Access Control

Lecture 6

CS4105 - Software Security

Q2 / 2015 – 2016

Eelco Visser

TU Delft

Previous: Dealing with Security Bugs

Lessons from previous lectures

- Use programming interfaces with safety guarantees
- Use a type-safe language that enforces these guarantees
- Validate your input: prevent interpreting user data as code

Intended to avoid security vulnerabilities caused by implementation bugs

- Or rather: inadequate ***low-level access control***
- Buffer overflow: enables unauthorized access to memory
- SQL injection attack: unauthorized access to database

TU Delft

Next: High-Level Access Control

Assumption

- You do want to give users access to your system
- But not everyone should have access to everything

Access control should be designed

- not an accidental outcome of your functional design

It should be possible to **reason about properties of the policy**

- Can you explain it to a user or customer?
- Does the explanation coincide with the implementation?

Separation of concerns

- Separate definition of policy and implementation

TU Delft

Outline

Basic concepts

- Subject, object, access mode
- Authentication, access control rules

Standard policies

- Mandatory access control
- Discretionary access control
- Role-based access control

Illustration: encoding policies in WebDSL

- Running example: Wiki with pages and users
- Extend with different access control policies
- Case study: access control policies in WebLab

TU Delft

C2 Wiki

Front Page

c2.com/cgi/wiki

Search

Front Page

This ContentCreationWiki is focused on PeopleProjectsAndPatterns in SoftwareDevelopment.

The idea of a **Wiki** may seem odd at first, but dive in, explore its links and it will soon seem familiar. "Wiki" is a composition system; it's a discussion medium; it's a repository; it's a mail system; it's a tool for collaboration. We don't know quite what it is, but we do know it's a fun way to communicate asynchronously across the network.

To find a page on any specific topic, go to FindPage. To see an auto-generated list of pages which have changed recently, try RecentChanges. If you want a short list of randomly-selected pages, try RandomPages. CategoryCategory is the top level of page categorization; you can use it to delve deeper into the site.

Edit pages by using the EditText link at the bottom of the page you wish to edit. Don't worry too much about messing up, as the original text is backed up and can be easily restored (meaning, everyone can see the changes made, and will be able to correct mistakes, erase, and so on, if necessary).

The TextFormattingRules are quite simple, and the TipsForBeginners will help you learn to apply them gracefully. You'll probably want to start by editing pages that already exist. The WikiWikiSandbox is set aside for editing practice. Go there now to try it. (Please don't edit this page; changes here will likely be reversed within a few minutes).

Once you've made a couple of updates, you'll be ready to participate in discussions on the "Wiki", and before you know it you will be AddingNewPages.

Please read WikiWikiSystemNotice for the latest information regarding the operation of this site.

This site's WikiEngine, WikiBase, has since spawned other WikiEngines, which themselves have become a popular way to document things.

If you want information on the content you can expect to encounter, take a look at OnTopic and OffTopic. WikiInTheNews collects mentions of this site in the wider world.

NOTES:

- Spam is *not* allowed on this site. Any spam encountered will be deleted. Spamming this site will not increase your PageRank.

Navigation Hints:

- StartingPoints
- ReallyValuablePages
- WelcomeVisitors
- WikiList
- WikiIsNotWikipedia
- OneMinuteWiki

A history of how this page has evolved over time can be viewed at WikiArchive or other archiving sites such as web.archive.org .

View edit of December 23, 2014 or FindPage with title or text search

TU Delft

WebDSL

Data model
- automatic persistence
- embedded queries

User interface templates
- parameterized definition of page fragments
- request and response handling

Data validation rules
- form validation
- data integrity

Access control rules and policies
- constraints over objects
- weaving of access checks

TU Delft

Data Model for Wiki

object identifier

```
entity Page {
  name    :: String (id)
  content :: WikiText
}
```

domain-specific type

TU Delft

Embedded Queries

```
entity Page {
  name     :: String (id)
  content  :: WikiText
  modified :: DateTime
}

function recently Changed(n : Int) : List<Page> {
  return from Page order by modified desc limit n;
}
```

