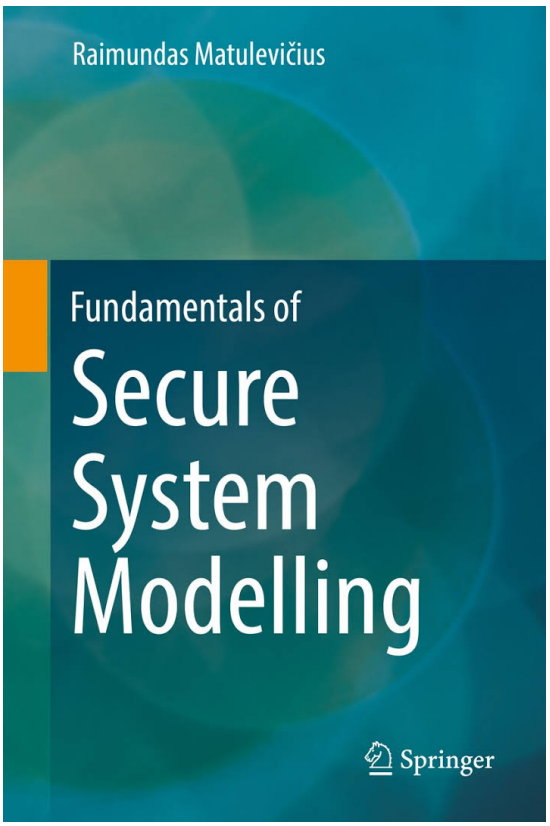


Fundamentals of
Secure System Modelling
Springer, 2017

Chapter 10:
Role-Based Access Control

Raimundas Matulevičius
University of Tartu, Estonia, rma@ut.ee

Goals

- Introduce principles of role-based access control (RBAC)
- Present requirements for RBAC solution development and administration
- Discuss how SecureUML and UMLsec could be used to define RBAC policies
- Overview principles of model driven security

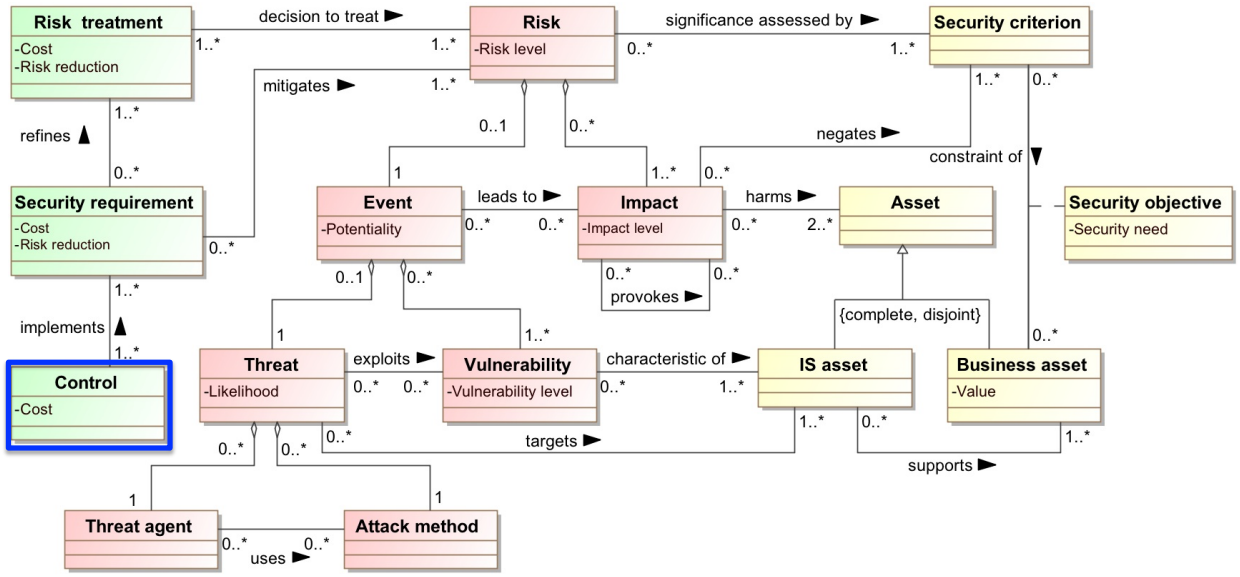
Outline

- Principles of role-based access control
- RBAC implementation requirements
- RBAC modelling languages
  - SecureUML
  - UMLsec
  - Language comparison
  - Transformation
- Model-driven security
  - Model-driven development
  - Security model transformation
- Further reading

Outline

- **Principles of role-based access control**
- RBAC implementation requirements
- RBAC modelling languages
  - SecureUML
  - UMLsec
  - Language comparison
  - Transformation
- Model-driven security
  - Model-driven development
  - Security model transformation
- Further reading

Security Risk Management Domain Model

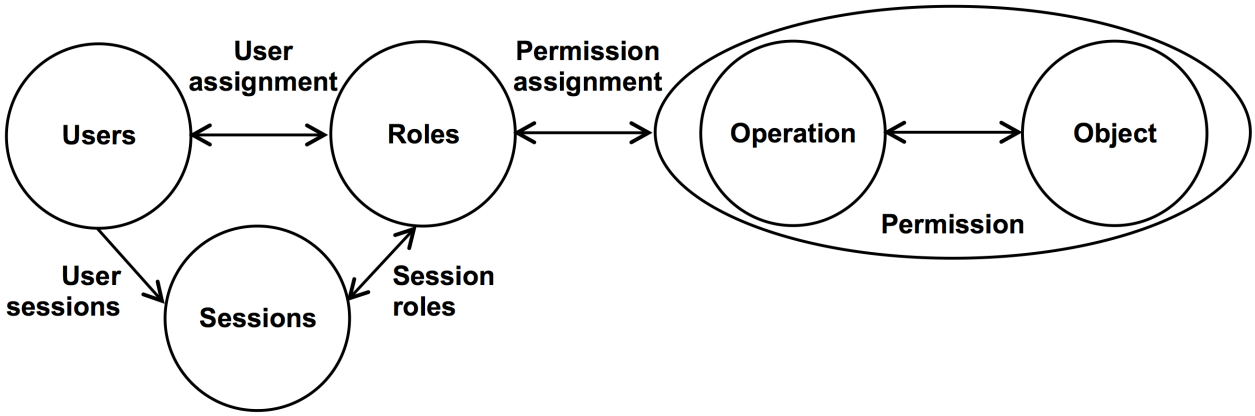


RBAC:
Role-based Access Control

Access – a specific type of interaction between a subject and an object that result in the flow of information from one to the other

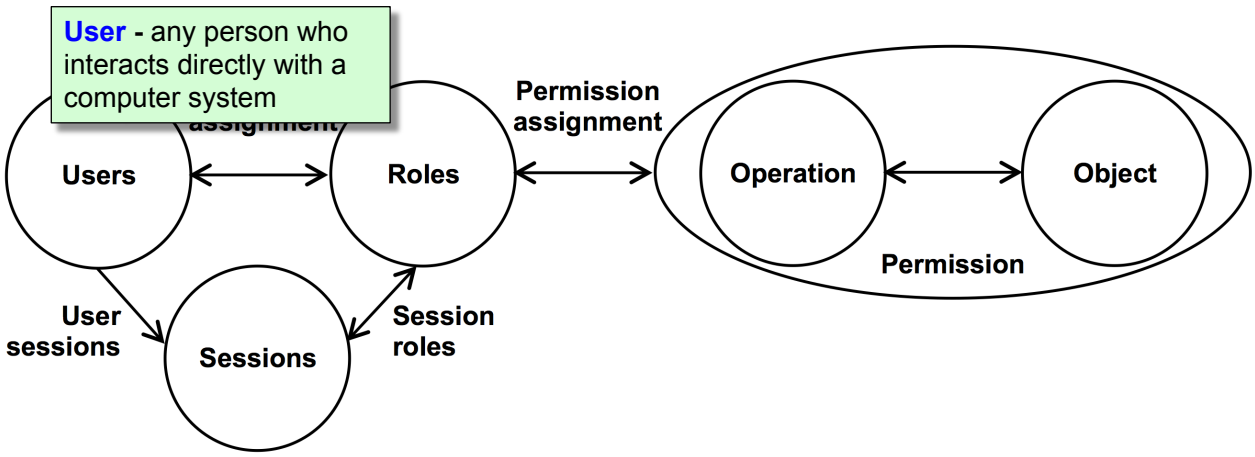
Access control – the process of limiting access to the resources of a system only to authorised programs, processes or other systems

$RBAC_0$



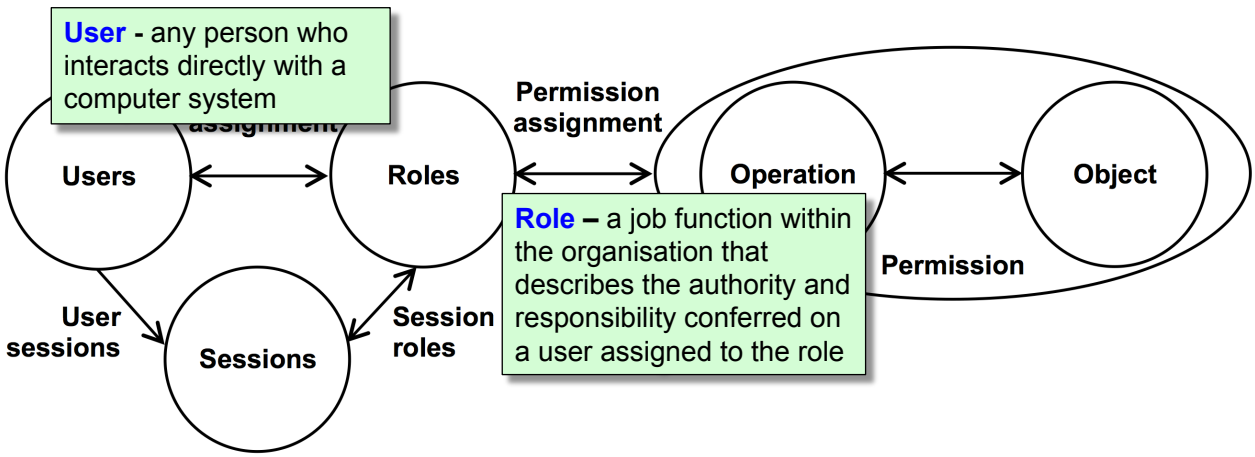
Sandhu and Coyne, 1996; Ferraiolo et al., 2001

$RBAC_0$



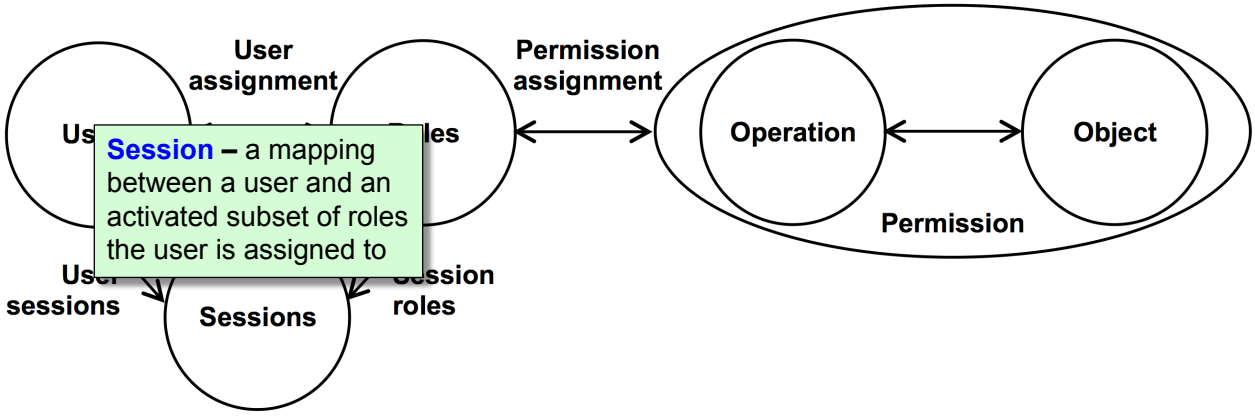
Sandhu and Coyne, 1996; Ferraiolo et al., 2001

$RBAC_0$



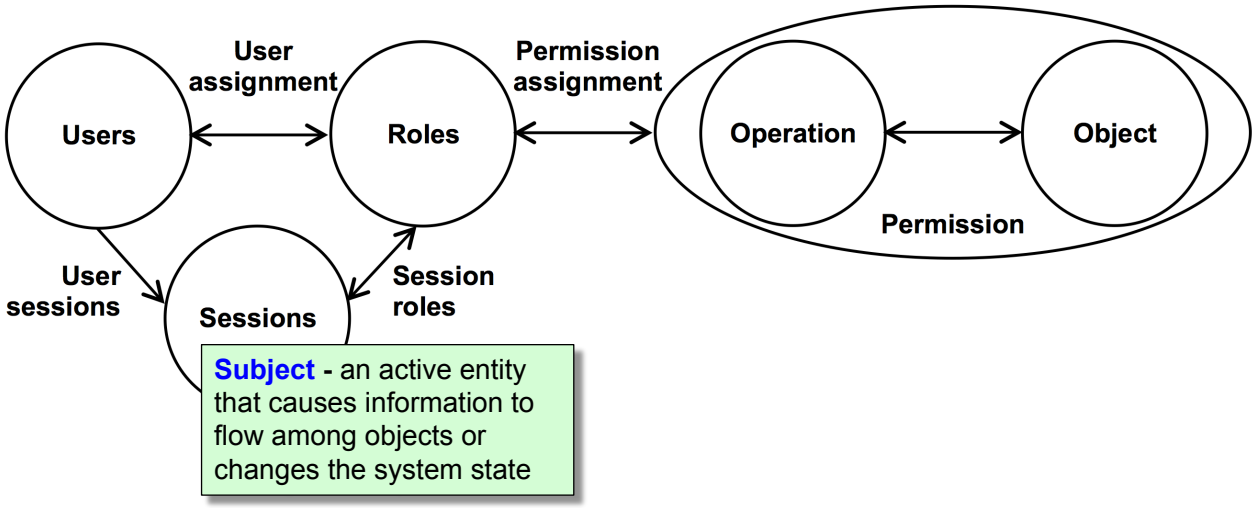
Sandhu and Coyne, 1996; Ferraiolo et al., 2001

$RBAC_0$



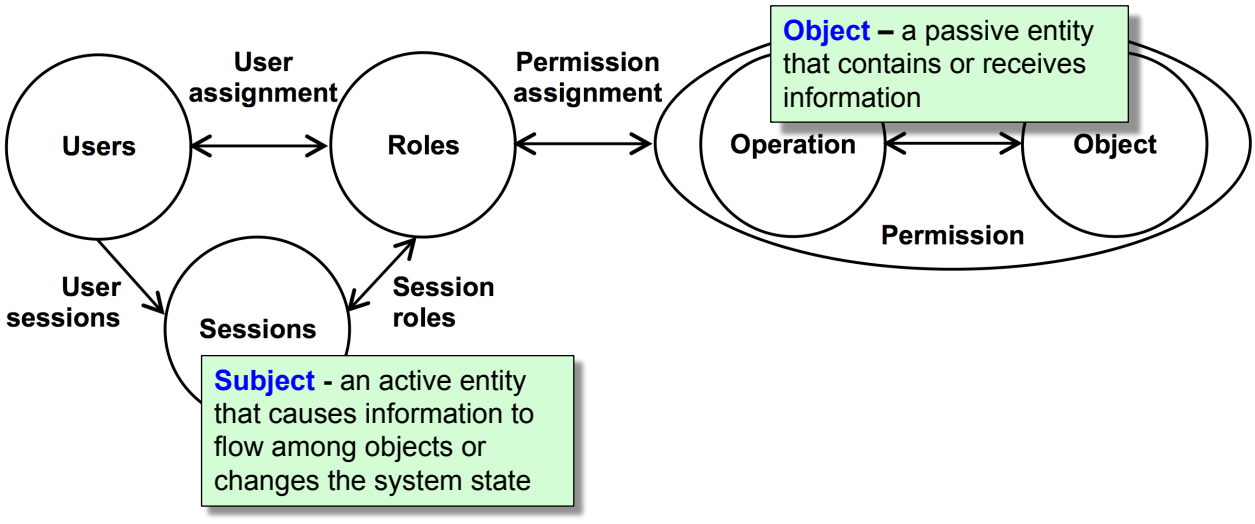
Sandhu and Coyne, 1996; Ferraiolo et al., 2001

$RBAC_0$



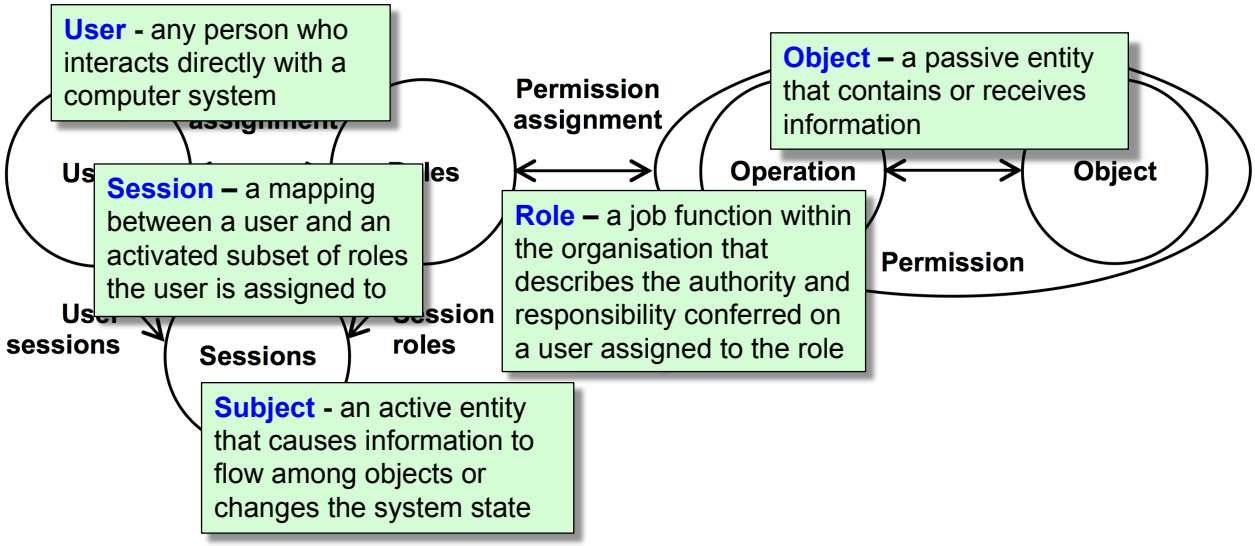
Sandhu and Coyne, 1996; Ferraiolo et al., 2001

