UNIVERSITÄT DUISBURG ESSEN

Open-Minded

Lab for Software Engineering
Winter Term 2021/2022

Lab for Software Engineering

Movie Rating Application

Majdi Maalej (3099671)
Muhammad Noman Khan (3085671)
Suman Timsina (3040954)
Roozbeh Haghiri (3099637)
Tantchou Franck (3088206)
Kafo Tsakou German (3099523)

February 13, 2022

University of Duisburg-Essen · Faculty of Engineering Working group Software Engineering – Prof. Dr. M. Heisel
Date: Do. 16-18 - Supervisor: Julien Lukasewycz

Contents

List of Figures

1 Analysis

1.1 A1

1.1.1 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 user name. To register, a person must be at least eighteen years old, otherwise the registration fails.

R3 The user name has to be unique.

R4 After the registration, the user can log in using his/her email address and user name to use the functionality of the app and log out if he/she wants to end the session.

R5 A logged in user can add movies he/she has watched from a database to his/her list and rate them from 1 to 10 with a optional comment.

R6 If a value differing from one to ten is entered, the rating process will fail.

R7 If no rating is entered, the movie is rated as zero.

R8 A user cannot give more than one rating per movie.

R9 If a movie is not yet contained in the database, the user can add it by providing the title, director, main actors (at least one) and original publishing date.

R10 Additionally, users can access a list of all movies in the database sorted by rating in descending order. For each movie in the database, the average rating is calculated and shown together with the comments.

R11 Registered users can add other registered users into a movie discussion group. A member of the group can leave it if he/she wants to.

R12 A group has a unique name.

R13 Within a group, members can see the movie lists of other members and can have a discussion in form of a group chat.

R14 In the chat, the messages are sorted by the order of their creation.

R15 The creator of the group is its administrator. He/she can ban members from the group if he/she thinks that the member is misbehaving. When the administrator leaves the group, the group is deleted.

R16 If a group consists only of one member, an automated method will delete it after a certain amount of time.

Facts

F1 A rating consists out of a mandatory point rating from one to ten (1 = very bad movie, 10 = excellent movie) and optionally a written comment.

F2 Each movie has a title, a director, at least one main actor and an original publishing date.

Assumptions

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

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

A3 Users only rate movies they have really watched, and their rating is based only on their own opinions.

A4 The administrator's decisions are always fair. In groups, members only discuss movie related topics.

1.1.2 Contextdiagram

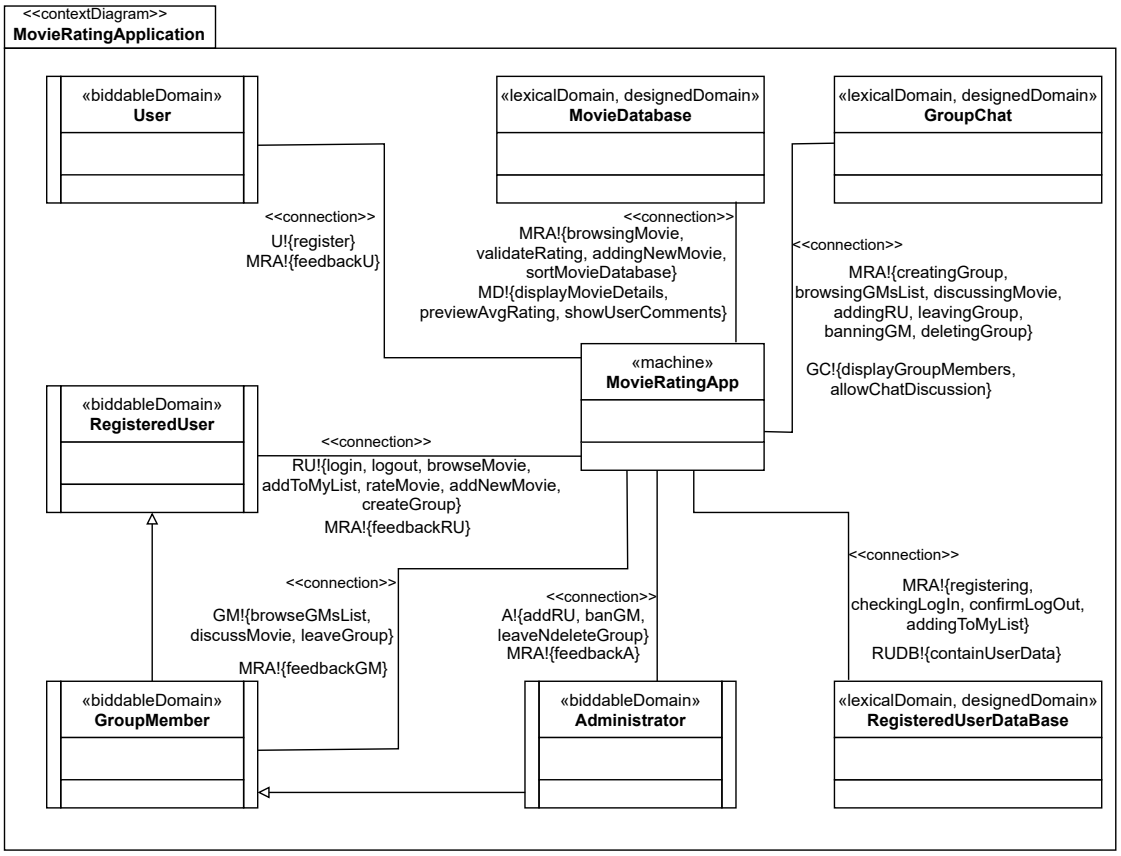


Figure 1.1: Contextdiagram

1.1.3 Validation

V1 The glossary contains the notions used in R and D.
The notions mentioned in R and D are contained in the glossary.

V2 Domains and phenomena of the context diagram must be consistent with R and D.

Notion in CD	**Notions in R/D**	**Type**
User	person	domain
register	user can register providing Email, age and a username	phenomenon
feedbackU	introduced to provide feedback to users if the registration was successful	phenomenon
RegisteredUser	user/registered user	domain
login, logout	can login and log out	phenomenon
browseMovie	can access a list of all movies in the database sorted by rating in descending order	phenomenon
addToMyList, rateMovie	can add movies to his/her list and rate them	phenomenon
addNewMovie	if a movie is not yet contained in the database, the user can add it	phenomenon
createGroup	create a group	phenomenon
feedbackRU	introduced to provide feedback to registered users	phenomenon
GroupMember	group member	domain
browseGMsList	can see the movie lists of other members	phenomenon
discussMovie	can have a discussion in form of a group chat	phenomenon
leaveGroup	can leave a group if he/she wanto to	phenomenon
feedbackGM	introduced to provide feedback to group members	phenomenon
Administrator	administrator	domain
addRU	registered users can add other registered users	phenomenon
banGM	can ban members from the group	phenomenon
leaveNdeleteGroup	when the administrator leaves, the group is deleted	phenomenon
feedbackA	introduced to provide feedback to administrators	phenomenon
MovieDatabase	database that is designed to store all movies related informations	domain
browsingMovie	counterpart to browseMovie	phenomenon
validateRating	counterpart to rateMovie to check if rating is allowed	phenomenon
addingNewMovie	counterpart to addNewMovie	phenomenon
sortMoviesDatabase	arrange movies by descending order of rating	phenomenon
displayMovieDetails	movie title, director, actors, publishing date	phenomenon
previewAvgRating	shows the calculated average rating of movie	phenomenon
showUserComments	comments written by registered users	phenomenon
GroupChat	designed to control all the activities in the groups	domain
creatingGroup	counterpart to createGroup	phenomenon
browsingGMsList	counterpart to browseGMsList	phenomenon
discussingMovie	counterpart to discussMovie	phenomenon
addingRU	counterpart to addRU	phenomenon
leavingGroup	counterpart to leaveGroup	phenomenon
banningGM	counterpart to banGM	phenomenon
deletingGroup	counterpart to leaveNdeleteGroup	phenomenon
displayGroupMembers	all members' usernames and lists shown in group	phenomenon
allowChatDiscussion	group members send chat messages	phenomenon
RegisteredUserDataBase	database designed to store users' and groups' data	domain
registering	counterpart to register	phenomenon

checkingLogIn	counterpart to login	phenomenon
confirmLogOut	counterpart to logout	phenomenon
addingToMyList	counterpart to addToMyList	phenomenon
containUserData	all informations about the users	phenomenon
MovieRatingApp	The software being built	domain

V3 There must be exactly one context diagram.
Only one context diagram is provided.

V4 A context diagram has at least one machine domain.
MovieRatingApp is one machine domain.
The following table validates V4, V6 and V7.

Domain	**Domain Type(s)**	**Connected Domain(s)**	**Connected Domain(s) Type(s)**
MovieRatingApp	machine domain	User	biddable domain
		RegisteredUser	biddable domain
		GroupMember	biddable domain
		Administrator	biddable domain
		MovieDatabase	lexical domain, designed domain
		GroupChat	lexical domain, designed domain
		RegisteredUserDataBase	lexical domain, designed domain
User	biddable domain	MovieRatingApp	machine domain
RegisteredUser	biddable domain	MovieRatingApp	machine domain
GroupMember	biddable domain	MovieRatingApp	machine domain
Administrator	biddable domain	MovieRatingApp	machine domain
MovieDatabase	lexical domain, designed domain	MovieRatingApp	machine domain
GroupChat	lexical domain, designed domain	MovieRatingApp	machine domain
RegisteredUserDataBase	lexical domain, designed domain	MovieRatingApp	machine domain

V5 The machine domain must control at least one interface.
MovieRatingApp controls several interfaces (feedbackRU, feedbackGM, . . .).

V6 Biddable domains cannot be directly connected to lexical domains.
No biddable domain is connected to a lexical domain.

V7 Causal, designed, lexical, display, machine domain type are not allowed together with biddable domain.
User, RegisteredUser, GroupMember and Administrator are biddable domains only.

V8 Phenomena controlled by a biddable domain must have counterpart phenomena located between machine and causal/lexical/designed domains.

biddable domain	**biddable domain phenomena**	**counterpart**
User	register	registering
RegisteredUser	login, logout	checkingLogIn, confirmLogOut
	browseMovie	browsingMovie
	addToMyList	addingToMyList
	rateMovie	validateRating
	addNewMovie	addingNewMovie
	createGroup	creatingGroup
GroupMember	browseGMsList	browsingGMsList
	discussMovie	discussingMovie
	leaveGroup	leavingGroup
Administrator	addRU	addingRU
	banGM	banningGM
	leaveNdeleteGroup	leavingGroup, deletingGroup

V9 Connection domains must have at least one observed and one controlled interface.
For each phenomenon controlled by a connection domain, there must be at least one phenomenon controlled by one of the connected domains, i.e. observed by the connection domain.
For each phenomenon observed by a connection domain, there must be at least one phenomenon controlled the connection domain, i.e. for each input there is an output.
Context diagram contains no connection domain.

1.2 A2

The following four requirements are summarized from 1.1.1 and their problem diagrams are derived.

R-I **Register user**

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 user name. To register, a person must be at least eighteen years old, otherwise the registration fails.
R3 The user name has to be unique.

R-II **Add movie into the database**

R9 If a movie is not yet contained in the database, the user can add it by providing the title, director, main actors (at least one) and original publishing date.

R-III **Rate movie**

R5 A logged in user can add movies he/she has watched from a database to his/her list and rate them from 1 to 10 with a optional comment.
R6 If a value differing from one to ten is entered, the rating process will fail.
R7 If no rating is entered, the movie is rated as zero.
R8 A user cannot give more than one rating per movie.

R-IV **Browse movie**

R10 Additionally, users can access a list of all movies in the database sorted by rating in descending order. For each movie in the database, the average rating is calculated and shown together with the comments.

1.2.1 Problem diagram for R-I

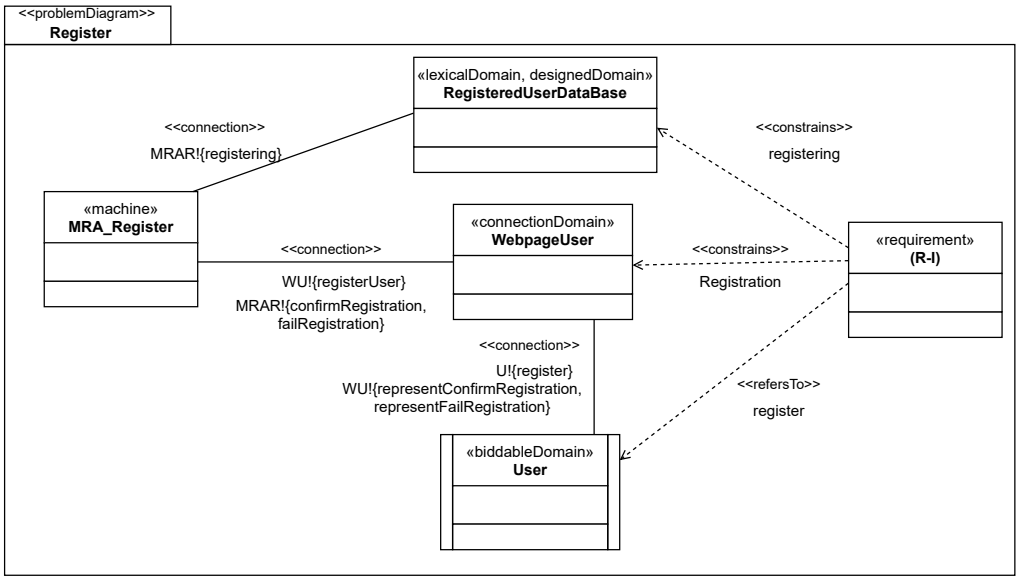
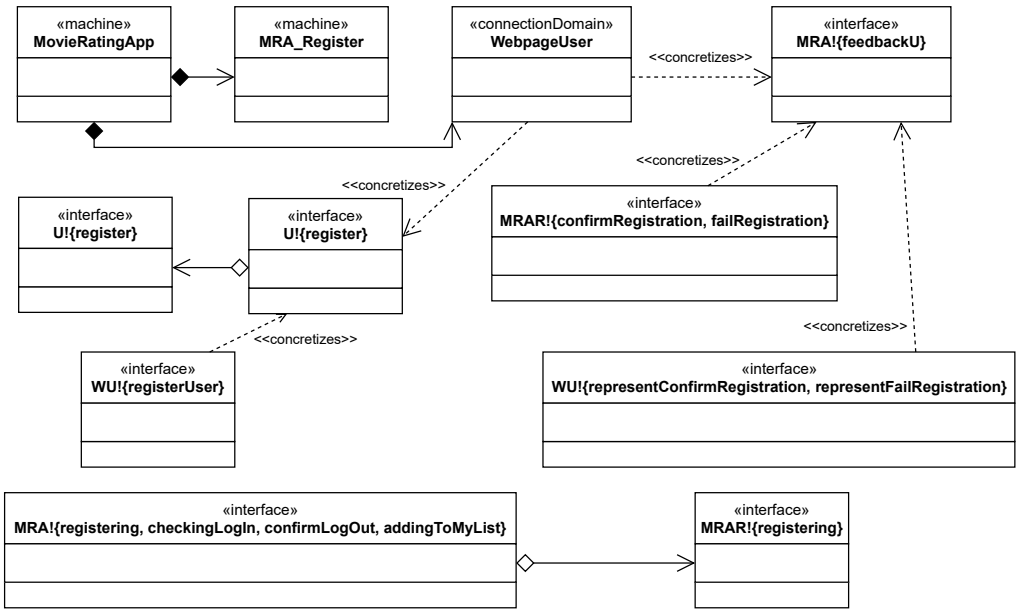


Figure 1.2: Problemdiagram for R-I

Concretize interface(s):



1.2.2 Problem diagram for R-II

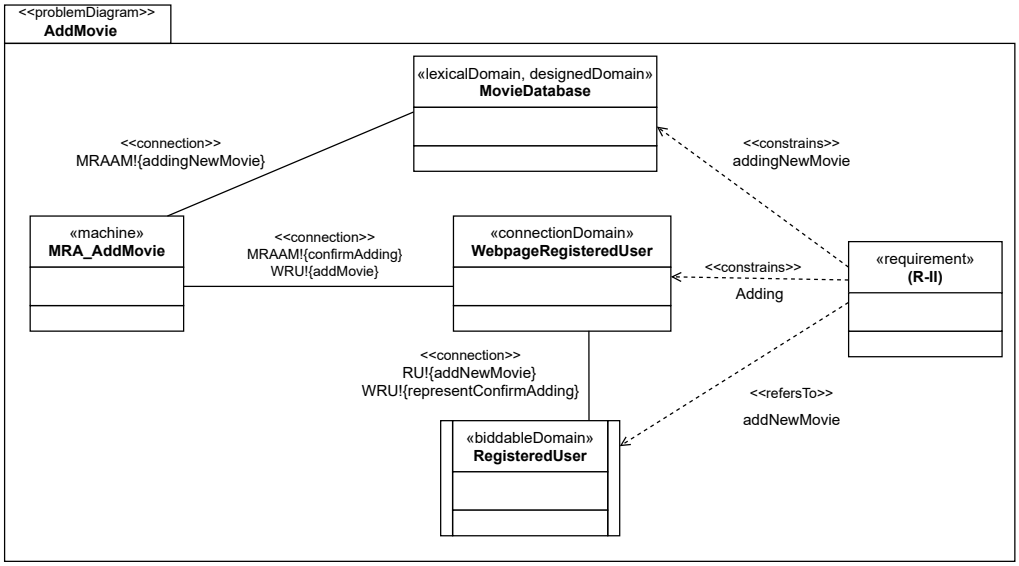
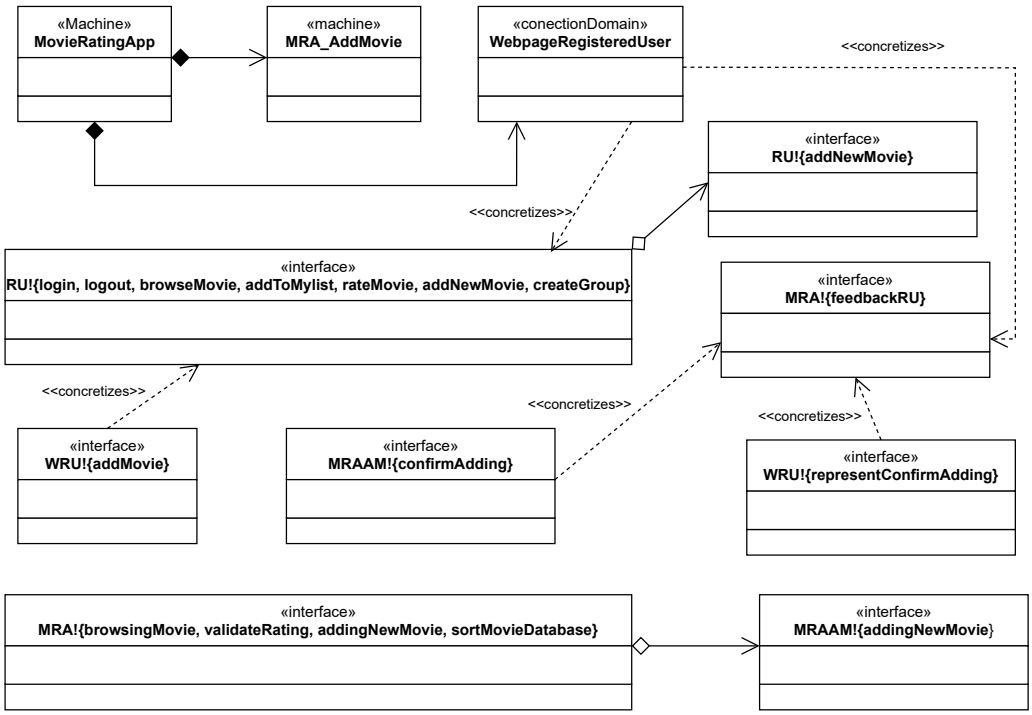


Figure 1.3: Problemdiagram for R-II

Concretize interface(s):



1.2.3 Problem diagram for R-III

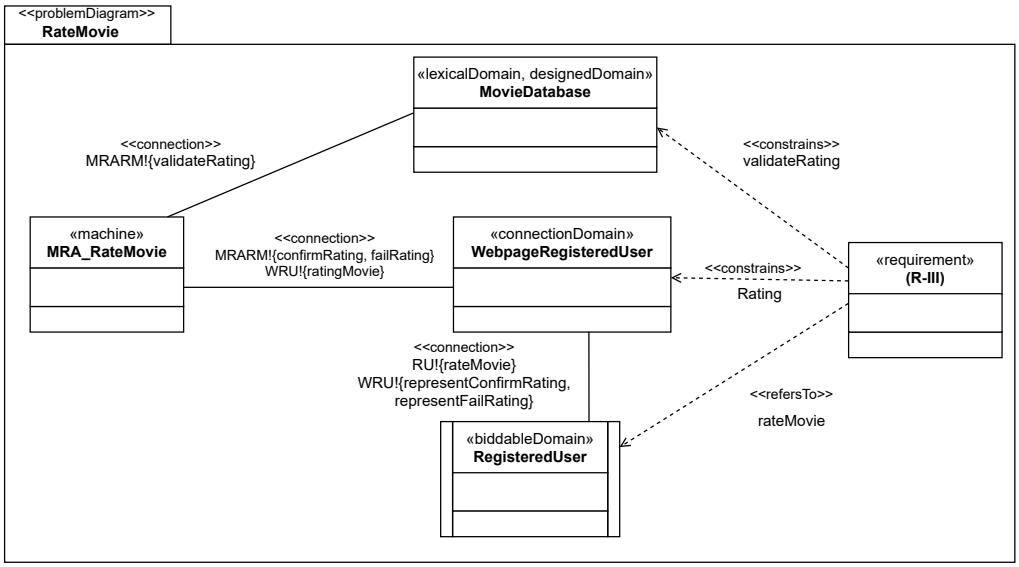
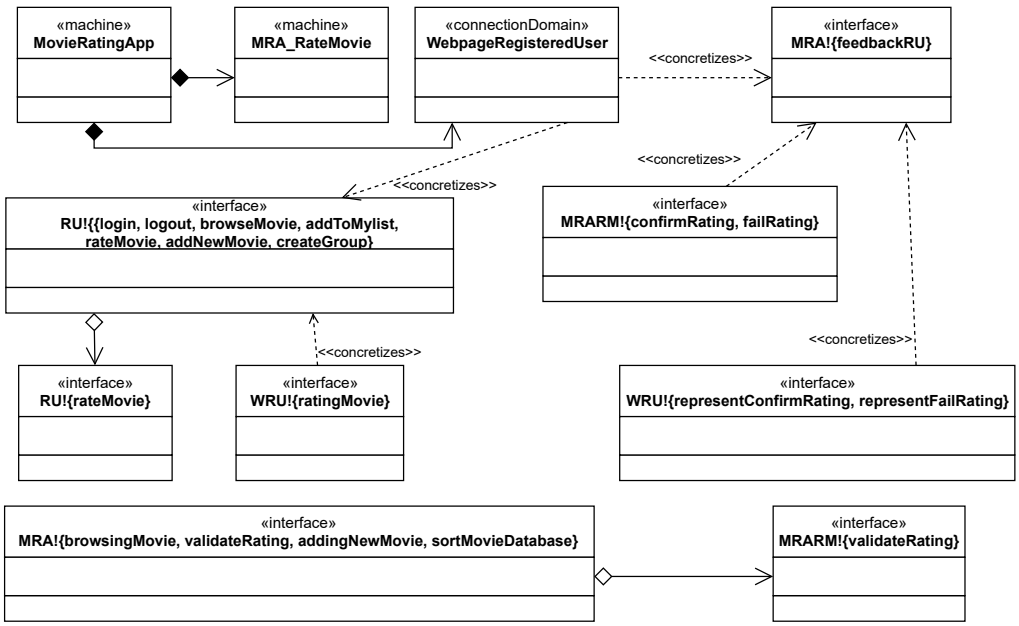


