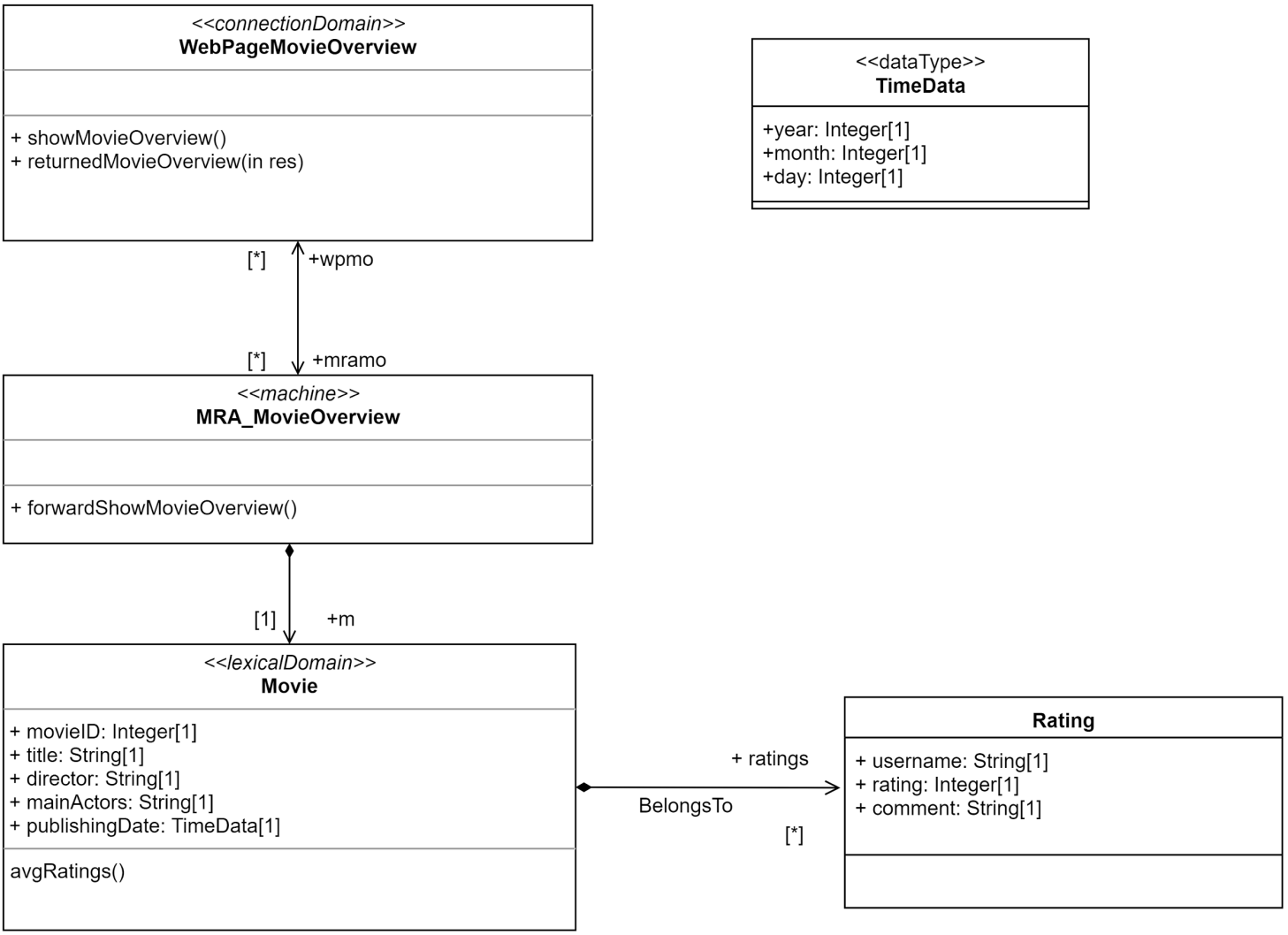
Operation specification must be consistent with class model of the machine state

- The operation specification is consistent with its class model.

All classes, associations, and attributes newly introduced in the class model must be motivated by some operation specification:

- The attribute moveID was introduced for practical reasons.

4.4. R4: The operation MovieOverview



Class Model: MovieOverview

Class Diagram R4

Name: forwardShowMovieOverview
Description: Add a new rating for a movie.

OCL constraint:

```
context MRA_Movie::forwardShowMovieOverview()
pre: true
post:
let res: OrderedSet(Movie)=
     m->select()->asOrderedSet()->sortedBy( mo: Movie |
     mo.avgRatings(mo.ratings)) in
     wpmo^returnedMovieOverview(res)
```

Remark: *avgRatings*() Will be implemented in the machine as a function(to be determined later).

OCL constraint:

```
context Movie
inv : Movie.allInstances()->isUnique(title) and
      not self.mainActors = ""
```

Remarks: *movies* uniquely identifies a movie

OCL constraint:

```
context Rating
inv : Movie.allInstances()->forAll(mo : Movie |
          mo.ratings->isUnique( username )) and
      self.rating <= 10 and self.rating > 0
```

Remarks: *username* uniquely identifies a rating and used to limit the rating of a movie to one rate per user.

Validation

Exactly the operations occurring in the abstract specification must be described

- The operations in the abstract specification are described

Operation specifications must be consistent with abstract specifications:

- The operation specification of movieOverview is consistent with the abstract specification.

For each described operation, a pre- and postcondition must exist

- Each operation has pre and postconditions

The pre- and postconditions expressed in OCL must be syntactically correct

- Pre and postconditions are syntactically correct

The postcondition covers all cases exhibited in the abstract specification:

- The normal case behaviors described in the abstract specification is covered in the postcondition.

All parameters of operations must be known by the caller and all parameters of sent messages must be known by the machine:

- There is no parameter here

Parameters must be used in the pre- and/or postcondition:

- There is no parameter here

Operation specification must be consistent with class model of the machine state

- The operation specification is consistent with its class model.

All classes, associations, and attributes newly introduced in the class model must be motivated by some operation specification:

- A new class *TimeData* is added to represent the dates for the movie publishing date.
- avgRatings(r : Rating) : Integer is added to the class Movie to calculate the average rating form the rating list of the movie
- The attribute moveID was introduced for practical reasons.

5. A6 Software Life-Cycle

5.1. Movie Rating App Life-Cycle

$$LC_{UnregisteredUser} = Register$$

$$LC_{RegisteredUser} = (AddMovie \mid (ShowMovieOverview\ ;[RateMovie]))^{*}$$

Note that RateMovie is possible only after ShowMovieOverview.

$$LC_{MovieRatingApp} = \left(||_{i=1}^{n} LC_{UnregisteredUser_i} \right) || \left(||_{j=1}^{n} LC_{RegisteredUser_j} \right)$$

Validation for A6

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

scenario	life-cycle expression
sdRegister	$LC_{UnregisteredUser}$
sdAddMovie	$LC_{RegisteredUser}$
sdRateMovie	$LC_{RegisteredUser}$
sdMovieOverview	$LC_{RegisteredUser}$

For the biddable domain UnregisterUser exactly one life-cycle exists, namely $LC_{UnregisteredUser}$

For the biddable domain RegisterUser exactly one life-cycle exists, namely $LC_{RegisteredUser}$

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

- Register can be executed if unregisteredUser is at least eighteen years old and its username is unique, otherwise the operation is denied.
- AddMovie can be executed if a movie is not yet contained in the database, otherwise the operation is denied.
- RateMovie can be executed if a movie already exists in the movieDatabase and a

MovieRating has to be valid, otherwise the operation is denied.
- MovieOverview has no state predicates at the beginning and end. Hence, it can be executed an arbitrary number of times.

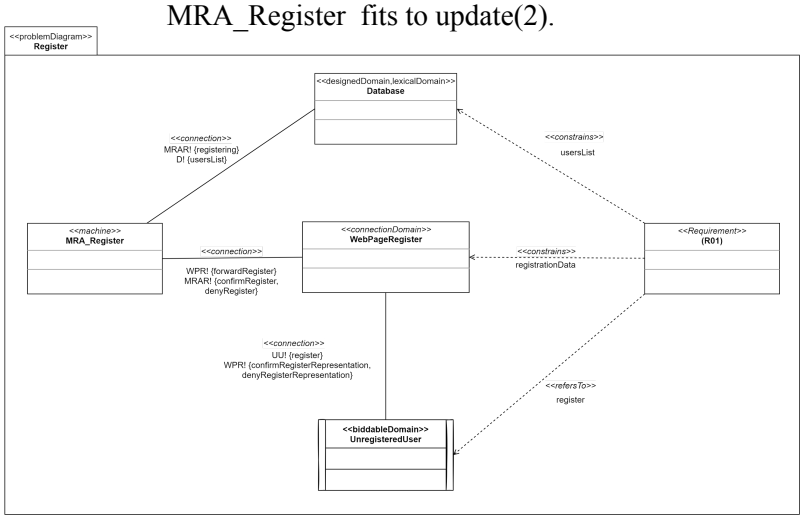
The life-cycles are consistent with the pre- and postconditions in Step A5: Operations and data specification:
- The sequence diagram Register contains the operation getUsersList. It has no precondition. Hence, it can be executed at any position of the life-cycle
- The sequence diagram AddMovie contains the operation getAllMovieList. It has no precondition. Hence, it can be executed at any position of the life-cycle
- The sequence diagram RateMovie contains the operation getUserRatingOfMovie. It has no precondition. Hence, it can be executed at any position of the life-cycle
- The sequence diagram MovieOverview contains the operation getMovieOverview. It has no precondition. Hence, it can be executed at any position of the life-cycle.

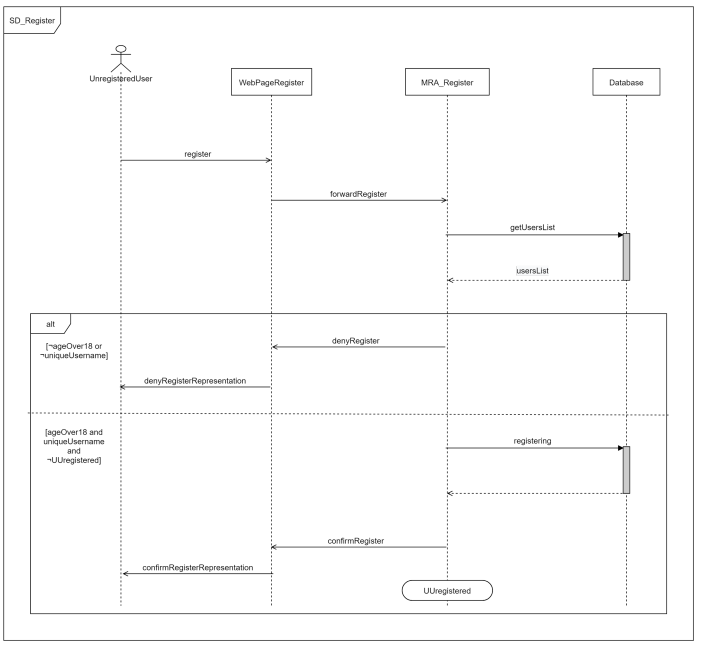
Exactly one life-cycle exists for the machine domain, that combines all life-cycles
The life-cycle $LC_{MovieRatingApp}$ exists for the machine domain. It combines all life-cycles.

6. D1: Software architecture

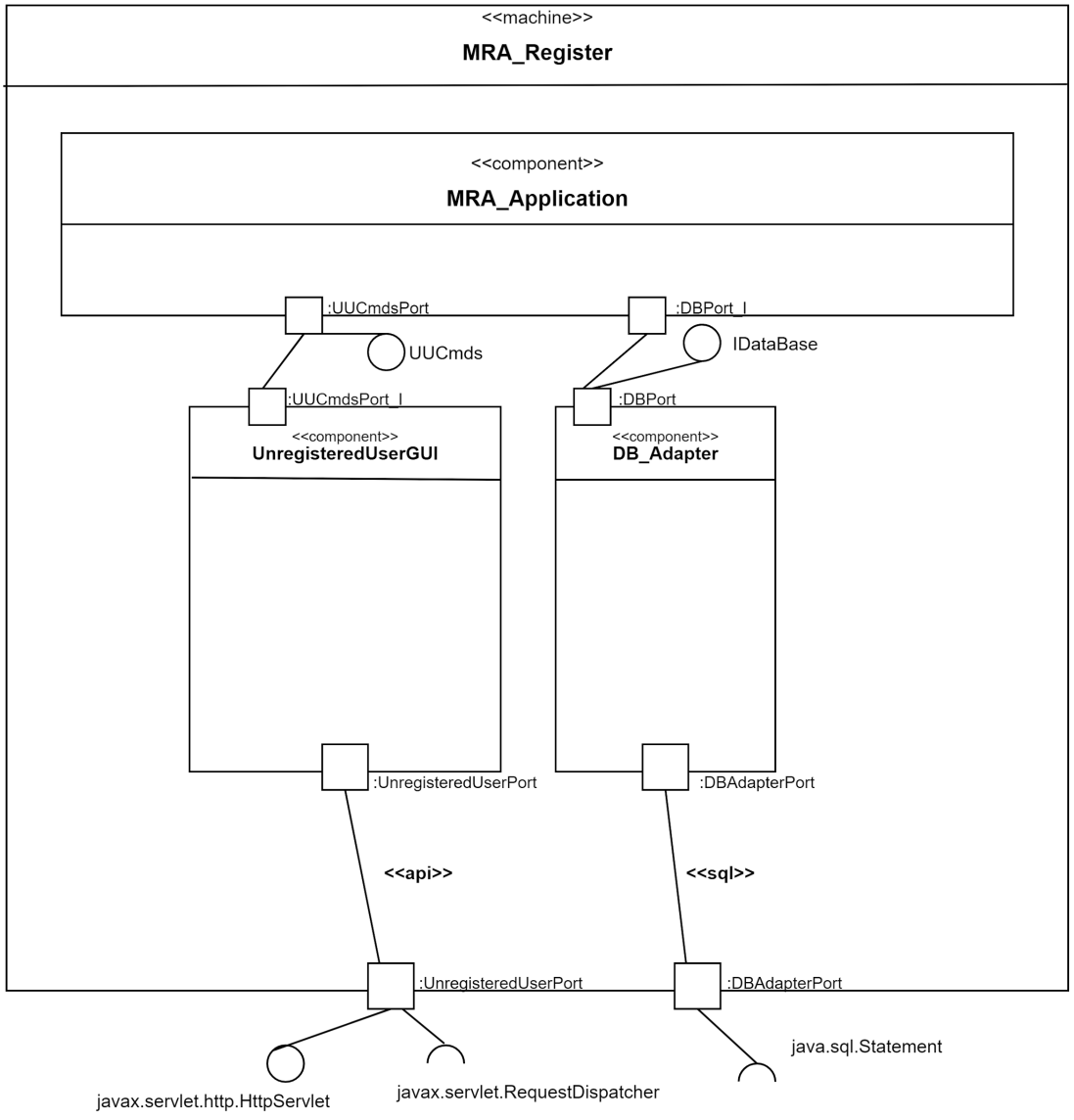
R1: Register

MRA_Register fits to update(2).

