Meta Resource Management System

Design Model

René Freude (707168) <rene.freude@web.de>
Henry Hinze (681566) <henry@lilit.net>
Daniel Sadilek (707297) <sadilek@tfh-berlin.de>
Stephan Weiß (706830) <steph.weiss@web.de>

Copyright © 2003

2003-10-20

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 from static model to analysis model.
Revision 0.5	2003-10-06	Incorporated optional feature "Resource Reservation" (see appendix of this document for the use cases derived from that feature); refined package structure; introduced distinction between physical resource containment hierarchy and resource usage.
Revision 0.6	2003-10-20	Extended from analysis model to design model

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

Table of Contents

1. Overview

So far, the design model only covers a subset of all use cases - completely defined in the document “Use Cases”. The remaining use cases will be considered during the next revisions of this document. The use cases covered so far are:

- User logs in
- Create filtered collection of resource entries
- Edit resources

This document is organized along the package structure of the MRMS. Every package describes one aspect of the system:

- *model.entity*: The MRMS can handle resources and the employees; the attributes that are to be saved for each resource type and employee type can be configured by an administrator. This common functionality is pulled up to the super type Entity. The package model.entity contains the classes to handle entities (resources, employees) and their attributes (number, text, boolean).
- *model.linkage*: Resources and employees do not exist detached. Resources can be organized in a physical

containment structure (e.g. a room contains workplaces, workplaces contain a computer, and so on) and resources can be used by employees. The package model.linkage contains the classes that are necessary to represent these links.

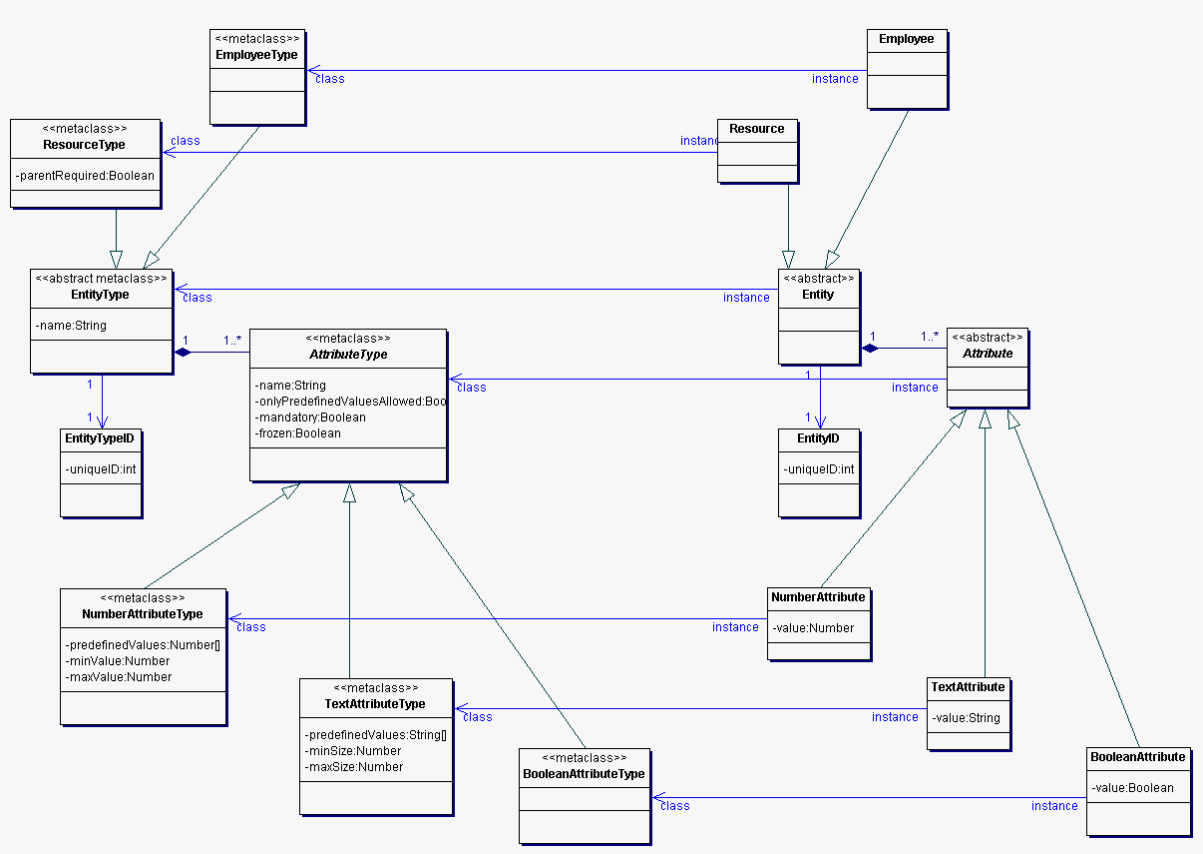
- *model.user*: The users of the system need different access rights according to the role they play in the business. The package model.user contains the classes that represent the rights users have to create and delete entities, edit their attributes and create links.
- *model.filter*: Creating a filtered collection of resources is a complex function that is required in different use cases. The package model.filter contains the classes needed to configure a filter with constraints and execute it.
- *client*: Classes needed to realize an interaction between the user and the MRMS.
- *server*: Classes for the MRMS server.

(The classes imported from others packages are colored yellow.)

2. Package: model.entity

The following diagram depicts the classes to handle entities (resources, employees) and their attributes (number, text, boolean).

Figure 1. Entity Classes



2.1. EntityType

Description An *EntityType* has a name and specifies (by composition) the *Attributes* that an *Entity* of this type has, it references a unique *EntityTypeID*.

Attributes *name* (String): the name of the *EntityType*

Operations ---

2.2. EntityTypeID

Description An *EntityTypeID* is a unique identifier for an *EntityType*.

Attributes *uniqueID* (int): an instance field making the *EntityTypeID* unique

Operations ---

2.3. ResourceType

Description A *ResourceType* is a specialised *EntityType* for defining *Resources*.

Attributes *parentRequired* (Boolean): specifies whether instances of this *ResourceType* must have a parent Resource

Operations ---

2.4. EmployeeType

Description An *EmployeeType* is a specialised *EntityType* for defining *Employees*.

Attributes ---

Operations ---

2.5. Entity

Description An *Entity* is composed of its *Attributes* and is an instance of an *EntityType* which specifies which *Attributes* the *Entity* may have, it references a unique *EntityID*.

Attributes ---

Operations ---

2.6. EntityID

Description An *EntityID* is a unique identifier for an *Entity*.

Attributes *uniqueID* (int): an instance field making the *EntityID* unique

Operations ---

2.7. Resource

Description A *Resource* is a specialised *Entity* for representing real-life-resources and is an instance of a *ResourceType* which specifies if this *Resource* must have a parent *Resource* within the Resources-Containment-Hierarchy.