$RBAC_0$



Sandhu and Coyne, 1996; Ferraiolo et al., 2001

$RBAC_0$



Sandhu and Coyne, 1996; Ferraiolo et al., 2001

RBAC family

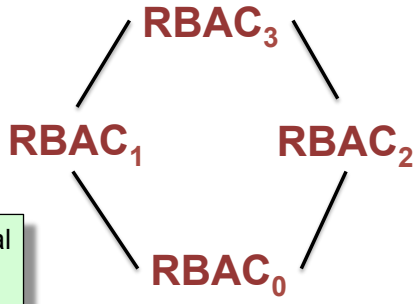
- **$RBAC_0$**
  - Everything except role hierarchies and constraints
- **$RBAC_1$**
  - $RBAC_0$ plus role hierarchies

Role hierarchy – a partial order relationship established among roles

- **$RBAC_2$**
  - $RBAC_0$ plus role constraints

Constraint – a relationship among roles

- **$RBAC_3$**
  - $RBAC_1$ plus $RBAC_2$



Sandhu and Coyne, 1996; Ferraiolo et al., 2001

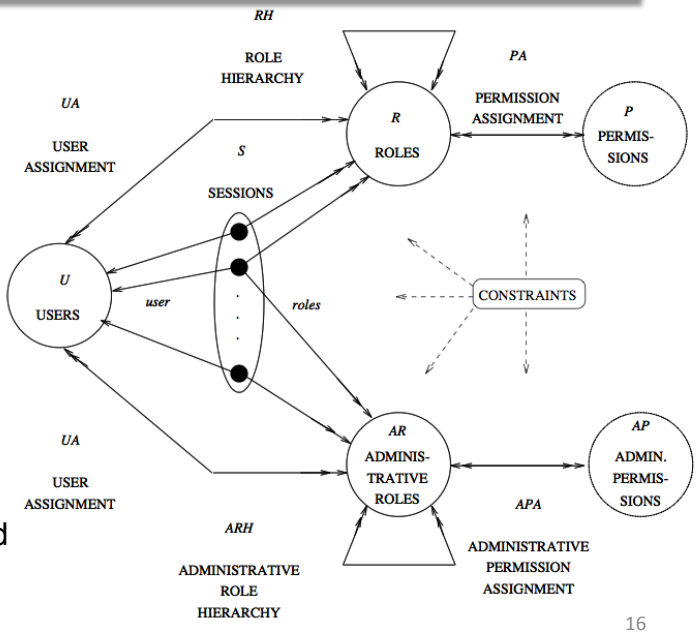
Outline

- Principles of role-based access control
- **RBAC implementation requirements**
- RBAC modelling languages
  - SecureUML
  - UMLsec
  - Language comparison
  - Transformation
- Model-driven security
  - Model-driven development
  - Security model transformation
- Further reading

Implementation requirements

System administrator – the individual who establishes the system security policies, performs the administrative roles and reviews the system audit trail

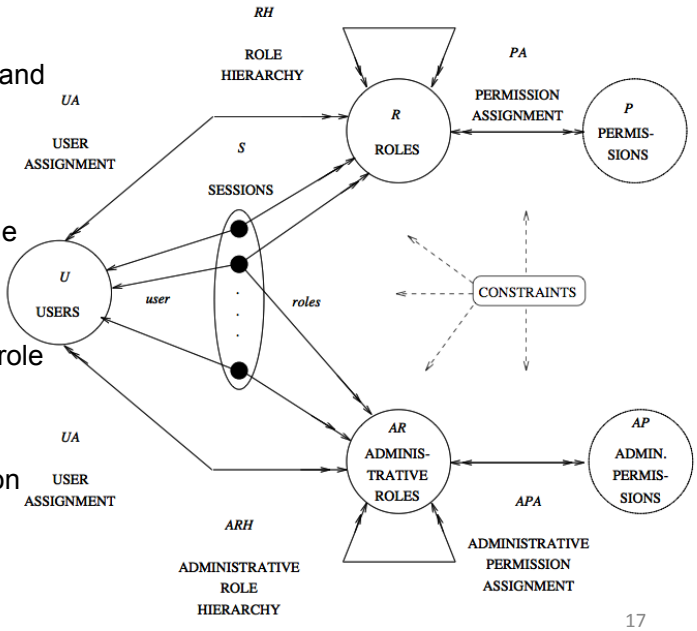
- ***Operations*** and ***Objects*** are considered predefined by the underlying system
- **Administrator**
  - manage ***Users***, ***Roles***
  - create assignment relationships
  - establish relationships between ***Roles*** and secured ***Operations*** and ***Objects***.



Sandhu and Coyne, 1996; Ferraiolo et al., 2001

Implementation requirements

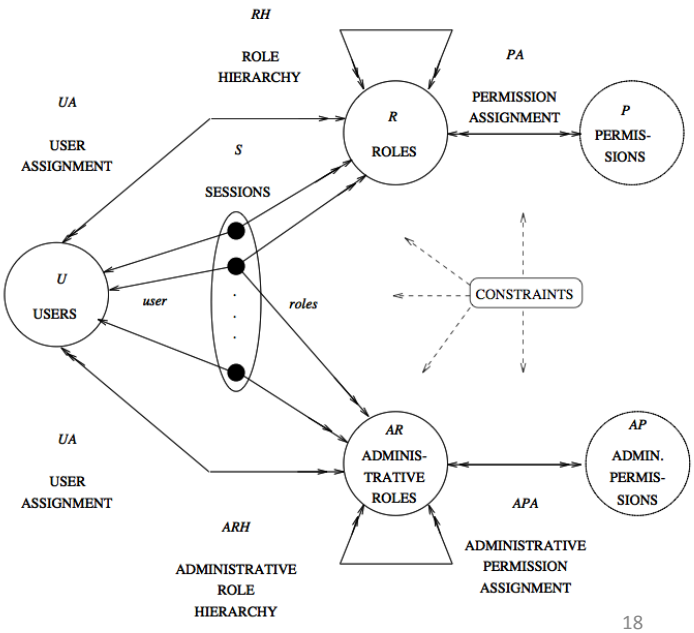
- **To activate RBAC**
  - **create session**
    - for creating a user session and assigning the user with a default set of roles
  - **add role**
    - for creating new roles for the current session
  - **drop role**
    - for deleting a role from the role set for the current session
  - **check access**
    - for determining if the session user has permission to perform the requested operation on an object



Sandhu and Coyne, 1996; Ferraiolo et al., 2001

Implementation requirements

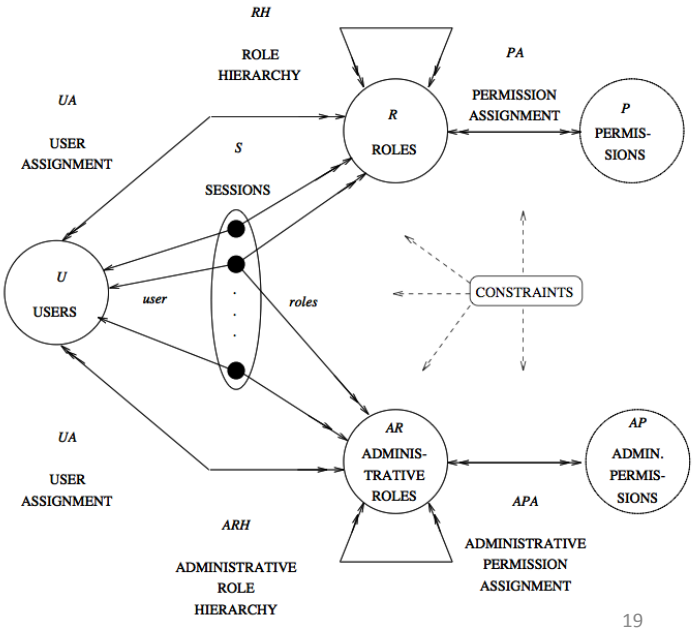
- **User Assignment** and **Permission Assignment**
  - **view assigned users**
    - for displaying a set of users assigned to a given role
  - **view assigned roles**
    - for displaying a set of roles assigned to a given user
  - **view role permissions**
    - for displaying a set of permissions granted to a given role
  - **view user permissions**
    - for displaying a set of permissions a given user gets through his or her assigned roles



Sandhu and Coyne, 1996; Ferraiolo et al., 2001

Implementation requirements

- **User Assignment** and **Permission Assignment**
  - **view session roles**
    - for displaying a set of roles associated with a session
  - **view session permissions**
    - for displaying a set of permissions available in the session
  - **view role operations on object**
    - for displaying a set of operations a given role may perform on a given object; and
  - **view user operations on object**
    - for displaying a set of operations a given user may perform on a given object

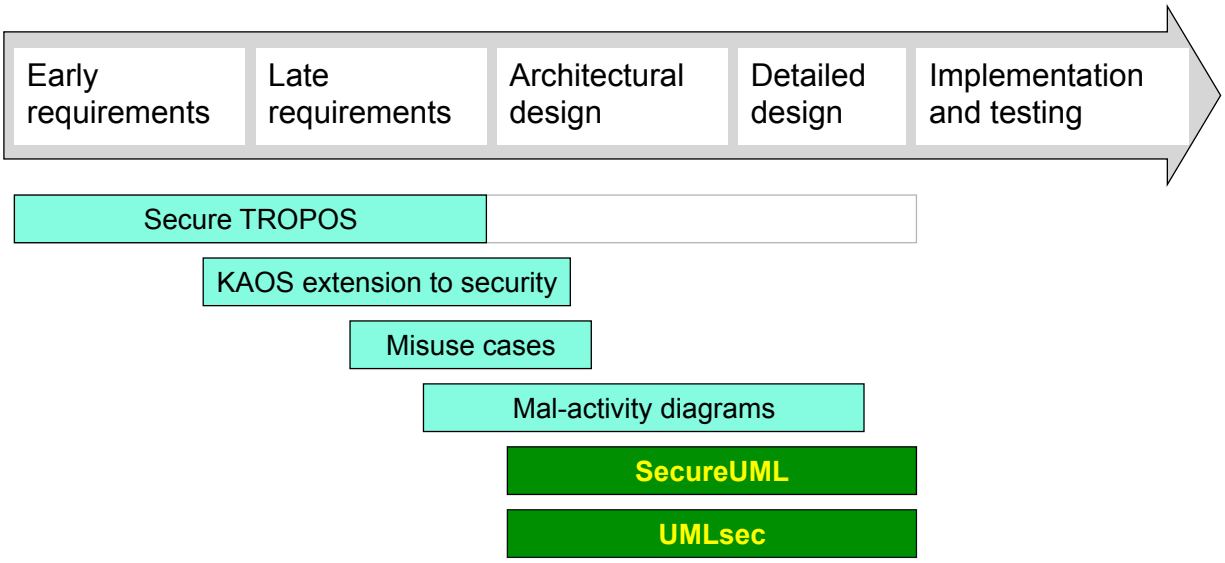


Sandhu and Coyne, 1996; Ferraiolo et al., 2001

Outline

- Principles of role-based access control
- RBAC implementation requirements
- **RBAC modelling languages**
  - SecureUML
  - UMLsec
  - Language comparison
  - Transformation
- Model-driven security
  - Model-driven development
  - Security model transformation
- Further reading

Security Modelling Languages

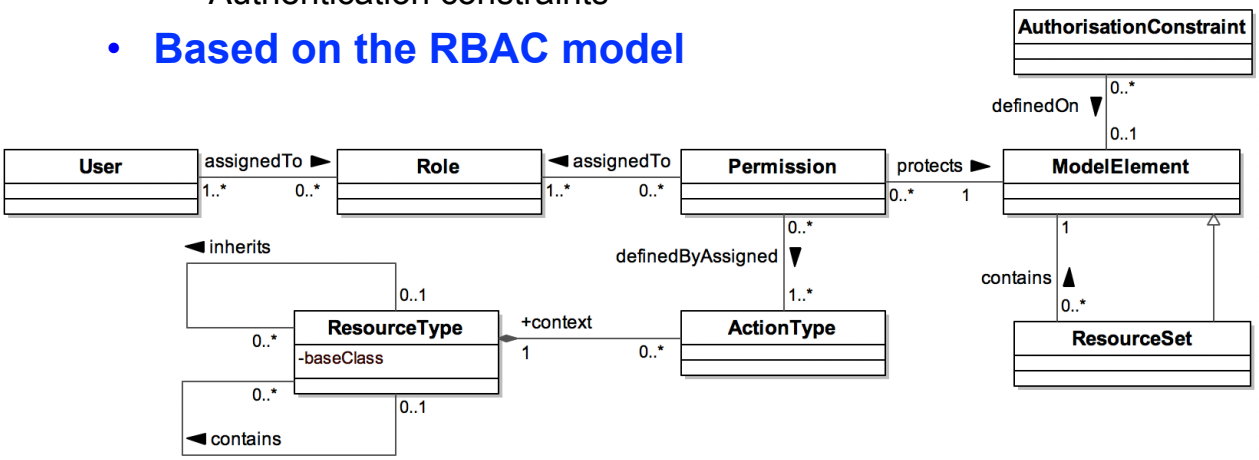


Outline

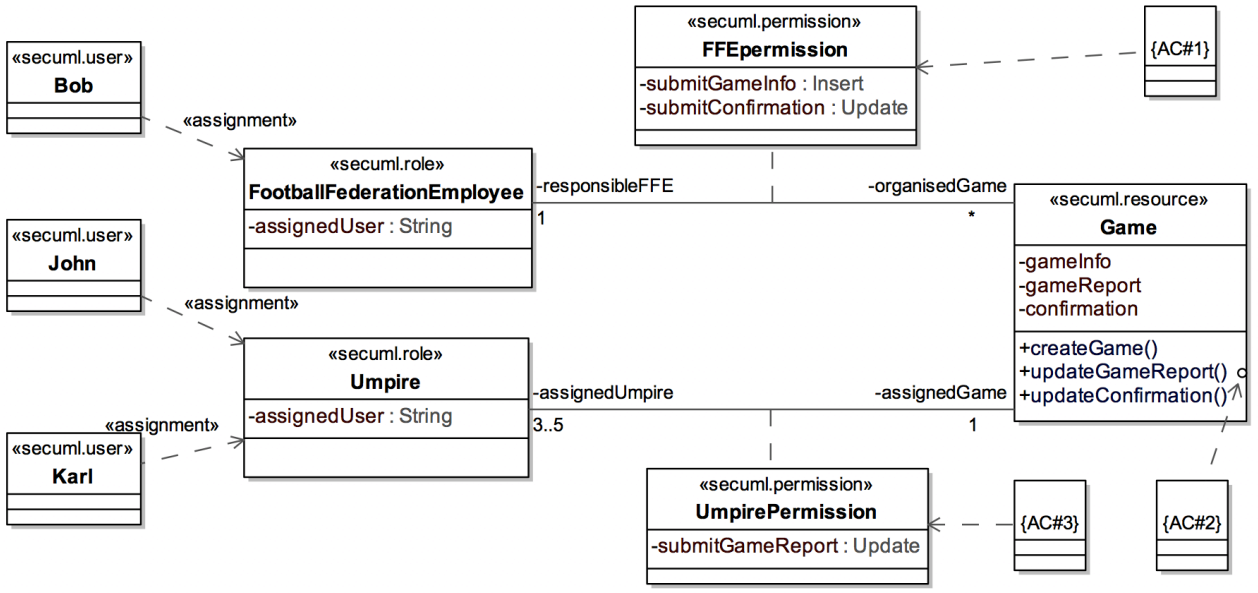
- Principles of role-based access control
- RBAC implementation requirements
- RBAC modelling languages
  - **SecureUML**
  - UMLsec
  - Language comparison
  - Transformation
- Model-driven security
  - Model-driven development
  - Security model transformation
- Further reading

SecureUML

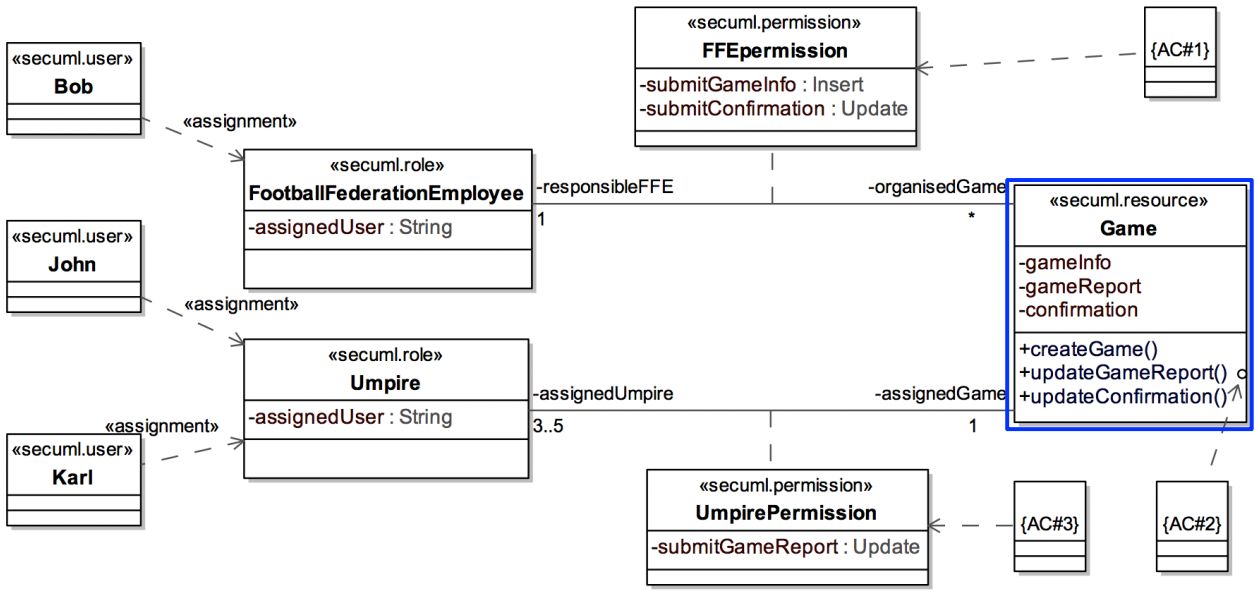
- **Extension of the UML class diagrams**
  - Stereotypes
  - Tagged values
  - Authentication constraints
- **Based on the RBAC model**