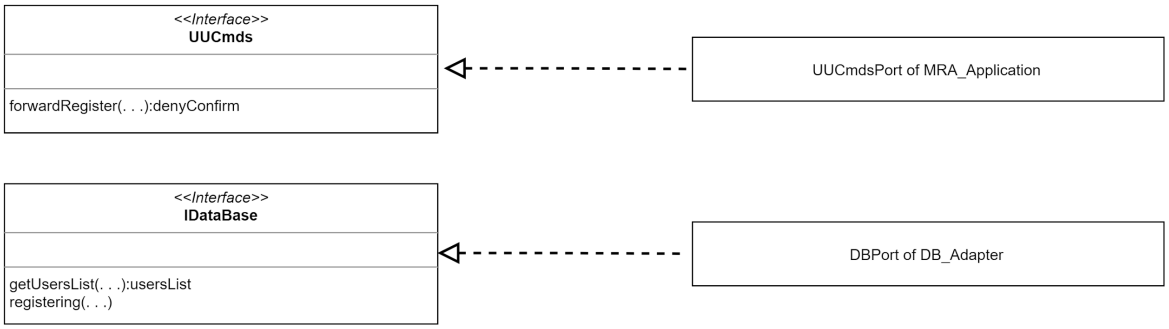




R1: Instantiated architectural pattern for MRA_Register:

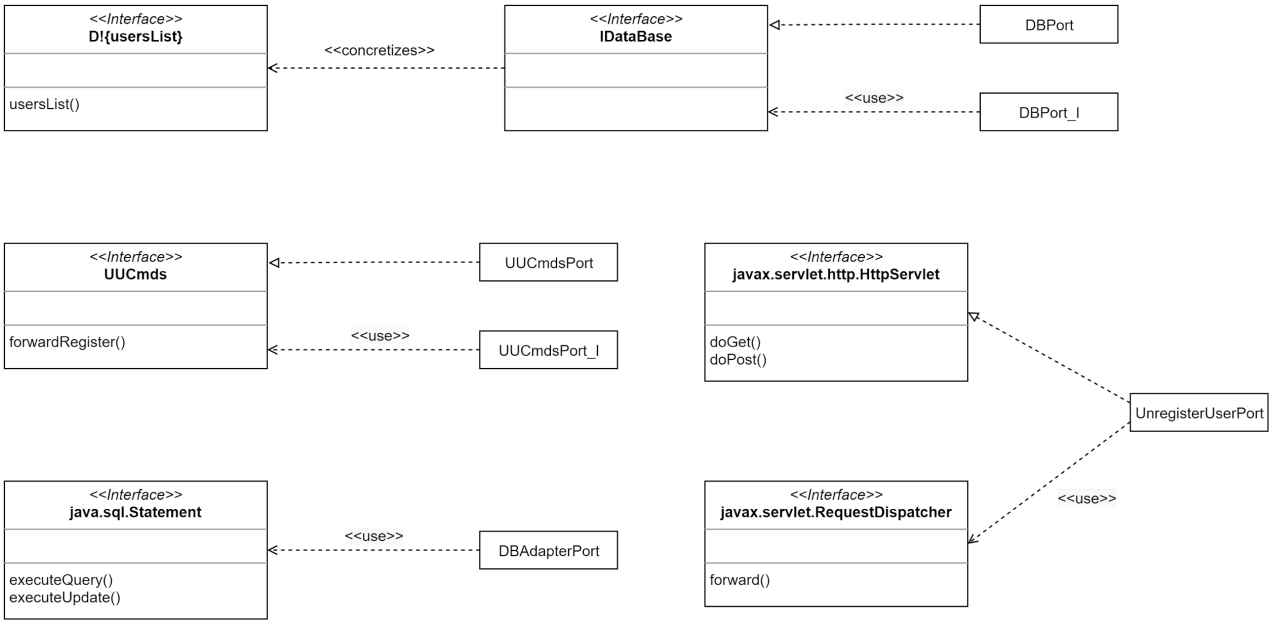


Internal interfaces in MRA_Register:



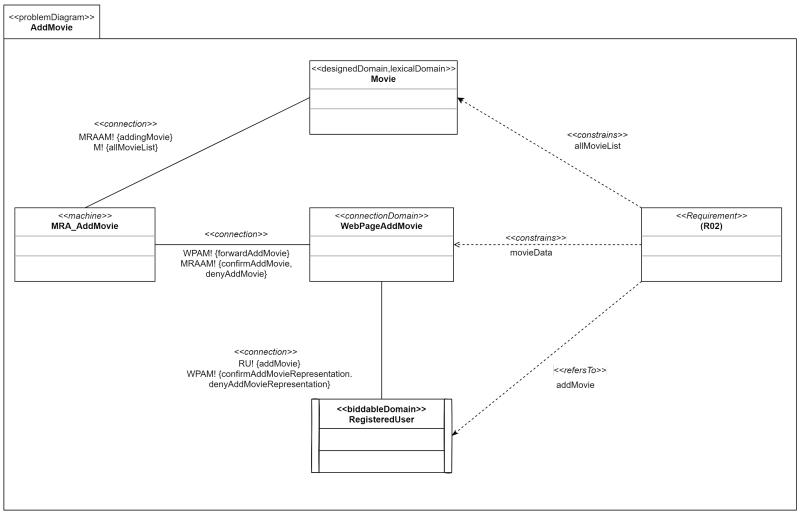
confirmRegister is realized by the return value of the registering and denyRegister is realized by the return value of the getUserList. Both functions are also related to the inputted age from the user.

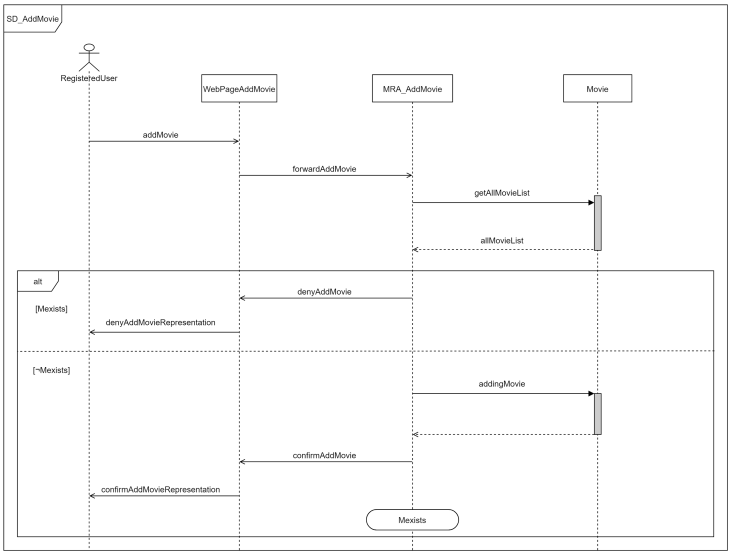
Port types and interface relations for MRA_Register



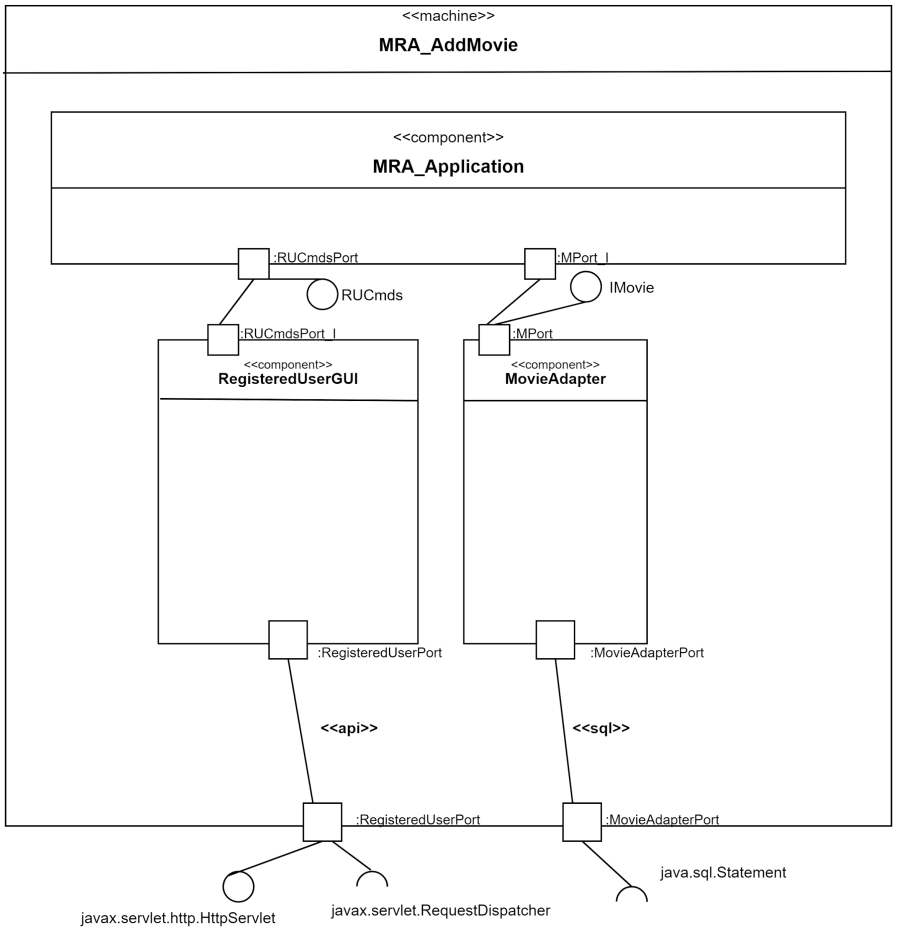
R2: AddMovie

MRA_AddMovie fits to update(2).

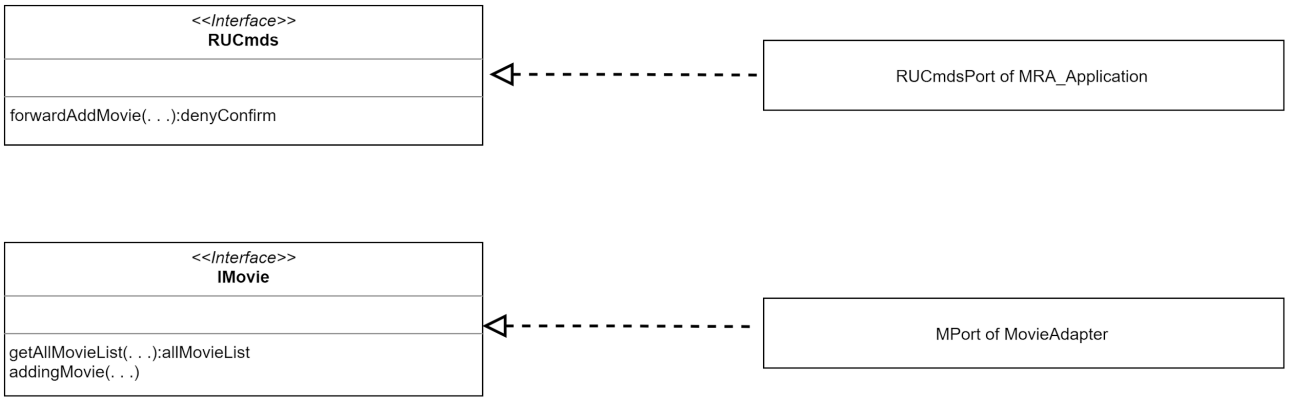




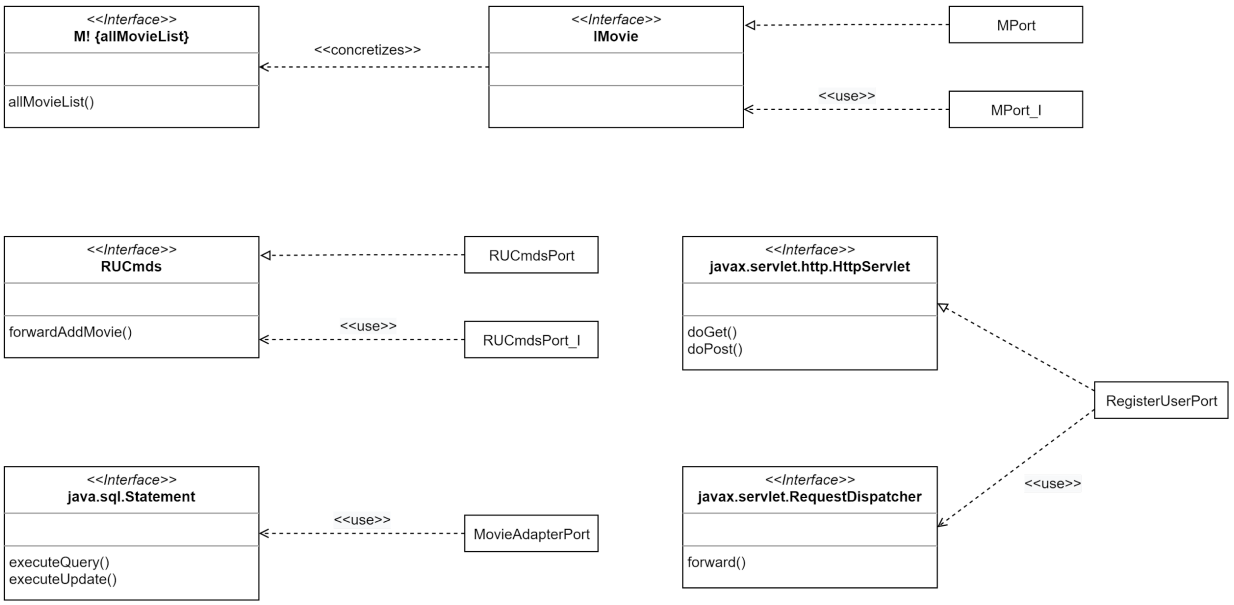
R2: Instantiated architectural pattern for MRA_AddMovie:



Internal interfaces in MRA_AddMovie:

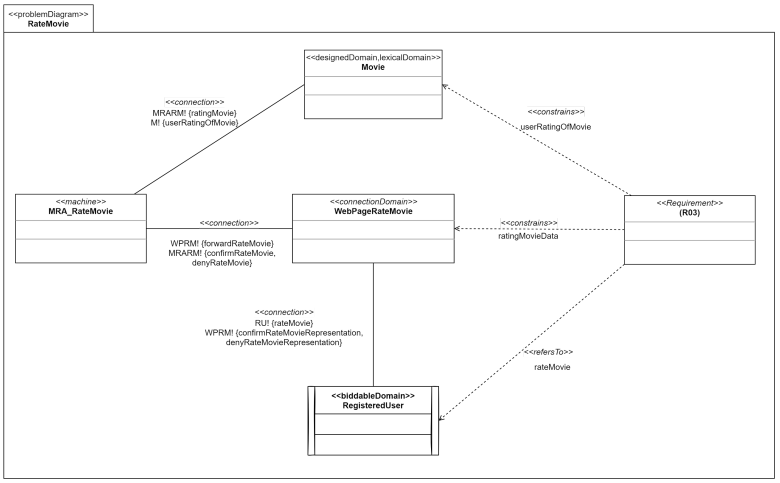


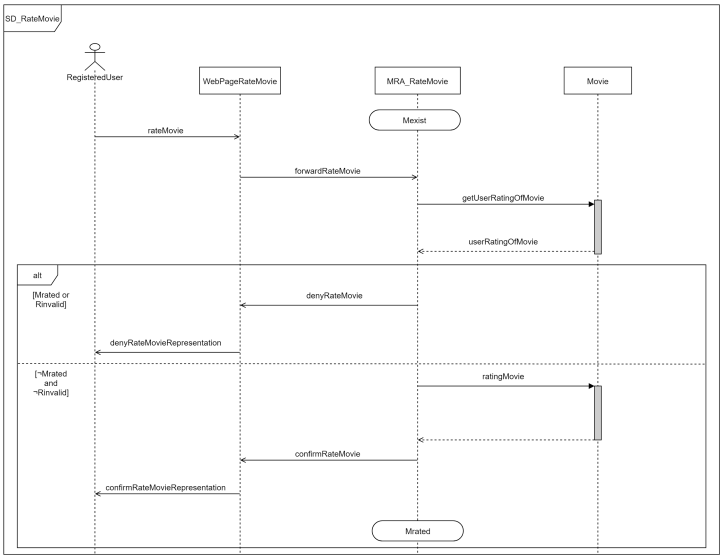
Port types and interface relations for MRA_AddMovie:



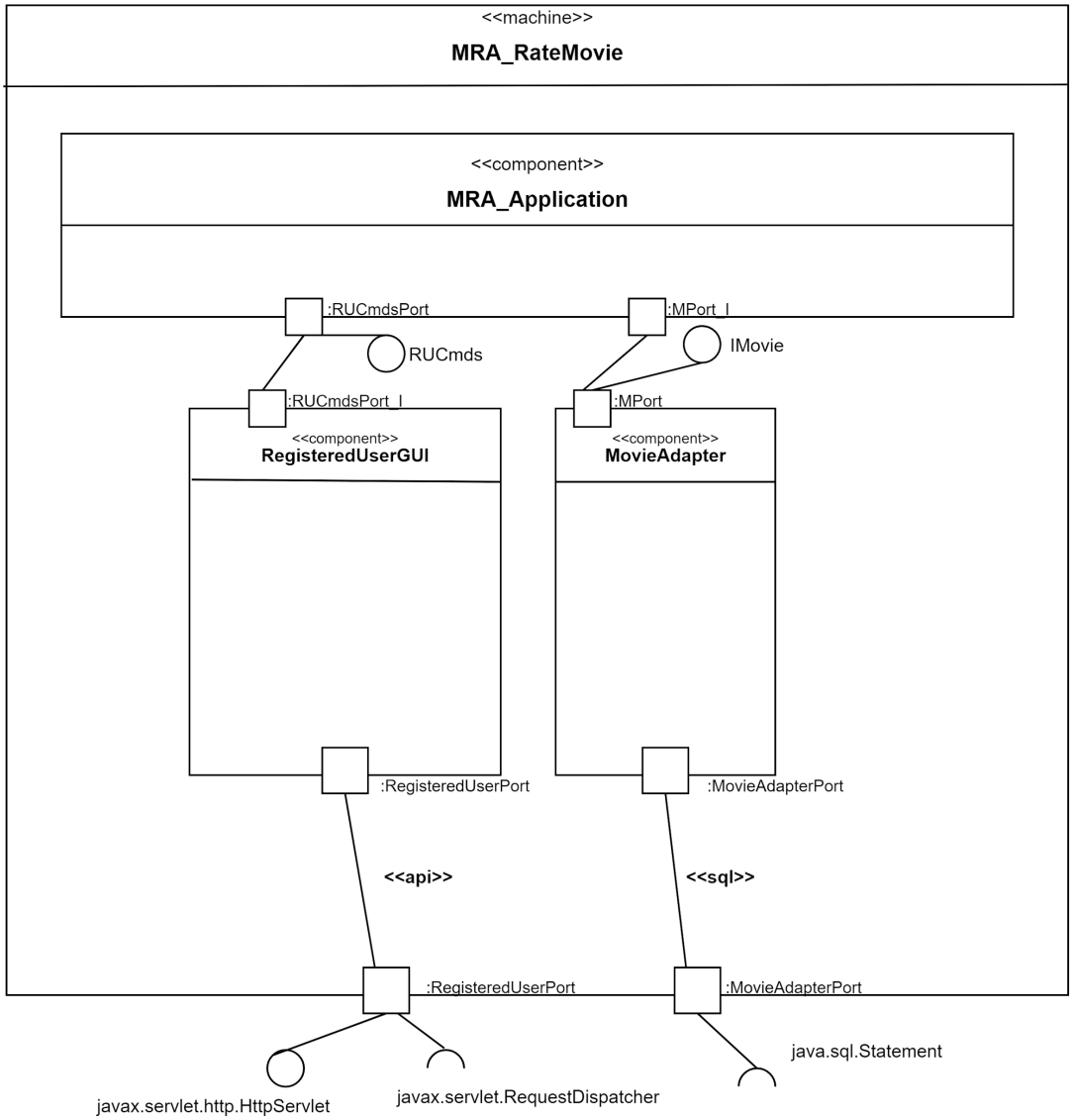
R3: RateMovie

MRA_RateMovie fits to update(2).