Figure 1.4: Problemdiagram for R-III

Concretize interface(s):



1.2.4 Problem diagram for R-IV

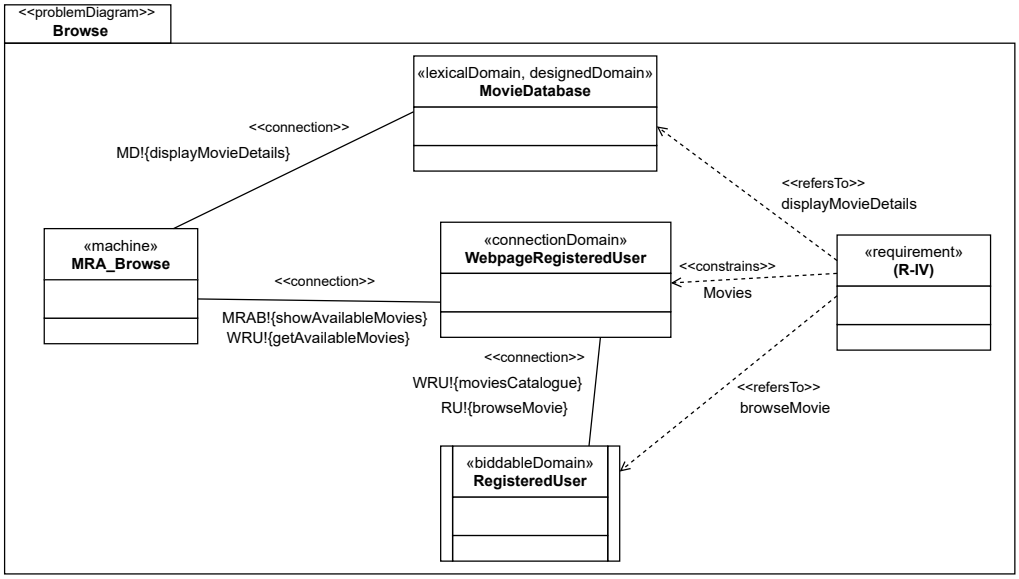
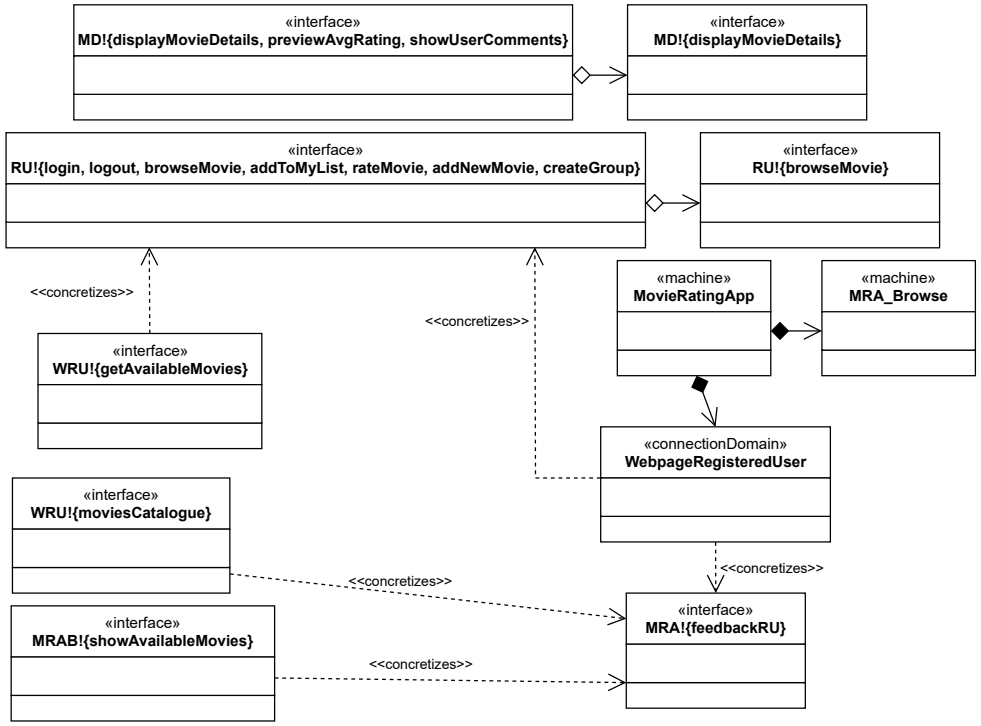


Figure 1.5: Problemdiagram for R-IV

Concretize interface(s):



1.2.5 Validation to problem diagrams

V1 All the related requirements R are covered in some subproblem.
The following table validates V1, V2, V3, V4, V5 and V6.

requirement	**covered in**	**contained domain**	**domain type**	**constrained**	**controlled phenomena**
R1, R2, R3 (R-I)	pdRegister	MRA_Register	machine		registering, confirmRegistration, failRegistration
		RegisteredUserDataBase	lexical, designed	X	
		WebpageUser	connection	X	registerUser, representConfirmRegistration, representFailRegistration
		User	biddable		register
R9 (R-II)	pdAddMovie	MRA_AddMovie	machine		addingNewMovie, confirmAdding
		MovieDatabase	lexical, designed	X	
		WebpageRegisteredUser	connection	X	addMovie, representConfirmAdding
		RegisteredUser	biddable		addNewMovie
R5, R6, R7, R8 (R-III)	pdRateMovie	MRA_RateMovie	machine		validateRating, confirmRating, failRating
		MovieDatabase	lexical, biddable	X	
		WebpageRegisteredUser	connection	X	ratingMovie, representConfirmRatng, representFailRating
		RegisteredUser	biddable		rateMovie
R10 (R-IV)	pdBrowse	MRA_Browse	machine		showAvailableMovies
		MovieDatabase	lexical, designed		displayMovieDetails
		WebpageRegisteredUser	connection	X	getAvailableMovies, moviesCatalogue
		RegisteredUser	biddable		browseMovie

V2 A problem diagram has exactly one machine domain.

V3 A problem diagram contains at least one requirement.

V4 The machine domain must control at least one interface.

V5 Requirements constrain at least one domain.

V6 Requirements do not constrain machine(s).

V7 If requirements do constrain biddable domains, a good argument is given and documented.
None of the requirements constrain biddable domains in the given problem diagrams.

V8 Connection domains must have at least one observed and one controlled interface.
The following table validates V8 and V9.

connection domain	**phenomenon controlled by connection domain**	**connected domain**	**phenomenon controlled by connected domain**
WebpageUser	registerUser	User	register
	representConfirmRegistration	MRA_Register	confirmRegistration
	representFailRegistration	MRA_Register	failRegistration
WebpageRegisteredUser	addMovie	RegisteredUser	addNewMovie
	representConfirmAdding	MRA_AddMovie	confirmAdding
	ratingMovie	RegisteredUser	rateMovie
	representConfirmRating	MRA_RateMovie	confirmRating
	representFailRating	MRA_RateMovie	failRating
	getAvailableMovies	RegisteredUser	browseMovie
	moviesCatalogue	MRA_Browse	showAvailableMovies

V9 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.

V10 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**
WebpageUser	register	registerUser
	confirmRegistration	representConfirmRegistration
	failRegistration	representFailRegistration
WebpageRegisteredUser	addNewMovie	addMovie
	confirmAdding	representConfirmAdding
	rateMovie	ratingMovie
	confirmRating	representConfirmRating
	failRating	representFailRating
	browseMovie	getAvailableMovies
	showAvailableMovies	moviesCatalogue

V11 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 below each of the problem diagrams.

V12 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, GroupMember, Administrator, MovieDatabase, GroupChat
	introduce connection/ display domain	WebpageUser
	split interface	MRA!{...}
	concretize interface	U!{...}, MRA!{...}

pdAddMovie	leave out domain	User, GroupMember, Administrator, RegisteredUserDataBase, GroupChat
	introduce connection/ display domain	WebpageRegisteredUser
	split interface	MRA!{...}, RU!{...}
	concretize interface	RU!{...}, MRA!{...}
pdRateMovie	leave out domain	User, GroupMember, Administrator, RegisteredUserDataBase, GroupChat
	introduce connection/ display domain	WebpageRegisteredUser
	split interface	MRA!{...}, RU!{...}
	concretize interface	RU!{...}, MRA!{...}
pdBrowse	leave out domain	User, GroupMember, Administrator, RegisteredUserDataBase, GroupChat
	introduce connection/ display domain	WebpageRegisteredUser
	split interface	MD!{...}, RU!{...}
	concretize interface	RU!{...}, MRA!{...}

Provided mapping diagrams to validate further.

1.2.6 Problem Frames

R-I fits to the problem frame **update (2)** (constrained lexical + connection domains, referred to biddable domain).
R-II fits to the problem frame **update (2)** (constrained lexical + connection domains, referred to biddable domain).
R-III fits to the problem frame **update (2)** (constrained lexical + connection domains, referred to biddable domain).
R-IV fits to the problem frame **query (2)** (constrained connection domain, referred to biddable + lexical domains).

1.2.7 Validation to problem frames

A problem diagram must be consistent to its problem frame.

V1 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!{confirmRegistration, failRegistration} WU!{registerUser}	UM!E4, IOD!E8	Machine	ConnectionDomain
		WU!{representConfirmRegistration, representFailRegistration} U!{register}	UO!E6, IOD!C7	Biddable-Domain	ConnectionDomain
AddMovie	update2	MRAAM!{addingNewMovie}	DB!Y1, UM!E2	Machine	LexicalDomain
		MRAAM!{confirmAdding} WRU!{addMovie}	UM!E4, IOD!E8	Machine	ConnectionDomain
		WRU!{representConfirmAdding} RU!{addNewMovie}	UO!E6, IOD!C7	Biddable-Domain	ConnectionDomain
RateMovie	update2	MRARM!{validateRating}	DB!Y1, UM!E2	Machine	LexicalDomain
		MRARM!{confirmRating, failRating} WRU!{ratingMovie}	UM!E4, IOD!E8	Machine	ConnectionDomain
		WRU!{representConfirmRating, representFailRating} RU!{rateMovie}	UO!E6, IOD!C7	Biddable-Domain	ConnectionDomain
Browse	query2	MD!{displayMovieDetails}	DB!Y1	Machine	LexicalDomain
		MRAB!{showAvailableMovies} WRU!{getAvailableMovies}	QM!Y3, IOD!C6	Machine	ConnectionDomain
		WRU!{moviesCatalogue} RU!{browseMovie}	IOD!E7, EO!E5	Biddable-Domain	ConnectionDomain

V2 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	RegisteredUserDataBase	Data Base	LexicalDomain
		WebpageUser	Input Output Device	ConnectionDomain
AddMovie	update2	MovieDatabase	Data Base	LexicalDomain
		WebpageRegisteredUser	Input Output Device	ConnectionDomain
RateMovie	update2	MovieDatabase	Data Base	LexicalDomain
		WebpageRegisteredUser	Input Output Device	ConnectionDomain
Browse	query2	WebpageRegisteredUser	Input Output Device	ConnectionDomain

V3 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	User	Update Operator	BiddableDomain
AddMovie	update2	RegisteredUser	Update Operator	BiddableDomain
RateMovie	update2	RegisteredUser	Update Operator	BiddableDomain
Browse	query2	RegisteredUser	Enquiry Operator	BiddableDomain
		MovieDatabase	Data Base	LexicalDomain

All problem diagrams are consistent to their problem frames.

1.3 A3

1.3.1 R-I Register user

Deriving the specification:
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 user name. To register, a person must be at least eighteen years old, otherwise the registration fails.
R3 The user name has to be unique.
Assumptions
A1 A web application is suitable to be used on different platforms including mobile devices.

We can derive the following specifications:
WebpageUser(S-Ia) When the webpage receives the command "register", then the command is forwarded to the machine with the command "registerUser". The feedback whether the request was confirmed or not is received via the commands "confirmRegistration" and "failRegistration". Both feedback types are shown to the user via the commands "representConfirmRegistration" and "representFailRegistration".
MRA_Register(S-Ib) When the machine receives the command "registerUser", then it is checked if the user is above at least 18 years old and if the username is unique with the command "get_Registered". If both the conditions are fulfilled, then the machine registers the user with command "registering" and informs the user with the command "confirmRegistration" to the user's webpage about the successful registration process. Otherwise, the machine informs the user with the command "failRegistration" to the user's webpage about the unsuccessful registration process.
RegisteredUserDataBase(S-Ic) When receiving the command "get_Registered", the required conditions for registration are checked. When the command "registering" is received, the user is successfully registered.

Correctness condition:
(S-Ia) $\wedge$ (S-Ib) $\wedge$ (S-Ic) $\wedge$ (A1) $\Longrightarrow$ (R-I)

Sequence diagram:

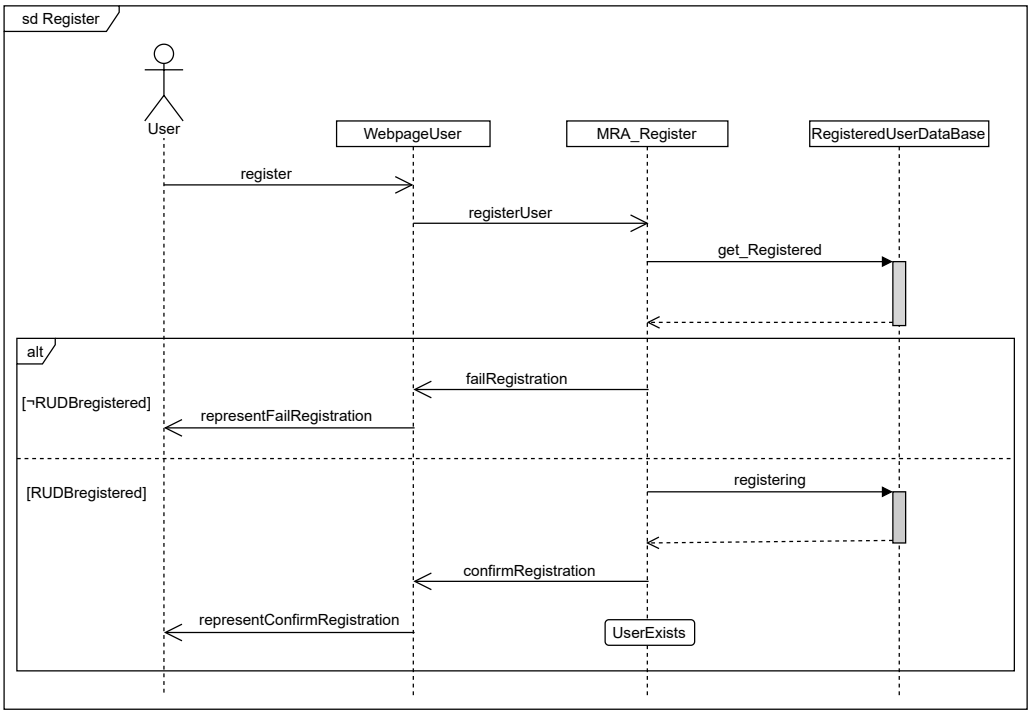


Figure 1.6: Sequencediagram for R-I

1.3.2 R-II Add movie into the database

Deriving the specification:
Requirements
R9 If a movie is not yet contained in the database, the user can add it by providing the title, director, main actors (at least one) and original publishing date.
Facts
F2 Each movie has a title, a director, at least one main actor and an original publishing date.
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.

We can derive the following specifications:
WebpageRegisteredUser(S-IIa) When the webpage receives the command "addNewMovie", then the command is forwarded to the machine with the command "addMovie". The results are received via the command "confirmAdding" and shown to the user by "representConfirmAdding".
MRA_AddMovie(S-IIb) When the machine receives the command "addMovie", adding a new movie occurs with the command "addingNewMovie", and thus the movie is added into the database. The results are returned via the command "confirmAdding".
MovieDatabase(S-IIc) After receiving the command "addingNewMovie", the movie is indeed added into the database.

Correctness condition:
(S-IIa) $\wedge$ (S-IIb) $\wedge$ (S-IIc) $\wedge$ (F2) $\wedge$ (A1) $\wedge$ (A2) $\Longrightarrow$ (R-II)

Sequence diagram:

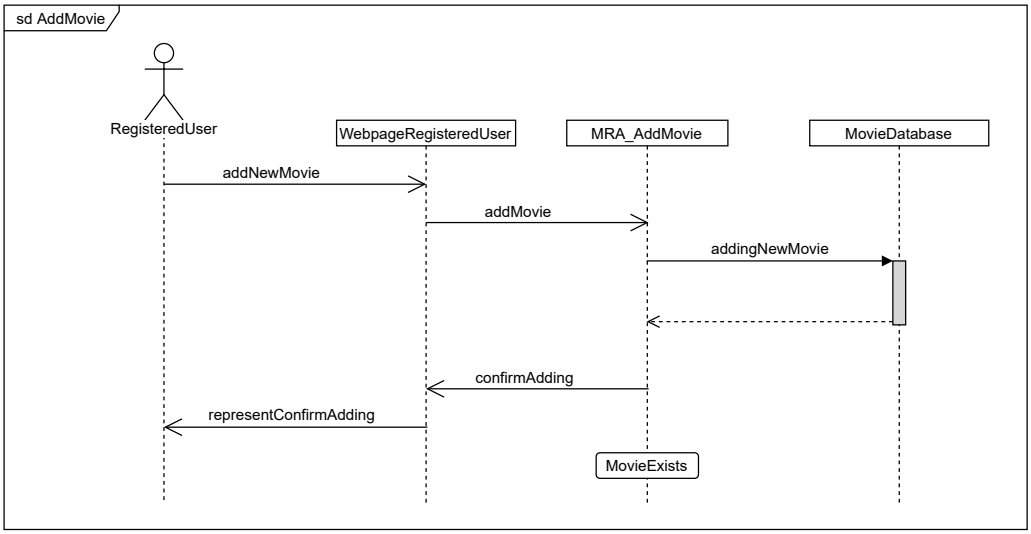


Figure 1.7: Sequencediagram for R-II

1.3.3 R-III Rate movie

Deriving the specification:
Requirements
R5 A logged in user can add movies he/she has watched from a database to his/her list and rate them from 1 to 10 with a optional comment.
R6 If a value differing from one to ten is entered, the rating process will fail.
R7 If no rating is entered, the movie is rated as zero.
R8 A user cannot give more than one rating per movie.
Facts
F1 A rating consists out of a mandatory point rating from one to ten (1 = very bad movie, 10 = excellent movie) and optionally a written comment.
Assumptions
A1 A web application is suitable to be used on different platforms including mobile devices.
A3 Users only rate movies they have really watched, and their rating is based only on their own opinions.

We can derive the following specifications:
WebpageRegisteredUser(S-IIIa) When the webpage receives the command “rateMovie”, then the command is forwarded to the machine with the command “ratingMovie”. The results are received via the command of either “confirmRating” or “failRating” and shown to the guest by “representConfirmRating” or “representFailRating”, respectively.
MRA_RateMovie(S-IIIb) When the machine receives the command “ratingMovie”, then is it checked if all required informations for the rating process are fulfilled with the command “get_Rated”. If all conditions are fulfilled, the machine completes the rating process with the command ”validateRating” and the registered user’s webpage is informed via the command “confirmRating”. Otherwise, the machine informs the registered user’s webpage with the command “failRating” about the unsuccessful rating process.
MovieDatabase(S-IIIc) After receiving the command “get_Rated”,the required conditions for the rating process are checked. When the command ”validateRating” is received, the movie is successfully rated.

Correctness condition:
(S-IIIa) $\wedge$ (S-IIIb) $\wedge$ (S-IIIc) $\wedge$ (F1) $\wedge$ (A1) $\wedge$ (A3) $\Longrightarrow$ (R-III)

Sequence diagram:

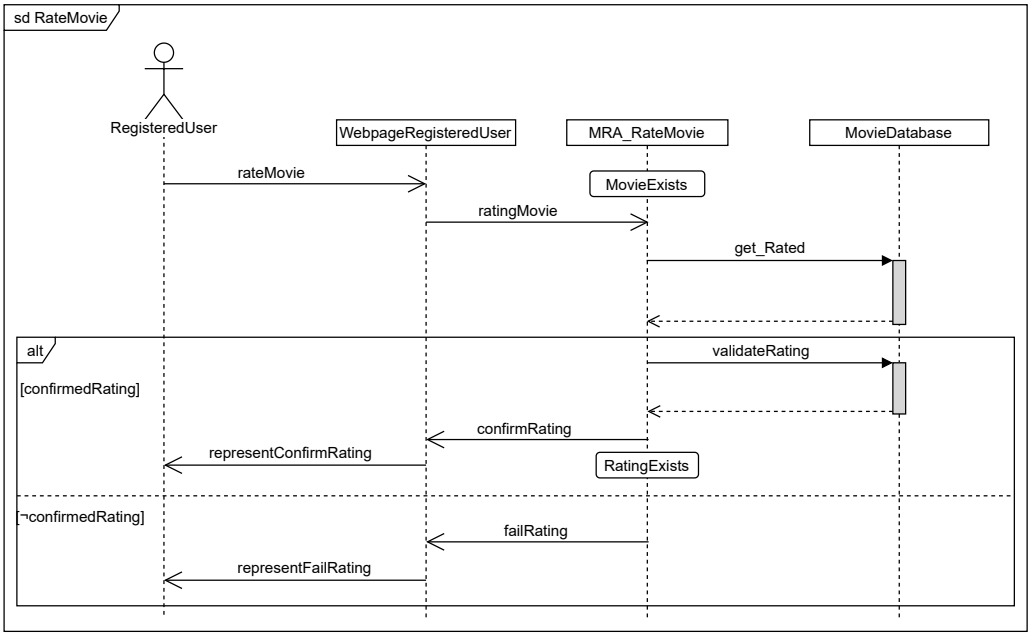


Figure 1.8: Sequencediagram for R-III

1.3.4 R-IV Browse movie

Deriving the specification:
Requirements
R10 Additionally, users can access a list of all movies in the database sorted by rating in descending order. For each movie in the database, the average rating is calculated and shown together with the comments.
Assumptions
A1 A web application is suitable to be used on different platforms including mobile devices.

