Meta Resource Management System

Analysis Model

René Freude (707168) <rene.freude@gmx.de>
Daniel Sadilek (707297) <sadilek@tfh-berlin.de>
Stephan Weiß (706830) <steph.weiss@web.de>

2003-06-23

Revision History

Revision 0.1	2003-05-19	Initial public release.
Revision 0.2	2003-06-09	Corrected some cardinalities, extended descriptions, added operations.
Revision 0.3	2003-06-15	Added "user logs in" sequence diagram.
Revision 0.4	2003-06-23	Extended form static model to analysis model.

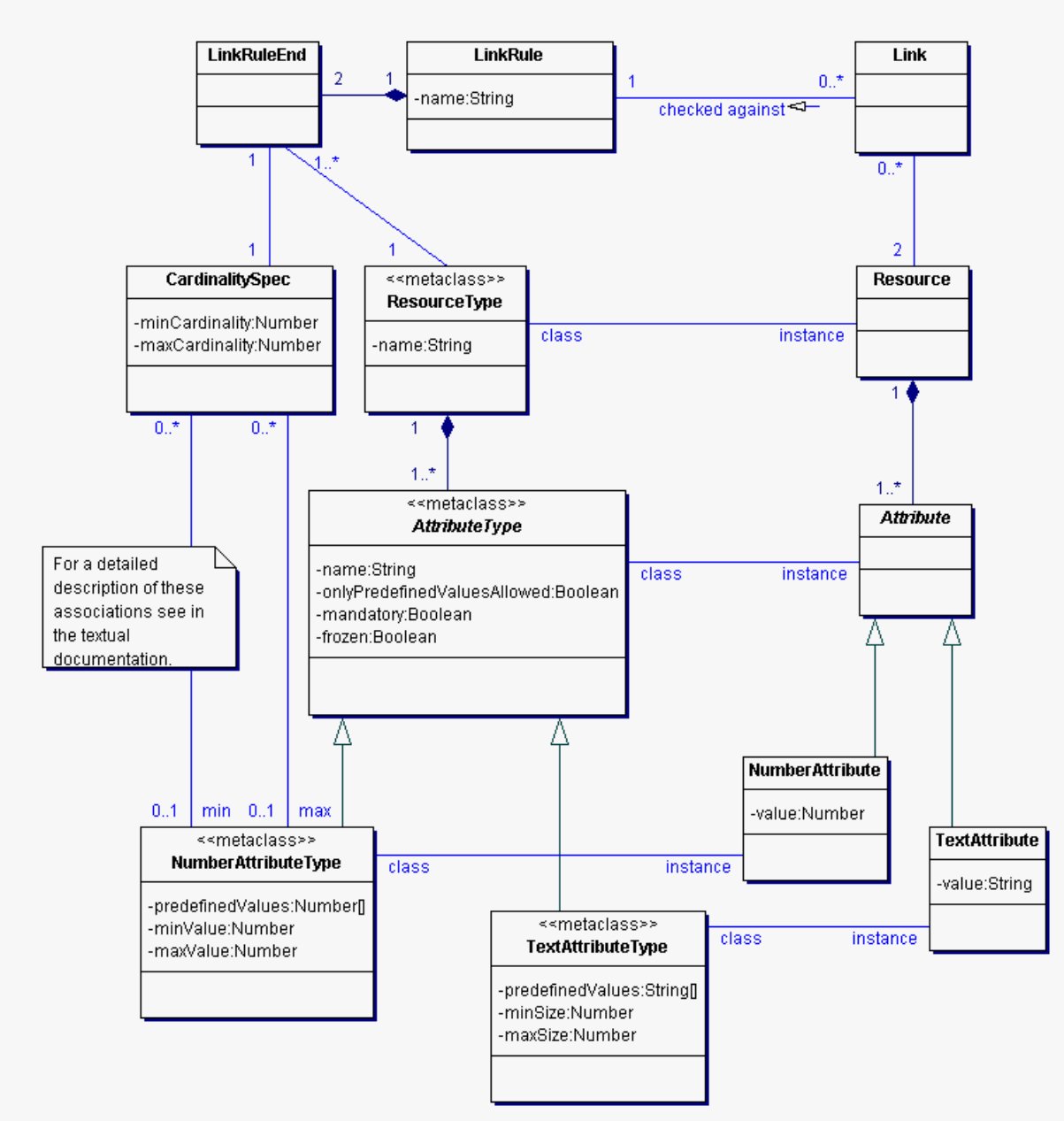
This document contains the class diagrams and class descriptions that resulted from the static analysis as well as sequence diagrams and state chart diagrams that we used to verify the class model.

Table of Contents

1. Data and Meta Data Management Classes

The following diagram depicts the data and meta data structures of the MRMS.

Figure 1. Data and Meta Data Classes



1.1. Resource

Description A *Resource* is composed of its *Attributes* and is an instance of a *ResourceType* which

	specifies which *Attributes* the *Resource* may have.
Attributes	---
Operations	---

1.2. ResourceType

Description	A *ResourceType* specifies (by composition) the *Attributes* that a *Resource* of this type has.
Attributes	*name* (String): the name of the *ResourceType*
Operations	---

1.3. AttributeType

Description	Abstract base class for attribute types that a *ResourceType* is composed of.
Attributes	*name* (String): the name of the *AttributeType* *onlyPredefinedValuesAllowed* (Boolean): if true, the user may only select the predefined values for an *Attribute* that has this type; if false, he may enter another value as well *mandatory* (Boolean): if true, the user must enter a value for *Attributes* of this type *frozen* (Boolean): if true, the user may not change the value of *Attributes* of this type
Operations	---

1.4. NumberAttributeType

Description	Concrete *AttributeType* for *NumericalAttributes*.
Attributes	*predefinedValues* (Number[]): an array specifying predefined values for *Attributes* of this type *minValue* (Number): the minimum value *Attributes* of this type may have *maxValue* (Number): the maximum value *Attributes* of this type may have
Operations	---