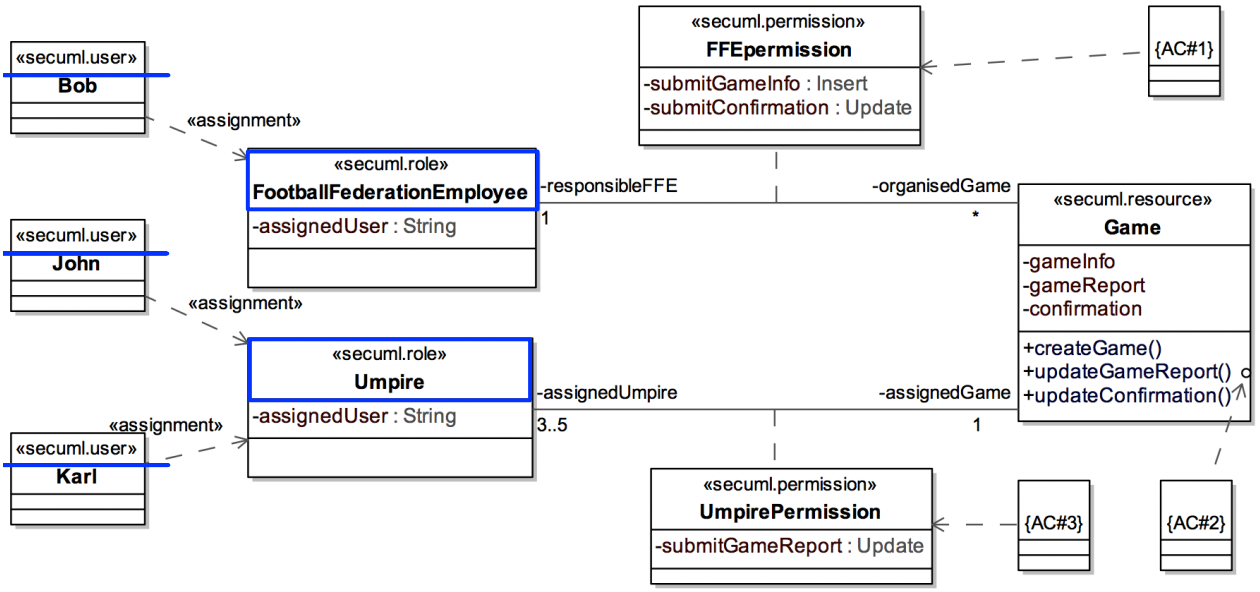
SecureUML



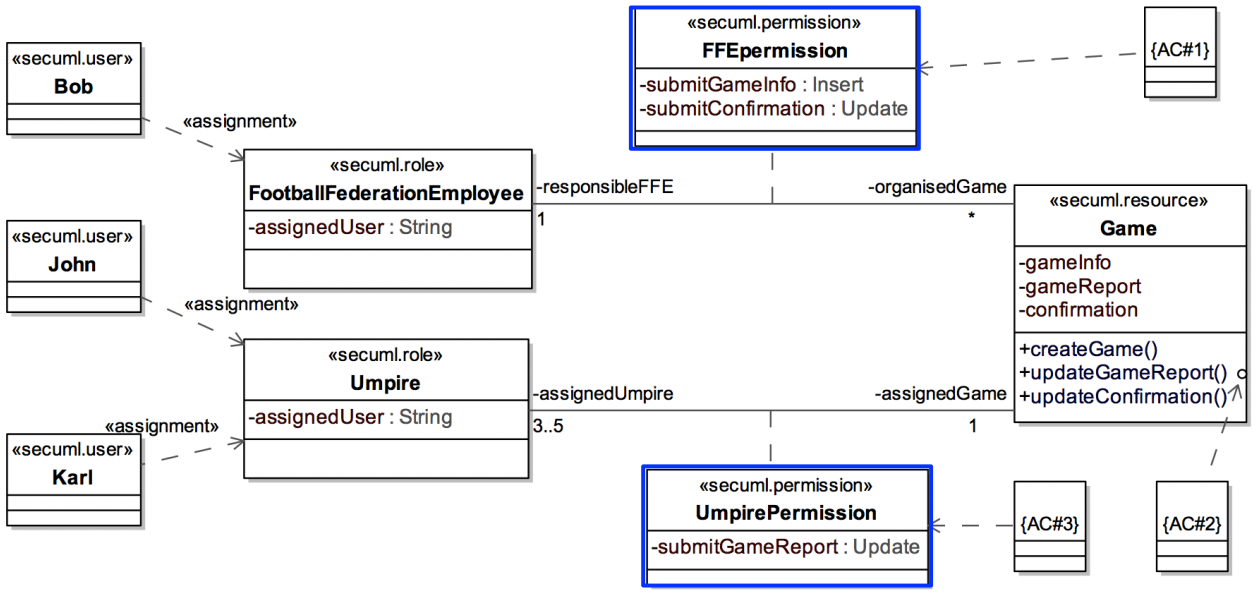
SecureUML



SecureUML

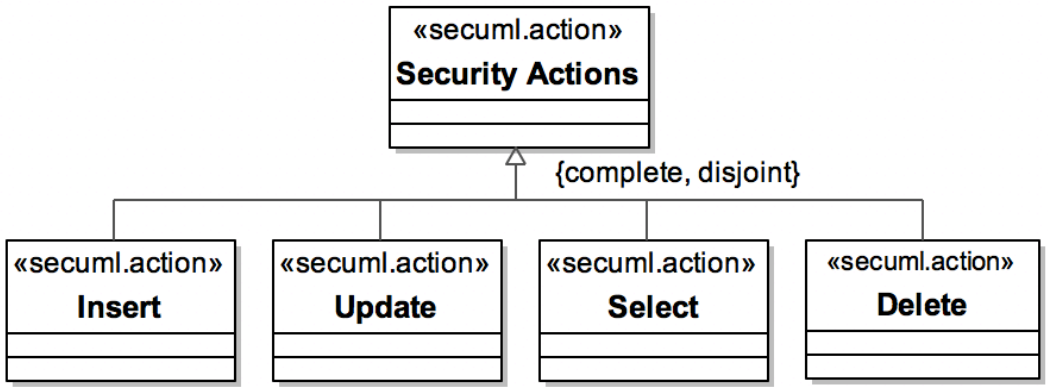


SecureUML



Access Rules

- **Security actions**



Authorisation Constraints

AC#1:

```
context Game::createGame():void
    pre: self.responsibleFFE.assignedUser ->
    exists(i | i.assignedUser = 'Bob')
```

AC#2:

```
context Game::updateConfirmation():void
    pre: self.responsibleFFE.assignedUser ->
    exists(i | i.assignedUser = 'Bob')
```


Outline

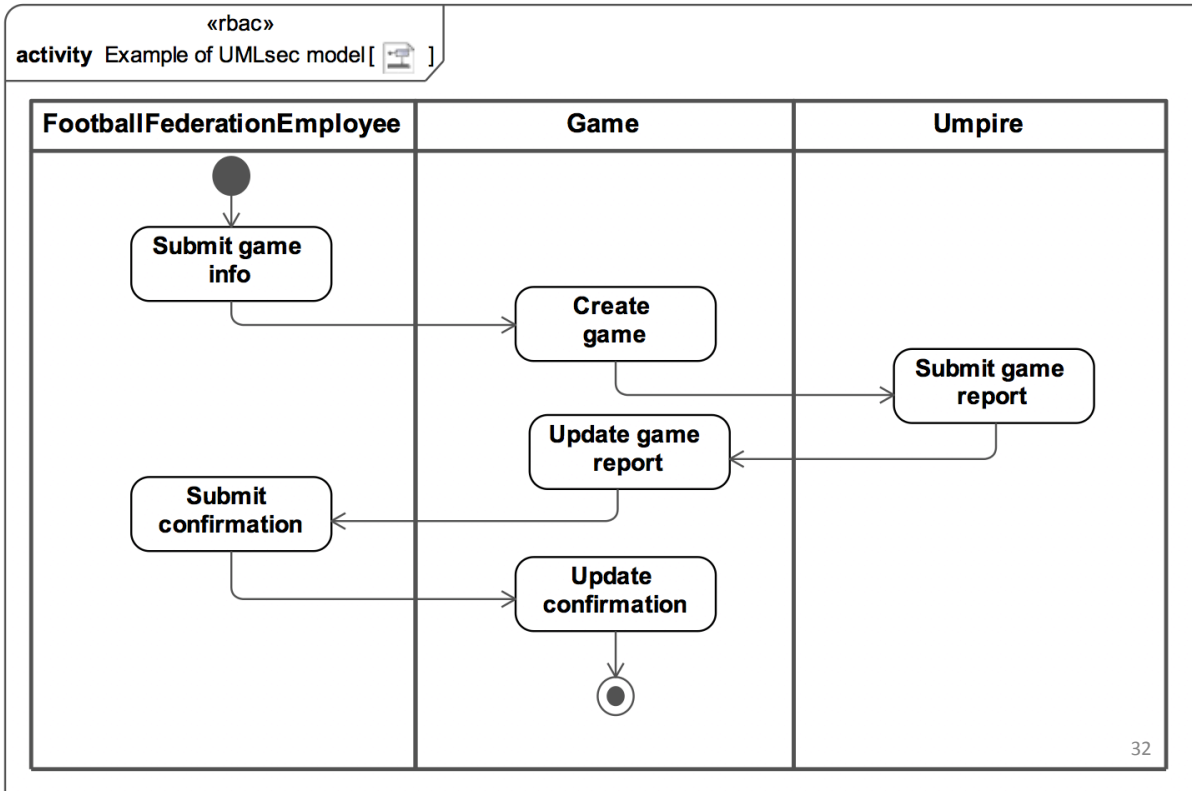
- Principles of role-based access control
- RBAC implementation requirements
- RBAC modelling languages
  - SecureUML
  - **UMLsec**
  - Language comparison
  - Transformation
- Model-driven security
  - Model-driven development
  - Security model transformation
- Further reading

UMLsec

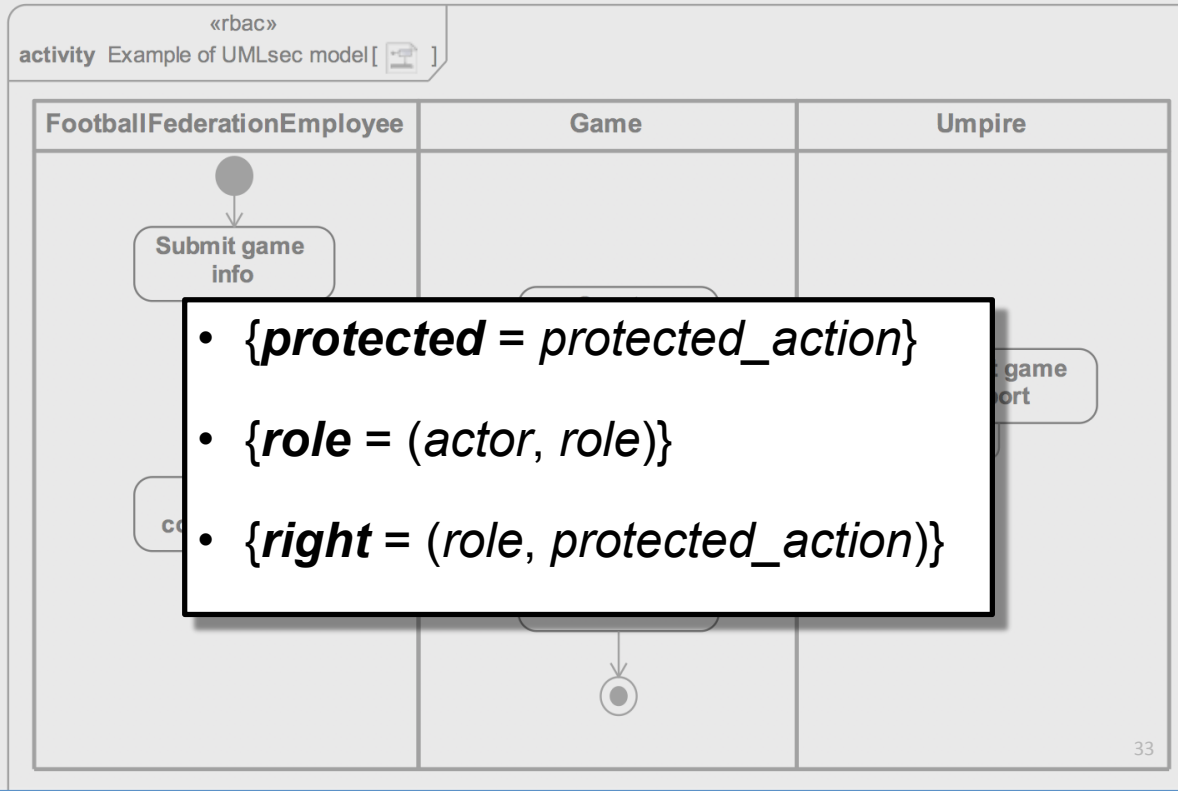
- Extension of the UML diagrams:
  - Stereotypes;
  - Tagged values;
  - Authentication constraints

Stereotype	Base class	Tags	Constraints	Description
fair exchange	subsystem	start, stop, adversary	after start eventually reach stop	Enforce fair exchange
smart card	node	-	-	smart card node
data security	subsystem	adversary, integrity, authenticity	Provides secrecy, integrity, authenticity, freshness	Basic data security constraints
rbac	***subsystem***	***protected, role, right***	***only permitted activities executed***	***enforces RBAC***

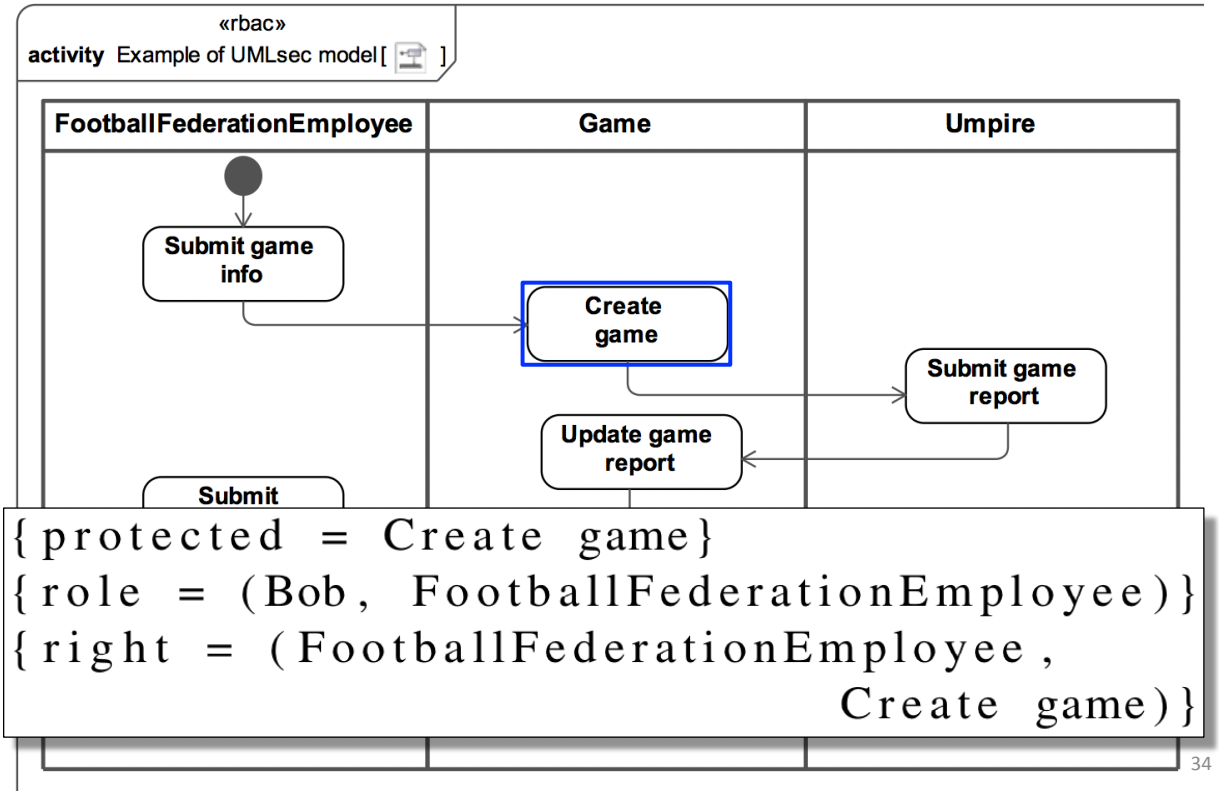
UMLsec



UMLsec



UMLsec



Outline

- Principles of role-based access control
- RBAC implementation requirements
- RBAC modelling languages
  - SecureUML
  - UMLsec
  - **Language comparison**
  - Transformation
- Model-driven security
  - Model-driven development
  - Security model transformation
- Further reading

Language comparison

Extension Mechanism

Criteria	SecureUML	UMLsec
Meta-model	Explicit based on the RBAC model	Not explicit as the UML profile extension
UML profile	Mainly *class* diagram	The whole UML profile i.e., *use cases*, *class*, *activity*, *state*, *component*, and other diagrams
Extension	Stereotypes, tagged values and authentication constraints	Stereotypes, tagged values and constraints
Constraints	OCL	Constraint language is not identified

Language comparison

Modelling Targets and Application Guidelines

Criteria	SecureUML	UMLsec
Security criteria	Not identified	Confidentiality, integrity (and derived ones, like authenticity and others)
Security requirements / controls	RBAC	RBAC but also non-repudiations, secure communication links, secrecy and integrity, authenticity, freshness, secure information flows, guard access
Method	Development of the RBAC models	Not explicit but implicitly supports standard security management methods