Attributes ---

Operations ---

2.8. Employee

Description An *Employee* is a specialised *Entity* for representing users of real-life-resources and is an instance of an *EmployeeType*.

Attributes ---

Operations ---

2.9. AttributeType

Description Abstract base class for attribute types that an *EntityType* 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 ---

2.10. BooleanAttributeType

Description Concrete *AttributeType* for logical property characterisation of an *Entity*.

Attributes *value* (Boolean): logical property characterisation of an *Entity*

Operations ---

2.11. 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 ---

2.12. 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 ---

2.13. Attribute

Description Abstract base class for *Attributes* that an *Entity* is composed of.

Attributes ---

Operations ---

2.14. BooleanAttribute

Description Concrete *Attribute* for a boolean property characterisation of an *Entity*.

Attributes *value* (Boolean): numerical property characterisation of an *Entity*

Operations ---

2.15. NumberAttribute

Description Concrete *Attribute* for a numerical property characterisation of an *Entity*.

Attributes *value* (Number): numerical property characterisation of an *Entity*

Operations ---

2.16. TextAttribute

Description Concrete *Attribute* for a textual property characterisation of an *Entity*.

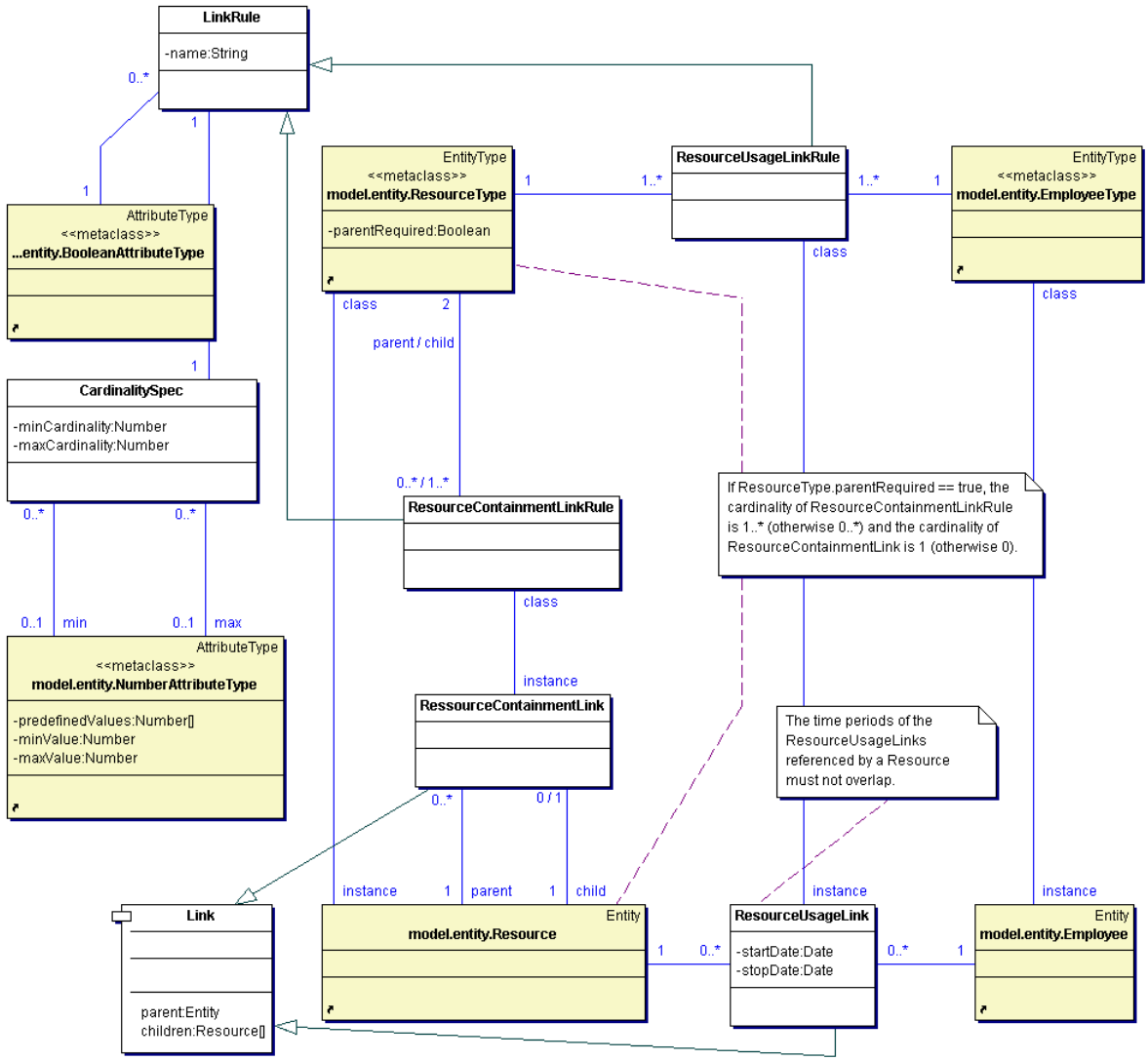
Attributes *value* (String): textual property characterisation of an *Entity*

Operations ---

3. Package: model.linkage

The following diagram depicts the classes that are necessary to represent the physical containment links between resources and resources and the usage links between resources and employees.

Figure 2. Linkage Classes



3.1. LinkRule

Description — A *Link Rule* defines the characteristics of a consistent *Link*. Both the physical containment structure of the resources as well as the usages of the resource by the users can be modelled as links. In both cases the corresponding link rules have the characteristic that the cardinality of one side is 1; for the physical containment links this side is the parent resource type and for the usage links this side is the employee type. The other side of the link rule can have an arbitrary cardinality (i.e. the number of children a parent has in the physical containment structure as well as the number of resources an employee may use is not constrained by the system but can be customized by the administrator); this cardinality is contained in the *CardinalitySpec* referenced by the *LinkRule*. A *LinkRule* can reference an *BooleanAttributeType* of the parent resource / using employee; in this case *Link*s of this *LinkRule* can only be created for those *Resource*s / *Employee*s where the corresponding *BooleanAttribute* is true.

Attributes — *name* (String): name of the *LinkRule*

Operations — ---

3.2. Link

Description Base class for *ResourceContainmentLink* and *ResourceUsageLink*.

Attributes ---

Operations ---

3.3. ResourceContainmentLinkRule

Description A *ResourceContainmentLinkRule* defines the characteristics of a consistent *ResourceContainmentLink*. It references two *ResourceTypes* which may be linked together by a *ResourceContainmentLink.*

Attributes ---

Operations ---

3.4. ResourceContainmentLink

Description A *ResourceContainmentLink* references two *Resources* that are linked together by it; one resource takes the parent role, the other is its child in the pysical containmet. Its consistency is checked against the *ResourceContainment LinkRule* references.