We can derive the following specifications:
WebpageRegisteredUser(S-IVa) When the webpage receives the command "browseMovie", then the command is forwarded to the machine with the command "getAvailableMovies". The results are received via the command "showAvailableMovies" and shown to the user by "moviesCatalogue".
MRA_Browse(S-IVb) When the machine receives the command "getAvailableMovies", the available movies are requested with the command "get_availableMovies" and received as the data "displayMovieDetails". The results are returned via the command "showAvailableMovies" to the webpage.
MovieDatabase(S-IVc) After receiving the command "get_availableMovies", the results are returned as the data "displayMovieDetails".

Correctness condition:
(S-IVa) $\wedge$ (S-IVb) $\wedge$ (S-IVc) $\wedge$ (A1) $\Longrightarrow$ (R-IV)

Sequence diagram:

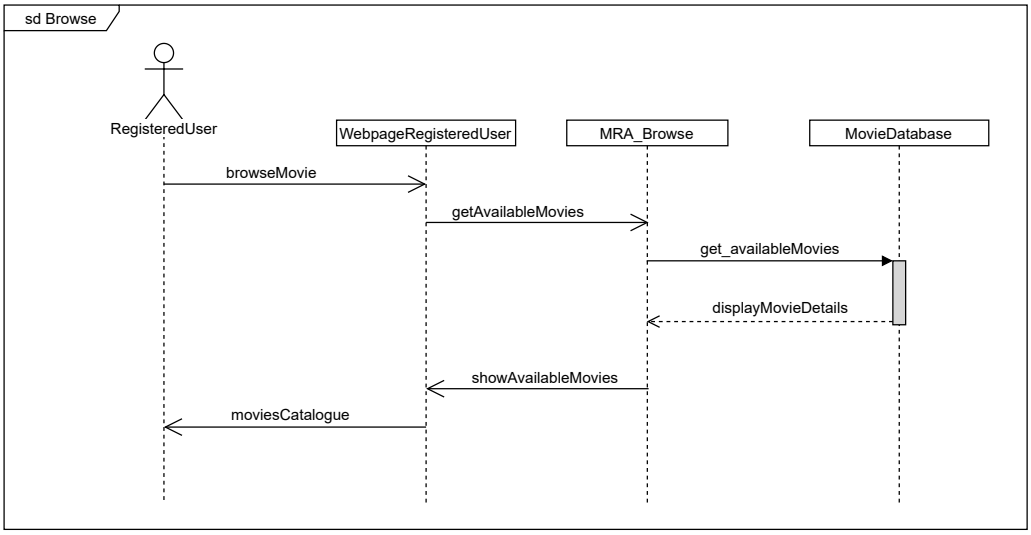


Figure 1.9: Sequencediagram for R-IV

1.3.5 Validation

V1 $S_{abstract} \wedge D$ are non-contradictory.
No contradictions can be found in $S_{abstract} \wedge D$.

V2 $S_{abstract} \wedge D \Longrightarrow R$.
(S-Ia) $\wedge$ (S-Ib) $\wedge$ (S-Ic) $\wedge$ (A1) $\Longrightarrow$ (R-I)
(S-IIa) $\wedge$ (S-IIb) $\wedge$ (S-IIc) $\wedge$ (F2) $\wedge$ (A1) $\wedge$ (A2) $\Longrightarrow$ (R-II)
(S-IIIa) $\wedge$ (S-IIIb) $\wedge$ (S-IIIc) $\wedge$ (F1) $\wedge$ (A1) $\wedge$ (A3) $\Longrightarrow$ (R-III)
(S-IVa) $\wedge$ (S-IVb) $\wedge$ (S-IVc) $\wedge$ (A1) $\Longrightarrow$ (R-IV)

V3 Messages and phenomena are consistent.

message in scenario	**source**	**target**	**phenomena in problem diagram**
register	User	WebpagerUser	U!{register}
registerUser	WebpageUser	MRA_Register	WU!{registerUser}
registering	MRA_Register	RegisteredUserDataBase	MRAR!{registering}
failRegistration	MRA_Register	WebpageUser	MRAR!{failRegistration}
representFailRegistration	WebpageUser	User	WU!{representFailRegistration}
confirmRegistration	MRA_Register	WebpageUser	MRAR!{confirmRegistration}
representConfirmRegistration	WebpageUser	User	WU!{representConfirmRegistration}
addNewMovie	RegisteredUser	WebpageRegisteredUser	RU!{addNewMovie}
addMovie	WebpageRegisteredUser	MRA_AddMovie	WRU!{addMovie}
addingNewMovie	MRA_AddMovie	MovieDatabase	MRAAM!{addingNewMovie}
confirmAdding	MRA_AddMovie	WebpageRegisteredUser	MRAAM!{confirmAdding}
representConfirmAdding	WebpageRegisteredUser	RegisteredUser	WRU!{representConfirmAdding}
rateMovie	RegisteredUser	WebpageRegisteredUser	RU!{rateMovie}

ratingMovie	WebpageRegisteredUser	MRA_RateMovie	WRU!{ratingMovie}
validateRating	MRA_RateMovie	MovieDatabase	MRARM!{validateRating}
confirmRating	MRA_RateMovie	WebpageRegisteredUser	MRARM!{confirmRating}
representConfirm-Rating	WebpageRegisteredUser	RegisteredUser	WRU!{representConfirmRating}
failRating	MRA_RateMovie	WebpageRegisteredUser	MRARM!{failRating}
representfailRating	WebpageRegisteredUser	RegisteredUser	WRU!{representfailRating}
browseMovie	RegisteredUser	WebpageRegisteredUser	RU!{browseMovie}
getAvailableMovies	WebpageRegisteredUser	MRA_Browse	WRU!{getAvailableMovies}
get_availableMovies	MRA_Browse	MovieDatabase	MD!{displayMovieDetails}
showAvailableMov-ies	MRA_Browse	WebpageRegisteredUser	MRAB!{showAvailableMovies}
moviesCatalogue	WebpageRegisteredUser	RegisteredUser	WRU!{moviesCatalogue}

V4 Lexical domains are not sources of messages.

message in scenario	**source**	**domain type**
register	User	BiddableDomain
registerUser	WebpageUser	CasualDomain
registering	MRA_Register	Machine
failRegistration	MRA_Register	Machine
representFailRegistration	WebpageUser	CasualDomain
confirmRegistration	MRA_Register	Machine
representConfirmRegistration	WebpageUser	CasualDomain
addNewMovie	RegisteredUser	BiddableDomain
addMovie	WebpageRegisteredUser	CasualDomain
addingNewMovie	MRA_AddMovie	Machine
confirmAdding	MRA_AddMovie	Machine
representConfirmAdding	WebpageRegisteredUser	CasualDomain
rateMovie	RegisteredUser	BiddableDomain
ratingMovie	WebpageRegisteredUser	CasualDomain
validateRating	MRA_RateMovie	Machine
confirmRating	MRA_RateMovie	Machine
representConfirmRating	WebpageRegisteredUser	CasualDomain
failRating	MRA_RateMovie	Machine
representfailRating	WebpageRegisteredUser	CasualDomain
browseMovie	RegisteredUser	BiddableDomain
getAvailableMovies	WebpageRegisteredUser	CasualDomain
get_availableMovies	MRA_Browse	Machine
showAvailableMovies	MRA_Browse	Machine
moviesCatalogue	WebpageRegisteredUser	CasualDomain

V5 There exists at least one scenario for each subproblem.

V6 Scenarios cover normal cases and possibly exceptional cases.

subproblem	**normal case**	**exceptional case**
pdRegister	sdRegister	sdRegister
pdAddMovie	sdAddmovie	
pdRateMovie	sdRateMovie	sdRateMovie
pdBrowse	sdBrowse	

1.4 A4

1.4.1 Technical Context Diagram

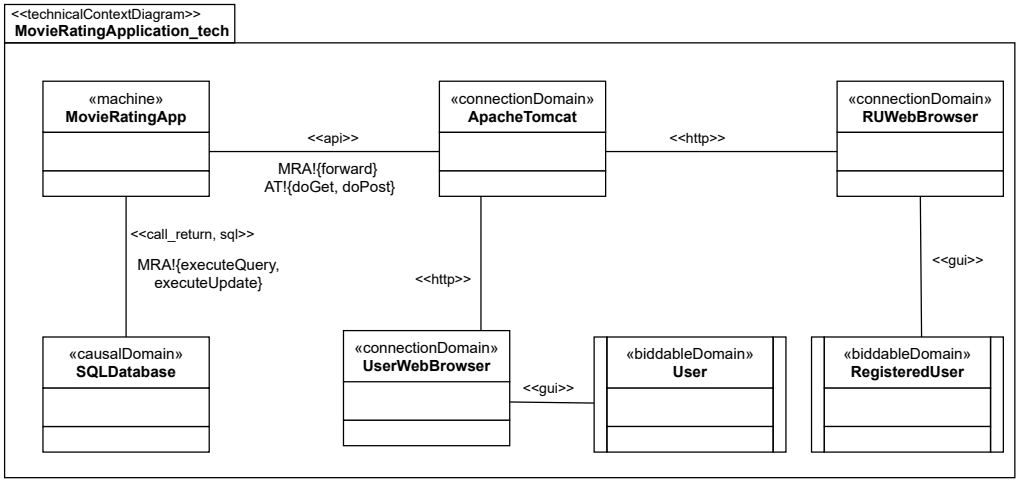


Figure 1.10: Technical Contextdiagram

Technical realization of domains from context and problem diagrams:

* **RegisteredUserDataBase, MovieDatabase:** 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.

* **WebpageUser:** Realized using ApacheTomcat and UserWebBrowser (browser of User).

* **WebpageRegisteredUser:** Realized using ApacheTomcat and RUWebBrowser (browser of RegisteredUser).

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)

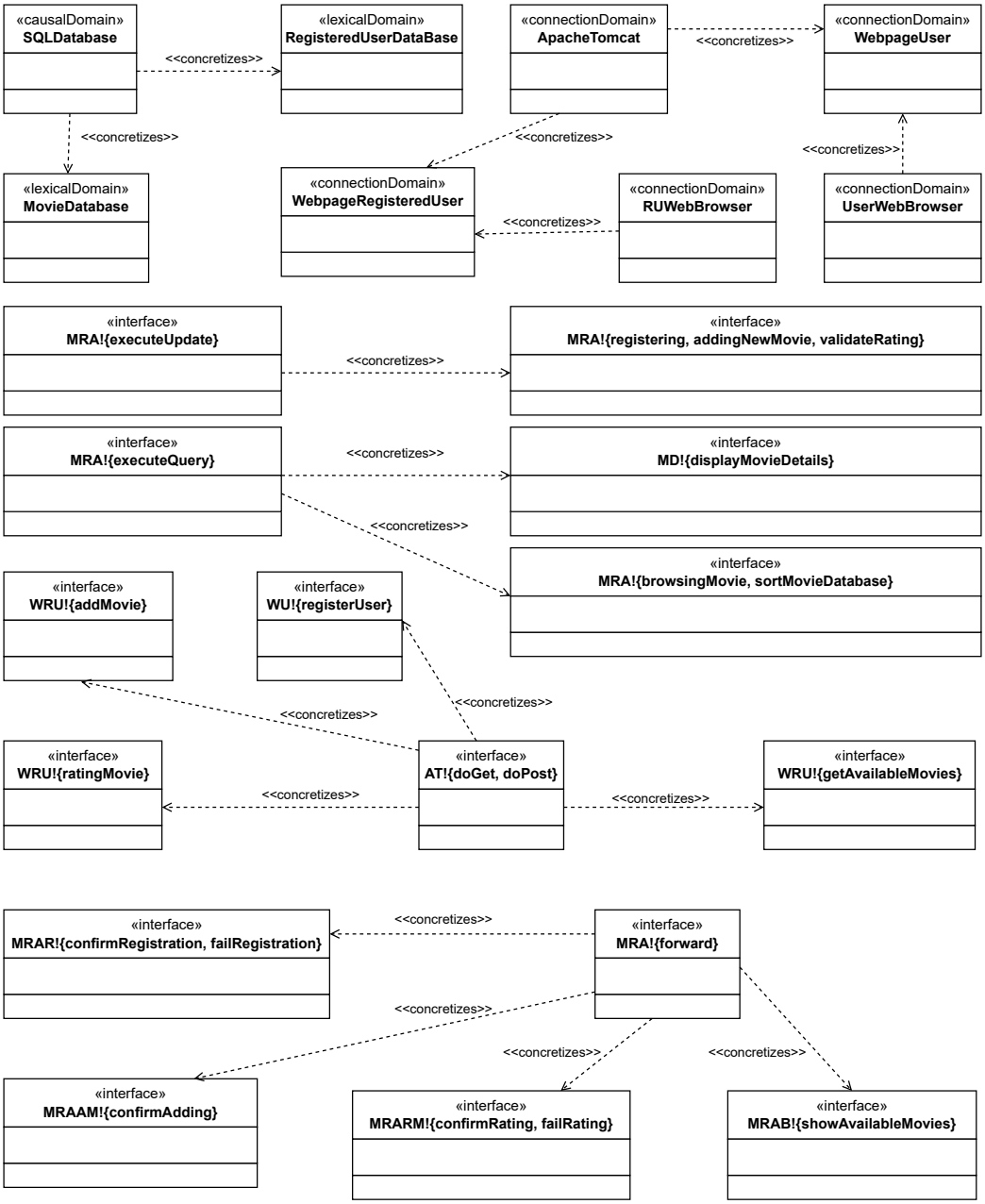
  - Operation forward defined in interface javax.servlet.RequestDispatcher (http://docs.oracle.com /javaee/7/api/javax/servlet/RequestDispatcher.html)

Technical interfaces in the environment:

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

* GUI: User interfaces of HTML webpages (defined by https://www.w3.org/TR/html5/) presented by WebBrowser

Mapping diagram for technical context diagram:



1.4.2 Validation

V1 New phenomena and domains are suitable to implement the external messages used in the abstract phenomena:

Message	**new phenomena and domains**
register	ApacheTomcat, HTTP
confirmRegistration, failRegistration	ApacheTomcat, HTTP
addNewMovie	ApacheTomcat, HTTP
confirmAdding	ApacheTomcat, HTTP
rateMovie	ApacheTomcat, HTTP
confirmRating, failRating	ApacheTomcat, HTTP
browseMovie	ApacheTomcat, HTTP
showAvailableMovies	ApacheTomcat, HTTP

All internal messages can be realized using SQL commands.

V2 All domains of the technical context diagram are related to domains in the problem diagrams.
All phenomena in the technical context diagram are related to elements in the problem diagrams.
Provided mapping diagram.

V3 All domains directly connected with the machine in the problem diagrams are related to elements in the technical context diagram:

Problem Diagram	**Domain connected with the machine**	**Element in the TCD**
Register	RegisteredUserDataBase	SQLDatabase
	WebpageUser	UserWebBrowser, ApacheTomcat
AddMovie	MovieDatabase	SQLDatabase
	WebpageRegisteredUser	RUWebBrowser, ApacheTomcat
RateMovie	MovieDatabase	SQLDatabase
	WebpageRegisteredUser	RUWebBrowser, ApacheTomcat
Browse	MovieDatabase	SQLDatabase
	WebpageRegisteredUser	RUWebBrowser, ApacheTomcat

1.5 A5

1.5.1 Register user

Class model

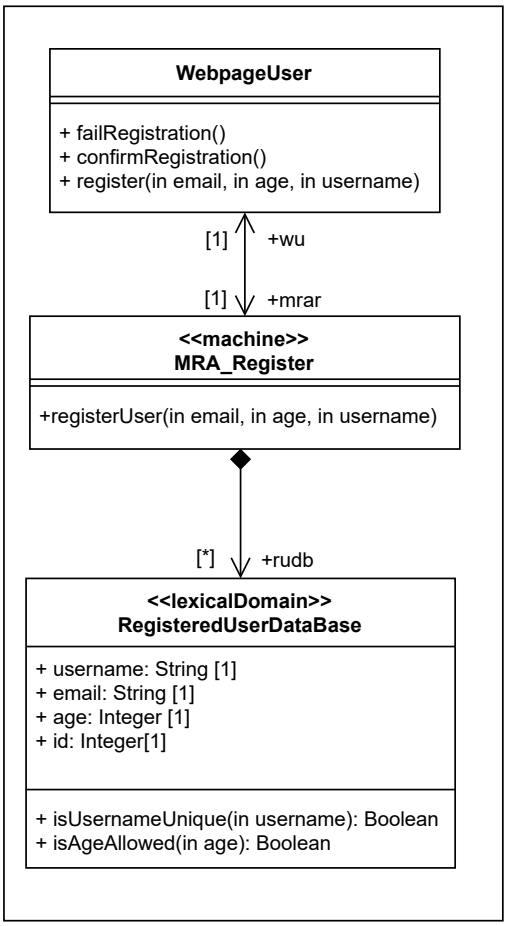


Figure 1.11: UserRegistration ClassModel

Operation specification

Name: register
Description: Forwards the register request from the User to the machine
OCL Constraint:

```
1 context WebpageUser::register(email : String, age : Integer,
      username : String)
2 pre: true
3 post:mrar^registerUser(email, age, username)
```

Name: isUsernameUnique
Description: Checks whether the username is unique
OCL Constraint:

```
context RegisteredUserDataBase::isUsernameUnique(username : String)
pre: true
post: result = not (rudb@pre->one(r:RUDB | r.username = username))
```

Name: isAgeAllowed
Description: Checks whether the entered age is at least 18
OCL Constraint:

```
context RegisteredUserDataBase::isAgeAllowed(age : Integer)
pre: true
post: if age >= 18 then result = true else result = false endif
```

Name: registerUser
Description: The machine will forward the request received from the user to the database for registration
OCL Constraint:

```
context MRA_Register::registerUser(email : String, age : Integer,
    username : String)
pre: true
post: if ru.isUsernameUnique(username) and ru.isAgeAllowed(age)
    then ru->one(r: RUDB | r.email = email and r.age = age and r.
    username = username) and ru->size() = ru@pre->size()+1 and wu^
    confirmRegistration() else wu^failRegistration() endif
```

Remarks:
1. We want to identify every users with an unique id.
OCL Constraint:

```
context RegisteredUserDataBase
inv: RegisteredUserDataBase.allInstances()->isUnique(id)
```

Validation

V1 Operation specifications must be consistent with abstract specifications:
The operation specification of 'register' is consistent with the abstract specification.

V2 The postcondition covers all cases exhibited in the abstract specification:
The normal and exceptional case behavior described in the abstract specification is covered in the postcondition.

V3 All parameters of operations must be known by the caller and all parameters of sent messages must be known by the machine:
User can input all responses to WebpageUser, which forwards these to this operation. The machine knows the arguments used in the message to WebpageUser.

V4 Parameters must be used in the pre- and/or postcondition:
The parameters are used in the postcondition.

V5 All classes, associations, and attributes newly introduced in the class model must be motivated by some operation specification.
The method "isUsernameUnique(in username): Boolean" is added to the class RegisteredUserDataBase because we need to check if the username is unique.
The attribute id was introduced for practical reasons.

1.5.2 Add movie into the database

Class model

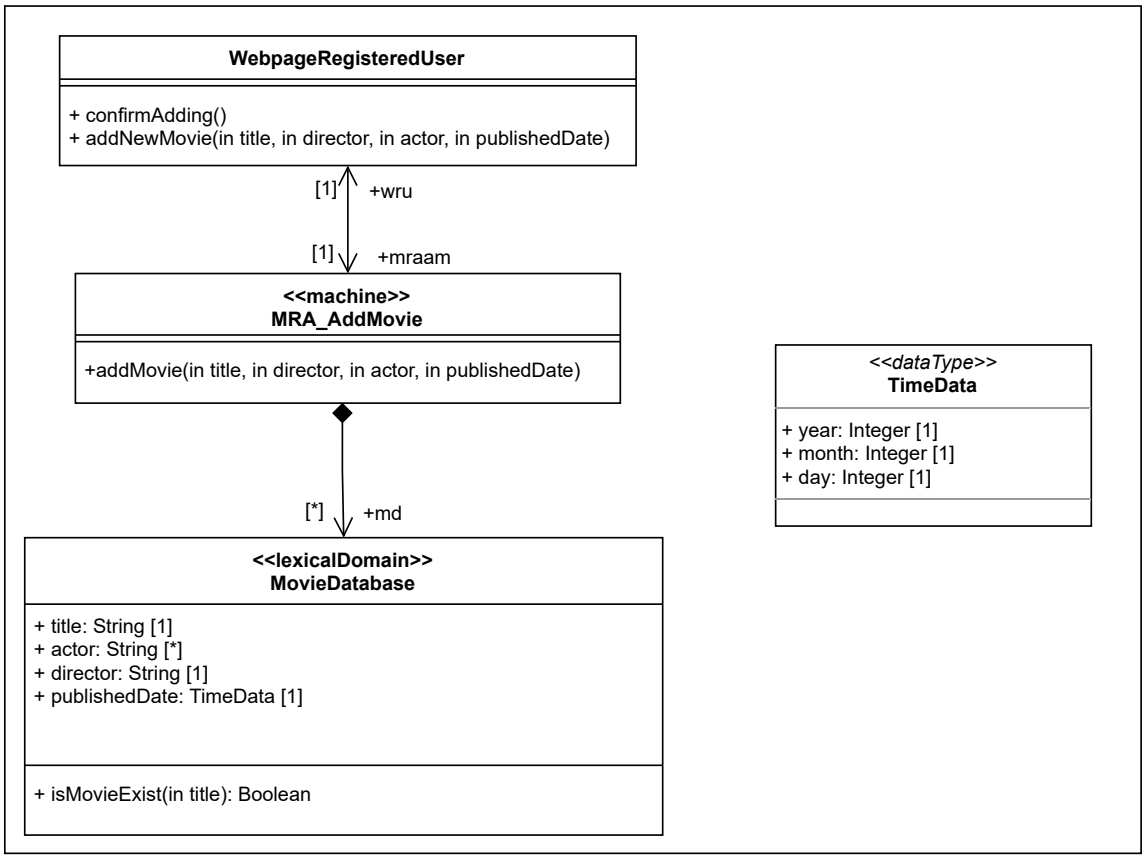


Figure 1.12: AddMovie ClassModel

Operation specification

Name: addNewMovie
Description: User is requesting to the machine through the WebpageRegisteredUser to add movie in the database
OCL Constraint:

```
1 context WebpageRegisteredUser::addNewMovie(title : String,
      director : String, actor : Set(String), publishedDate :
      TimeData)
2 pre: true
3 post: mraam^addMovie(title, director, actor, publishedDate)
```

Name: isMovieExists
Description: Checks whether the movie already exists in the database
OCL Constraint:

```
1 context MovieDatabase::isMovieExist(title : String)
2 pre: true
3 post: result = not (md@pre->one(m:MD | m.title = title))
```

Name: addMovie
Description: The machine will forward the request received from the user to the database for adding movie
OCL Constraint:

```
1 context MRA_AddMovie::addMovie(title : String, director : String,
      actor : Set(String), publishedDate : TimeData)
2 pre: true
3 post: if md.isMovieExist(title, director, actor, publishedDate)
    then md->one(a:MD | a.title = title and a.director = director
    and a.actor=actor and a.publishedDate=publishedDate) and wru^
    confirmAdding() else wru^failAdding() endif
```

Remarks:
1. The new phenomenon 'failAdding' is added so that the user gets notified in case the adding process is not successful.
2. We want to identify every movies by an unique movie title.
OCL Constraint:

```
1 context MovieDatabase
2 inv: MovieDatabase.allInstances()->isUnique(title)
```

Validation

V1 Operation specifications must be consistent with abstract specifications:
The operation specification of 'addNewMovie' is consistent with the abstract specification.