TU Delft

Page Definition and Navigation

page navigation (page call)

```
entity A { b -> B }
entity B { name :: String }

define page a(x : A) {
  navigate b(x.b){ output(x.b.name) }
}
define page b(y : B) {
  output(y.name)
}
```

page definition

Rendering Data

rendering values

```
define page page(p : Page) {
  header{output(p.name)}
  par{ output(p.content) }
  navigate editpage(p) { "[edit]" }
}
```

markup

TU Delft

Templates (Page Fragments)

template definition

```
define main() {
  includeCSS("wiki.css")
  top()
  block[class="content"] {
    elements()
  }
}
define span top() {
  navigate root() {"Wiki"}
}
```

template call

parameter

TU Delft

Forms

```
define page editpage(p : Page) {
  main{
    header{output(p.name) " (Edit)"}
    form{
      input(p.content)
      submit action{ return page(p); } { "Save" }
    }
  }
}
```

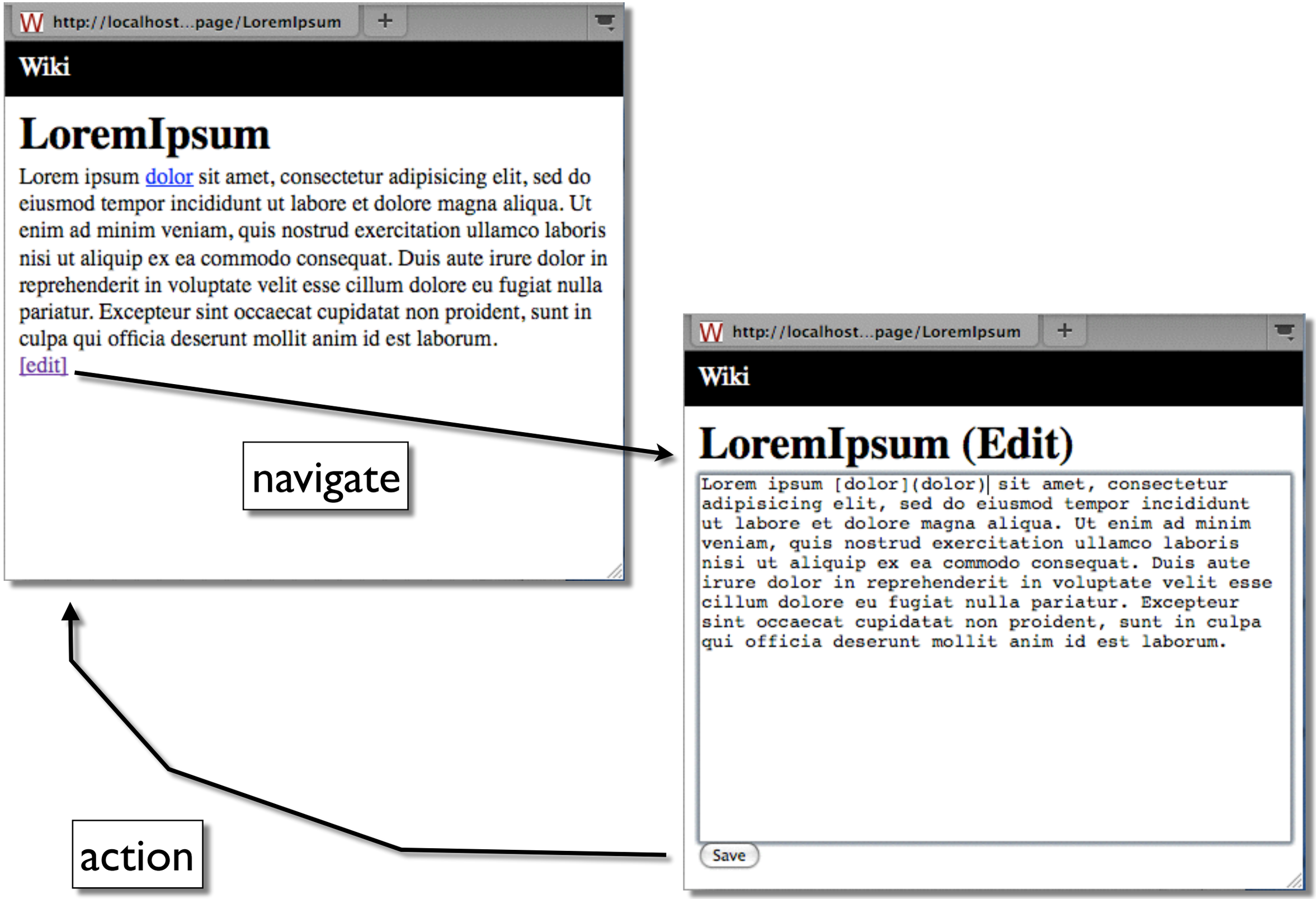
data binding

submit

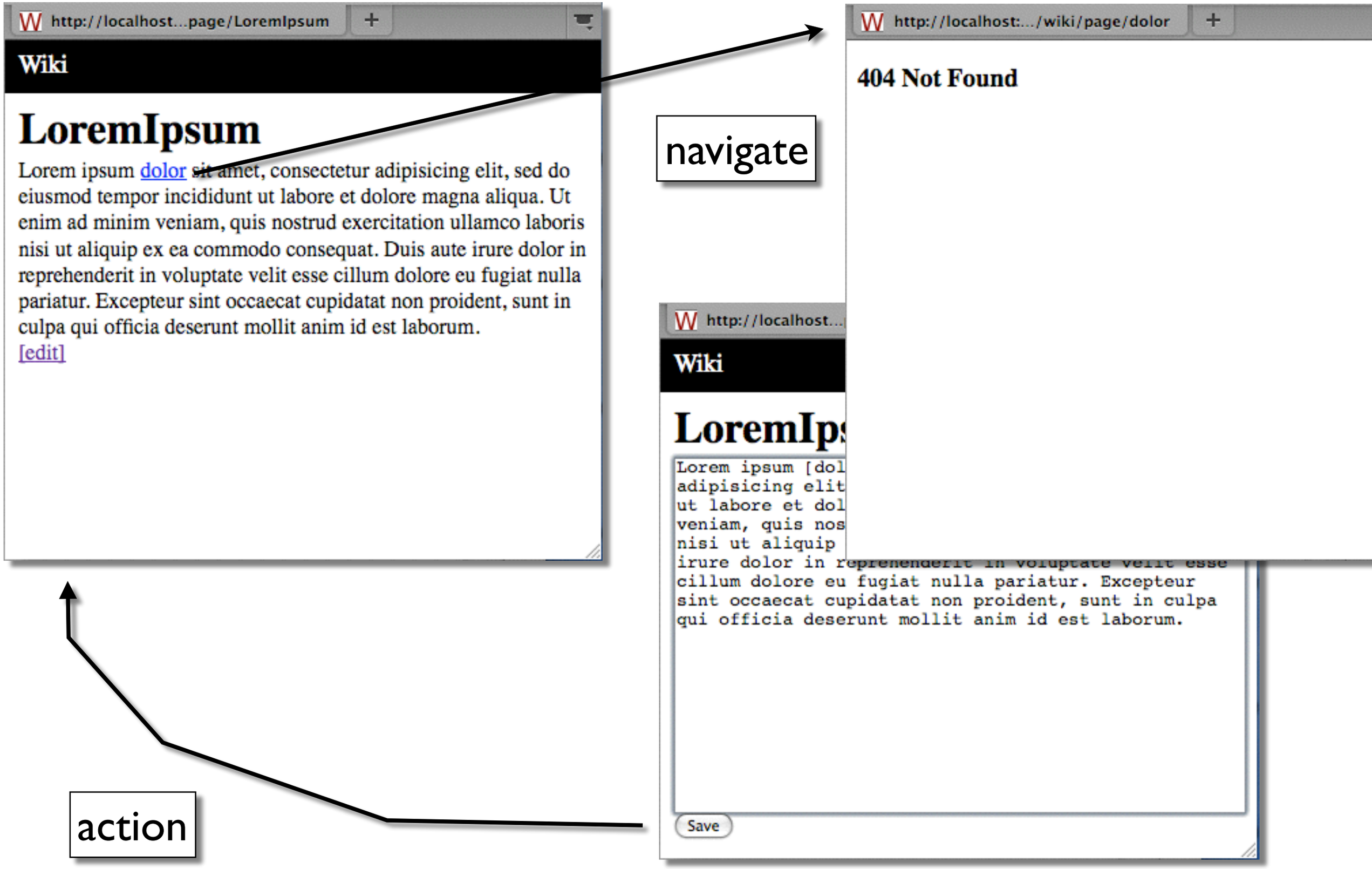
page flow

TUDelft

Forms


http://localhost...page/LoremIpsum
Wiki
LoremIpsum
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
[edit]
navigate
http://localhost...page/LoremIpsum
Wiki
LoremIpsum (Edit)
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
Save
action

Non-Existing Wiki Pages



Creating Objects

find/create object by id

```
define page page(name : String) {
  var p := getUniquePage(name)
  main{
    header{output(p.name)}
    par{ output(p.content) }
    navigate editpage(p) { "[edit]" }
  }
}
```

TU Delft

Modifying Data

```
define page editpage(p : Page) {
  main{
    header{output(p.name) " (Edit)"}
    form{
      input(p.content)