Attributes ---

Operations ---

3.5. ResourceUsageLinkRule

Description A *ResourceUsageLinkRule* defines the characteristics of a consistent *ResourceUsageLink*. It references one *ResourceType* and one *EmployeeType* whose instances may be linked together by a *ResourceUsageLink.*

Attributes ---

Operations ---

3.6. ResourceUsageLink

Description A *ResourceUsageLink* references one *Resource* and one *Employee* that are linked together by it. Its consistency is checked against the *ResourceUsageLinkRule* it references. There may be more than one *ResourceUsageLink* at a *Resource*; but only one of can be active at a certain time.

Attributes *startDate* (Date): Time when usage starts.

stopDate (Date): Time when usage expires.

Operations ---

3.7. CardinalitySpec

Description Specifies the minimum and maximum cardinality for a certain *ResourceType*, referenced by a

ResourceUsageLinkRule or a *ResourceContainmentLinkRule*. Example: A *ResourceContainmentLinkRule* has two ends *ResourceType*1 (parent) and *ResourceType*2 (child). The *ResourceType*1 always has the cardinality 1 while *ResourceType*2 has the cardinality min=1 and max=4, this means that one specific *Resource* of *ResourceType*1 must have at least 1 and may have up to 4 Links to *Resources* of *ResourceType*2. The *CardinalitySpec* may also reference a *NumberAttributeType* of the *ResourceType*1.

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

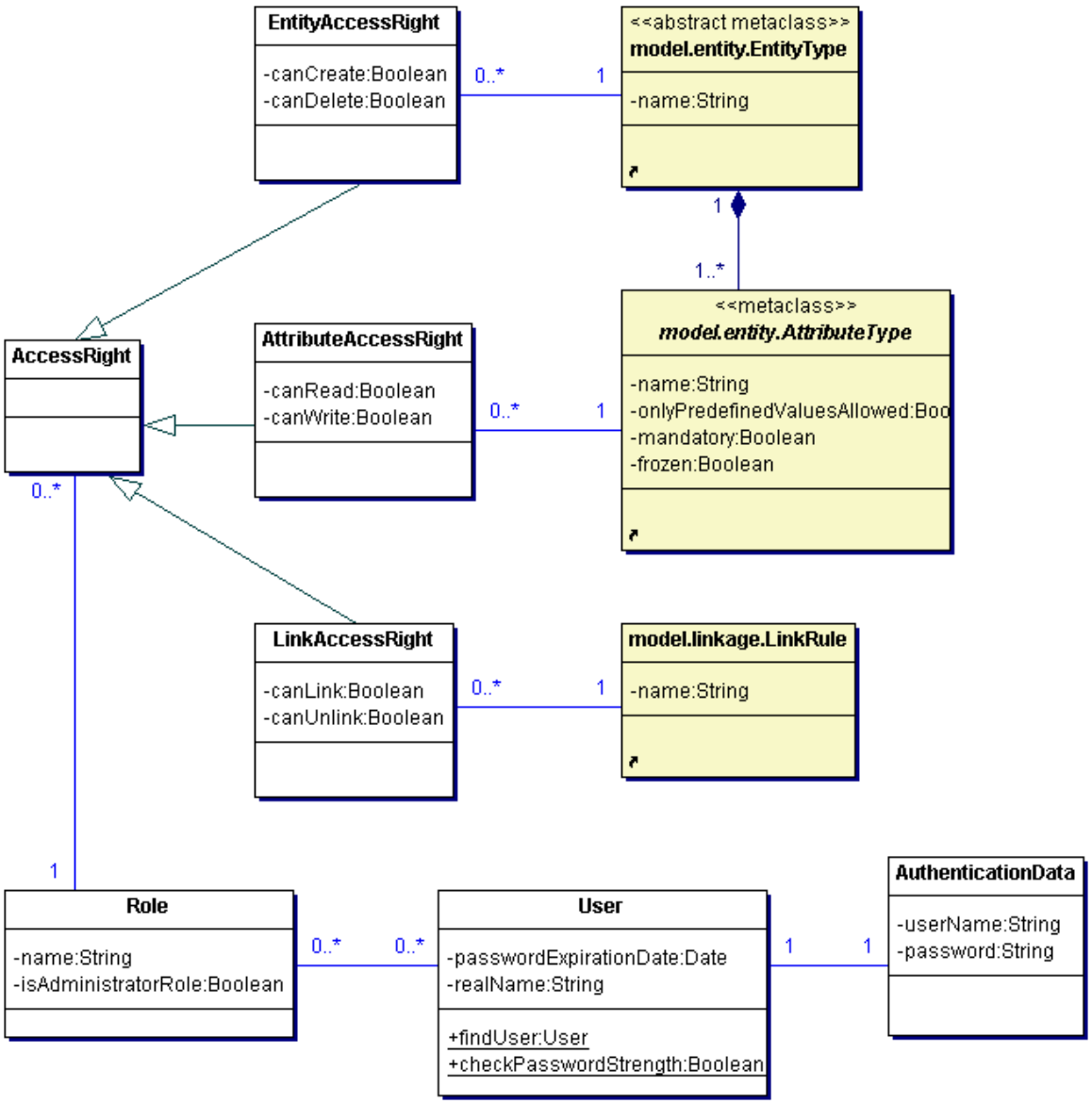
Operations

---

4. Package: model.user

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

Figure 3. User and Access Rights Management Classes



4.1. 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 ---

4.2. 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”.

