ActionGUI
A Model-Driven Methodology for Developing Secure Data-Intensive Applications

Marco Guarnieri

Institute of Information Security
ETH Zürich

September 4, 2013

Visit us!
http://www.actiongui.org

Institute of Information Security

D. Basin

G. Ortiz

T. Weghorn

M. Guarnieri

Modeling Lab

M. Clavel

M. Egea (ATOS)

M. A. García

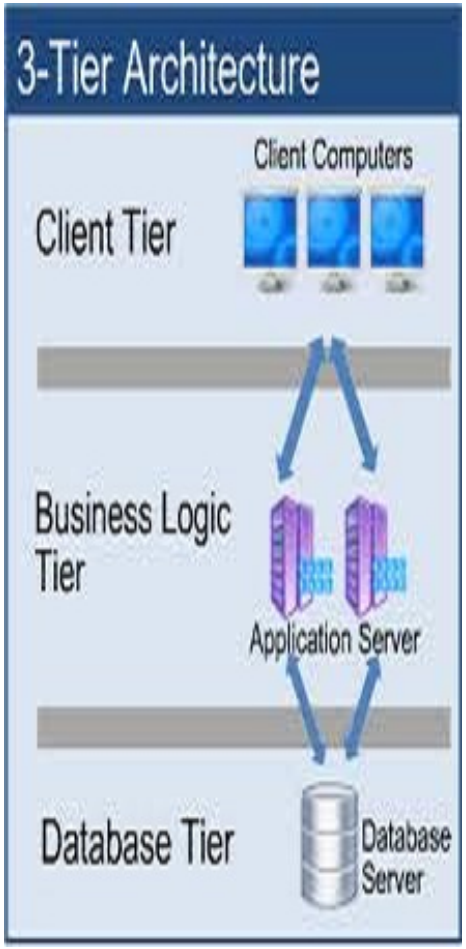
C. Dania

Model-driven engineering (MDE)

- Model-driven engineering is a software development technique that *defines* systems using *models* and *generates implementations* from these models
  - Models are artifacts (graphical or textual) that *specify the different components, aspects or views of a system*
  - Models have a precise meaning and semantics
  - Models are *technology and platform independent* artifacts
    - Transformation technologies are used to convert models into implementation code

Data-centric applications

- Data-centric applications: applications where data and its management plays a key role
- They are often implemented as multi-tier systems
  - The application manipulates data stored in a database
  - The application interacts with users through a graphical interface
- Data centric applications might process sensitive data (e.g., eHealth Records), therefore **security** is a concern