V2 The postcondition covers all cases exhibited in the abstract specification:
The normal case behavior described in the abstract specification is covered in the postcondition.

V3 All parameters of operations must be known by the caller and all parameters of sent messages must be known by the machine:
Registered User can input all responses to WebpageRegisteredUser, which forwards these to this operation. The machine knows the arguments used in the message to WebpageRegisteredUser.

V4 Parameters must be used in the pre- and/or postcondition:
The parameters are used in the postcondition.

V5 All classes, associations, and attributes newly introduced in the class model must be motivated by some operation specification.
The method "isMovieExist(in title, in director, in actor, in publishedDate): Boolean" is added to the class MovieDatabase to check if the movie already exists in the database. The attribute id was introduced for practical reasons.

1.5.3 Rate Movie

Class model

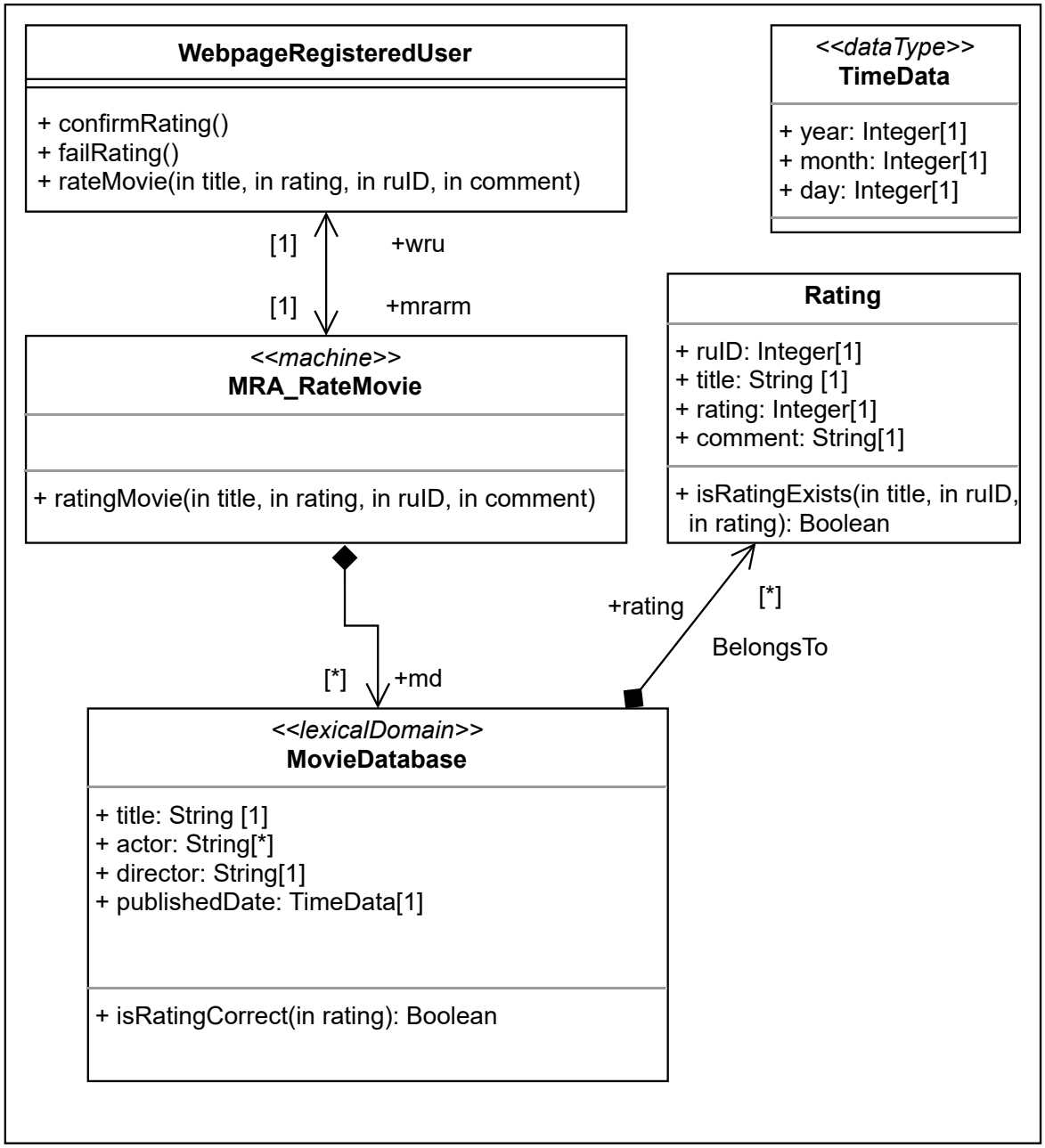


Figure 1.13: RateMovie ClassModel

Operation specification

Name: rateMovie
Description: User is requesting to the machine through the WebpageRegisteredUser to rate a movie in the database
OCL Constraint:

```
1 context WebpageRegisteredUser::rateMovie(title: String, rating:
    Integer, ruID: Integer)
2 pre: true
3 post: mrarm^ratingMovie(title, rating, ruID)
```

Name: isRatingExists
Description: Checks whether the user already rated selected movie
OCL Constraint:

```
1 context Rating::isRatingExists(title : String, ruID : Integer,
    rating : Integer)
2 pre: true
3 post: result = not (r@pre->one(ra:Rating | ra.title = title and ra
    .ruID = ruID and ra.rating = rating))
```

Name: isRatingCorrect
Description: Checks whether the entered value of rating is allowed
OCL Constraint:

```
1 context MovieDatabase::isRatingCorrect(rating: Integer)
2 pre: true
3 post: if rating >= 1 and rating <= 10 then result = true else
    result = false endif
```

Name: ratingMovie
Description: The machine will forward the request received from the user to the database for rating a selected movie
OCL Constraint:

```
1 context MRA_RateMovie::ratingMovie(title: String, rating: Integer,
     ruID: Integer)
2 pre: true
3 post: if md.isRatingExists(title, ruID, rating) md.isRatingCorrect
    (rating) then md->one(r.Rating | r.title = title and r.rating =
     rating and r.ruID = ruID and r.comment = comment) and wru^
    confirmRating() else wru^failRating() endif
```

Validation

V1 Operation specifications must be consistent with abstract specifications:
The operation specification of 'rateMovie' is consistent with the abstract specification.

V2 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.

V3 All parameters of operations must be known by the caller and all parameters of sent messages must be known by the machine:
Registered User can input all responses to WebpageRegisteredUser, which forwards these to this operation. The machine knows the arguments used in the message to WebpageRegisteredUser.

V4 Parameters must be used in the pre- and/or postcondition:
The parameters are used in the postcondition.

V5 All classes, associations, and attributes newly introduced in the class model must be motivated by some operation specification.
The method "isRatingCorrect(in rating): Boolean" is added to the class MovieDatabase to check if the user rated a movie with a valid number.
The attribute id was introduced for practical reasons.

1.5.4 Browse Movie

Class model

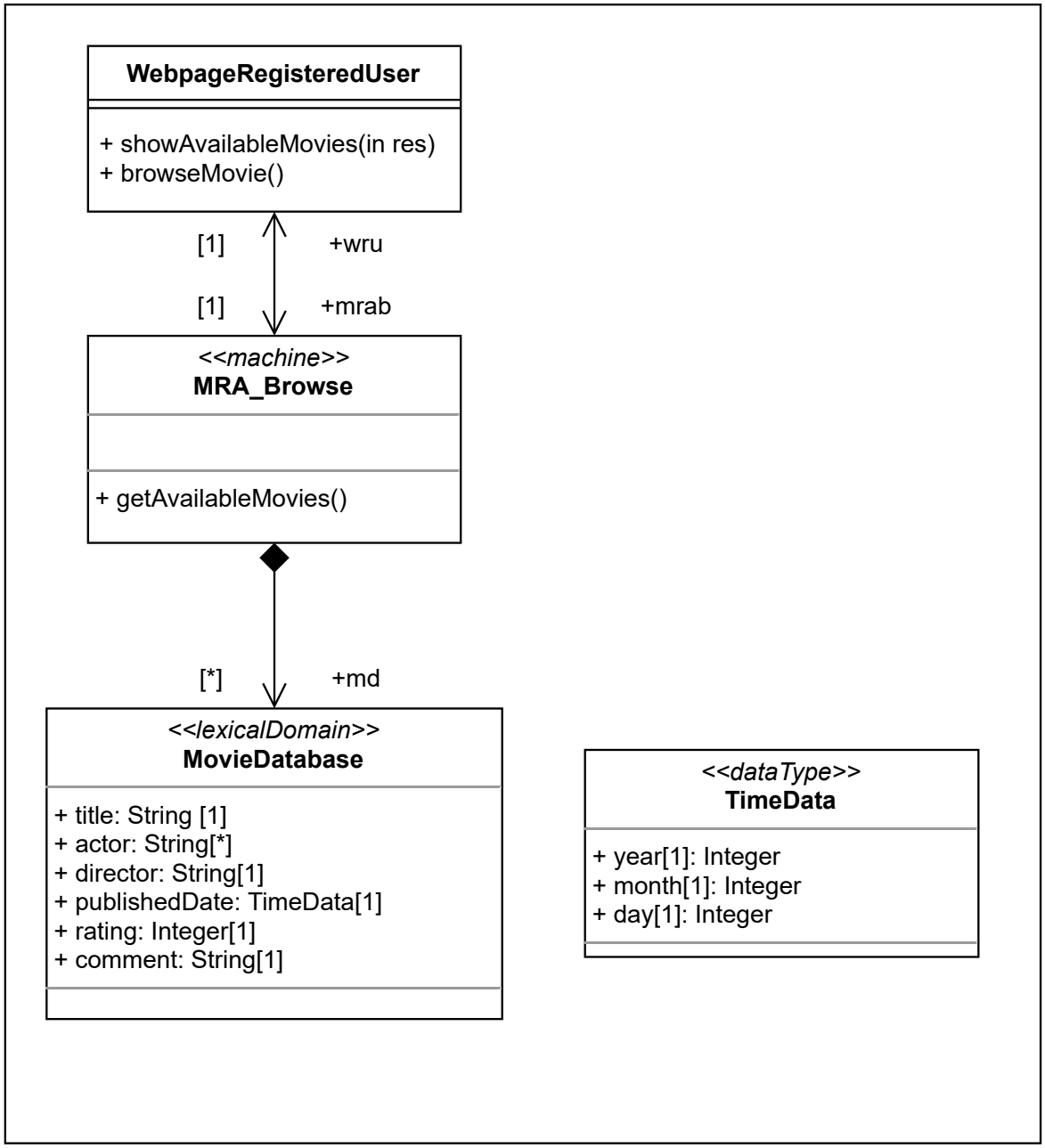


Figure 1.14: BrowseMovie ClassModel

Operation specification

Name: browseMovie
Description: User is requesting to the machine through the WebpageRegisteredUser to browse (or select) a movie in the database
OCL Constraint:

```
1 context WebpageRegisteredUser::browseMovie()
2 pre: true
3 post: mrab^getAvailableMovies()
```

Name: getAvailableMovies
Description: The machine will forward the request received from the user to the database for getting requested movie.
OCL Constraint:

```
1 context MRA_Browse::getAvailableMovies()
2 pre: true
3 post: let res : orderedSet(MovieDatabase) = md->select(m:
    MovieDatabase | m.title = title and m.rating = rating->sum()/
    rating->size() and m.comment = comment)->asOrderedSet() in wru^
    showAvailableMovies(res)
```

Validation

V1 Operation specifications must be consistent with abstract specifications:
The operation specification of 'browseMovie' is consistent with the abstract specification.

V2 The postcondition covers all cases exhibited in the abstract specification:
The normal case behavior described in the abstract specification is covered in the postcondition.

V3 All parameters of operations must be known by the caller and all parameters of sent messages must be known by the machine:
Registered User can input all responses to WebpageRegisteredUser, which forwards these to this operation. The machine knows the arguments used in the message to WebpageRegisteredUser.

V4 Parameters must be used in the pre- and/or postcondition:
The parameters are used in the postcondition.

1.6 A6

1.6.1 Life-cycle

$LC_{user} = Register$

$LC_{ru} = (AddMovie|(BrowseMovie;[RateMovie]))^*$

$LC_{MovieRatingApp} = (||_{i=1}^{n} LC_{user_i})||(||_{j=1}^{m} LC_{ru_j})$
where $||_{i=1}^{n} LC_i$ denotes the parallel composition of n copies of life-cycle LC.

1.6.2 Validation

V1 Each sequence diagram is contained in at least one life-cycle expression.

scenario	**life-cycle expression**
sdRegister	LC_{user}
sdAddMovie	LC_{ru}
sdRateMovie	LC_{ru}
sdBrowse	LC_{ru}

V2 For each biddable domains User/RegisteredUser exactly one life-cycle exists, namely LC_{user} / LC_{ru}.

V3 The life-cycles are consistent with the state predicates in the sequence-diagrams.

* Browse has no state predicates at the beginning and end. Hence, it can be executed an arbitrary number of times.

* RateMovie can be executed if a movie is available in the database beforehand.

* Register and AddMovie has no state predicates at the beginning. Hence, it can be executed an arbitrary number of times.

V4 The life-cycles are consistent with the pre- and postconditions in Operations and data specification.

* The sequence diagram Browse contains the operation getAvailableMovies. It has no precondition. Hence, it can be executed at any position of the life-cycle.

* The sequence diagram AddMovie contains the operation addMovie. It has no precondition. Hence, it can be executed at any position of the life-cycle.

V5 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.

2 Design

2.1 D1

2.1.1 Subproblem architecture I: Register

MRA_Register fits to the problem frame **update (2)**.

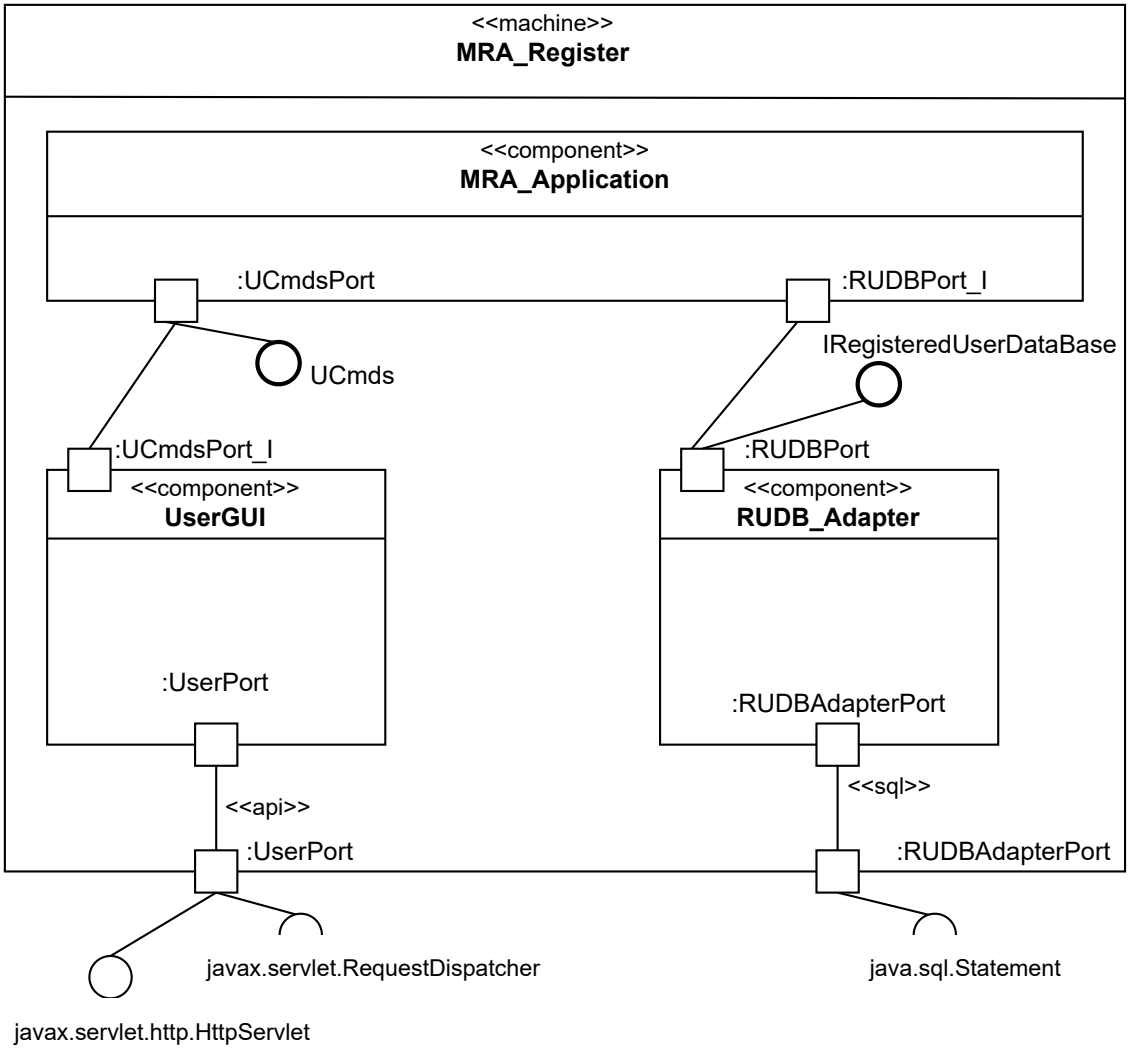
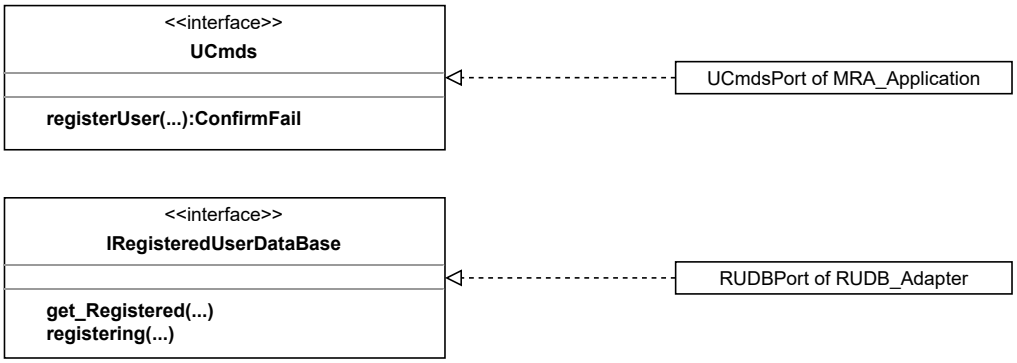
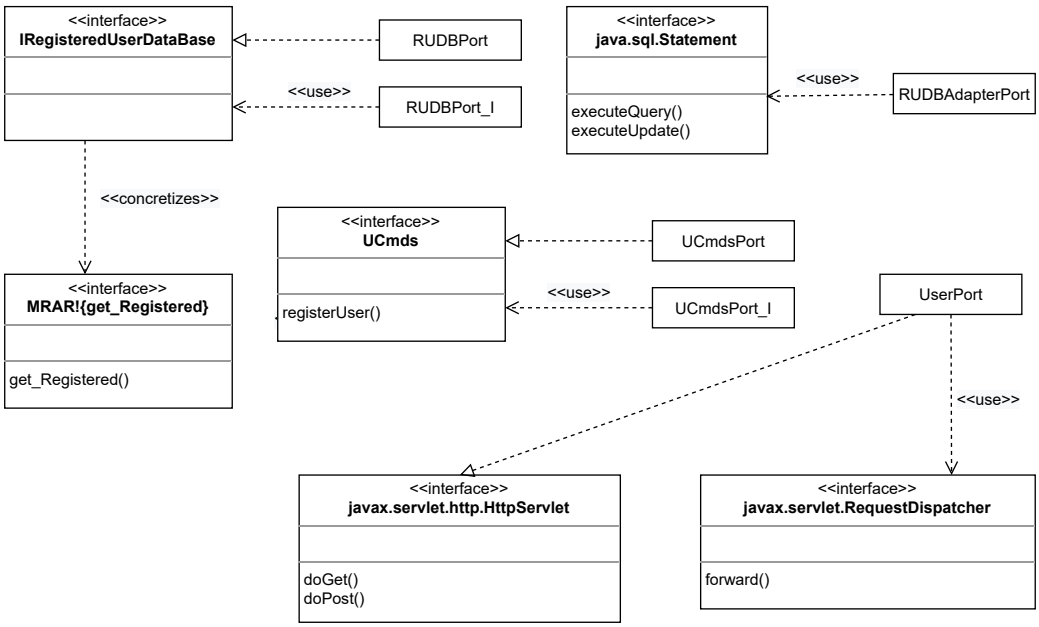


Figure 2.1: Instantiated architectural pattern for MRA_Register

Internal interfaces in MRA_Register:



Port types and interface relations MRA_Register:



2.1.2 Subproblem architecture II: AddMovie

MRA_AddMovie fits to the problem frame **update (2)**.