4.3. 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: ---

4.4. 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: ---

4.5. 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	---

4.6. 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	---

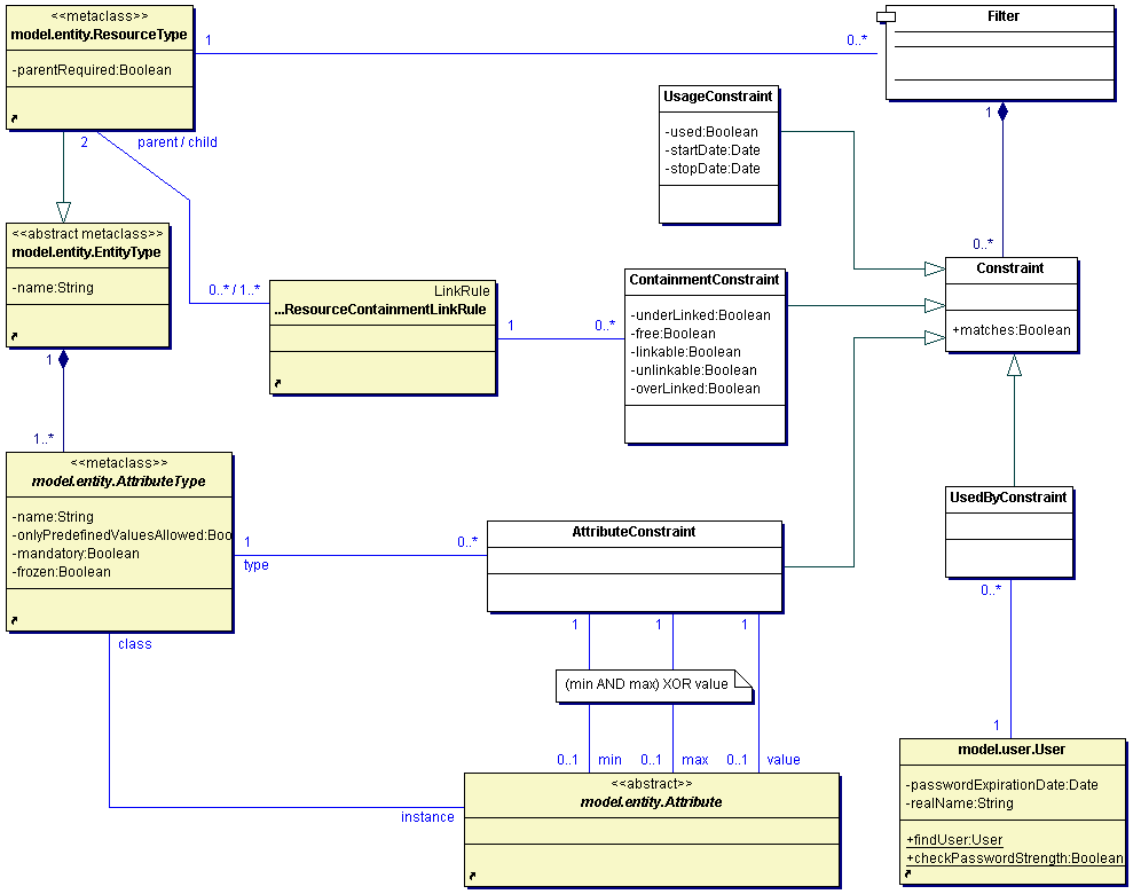
4.7. LinkAccessRight

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	---

5. Package: model.filter

The following diagram depicts the classes needed to configure a filter and get a collection of *Resource*s out of it.

Figure 4. Filter Classes



5.1. 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”.

5.2. 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*.

5.3. 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 ---

5.4. ContainmentConstraint

Description A *ContainmentConstraint* is a concrete *Constraint* that checks whether a *Resource* matches the physical containment state that is described by the following attributes. A *ContainmentConstraint* references the *LinkRule* it refers to. If in this *LinkRule* the *ResourceType* that is to be filtered has (1) the parent role minimum and maximum cardinality are taken from *LinkRule*'s *CardinalitySpec* and refer to the number of children; if it has (2) the client role then min = max = 1 iff the field requiresParent of the *ResourceType* is *true*, min = max = 0 otherwise.

Attributes *underLinked* (Boolean): cur < min

free (Boolean): cur = 0

linkable (Boolean): cur < max

unlinkable (Boolean): cur >= max

overLinked (Boolean): cur > max

Operations ---

5.5. UsageConstraint

Description A *UsageConstraint* is a concrete *Constraint* that checks whether a *Resource* is used or unused in a given time period.

Attributes *used* (Boolean): Defines whether the filtered *Resources* have to be used or unused in the given time period.

startDate (Date): Start time of the time time period.

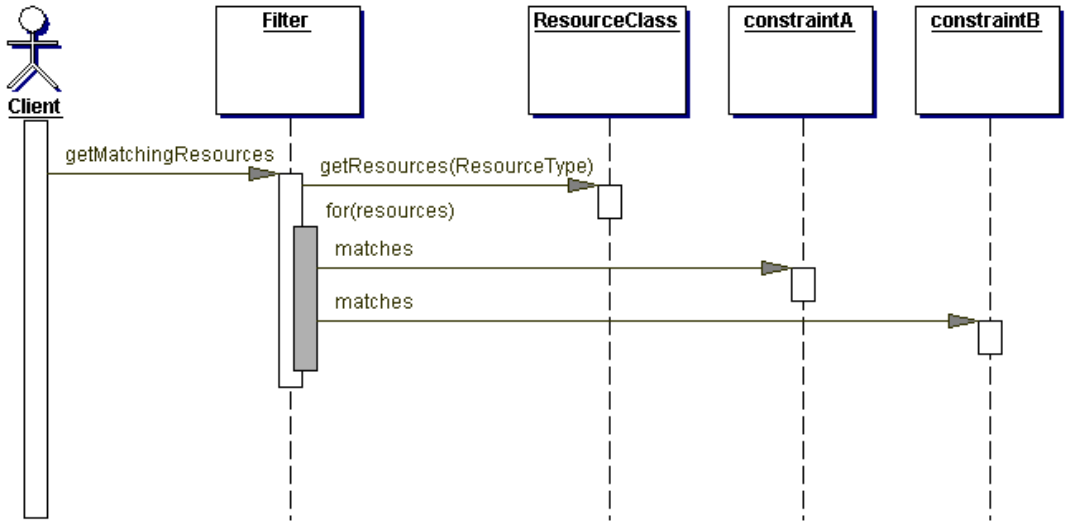
stopDate (Date): End time of the time period.