      submit action{return page(p.name);}{"Save"}
    }
  }
}
```

pass string

TU Delft

Core Wiki


http://localhost...page/LoremIpsum
Wiki
LoremIpsum
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
[edit]
http://localhost:.../wiki/page/dolor
Wiki
dolor
[edit]
navigate creates page
http://localhost...
Wiki
LoremIps
Lorem ipsum [dol
adipisicing elit
ut labore et dol
veniam, quis nos
nisi ut aliquip
irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur
sint occaecat cupidatat non proident, sunt in culpa
qui officia deserunt mollit anim id est laborum.
Save
action

Access Control Terminology

Subject

- User, often represented by program running on behalf of user
- Aka principal

Object

- Thing on which action can be performed
- Files, tables, programs, memory objects, hardware devices, strings, data fields, network connections
- Users (and programs representing users) are also objects that can be executed, halted, or assigned privileges

Access mode

- Controllable action of a subject on an object
- Examples: read, write, modify, delete, execute, create, destroy, copy, export, import, etc.

TU Delft

Meta Access Control

Access to permissions

- Who can set the access control permissions?
- What are the access control measures on that interface?
- What mechanisms does that use?

TU Delft

Principal and securityContext

```
entity User {
  username : String (name)
  email    : Email
  password : Secret
  topics   : Set<Topic>
}

principal is User with credentials username, password
```

representation of principal

turn on access control

```
session securityContext {
  principal -> User
}
```