Access control policies for data-centric applications

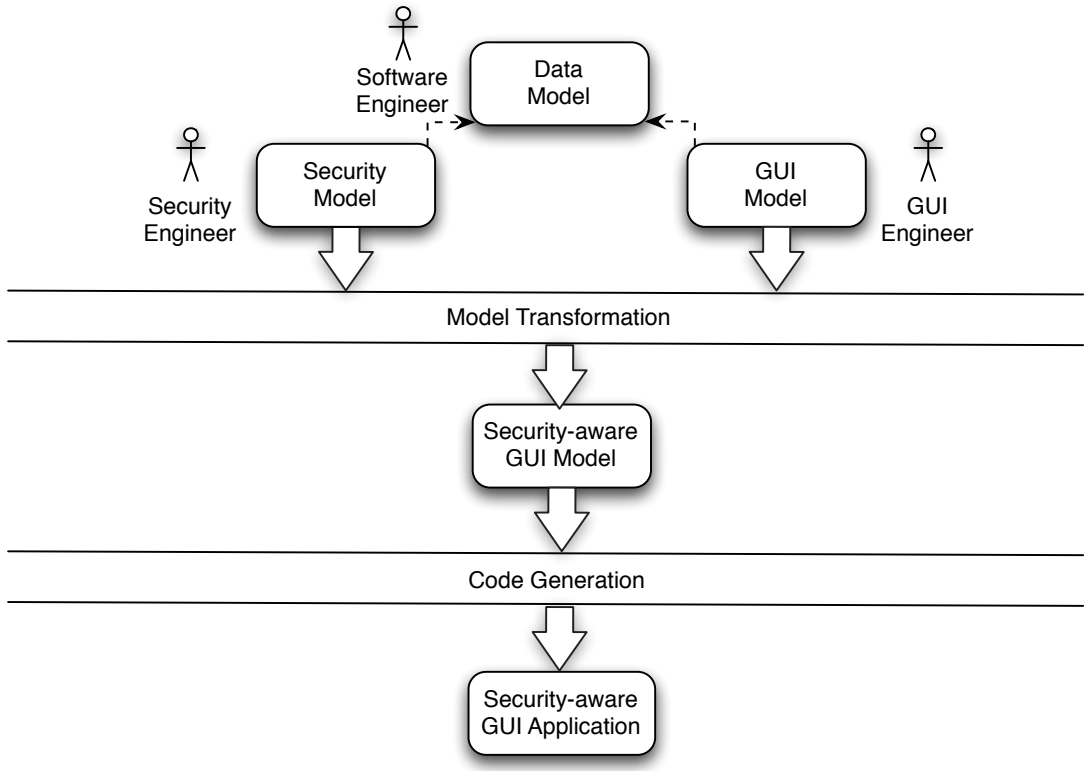
- Access control policies regulate the access to the data stored in the database
  - Declarative access control (e.g., role-based access control): access control decisions depend on the user's credentials (e.g., roles)
  - Fine-grained access control: access control decisions depend also on the satisfaction of constraints on the current state of the database

Access control policies for data-centric applications

- Enforcing access control policies on data-centric applications is nontrivial
  - Authorization checks are typically implemented by directly encoding checks at appropriate places in the application's code
  - Cumbersome, error prone, and scales poorly
  - Difficult to audit and maintain:
    - the authorization checks are spread throughout the application's code
    - each time we change the security policy, we have to change application's code

Secure data-centric application development

- An application is modeled using three different models: a *data model*, a *security model*, and a *GUI model*