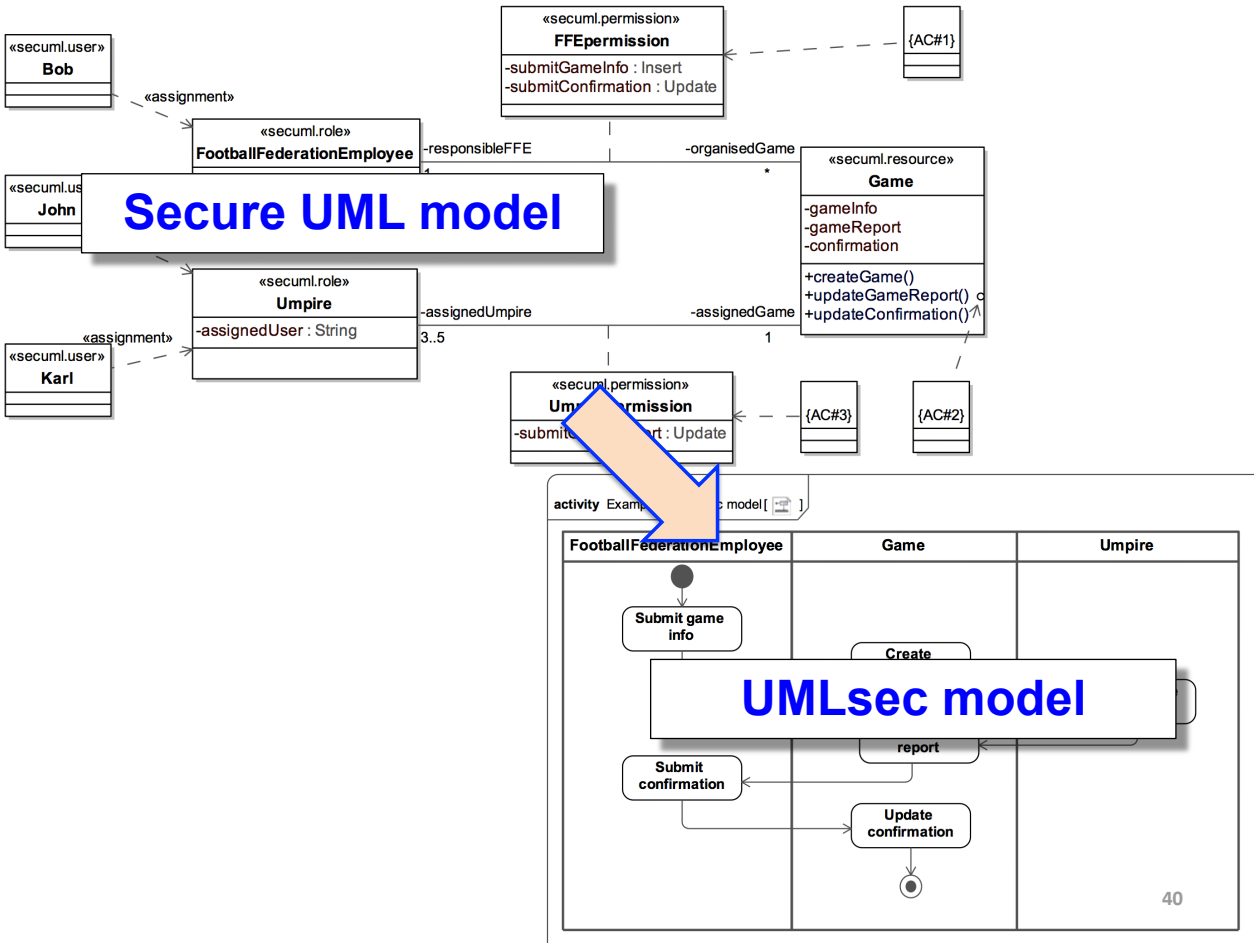
Language comparison

Construct Semantics

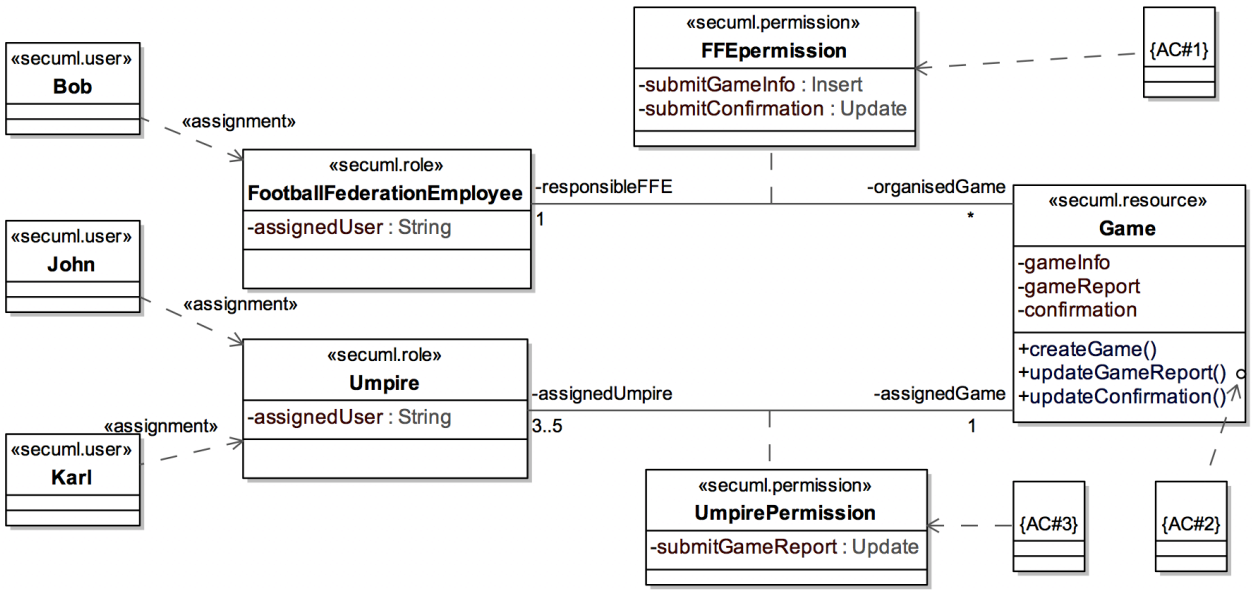
RBAC concepts	SecureUML	UMLsec
User	Class stereotype «***securml.user***»	*Actor* value of association tag {***role***}
User assignment	Dependency stereotype «***assignment***»	Associated tag {***role***}
Roles	Class stereotype «***securml.role***»	➢ *Activity partition* ➢ *Role* value of association tag {***role***}
Permission assignment	Association class stereotype «***securml.permission***»	➢ *Action* ➢ Associated tag {***right***}
Object	Class stereotype «***securml.resource***»	*Activity partition*
Operation	Operation of «***securml.resource***» class	➢ *Action* ➢ Associated tag {***protected***}
Permission	Authorisation constraints	Not defined

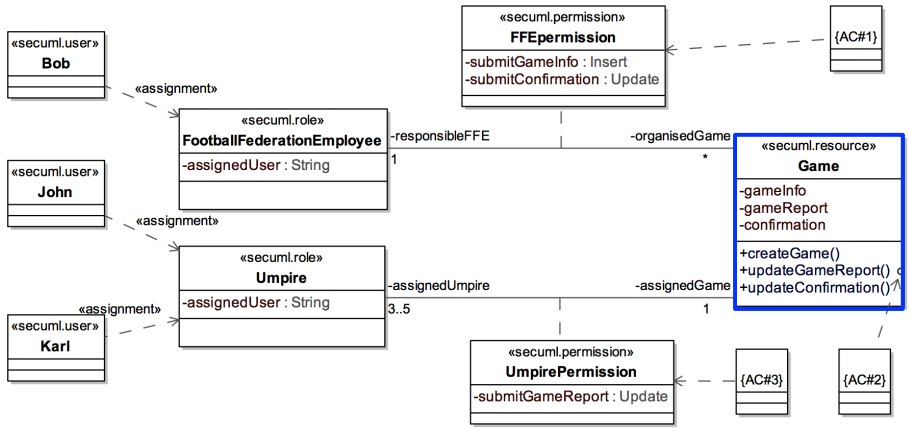
Outline

- Principles of role-based access control
- RBAC implementation requirements
- RBAC modelling languages
  - SecureUML
  - UMLsec
  - Language comparison
  - **Transformation**
- Model-driven security
  - Model-driven development
  - Security model transformation
- Further reading

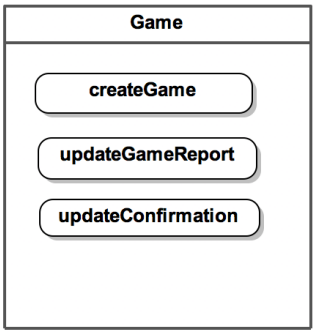


SecureUML model





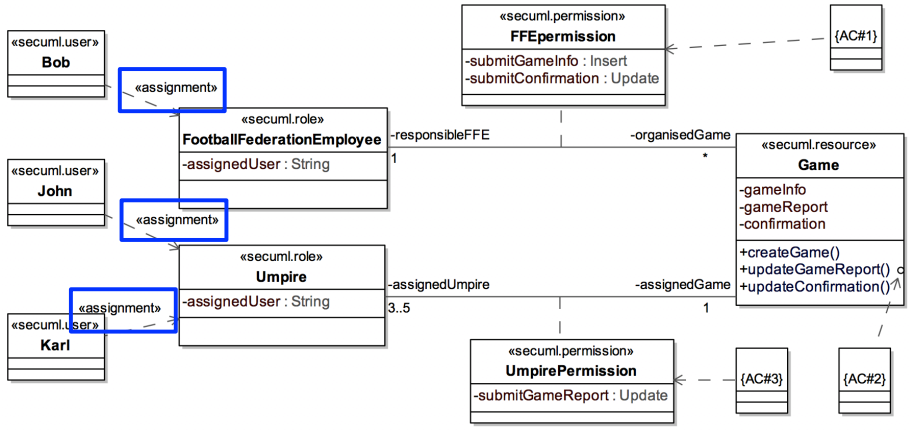
- **SU.1**: A class with a stereotype <<***securml.resource***>> is transformed to an activity partition in the UMLsec model
- *Operations* of this class become *actions* belonging to this partition
  - each operation becomes a value the UMLsec associated tag {***protected***}



{**protected** = (*createGame*)}

{**protected** = (*updateGameReport*)}

{**protected** = (*updateConfirmation*)}

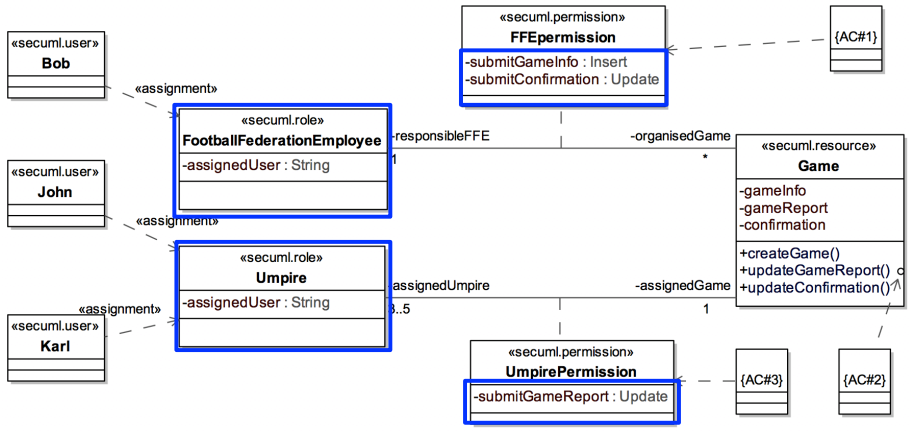


- **SU.2**: A relationship with a stereotype <<***assignment***>> relationship used to connect users and their roles is transformed to an associated tag {***role***}

{**role** = (*Bob, FootballFederationEmployee*)}

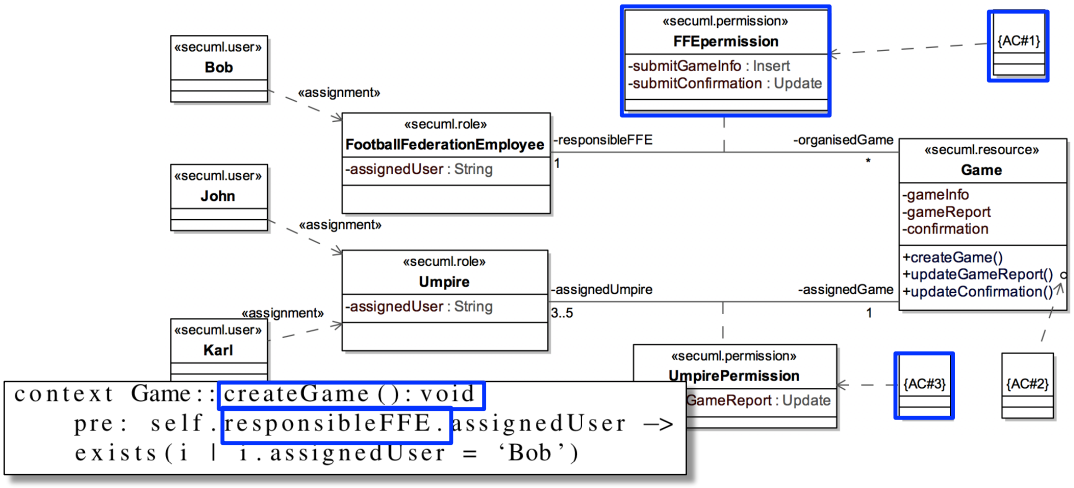
{**role** = (*John, Umpire*)}

{**role** = (*Karl, Umpire*)}



- **SU.3**: A class with the stereotype <<***securml.roles***>> is transformed to the UMLsec activity partition
  - The attributes of an association class that connects the <<***securml.roles***>> class with <<***securml.resource***>> class, become actions in the corresponding activity partition

FootballFederationEmployee	Game	Umpire
submitGameInfo	createGame	submitGameReport
	updateGameReport	
submitConfirmation	updateConfirmation	

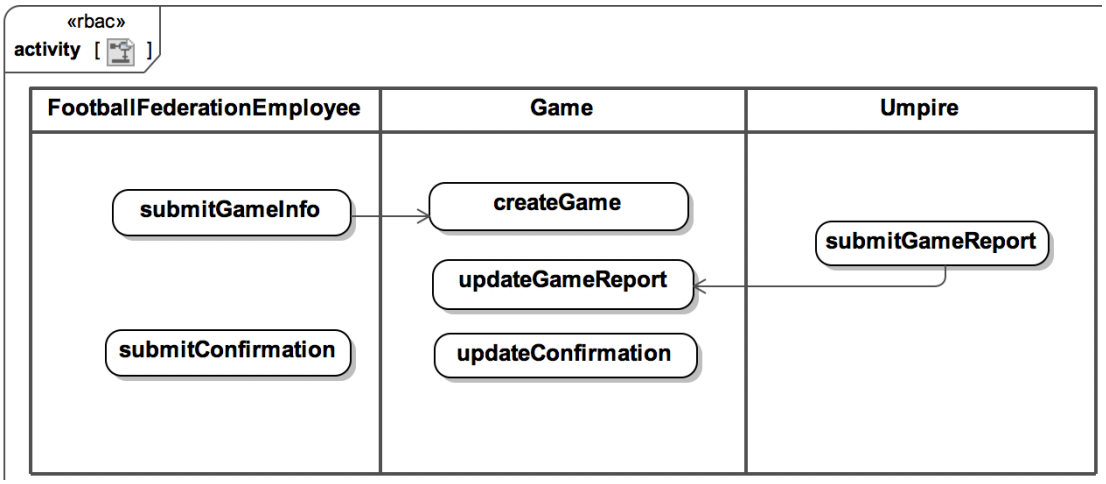


- **SU.4**: The association class with the stereotype <<***securml.permission***>> defines the *role* value for the associated tag {***right***}
  - The value of *right* can be determined from the authorisation constraint defined for the attribute of the SecureUML association class.