TU Delft

Authentication

```
function signin(username : String, password : Secret) {
  var user := findUser(username);
  validate(user != null && user.password.check(password),
    "That combination of username and password is not correct.");
  securityContext.principal := user;
}

function signoff() {
  securityContext.principal := null;
}
```

TU Delft

Authentication

```
define page signin() {
  var username : String
  var password : Secret
  action doit(){ signin(username, password); }
  main{
    header{"Sign In"}
    form{
      par{ label("Username: "){ input(us
      par{ label("Password: "){ input(pa
      par{ action("Sign in", doit()) }
    }
    section{
      header{"Register"}
      par{ "No account? " navigate(regis
    }
  }
}
```

Registration

```
define page register() {
  var u := User{}
  main {
    header{"Register"}
    form{
      table{
        derive editRows from u for (
          username, fullname, email, passw
        )
      }
      action("Register", action{
        u.password := u.password.digest();
        u.save();
        message("You are registered; go ahead and sign in");
        return root();
      })
    }
  }
}
```

Access Control Rules

Constraints over data model

- boolean expression over properties of objects

Rules restrict access to resources

- page, template, action

Infer restriction of navigation

- don’t show links to inaccessible page or forbidden action

TUDelft

Wiki Access Control Rules

```
access control rules

  rule page f(x : T) { e }

  rule template g(x : T1, y : T2) { e }

  rule page f(x : T) {
    e1
    rule action a(y : T1) { e2 }
  }
```