Operations ---

5.6. Sequence: Filter.getMatchingResources

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

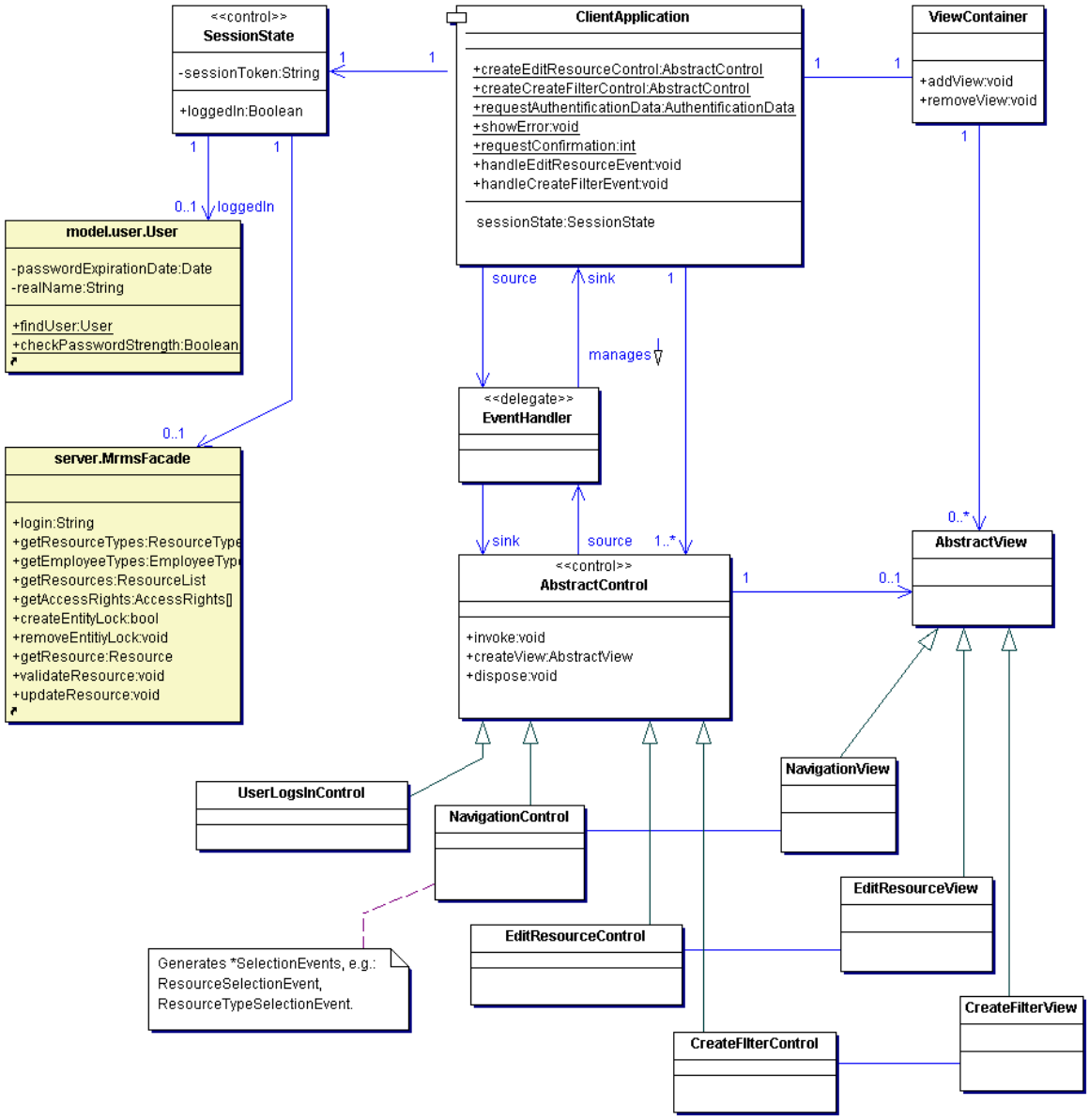
Figure 5. Sequence: Filter.getMatchingResources



6. Package: client

The following diagram depicts the main classes needed to realize an interaction between the user and the MRMS.

Figure 6. Control and Boundary Classes



6.1. 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*. It logged in it has a remote reference to an instance of *MrmsFacade* on the server which can be used by the control to communicate with the server.

Attributes — ---

Operations

- loggedIn(user: User): Boolean

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

  Parameters — ---

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

Exceptions ---

Actor *ClientApplication*

6.2. ClientApplication

Description The *ClientApplication* manages concrete *AbstractControls*. It provides a *ViewContainer* were *AbstractViews* of *AbstractControls* may be plugged in. It is associated with a *SessionState* that provides a reference to the suitable MRMS server facade. Managed *AbstractControls* may interact with the *ClientApplication* by using Events. Therefore the *ClientApplication* provides operations that may be registered at the appropriate *EventHandle*. Moreover it contains static helper operations for showing dialogs to the user (used by *AbstractControls*). The *ClientApplication* implements the *Mediator* pattern as described by the GoF.

Attributes ---

Operations

- handleEditResourceEvent(sender: AbstractControl, args: System.Args): void

  Effect An *EditResourceControl* for the selected *Resource* is created and invoked (uses *static createEditResourceControl()*) .

  Parameters *sender*: The Sender of the event that invoked this operation

  args: Arguments of the event that invoked this operation

  Exceptions ---

  Actor *AbstractControls* using an *EventHandle*

- handleCreateFilterEvent(sender: AbstractControl, args: System.Args): void

  Effect A *CreateFilterControl* is created and invoked (uses *static createCreateFilterControl()*) .

  Parameters *sender*: The Sender of the event that invoked this operation

  args: Arguments of the event that invoked this operation

  Return ---

  Exceptions ---

  Actor *AbstractControls* using an *EventHandle*

- static createEditResourceControl(entityID: EntityID): CreateFilterControl

  Effect Factory method for creating an *EditResourceControl*

  Parameters *entityID*: The *EntityID* of the *Resource* to be eddited

Return	---
Exceptions	---
Actor	this

- static createCreateFilterControl(resourceType: ResourceType): void

Effect	Factory method for creating a *CreateFilterControl*
Parameters	*resourceType*: The *ResourceType* the *CreateFilterControl* is created for
Return	---
Exceptions	---
Actor	this

- static requestAuthenticationData(): AuthenticationData

Effect	Requests authentication data (user name and password) from the user.
Parameters---	---
Return	An *AuthenticationData* object holding the values for user name and password provided by the user.
Exceptions	AuthenticationAbortedException if the user cancelled the operation
Actor	*UserLogsInControl*

- static showError(text: String): void

Effect	An error pop up is shown to the user.
Parameters	*text*: Error message
Return	---
Exceptions	---
Actor	*AbstractControls*

- static requestConfirmation(text: String): int

Effect	A confirmation dialog is shown to the user.
Parameters	*text*: Confirmation message
Return	An int value that is representing the decision of the user
Exceptions	---
Actor	*AbstractControls*

6.3. AbstractControl

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

Attributes ---

Operations

- invoke(): void

  Effect Constructor operation that activates this AbstractControl instance.

  Parameters ---

  Exceptions ---

  Actor *ClientApplication*.

- createView(): AbstractView

  Effect The AbstractControl is told to create its view component.

  Parameters ---

  Return The created view component

  Exceptions ---

  Actor *ClientApplication*

- dispose(): void

  Effect Destroys the AbstractControl and its view component.

  Parameters ---

  Return ---

  Exceptions ---

  Actor *ClientApplication*

The following diagrams show the activation and invocation of a concrete use case control.