1.5. TextAttributeType

Description	Concrete *AttributeType* for *TextAttributes*.
Attributes	*predefinedValues* (String): an array specifying predefined values for *Attributes* of this type *minSize* (Number): the minimum number of characters *Attributes* of this type may have

	maxSize (Number): the maximum number of characters *Attributes* of this type may have
Operations	---

1.6. Attribute

Description	Abstract base class for attributes that a *Resource* is composed of.
Attributes	---
Operations	---

1.7. NumberAttribute

Description	Concrete *Attribute* for a numerical property characterisation of a *Resource*.
Attributes	*value* (Number): numerical property characterisation of a *Resource*
Operations	---

1.8. TextAttribute

Description	Concrete *Attribute* for a textual property characterisation of a *Resource*.
Attributes	*value* (Number): textual property characterisation of a *Resource*
Operations	---

1.9. Link

Description	A *Link* references the two *Resources* that are linked together by it. Its consistency is checked against the *LinkRule* it is referenced with.
Attributes	---
Operations	---

1.10. LinkRule

Description	A *LinkRule* defines characteristics of a consistent *Link*. It aggregates two *LinkRuleEnds*.
Attributes	*name* (String): name of the *LinkRule*.
Operations	---

1.11. LinkRuleEnd

Description	A *LinkRuleEnd* belongs to a *LinkRule*. One *LinkRuleEnd* of a *LinkRule* specifies that *Resources* of the referenced *ResourceType* may be linked (to *Resources* of the *ResourceType* referenced by the other *LinkRuleEnd*) according to the referenced *CardinalitySpec*.
Attributes	---
Operations	---

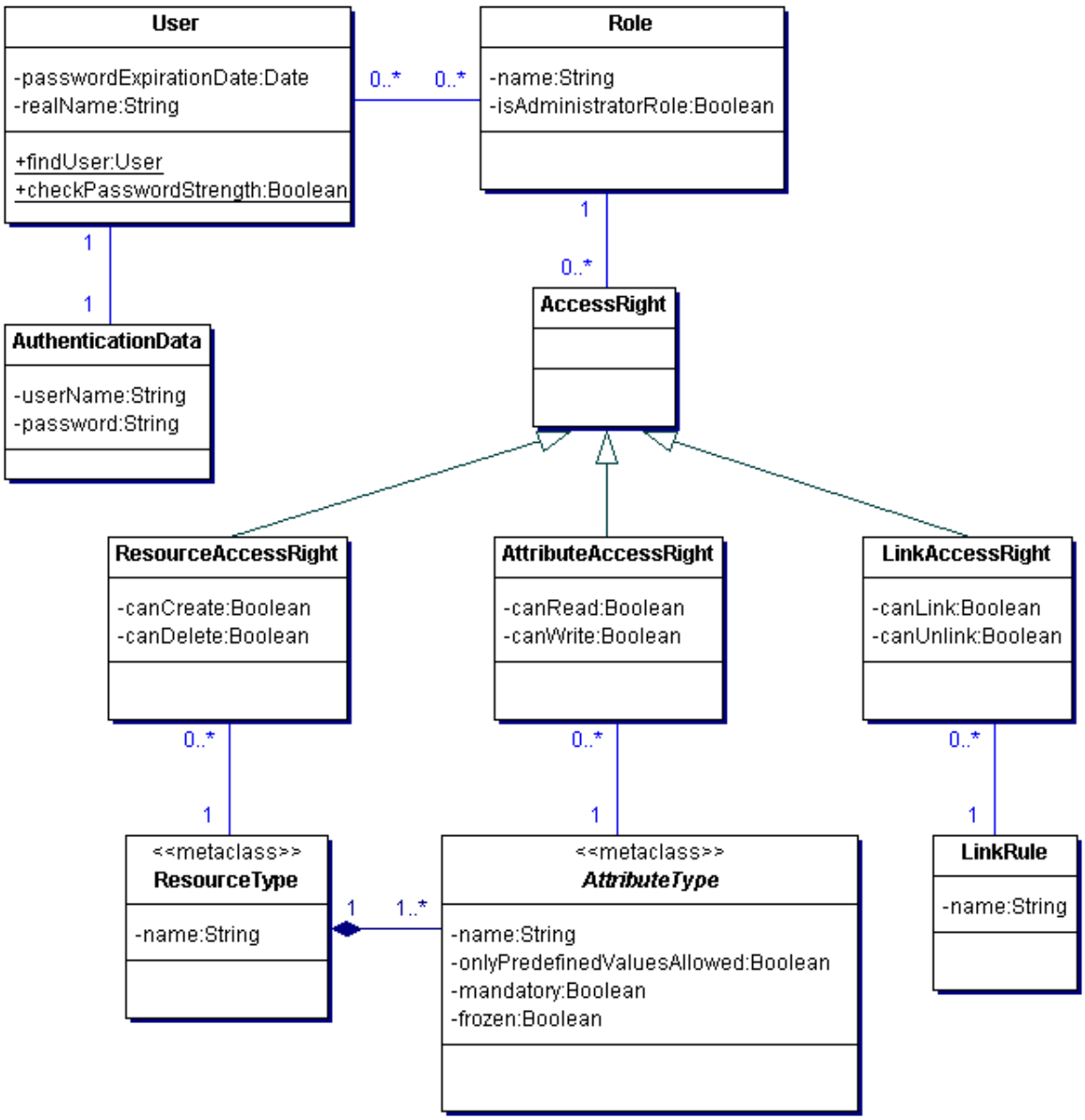
1.12. CardinalitySpec

Description	Specifies the minimum and maximum cardinality for a link rule end of a link rule. Example: A *LinkRule* has two ends *LinkRuleEnd* l1 and *LinkRuleEnd* l2. The *LinkRuleEnd* l1 references the *ResourceType* t1 and *CardinalitySpec* c1 while l2 references t2 and c2. If c1 is min=1 and max=4, this means that one specific *Resource* of type t2 must have at least 1 and may have up to 4 *Links* to *Resources* of type t1. The *CardinalitySpec* c1 may reference a *NumberAttributeType* of the *ResourceType* t2.
Attributes	*minCardinality* (Number): value for the minimum cardinality; will be ignored when there is a “min”-reference to a *NumberAttributeType*, in this case the *NumberAttribute*'s value will be used instead *maxCardinality* (Number): value for the maximum cardinality; will be ignored when there is a “max”-reference to a *NumberAttributeType*, in this case the *NumberAttribute*'s value will be used instead
Operations	---

2. User Management Classes

The following diagram depicts the classes for user management of the MRMS.