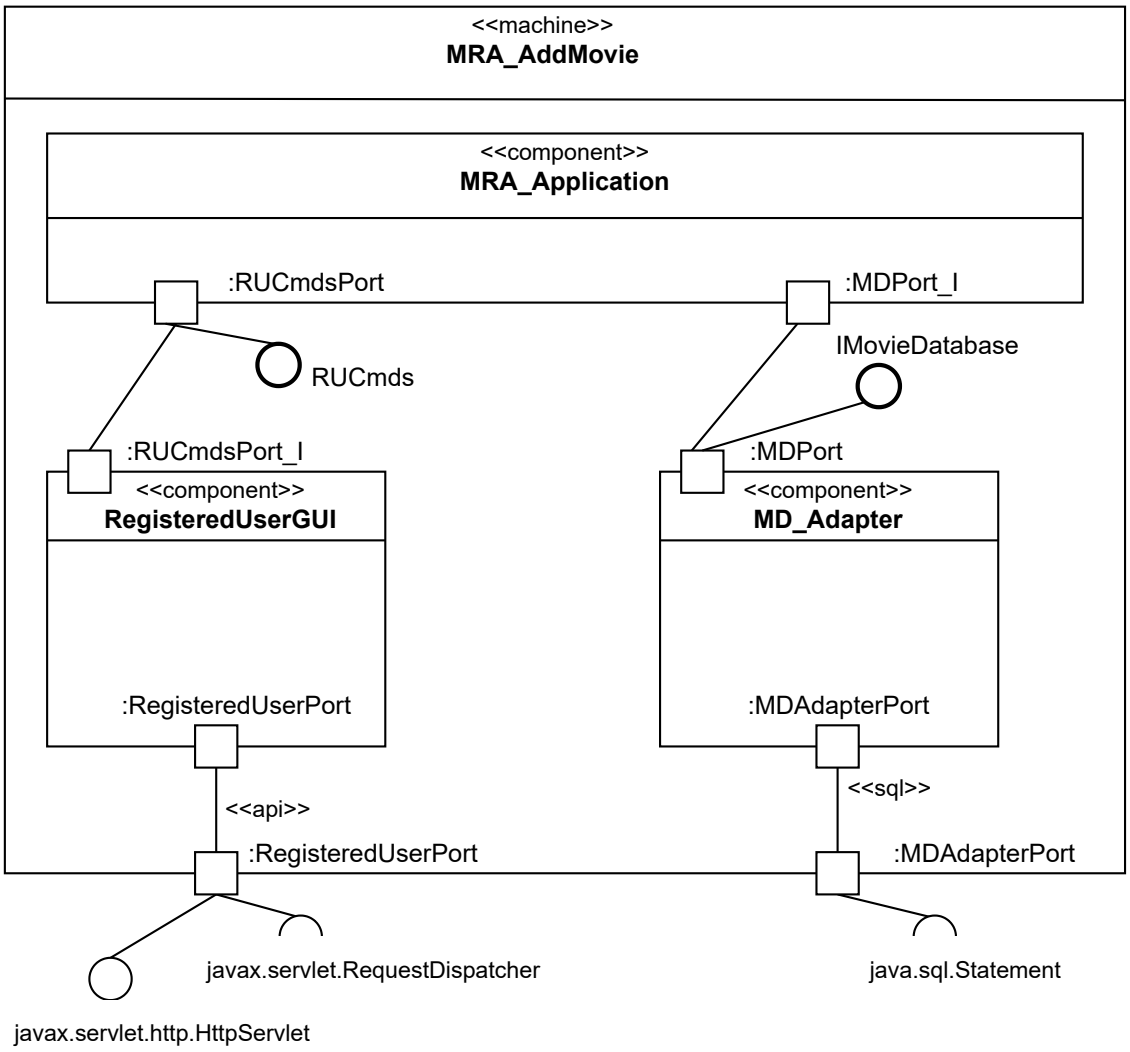
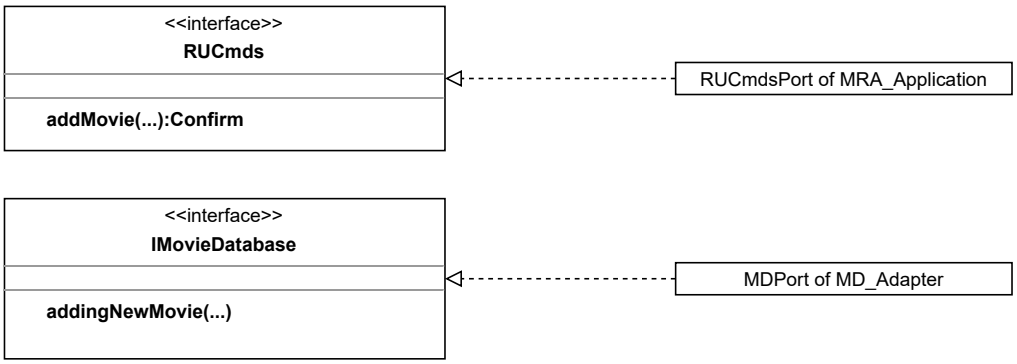
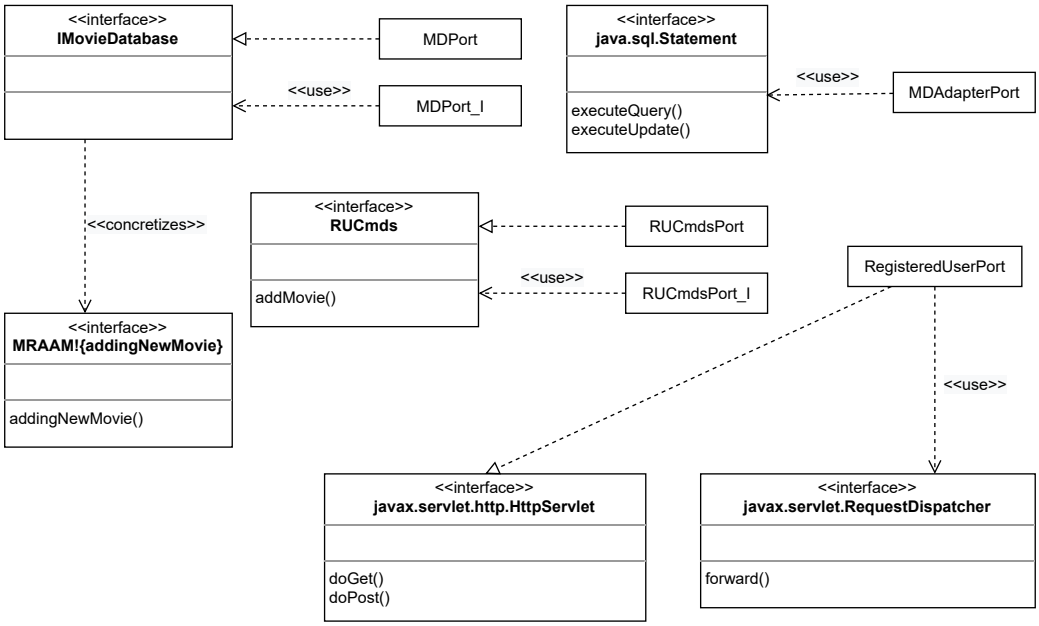


Figure 2.2: Instantiated architectural pattern for MRA_AddMovie

Internal interfaces in MRA_AddMovie:



Port types and interface relations MRA_AddMovie:



2.1.3 Subproblem architecture III: RateMovie

MRA_RateMovie fits to the problem frame **update (2)**.

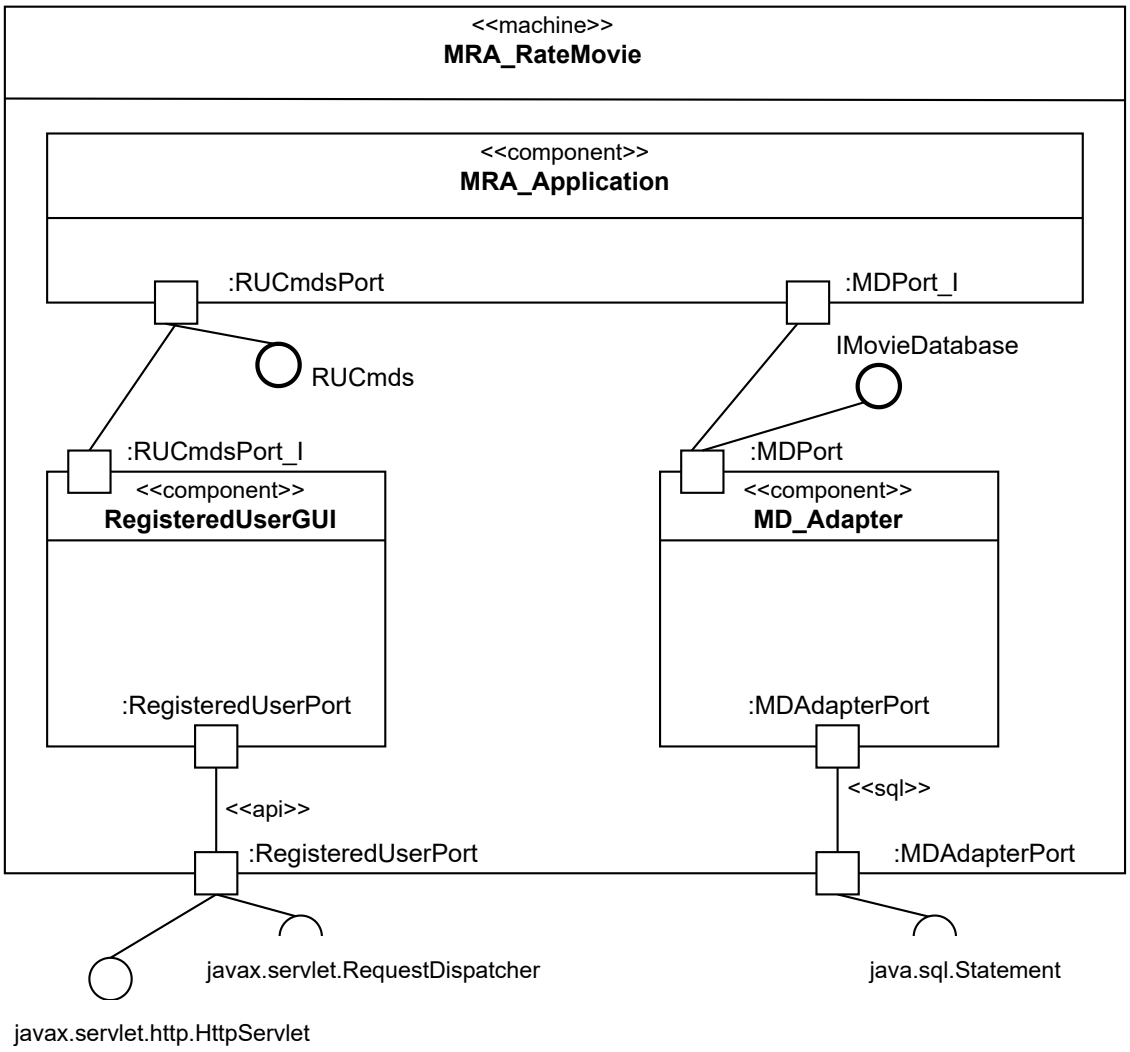
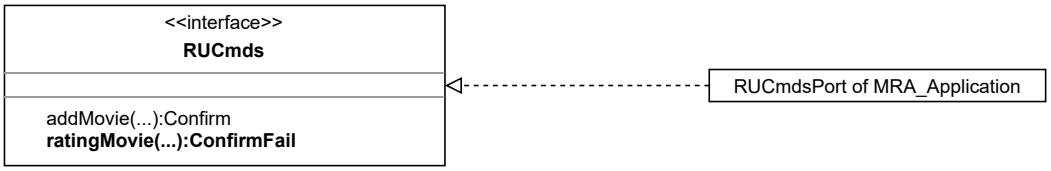


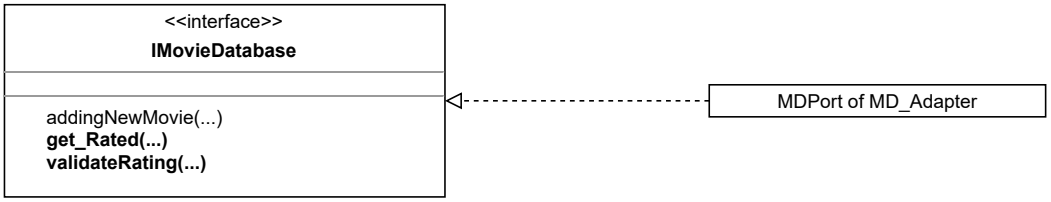
Figure 2.3: Instantiated architectural pattern for MRA_RateMovie

Internal interfaces in MRA_RateMovie:

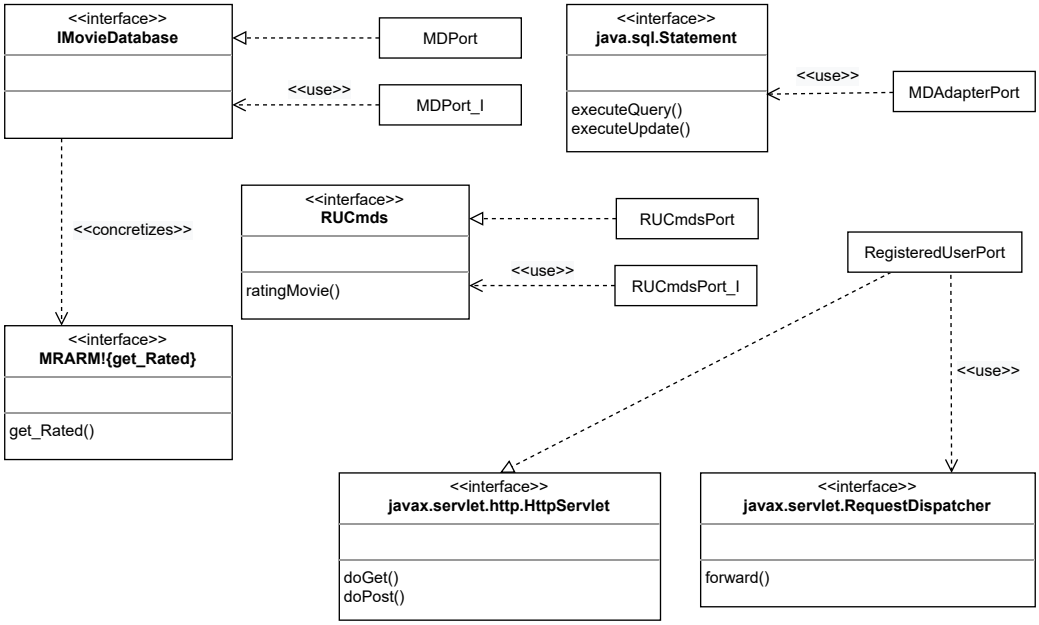
RUCmds from MRA_AddMovie is extended as follows:



IMovieDatabase from MRA_AddMovie is extended as follows:



Port types and interface relations MRA_RateMovie:



2.1.4 Subproblem architecture IV: Browse

MRA_Browse fits to the problem frame **query (2)**.

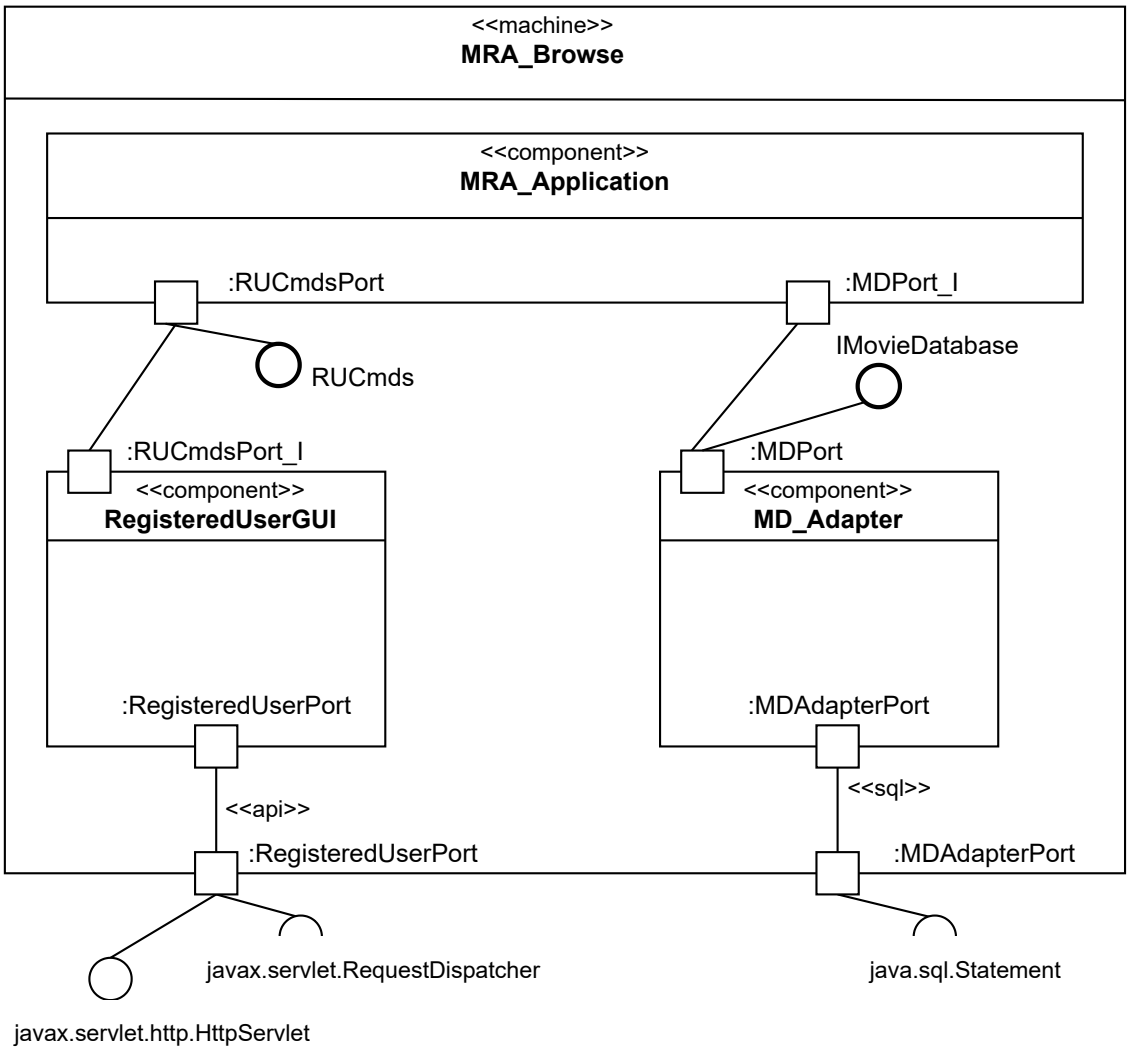
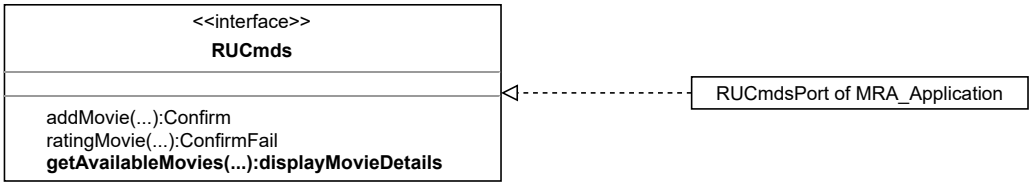


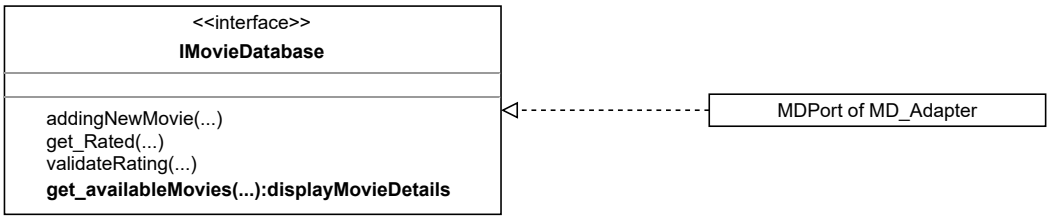
Figure 2.4: Instantiated architectural pattern for MRA_Browse

Internal interfaces in MRA_Browse:

RUCmds from MRA_RateMovie is extended as follows:

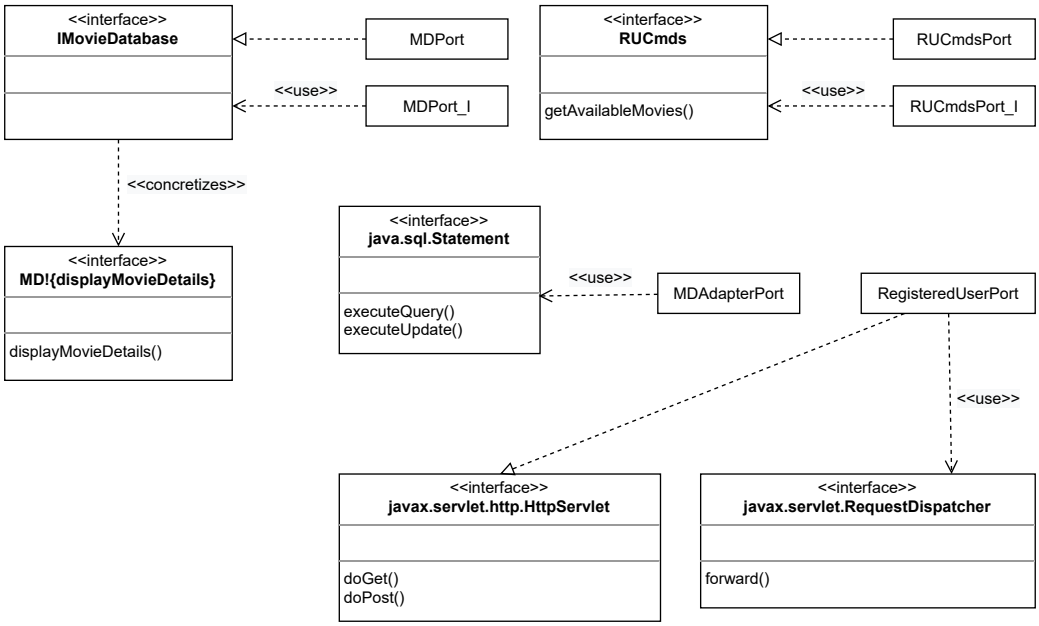


IMovieDatabase from MRA_RateMovie is extended as follows:



showAvailableMovies is realized by the return value displayMovieDetails of getAvailable-Movies.