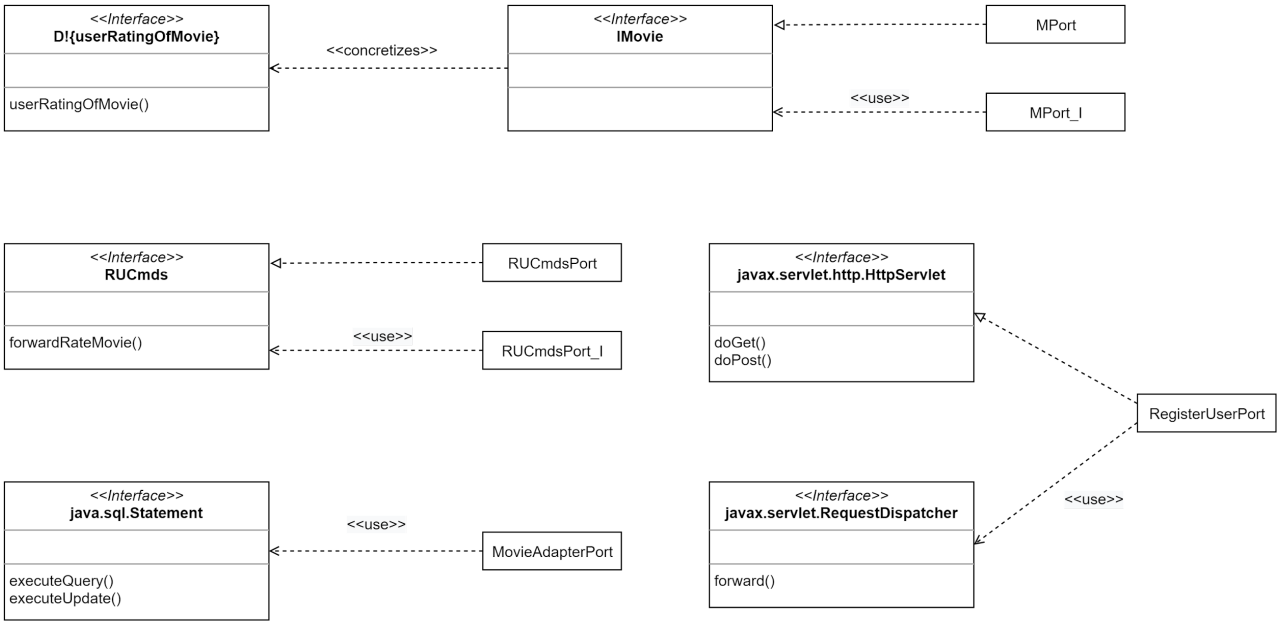


R3: Instantiated architectural pattern for MRA_RateMovie:



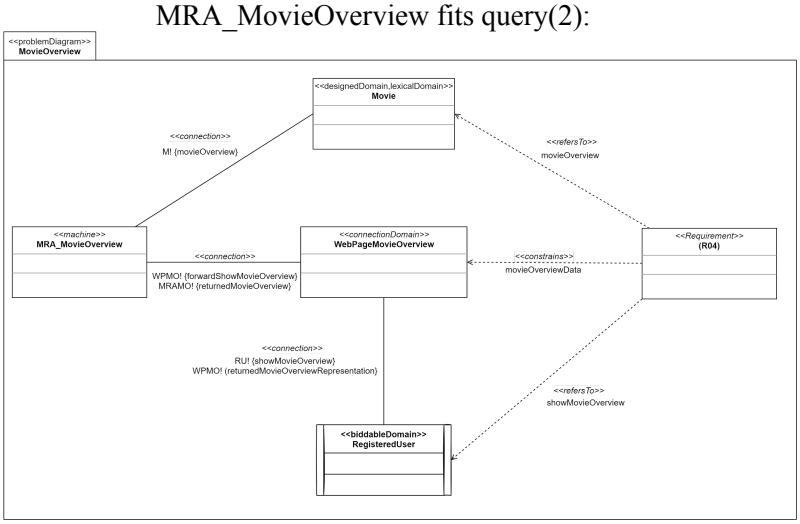
Internal interfaces in MRA_RateMovie:

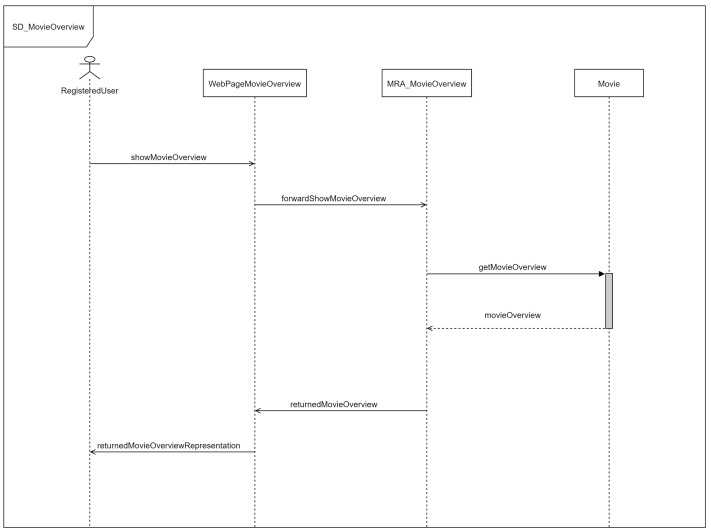
Port types and interface relations for MRA_RateMovie:



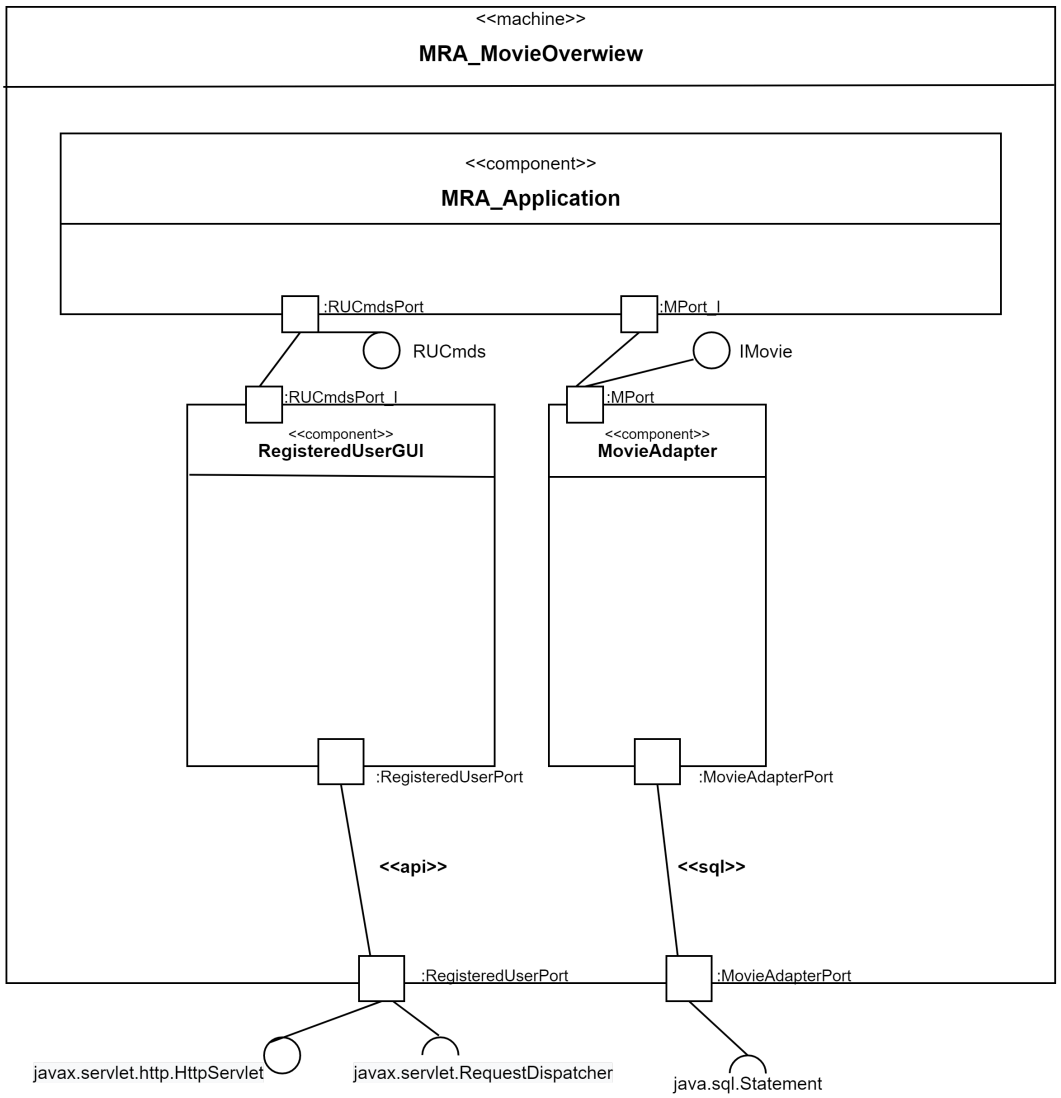
R4: MovieOverview

MRA_MovieOverview fits query(2):



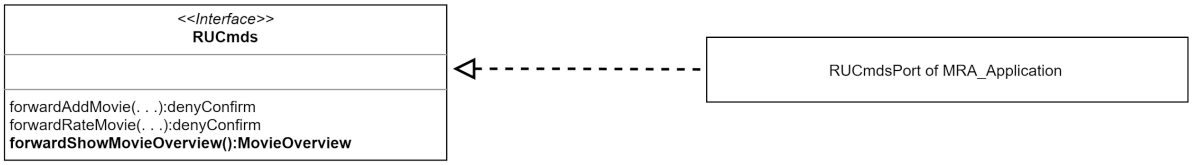


R4: Instantiated architectural pattern for MRA_MovieOverview:

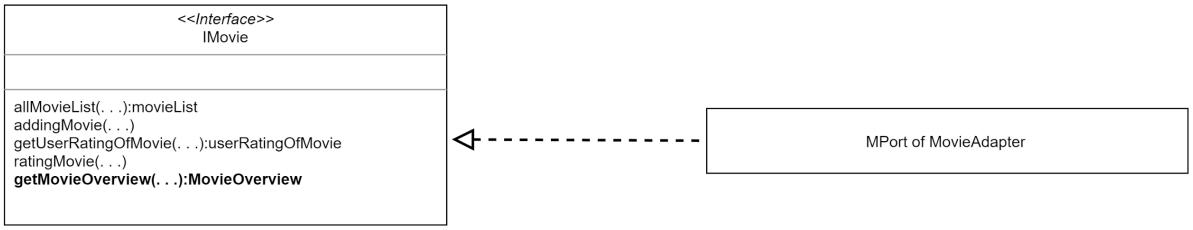


Internal interfaces in MRA_MovieOverview:

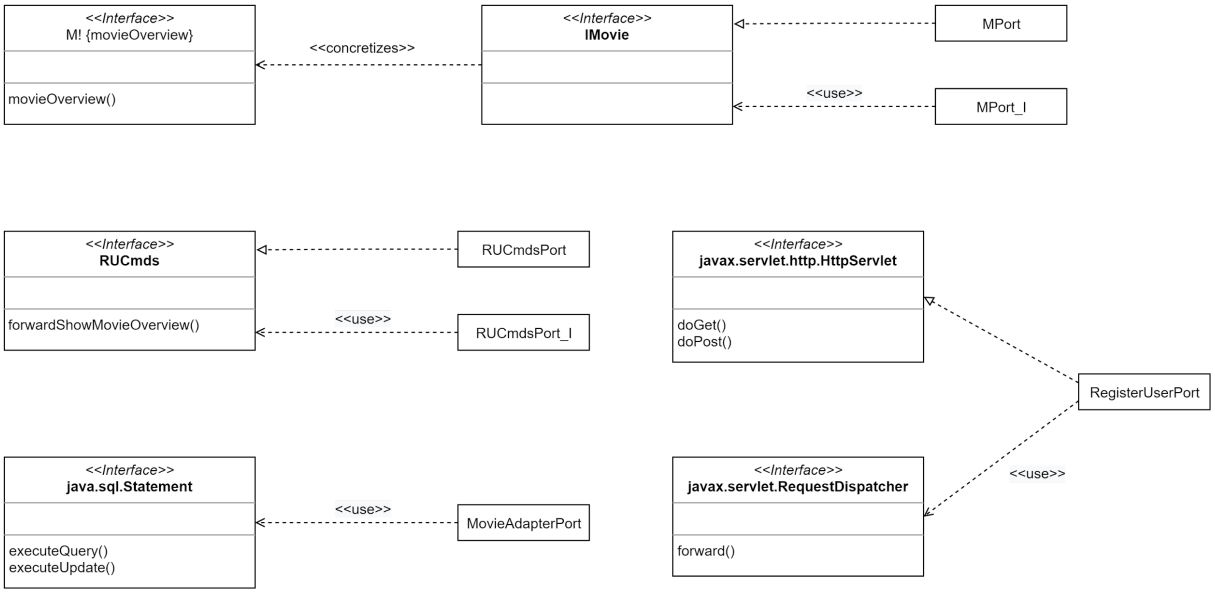
RUCmds from MRA_RateMovie is extended as follows:



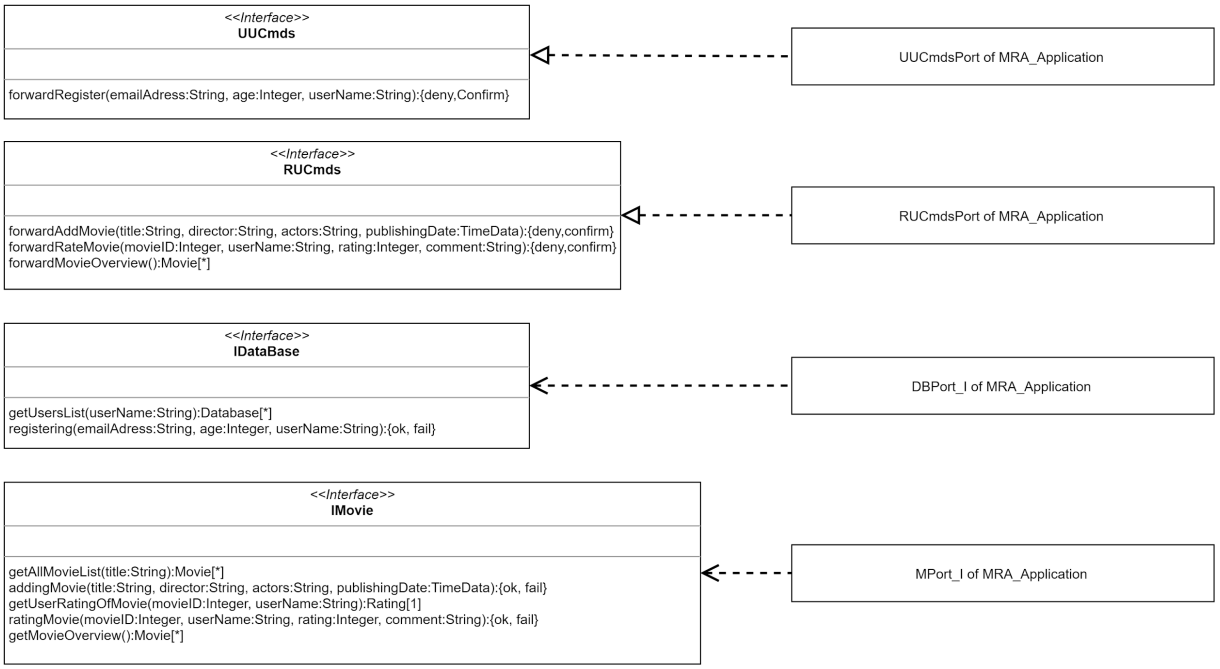
IMovie from MRA_RateMovie is extended as follows:



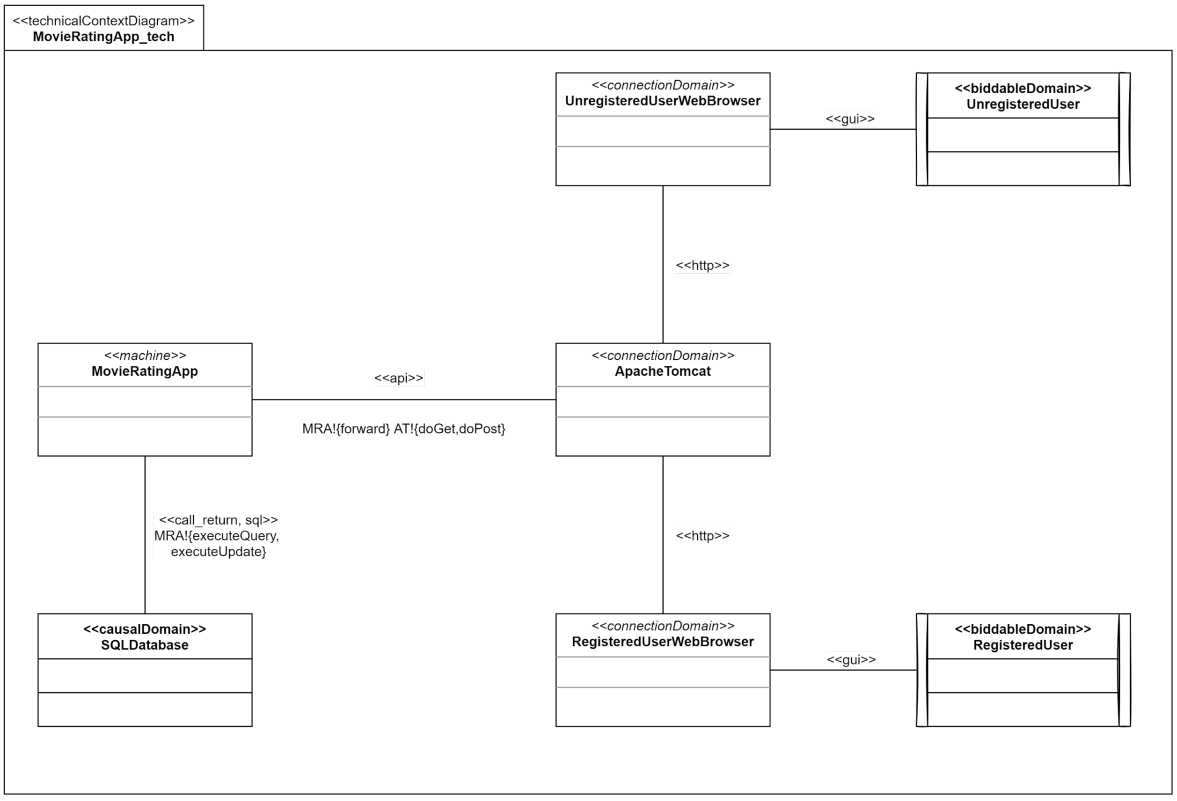
Port types and interface relations for MRA_MovieOverview:



Refining app if interface classes



Refining tech if'' interface classes



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

Refining adapter_if' interface classes:

There are no HAL components in the subproblem architectures. Hence, there are no "adapter_if" interface classes that need to be refined

The components can be merged a follows:

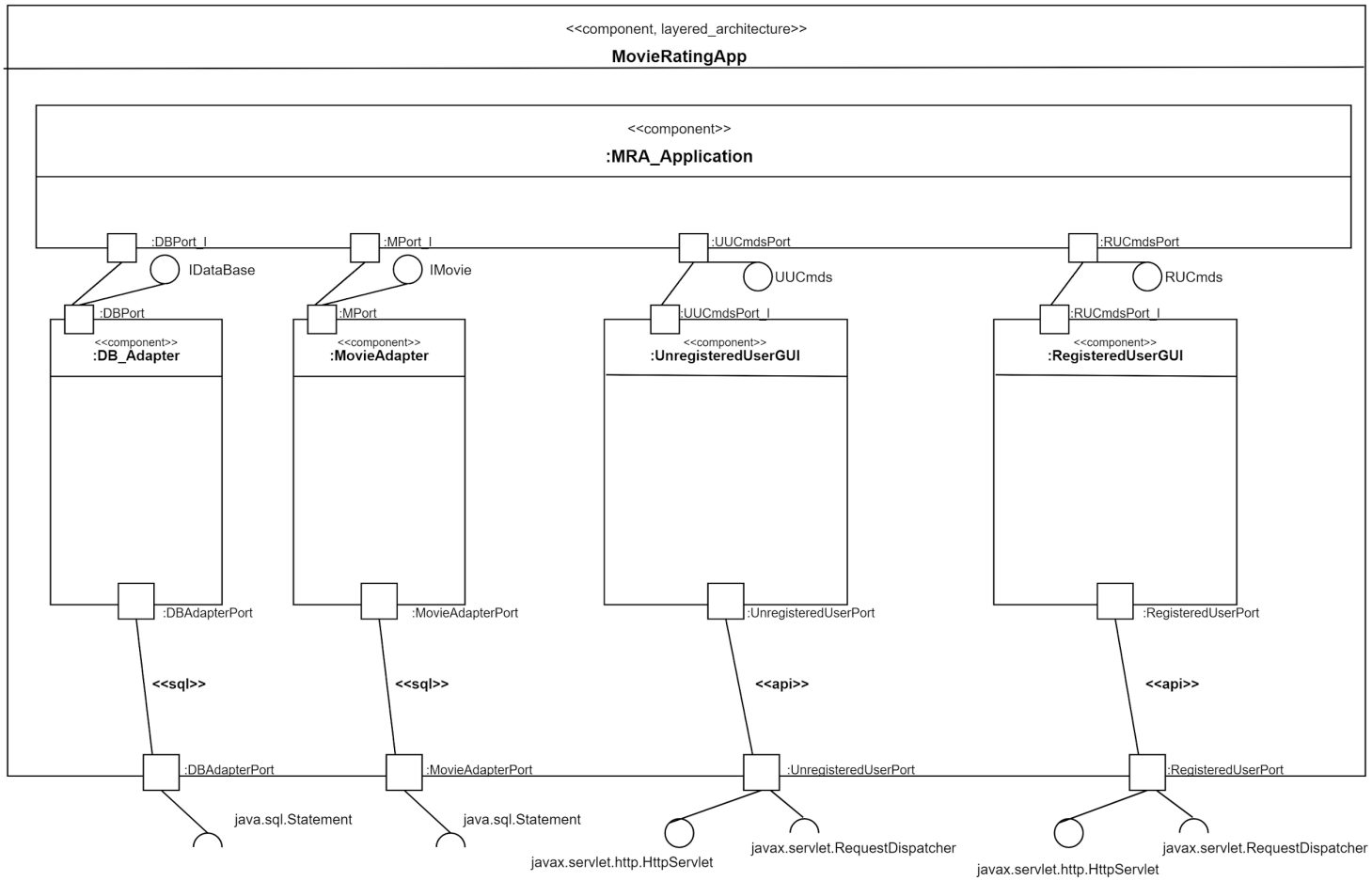
- The application components in all architectures for the subproblems except MRA_Register should be merged because the Subproblems MovieOverview and RateMovie are related sequentially. The Subproblem AddMovie is also merged because of reasons of simplicity.
- The **MovieAdapters** and **DB_Adapter** in all architectures for the subproblems should be merged into **DB_Adapter** because of establishing the connection to the DB
- The **RegisteredUserGUI** for the Registered User in all architectures for the subproblems uses the same technology and should be merged.

Reusable Components:

- There is no reusable components

Components to Develop

- We have to develop the MRA_Application, RegisteredUserGUI, UnregisteredUserGUI and adapters for the external components.
- The DB_Adapter is responsible for creating and maintaining tables for all persistent classes.



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
Register	forwardRegister	in	UUCmds::forwardRegister	provided
	getUsersList	out	IDataBase::getUsersList	required
	usersList	in	return value of IDataBase::getUsersList	required
	denyRegister	out	return value of UUCmds::forwardRegister	provided
	registering	out	IDataBase::registering	required
	confirmRegister	out	return value of UUCmds::forwardRegister	provided

AddMovie	forwardAddMovie	in	RUCmds::forwardAddMovie	provided
	getAllMovieList	out	IMovie::getAllMovieList	required
	allMovieList	in	return value of IMovie::getAllMovieList	required
	denyAddMovie	out	return value of RUCmds::forwardAddMovie	provided
	addingMovie	out	IMovie::addingMovie	required
	confirmAddMovie	out	return value of RUCmds::forwardAddMovie	provided
RateMovie	forwardRateMovie	in	RUCmds::forwardRateMovie	provided
	getUserRatingOfMovie	out	IMovie::getUserRatingOfMovie	required
	userRatingOfMovie	in	return value of IMovie::getUserRatingOfMovie	required
	denyRateMovie	out	return value of RUCmds::forwardRateMovie	provided
	ratingMovie	out	IMovie::ratingMovie	required
	confirmRateMovie	out	return value of RUCmds::forwardRateMovie	provided
MovieOverview	forwardShowMovieOverview	in	RUCmds::forwardShowMovieOverview	provided
	getMovieOverview	out	IMovie::getMovieOverview	required
	movieOverview	in	return value of IMovie::getMovieOverview	required
	returnedMovieOverview	out	return value of RUCmds::forwardShowMovieOverview	provided

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

provided by machine	**required by adapter/ provided by app**
java.servlet.http.HttpServlet	UUCmds, RUCmds

required by the machine	**provided by adapter / required by app**

javax.servlet.RequestDispatcher	return values in UUCmds and RUCmds
java.sql.Statement	IDatabase, IMovie

external port type	**interface in architecture**	**required/ provided**	**interface in technical context diagram**
UnregisteredUserPort	javax.servlet.http.HttpServlet	provided	AT!{doGet, doPost}
	javax.servlet.RequestDispatcher	required	VR!{forward}
DBAdapterPort	java.sql.Statement	required	VR!{executeQuery, executeUpdate}

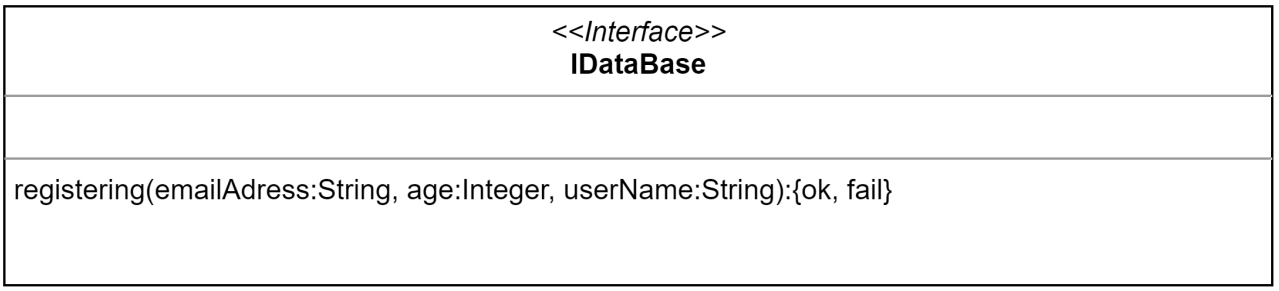
7. D2 Inter-Component Interaction

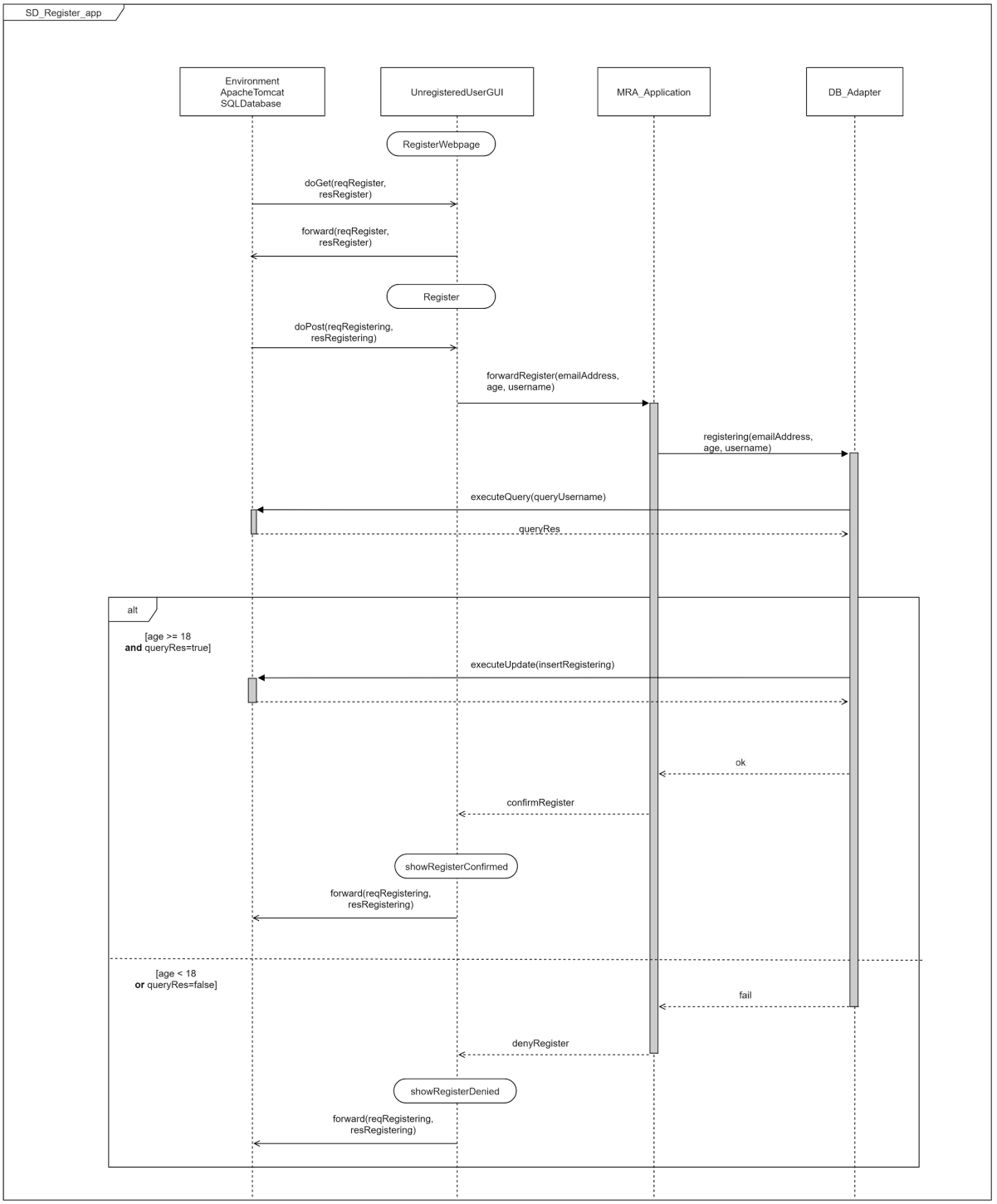
- In our example, we will use MySQL as a database system.
- This relational database system requires a server software, i.e. MySQL server.
- We do not consider transactions or other commitment strategies for the presented implementation in this lecture.