Figure 2. User Management Classes



2.1. AuthenticationData

Description	Value class, encapsulating the authentication data of a user.
Attributes	*userName* (String): the user's name
	password (String): the user's password
Operations	---

2.2. User

Description	Class for user accounts of the MRMS. Its instances may play *Roles* in the system.
Attributes	*passwordExpirationDate* (Date): date after which the user has to enter a new

password

realName (String): real name of the user

Operations

- static checkPasswordStrength(password: String): Boolean

Effect	Checks, if the given password String is strong enough (minimum length, mixed letters and numbers, ...) to be accepted by the system.
Parameters	*password*: the password to be checked
Return	The boolean value *true*, iff the password is strong enough.
Exceptions	---
Actor	Control class of the use case “User changes password”.

- static findUser(authData: AuthenticationData): User

Effect	Searches the system for a *User* matching the given *AuthenticationData*.
Parameters	*authData*: the *AuthenticationData* to search for
Return	If a matching *User* object could be found it is returned, otherwise the operation returns the *null* pointer.
Exceptions	---
Actor	Control class of the use case “User logs in”.

2.3. Role

Description	A *Role* defines which *AccessRights* its players (*Users*) have.
Attributes	*name* (String): name of the *Role* *isAdministratorRole* (Boolean): defines if *Users* of the *Role* have administration rights
Operations	---

2.4. AccessRight

Description	Abstract base class for access rights. If a *Role* references an *AccessRight* it has this *AccessRight*. *Users* have the *AccessRights* which the *Roles* they play have.
Attributes	---

Operations	---

2.5. ResourceAccessRight

Description	Concrete *AccessRight* that defines owner's authority of working with *Resources* that are of a specific *ResourceType*.
Attributes	*canCreate* (Boolean): defines if *Resource*s of the referenced *ResourceType* may be created
	canDelete (Boolena): defines if *Resource*s of the referenced *ResourceType* may be deleted
Operations	---

2.6. AttributeAccessRight

Description	Concrete *AccessRight* that defines owner's authority of working with *Atrributes* of a specific *AttributeType* that belongs to a specific *ResourceType*.
Attributes	*canRead* (Boolean): defines if *Attributes* of the referenced *AttributeType* may be read
	canWrite (Boolean): defines if *Attributes* of the referenced *AttributeType* may be written
Operations	---