Figure 7. Sequence: EditResourceControl activation

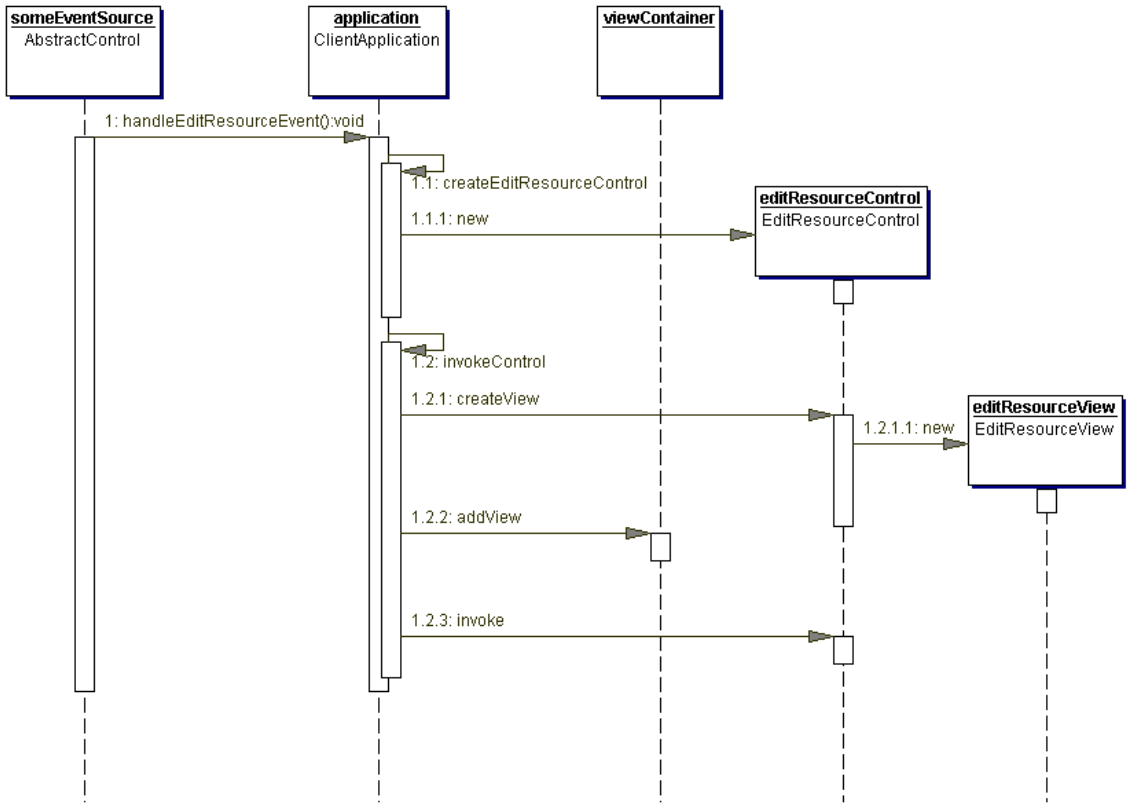
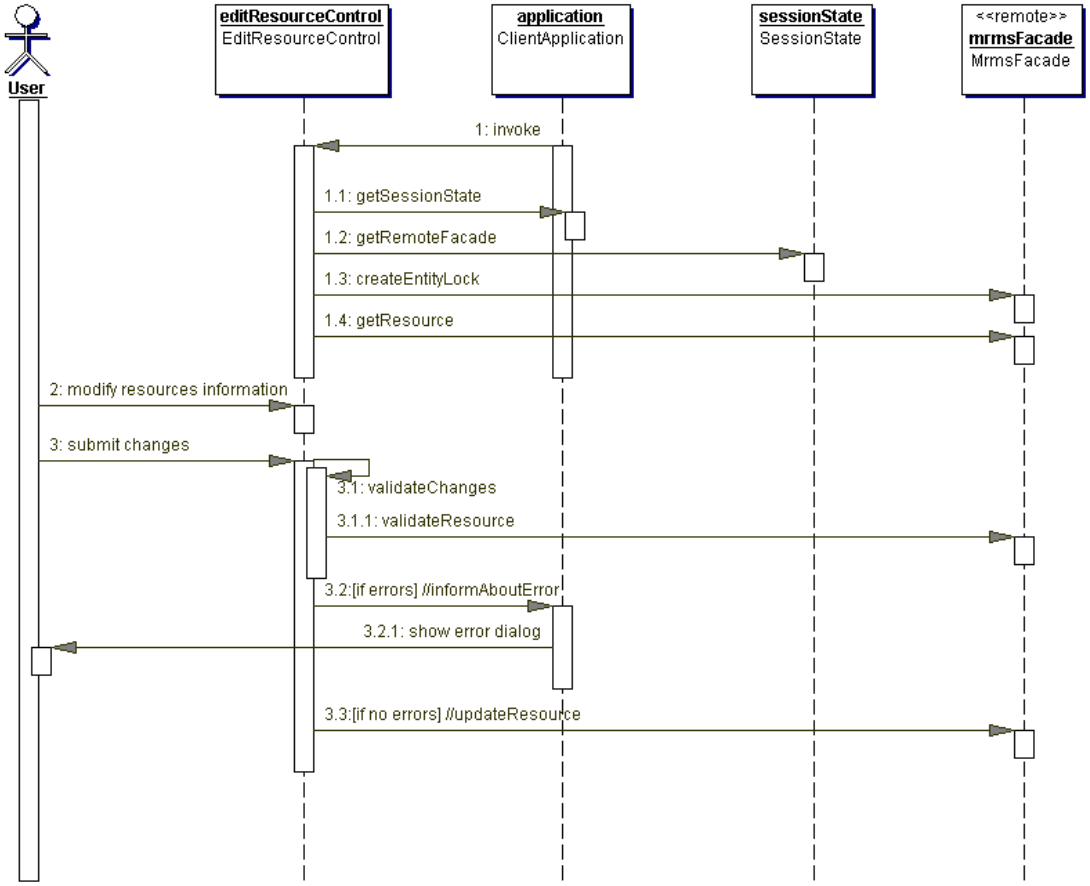


Figure 8. Sequence: EditResourceControl user interaction



6.4. ViewContainer

Description A *ViewContainer* is a component were the *ClientApplication* may plug in *AbstractViews* of *AbstractControls*.

Attributes ---

Operations

- addView(view: AbstractView,location: int): void

  Effect Adds the given *AbstractView*.

  Parameters *view*: *AbstractView* to be added

  location: An identifier determining the location to place the *AbstractView*

  Return ---

  Exceptions ---

  Actor *ClientApplication*

- removeView(view: AbstractView): void

Effect	Removes the given *AbstractView*.
Parameters	*view*: *AbstractView* to be removed
Return	---
Exceptions	---
Actor	*ClientApplication*

6.5. UserLogsInControl

Description A concrete *AbstractControl* for logging a user in.

Attributes ---

Operations ---

6.6. NavigationControl

Description A concrete *AbstractControl* for navigating entities managed by the MRMS system.

Attributes ---

Operations ---

6.7. EditResourceControl

Description A concrete *AbstractControl* for editing *Resources* managed by the MRMS system.

Attributes ---

Operations ---

6.8. CreateFilterControl

Description A concrete *AbstractControl* editing *Resources* managed by the MRMS system.

Attributes ---

Operations ---

6.9. NavigationView

Description A concrete *AbstractView* for navigating entities managed by the MRMS system.