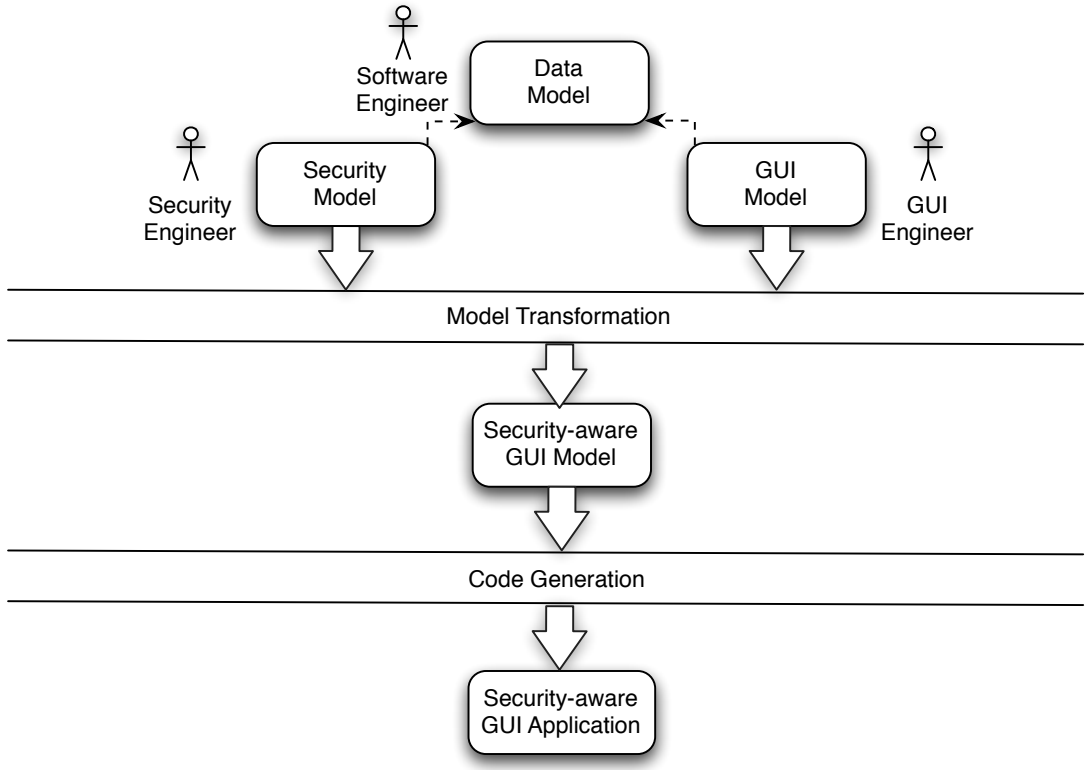


Secure data-centric application development

- The data model defines the application's data domain in terms of its classes, attributes, associations, and methods
- The security model defines the application's security policy in terms of authorized access to the actions on the resources provided by the data model (data actions)
- The GUI model defines the application's graphical interface and application logic i.e, it formalizes both *layout* and *control* (behavior) information

Secure data-centric application development

- *The key component*: A model transformation function that automatically enforces the access control policy on the GUI model (each data action is executed iff the authorization check specified in the security model holds)



Secure data-management application development - benefits

- By working with models, designers can focus on the application's data, behavior, and presentation, independent of the different, often complex, technologies that are used to implement them
- Our use of model transformations leads to modularity and separation of concerns:
  - the GUI model and the security model can be changed independently and by different shareholders (application developer vs security administrator)
  - it avoids the problems with brittle, error prone, hard-coded security policies that are difficult to maintain and audit

ActionGUI Toolkit

- ActionGUI features specialized model editors for constructing and manipulating data, security and GUI models
- It implements the model transformation that lifts up the policy that is specified in the security model to the GUI model
- From the resulting security-aware GUI model, it generates a complete, deployable, web application, along with all support for fine-grained access control

Running Example

- Our running example is a simple message board
- In a message board *Persons* can publish *Messages*, and *Replies* to *Messages*
- **Functional Requirements**
  - Each *Person* should be allowed to completely manage (create, edit, delete) its own *Messages* and *Replies*
- **Security Requirements**
  - Only the author of a *Message*/*Reply* can edit or delete the *Message*/*Reply*

Data Model - Message Board

- There are three main concepts:
- **Person**
  - three attributes: *login*, *password*, and *personalRole*
  - two associations: $messages \subseteq Person \times Message$, and $replies \subseteq Person \times Reply$
- **Message**
  - two attributes: *title*, *text*
  - two associations: $messageOwner \subseteq Message \times Person$, and $messageReplies \subseteq Message \times Reply$
- **Reply**
  - one attribute: *text*
  - two associations: $replyOwner \subseteq Reply \times Person$, and $message \subseteq Reply \times Message$

Data Model - *ComponentUML*

ComponentUML is a simplified version of the UML class diagrams
In a *ComponentUML* data model we can represent:

- classes with attributes and methods
- enumerated types
- binary associations between classes

```
entity Person {
    String  name
    String  password
    Role    personalRole
    Set(Message) messages oppositeTo messageOwner
    Set(Reply) replies oppositeTo replyOwner
}

entity Message {
    String title
    String text
    Set(Person) messageOwner oppositeTo messages
    OrderedSet(Reply) messageReplies oppositeTo message
}

entity Reply {
    String text
    Set(Person) replyOwner oppositeTo replies
    Set(Message) message oppositeTo messageReplies
}
```

Security Model - SecureUML

SecureUML is a modeling language for specifying stateful Role-Based Access Control (RBAC) policies

- It supports the modeling of *roles* and their hierarchies, *permissions*, *actions*, *resources*, and *authorization constraints*

Resource	Atomic Actions	Composite Actions
Entity	create, delete	read, update, full access
Attribute	read, update	full access
Method	execute	
Association-end	read, add, remove	full access