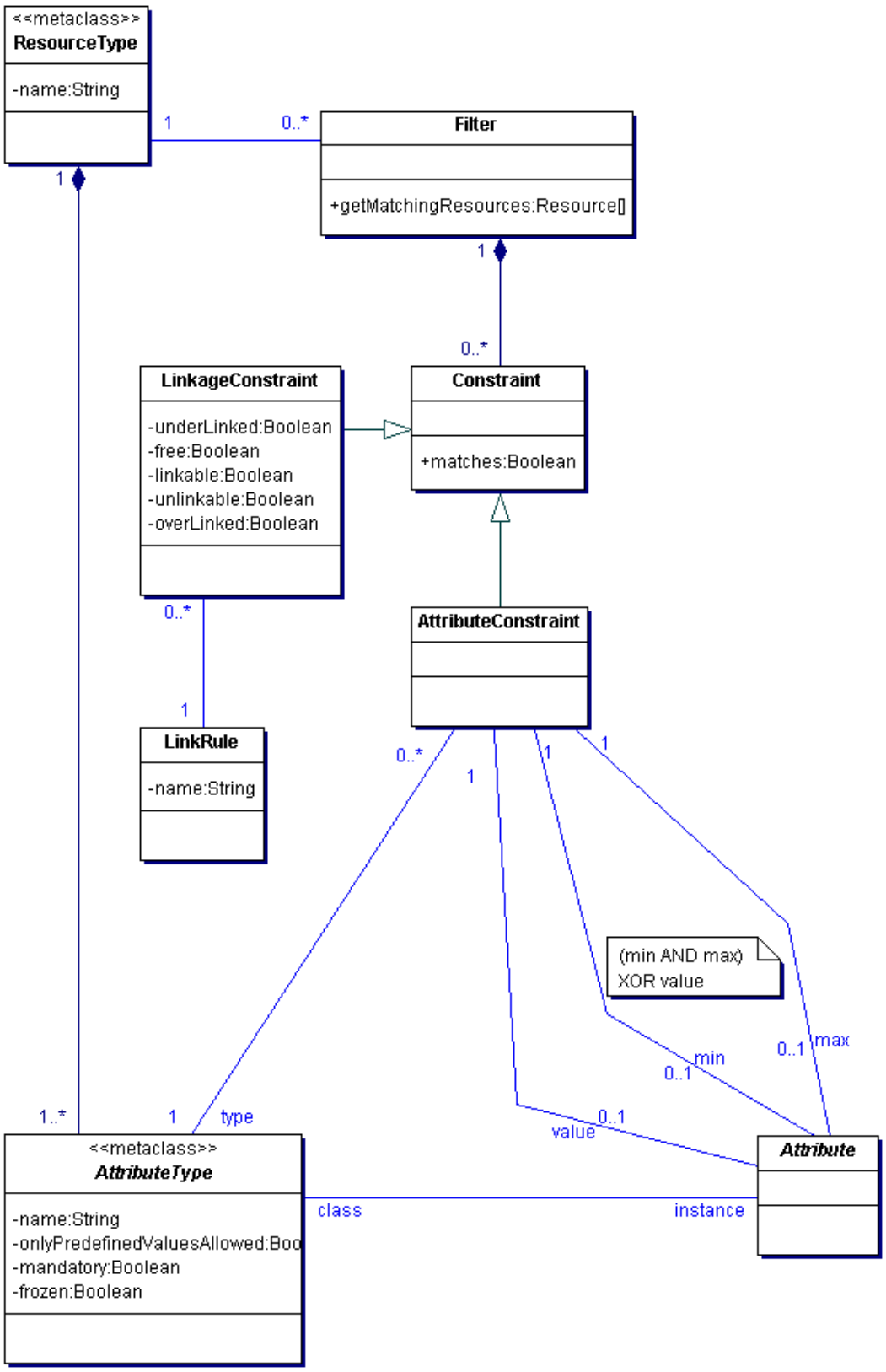
2.7. LinkAccessRight

Description	Concrete *AccessRight* that defines owner's authority of creating and deleting *Links* according to a specific *LinkRule*.
Attributes	*canLink* (Boolean): defines if *Links* according to the referenced *LinkRule* may be created
	canUnlink (Boolean): defines if *Links* according to the referenced *LinkRule* may be deleted
Operations	---

3. Filter Classes

The following diagram depicts the classes needed to create a filtered collection of *Resources*.

Figure 3. Filter Classes



3.1. Filter

Description A *Filter* is used to get a subset of all *Resources* of the referenced *ResourceType*. The

Filter is defined by the *Constraints* it is composed of.

Attributes ---

Operations

- getMatchingResources(): Resource[]

Effect	Searches the system for *Resources* matching the referenced *Constraints*.
Parameters	---
Return	An array of the matching *Resources*.
Exceptions	---
Actor	Control class of the use case “Create filtered collection of resource entries”.

3.2. Constraint

Description Abstract base class for constraints. *Constraints* are used by a *Filter* to describe a specific state that *Resource* must fulfill to pass.

Attributes ---

Operations

- matches(resource: Resource): Boolean

Effect	Tests, if the given *Resource* matches this *Constraint*.
Parameters	*resource*: the *Resource* to be tested
Return	The boolean value *true*, iff the given *Resource* matches this *Constraint*.
Exceptions	---
Actor	Class *Filter*.

3.3. AttributeConstraint

Description An *AttributeConstraint* is a concrete *Constraint* that checks whether an *Attribute* of the referenced *AttributeType* is either equal to the referenced *Attribute* or lays between the two referenced min- and max-*Attributes*.

Attributes ---

Operations ---

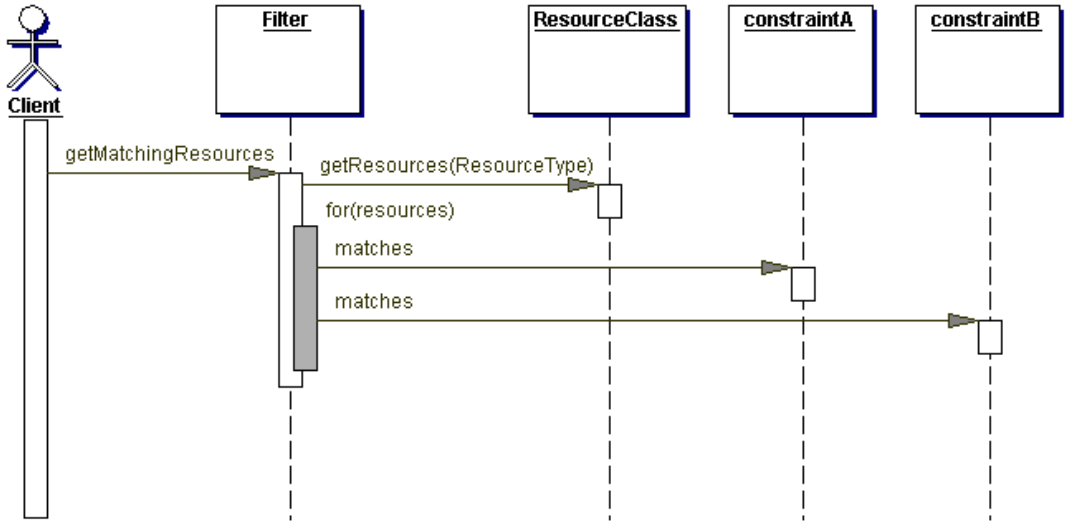
3.4. LinkageConstraint

Description	A *LinkageConstraint* is a concrete *Constraint* that checks whether a *Resource* matches the linkage state that is described by the following attributes. A LinkageConstraint references a LinkRule it refers to: the Link Rule defines a minimum and a maximum cardinality (min and max); the Resource has a current cardinality (cur).
Attributes	*underLinked* (Boolean): cur < min *free* (Boolean): cur = 0 *linkable* (Boolean): cur < max *unlinkable* (Boolean): cur >= max *overLinked* (Boolean): cur > max
Operations	---

3.5. Sequence: Filter.getMatchingResources

The following diagram shows how a filter determines the matching resources.