R1 - Register:

From now on the function "**getUserList**" will be merged with the function "**registering**(emailAddress:String, age:Integer, userName:String):{ok, fail}" for simplicity.

```
context MRA_Register::forwardRegister(emailAddress: String, age:
Integer, username: String)
pre: true
post: if not db@pre->one (u:Database | u.username = username)
and age >= 18
      then
            db->one(user : Database |
            user.emailAddress = emailAddress and
            user.age = age and
            user.username = username) and
            db->size() = db@pre->size() + 1 and
            wpr^confirmRegister()
     else wpr^denyRegister()
     endif
```





- *reqRegister* and *reqRegistering* represent HTTPServletRequest objects containing the required user input.
- *resRegister* and *resRegistering* represent HttpServletResponse objects as the counterpart for the request.
- The state predicate *Register* represents that the input form for Registering is shown.
- The state predicate *showRegisterConfirmed* represents that the confirmation is shown.
- The state predicate *showRegisterDenied* represents that an error message is shown.

- *forward(...)* sends the request and response back to the server to generate the HTML webpage.
- Since we use a MySQL database, we do not need to specify the interfaces to lexical domains in more detail. We use standardized SQL statements to access the database.

queryUsername

```
SELECT count(*) from Users WHERE username = "username"
```

insertRegistering

```
INSERT INTO Users ( username, emailAddress, age ) VALUES
("username", "emailAddress", "age")
```

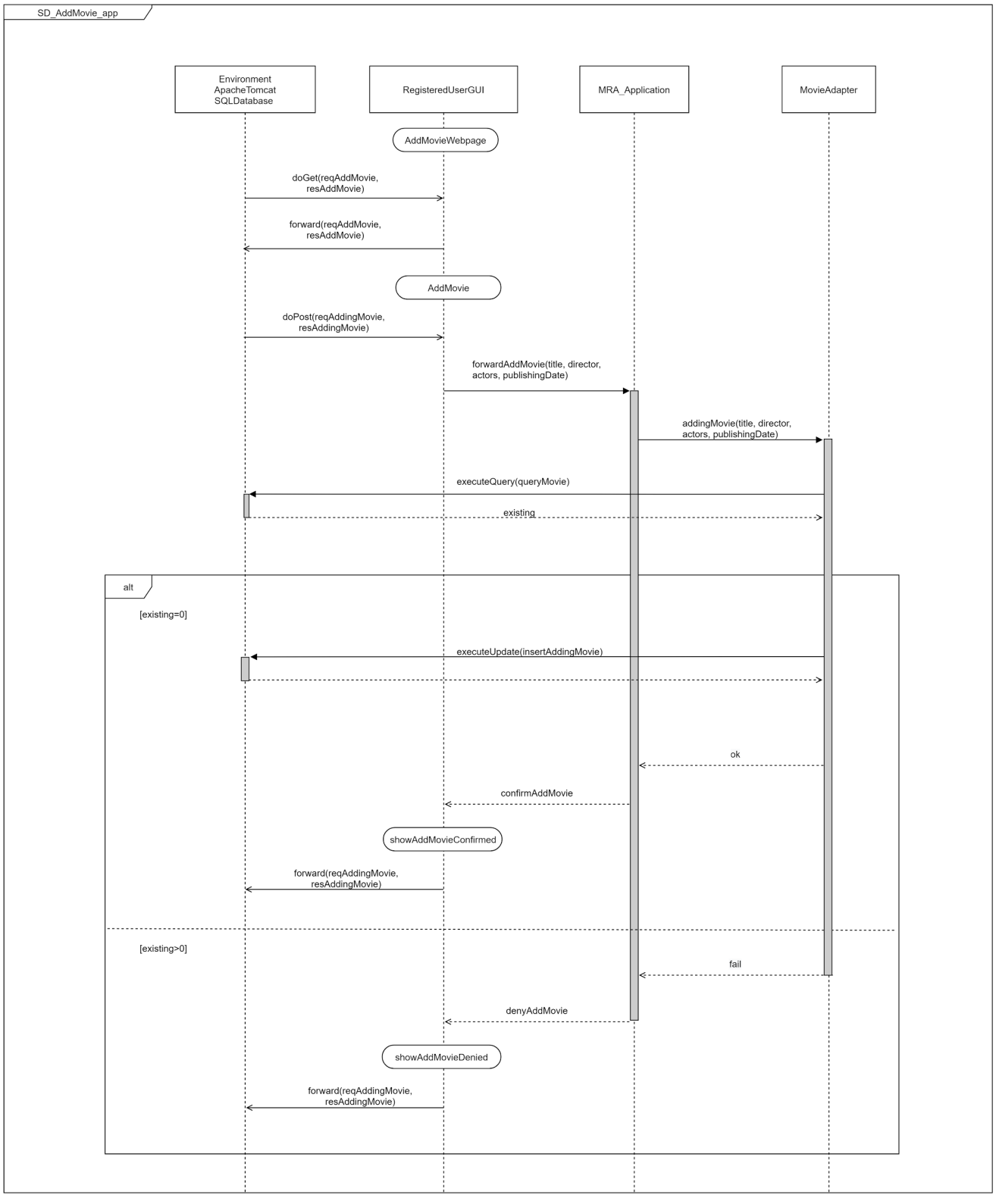
R2 - AddMovie:

From now on the function "**getAllMovieList**" will be merged with the function "**addingMovie**(title:String, director:String, actors:String, publishingDate:TimeData):{ok, fail}" for simplicity.

```
context MRA_AddMovie::forwardAddMovie(title: String, director:
String, mainActors: String, publishingDate: TimeData )
pre: true
post: if not m@pre->one (m: Movie | m.title = title)
      then
           m->one( a : Movie |
           a.title = title and
           a.director = director and
           a.mainActors = mainActors and
           a.publishingDate = publishingDate) and
           m->size() =  m@pre->size() + 1 and
           wpr^confirmAddMovie()
     else wpr^denyAddMovie()
     endif
```

<<*Interface*>>
IMovie

addingMovie(title:String, director:String, actors:String, publishingDate:TimeData):{ok, fail}



- *reqAddMovie* and *reqAddingMovie* represent HTTPServletRequest objects containing the required user input.
- *resAddMovie* and *resAddingMovie* represent HttpServletResponse objects as the counterpart for the request.
- The state predicate *AddMovie* represents that the input form for AddingMovie is shown.
- The state predicate *showAddMovieConfirmed* represents that the confirmation is shown.
- The state predicate *showAddMovieDenied* represents that an error message is shown.
- *forward(...)* sends the request and response back to the server to generate the HTML webpage.
- Since we use a MySQL database, we do not need to specify the interfaces to lexical domains in more detail. We use standardized SQL statements to access the database.

queryMovie

```
SELECT count(*)  from Movies WHERE title= "title"
```

insertAddingMovie

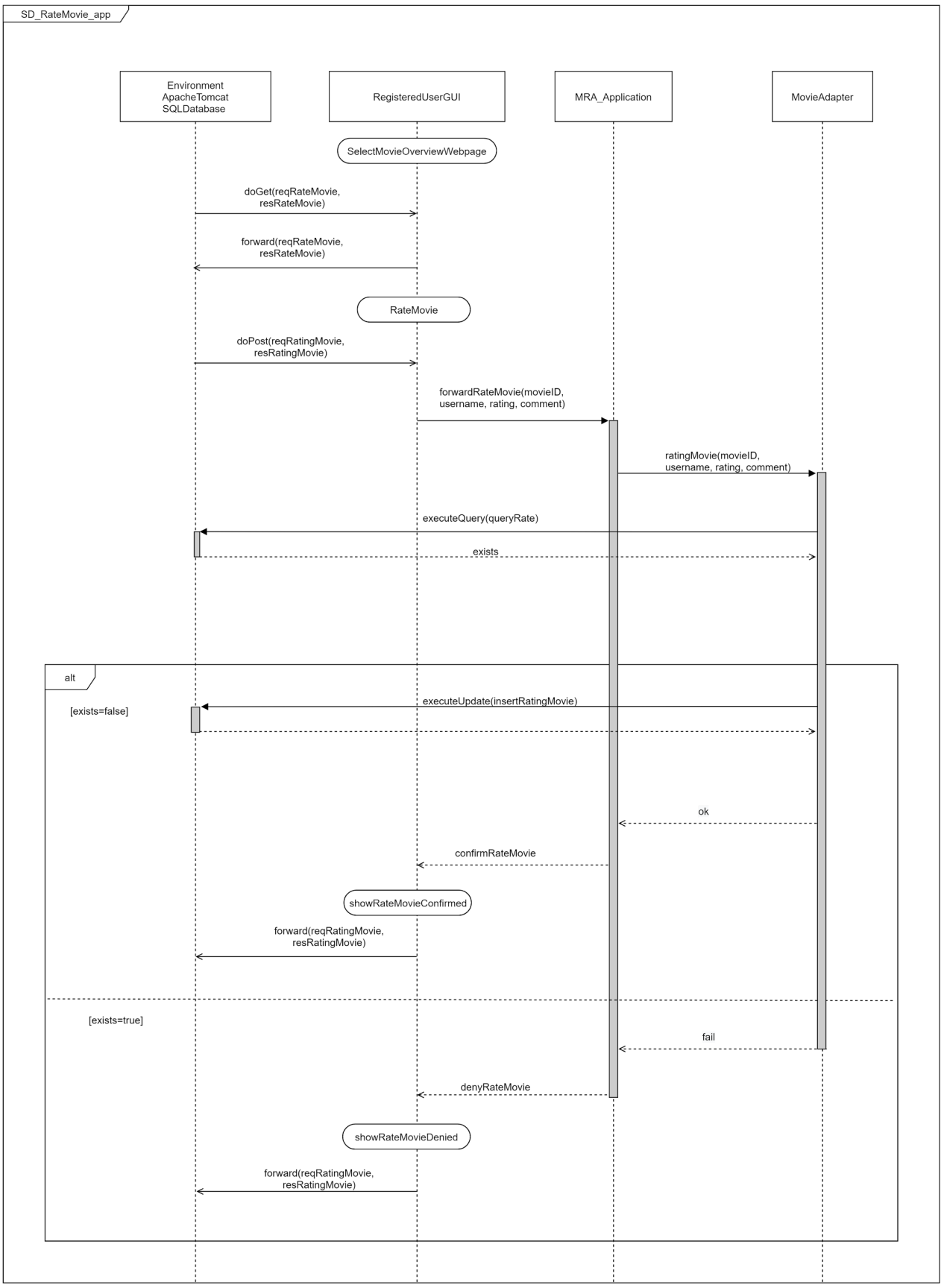
```
INSERT INTO Movies (title, director, actors, publishingDate) VALUES
("title", "director", "actors", "publishingDate")
```

R3 - RateMovie:

From now on the function "**getUserRatingOfMovie**" will be merged with the function "**ratingMovie**(movieID: **Integer**, username: **String**, rating: **Integer**, comment: **String**):{ok, fail}" for simplicity.

```
context MRA_RateMovie::forwardRateMovie(movieID: Integer,
username: String, rating: Integer, comment: String)
pre: true
post: let movie: Movie = m->any (m: Movie | m.movieID = movieID) in
      if not movie.oclIsUndefined and
      not movie@pre.ratings->one(r:Rating|r.username = username)and
      rating <= 10 and
      rating >= 1
      then
          movie.ratings->one( ra : Rating|
          ra.username = username and
          ra.rate = rate and
          ra.comment = comment) and
          movie.ratings->size() = movie@pre.ratings->size() + 1 and
          wprm^confirmRateMovie()
      else wprm^denyRateMovie()
      endif
```

<<Interface>> **IMovie**
addingMovie(title: String, director: String, mainActors: String, publishingDate: TimeData):{ok, fail} ratingMovie(movieID: Integer, username: String, rating: Integer, comment: String):{ok, fail}


SD_RateMovie_app
Environment
ApacheTomcat
SQLDatabase
RegisteredUserGUI
MRA_Application
MovieAdapter
SelectMovieOverviewWebpage
doGet(reqRateMovie,
resRateMovie)
forward(reqRateMovie,
resRateMovie)
RateMovie
doPost(reqRatingMovie,
resRatingMovie)
forwardRateMovie(movieID,
username, rating, comment)
ratingMovie(movieID,
username, rating, comment)
executeQuery(queryRate)
exists
alt
[exists=false]
executeUpdate(insertRatingMovie)
ok
confirmRateMovie
showRateMovieConfirmed
forward(reqRatingMovie,
resRatingMovie)
[exists=true]
fail
denyRateMovie
showRateMovieDenied
forward(reqRatingMovie,
resRatingMovie)

- *reqRateMovie* and *reqRatingMovie* represent HttpServletRequest objects containing the required user input.
- *resRateMovie* and *resRatingMovie* represent HttpServletResponse objects as the counterpart for the request.
- The state predicate *RateMovie* represents that the input form for RatingMovie is shown.
- The state predicate *showRateMovieConfirmed* represents that the confirmation is shown.
- The state predicate *showRateMovieDenied* represents that an error message is shown.
- *forward(...)* sends the request and response back to the server to generate the HTML webpage.
- Since we use a MySQL database, we do not need to specify the interfaces to lexical domains in more detail. We use standardized SQL statements to access the database.

queryRate

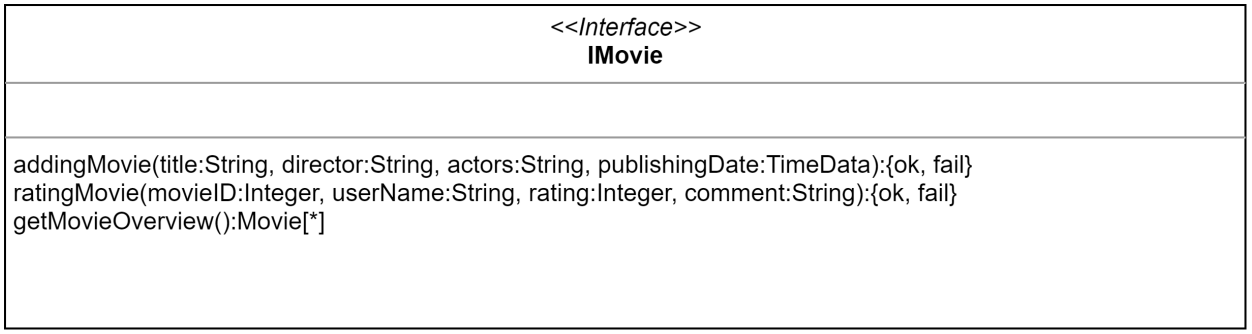
```
SELECT count(*) from Rates WHERE movieID="movieID" AND
username="username"
```