Security Model - SecureUML

- Authorization constraints are specified using OCL predicates
- The context of an authorization constraint is the underlying data model. It can contain the following variables:
  - *self*: refers to the resource upon which the action will be performed.
  - *caller*: refers to the user that will perform the action
  - *value*: refers to the value that will be used to update an attribute
  - *target*: refers to the object that will be added (or removed) at the end of an association

Security Model - Message Board

- Only one role `User`. Some of the permissions of a `User` u are:
  - ...
  - create an instance m of class *Message*
  - delete an instance m of class *Message* iff u is the author of m and m does not contain any reply
  - read the values of *title*, *text*, *messageOwner*, *messageReplies*
  - update the values of *title* and *text* of *Message* m iff u is the author of m
  - add a user u' to the association *messageOwner* of a *Message* m iff $u = u'$ and m does not have an owner yet
  - remove a user u' from the association *messageOwner* of a *Message* m iff $u = u'$ and u' is the owner of m
  - add a reply r to the association *messageReplies* of a *Message* m iff u is the author of r and r is not in the replies of m
  - remove a reply r from the association *messageReplies* of a *Message* m iff u is the author of r and r is in the replies of m
  - ...

Security Model - Message Board

```
role USER {
...
  Message {
        create, read title, read text, read messageOwner, read messageReplies

        delete constrainedBy [self.messageOwner->includes(caller)  and
      self.messageReplies->size()=0]

        update title, update text constrainedBy
      [self.messageOwner->includes(caller)]

        add messageOwner constrainedBy [self.messageOwner->size()=0 and target
      = caller]
        remove messageOwner constrainedBy [self.messageOwner->includes(caller)
      and target = caller]

        add messageReplies constrainedBy [target.replyOwner->includes(caller)
      and not(self.messageReplies->includes(target))]
        remove messageReplies constrainedBy[target.replyOwner->includes(caller)
      and self.messageReplies->includes(target)]
      }
      ...
}
```

GUI Model - GUIML

GUIML provides support to formally model widgets, events, and actions, which can be on data or on other widgets. More specifically:

- Widgets can be displayed in containers, which are also widgets.
- Widgets may own variables, which are in charge of storing information for later use.
- Widgets may trigger events, which may themselves execute actions.

The layout of the web application must be defined in separate CSS files (in the current version of ActionGUI)

GUI Model - GUIML

- Actions may take their arguments (values that instantiate their parameters) from the information stored in the widgets' variables or in the database.
- Actions' conditions and arguments are specified in GUI models using OCL, extended with the widget's variables.
- GUIML support several statements
  - if-then-else statements where conditions can depend on the information stored in the widgets' variables or in the database
  - for-each statements
  - try-and-catch statements

GUI Model - Message Board

- Our Message Board application has 8 different windows:
  - `LoginWindow` contains the registration and login procedure,
  - `MainWindow` shows the *Messages* and allows their management
  - `CreateMessageWindow` allows the creation of a new *Message*
  - `EditMessageWindow` can be used to modify an already existing *Message*
  - `ViewMessageWindow` can be used to see the detailed content and the replies of a *Message*
  - `CreateReplyWindow` allows the creation of a new *Reply* to a given *Message*
  - `EditMessageWindow` can be used to modify a *Reply*
  - `ViewMessageWindow` can be used to see the detailed content of a *Reply*

GUI Model - GUIML

```
Window CreateNewMessage{
    ...
    Button Publish_B {
     ...
        event onClick {
            ...
            if [$MessageTitle_TF.error$ = null and $MessageText_TF.error$ =
     null] {
                newMessage := new Message
                [$newMessage$.messageOwner] += [$CreateMessageWindow.caller$]
                [$newMessage$.title] := [$MessageTitle_TF.text$]
                [$newMessage$.text] := [$MessageText_TF.text$]

                MessageTitle_TF.text := [null]
                MessageText_TF.text := [null]

                notification (['Success'],['An instance of Message has been
     created successfully.'],[500])
            }
            else {
                notification(['Error'],['The form contains errors. Please,
     check the form.'],[500])
            }
        }
     }
    ...
}
```