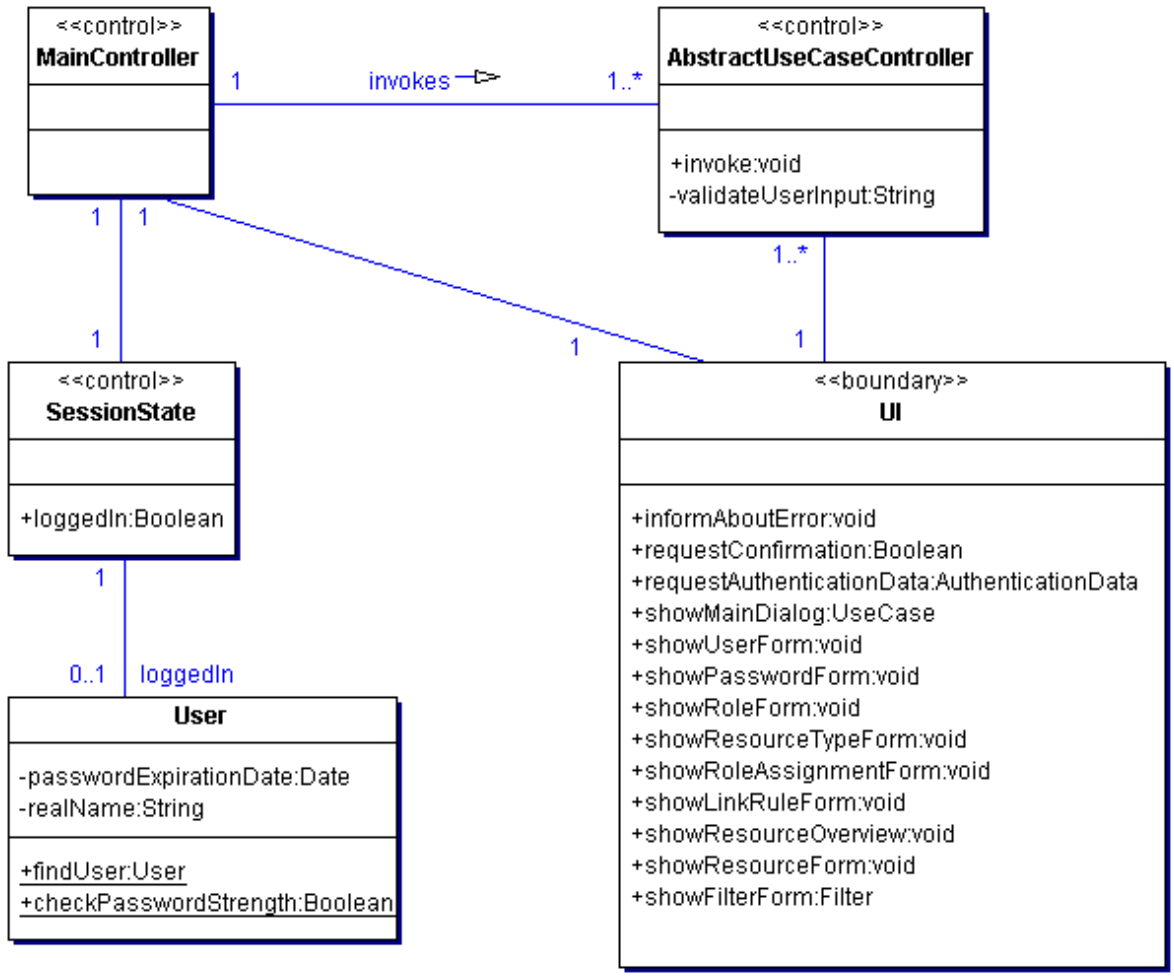
Figure 4. Sequence: Filter.getMatchingResources



4. Control and Boundary Classes

The following diagram depicts the main classes needed to realize an interaction between the user and the MRMS. All use cases have a similar structure, so we have only included the AbstractUseCaseController that serves as a base class for all concrete use case controllers (which we have omitted in this analysis model).

Figure 5. Control and Boundary Classes



4.1. SessionState

Description — A *SessionState* describes a session of interaction between the MRMS and a user. A *User* is logged in in a *SessionState* if it references that *User*.

Attributes — ---

Operations

- loggedIn(user: User): Boolean

  Effect — Checks whether the given *User* is logged in in this *SessionState*.

  Parameters — ---

  Return — The boolean value *true*, iff the given *User* is logged in in this *SessionState*.

  Exceptions — ---

  Actor — All control classes.

4.2. MainController

Description	This controller requests the *UI* to display the main dialog to get the use case the user wants to perform next. It invokes the concrete use case controllers according to the user's choice.
Attributes	---
Operations	---

4.3. AbstractUseCaseController

Description	Base class for all concrete use case controllers. Encapsulates the common control flow.
Attributes	---
Operations	

- invoke(): void

Effect	Constructor operation that starts this controller and requests the *UI* to show the appropriate form for the concrete use case.
Parameters	---
Exceptions	---
Actor	*MainController*.

- private validateUserInput(input: Object): String

Effect	Validates the input the user made.
Parameters	*input*: The input the user has made.
Return	If not valid, a String describing the input errors; otherwise the *null* pointer.
Exceptions	---
Actor	this

The following diagrams show the control flow of an invocation of a concrete use case controller and the states it passes through.