'may access page f with argument x if boolean expression e is true'

TU Delft

Wiki Access Control Rules

```
access control rules

rule template *(*) { true }

rule page page(n : String) {
  loggedIn() || findPage(n) != null
}

rule page editpage(p : Page) {
  loggedIn()
}
```

‘anyone can view existing pages, only logged in users can create pages’

‘only logged in users may edit pages’

TU Delft

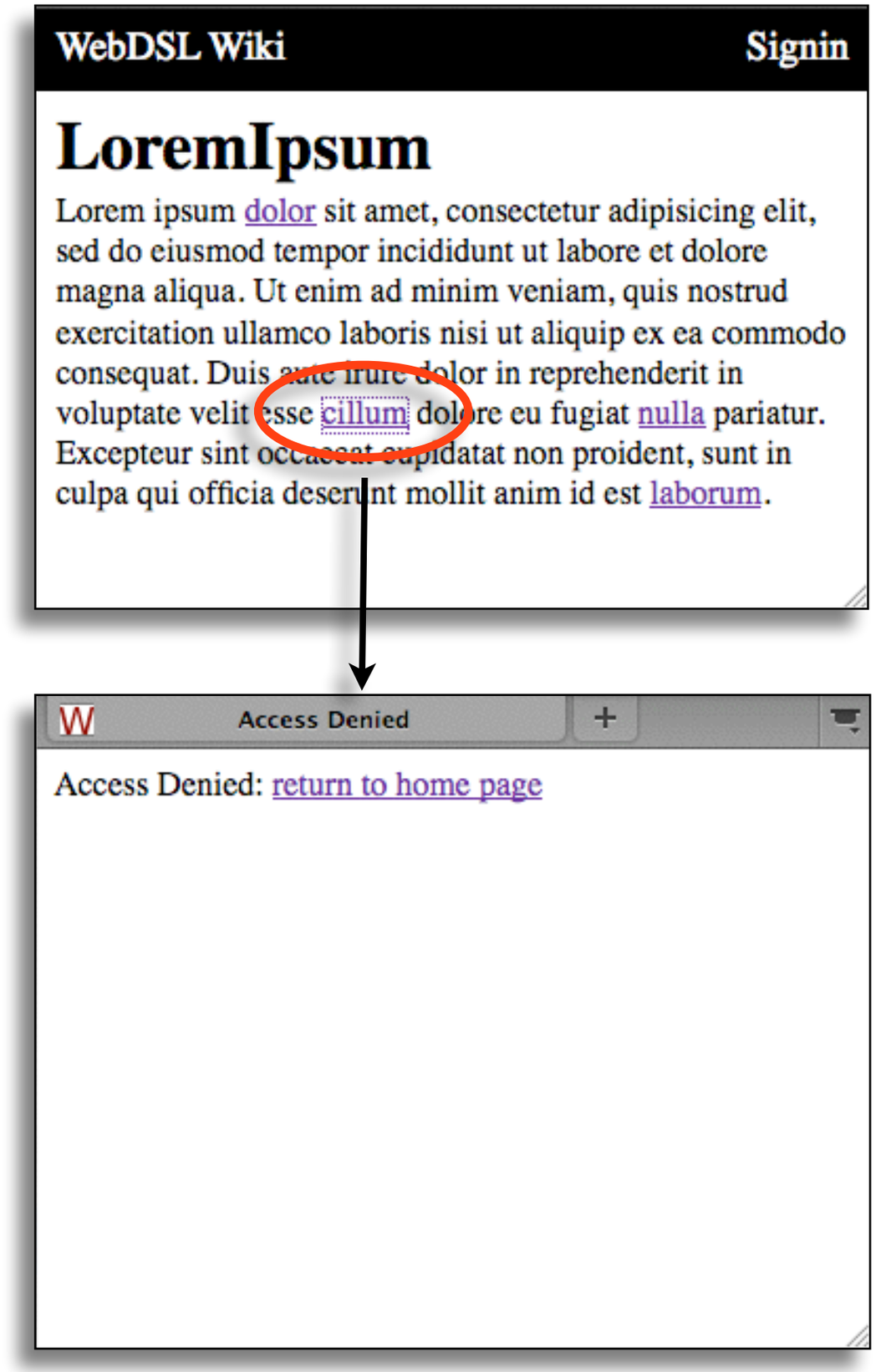
Wiki Access Control Rules

```
access control rules

rule template *(*) { true }

rule page page(n : String) {
  loggedIn() || findPage(n) != null
}

rule page editpage(p : Page) {
  loggedIn()
}
```