Demo

- We modify the message board application presented before in the following way:
  - We enforce the security policy "Only the author of a *Message*/*Reply* can edit or delete his own *Messages*/*Replies*"
  - We add a new role (**Moderator**) with administrative rights (can delete/edit any user's *Message* or *Reply* regardless of the author and on the presence of replies to a message)
  - We extend the application with a new functionality: a *Person* can create private messages that are shared with a group of friends and only *Persons* in this group can view and reply to the message

Demo - 1

- We enforce the security policy "Only the author of a *Message*/*Reply* can edit or delete his own *Messages*/*Replies*".
- This can be done in two steps:
  - Create the security policy
  - Move all data actions in `try/catch` blocks to deal with possible `SecurityExceptions`

Demo - 1

```
role USER {
  ...
  Message {
        create, read title, read text, read messageOwner, read messageReplies

        delete constrainedBy [self.messageOwner->includes(caller)  and
      self.messageReplies->size()=0]

        update title, update text constrainedBy
      [self.messageOwner->includes(caller)]

        add messageOwner constrainedBy [self.messageOwner->size()=0 and target
      = caller]
        remove messageOwner constrainedBy [self.messageOwner->includes(caller)
      and target = caller]

        add messageReplies constrainedBy [target.replyOwner->includes(caller)
      and not(self.messageReplies->includes(target))]
        remove messageReplies constrainedBy[target.replyOwner->includes(caller)
      and self.messageReplies->includes(target)]
      }
  ...
}
```

Demo - 1

```
role USER {
  ...
  Reply {
        create, read text, read replyOwner, read message
        delete constrainedBy [self.replyOwner->includes(caller)]

        update text constrainedBy [self.replyOwner->includes(caller)]

        add replyOwner constrainedBy [self.replyOwner->size()=0 and target =
     caller]
        remove replyOwner constrainedBy [self.replyOwner->size()=1 and target =
     caller and self.replyOwner->includes(caller)]

        add message constrainedBy [self.replyOwner->includes(caller) and
     not(self.message->includes(target))]
        remove message constrainedBy [self.replyOwner->includes(caller) and
     self.message->includes(target)]
     }
  ...
}
```

Demo - 1

```
role USER {
  ...
  Person {
        read login, read personalRole
        read password, update password, update login constrainedBy [caller =
    self]
        read messages, read replies constrainedBy [caller = self]

        add messages constrainedBy [target.messageOwner->size()=0 and self =
    caller]
        remove messages constrainedBy [target.messageOwner->size()=1 and self =
    caller and target.messageOwner->includes(caller)]

        add replies constrainedBy [target.replyOwner->size()=0 and self =
    caller]
        remove replies constrainedBy [target.replyOwner->size()=1 and self =
    caller and target.replyOwner->includes(caller)]

    }
}
```

Demo - 1

```
Window CreateNewMessage{
    ...
    Button Publish_B {
     ...
        event onClick {
            ...
            if [$MessageTitle_TF.error$ = null and $MessageText_TF.error$ =
     null] {
             try{
                 newMessage := new Message
                 [$newMessage$.messageOwner] += [$CreateMessageWindow.caller$]
                 [$newMessage$.title] := [$MessageTitle_TF.text$]
                 [$newMessage$.text] := [$MessageText_TF.text$]
                 MessageTitle_TF.text := [null]
                 MessageText_TF.text := [null]
                 notification (['Success'],['An instance of Message has been
     created successfully.'],[500])
                }catch(SecurityException)
                {
                  notification(['Error'],['You don't have enough permission for
     creating the Message.'],[500])
                }

            }
            else {
                notification(['Error'],['The form contains errors. Please,
     check the form.'],[500])
            }
        }
     }
    ...
}
```