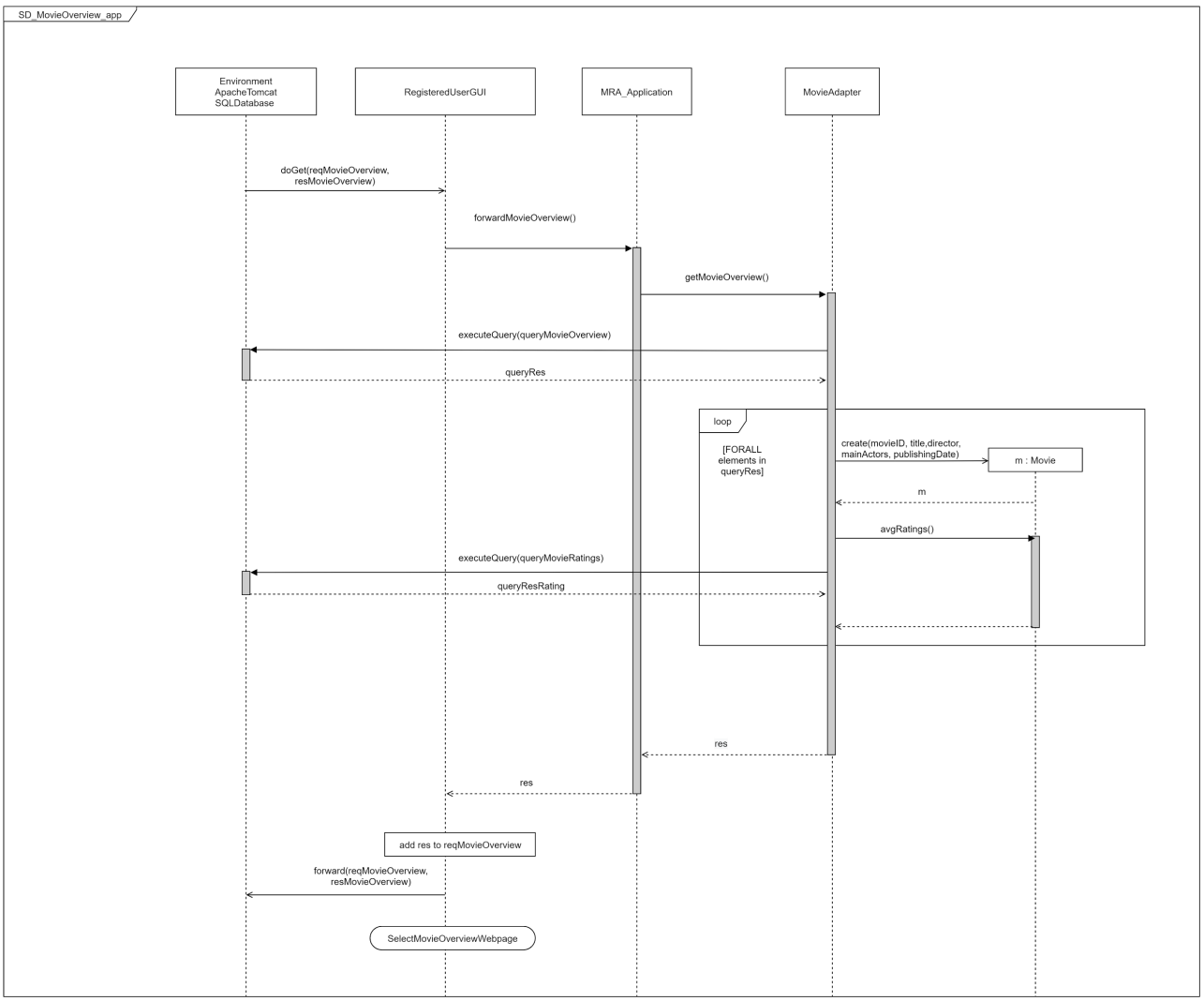
insertRatingMovie

```
INSERT INTO Rates (movieID, username, rating, comment) VALUES
("movieID", "username", "rating", "comment")
```

R4 - MovieOverview:

```
context MRA_Movie::forwardShowMovieOverview()
pre: true
post:
let res: OrderedSet(Movie)=
     m->select()->asOrderedSet()->sortedBy( mo: Movie |
     mo.avgRatings(mo.ratings)) in
     wpmo^returnedMovieOverview(res)
```





- *reqMovieOverview* represents HttpServletRequest objects.
- *resMovieOverview* represents HttpServletResponse objects as the counterpart for the request.
- *m* refers to an object of class *Movie*.
- *avgRatings(...)* calculates the average of the ratings given by each user for a movie.
- *res* is the data about movies, which includes the average ratings and overview.
- *forward(...)* sends the request and response back to the server to generate the HTML webpage.
- The state predicate *SelectMovieOverviewWebpage* represents that *WebpageMovieOverview* is selected.
- 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.

queryMovieOverview
`SELECT * from Movies`

queryMovieRatings
`SELECT avg(rating) from Rates WHERE movieID="movieID"`

Validation for D2

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

- Consistency of SD_Register_app and SD_Register

Message in D2	Corresponding message in A3
doGet(reqRegister,resRegister)	refines register
forward(reqRegister,resRegister)	refines register
doPost(reqRegistering,resRegistering)	refines register
forwardRegister(. . .)	forwardRegister
registering(. . .)	registering
executeQuery(queryUsername)	refines registering
executeUpdate(insertRegistering)	refines registering
forward(reqRegistering,resRegistering)	refines confirmRegisterRepresentation
forward(reqRegistering,resRegistering)	refines denyRegisterRepresentation

- Consistency of SD_AddMovie_app and SD_AddMovie

Message in D2	Corresponding message in A3
doGet(reqAddMovie,resAddMovie)	refines addMovie
forward(reqAddMovie,resAddMovie)	refines addMovie
doPost(reqAddingMovie,resAddingMovie)	refines addMovie
forwardAddMovie(. . .)	forwardAddMovie
addingMovie(. . .)	addingMovie
executeQuery(queryMovie)	refines addingMovie
executeUpdate(insertAddingMovie)	refines addingMovie
forward(reqAddingMovie,resAddingMovie)	refines confirmAddMovieRepresentation

forward(reqAddingMovie,resAddingMovie)	refines denyAddMovieRepresentation

- Consistency of SD_RateMovie_app and SD_RateMovie

Message in D2	**Corresponding message in A3**
doGet(reqRateMovie,resRateMovie)	refines rateMovie
forward(reqRateMovie,resRateMovie)	refines rateMovie
doPost(reqRatingMovie,resRatingMovie)	refines rateMovie
forwardRateMovie(. . .)	forwardRateMovie
ratingMovie(. . .)	ratingMovie
executeQuery(queryRate)	refines ratingMovie
executeUpdate(insertRatingMovie)	refines ratingMovie
forward(reqRatingMovie,resRatingMovie)	refines confirmRateMovieRepresentation
forward(reqRatingMovie,resRatingMovie)	refines denyRateMovieRepresentation

- Consistency of SD_MovieOverview_app and SD_MovieOverview

Message in D2	**Corresponding message in A3**
doGet(reqMovieOverview,resMovieOverview)	refines showMovieOverview
forwardShowMovieOverview(. . .)	forwardShowMovieOverview
getMovieOverview(. . .)	getMovieOverview
executeQuery(queryMovieOverview)	refines getMovieOverview
executeUpdate(insertMovieRatings)	refines getMovieOverview
avgRatings()	refines getMovieOverview
forward(reqMovieOverview,resMovieOverview)	refines returnedMovieOverviewRepresentation

- Consistency with life-cycle:
  - *Register*:

SD_Register_app can be executed without a precondition.

- $AddMovie$ and $ShowMovieOverview$:

SD_AddMovie_app and SD_MovieOverview_app can both be executed without a precondition.

- *(ShowMovieOverview ; [RateMovie]):*

At the end of SD_MovieOverview_app RegisteredUserGUI is in the state SelectMovieOverviewWebpage, such that the precondition of SD_RateMovie_app is fulfilled. Hence, SD_MovieOverview_app and SD_RateMovie_app can be executed in sequence as described by the life-cycle.

- $(AddMovie \mid (ShowMovieOverview ; [RateMovie]))^*$:

The registered user can execute SD_AddMovie_app or SD_MovieOverview_app without a precondition. SD_RateMovie_app can only be executed after SD_MovieOverview_app, but it's optional. These executions can take place an arbitrary amount of times.

- $Register \mid\mid (AddMovie \mid (ShowMovieOverview ; [RateMovie]))^*$:

SD_Register_app can be executed concurrently with SD_AddMovie_app or the sequence of SD_MovieOverview_app and SD_RateMovie_app without unwanted side effects.

- The Sequence diagrams must realize the operations described in step A5:
  Operation and data specification

- forwardRegister(...) is realized in SD_Register_app

  - **Precondition** does not have to be established, because it is true.
  - **Postcondition** MRA_Application delegates the message to DB_Adapter. Using the SQL command queryUsername, DB_Adapter checks if the username already exists in the Database for the given parameter "username". Then the DB_Adapter checks the result of the query to be "0" and the given parameter "age" is 18 or higher. If the two conditions are met, then the input parameters will be inserted into the database using the SQL command insertRegistering and a confirmation message will be forwarded to the MRA_Application. If the two conditions are NOT met a denying message will be forwarded to *MRA_Application. MRA_Application* forwards the result to UnregisteredUserGUI.
    That realizes wpr^confirmRegister() Or wpr^denyRegister()

- forwardAddMovie (...) is realized in SD_AddMovie_app

  - **Precondition** does not have to be established, because it is true.
  - **Postcondition** MRA_Application delegates the message to MovieAdapter. Using the SQL command queryMovie, MovieAdapter checks if the movie already exists in the Database for the given parameter "movie_id". Then the MovieAdapter checks the result of the query to be "0". If the condition is met, then the input parameters will be inserted into the database using the SQL command insertAddingMovie and a confirmation message will be forwarded to the MRA_Application. If the condition is not met, a denying message will be forwarded to MRA_Application. *MRA_Application* forwards the result to RegisteredUserGUI. MRA_Application forwards the result to RegisteredUserGUI. That realizes wpam^confirmAddMovie().

- forwardRateMovie (...) is realized in SD_RateMovie_app

- **Precondition** does not have to be established, because it is true.
- **Postcondition** MRA_Application delegates the message to MovieAdapter. Using the SQL command queryRate, MovieAdapter checks if the rating already exists in the Database by the given username and the given movie_id. Then the MovieAdapter checks the result of the query to be "0". If the condition is met, then the input parameters will be inserted into the database using the SQL command insertRatingMovie and a confirmation message will be forwarded to the MRA_Application. If the condition is not met, a denying message will be forwarded to MRA_Application. *MRA_Application* forwards the result to RegisteredUserGUI. MRA_Application forwards the result to RegisteredUserGUI. That realizes wpam^confirmRateMovie().

- forwardShowMovieOverview (...) is realized in SD_MovieOverview_app

  - **Precondition** does not have to be established, because it is true.
  - **Postcondition** MRA_Application delegates the message to MovieAdapter. Using the SQL command queryMovieOverview, MovieAdapter checks if the rating already exists in the Database by the given username and the given movie_id. Then the MovieAdapter checks the result of the query to be "0". If the condition is met, then the input parameters will be inserted into the database using the SQL command insertRatingMovie and a confirmation message will be forwarded to the MRA_Application. If the condition is not met, a denying message will be forwarded to MRA_Application. *MRA_Application* forwards the result to RegisteredUserGUI. MRA_Application forwards the result to RegisteredUserGUI. That realizes wpam^confirmRateMovie().

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

Interface	**Message**	**Used in sequence diagram**
UUCmds	forwardRegister confirmRegister denyRegister	SD_Register_app SD_Register_app SD_Register_app
IDatabase	registering	SD_Register_app
RUCmds	forwardAddMovie confirmAddMovie forwardRateMovie denyAddMovie forwardShowMovieOverview returnedMovieOverview confirmRateMovie forwardRateMovie	SD_AddMovie_app SD_AddMovie_app SD_RateMovie_app SD_AddMovie_app SD_MovieOverview_app SD_MovieOverview SD_RateMovie_app SD_RateMovie_app
IMovie	allMovieList addingMovie userRatingOfMovie ratingMovie	SD_AddMovie_app SD_AddMovie_app SD_RateMovie_app SD_RateMovie_app

	getMovieOverview MovieOverview	SD_MovieOverview SD_MovieOverview

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

Interface	**Provided by**	**Required by**
Message	**Recipient**	**Sender**
UUCmds	MRA_Application	UnregisteredUserGUI
forwardRegister confirmRegister denyRegister	MRA_Application MRA_Application MRA_Application	UnregisteredUserGUI UnregisteredUserGUI UnregisteredUserGUI
IDatabase	DB_Adapter	MRA_Application
registering	DB_Adapter	MRA_Application
RUCmds	MRA_Application	RegisteredUserGUI
forwardAddMovie confirmAddMovie forwardRateMovie denyAddMovie forwardShowMovieOverview returnedMovieOverview confirmRateMovie forwardRateMovie	MRA_Application MRA_Application MRA_Application MRA_Application MRA_Application MRA_Application MRA_Application MRA_Application	RegisteredUserGUI RegisteredUserGUI RegisteredUserGUI RegisteredUserGUI RegisteredUserGUI RegisteredUserGUI RegisteredUserGUI RegisteredUserGUI
IMovie	MovieAdapter	MRA_Application
allMovieList addingMovie userRatingOfMovie ratingMovie getMovieOverview MovieOverview	MovieAdapter MovieAdapter MovieAdapter MovieAdapter MovieAdapter MovieAdapter	MRA_Application MRA_Application MRA_Application MRA_Application MRA_Application MRA_Application

- Messages must connect components as connected in the software architecture of step D1:Software Architecture.

Component	**Connected components in architecture**	**Connected components in the sequence diagram**
MRA_Application	UnregisteredUserGUI, RegisteredUserGUI, DB_Adapter, MovieAdapter	UnregisteredUserGUI, RegisteredUserGUI, DB_Adapter, MovieAdapter
DB_Adapter	MRA_Application, Environment	MRA_Application, Environment
MovieAdapter	MRA_Application, Environment	MRA_Application, Environment
UnregisteredUserGUI	MRA_Application, Environment	MRA_Application, Environment
RegisteredUserGUI	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.

D3 Intra-Component Interaction

Intra-component Interaction for R1

There is no need to specify an intra-component interaction. The DB_Adapter is not complicated and we do not need to split it into sub-components. There is no decomposition of the components defined in Step D1. There is nothing to refine.

Intra-component Interaction for R2

There is no need to specify an intra-component interaction. The MovieAdapter is not complicated and we do not need to split it into sub-components. There is no decomposition of the components defined in Step D1. There is nothing to refine.

Intra-component Interaction for R3

There is no need to specify an intra-component interaction. The MovieAdapter is not complicated and we do not need to split it into sub-components. There is no decomposition of the components defined in Step D1. There is nothing to refine.

Intra-component Interaction for R4

There is no need to specify an intra-component interaction. The MovieAdapter is not complicated and we do not need to split it into sub-components. There is no decomposition of the components defined in Step D1. There is nothing to refine.

D4 State Machine

Check whether a state machine is necessary:

- 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 group member. It is not necessary to create a state machine for this component.

- The component DB_Adapter: no refinement exists in Step D3 continues with looking at Step D2. There are less than two states. Therefore, no state machine is necessary.

- The component MovieAdapter: no refinement exists in Step D3 continue with looking at Step D2. There are less than two states. Therefore, no state machine is necessary.

- The component UnregisteredUserGUI: 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.

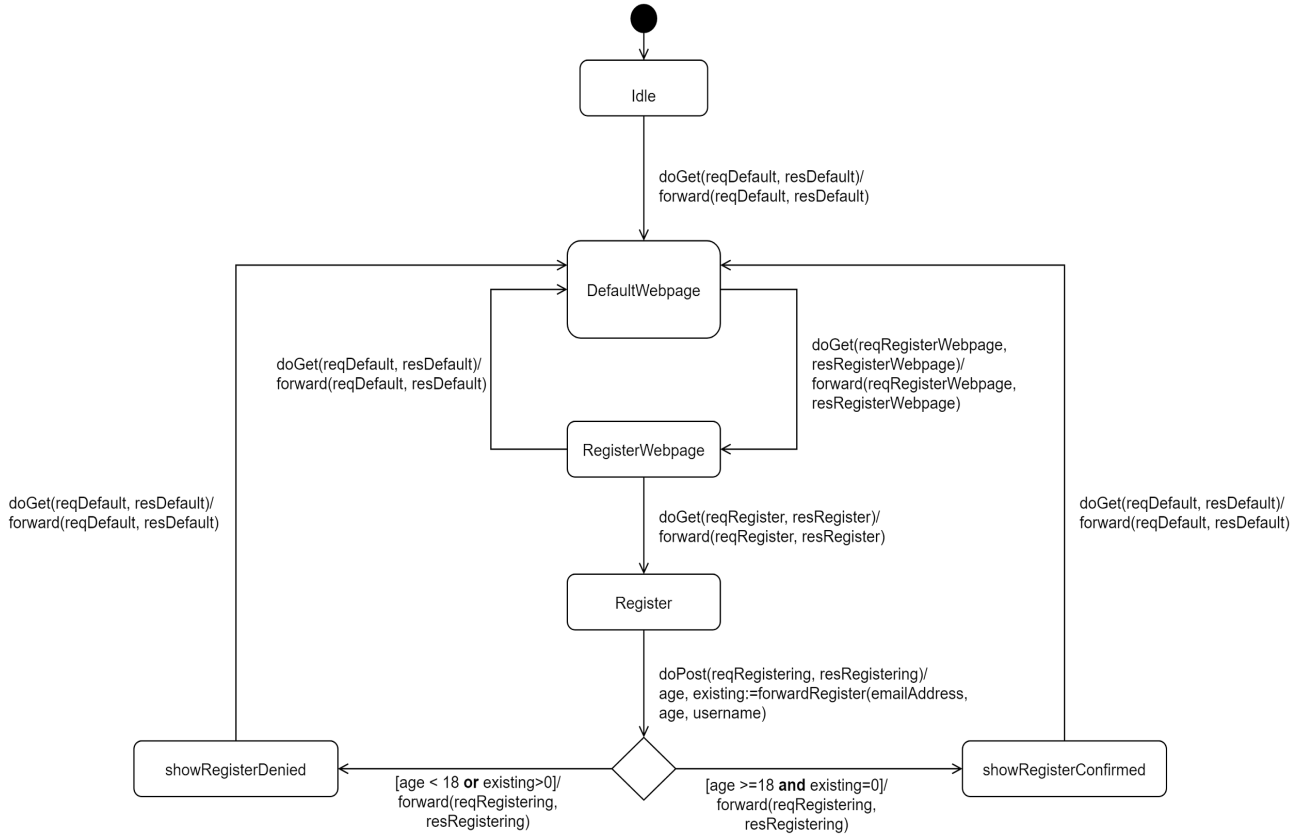
- The component RegisteredUserGUI: no refinement exists in Step D3: Intra-Component Interaction; continue with looking at Step D2: Inter-Component Interaction. There are more than two states. Therefore, a state machine is required.