Attributes ---

Operations ---

6.10. EditResourceView

Description A concrete *AbstractView* for editing *Resources* managed by the MRMS system.

Attributes ---

Operations ---

6.11. CreateFilterView

Description A concrete *AbstractView* for creating a *Filter* on *Resources* managed by the MRMS system.

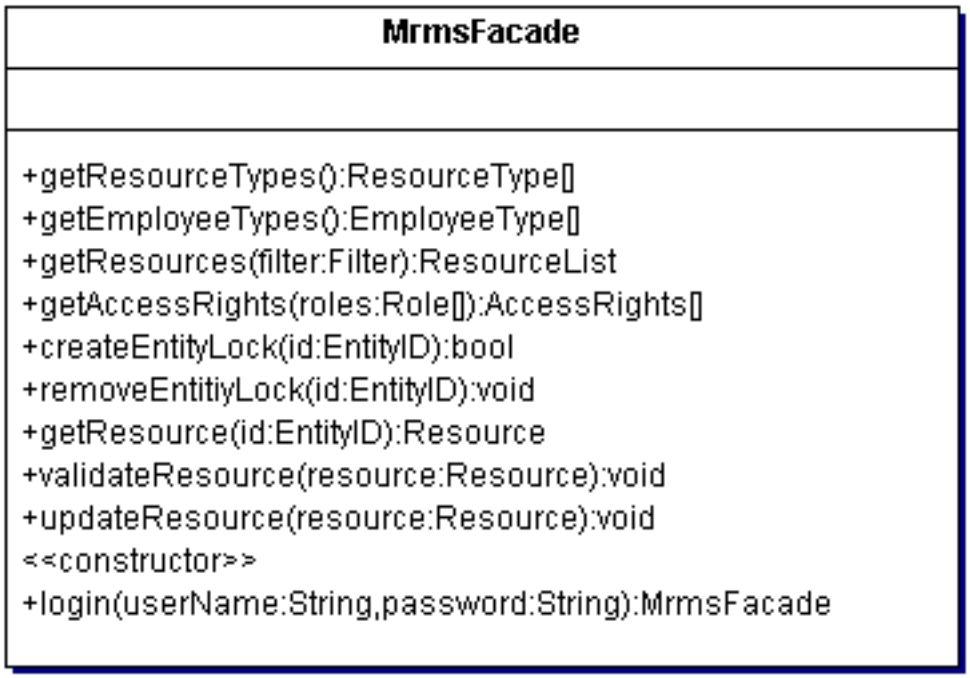
Attributes ---

Operations ---

7. Package: server

Currently we have only modelled the facade over which the client accesses the server. It provides methods to navigate over and to change the MRMS's data.

Figure 9. The MRMS's Facade which can be accessed by the client



8. Appendix: Use Cases

These are the use cases derived from the additional functionality “Resource Reservation”.

8.1. Create resource reservation

Goal	A new reservation for a resource is created.
Category	Primary
External Actors	User
Precondition	A user is logged in who has proper access rights to create the new resource reservation.
Triggering Event	The user requests the system to create a new resource reservation.
Postcondition Success	A new resource reservation has been created according to the users input.
Postcondition Failure	No new resource reservation has been created.
Description	1. The system requests the user to choose the resource, start-, endtime of the reservation and the customer. 2. The user determines resource reservation and submits his input. 3. The system creates a new resource reservation.
Extensions	---
Alternatives	---
Additional Requirements	---
Annotation	---

8.2. Delete resource reservation

Goal	The reservation for a resource is deleted.
Category	Primary
External Actors	User
Precondition	A user is logged in who has proper access rights to delete the resource reservation.
Triggering Event	The user requests the system to delete a resource reservation.
Postcondition Success	The resource reservation has been deleted.
Postcondition Failure	The resource reservation has not been deleted.
Description	1. The system requests the user to choose a resource reservation. 2. The user determines the resource reservation to delete and submits his input. 3. The system deletes the resource reservation.

Extensions	---
Alternatives	---
Additional Requirements	---
Annotation	---

8.3. Change resource reservation

Goal	The reservation for a resource is changed.
Category	Secondary
External Actors	User
Precondition	A user is logged in.
Triggering Event	The user requests the system to create a filtered collection of resource reservations.
Postcondition Success	The user is shown a collection of resource reservations that passed the filter he created.
Postcondition Failure	---
Description	1. The system requests the user to choose a resource reservation for changing. 2. The user determines the resource reservation to change and submits his input. 3. The system changes the resource reservation.
Extensions	---
Alternatives	---
Additional Requirements	---
Annotation	---

8.4. Create filtered collection of resource reservation entries

Goal	Collect a set of resources reservations meeting a specific criterion and offer it to the user for further processing.
Category	Secondary
External Actors	User
Precondition	A user is logged in who has proper access rights to change the resource reservation.
Triggering Event	The user requests the system to change a resource reservation.

Postcondition Success	The resource reservation has been changed.
Postcondition Failure	The resource reservation has not been changed.
Description	

1. The system requests the user to configure a filter listed resource reservations will have to pass
2. The system collects all resource reservations passing the specified filter and offers them to the user for further processing.

Extensions	---
Alternatives	---
Additional Requirements	---
Annotation	---

9. Appendix: .NET Event Handling

The MRMS is implemented for the .NET platform and therefore partly builds up on the .NET model for event handling. The following two figures are an overview on how that mechanism works.