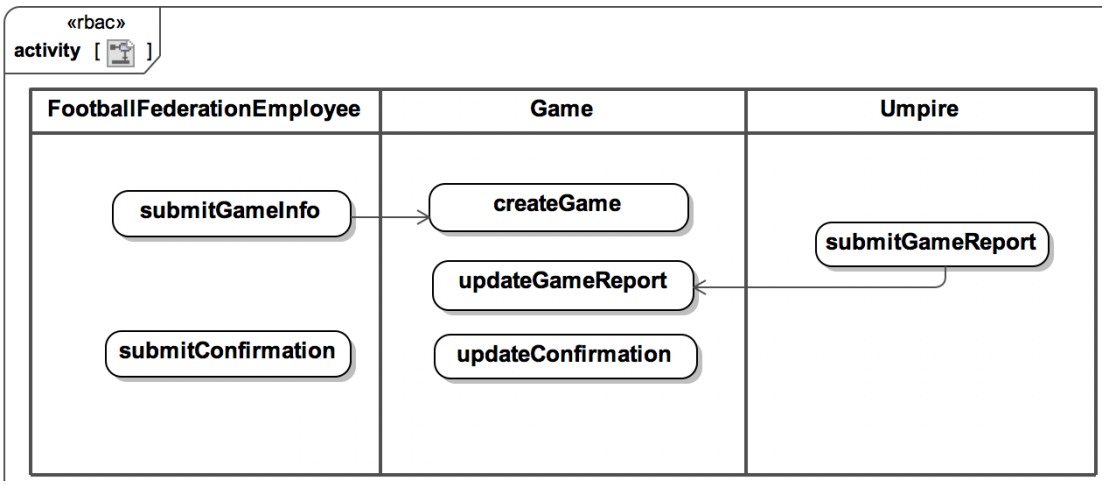
{**right** = (*FootballFederationEmployee*, *createGame*)}

{**right** = (*Umpire*, *updateGameReport*)}

SU.5: Received activity diagram is annotated with the <<***rbac***>> stereotype



UMLsec model

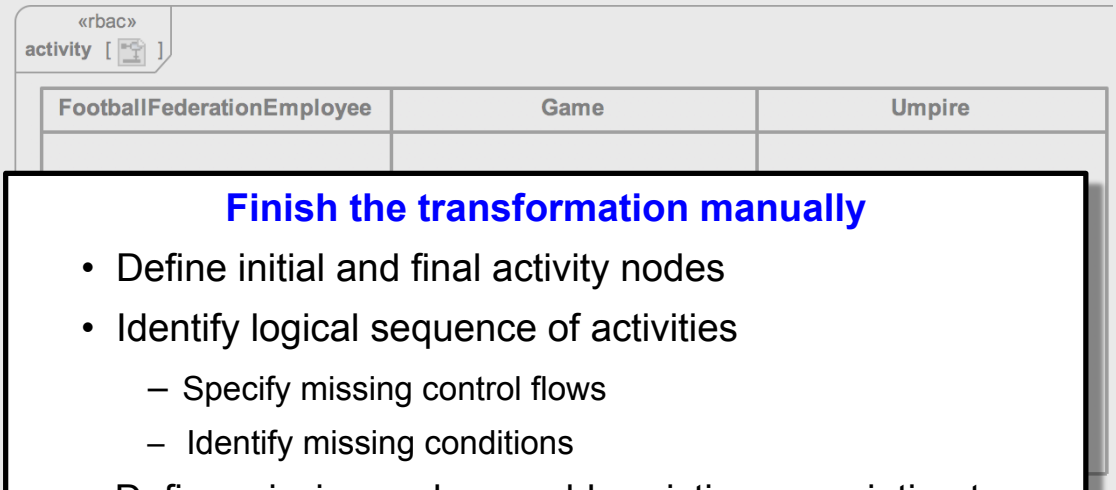


{**protected** = (*createGame*)}
{**protected** = (*updateGameReport*)}
{**protected** = (*updateConfirmation*)}

{**right** = (*FootballFederationEmployee*, *createGame*)}
{**right** = (*Umpire*, *updateGameReport*)}

{**role** = (*Bob*, *FootballFederationEmployee*)}
{**role** = (*Karl*, *Umpire*)}
{**role** = (*John*, *Umpire*)}

UMLsec model



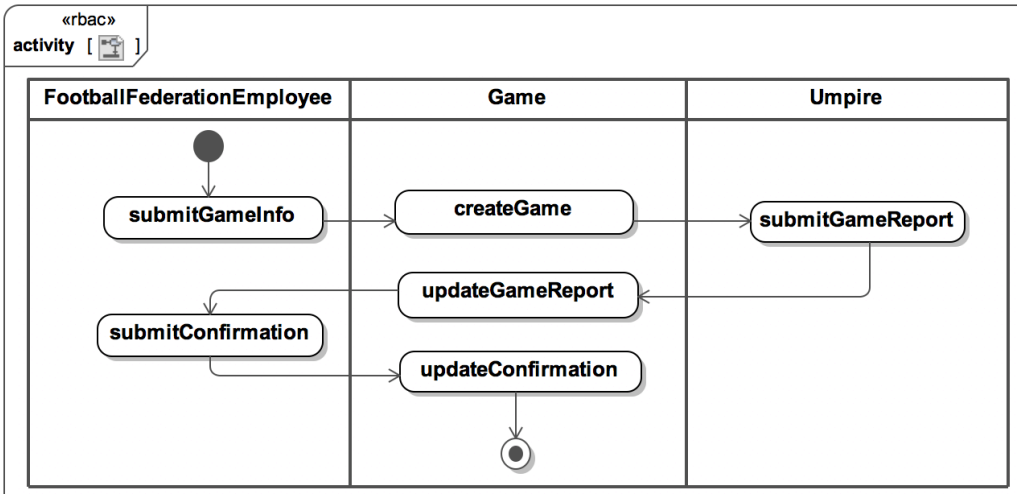
Finish the transformation manually

- Define initial and final activity nodes
- Identify logical sequence of activities
  - Specify missing control flows
  - Identify missing conditions
- Define missing and assembly existing association tags

{**protected** = (*updateConfirmation*)}

{**role** = (*Bob*, *FootballFederationEmployee*)}
{**role** = (*Karl*, *Umpire*)}
{**role** = (*John*, *Umpire*)}

UMLsec model

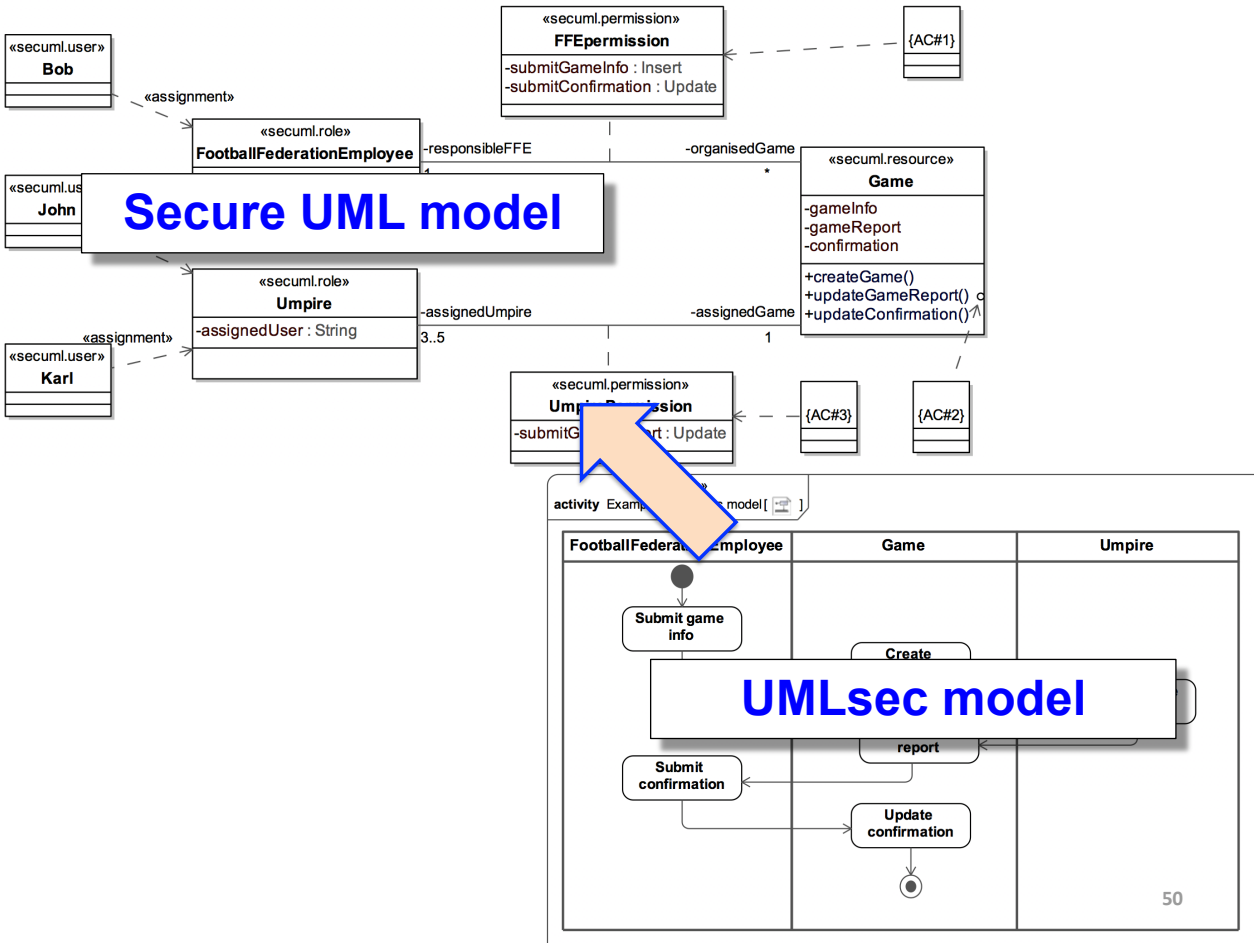


{**protected** = (*createGame*)}
{**right** = (*FootballFederationEmployee*, *createGame*)}
{**role** = (*Bob, FootballFederationEmployee*)}

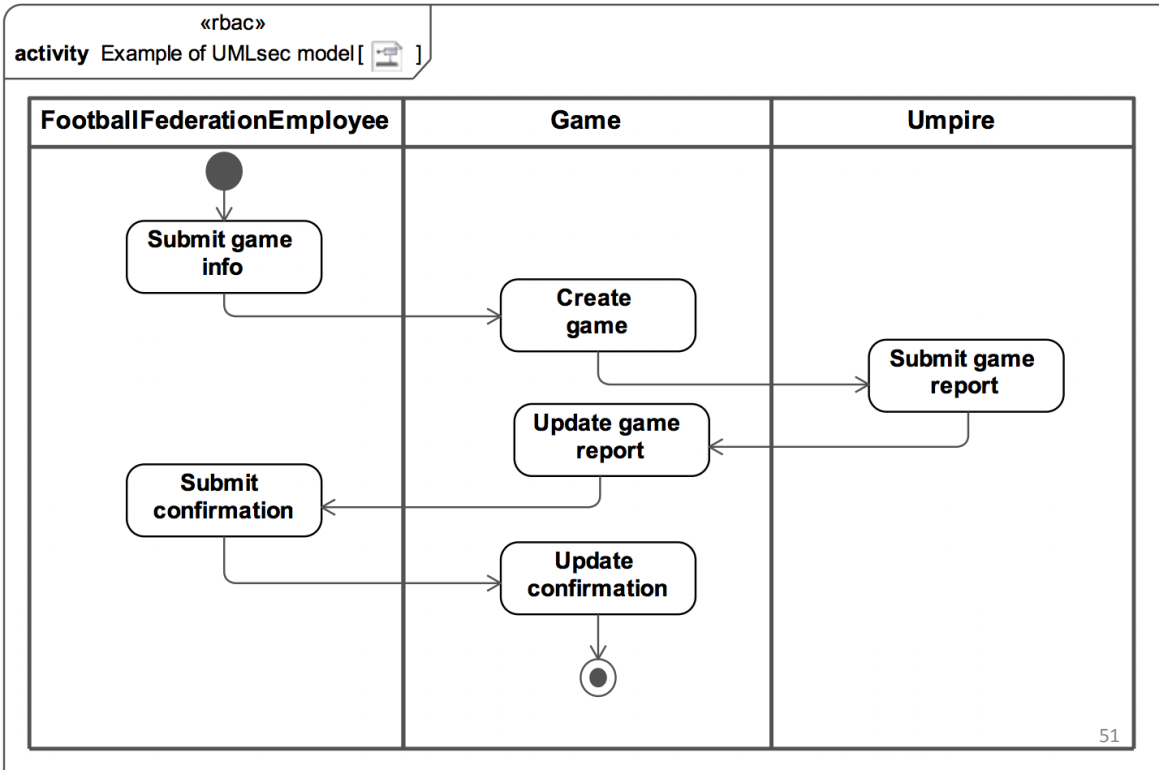
{**protected** =(*updateConfirmationt*)}
{**right** = (*FootballFederationEmployee*, *updateConfirmation*)}
{**role** = (*Bob, FootballFederationEmployee*)}

{**protected** = (*updateGameReport*)}
{**right** = (*Umpire*, *updateGameReport*)}
{**role** = (*John, Umpire*)}

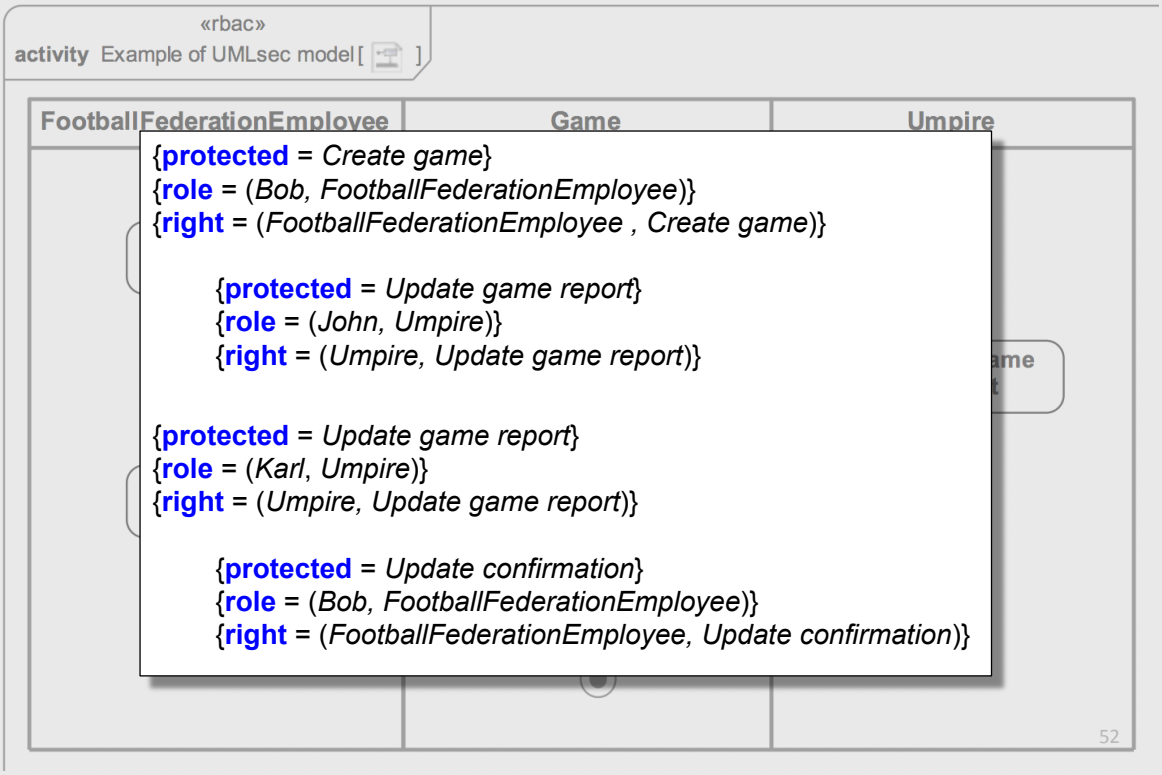
{**protected** = (*updateGameReport*)}
{**right** = (*Umpire*, *updateGameReport*)}
{**role** = (*Karl, Umpire*)}

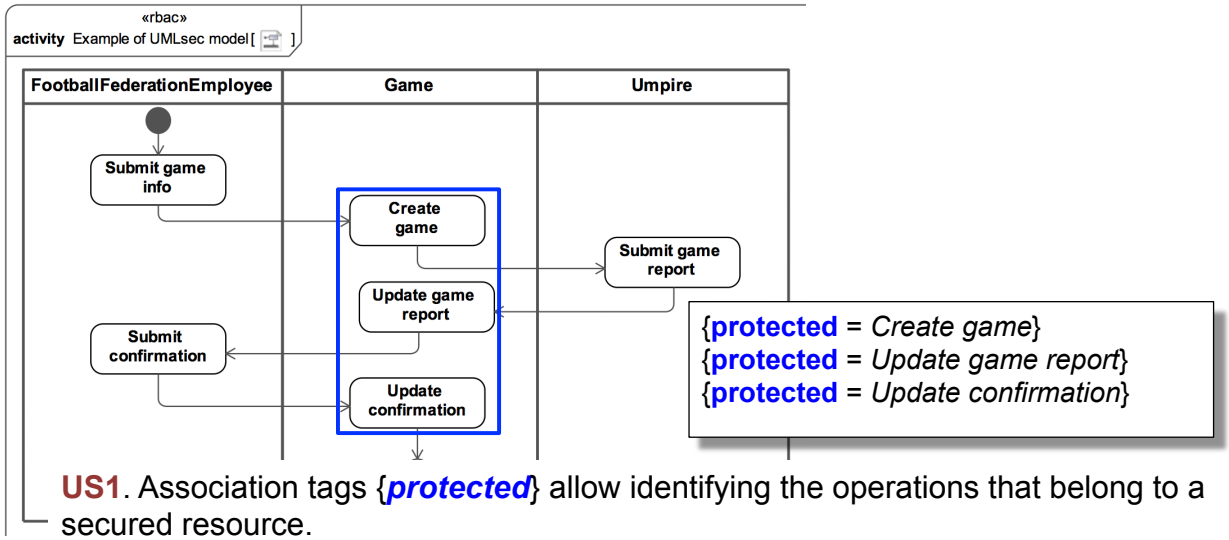


UMLsec model



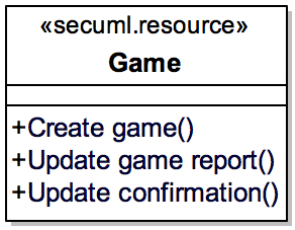
UMLsec model

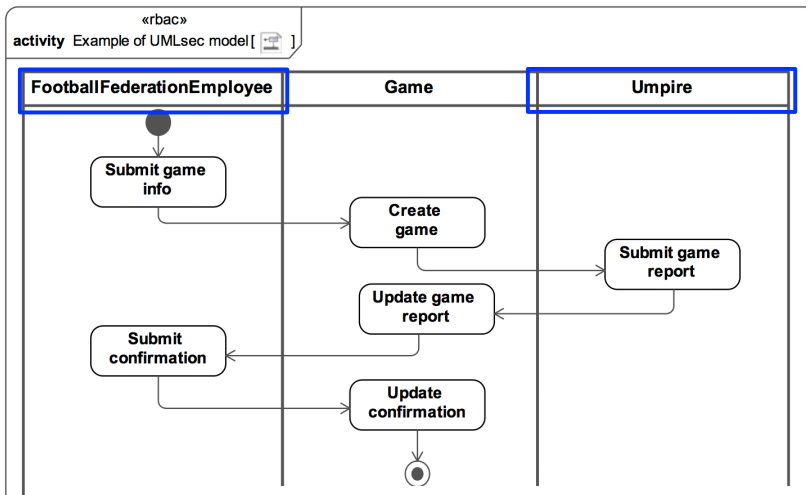




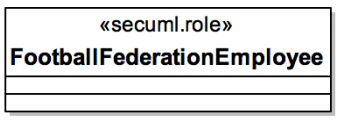
US1. Association tags {***protected***} allow identifying the operations that belong to a secured resource.

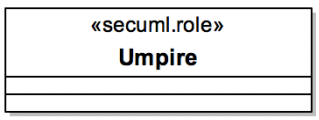
- The activity partitions which hold these operations are transformed to the SecureUML class with a stereotype «***securml.resource***».