Demo - 2

- We add a new role (**Moderator**) with administrative rights (can delete/edit any user's *Message* or *Reply* regardless of the author and on the presence of replies to a message)
- This can be done in two steps:
  - Add a new role `MODERATOR` and adjust the security policy
  - Create a button for registering moderators to the system

Demo - 2

```
Button RegisterMod_B {
        String text := ['Register as Moderator']
        event onClick {
            ...
            if [$Name_TF.error$ = null and $Password_TF.error$ = null] {
                if[Person.allInstances()->forAll(c|c.login <> $Name_TF.text$)] {
                    newUser := new Person
                    [$newUser$.login] := [$Name_TF.text$]
                    [$newUser$.password] := [$Password_TF.text$]
                    [$newUser$.personalRole] := [Role::MODERATOR]
                    open MainWindow(caller:[$newUser$],
    role:[$newUser$.personalRole])
                } else {
                    notification(['Error'],['This login name already exists.
    Choose another one.'],[500])
                    Name_TF.text :=  [null]
                    Password_TF.text :=  [null]
                }
            }
            else {
                notification(['Error'],['The form contains errors. Please,
    check the form.'],[500])
            }
        }
    }
```

Demo - 2

```
role MODERATOR extends USER{
    Person {
        read messages, read replies

        add messages constrainedBy [target.messageOwner->size()=0 and self =
     caller]
        remove messages constrainedBy [target.messageOwner->size()=1 and self =
     caller and target.messageOwner->includes(caller)]

        add replies constrainedBy [target.replyOwner->size()=0 and self =
     caller]
        remove replies constrainedBy [target.replyOwner->size()=1 and self =
     caller and target.replyOwner->includes(caller)]
        read personalRole
     }
    ...
}
```

Demo - 2

```
role MODERATOR extends USER{
    ...
    Message {
        // a moderator can delete any user's message
        delete constrainedBy [self.messageOwner->forAll(u | u.personalRole =
     Role::USER)]

        // a moderator can edit any user's message
        update title, update text constrainedBy [self.messageOwner->forAll(u |
     u.personalRole = Role::USER)]

     }
    ...
}
```

Demo - 2

```
role MODERATOR extends USER{
    ...
    Reply {
        // a moderator can delete any user's reply
        delete constrainedBy [self.replyOwner->forAll(u | u.personalRole =
     Role::USER)]

        // a moderator can edit any user's reply
        update text constrainedBy [self.replyOwner->forAll(u | u.personalRole =
     Role::USER)]
     }
}
```

Demo - 3

- We extend the application with a new functionality: a *Person* can create private messages that are shared with a group of friends and only *Persons* in this group can view and reply to the message
- This can be done in three steps:
  - Extend the data model with the new association ends $friendMessages \subseteq Person \times Message$ and $sharedWith \subseteq Message \times Person$
  - Modify the security policy
  - Modify the `CreateMessageWindow` and `EditMessageWindow` windows

Demo - 3

```
 entity Person {
    String login
    String password
    Role personalRole
    Set(Message) messages oppositeTo messageOwner
    Set(Reply) replies oppositeTo replyOwner
    Set(Message) friendsMessages oppositeTo sharedWith
}


entity Message {
    String title
    String text
    Set(Person) messageOwner oppositeTo messages
    OrderedSet(Reply) messageReplies oppositeTo message
    Set(Person) sharedWith oppositeTo friendsMessages
}
```

Demo - 3

```
role USER{
Person {
        read login, read personalRole
        read password, update password, update login constrainedBy [caller =
      self]
        read messages, read replies, read friendsMessages constrainedBy [caller
      = self]

        add messages constrainedBy [target.messageOwner->size()=0 and self =
      caller]
        remove messages constrainedBy [target.messageOwner->size()=1 and self =
      caller and target.messageOwner->includes(caller)]

        add replies constrainedBy [target.replyOwner->size()=0 and self =
      caller]
        remove replies constrainedBy [target.replyOwner->size()=1 and self =
      caller and target.replyOwner->includes(caller)]

        add friendsMessages constrainedBy
      [not(target.messageOwner->includes(self)) and
      target.messageOwner->includes(caller) and
      not(self.friendsMessages->includes(target))]
        remove friendsMessages constrainedBy
      [not(target.messageOwner->includes(self)) and
      target.messageOwner->includes(caller) and
      self.friendsMessages->includes(target)]
      }
...
}
```