Figure 10. Class Diagram: Classes within the .NET event model

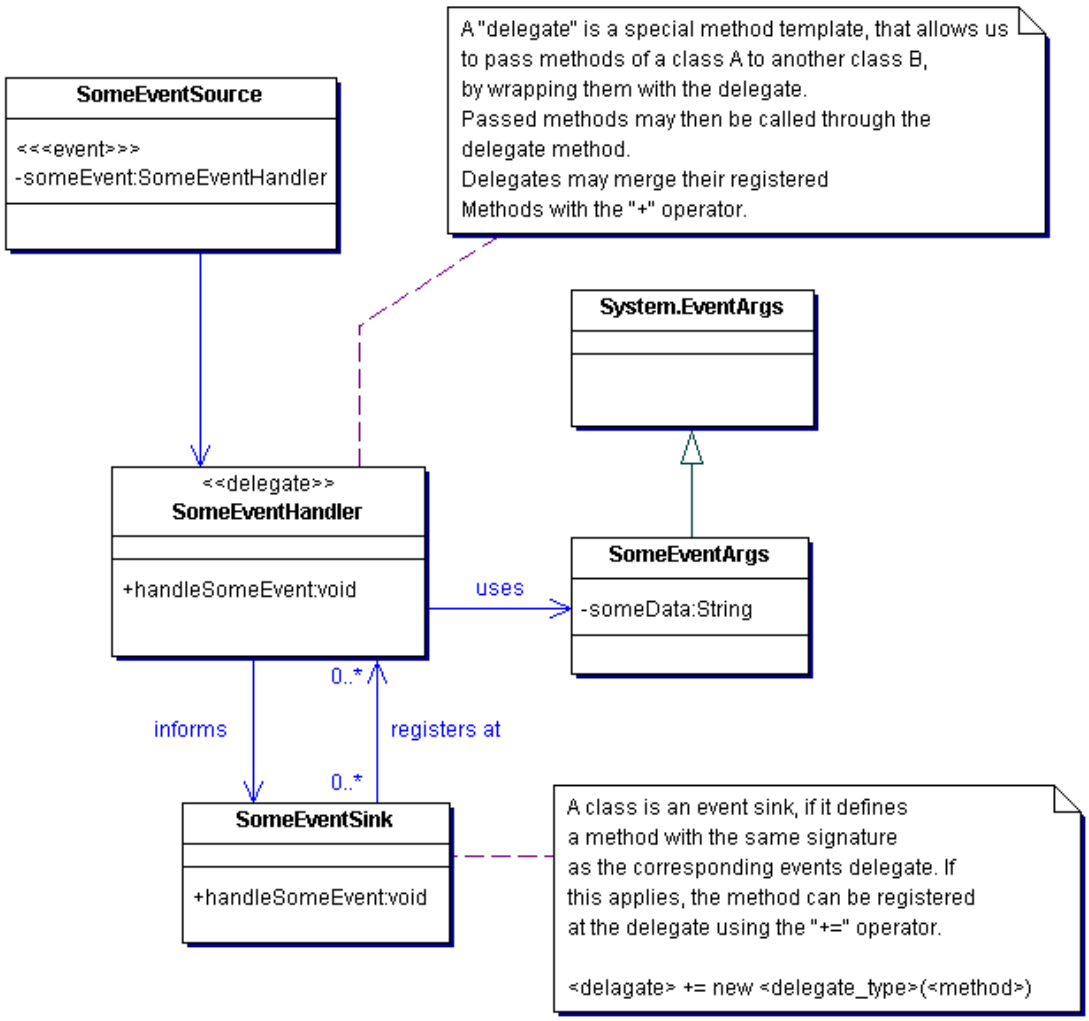
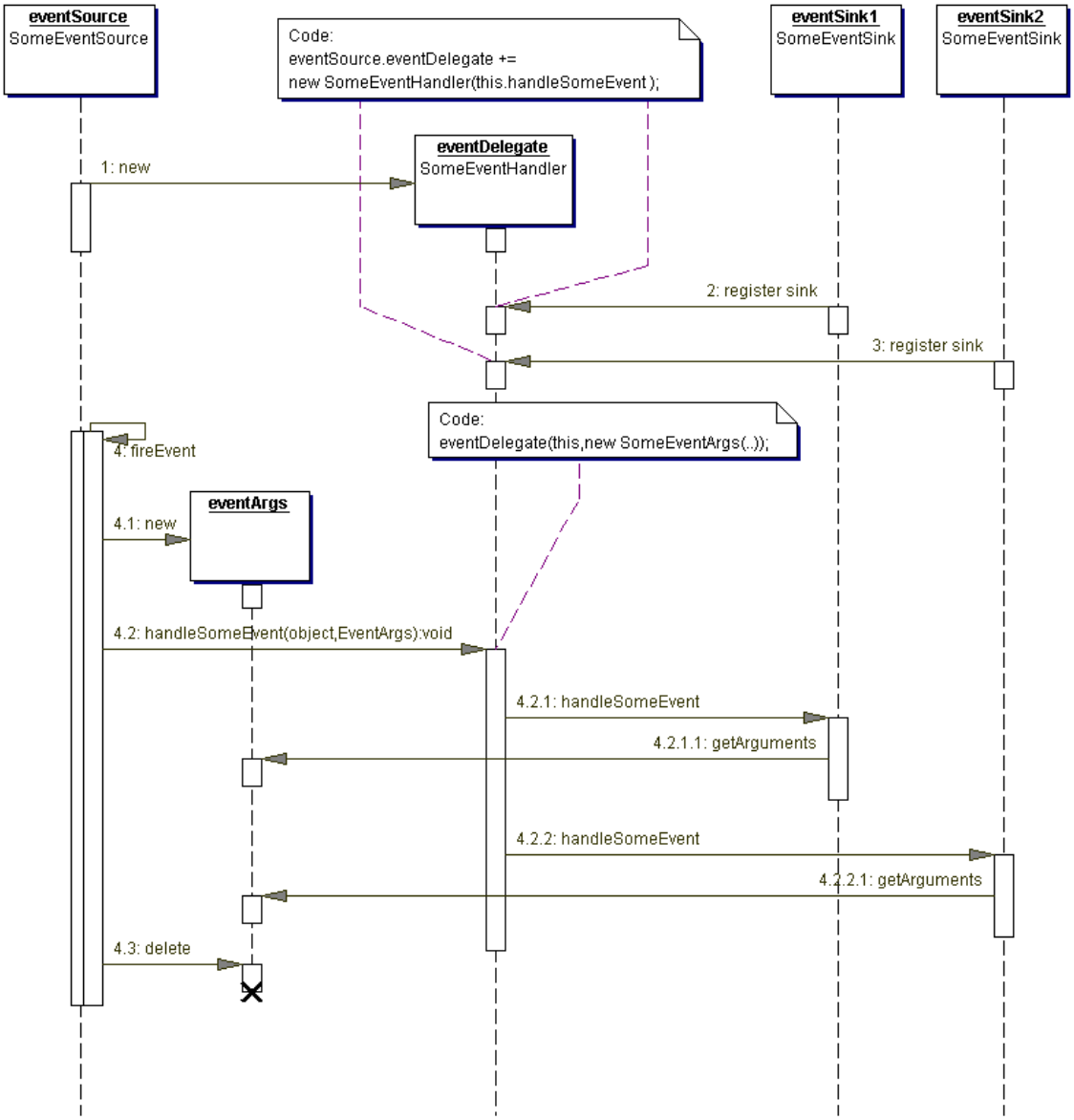


Figure 11. Sequence: Sample setup and action


eventSource
SomeEventSource
Code:
eventSource.eventDelegate +=
new SomeEventHandler(this.handleSomeEvent);
eventSink1
SomeEventSink
eventSink2
SomeEventSink
eventDelegate
SomeEventHandler
1: new
2: register sink
3: register sink
Code:
eventDelegate(this,new SomeEventArgs(..));
4: fireEvent
eventArgs
4.1: new
4.2: handleSomeEvent(object,EventArgs):void
4.2.1: handleSomeEvent
4.2.1.1: getArguments
4.2.2: handleSomeEvent
4.2.2.1: getArguments
4.3: delete