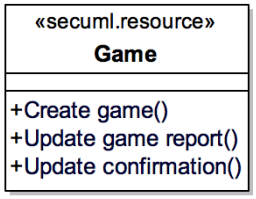


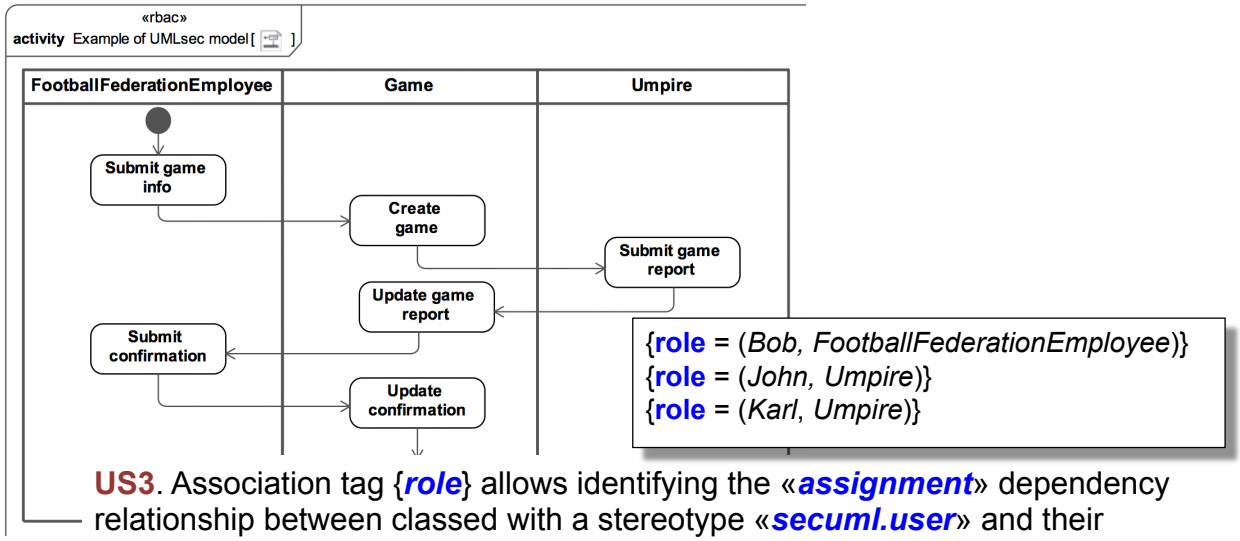


US2. The UMLsec activity partitions which do not hold secured protected actions can be transformed to «***securml.role***» stereotyped classes.

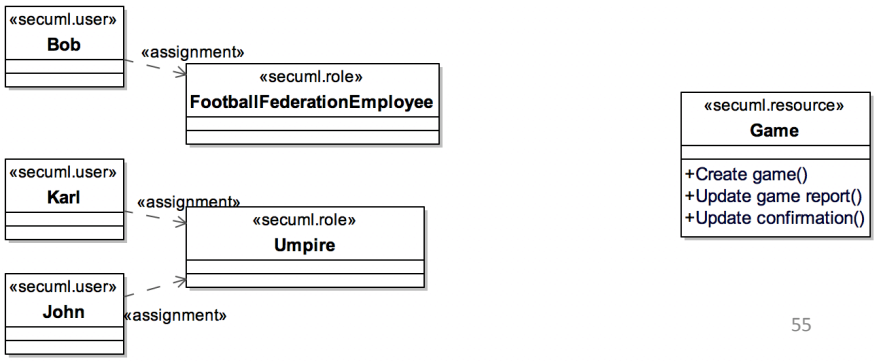


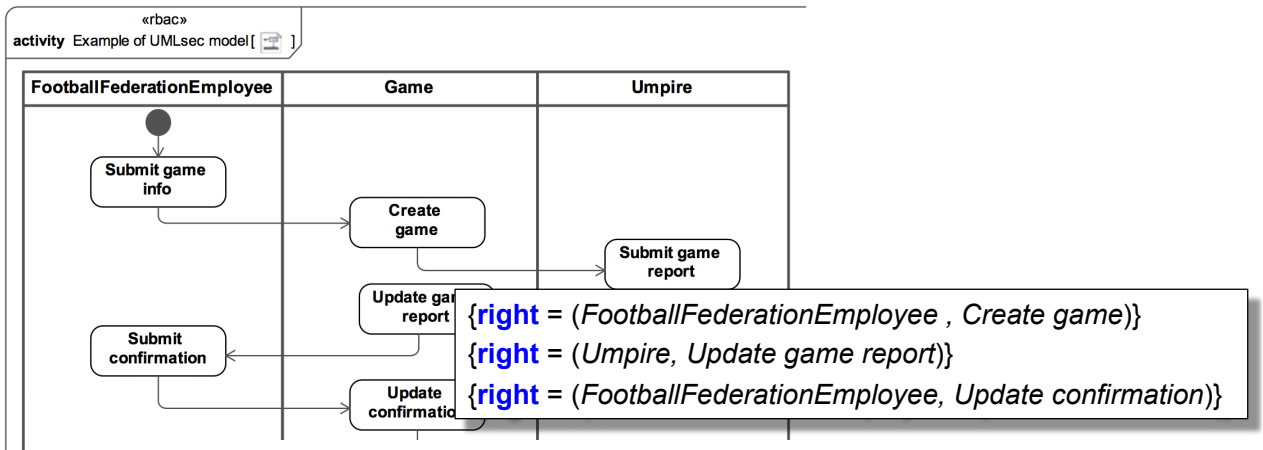




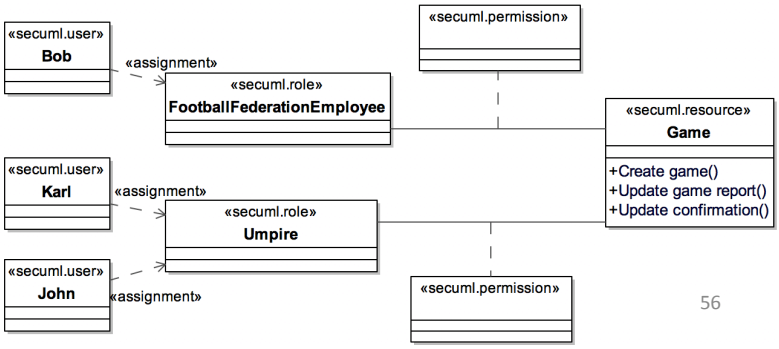


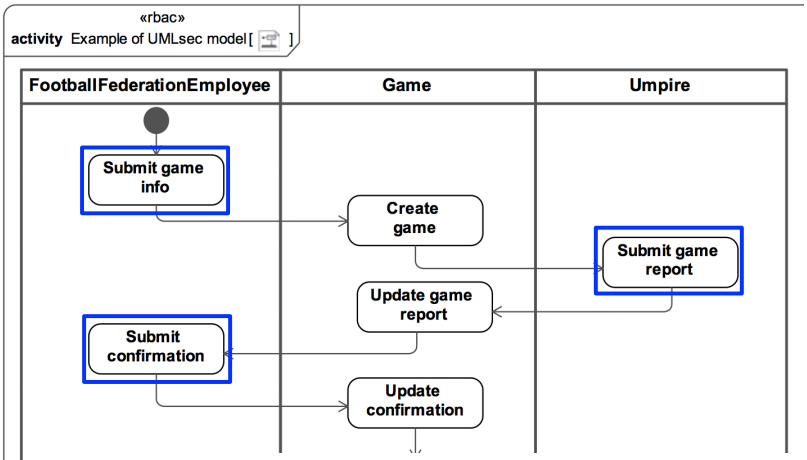
US3. Association tag {***role***} allows identifying the «***assignment***» dependency relationship between classed with a stereotype «***securml.user***» and their «***securml.role***» stereotypes



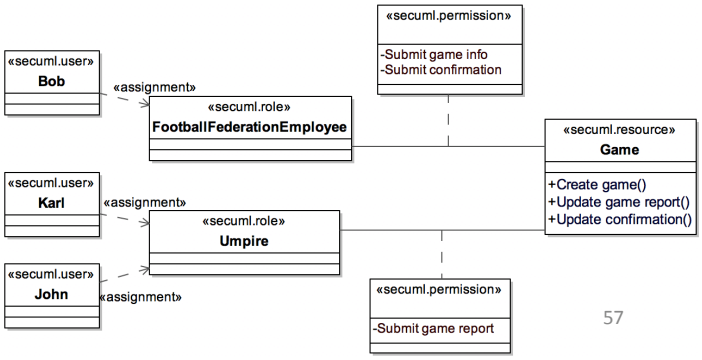


US4. From UMLsec association tag {***right***} we are able to identify on which operations the role can perform security actions. Thus, from each occurrence of this association tag in the SecureUML model, a corresponding association class between a «***umlsec.role***» and a «***umlsec.resource***» is introduced

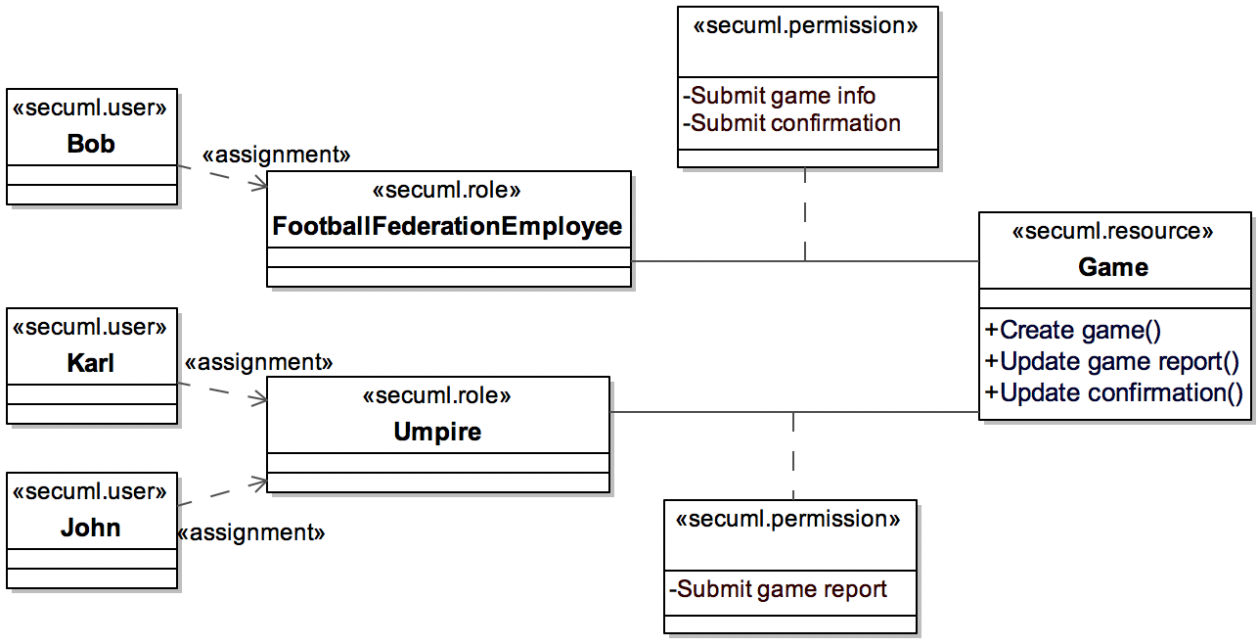




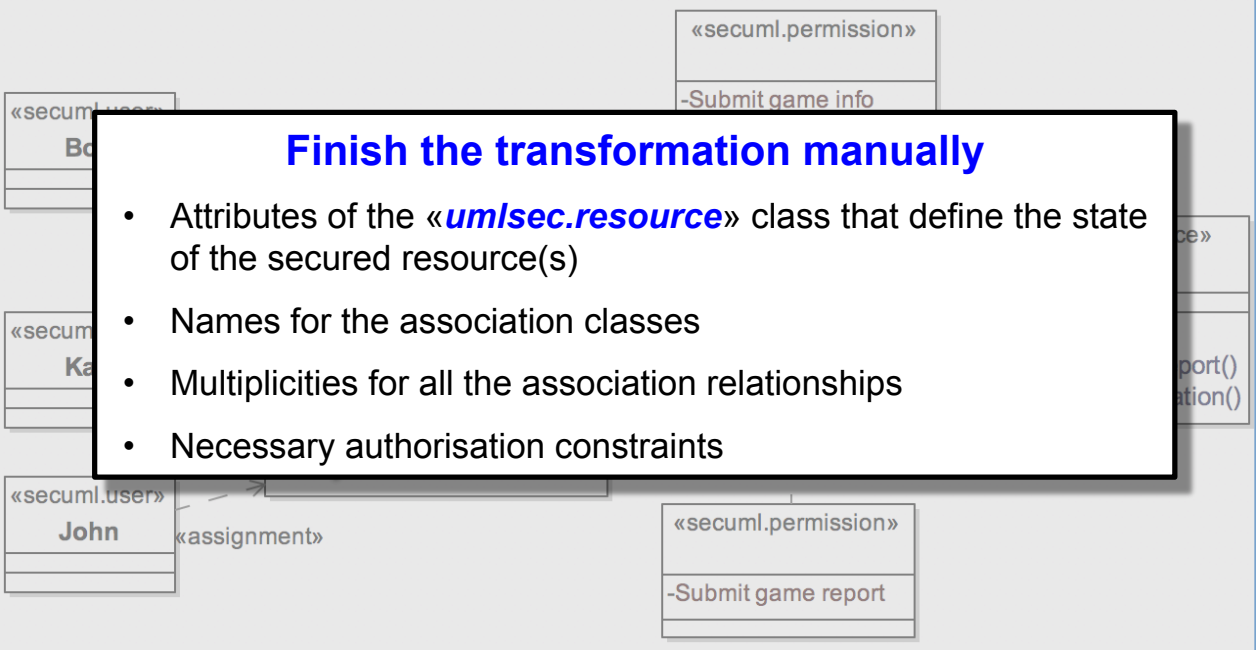
US5. In the UMLsec activity diagram it is possible to identify the security actions that are carried towards the secured operations: these are unprotected actions performed before the protected ones.