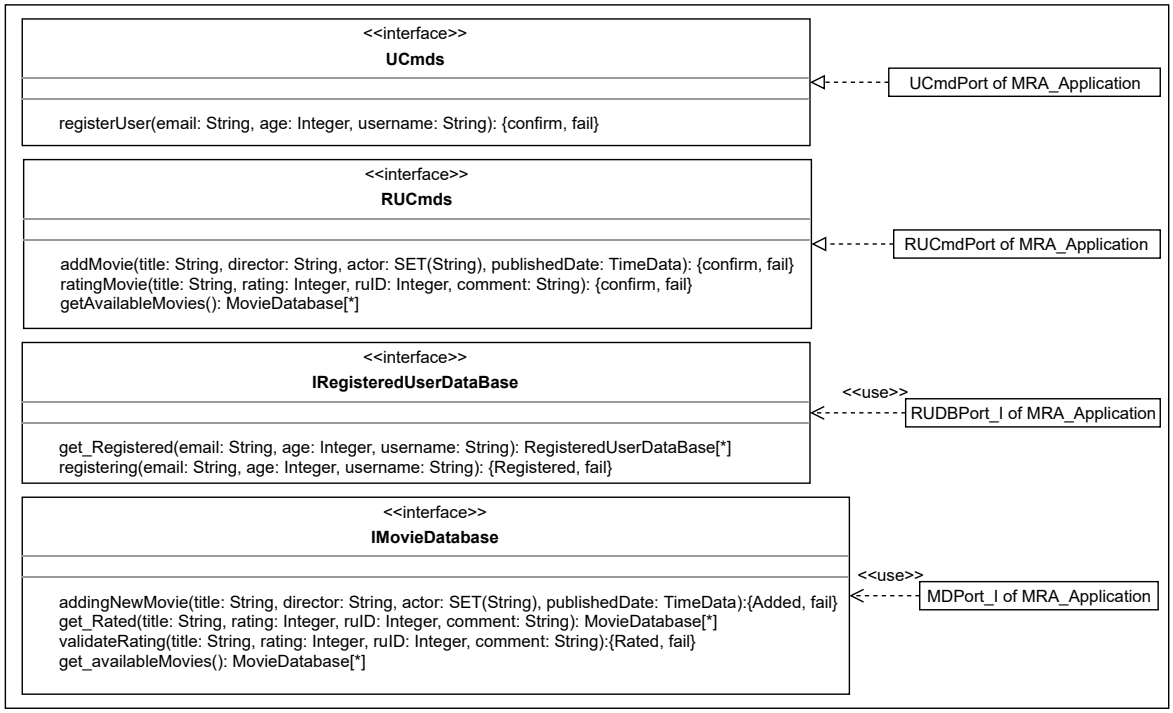
Port types and interface relations MRA_Browse:



The interface with the stereotype ≪sql≫ is a machine interface from the technical context diagram and will be defined later.

2.1.5 Refining interface classes

app_if



tech_if

Considered interface in subproblem architecture	**technical interface**
≪api≫ javax.servlet.http.HttpServlet in MRA_Register	≪api≫ AT!{doGet, doPost}
≪api≫ javax.servlet.RequestDispatcher in MRA_Register	≪api≫ MRA!{forward}
≪sql≫ java.sql.Statement in MRA_Register	≪call_return, sql≫ MRA!{executeQuery, executeUpdate}
≪api≫ javax.servlet.http.HttpServlet in MRA_AddMovie	≪api≫ AT!{doGet, doPost}
≪api≫ javax.servlet.RequestDispatcher in MRA_AddMovie	≪api≫ MRA!{forward}
≪sql≫ java.sql.Statement in MRA_AddMovie	≪call_return, sql≫ MRA!{executeQuery, executeUpdate}
≪api≫ javax.servlet.http.HttpServlet in MRA_RateMovie	≪api≫ AT!{doGet, doPost}
≪api≫ javax.servlet.RequestDispatcher in MRA_RateMovie	≪api≫ MRA!{forward}
≪sql≫ java.sql.Statement in MRA_RateMovie	≪call_return, sql≫ MRA!{executeQuery, executeUpdate}
≪api≫ javax.servlet.http.HttpServlet in MRA_Browse	≪api≫ AT!{doGet, doPost}
≪api≫ javax.servlet.RequestDispatcher in MRA_Browse	≪api≫ MRA!{forward}
≪sql≫ java.sql.Statement in MRA_Browse	≪call_return, sql≫ MRA!{executeQuery, executeUpdate}

adapter_if

There are no HAL components in the subproblem architectures. Hence, there are no **adapter if'** interface classes that need to be refined.

2.1.6 Merged architecture

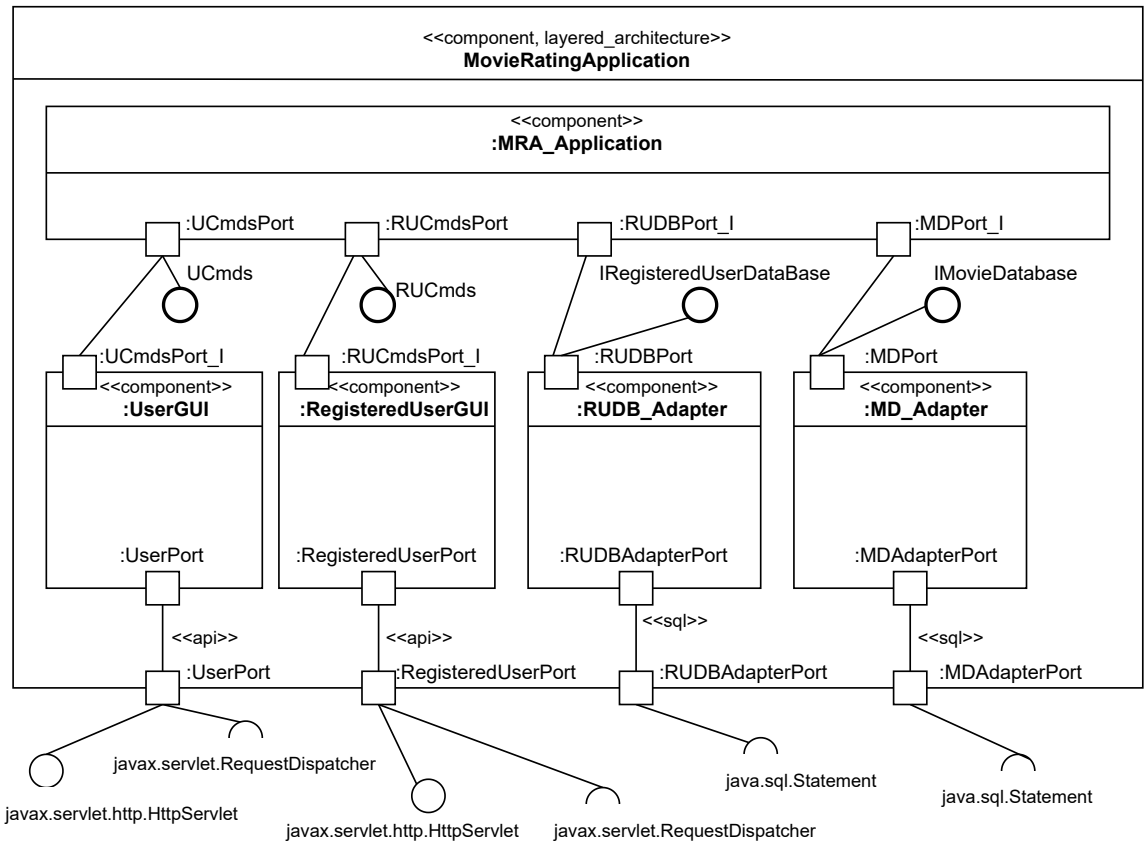


Figure 2.5: Merged architectures

2.1.7 Validation

V1 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	registerUser	in	UCmds::registerUser	provided
	get_Registered	out	IRegisteredUserDataBase::get_Registered	required
	registering	out	IRegisteredUserDataBase::registering	required
	failRegistration	out	return value of UCmds::registerUser	provided
	confirmRegistration	out	return value of UCmds::registerUser	provided
AddMovie	addMovie	in	RUCmds::addMovie	provided
	addingNewMovie	out	IMovieDatabase::addingNewMovie	required
	confirmAdding	out	return value of RUCmds::addMovie	provided
RateMovie	ratingMovie	in	RUCmds::ratingMovie	provided
	get_Rated	out	IMovieDatabase::get_Rated	required
	validateRating	out	IMovieDatabase::validateRating	required
	confirmRating	out	return value of RUCmds::ratingMovie	provided
	failRating	out	return value of RUCmds::ratingMovie	provided
Browse	getAvailableMovies	in	RUCmds::getAvailableMovies	provided
	get_availableMovies	out	IMovieDatabase::get_availableMovies	required
	displayMovieDetails	in	return value of IMovieDatabase::get_availableMovies	required
	showAvailableMovies	out	return value of RUCmds::getAvailableMovies	provided

V2 For global architecture: direction of all messages consistent to each other and input.

provided by machine	required by adapter / provided by app
javax.servlet.http.HttpServlet	UCmds / RUCmds

required by machine	required by adapter / provided by app
javax.servlet.RequestDispatcher	return values in UCmds / RUCmds
java.sql.Statement	IRegisteredUserDataBase / IMovieDatabase

V3 The external ports of the subproblem architectures and the global architecture correspond to the interfaces and connection types in the technical context diagram.

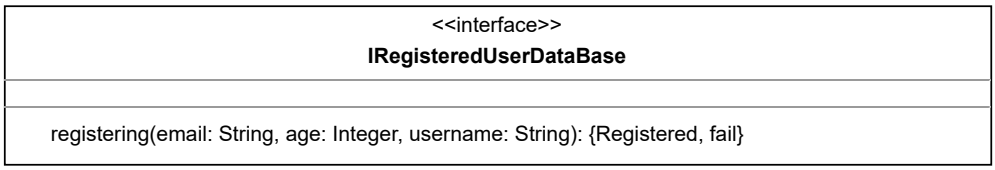
external port type	interface in architecture	required/provided	interface in technical context diagram
RegisteredUserPort	javax.servlet.http.HttpServlet	provided	AT!{doGet, doPost}
	javax.servlet.RequestDispatcher	required	MRA!{forward}
MDAdapterPort	java.sql.Statement	required	MRA!{executeQuery, executeUpdate}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

2.2 D2

2.2.1 Inter-component interaction - Register

Operation specification

```
context MRA_Register::registerUser(email : String, age : Integer,
    username : String)
pre: true
post: if ru.isUsernameUnique(username) and ru.isAgeAllowed(age)
    then ru->one(r: RUDB | r.email = email and r.age = age and r.
    username = username) and ru->size() = ru@pre->size()+1 and wu^
    confirmRegistration() else wu^failRegistration() endif
```



Sequence diagram

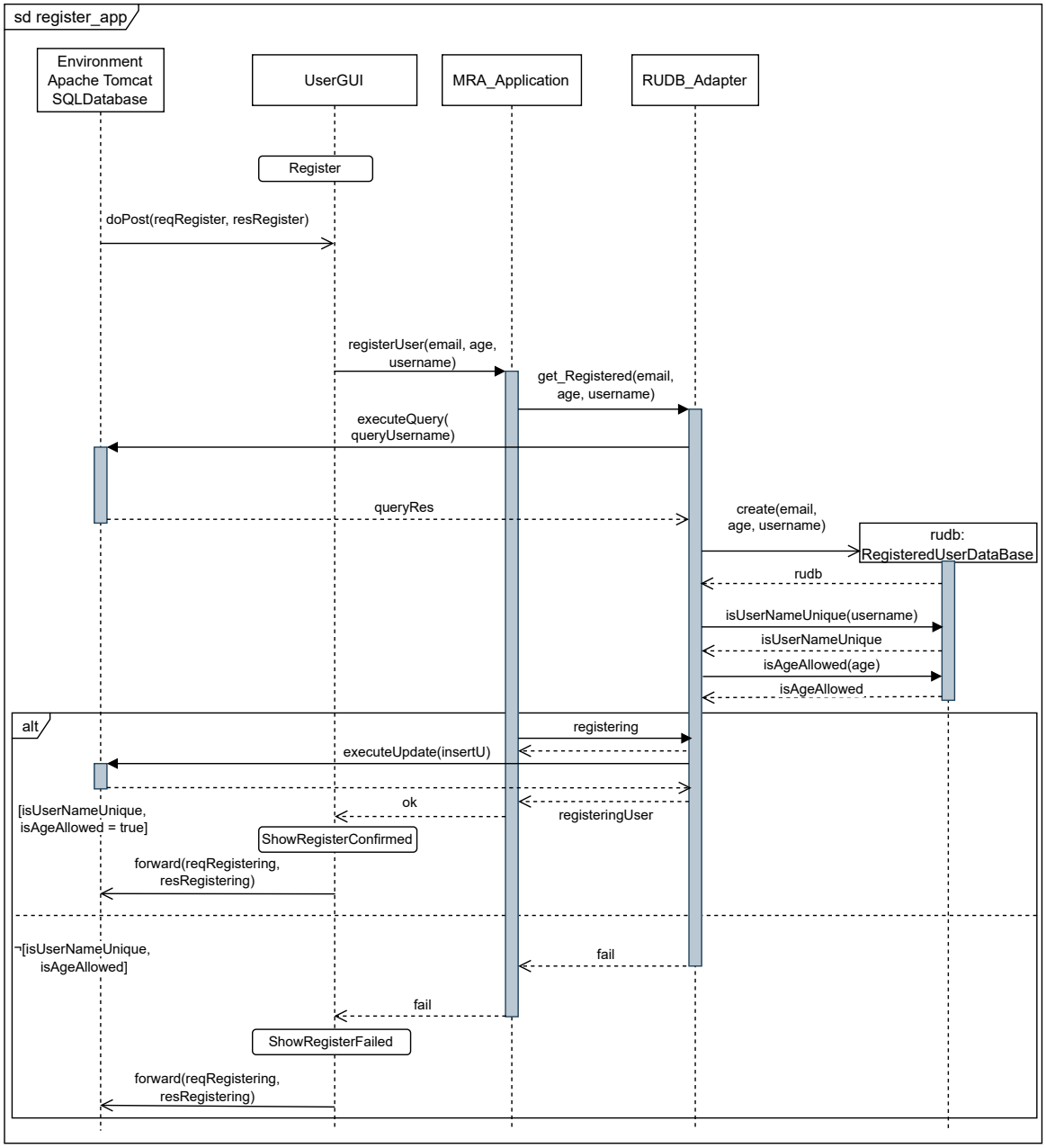


Figure 2.6: sdregister_app

Remarks

R1 reqRegister represent HTTPServletRequest objects containing the required user input.

R2 resRegister represent HttpServletResponse objects as the counterpart for the request.

R3 The state predicate Register represents that the input form for registering in the app.

R4 The state predicate ShowRegisterConfirmed represents that the the confirmation is shown.

R5 The state predicate ShowRegisterFailed represents that an error message is shown.

R6 forward(...) sends the request and response back to the server to generate the HTML webpage.

R7 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.

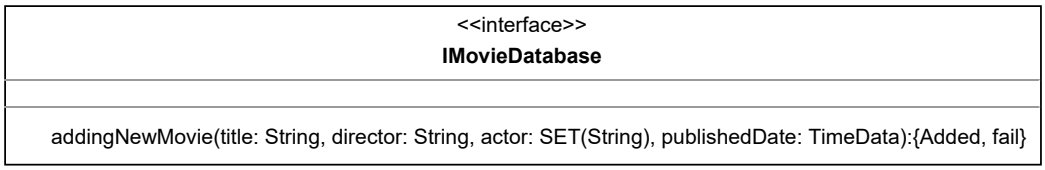
R8 **queryUsername**
SELECT * FROM RegisteredUserDataBase WHERE username = "username"

R9 **insertU**
INSERT INTO RegisteredUserDataBase(email, age, username) VALUES
("email", "age", "username")

2.2.2 Inter-component interaction - AddMovie

Operation specification

```
context MRA_AddMovie::addMovie(title : String, director : String,
    actor : Set(String), publishedDate : TimeData)
pre: true
post: if md.isMovieExist(title, director, actor, publishedDate)
    then md->one(a:MD |  a.title = title and a.director = director
    and a.actor=actor and a.publishedDate=publishedDate) and wru^
    confirmAdding() else wru^failAdding() endif
```



Sequence diagram

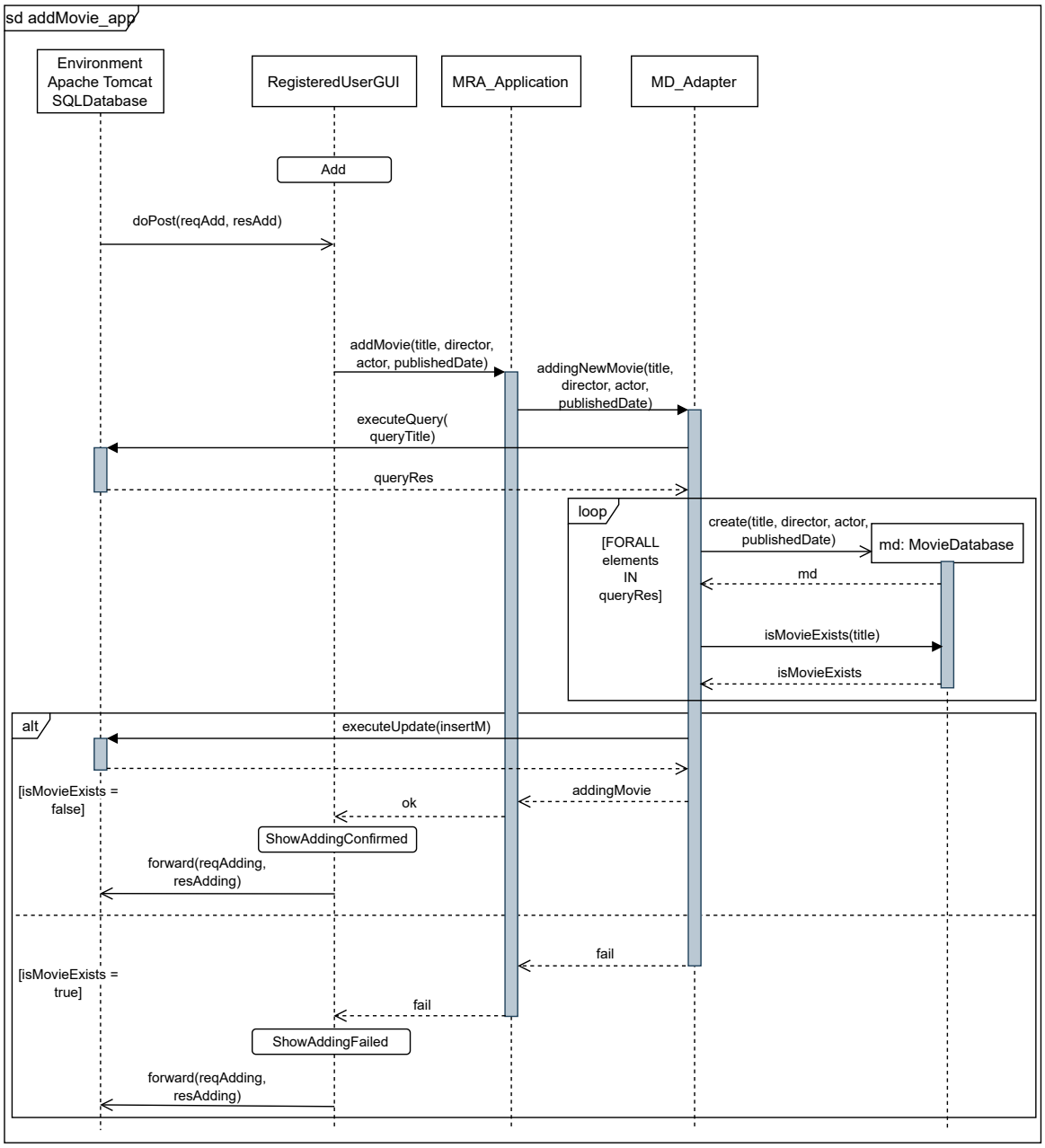


Figure 2.7: sdaddMovie_app

Remarks

R1 reqAdd represent HTTPServletRequest objects containing the required user input.

R2 resAdd represent HttpServletResponse objects as the counterpart for the request.

R3 The state predicate Add represents that the input form for adding a movie is shown.

R4 The state predicate ShowAddingConfirmed represents that the confirmation is shown.

R5 The state predicate ShowAddingFailed represents that an error message is shown.

R6 forward(...) sends the request and response back to the server to generate the HTML webpage.

R7 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.