Figure 6. Sequence: Concrete Use Case Controller

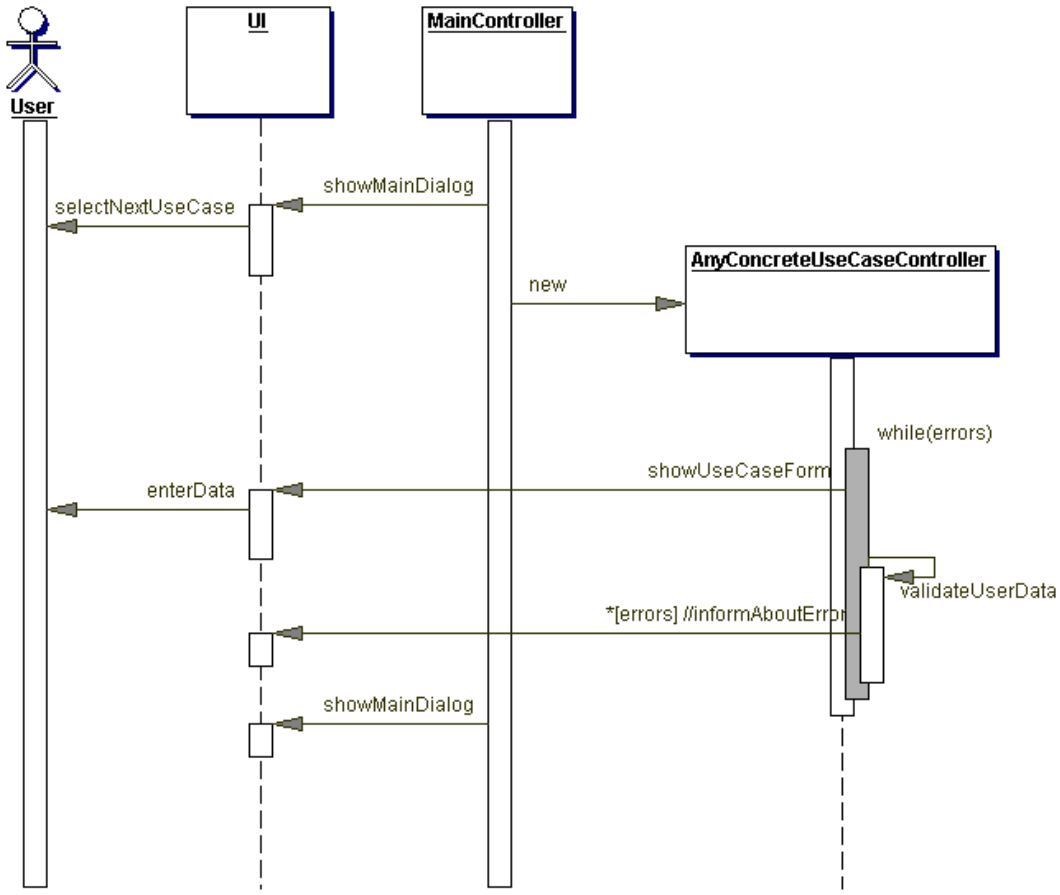
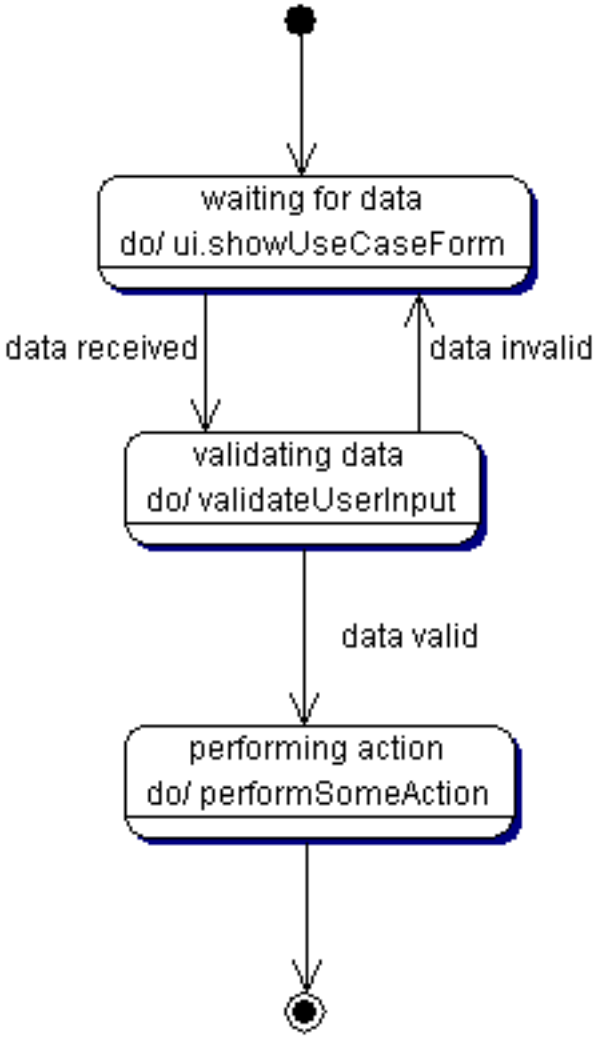


Figure 7. State Chart: Concrete Use Case Controller



4.4. UI

Description	The UI is responsible for providing all facilities the MRMS needs to interact with a user.
Attributes	---
Operations	

- informAboutError(text: String): void

Effect	The user is informed about an error that occured in the system.
Parameters	*text*: the description of the error that occured
Return	---
Exceptions	---
Actor	Any control class.

- requestConfirmation(text: String): Boolean

Effect	The user is requested to confirm or cancel an action.
Parameters	*text*: the question to be answered by the user
Return	The boolean value *true*, iff the user confirmed
Exceptions	---
Actor	Any control class.

- requestAuthenticationData(): AuthenticationData

Effect	Requests authentication data (username and password) from the user.
Parameters	---
Return	An *AuthenticationData* object holding the values for username and password provided by the user.
Exceptions	AuthenticationAbortedException if the user cancelled the operation
Actor	Control class of the use case “User logs in”.

- showMainDialog(): void

Effect	The user is shown the main dialog for interaction with the system. There he selects which use case he wants to perform next.
Parameters	---
Return	The use case that the user wants to perform next.
Exceptions	---
Actor	*MainController*.

- showUserForm(io_user: User): void

Effect	The user is shown a form for editing a *User*'s attributes.
Parameters	*io_user*: the *User* to be edited
Return	---
Exceptions	---
Actor	Control class of the use case “Create user account”.

- showPasswordForm(io_password: String): void

Effect	The user is shown a form for editing a *User*'s password.
Parameters	*io_password*: the password String to be edited
Return	---
Exceptions	---
Actor	Control class of the use case “User changes password”.

- showRoleForm(io_role: Role): void

Effect	The user is shown a form for editing a *Role*'s name.
Parameters	*io_role*: the *Role* to be edited
Return	---
Exceptions and	---
Actor	Control class of the use case “Create role”.