SecureUML model



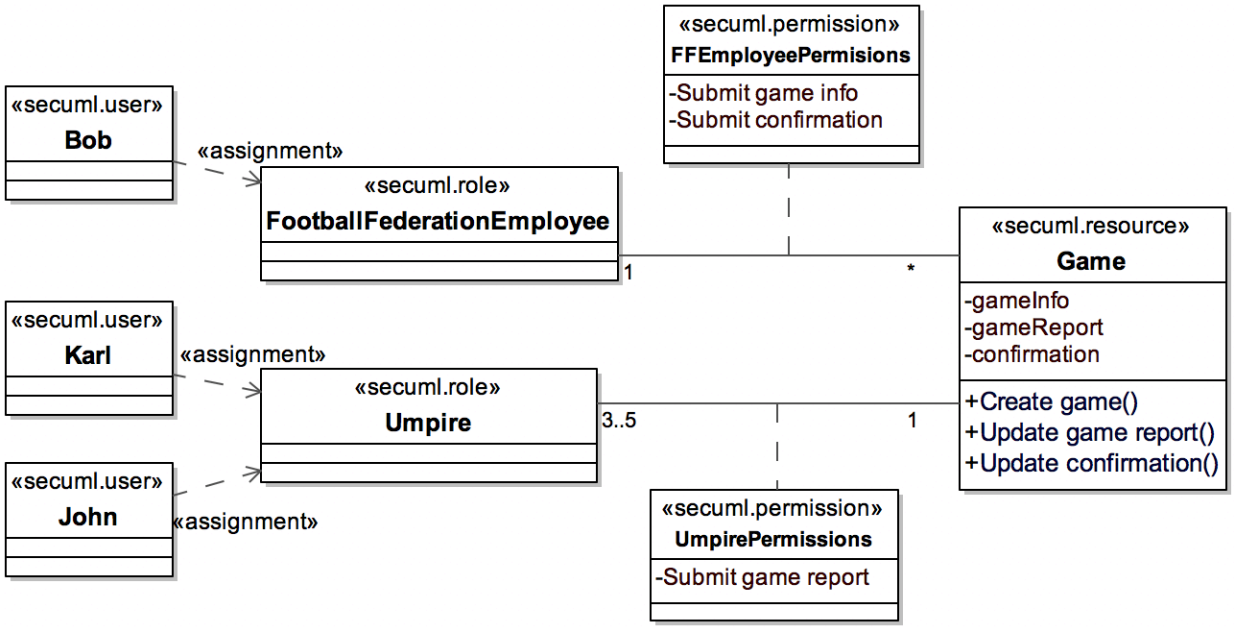
SecureUML model

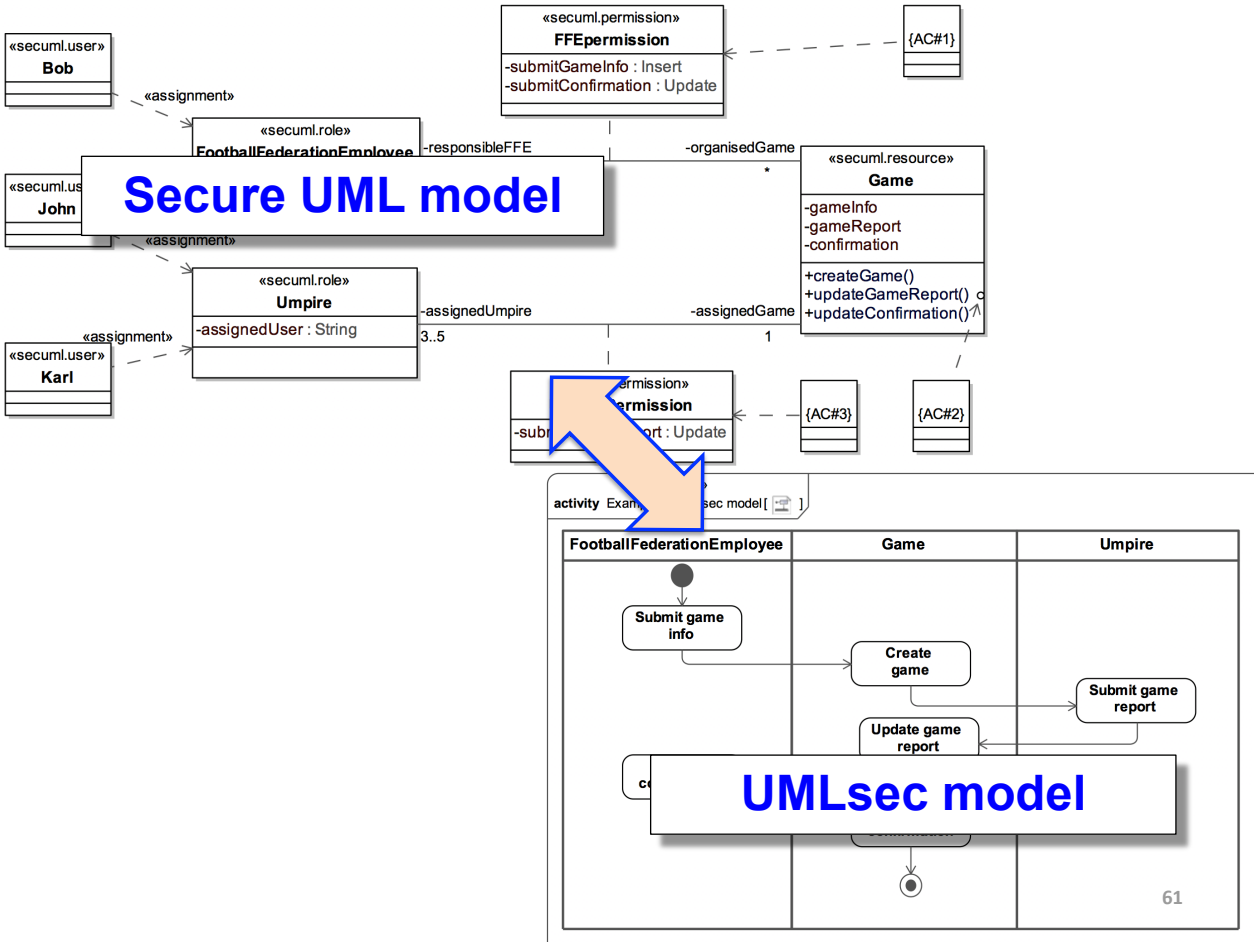


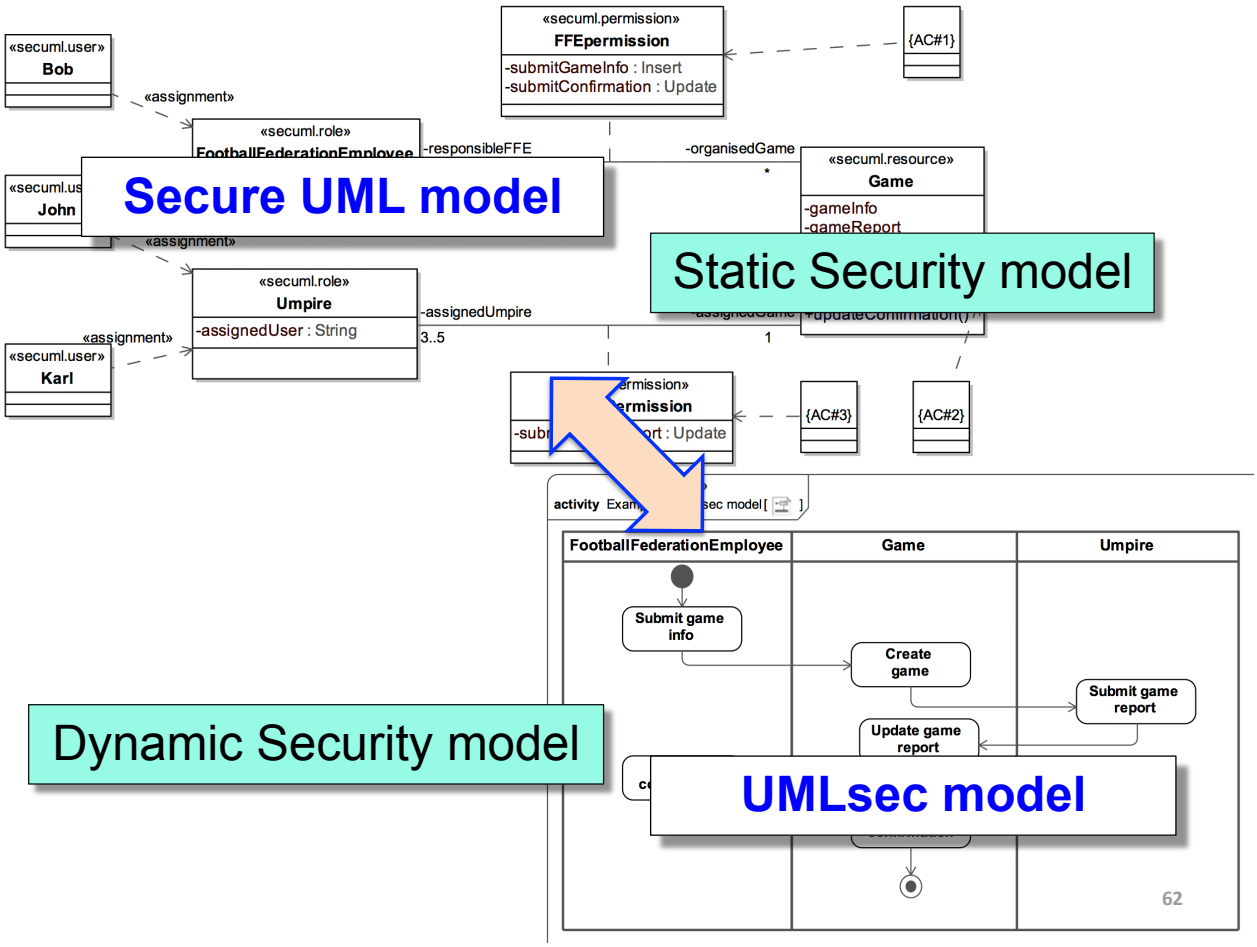
Finish the transformation manually

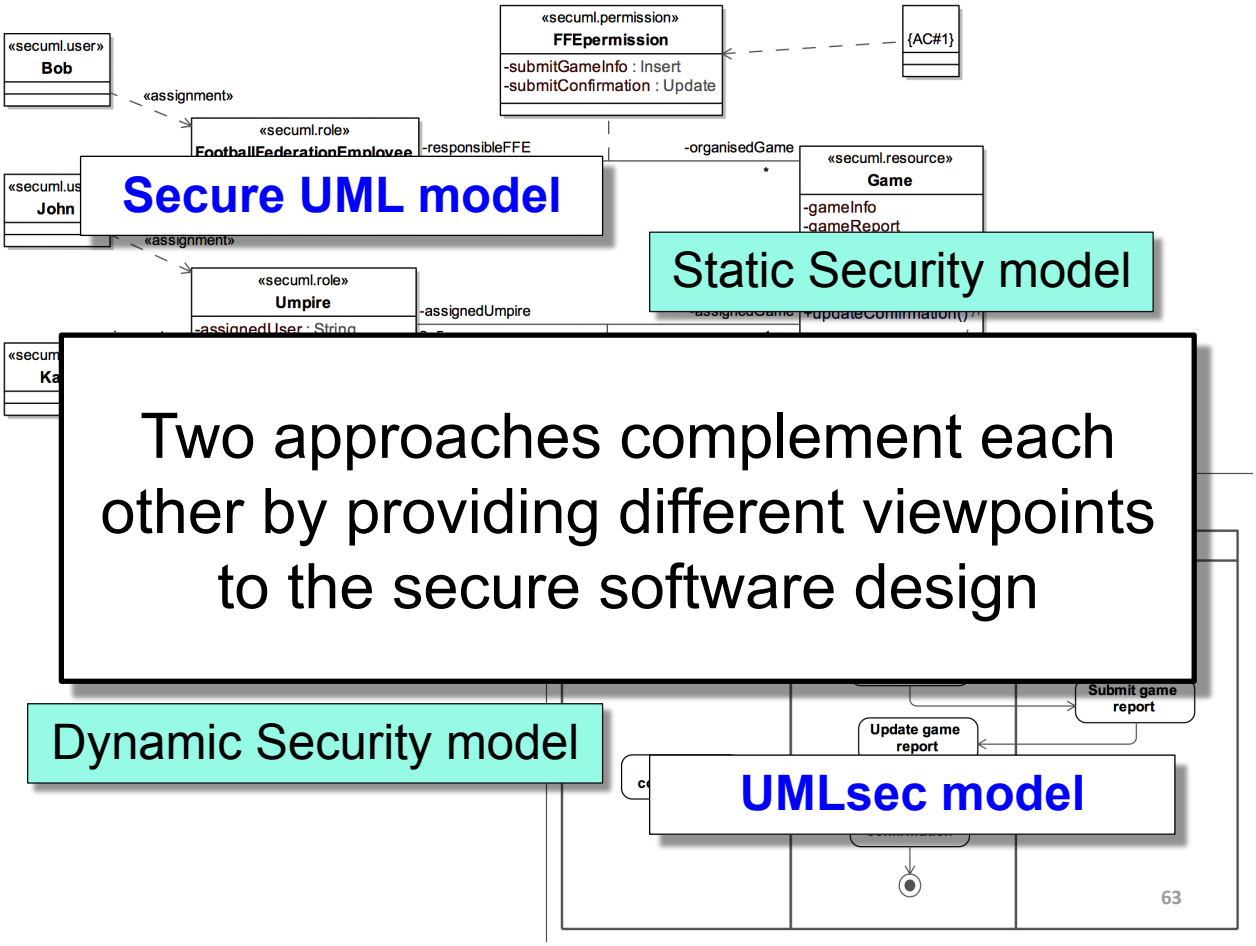
- Attributes of the «***umlsec.resource***» class that define the state of the secured resource(s)
- Names for the association classes
- Multiplicities for all the association relationships
- Necessary authorisation constraints

SecureUML model




«securml.permission»
FFEpermission
-submitGameInfo : Insert
-submitConfirmation : Update
{AC#1}
«securml.user»
Bob
«assignment»
«securml.role»
-responsibleFFE
-organisedGame
«securml.resource»
Game
-gameInfo
-gameReport
-confirmation
+createGame()
+updateGameReport()
+updateConfirmation()
Secure UML model
«securml.role»
Umpire
-assignedUser : String
-assignedUmpire
3..5
-assignedGame
1
«securml.user»
Karl
{AC#3}
{AC#2}
activity
FootballFederationEmployee
Game
Umpire
Submit game info
Create game
Submit game report
Update game report
UMLsec model
61

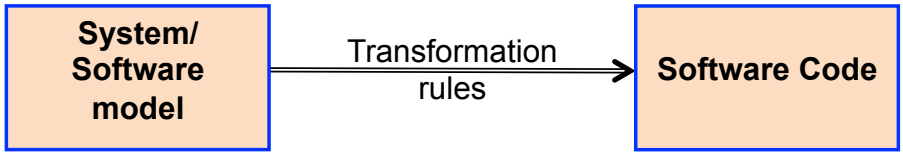

«securml.permission»
FFEpermission
-submitGameInfo : Insert
-submitConfirmation : Update
{AC#1}
«securml.user»
Bob
«assignment»
«securml.role»
-responsibleFFE
-organisedGame
«securml.resource»
Game
-gameInfo
Secure UML model
Static Security model
«securml.role»
Umpire
-assignedUser : String
-assignedUmpire
3..5
1
«securml.user»
Karl
{AC#3}
{AC#2}
activity
FootballFederationEmployee
Game
Umpire
Submit game info
Create game
Submit game report
Update game report
Dynamic Security model
UMLsec model
62



Outline

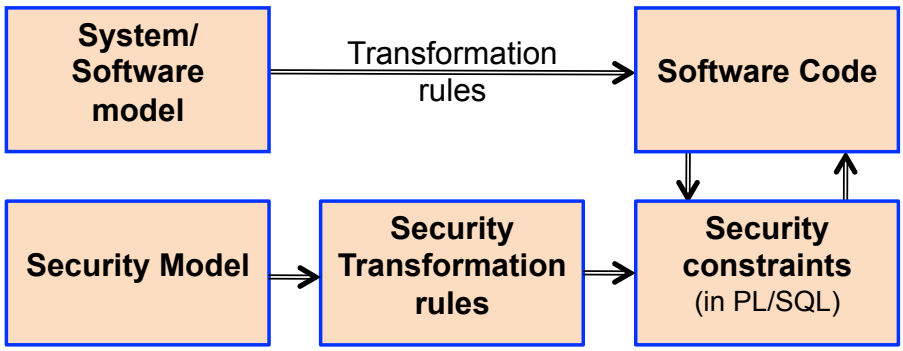
- Principles of role-based access control
- RBAC implementation requirements
- RBAC modelling languages
  - SecureUML
  - UMLsec
  - Language comparison
  - Transformation
- Model-driven security
  - **Model-driven development**
  - Security model transformation
- Further reading

Model Driven Development



- Definition of the system/software model
- Systematic development of the set of the transformation rules
- Application of these rules to generate executable software code from the model

Model Driven Security

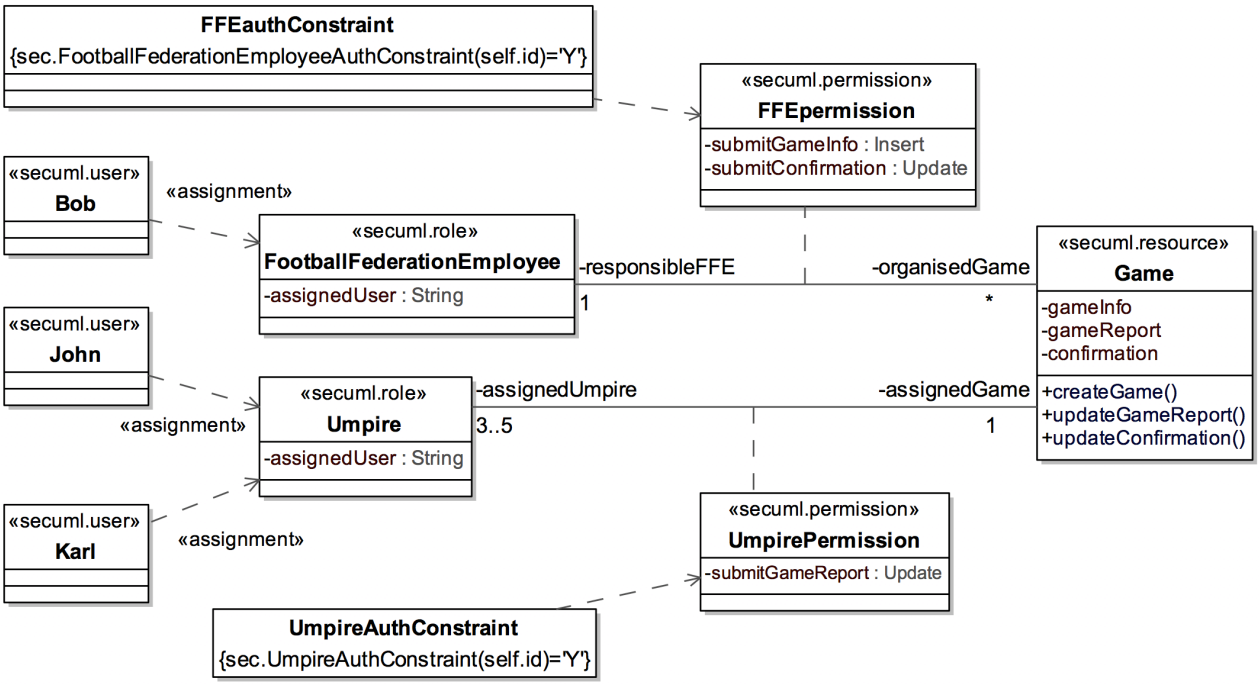


- Security model is translated to security code
- Software code and security code are generated into system architectures

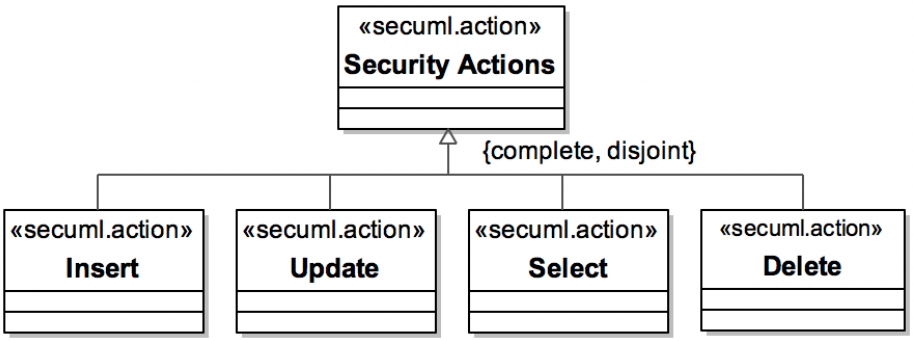
Outline

- Principles of role-based access control
- RBAC implementation requirements
- RBAC modelling languages
  - SecureUML
  - UMLsec
  - Language comparison
  - Transformation
- Model-driven security
  - Model-driven development
  - **Security model transformation**
- Further reading