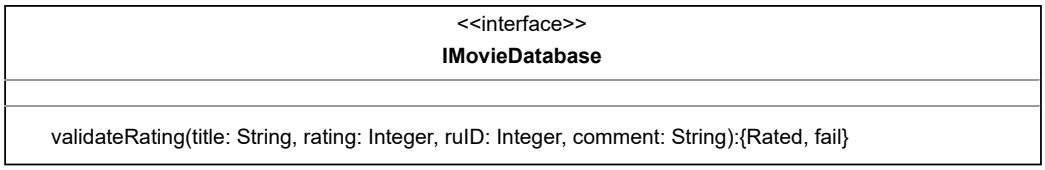
R8 **queryTitle**
SELECT * FROM MovieDatabase WHERE title = "title"

R9 **insertM**
INSERT INTO MovieDatabase(title, director, actor, publishedDate) VALUES ("title", "director", "actor", "publishedDate")

2.2.3 Inter-component interaction - RateMovie

Operation specification

```
context MRA_RateMovie::ratingMovie(title: String, rating: Integer,
    ruID: Integer)
pre: true
post: if md.isRatingExists(title, ruID, rating) md.isRatingCorrect
    (rating) then md->one(r.Rating | r.title = title and r.rating =
     rating and r.ruID = ruID and r.comment = comment) and wru^
    confirmRating() else wru^failRating() endif
```



Sequence diagram

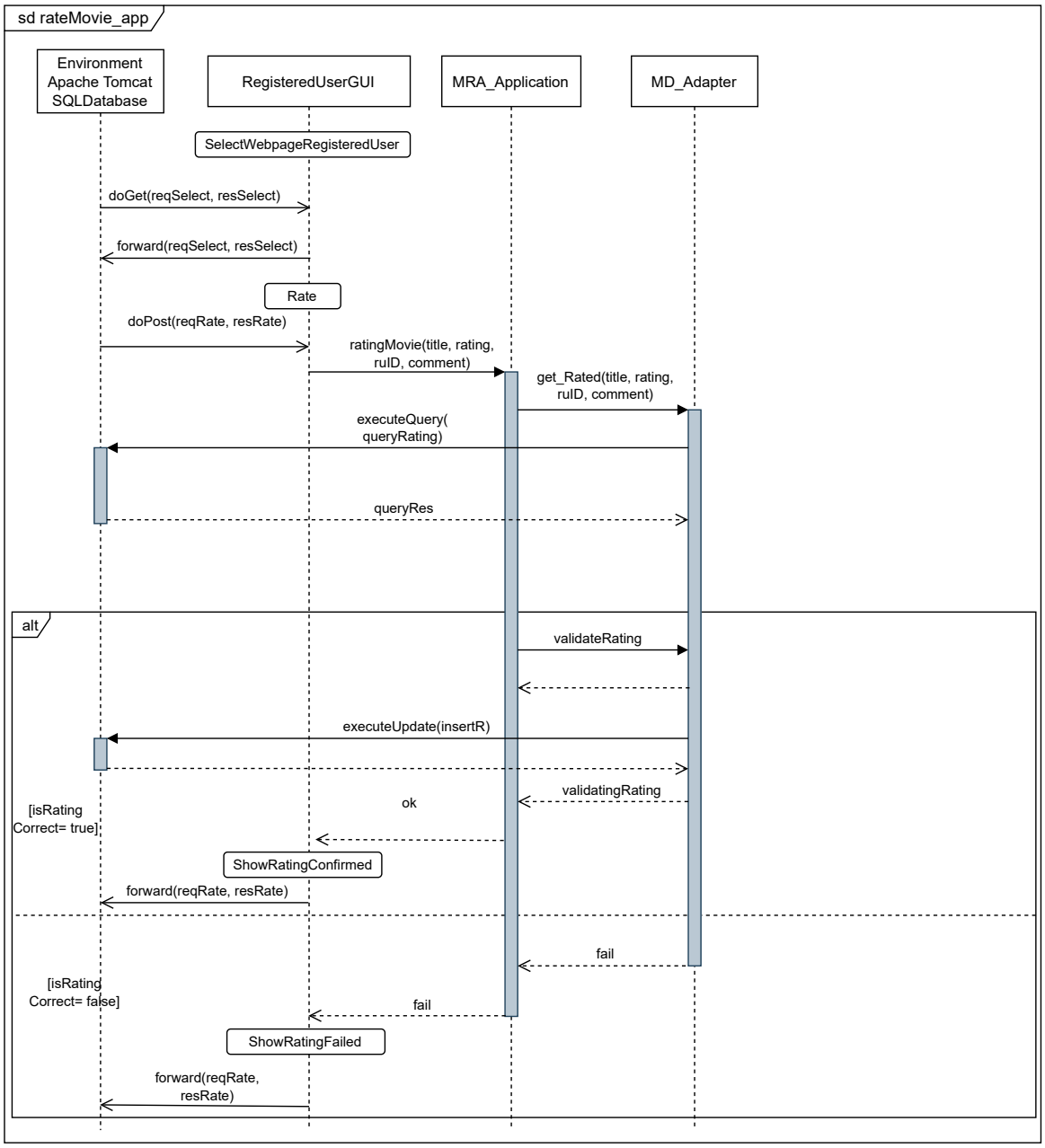


Figure 2.8: sdrateMovie_app

Remarks

R1 reqRate represent HTTPServletRequest objects containing the required user input.

R2 resRate represent HttpServletResponse objects as the counterpart for the request.

R3 The state predicate Rate represents that the input form for rating a movie is shown.

R4 The state predicate ShowRatingConfirmed represents that the the confirmation is shown.

R5 The state predicate ShowRatingFailed represents that an error message is shown.

R6 forward(...) sends the request and response back to the server to generate the HTML webpage.

R7 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.

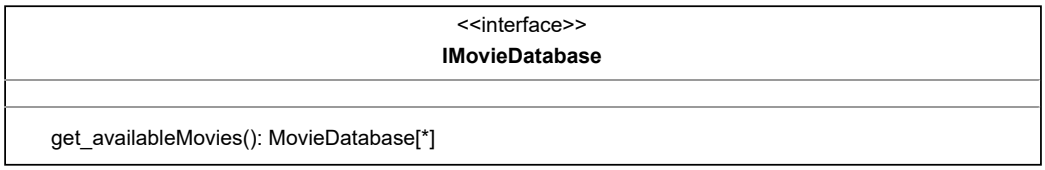
R8 **queryRating**
SELECT * FROM Rating WHERE title="title" AND ruID = "ruID"

R9 **insertR**
INSERT INTO Rating(ruID, title, rating, comment) VALUES
("RUID", "Title", "Rating", "Comment")

2.2.4 Inter-component interaction - Browse

Operation specification

```
context MRA_Browse::getAvailableMovies()
pre: true
post: let res : orderedSet(MovieDatabase) = md->select(m:
    MovieDatabase | m.title = title and m.rating = rating->sum()/
    rating->size() and m.comment = comment)->asOrderedSet() in wru^
    showAvailableMovies(res)
```



Sequence diagram

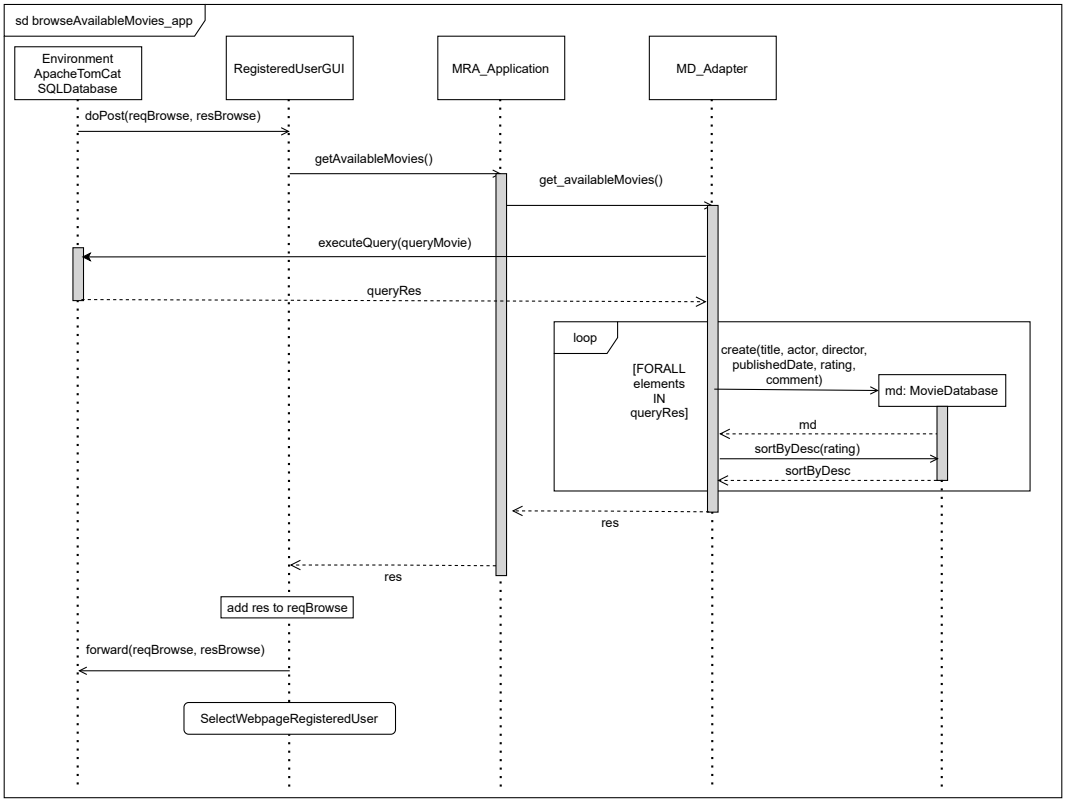


Figure 2.9: sdbrowseAvailableMovies_app

Remarks

R1 reqBrowse represent HTTPServletRequest objects containing the required user input.

R2 resBrowse represent HttpServletResponse objects as the counterpart for the request.

R3 The state predicate SelectRegisteredUserWebpage represents that the list of available offers is shown.

R4 md refers to an object of class MovieDatabase.

R5 res is a set of available movies that fit to the selection criteria. The set will be added to the request object to be processed by the server.

R6 forward(...) sends the request and response back to the server to generate the HTML webpage.

R7 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.

R8 **queryMovie**
SELECT * FROM MovieDatabase ORDER BY rating DESC

2.2.5 Validation

V1 The sequence diagrams must be consistent with the behavior described in Step A3: Abstract software specification.

Consistency of sdregister_app and sdRegister

Message in D2	**Corresponding message in A3**
doPost(reqRegister, resRegister)	refines register
registerUser(...)	registerUser
registering(...)	registering
executeQuery(queryUsername)	refines registering
executeUpdate(insertU)	refines registering
forward(reqRegistering,resRegistering)	refines representFailRegistration
forward(reqRegistering,resRegistering)	refines representConfirmRegistration

Consistency of sdaddMovie_app and sdAddMovie

Message in D2	**Corresponding message in A3**
doPost(reqAdd,resAdd)	refines addNewMovie
addMovie(...)	addMovie
addingNewMovie(...)	addingNewMovie
executeQuery(queryTitle)	refines addingNewMovie
executeUpdate(insertM)	refines addingNewMovie
forward(reqAdding,resAdding)	refines representFailAdding
forward(reqAdding,resAdding)	refines representConfirmAdding

Consistency of sdrateMovie_app and sdRateMovie

Message in D2	**Corresponding message in A3**
doPost(reqRate,resRate)	refines rateMovie
ratingMovie(...)	ratingMovie
validateRating(...)	validateRating
executeQuery(queryRating)	refines validateRating
executeUpdate(insertR)	refines validateRating
forward(reqRating,resRating)	refines representFailRating
forward(reqRating,resRating)	refines representConfirmRating

Consistency of sdbrowseAvailableMovies_app and sd Browse

Message in D2	**Corresponding message in A3**
doPost(reqBrowse,resBrowse)	refines browseAvailableMovies
getAvailableMovies(...)	getAvailableMovies
get_availableMovies(...)	get_availableMovies
executeQuery(...)	refines get_availableMovies
available(...)	refines get _AvailableMovies
forward(reqBrowse,resBrowse)	refines moviesCatalogue

V2 The sequence diagrams must be consistent with the behavior described in Step A6: Software lifecycle.

Consistency with life-cycle

Browse and Register:
sdbrowseAvailableMovies_app and sdregister_app can be both executed an arbitrary number of times without a precondition.

AddMovie—(BrowseMovie; [RateMovie]):
The sequence of sdaddMovie_app and sdbrowseAvailableMovies_app can be executed concurrently with sdrateMovie_app without unwanted side-effects.

V3 The sequence diagrams must realize the operations described in Step A5: Operations and data specification.

getAvailableMovies(...) is realized in sdbrowseAvailableMovies_app
- Precondition: does not have to be established, because it is true.
- Postcondition: MRA_Application delegates the message to MD_Adapter. Using the SQL command queryMovie, MD_Adapter selects the movie. The movie is then returned to MRA_Application. MRA_Application forwards the result to UserGUI. That realizes wruˆshowAvailableMovies(res).

registerUser(...) is realized in sdRegister_app
- Precondition: is established, because before registerUser is executed, an existing username is selected by the message doGet(reqRegister,resRegister).
- Postcondition: is established, because the requested username is first selected from the database by SQL command queryUsername and then it is checked if it is available. If it is available, then a corresponding user is added using insertU. The UserGUI is instructed to show the confirmation by the return value ok. If it is not available then the UserGUI is instructed to show the fail information by the return value fail.

addMovie(...) is realized in sdAddMovie_app
- Precondition: is established, because before addMovie is executed, an existing title is selected by the message doGet(reqMovie,resMovie).
- Postcondition: is established, because the requested title is first selected from the database by SQL command queryTitle and then it is checked if it is available. If it is available, then a corresponding movie is added using insertM. The UserGUI is instructed to show the confirmation by the return value ok. If it is not available then the UserGUI is instructed to show the fail information by the return value fail.

rateMovie(...) is realized in sdRateMovie_app
- Precondition: is established, because before rateMovie is executed, an existing rating is selected by the message doGet(reqRate,resRate).
- Postcondition: is established, because the requested rating is first selected from the database by SQL command queryRating and then it is checked if it is available. If it is available, then a corresponding rating is added using insertR. The UserGUI is instructed to show the confirmation by the return value ok. If it is not available then the UserGUI is instructed to show the fail information by the return value fail.

V4 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**
UCmds	registerUser	sdRegister_app
RUCmds	getAvailableMovies addMovie ratingMovie	sdbrowseAvailableMovies_app sdaddMovie_app sdrateMovie_app
IRegisteredUserDataBase	get_Registered, registering	sdregister_app
IMovieDatabase	addingNewMovie get_Rated, validateRating get_availableMovies	sdaddMovie_app sdrateMovie_app sdbrowseAvailableMovies_app

V5 The directions of messages must be consistent with the required and provided interfaces of Step D1: Software architecture.

Interface **Message**	**Provided by** **Recipient**	**Required by** **Sender**
UCmds	MRA_Application	UserGUI
registerUser	MRA_Application	UserGUI
RUCmds	MRA_Application	UserGUI
getAvailableMovies	MRA_Application	UserGUI
addMovie	MRA_Application	UserGUI
ratingMovie	MRA_Application	UserGUI
IRegisteredUserDataBase	RUDB_Adapter	MRA_Application
get_Registered	RUDB_Adapter	MRA_Application
registering	RUDB_Adapter	MRA_Application
IMovieDatabase	MD_Adapter	MRA_Application
addingNewMovie	MD_Adapter	MRA_Application
get_Rated	MD_Adapter	MRA_Application
validateRating	MD_Adapter	MRA_Application
get_availableMovies	MD_Adapter	MRA_Application

V6 Messages must connect components as connected in the software architecture of Step D1: Software architecture.

Component	**Connected components in architecture**	**Connected components in sequence diagrams**
MRA_Application	UserGUI, RUDB_Adapter, MD_Adapter	UserGUI, RUDB_Adapter, MD_Adapter
RUDB_Adapter	MRA_Application, Environment	MRA_Application, Environment
MD_Adapter	MRA_Application, Environment	MRA_Application, Environment
UserGUI	MRA_Application, Environment	MRA_Application, Environment

2.3 D3

Preliminary architectural description of RUDB Adapter

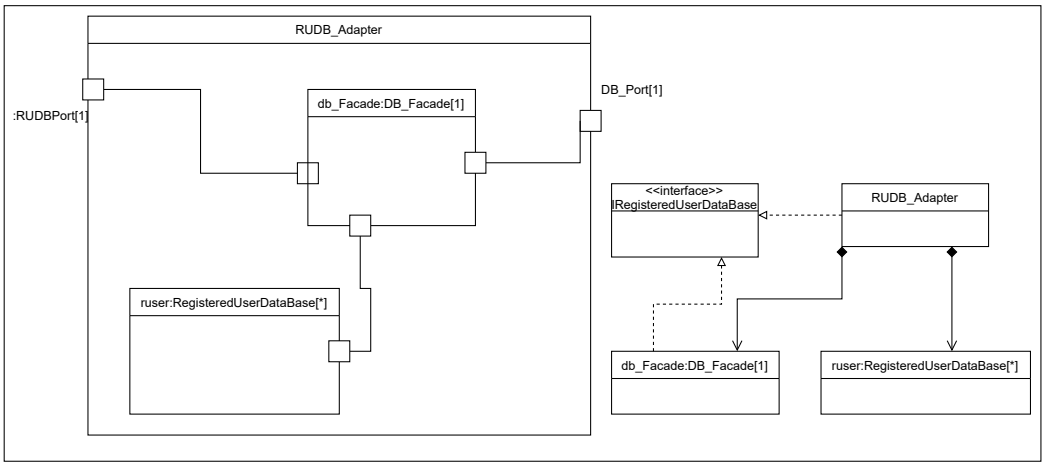


Figure 2.10: Preliminary architecture of RUDB Adapter

Final architectural description of RUDB Adapter

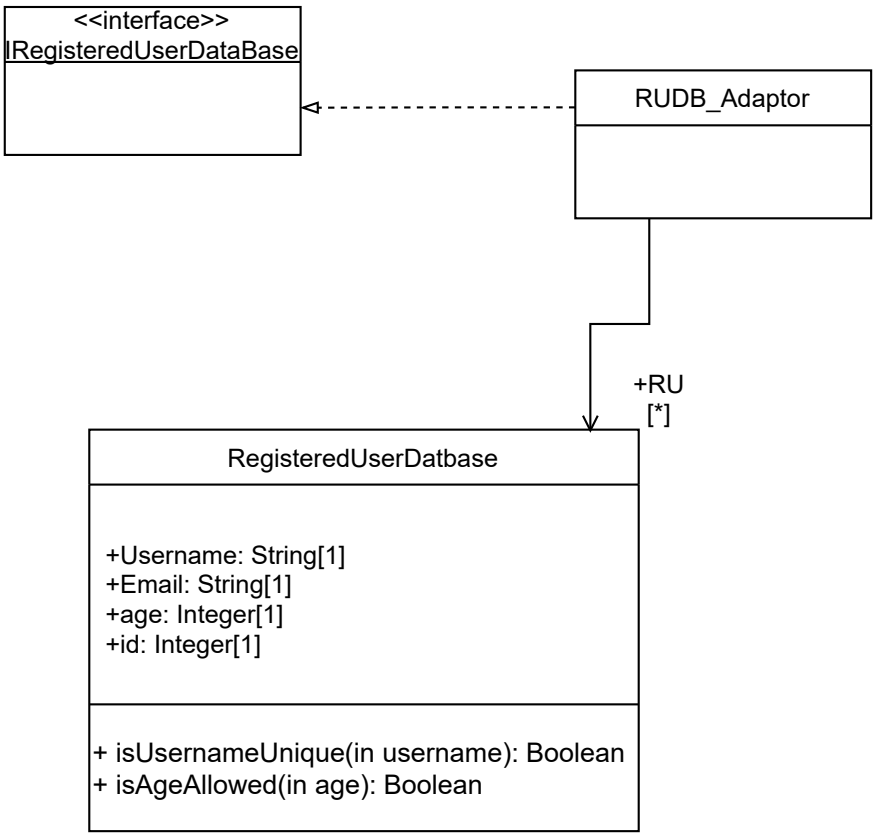


Figure 2.11: Final Architecture of RUDB Adapter

2.3.1 Validation

V1 Sequence diagrams of one component must be consistent with the corresponding interface behavior in step D2: Inter-Component Interaction.

Messages of the sequence diagram sd register_intra in step D3: Intra-Component Interaction	**Messages of the sequence diagram sd register_app in step D2: Inter-Component Interaction**
get_Registered(...)	get_Registered(...)
executeQuery(queryUsername)	executeQuery(queryUsername)
create(...)	Refinement
executeUpdate(...)	executeUpdate(...)
create(...)	Refinement

V2 It must be possible to relate any new state (predicates) to the state predicates of Step D2: Inter-Component Interaction.
No new state (predicates) is introduced in step D3: Intra-Component Interaction.

2.4 D4

1 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 guest. It is not necessary to create a state machine for this component.

2 The component RUDB Adapter: there exists a refinement in Step D3: Intra-Component Interaction. Hence, we have to look at the DBFacade, no state machine is necessary for this component. Furthermore, it is not necessary to create state machines for the components RegisterUserDataBase as the data base with its corresponding DBMS handles the states and state changes.

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

4 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.

5 The component MDAdapter: no refinement exists in Step D3: Intra-Component Interaction; continue with looking at Step D2: Inter-Component Interaction. There are no states. Therefore, no state machine is necessary.