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

Considering UnregisteredUserGUI:

- messages doGet(...) and doPost(...) must be triggers attransitions
- messages forwardRegister(...), and forward(...) must be output signals or actions
- state predicates RegisterWebpage, Register, ShowRegisterDenied, and ShowRegisterConfirmed must be states
- Idle and DefaultWebpage are new states required to represent the functionalities of the web application

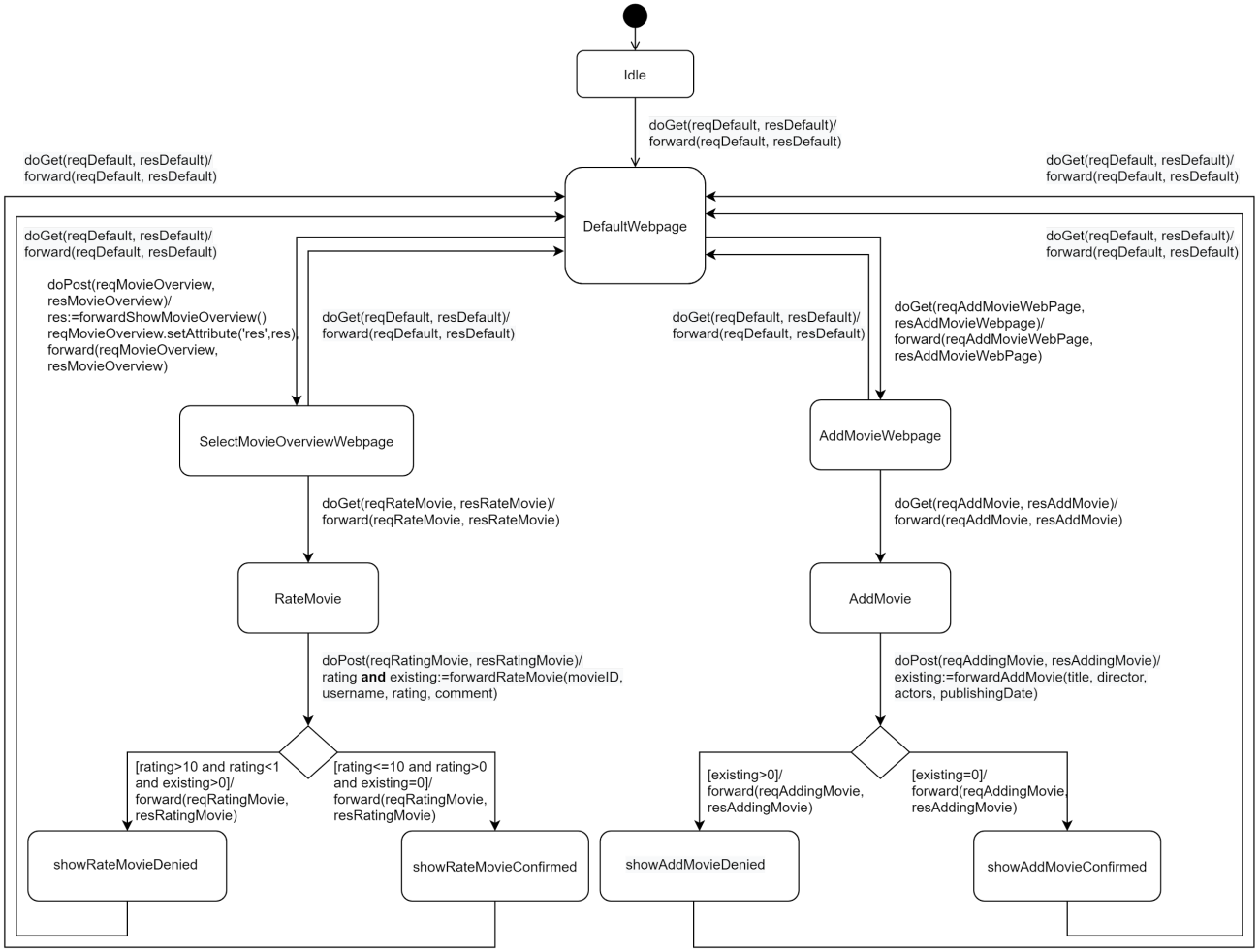
State Machine for UnregisteredUserGUI



Considering RegisterUserGUI:

- messages doGet(...) and doPost(...) must be triggers attransitions
- messages forwardAddMovie(...), forwardRateMovie(...), forwardShowMovieOverview(...) and forward(...) must be output signals or actions
- state predicates AddMovieWebpage, AddMovie, ShowAddMovieDenied, and ShowAddMovieConfirmed, SelectMovieOverviewWebpage, RateMovie, ShowRateMovieDenied, and ShowRateMovieConfirmed must be states
- Idle and DefaultWebpage are new states required to represent the functionalities of the web application

State Machine for RegisteredUserGUI



Validation for D4 (UnregisteredUserGUI)

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

Component UnregisteredUserGUI					
Source State	Target State	Input Signal	*Mapped to message(s)*	Output Signal	*Mapped to message(s)*
Init	DefaultWebpage	doGet(reqDefault…)	-	forward(reqDefault…)	-
DefaultWebpage	RegisterWebpage	doGet(reqRegisterWebpage…)	doGet(reqRegisterWebpage…)	forward(reqRegisterWebpage...)	forward(reqRegisterWebpage...)
RegisterWebpage	DefaultWebpage	doGet(reqDefault…)	-	forward(reqDefault…)	-
RegisterWebpage	Register	doGet(reqRegister…)	doGet(reqRegister…)	forward(reqRegister...)	forward(reqRegister...)
Register	showRegisterDenied	doPost(reqRegistering…)	doPost(reqRegistering…)	forwardRegister(emailAddress, ...) forward(reqRegistering, …)	forwardRegister(reqRegistering, …) forward(reqRegistering, …)
Register	showRegisterConfirmed	doPost(reqRegistering…)	doPost(reqRegistering…)	forwardRegister(emailAddress, ...) forward(reqRegistering, …)	forwardRegister(reqRegistering, …) forward(reqRegistering, …)
ShowRegisterDenied	DefaultWebpage	doGet(reqDefault…)	-	forward(reqDefault…)	-
ShowRegisterConfirmed	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 UnregisteredUserGUI	
$LC_{UnregisteredUser} = Register$	
State	**Covered by Lifecycle Part**
Init	-
DefaultWebpage	-
RegisterWebpage	-
Register	Register
ShowRegisterDenied	Register
ShowRegisterConfirmed	Register

- All transitions are covered by a life-cycle.

Component UnregisteredUserGUI				
$LC_{UnregisteredUser} = Register$				
Source State	Target State	Input Signal	Output Signal	*Life cycle part*
Init	DefaultWebpage	doGet(reqDefault…)	forward(reqDefault…)	-
DefaultWebpage	RegisterWebpage	doGet(reqRegisterWebpage…)	forward(reqRegisterWebpage...)	-
RegisterWebpage	DefaultWebpage	doGet(reqDefault…)	forward(reqDefault…)	-
RegisterWebpage	Register	doGet(reqRegister…)	forward(reqRegister...)	Register
Register	showRegisterDenied	doPost(reqRegistering…)	forwardRegister(emailAddress, ...) forward(reqRegistering, …)	Register
Register	showRegisterConfirmed	doPost(reqRegistering…)	forwardRegister(emailAddress, ...) forward(reqRegistering, …)	Register
ShowRegisterDenied	DefaultWebpage	doGet(reqDefault…)	forward(reqDefault…)	Register
ShowRegisterConfirmed	DefaultWebpage	doGet(reqDefault…)	forward(reqDefault…)	Register

Validation for D4 (RegisteredUserGUI)

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

Component RegisteredUserGUI					
Source State	**Target State**	**Input Signal**	***Mapped to message(s)***	**Output Signal**	***Mapped to message(s)***
Init	DefaultWebpage	doGet(reqDefault…)	-	forward(reqDefault…)	-
DefaultWebpage	AddMovieWebpage	doGet(reqAddMovieWebpage…)	doGet(reqAddMovieWebpage…)	forward(reqAddMovieWebpage…)	forward(reqAddMovieWebpage…)
DefaultWebpage	SelectMovieOverviewWebpage	doPost(reqMovieOverview…)	doPost(reqMovieOverview…)	res:=forwardShowMovieOverview(...), reqMovieOverview.setAttribute(...), forward(reqMovieOverview,...)	forwardShowMovieOverview(...), forward(reqMovieOverview,...)
AddMovieWebpage	AddMovie	doGet(reqAddMovie…)	doGet(reqAddMovie…)	forward(reqAddMovie…)	forward(reqAddMovie…)
AddMovie	showAddMovieDenied	doPost(reqAddingMovie…)	doPost(reqAddingMovie…)	existing:=forwardAddMovie(...), forward(reqAddingMovie…)	forwardAddMovie(...), forward(reqAddingMovie…)
AddMovie	showAddMovieConfirmed	doPost(reqAddingMovie…)	doPost(reqAddingMovie…)	existing:=forwardAddMovie(...), forward(reqAddingMovie…)	forwardAddMovie(...), forward(reqAddingMovie…)
SelectMovieOverviewWebpage	RateMovie	doGet(reqRateMovie…)	doGet(reqRateMovie…)	forward(reqRateMovie, …)	forward(reqRateMovie, …)
RateMovie	showRateMovieDenied	doPost(reqRatingMovie…)	doPost(reqRatingMovie…)	rating and existing:=forwardRateMovie(...), forward(reqRatingMovie, …)	forwardRateMovie(...), forward(reqRatingMovie, …)
RateMovie	showRateMovieConfirmed	doPost(reqRatingMovie…)	doPost(reqRatingMovie…)	rating and existing:=forwardRateMovie(...), forward(reqRatingMovie, …)	forwardRateMovie(...), forward(reqRatingMovie, …)
showAddMovieDenied	DefaultWebpage	doGet(reqDefault…)	-	forward(reqDefault…)	-
showAddMovieConfirmed	DefaultWebpage	doGet(reqDefault…)	-	forward(reqDefault…)	-
showRateMovieDenied	DefaultWebpage	doGet(reqDefault…)	-	forward(reqDefault…)	-
showRateMovieConfirmed	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 UnregisteredUserGUI	
$LC_{RegisteredUser} = (AddMovie \mid (ShowMovieOverview\ ; [RateMovie]))^*$	
State	**Covered by Lifecycle Part**
Init	-
DefaultWebpage	-
AddMovieWebpage	$AddMovie$
AddMovie	$AddMovie$
SelectMovieOverviewWebpage	$ShowMovieOverview$
RateMovie	$RateMovie$
showAddMovieDenied	$AddMovie$
showAddMovieConfirmed	$AddMovie$
showRateMovieDenied	$RateMovie$
showRateMovieConfirmed	$RateMovie$

- All transitions are covered by a life-cycle.

Component RegisteredUserGUI				
$LC_{RegisteredUser} = (AddMovie \mid (ShowMovieOverview\ ; [RateMovie]))^*$				
Source State	**Target State**	**Input Signal**	**Output Signal**	**life cycle part**
Init	DefaultWebpage	doGet(reqDefault. . .)	forward(reqDefault. . .)	(
DefaultWebpage	AddMovieWebpage	doGet(reqAddMovieWebpage…)	forward(reqAddMovieWebpage…)	*AddMovie (| (ShowMovieOverview*
AddMovieWebpage	DefaultWebpage	doGet(reqDefault,. . .)	forward(reqDefault,. . .)	*AddMovie*
AddMovieWebpage	AddMovie	doGet(reqAddMovie…)	forward(reqAddMovie …)	$AddMovie$
AddMovie	showAddMovieDenied	doPost(reqAddingMovie…)	existing:=forwardAddMovie(...), forward(reqAddingMovie…)	$AddMovie$
AddMovie	showAddMovieConfirmed	doPost(reqAddingMovie…)	existing:=forwardAddMovie(...), forward(reqAddingMovie…)	$AddMovie$
showAddMovieDenied	DefaultWebpage	doGet(reqDefault,. . .)	forward(reqDefault,. . .)	)*
showAddMovieConfirmed	DefaultWebpage	doGet(reqDefault,. . .)	forward(reqDefault,. . .)	)*

DefaultWebpage	SelectMovieOverviewWebpage	doPost(reqMovieOverview…)	res:=forwardShowMovieOverview(...) ,reqMovieOverview.setAttribute(...), forward(reqMovieOverview, ...)	AddMovie (| (ShowMovieOverview
SelectMovieOverview Webpage	DefaultWebpage	doGet(reqDefault,. . .)	forward(reqDefault,. . .)	ShowMovieOverview
SelectMovieOverview Webpage	RateMovie	doGet(reqRateMovie,. . .)	forward(reqRateMovie,. . .)	ShowMovieOverview
RateMovie	showRateMovieDenied	doPost(reqRatingMovie…)	rating and existing:=forwardRateMovie(...), forward(reqRatingMovie)	RateMovie
RateMovie	showRateMovieConfirmed	doPost(reqRatingMovie…)	rating and existing:=forwardRateMovie(...), forward(reqRatingMovie)	RateMovie
showRateMovieDenied	DefaultWebpage	doGet(reqDefault,. . .)	forward(reqDefault,. . .)	[RateMovie])*
showRateMovieConfirmed	DefaultWebpage	doGet(reqDefault,. . .)	forward(reqDefault,. . .)	[RateMovie])*

Implementation & Testing

Application Tests:

- MovieRatingApplicationTest
  - testRegister()
    - confirms that it triggers the register() command correctly.
  - testAddMovie()
    - confirms that it triggers the addMovie() command correctly.
  - testRateMovie()
    - confirms that it triggers the rateMovie() command correctly.
  - testMovieOverview()
    - confirms that it triggers the movieOverview() command correctly.

T1 Component Tests

All component tests are implemented via _ _ _.java using PowerMockito and JUnit

- _ _ _test contains:

  testForwardRegister()
  - Tests that an unregistered user registered successfully and uniquely added to the database.

  testForwardAddMovie()
  - Tests that a registered user added a movie successfully and uniquely added to the movie database.

  testForwardRateMovie()
  - Tests that a registered user rate a movie successfully and uniquely added to the movie database.

  testForwardMovieOverview()
  - Test that if movie exist, then the result is returned based on average rating.