Demo - 3

```
role USER{
...
    Message {
        create
        delete constrainedBy [self.messageOwner->includes(caller)  and
     self.messageReplies->size()=0]

        read title, read text, read messageOwner, read messageReplies, read
     sharedWith constrainedBy [self.messageOwner->includes(caller) or
     self.sharedWith->includes(caller) or self.sharedWith->isEmpty()]
        update title, update text constrainedBy
     [self.messageOwner->includes(caller)]

        add messageOwner constrainedBy [self.messageOwner->size()=0 and target
     = caller]
        remove messageOwner constrainedBy [self.messageOwner->size()=1 and
     target = caller and self.messageOwner->includes(caller)]
        ...
     }
...
}
```

Demo - 3

```
role USER{
...
    Message {
        ...
        add messageReplies constrainedBy [target.replyOwner->includes(caller)
     and not(self.messageReplies->includes(target)) and
     (self.sharedWith->includesAll(target.replyOwner) or
     self.sharedWith->isEmpty() or
     self.messageOwner->includesAll(target.replyOwner))]
        remove messageReplies constrainedBy
     [target.replyOwner->includes(caller) and
     self.messageReplies->includes(target) and
     (self.sharedWith->includesAll(target.replyOwner) or
     self.sharedWith->isEmpty() or
     self.messageOwner->includesAll(target.replyOwner))]

        add sharedWith constrainedBy [not(self.messageOwner->includes(target))
     and self.messageOwner->includes(caller) and
     not(self.sharedWith->includes(target))]
        remove sharedWith constrainedBy
     [not(self.messageOwner->includes(target)) and
     self.messageOwner->includes(caller) and self.sharedWith->includes(target)]
     }
...
}
```

Demo - 3

```
role USER{
...
      Reply {
         create
         delete constrainedBy [self.replyOwner->includes(caller)]

         read text, read replyOwner, read message constrainedBy
      [self.replyOwner->includes(caller) or
      self.message.messageOwner->includes(caller) or
      self.message.sharedWith->includes(caller)  or
      self.message.sharedWith->isEmpty()]
         update text constrainedBy [self.replyOwner->includes(caller)]

         add replyOwner constrainedBy [self.replyOwner->size()=0 and target =
      caller]
         remove replyOwner constrainedBy [self.replyOwner->size()=1 and target =
      caller and self.replyOwner->includes(caller)]

         add message constrainedBy [self.replyOwner->includes(caller) and
      not(self.message->includes(target)) and
      (self.message.sharedWith->includes(caller) or
      self.message.sharedWith->isEmpty() or
      self.message.messageOwner->includes(caller))]
         remove message constrainedBy [self.replyOwner->includes(caller) and
      self.message->includes(target) and
      (self.message.sharedWith->includes(caller) or
      self.message.sharedWith->isEmpty() or
      self.message.messageOwner->includes(caller))]

      }
}
```

Demo - 3

```
role MODERATOR extends USER{
 Person {
        read messages, read replies

        add messages constrainedBy [target.messageOwner->size()=0 and self =
      caller]
        remove messages constrainedBy [target.messageOwner->size()=1 and self =
      caller and target.messageOwner->includes(caller)]

        add replies constrainedBy [target.replyOwner->size()=0 and self =
      caller]
        remove replies constrainedBy [target.replyOwner->size()=1 and self =
      caller and target.replyOwner->includes(caller)]
        read personalRole
      }
      ...
}
```