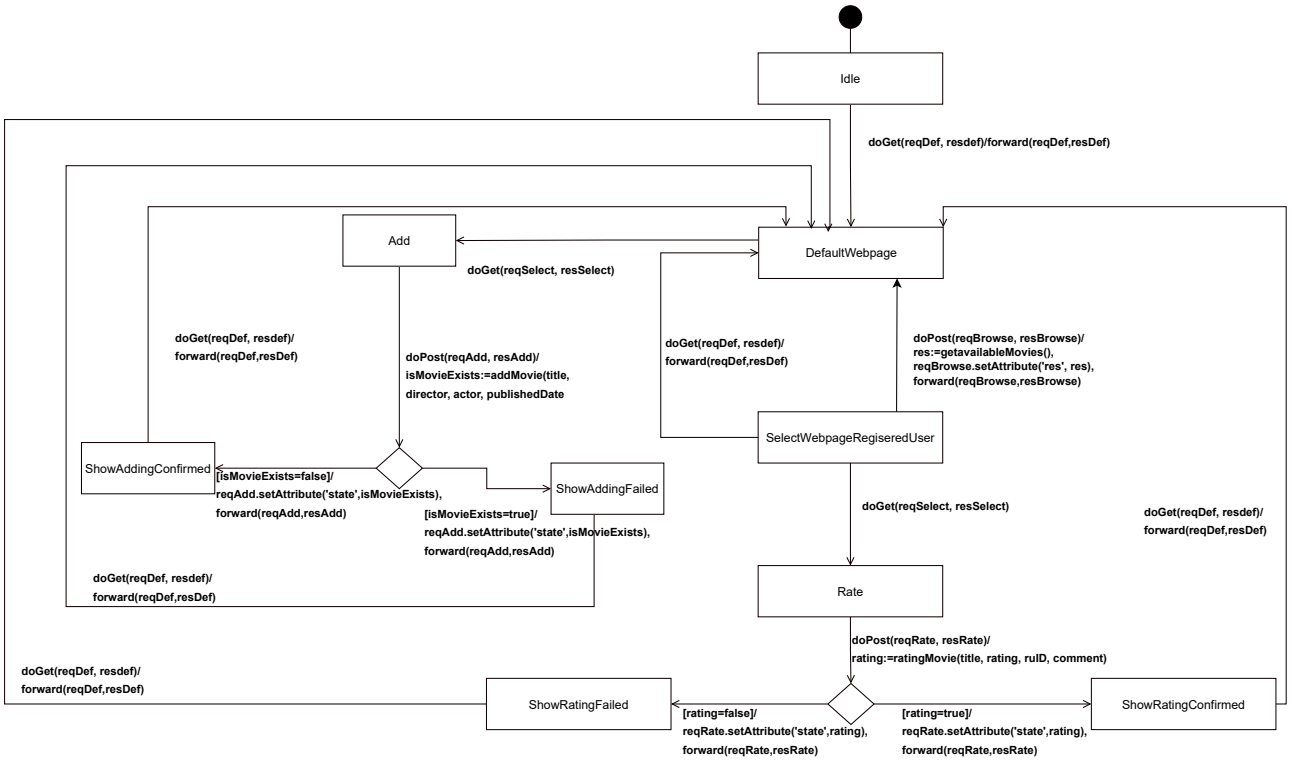


Figure 2.12: State Machine RegisteredUserGUI

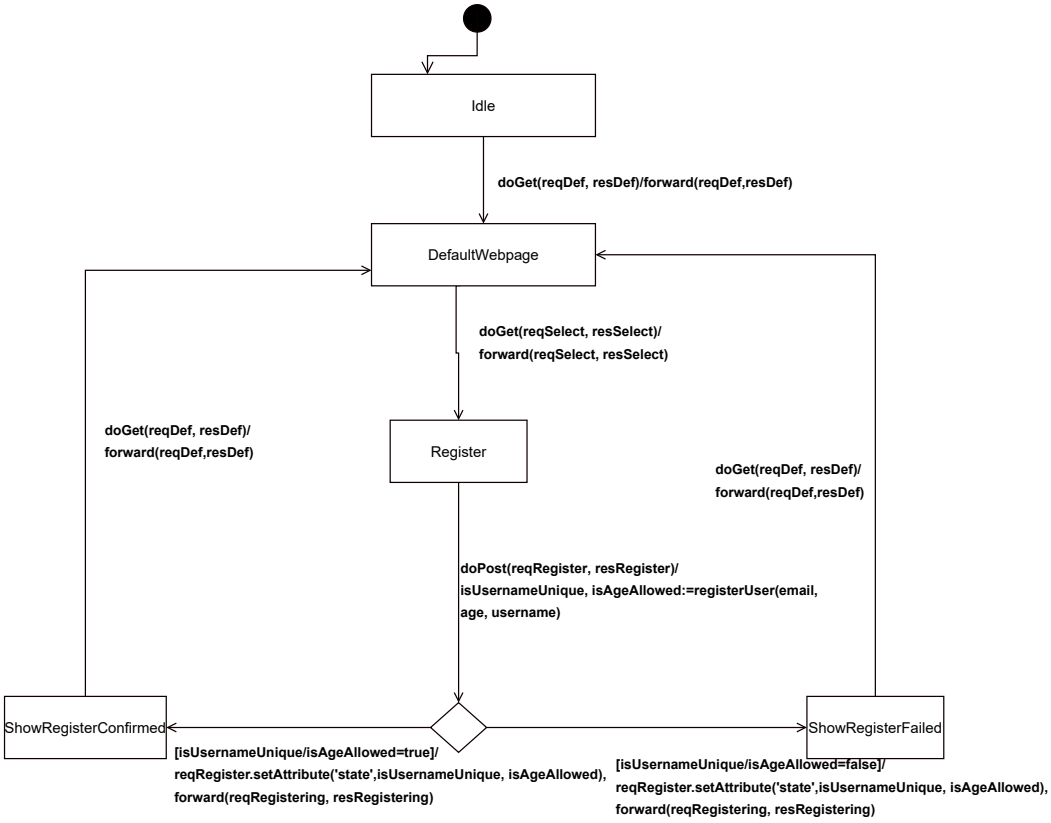


Figure 2.13: State Machine UserGUI

2.4.1 Validation

V1 The state machines describe the same behavior as in Step D2: Inter-Component Interaction or Step D3: Intra-Component Interaction

RegisteredUserGUI

Source State	**Target State**	**Input Signal**	**Mapped to message(s)**	**Output Signal**	**Mapped to message(s)**
init	DefaultWebpage	doGet(reqDefault,...)	-	forward(reqDefault,...)	-
DefaultWebpage	SelectWebpage-RegisteredUser	doPost(reqBrowse,...)	doPost(reqBrowse,...)	res:=getavailableMovies(), reqBrowse.setAttribute(...), forward(reqBrowse,...)	getavailableMovies(), forward(reqBrowse,...)
SelectWebpage-RegisteredUser	DefaultWebpage	doGet(reqDef,...)	-	forward(reqDef,...)	-
SelectWebpage-RegisteredUser	Add	doGet(reqSelect,...)	doGet(reqSelect,...)	forward(reqSelect,...)	forward(reqSelect,...)
SelectWebpage-RegisteredUser	Rate	doGet(reqSelect,...)	doGet(reqSelect,...)	forward(reqSelect,...)	forward(reqSelect,...)

Add	ShowAdding-Confirmed	doPost-(reqAdd,...)	doPost-(reqAdd,...)	result:= addMovie(...), reqAdd.setAttribute(...), forward(reqAdd,...)	addMovie(...), forward(reqAdd,...)
Add	ShowAdding-Failed	doPost-(reqAdd,...)	doPost-(reqAdd,...)	result:= addMovie(...), reqAdd.setAttribute(...), forward(reqAdd,...)	addMovie(...), forward(reqAdd,...)
Rate	ShowRating-Confirmed	doPost-(reqRate,...)	doPost-(reqRate,...)	result:= ratingMovie(...), reqRate.setAttribute(...), forward(reqRate,...)	ratingMovie(...), forward(reqRate,...)
Rate	ShowRating-Failed	doPost-(reqRate,...)	doPost-(reqRate,...)	result:= ratingMovie(...), reqRate.setAttribute(...), forward(reqRate,...)	ratingMovie(...), forward(reqRate,...)
ShowRating-Confirmed	DefaultWebpage	doGet-(reqDef,...)	-	forward(reqDef,...)	-
ShowRating-Failed	DefaultWebpage	doGet-(reqDef,...)	-	forward(reqDef,...)	-
ShowAdding-Confirmed	DefaultWebpage	doGet-(reqDef,...)	-	forward(reqDef,...)	-
ShowAdding-Failed	DefaultWebpage	doGet-(reqDef,...)	-	forward(reqDef,...)	-

V2 The state machines describe the same behavior as in Step D2: Inter-Component Interaction or Step D3: Intra-Component Interaction

All states are covered by a life-cycle:

Component RegisteredUserGUI

$LC_{ru} = (AddMovie|(BrowseMovie;[RateMovie]))^*$

State	**Covered by Life Cycle Part**
Init	BrowseMovie
DefaultWebpage	BrowseMovie
SelectWebpageRegisteredUser	BrowseMovie
Rate	RateMovie
ShowRatingFailed	RateMovie
ShowRatingConfirmed	RateMovie
Add	AddMovie
ShowAddingFailed	AddMovie
ShowAddingConfirmed	AddMovie

All transitions are covered by a life-cycle:

Component RegisteredUserGUI

$LC_{ru} = (AddMovie|(BrowseMovie;[RateMovie]))^*$

Source State	**Target State**	**Input Signal**	**Output Signal**	**Life cycle part**
init	DefaultWebpage	doGet(reqDefault,...)	forward(reqDefault,...)	(BrowseMovie

DefaultWebpage	SelectWebpageRegisteredUser	doPost(reqBrowse,...)	res:=getavailableMovies(), reqBrowse.setAttribute(...), forward(reqBrowse,...)	**BrowseMovie**
SelectWebpageRegisteredUser	DefaultWebpage	doGet(reqDef,...)	forward(reqDef,...)	(BrowseMovie)*
SelectWebpageRegisteredUser	Add	doGet(reqSelect,...)	forward(reqSelect,...)	$(BrowseMovie\|AddMovie)^*$
SelectWebpageRegisteredUser	Rate	doGet(reqSelect,...)	forward(req-Select,...)	(BrowseMovie; RateMovie)*
Add	ShowAddingConfirmed	doPost(reqAdd,...)	result:= addMovie(...), reqAdd.setAttribute(...), forward(reqAdd,...)	AddMovie
Add	ShowAddingFailed	doPost(reqAdd,...)	result:= addMovie(...), reqAdd.setAttribute(...), forward(reqAdd,...)	AddMovie
Rate	ShowRatingConfirmed	doPost(reqRate,...)	result:= ratingMovie(...), reqRate.setAttribute(...), forward(reqRate,...)	RateMovie
Rate	ShowRatingFailed	doPost(reqRate,...)	result:= ratingMovie(...), reqRate.setAttribute(...), forward(reqRate,...)	RateMovie
ShowRatingConfirmed	DefaultWebpage	doGet(reqDef,...)	forward(reqDef,...)	(RateMovie)*
ShowRatingFailed	DefaultWebpage	doGet(reqDef,...)	forward(reqDef,...)	(RateMovie)*
ShowAddingConfirmed	DefaultWebpage	doGet(reqDef,...)	forward(reqDef,...)	(AddMovie)*
ShowAddingFailed	DefaultWebpage	doGet(reqDef,...)	forward(reqDef,...)	(AddMovie)*

3 Implementation & Testing

3.1 I

3.2 T1

3.3 T2

3.4 T3

4 Glossary

Table 4.1: Glossary

Name	Type	Description	Source
A			
A		abbreviation for Administrator	CD
Adding	phenomenon	requirement R-II constraining connection domain to add a movie into the database	pdAddMovie
addingNewMovie	phenomenon	counterpart to addNewMovie	CD, pdAddMovie,ProblemFrames sdAddMovie, subArch II
addingRU	phenomenon	counterpart to addRU	CD
addingToMyList	phenomenon	counterpart to addToMyList	CD
addMovie	phenomenon	webpage notifying the machine that the user wants to add a new movie	pdAddMovie, ProblemFrames, sdAddMovie, CM, subArch II, State Machine
addNewMovie	phenomenon	a registered user can add a new movie into database	CD, pdAddMovie, ProblemFrames, sdAddMovie, CM
addRU	phenomenon	a registered user can add a registered user into a group	CD
addToMyList	phenomenon	add a movie to a registered user's list	CD
Administrator	biddable Domain	the leader of a group	R&D, CD
age	attribute	represents age of a user	CM
allowChatDiscussion	phenomenon	group members can chat in the group	CD
ApacheTomcat	connection Domain	An Open Source JSP and Servlet Container from the Apache Foundation	TCD
api	technical phenomenon	Application Programming Interface, that allows domains to talk to each other	TCD
B			
banGM	phenomenon	the administrator can ban group members	CD
banningGM	phenomenon	counterpart to banGM	CD
browsingGMsList	phenomenon	counterpart to browseGMsList	CD
browsingMovie	phenomenon	counterpart to browseMovie	CD, CM
browseGMsList	phenomenon	a group member can see the list of other group members' list	CD

Table 4.1: Glossary

Name	Type	Description	Source
browseMovie	phenomenon	a register user can see the list of all movies in the database	CD, pdBrowse, ProblemFrames, sdBrowse, CM
C			
call_return	technical phenomenon	SQL Commands: defined in FIPS PUB 127-2	TCD
Chat	phenomenon	section where conversations between group members occur	R&D
checkingLogIn	phenomenon	counterpart to login	CD
confirmAdding	phenomenon	machine informing web page about adding a new movie	pdAddMovie, ProblemFrames, sdAddMovie, TCD, CM
confirmedRating	state predicate	the rating process is cofirmed	sdRateMovie
confirmRating	phenomenon	machine informing web page that the rating was successful	pdRateMovie, ProblemFrames, sdRateMovie, TCD, CM
confirmRegistration	phenomenon	machine informing web page that the registration was successful	pdRegister, ProblemFrames, sdRegister, TCD, CM
confirmLogOut	phenomenon	counterpart to logout	CD
containUserData	phenomenon	the database contains all the users' and groups' information	CD
create	method	create a specified object	sd register_app_intra
createGroup	phenomenon	a registered user can create a group	CD
creatingGroup	phenomenon	counterpart to createGroup	CD
D			
day	attribute	stores the day in a month as an integer	CM, OCL
DBFacade	object	represents the database in D3	sd register_app_intra
DefaultWebpage	state	indicates the starting page	State Machine
deletingGroup	phenomenon	a group is being deleted	CD
discussingMovie	phenomenon	counterpart to discussMovie	CD
discussMovie	phenomenon	group members can discuss a movie in a chat form	CD
displayGroupMembers	phenomenon	all members' usernames and lists shown in group	CD
displayMovieDetails	phenomenon	the database provides movie title, director, actors, publishing date	CD, pdBrowse, ProblemFrames, sdBrowse, subArch IV
doGet	technical phenomenon	defined in abstract class javax.servlet.http.HttpServlet	TCD, subArch, State Machine

Table 4.1: Glossary

Name	Type	Description	Source
doPost	technical phenomenon	defined in abstract class javax.servlet.http.HttpServlet	TCD, subArch, sdregister_app, sdaddMovie_app, sdrateMovie_app, sdbrowseAvailable-Movies_app, State Machine
E			
email	attribute	represents email id of a user	CM
executeQuery	technical phenomenon	defined in interface java.sql.Statement	TCD, subArch, sdregister_app, sdaddMovie_app, sdrateMovie_app, sd register_app_intra
executeUpdate	technical phenomenon	defined in interface java.sql.Statement	TCD, subArch, sdregister_app, sdaddMovie_app, sdrateMovie_app, sd register_app_intra
F			
failRating	phenomenon	machine notifying webpage that the rating process has failed	pdRateMovie, ProblemFrames, sdRate-Movie, TCD, CM
failRegistration	phenomenon	notifying a webpage that the registration failed	pdRegister, Problem-Frames, sdRegister, TCD, CM
feedbackA	phenomenon	capture all the feedback going to the administrators	CD
feedbackGM	phenomenon	capture all the feedback going to the group members	CD
feedbackRU	phenomenon	capture all the feedback going to the registered users	CD
feedbackU	phenomenon	capture all the feedback going to the users	CD
forward	technical phenomenon	defined in interface javax.servlet.RequestDispatcher	TCD, subArch, State Machine
G			
GC		abbreviation for GroupChat	CD
GM		abbreviation for GroupMember	CD
getAvailableMovies	phenomenon	requesting available movies by a webpage to the machine	pdBrowse, Problem-Frames, sdBrowse, TCD, CM, OCL, , subArch IV, State Machine

Table 4.1: Glossary

Name	Type	Description	Source
get_availableMovies	message	returns all the available movies from the database	sdBrowse, subArch IV
get_Rated	message	checks all the conditions for a rating process	sdRateMovie, subArch III
get_Registered	message	checks all the conditions during registration	sdRegister, subArch I
GroupChat	lexical Domain, designed Domain	a place where group members can join, leave, or discuss a movie	CD
Group Member	biddable Domain	registered user who joined a group	R&D
gui	technical phenomenon	User interface of HTML webpages	TCD
H			
http	technical phenomenon	Hypertext Transfer Protocol, defined in RFC 2616	TCD
I			
id	attribute	represents unique id of a user	CM
idle	state	indicates that the server waits for incoming requests	State Machine
isUserNameUnique	message, auxiliary function	checks whether the username is unique	CM, sdregister_app
isAgeAllowed	message, auxiliary function	checks whether the user's age is at least 18	CM, sdregister_app
IRegisteredUserDataBase	interface	interface for RUDB	subArch I
IMovieDatabase	interface	interface for MD	subArch II, III, IV
J			
K			
L			
$LC_{MovieRatingApp}$	life-cycle	Combined life-cycle (all users and the internal operation)	LC
LC_{ru}	life-cycle	Life-cycle for one registered user	LC
LC_{user}	life-cycle	Life-cycle for one user	LC
leaveGroup	phenomenon	a registered user can leave a group	CD
leaveNdeleteGroup	phenomenon	the administrator can leave a group, then the group is deleted	CD
leavingGroup	phenomenon	counterpart to leaveGroup	CD
login	phenomenon	a registered user can login to start a session	CD
logout	phenomenon	a registered user can logout to end the session	CD
M			
MD		abbreviation for MovieDatabase	CD, PD, ProblemFrames, CM

Table 4.1: Glossary

Name	Type	Description	Source
MD_Adapter	component	responsible to create and maintain tables for all persistent classes	subArch II, III, IV
month	attribute	stores the month in a numerical format between 1 and 12	CM, OCL
MovieDatabase	lexical Domain, designed Domain	a database containing all the information about movies, their ratings and comments	CD, PD, SD, CM, OCL
MovieExists	state predicate	the movie is added into the database and exists	SD
Movies	phenomenon	requirement R-IV constraining connection domain to show movies from the database	pdBrowse
moviesCatalogue	phenomenon	webpage showing all the available movies details to the user	pdBrowse, ProblemFrames, sdBrowse
MRA		abbreviation for MovieRatingApp	CD, TCD, CM
MRA_AddMovie	machine Domain	machine introduced for adding a new movie into the database	pdAddMovie, sdAddMovies, CM, subArch II
MRAAM		abbreviation for MRA_AddMovie	pdAddMovie, ProblemFrames, TCD, CM
MRA_Browse	machine Domain	machine introduced for viewing all the list of movies from the database	pdBrowse, sdBrowse, CM, subArch IV
MRAB		abbreviation for MRA_Browse	pdBrowse, ProblemFrames, TCD, CM
MRA_RateMovie	machine Domain	machine introduced for rating movies from the database	pdRateMovie, sdRateMovie, CM, subArch III
MRARM		abbreviation for MRA_RateMovie	pdRateMovie, ProblemFrames, TCD, CM
MRA_Register	machine Domain	machine introduced for registering new users	pdRegister, sdRegister, CM, subArch I
MRAR		abbreviation for MRA_Register	pdRegister, ProblemFrames, TCD, CM
N			
O			
OCL		abbreviation for Object Constraint Language	OCL
P			
previewAvgRating	phenomenon	shows the calculated average rating of movie	CD
Q			

Table 4.1: Glossary

Name	Type	Description	Source
R			
Rating	phenomenon	requirement R-III constraining connection domain to rate a movie from the database	pdRateMovie
Rating	class	represents a movie rated by a registered user	CM RateMovie
RatingExists	state predicate	the movie is successfully rated and stored into the database	sdRateMovie
ratingMovie	phenomenon	webpage informing machine that the user is rating a movie	pdRateMovie, ProblemFrames, sdRateMovie, TCD, subArch III, State Machine
rateMovie	phenomenon	a registered user can rate a movie with a optional comment	CD, pdRateMovie, ProblemFrames, sdRateMovie, CM, OCL
register	phenomenon	a user can register in the app entering his Email, age and a username	CD, pdRegister, ProblemFrames, sdRegister, CM
registerUser	phenomenon	webpage requesting the machine to register the user	pdRegister, ProblemFrames, CM
Registered User	biddable Domain	person who can login and use app functionality	R&D
RegisteredUserDataBase	lexical Domain, designed Domain	organised collection of users' and groups' data incl. chat	CD, pdRegister, sdRegister, TCD, CM, subArch
RegisteredUserGUI	component	web interface for registered users	subArch II, III, IV
registering	phenomenon	counterpart to register	CD, pdRegister, ProblemFrames, sdRegister, subArch I
Registering	class	represents a new user being registered	CM
Registration	phenomenon	requirement R-I constraining connection domain to register a new user	pdRegister
representConfirmAdding	phenomenon	webpage feedbacking user about adding a new movie into the database	pdAddMovie, ProblemFrames, sdAddMovie
representConfirmRating	phenomenon	webpage notifying the user about a successfull rating	pdRateMovie, ProblemFrames, sdRateMovie
representConfirmRegistration	phenomenon	webpage notifying successfull registration to the user	pdRegister, ProblemFrames, sdRegister
representFailRating	phenomenon	webpage notifying user that rating process has failed	pdRateMovie, ProblemFrames, sdRateMovie
representFailRegistration	phenomenon	webpage notifying user that registration has failed	pdRegister, ProblemFrames, sdRegister

Table 4.1: Glossary

Name	Type	Description	Source
RU		abbreviation for RegisteredUser	CD, PD, ProblemFrames
RUDB		abbreviation for RegisteredUserDataBase	CD, CM
RUDB_Adapter	component	responsible to create and maintain tables for all persistent classes	subArch I
RUDBregistered	state predicate	the user is registered	sdRegister
RUCmdsPort	port	registered user command port of MRA_Application	subArch II, III, IV
RUCmdsPort_I	port	registered user command port of RegisteredUserGUI	subArch II, III, IV
RegisteredUserPort	port	connects the registered user to the interface RegisteredUserGUI	subArch II, III, IV
S			
SelectWebpageRegisteredUser	state predicate	a list of movies is displayed	sdbrowseAvailableMovi State Machine
showAvailableMovies	phenomenon	machine providing available movies from the database to a webpage	pdBrowse, ProblemFrames, sdBrowse, TCD, subArch IV
showUserComments	phenomenon	the database passes comments written by registered Users	CD
ShowAddingConfirmed	state	indicates that adding a movie took place successfully	State Machine
ShowAddingFailed	state	indicates that adding a movie failed	State Machine
ShowRatingConfirmed	state	indicates that rating a movie took place successfully	State Machine
ShowRatingFailed	state	indicates that rating a movie failed	State Machine
sortMovieDatabase	phenomenon	arrange movies by descending order of rating	CD
sql		abbreviation for Structured Query Language	TCD
SQLDatabase	casual Domain	a SQL database to concretize RegisteredUserDataBase and MovieDatabase	TCD
T			
TimeData	class, dataType	DataType with the information about the date	CM, OCL
U			
U		abbreviation for User	CD, PD, ProblemFrames
User	biddable Domain	unregistered user	CD, PD, TCD
UserGUI	component	web interface for users	subArch I
UserExists	state predicate	the user is registered and exists in the database	sdRegister

Table 4.1: Glossary

Name	Type	Description	Source
username	attribute	represents unique username of a user	CM
UCmdsPort	port	user command port of MRA_Application	subArch I
UCmdsPort_I	port	user command port of UserGUI	subArch I
UserPort	port	connects the user to the interface UserGUI	subArch I
V			
validateRating	phenomenon	counterpart to rateMovie	CD, pdRateMovie, ProblemFrames, sdRateMovie, subArch III
W			
WebBrowser	connection Domain	Web browser used by user, registered user, e.g. Chrome	TCD
WebpageRegisteredUser	connection Domain	domain introduced to connect RegisteredUser and the machine	PD, TCD, CM
WebpageUser	connection Domain	domain introduced to connect User and the machine	pdRegister, sdRegister, TCD, CM
WRU		abbreviation for WebpageRegisteredUser	PD, ProblemFrames, TCD, CM
WU		abbreviation for WebpageUser	pdRegister, ProblemFrames, TCD, CM
X			
Y			
year	attribute	stores the year numerically	CM, OCL
Z			