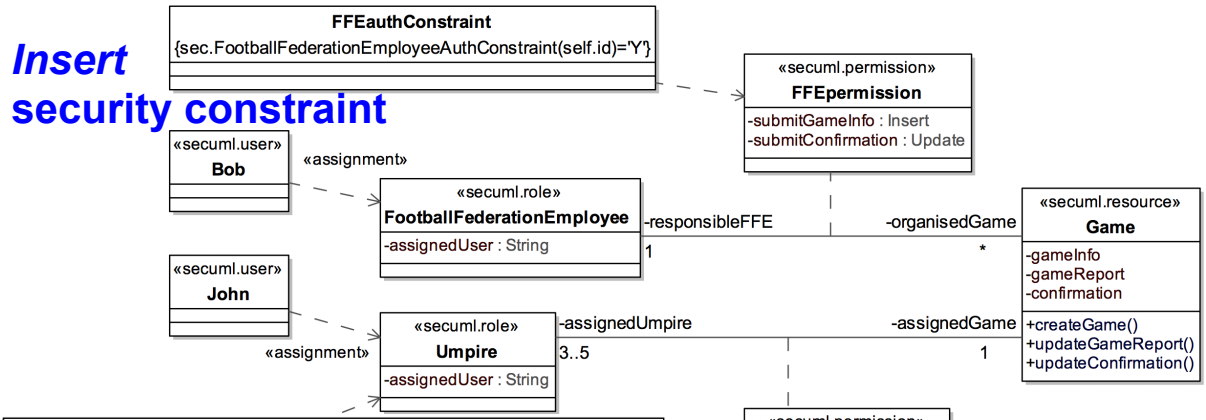
Security Model



Security transformation rules

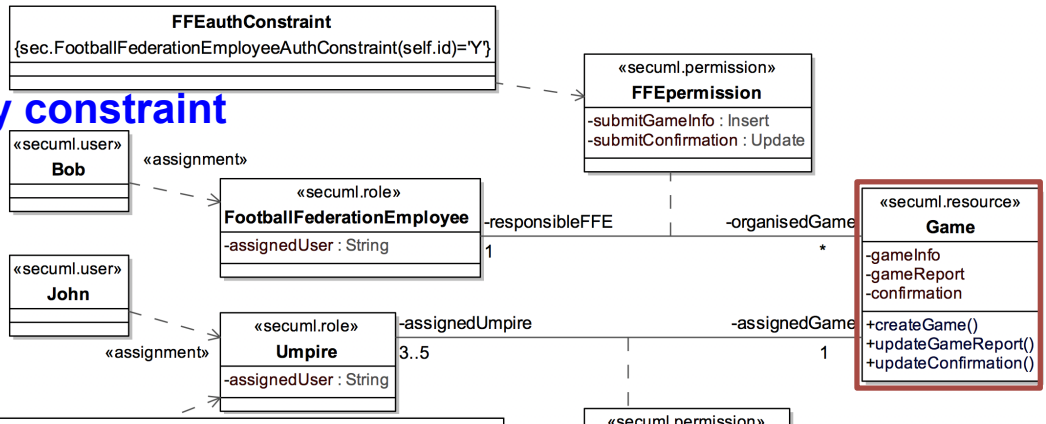


Insert
security constraint




```
-- Imported common-sql.vtl
CREATE OR REPLACE TRIGGER Game_sec_insert_trg
  INSTEAD OF INSERT ON Game
  REFERENCING NEW AS NEW
  FOR EACH ROW
DECLARE
  ex_denied EXCEPTION;
BEGIN
  IF sec.is_role('FootballFederationEmployee') = 'Y' AND
     sec.FootballFederationEmployeeAuthConstraint(self.id)='Y'
  THEN
    INSERT INTO Game (
          gameInfo, gameReport, confirmation)
    VALUES (
          :NEW.gameInfo, :NEW.gameReport, :NEW.confirmation);
  ELSE
    RAISE ex_denied;
  END IF;
EXCEPTION
  WHEN ex_denied THEN
    raise_application_error (-20000, 'Access denied!');
END;
/
```

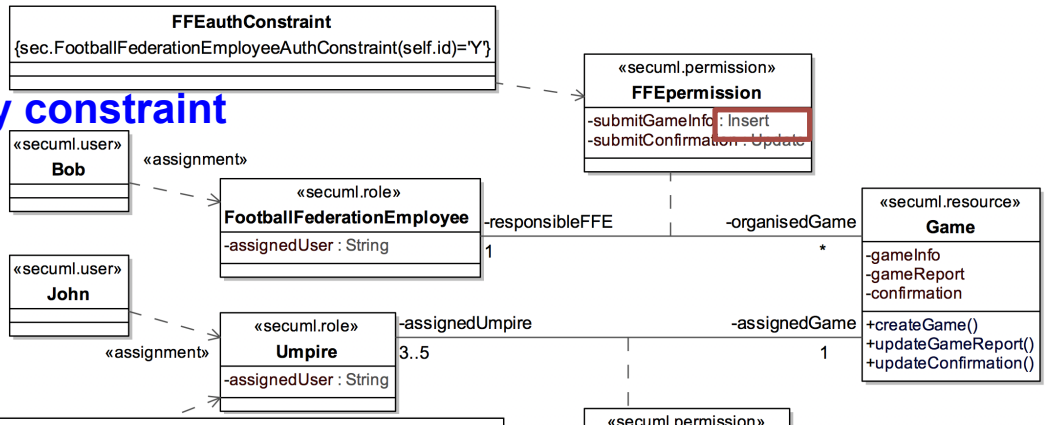
Insert security constraint




```
-- Imported common-sql.vtl
CREATE OR REPLACE TRIGGER Game_sec_insert_trg
  INSTEAD OF INSERT ON Game
  REFERENCING NEW AS NEW
  FOR EACH ROW
DECLARE
  ex_denied EXCEPTION;
BEGIN
  IF sec.is_role('FootballFederationEmployee') = 'Y' AND
     sec.FootballFederationEmployeeAuthConstraint(self.id)='Y'
  THEN
    INSERT INTO Game (
          gameInfo, gameReport, confirmation)
    VALUES (
          :NEW.gameInfo, :NEW.gameReport, :NEW.confirmation);
  ELSE
    RAISE ex_denied;
  END IF;
EXCEPTION
  WHEN ex_denied THEN
    raise_application_error (-20000, 'Access denied!');
END;
/
```

- **Resource** that needs to be secure
  - gameInfo
  - gameReport
  - confirmation

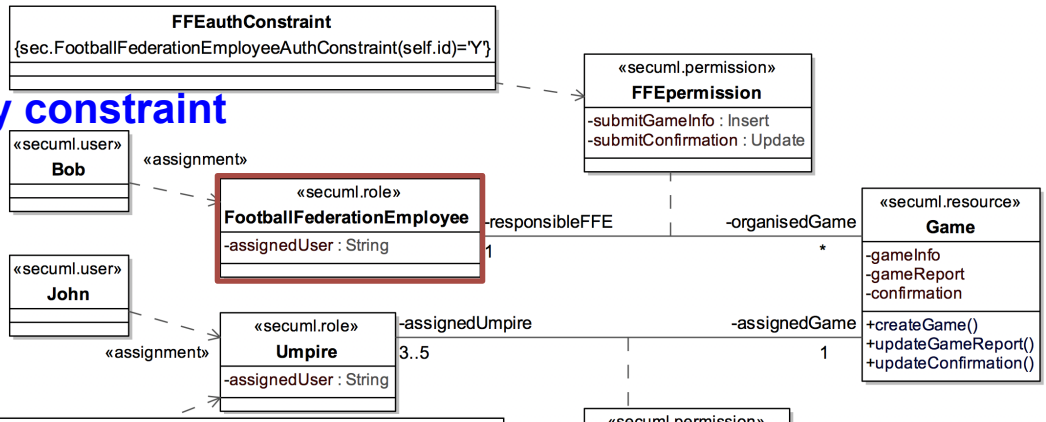
Insert security constraint




```
-- Imported common-sql.vtl
CREATE OR REPLACE TRIGGER Game_sec_insert_trg
  INSTEAD OF INSERT ON Game
  REFERENCING NEW AS NEW
  FOR EACH ROW
DECLARE
  ex_denied EXCEPTION;
BEGIN
  IF sec.is_role('FootballFederationEmployee') = 'Y' AND
     sec.FootballFederationEmployeeAuthConstraint(self.id)='Y'
  THEN
    INSERT INTO Game (
          gameInfo, gameReport, confirmation)
    VALUES (
          :NEW.gameInfo, :NEW.gameReport, :NEW.confirmation);
  ELSE
    RAISE ex_denied;
  END IF;
EXCEPTION
  WHEN ex_denied THEN
    raise_application_error (-20000, 'Access denied!');
END;
/
```

- Security action
  - **Insert**

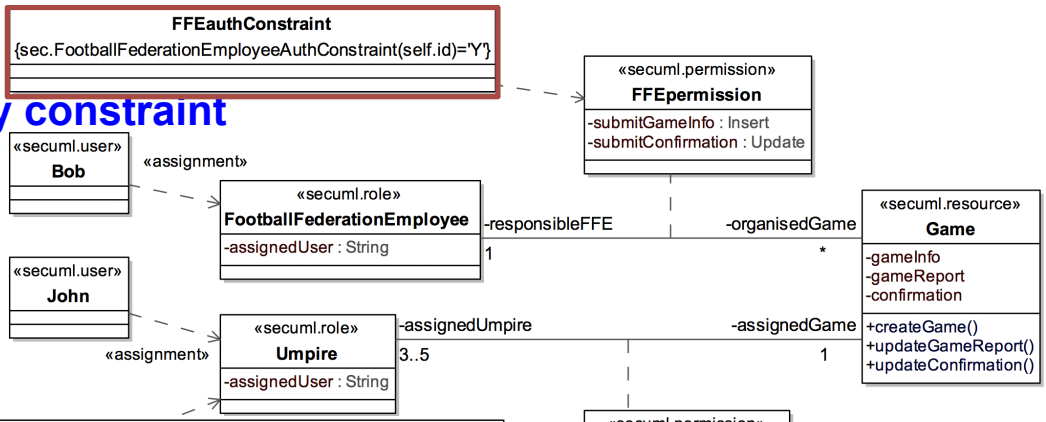
Insert security constraint




```
-- Imported common-sql.vtl
CREATE OR REPLACE TRIGGER Game_sec_insert_trg
  INSTEAD OF INSERT ON Game
  REFERENCING NEW AS NEW
  FOR EACH ROW
DECLARE
  ex_denied EXCEPTION;
BEGIN
  IF sec.is_role('FootballFederationEmployee') = 'Y' AND
    sec.FootballFederationEmployeeAuthConstraint(self.id)='Y'
  THEN
    INSERT INTO Game (
          gameInfo, gameReport, confirmation)
    VALUES (
          :NEW.gameInfo, :NEW.gameReport, :NEW.confirmation);
  ELSE
    RAISE ex_denied;
  END IF;
EXCEPTION
  WHEN ex_denied THEN
    raise_application_error (-20000, 'Access denied!');
END;
/
```

Checking the **Role** and …

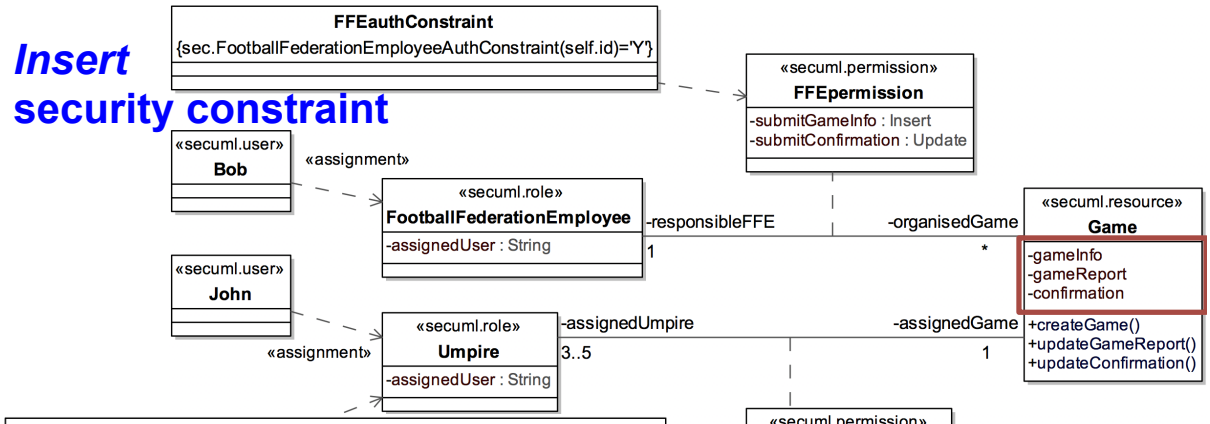
Insert security constraint




```
-- Imported common-sql.vtl
CREATE OR REPLACE TRIGGER Game_sec_insert_trg
  INSTEAD OF INSERT ON Game
  REFERENCING NEW AS NEW
  FOR EACH ROW
DECLARE
  ex_denied EXCEPTION;
BEGIN
  IF sec.is_role('FootballFederationEmployee') = 'Y' AND
    sec.FootballFederationEmployeeAuthConstraint(self.id)='Y'
  THEN
    INSERT INTO Game (
          gameInfo, gameReport, confirmation)
    VALUES (
          :NEW.gameInfo, :NEW.gameReport, :NEW.confirmation);
  ELSE
    RAISE ex_denied;
  END IF;
EXCEPTION
  WHEN ex_denied THEN
    raise_application_error (-20000, 'Access denied!');
END;
/
```

… the **authorisation constraint**

Insert security constraint

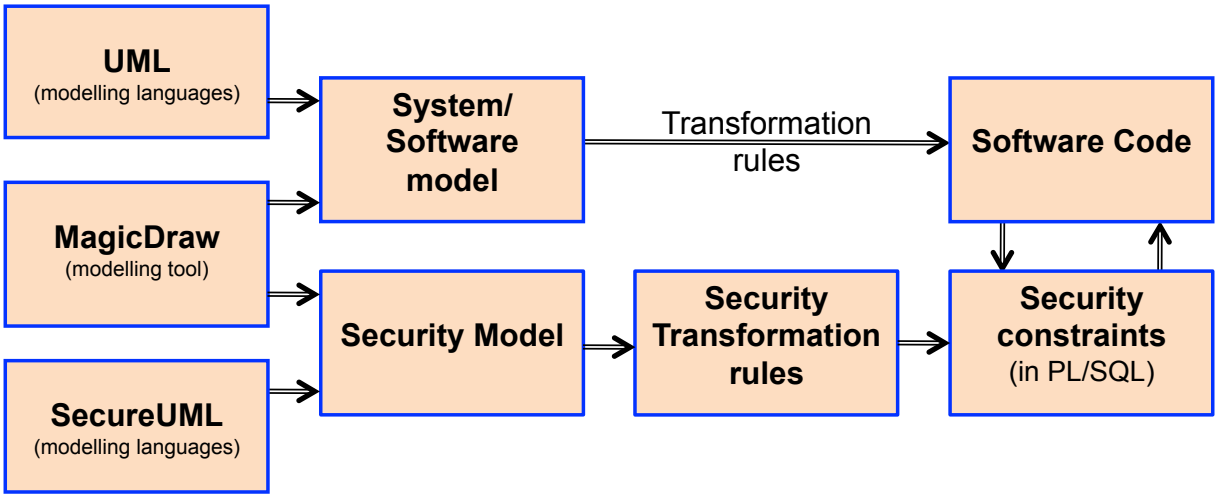



```
-- Imported common-sql.vtl
CREATE OR REPLACE TRIGGER Game_sec_insert_trg
  INSTEAD OF INSERT ON Game
  REFERENCING NEW AS NEW
  FOR EACH ROW
DECLARE
  ex_denied EXCEPTION;
BEGIN
  IF sec.is_role('FootballFederationEmployee') = 'Y' AND
     sec.FootballFederationEmployeeAuthConstraint(self.id)='Y'
  THEN
    INSERT INTO Game (
          gameInfo, gameReport, confirmation)
    VALUES (
          :NEW.gameInfo, :NEW.gameReport, :NEW.confirmation);
  ELSE
    RAISE ex_denied;
  END IF;
EXCEPTION
  WHEN ex_denied THEN
    raise_application_error (-20000, 'Access denied!');
END;
/
```

Insert **values**
- gameInfo
- gameReport
- confirmation

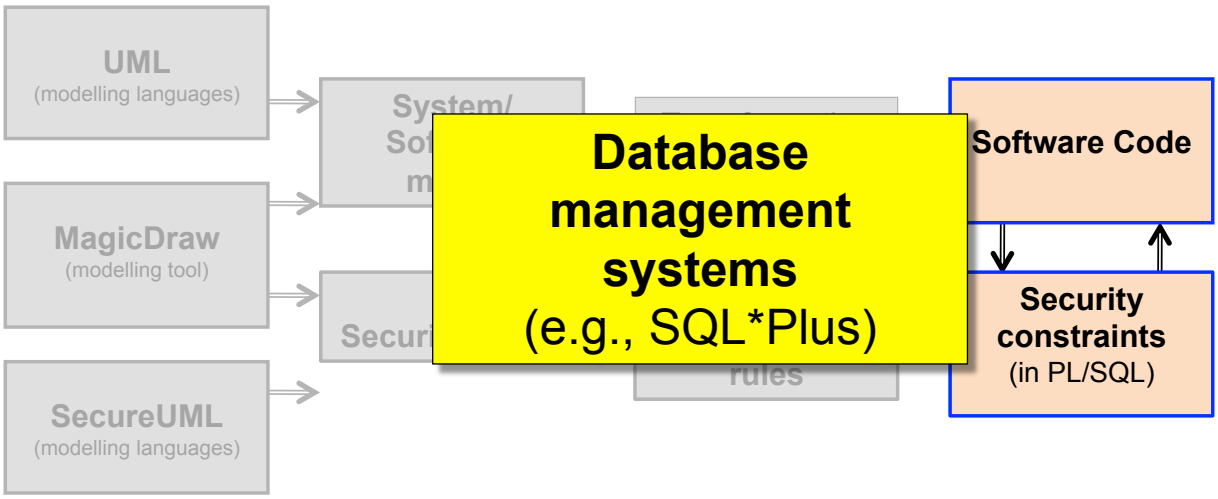
Model Driven Security

Applying Authorisation Constraints



Model Driven Security

Applying Authorisation Constraints



Outline

- Principles of role-based access control
- RBAC implementation requirements
- RBAC modelling languages
  - SecureUML
  - UMLsec
  - Language comparison
  - Transformation
- Model-driven security
  - Model-driven development
  - Security model transformation
- **Further reading**

Further reading

Access Control Approaches

- **ABAC**: Attribute-based access control
  [Hu et al.,2014, 2015]
- **UCON**: Usage control model
  [Park and Sandhu, 2004]
- **RAdAC**: Risk-adaptive access control
  [McGraw, 2009; Shaikh et al., 2012]
- **TBAC**: Token-based access control
  [Radhakrishnan, 2012]

Further reading

Model-driven security

- Framework for RBAC modelling using XACML architecture **[Xin, 2006]**
- UML for access control features to support policy validation using OCL **[Ahn and Hu, 2007]**
- UML Profile for RBAC to integrate access control specifications with the development process **[Cirit and Buzluca, 2009]**
- SecureUML is applied to define RBAC policy on XML documents to dynamically define document structure and security policy **[Tark and Matulevicius, 2014]**
- A method to recover the RBAC security model from structural and behavioural models of Web applications **[Alalfi et al., 2012]**
- Access control policies are captured from the Spring Framework applications to facilitate needed access changes **[Sergeev, 2016]**

Outline

- Principles of role-based access control
- RBAC implementation requirements
- RBAC modelling languages
  - SecureUML
  - UMLsec
  - Language comparison
  - Transformation
- Model-driven security
  - Model-driven development
  - Security model transformation
- Further reading