T2 System Tests

The following testing classes and methods included in this submission:

- UUGUIWebTestCaseAddMovie

  testForwardRegister()

  - Tests that the default website is fully displayed

8. Glossary

Name	Type	Description	Source
A			
addingUser	Phenomenon	add other registered users into a movie discussion group	Contextdiagram
addingMovie()	Auxiliary function	Creates a class with certain parameters	SD_AddMovie_app
AddMovie	state	Indicates that the Movie´s Addition has been started.	State Machine RegisteredUserGUI
addMovie	Phenomenon	add a movie if a movie is not yet contained in the database	Contextdiagram, pdAddMovie, sdAddMovie, Class Model
AddMovieWebpage	state	Indicates that the AddMovie web page is accessed	State Machine RegisteredUserGUI
addUser	Phenomenon	add other registered users into a movie discussion group	Contextdiagram
age	attribute	Represents the users age that need for successful registration	Class Model
ageOver18	Condition	the condition which shows that the age of the user is more than 18	sdRegister
allMovieList	Phenomenon	access a list of all movies in the database	Contextdiagram, pdAddMovie, sdAddMovie
api	technical phenomenon	methods provided by given libraries	TCD
ApacheTomcat	connection domain	An Open Source JSP and Servlet Container from the Apache Foundation.	TCD, SD_Register_app, SD_AddMovie_app, SD_RateMovie_app, SD_MovieOverview_app
avgRatings()	auxiliary function	Returns the Calculation of average rating of the Movie	Class Model, SD_MovieOverview_app
B			
banMember	Phenomenon	ban members from the group	Contextdiagram
banningMember	Phenomenon	ban members from the group	Contextdiagram
C			
confirmAddMovie	Phenomenon	feedback to confirm adding the movie	pdAddMovie, sdAddMovie, Class Model, SD_AddMovie_app
confirmAddMovieRepresentation	Phenomenon	representation of the feedback to confirm adding the movie	pdAddMovie, sdAddMovie
confirmRateMovie	Phenomenon	feedback to confirm rating the movie	pdRateMovie, sdRateMovie Class Model
confirmRateMovieRepresentation	Phenomenon	representation of the feedback to confirm rating the movie	pdRateMovie, sdRateMovie
confirmRegister	Phenomenon	feedback to confirm registration	pdRegister, sdRegister, Class Model, SD_Register_app
confirmRegisterRepresentation	Phenomenon	representation of the feedback to confirm registering the user	pdRegister, sdRegister

create()	Auxiliary function	Creates a class with certain parameters	SD_MovieOverview_app
createGroup	Phenomenon	create a movie discussion group	Contextdiagram
creatingGroup	Phenomenon	creating a movie discussion group	Contextdiagram
D			
Database	Designed Domain, Lexical Domain	database which holds necessary data	Contextdiagram, pdRegister Class Model
day	attribute	Represents day of a TimeData	Class Model
DB_Adapter	component	adapter responsible for the user database	subArchRegister, globalArch, SD_Register_app
DBAdapterPort	Port	Connects DB_Adapter to the machine	subArchRegister, globalArch
DBPort	Port	Connects IDataBase to BD_Adapter	subArchRegister, globalArch
DBPort_I	Port	Connects IDataBase to the application	subArchRegister
DefaultWebpage	state	Indicates the starting page.	State Machine UnregisteredUserGUI, State Machine RegisteredUserGUI
deletingStaleGroup	Phenomenon	delete a group with one member after a period of time	Contextdiagram
denyAddMovie	message	feedback to adding the movie	pdRateMovie sdAddMovie, Class Model
denyAddMovieRepresentation	message	representation of the feedback to deny adding the movie	pdRateMovie sdAddMovie
denyRateMovie	message	feedback to rate the movie	pdRateMovie sdRateMovie Class Model, SD_RateMovie_app
denyRateMovieRepresentation	message	representation of the feedback to deny rating the movie	pdRateMovie sdRateMovie
denyRegister	message	feedback to deny registration	sdRegister, Class Model, SD_Register_app
denyRegisterRepresentation	message	representation of the feedback to deny registering the user	pdRateMovie sdRegister
director	attribute	Represents the directors name of a Movie that we adding to a Movie Database	Class Model
doGet	technical phenomenon	Operations doGet is defined in abstract class javax.servlet.http.HttpServlet (https://docs.oracle.com/javaee/7/api/javax/ servlet/http/HttpServlet.html)	TCD, SD_Register_app, SD_AddMovie_app, SD_RateMovie_app, SD_MovieOverview_app State Machine UnregisteredUserGUI, State Machine RegisteredUserGUI
doPost	technical phenomenon	Operations doPost is defined in abstract class javax.servlet.http.HttpServlet (https://docs.oracle.com/javaee/7/api/javax/ servlet/http/HttpServlet.html)	TCD, SD_Register_app, SD_AddMovie_app, SD_RateMovie_app, SD_MovieOverview_app State Machine UnregisteredUserGUI, State Machine RegisteredUserGUI

E			
emailAddress	attribute	Represents the email address that need for successful registration	Class Model
executeQuery	technical phenomenon	Java API function to send an SQL query command to a MySQL database.	TCD, SD_Register_app, SD_AddMovie_app, SD_RateMovie_app, SD_MovieOverview_app
executeUpdate	technical phenomenon	Java API function to send an SQL query command to a MySQL database.	TCD, SD_Register_app, SD_AddMovie_app, SD_RateMovie_app
existing	condition	condition if movie already exists in the Movie Database	SD_AddMovie_app, SD_RateMovie_app
F			
fail	Phenomenon	the response from the database, when exists condition is false for SD_AddMovie_app and SD_RateMovie_app, and also when age >18 or queryRes= true for SD_Register_app	SD_Register_app, SD_AddMovie_app, SD_RateMovie_app
feedbackGA	Phenomenon	introduced to provide feedback to Group Admin	Contextdiagram
feedbackGM	Phenomenon	introduced to provide feedback to Group Member	Contextdiagram
feedbackRU	Phenomenon	introduced to provide feedback to Registered User	Contextdiagram
feedbackUU	Phenomenon	introduced to provide feedback to Unregistered User	Contextdiagram
forward	technical Phenomenon	Operation forward defined in interface javax.servlet.RequestDispatcher (http://docs.oracle.com/javaee/7/api/javax/ servlet/RequestDispatcher.html)	TCD, SD_Register_app, SD_AddMovie_app, SD_RateMovie_app, SD_MovieOverview_app State Machine UnregisteredUserGUI, State Machine RegisteredUserGUI
forwardAddMovie	Phenomenon	forward add movie request from the user to the machine	pdAddMovie, sdRegister Class Model, SD_AddMovie_app, State Machine RegisteredUserGUI
forwardRateMovie	Phenomenon	forward rate movie request from the user to the machine	pdRateMovie, sdRateMovie, Class Model, State Machine RegisteredUserGUI
forwardRegister	Phenomenon	forward the register request from the user to the machine	pdRegister, sdRegister, Class Model, SD_Register_app State Machine UnregisteredUserGUI
forwardShowMovieOverview	Phenomenon	forward show movie request from the user to the machine	pdMovieOverview, sdMovieOverview

			Class Model
G			
getAllMovieList	message	getter function which gets every movie's name	sdAddMovie
getMovieOverview	message	getter function which gets the movie overview	sdMovieOverview, SD_MovieOverview_app
getUserRatingOfMovie	message	getter function which gets the rating given by the user for a movie	sdRateMovie
getUsersList	phenomenon	(merged with phenomenon registering) getter function which gets every user's username	sdRegister
Group	Designed Domain, Lexical Domain	movie discussion group	Contextdiagram
GroupAdmin	Biddable Domain	group member who is also the admin of the movie discussion group	Contextdiagram
GroupMember	Biddable Domain	user who is also a member of a movie discussion group	Contextdiagram
GUI	interface	User interface of HTML page	TCD
H			
HTTP	technical phenomenon	defined in RFC 2616, (Network Working Group, 1999)	TCD
I			
IDataBase	interface	provides operations and functions defined by DB_Adapter	subArchRegister, globalArch
Idle	state	Indicates that the server waits for incoming requests.	State Machine UnregisteredUserGUI, State Machine RegisteredUserGUI
IMovie	interface	provides operations and functions defined by M_Adapter	subArchAddMovie, subArchRateMovie, subArchMovieOverview, globalArch
J			
K			
L			
$LC_{MovieRatingApp}$	life-cycle	Combined life-cycle (all users and the internal operation)	LC
$LC_{RegisterUser}$	life-cycle	Life-cycle for registerUser	LC
$LC_{UnregisterUser}$	life-cycle	Life-cycle for unregisterUser	LC
leaveGroup	Phenomenon	leave the movie discussion group	Contextdiagram
leavingGroup	Phenomenon	leaving the movie discussion group	Contextdiagram
loggingIn	Phenomenon	logging in using his/her email address and username	Contextdiagram
loggingOut	Phenomenon	logging out if he/she wants to end the session	Contextdiagram
logIn	Phenomenon	log in using his/her email address and username	Contextdiagram
logOut	Phenomenon	log out if he/she wants to end the session	Contextdiagram
M			
mainActors	attribute	Represents the actors name of a Movie that we adding to a Movie Database	Class Model
MemberMoviesList	Phenomenon	see the movie lists of other members	Contextdiagram
MessagesList	Phenomenon	see messages in the group chat	Contextdiagram
Mexists	State Predicate	state showing that the movie is created	sdAddMovie,

		and exists	sdRateMovie
month	attribute	Represents month of a TimeData	Class Model
Movie	Designed Domain, Lexical Domain	movie database in the MovieRatingApp	Contextdiagram, pdAddMovie, pdRateMovie, pdMovieOverview Class Model
MovieAdapter	component	adapter responsible for the movie database	subArchAddMovie, subArchRateMovie, subArchMovieOver view, globalArch, SD_AddMovie_app, SD_RateMovie_app, SD_MovieOverview _app
MovieAdapterPort	Port	Connects MovieAdapter to the machine	subArchAddMovie, subArchRateMovie, subArchMovieOver view, globalArch
movieID	attribute	Represents the each of Movie we need to to rate in the a Movie Database	Class Model
movieOverview	Phenomenon	showing overview of the movie in the database	Contextdiagram, pdMovieOverview, sdMovieOverview
MovieRatingApp	Machine Domain	the software to be developed	Contextdiagram, TCD
MPort	Port	Connects IMovie to MovieAdapter	subArchAddMovie, subArchRateMovie, subArchMovieOver view, globalArch
MPort_I	Port	Connects IMovie to the application	subArchAddMovie, subArchRateMovie, subArchMovieOver view
MRA_AddMovie	Machine Domain	the part of the machine for adding a movie	pdAddMovie sdAddMovie Class Model
MRA_Application	Component	Main application which controls the interaction between the other components	subArchRegister, subArchAddMovie, subArchRateMovie, subArchMovieOver view, globalArch, SD_Register_app, SD_AddMovie_app, SD_RateMovie_app, SD_MovieOverview _app
MRA_MovieOverview	Machine Domain	the part of the machine for showing the movie overview	pdMovieOverview sdMovieOverview Class Model
MRA_RateMovie	Machine Domain	the part of the machine for rating a movie	pdRateMovie sdRateMovie Class Model
MRA_Register	Machine Domain	the part of the machine for registering	pdRegister sdRegister Class Model
Mrated	State Predicate	state showing that the movie is rated	sdRateMovie
N			
O			
ok	Phenomenon	the response from the database, when exists condition is true for SD_AddMovie_app and SD_RateMovie_app, and also when age	SD_Register_app, SD_AddMovie_app, SD_RateMovie_app

		<18 or queryRes= false for SD_Register_app	
P			
publishingDate	attribute	Represents the publishing Date of a Movie that we adding to a Movie Database	Class Model
Q			
queryRes			
R			
RateMovie	state	Indicates that a movie´s evaluation has been started.	State Machine RegisteredUserGUI
rateMovie	Phenomenon	rate the movies in the database and add to the watched list	Contextdiagram, pdRateMovie, sdRateMovie Class Model
Rating	class	represents a rating given by a registered user for a movie	Class Model
rating	attribute	Represents the Movie rating in the Movie Database	Class Model
ratingMovie	Phenomenon	rating the movies in the database and add to the watched list	Contextdiagram, pdRateMovie, sdRateMovie
Register	state	Indicates that a registration is started.	State Machine UnregisteredUserGUI
register	Phenomenon	register to the app	Contextdiagram, pdRegister, sdRegister Class Model
RegisterWebPage	state	Indicates that the Register web page is accessed	State Machine UnregisteredUserGUI
RegisteredUser	Biddable Domain	user who has registered to the app	Contextdiagram, pdAddMovie, pdRateMovie, pdMovieOverview TCD
RegisteredUserGUI	component	web interface for registered users	subArchAddMovie, subArchRateMovie, subArchMovieOverview, globalArch, SD_AddMovie_app, SD_RateMovie_app, SD_MovieOverview_app
RegisteredUserPort	Port	Connects the RegisteredUserGUI to the machine	subArchAddMovie, subArchRateMovie, subArchMovieOverview, globalArch
RegisteredUserWebBrowser	connection domain	Web browser used by registered users	TCD
registering	Phenomenon	registering to the app	Contextdiagram, pdRegister, sdRegister, SD_Register_app
registrationData	Phenomenon	It holds the registration data in the connection domain	pdRegister
returnedMovieOverview	Phenomenon	a phenomenon to return the movie from the database	pdMovieOverview, sdMovieOverview Class Model
returnedMovieOverviewRepresentation	Phenomenon	a representation of the movie returned from the database	pdMovieOverview, sdMovieOverview

Rinvalid	Condition	the condition which shows that the rating which is provided by the user is invalid not a number between one to ten	sdRateMovie
RUCmds	Interface	Provides operations and functions for interaction with RegisteredUserGUI	subArchAddMovie, subArchRateMovie, subArchMovieOverview, globalArch
RUCmdsPort	Port	Connects RUCmds to the application	subArchAddMovie, subArchRateMovie, subArchMovieOverview, globalArch
RUCmdsPort_I	Port	Connects RUCmds to RegisteredUserGUI	subArchAddMovie, subArchRateMovie, subArchMovieOverview, globalArch
S			
SelectMovieOverviewWebPage	state	Indicates that a MovieOverview web page is accessed	State Machine RegisteredUserGUI
sendingMessage	Phenomenon	counterpart to send message in the group chat	Contextdiagram
sendMessage	Phenomenon	send message in the group chat	Contextdiagram
showAddMovieConfirmed	state	Indicates that the movie has been added.	State Machine RegisteredUserGUI
showAddMovieDenied	state	Indicates that the movie has not been added.	State Machine RegisteredUserGUI
showAllMovieList	Phenomenon	access a list of all movies in the database	Contextdiagram
showMemberMoviesList	Phenomenon	see the movie lists of other members	Contextdiagram
showMessagesList	Phenomenon	see messages list in the group chat	Contextdiagram
showMovieOverview	Phenomenon	show overview of the movie in the database	Contextdiagram, pdMovieOverview, sdMovieOverview, Class Model, State Machine RegisteredUserGUI
showRateMovieConfirmed	state	Indicates that the movie has been rated.	State Machine RegisteredUserGUI
showRateMovieDenied	state	Indicates that the movie has not been rated.	State Machine RegisteredUserGUI
showRegisterConfirmed	state	Indicates that registration has been completed.	State Machine UnregisteredUserGUI
showRegisterDenied	state	Indicates that registration has been denied.	State Machine UnregisteredUserGUI
sql	technical phenomenon	Use of SQL statements to retrieve data from the database	TCD
SQLDatabase	casual Domain	Database: Realized as SQLDatabase on the same computer as the machine. Therefore, the database is connected by a call-and-return interface and used with SQL commands.	TCD
T			
title	attribute	Represents the title of a Movie that we add to a Movie Database	Class Model
U			
uniqueUsername	Condition	the condition which shows that the username of the user is not same with any other username in the database	sdRegister
UnregisteredUser	Biddable Domain	user who has not registered to the app	Contextdiagram, pdRegister TCD

UnregisteredUserGUI	component	web interface for unregistered users	subArchRegister, globalArch, SD_Register_app
UnregisteredUserPort	Port	Connects the UnregisteredUserGUI to the machine	subArchRegister, globalArch
UnregisteredUserWebBrowser	connection domain	Web browser used by unregistered users	TCD
userName	attribute	Represents the username that need for successful registration	Class Model
UserRatingOfMovie	Phenomenon	the rating given by the user for a movie	ContextDiagram, pdRateMovie, sdRateMovie
UUCmds	Interface	Provides operations and functions for interaction with UnregisteredUserGUI	subArchRegister, globalArch
UUCmdsPort	Port	Connects UUCmds to the application	subArchRegister, globalArch
UUCmdsPort_I	Port	Connects UUCmds to UnregisteredUserGUI	subArchRegister, globalArch
UURegistered	State Predicate	state when UnregisteredUser gets registered	sdRegister
V			
W			
WebPageAddMovie	connection Domain	web page to register the user	pdAddMovie sdAddMovie Class Model
WebPageMovieOverview	connection Domain	web page to view movie overview	pdMovieOverview sdMovieOverview Class Model
WebPageRateMovie	connection Domain	web page to rate the movie	pdRateMovie sdRateMovie Class Model
WebPageRegister	connection Domain	web page to register the user	pdRegister sdRegister Class Model
X			
Y			
year	attribute	Represents year of a TimeData	Class Model
Z			

Table 4.1: Glossary