- showRoleAssignmentForm(io_user: User): void

Effect	The user is shown a form for editing a *User*'s role assignment.
Parameters	*io_user*: the *User* whose role assignment should be edited
Return	---
Exceptions	---
Actor	Control class of the use case “Edit role assignment of user account”.

- showRoleAccessRightsForm(io_role: Role): void

Effect	The user is shown a form for editing a *Role*'s access rights.
Parameters	*io_role*: the *Role* to be edited
Return	---
Exceptions	---
Actor	Control class of the use case “Edit rights for role”.

- showResourceTypeForm(io_resourceType: ResourceType): void

Effect	The user is shown a form for editing a *ResourceType*'s name and the *AttributeTypes* it is composed of.

Parameters	*io_resourceType*: the *ResourceType* to be edited
Return	---
Exceptions	---
Actor	Control class of the use case “Define resource type”.

- showLinkRuleForm(io_linkRule: LinkRule): void

Effect	The user is shown a form for editing a *LinkRule*'s attributes.
Parameters	*io_linkRule*: the *LinkRule* to be edited
Return	---
Exceptions	---
Actor	Control class of the use case “Define link rule”.

- showResourceOverviewForm(resources: Resource[]): void

Effect	The user is shown a list of the given *Resources*.
Parameters	*resources*: the *Resources* to be shown
Return	---
Exceptions	---
Actor	*MainController* and control class of the use case “Create filtered collection of resource entries”.

- showResourceForm(io_resource: Resource): void

Effect	The user is shown a form for editing a *Resource*'s *Attributes*.
Parameters	*io_resource*: the *Resource* to be edited
Return	---
Exceptions	---
Actor	Control classes of the use cases “Create resource” and “Edit resource”.

- showFilterForm(): Filter

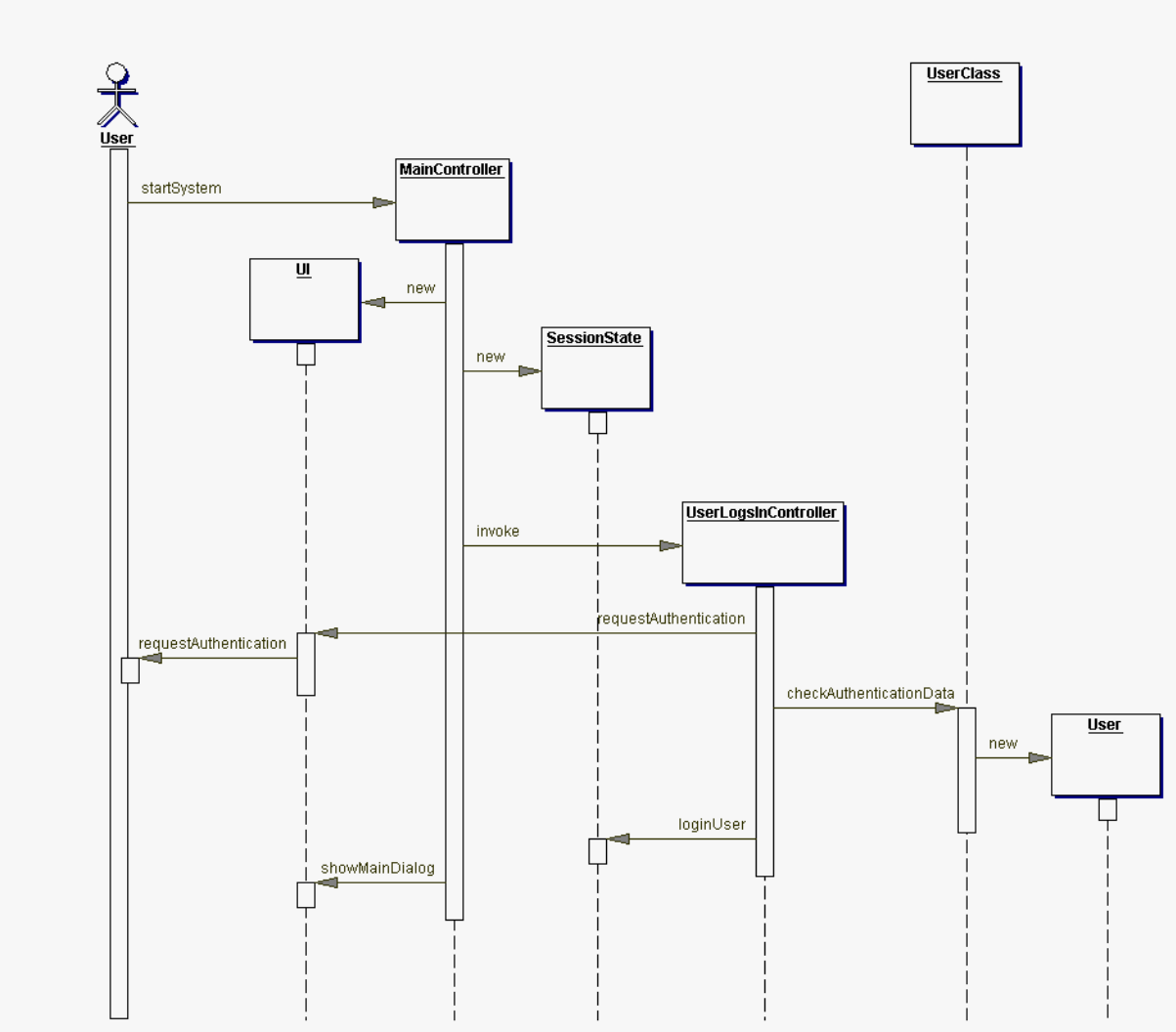
Effect	The user is shown a form for specifying a *Filter*.
Parameters	---
Return	A *Filter* configured according to the users input.

Exceptions	---
Actor	Control class of the use case “Create filtered collection of resource entries”.