Demo - 3

```
role MODERATOR extends USER{
      ...
     Message {
         // a moderator can delete any user's message
         delete constrainedBy [self.messageOwner->forAll(u | u.personalRole =
      Role::USER)]

         // a moderator can edit any user's message
         update title, update text constrainedBy [self.messageOwner->forAll(u |
      u.personalRole = Role::USER)]

         // a moderator can reply to any message (regardless of visibility)
         add messageReplies constrainedBy [target.replyOwner->includes(caller)
      and not(self.messageReplies->includes(target))]
         remove messageReplies constrainedBy
      [target.replyOwner->includes(caller) and
      self.messageReplies->includes(target)]

         // a moderator can read  any message (regardless of visibility)
         read title, read text, read messageOwner, read messageReplies, read
      sharedWith
      }
...
}
```

Demo - 3

```
role MODERATOR extends USER{
...
    Reply {
        // a moderator can delete any user's reply
        delete constrainedBy [self.replyOwner->forAll(u | u.personalRole =
     Role::USER)]

        // a moderator can edit any user's reply
        update text constrainedBy [self.replyOwner->forAll(u | u.personalRole =
     Role::USER)]

        // a moderator can reply to any message (regardless of visibility)
        add message constrainedBy [self.replyOwner->includes(caller) and
     not(self.message->includes(target))]
        remove message constrainedBy [self.replyOwner->includes(caller) and
     self.message->includes(target)]

        // a moderator can read  any reply (regardless of visibility)
        read text, read replyOwner, read message

    }
}
```

Demo - 3

```
role MODERATOR extends USER{
...
    Reply {
        // a moderator can delete any user's reply
        delete constrainedBy [self.replyOwner->forAll(u | u.personalRole =
     Role::USER)]

        // a moderator can edit any user's reply
        update text constrainedBy [self.replyOwner->forAll(u | u.personalRole =
     Role::USER)]

        // a moderator can reply to any message (regardless of visibility)
        add message constrainedBy [self.replyOwner->includes(caller) and
     not(self.message->includes(target))]
        remove message constrainedBy [self.replyOwner->includes(caller) and
     self.message->includes(target)]

        // a moderator can read  any reply (regardless of visibility)
        read text, read replyOwner, read message

    }
}
```

Demo - 3

```
Window CreateMessageWindow{
...
    Table ListOfUsers {
        Set(Person) rows := [Person.allInstances()->select(u | u.personalRole =
     Role::USER and u <> $CreateMessageWindow.caller$)]
        Set(Person) selected := [null]
        columns {
            ['Username'] : Label users {
                String text {
                    if [$text$.oclIsInvalid()] {
                        error := ['no permission']
                    }
                    else {
                        error := [null]
                    }
                }
                event onView (text) {
                    text := [null]
                    try {
                        text := [$ListOfUsers.row$.login]
                    }
                    catch(SecurityException) {
                        text := [invalid]
                    }
                }
            }
        }
    }
    ...
}
```

Demo - 3

```
Window CreateMessageWindow{
...
   Button Publish_B {
        String text := ['Publish it']
        event onClick {
            ...
            if [$MessageTitle_TF.error$ = null and $MessageText_TF.error$ =
     null] {
                try{
                newMessage := new Message
                [$newMessage$.messageOwner] += [$CreateMessageWindow.caller$]
                [$newMessage$.title] := [$MessageTitle_TF.text$]
                [$newMessage$.text] := [$MessageText_TF.text$]
                if [not($ListOfUsers.selected$ = null)]{
                   foreach user in [$ListOfUsers.selected$]
                   {
                         [$newMessage$.sharedWith] += [$user$]
                   }
                }
                MessageTitle_TF.text := [null]
                MessageText_TF.text := [null]
                notification (['Success'])
                }
                 catch(SecurityException)
                {
                   notification(['Error'])
                }
            }
            else { notification(['Error']) }
        }
     }
     ...
}
```

Questions?

Visit us!

http://www.actiongui.org

Institute of Information Security

D. Basin

G. Ortiz

T. Weghorn

M. Guarnieri

Modeling Lab

M. Clavel

M. Egea (ATOS)

M. A. García

C. Dania