4.5. Sequence: User logs in

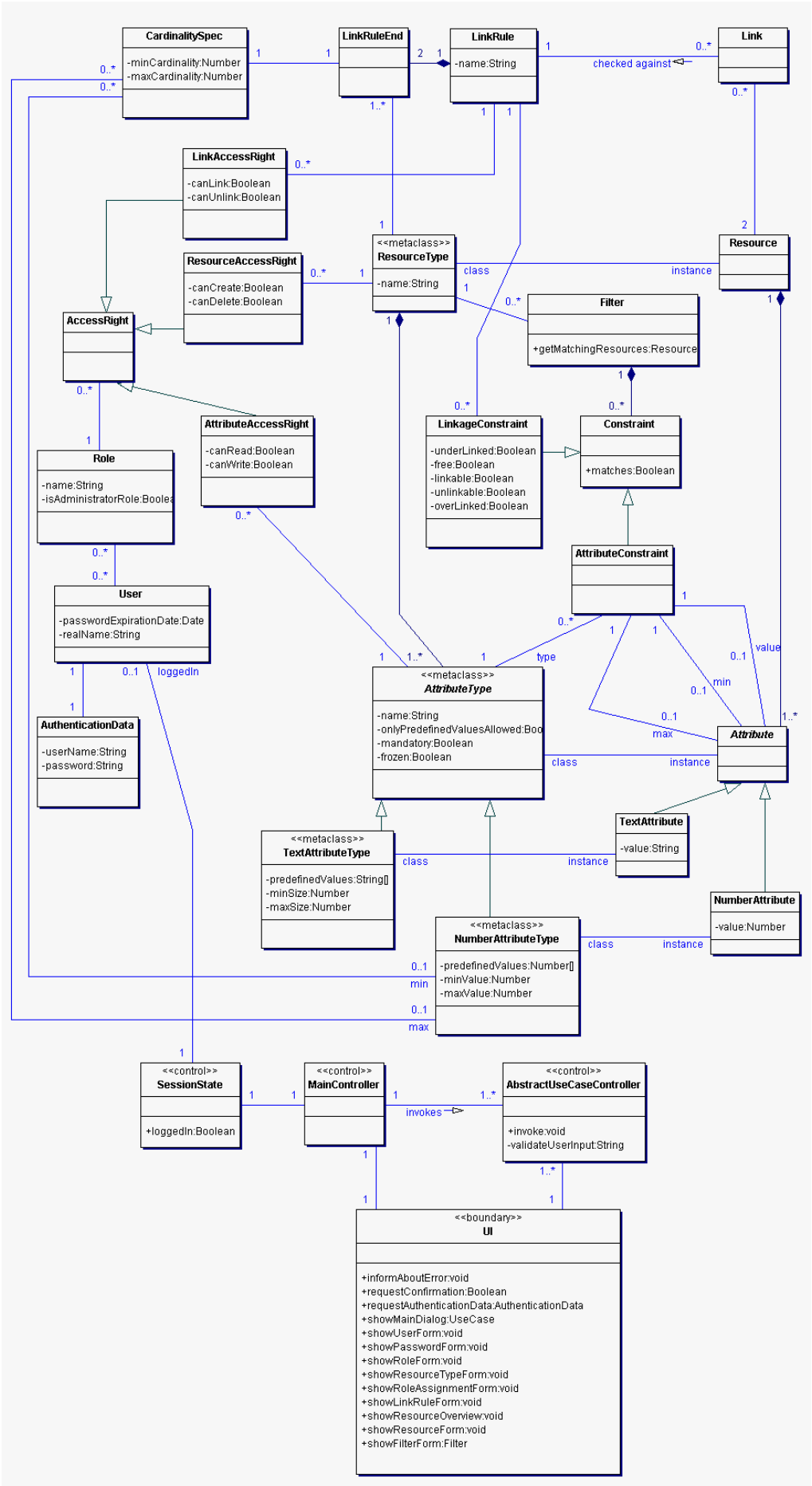
This diagram shows the interactions of a successful log in.

Figure 8. Sequence: User logs in



5. Extensive Overview

Figure 9. All Classes


CardinalitySpec
-minCardinality:Number
-maxCardinality:Number
LinkRuleEnd
LinkRule
-name:String
Link
checked against
LinkAccessRight
-canLink:Boolean
-canUnlink:Boolean
<<metaclass>>
ResourceType
-name:String
Resource
class
instance
ResourceAccessRight
-canCreate:Boolean
-canDelete:Boolean
AccessRight
Filter
+getMatchingResources:Resource
AttributeAccessRight
-canRead:Boolean
-canWrite:Boolean
LinkageConstraint
-underLinked:Boolean
-free:Boolean
-linkable:Boolean
-unlinkable:Boolean
-overLinked:Boolean
Constraint
+matches:Boolean
Role
-name:String
-isAdministratorRole:Boole
AttributeConstraint
User
-passwordExpirationDate:Date
-realName:String
loggedIn
type
value
min
max
AuthenticationData
-userName:String
-password:String
<<metaclass>>
AttributeType
-name:String
-onlyPredefinedValuesAllowed:Boo
-mandatory:Boolean
-frozen:Boolean
Attribute
TextAttribute
-value:String
<<metaclass>>
TextAttributeType
-predefinedValues:String[]
-minSize:Number
-maxSize:Number
NumberAttribute
-value:Number
<<metaclass>>
NumberAttributeType
-predefinedValues:Number[]
-minValue:Number
-maxValue:Number
<<control>>
SessionState
+loggedIn:Boolean
<<control>>
MainController
<<control>>
AbstractUseCaseController
invokes
+invoke:void
-validateUserInput:String
<<boundary>>
UI
+informAboutError:void
+requestConfirmation:Boolean
+requestAuthenticationData:AuthenticationData
+showMainDialog:UseCase
+showUserForm:void
+showPasswordForm:void
+showRoleForm:void
+showResourceTypeForm:void
+showRoleAssignmentForm:void
+showLinkRuleForm:void
+showResourceOverview:void
+showResourceForm:void
+showFilterForm:Filter