TUDelft

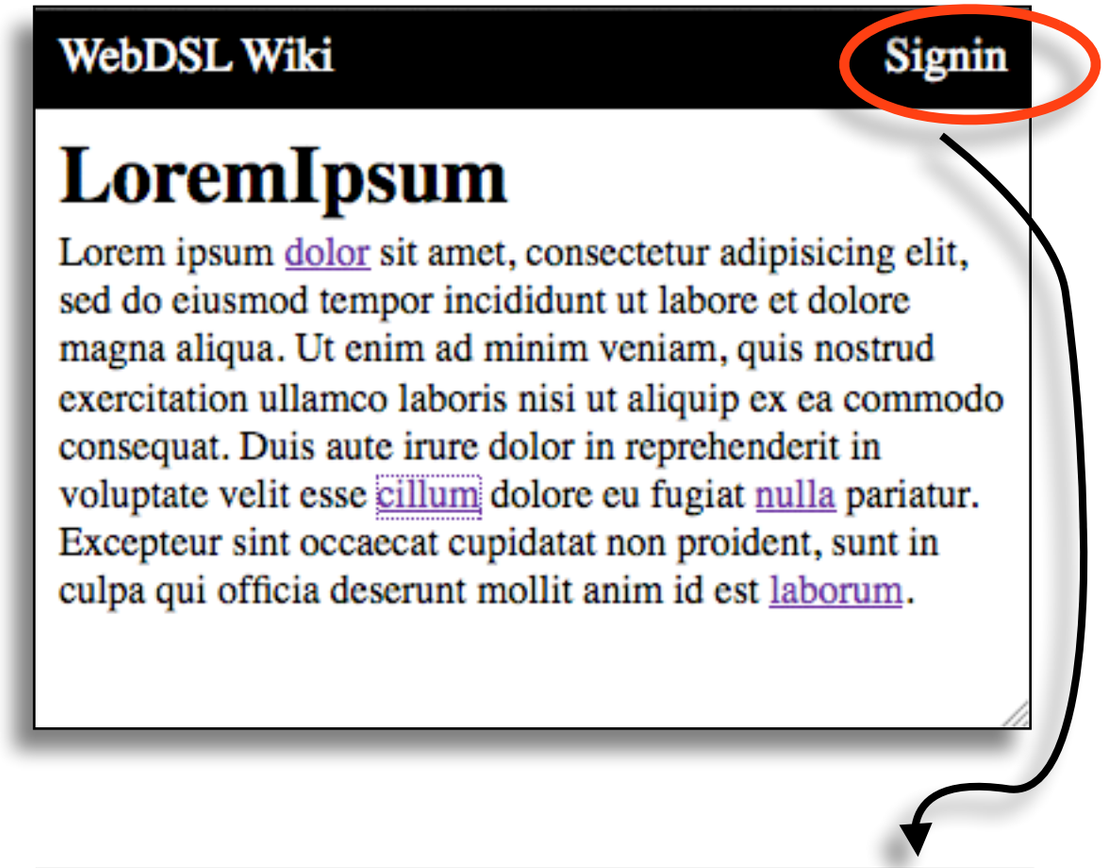
Wiki Access Control Rules

```
access control rules

rule template *(*) { true }

rule page page(n : String) {
  loggedIn() || findPage(n) != null
}

rule page editpage(p : Page) {
  loggedIn()
}
```



TU Delft

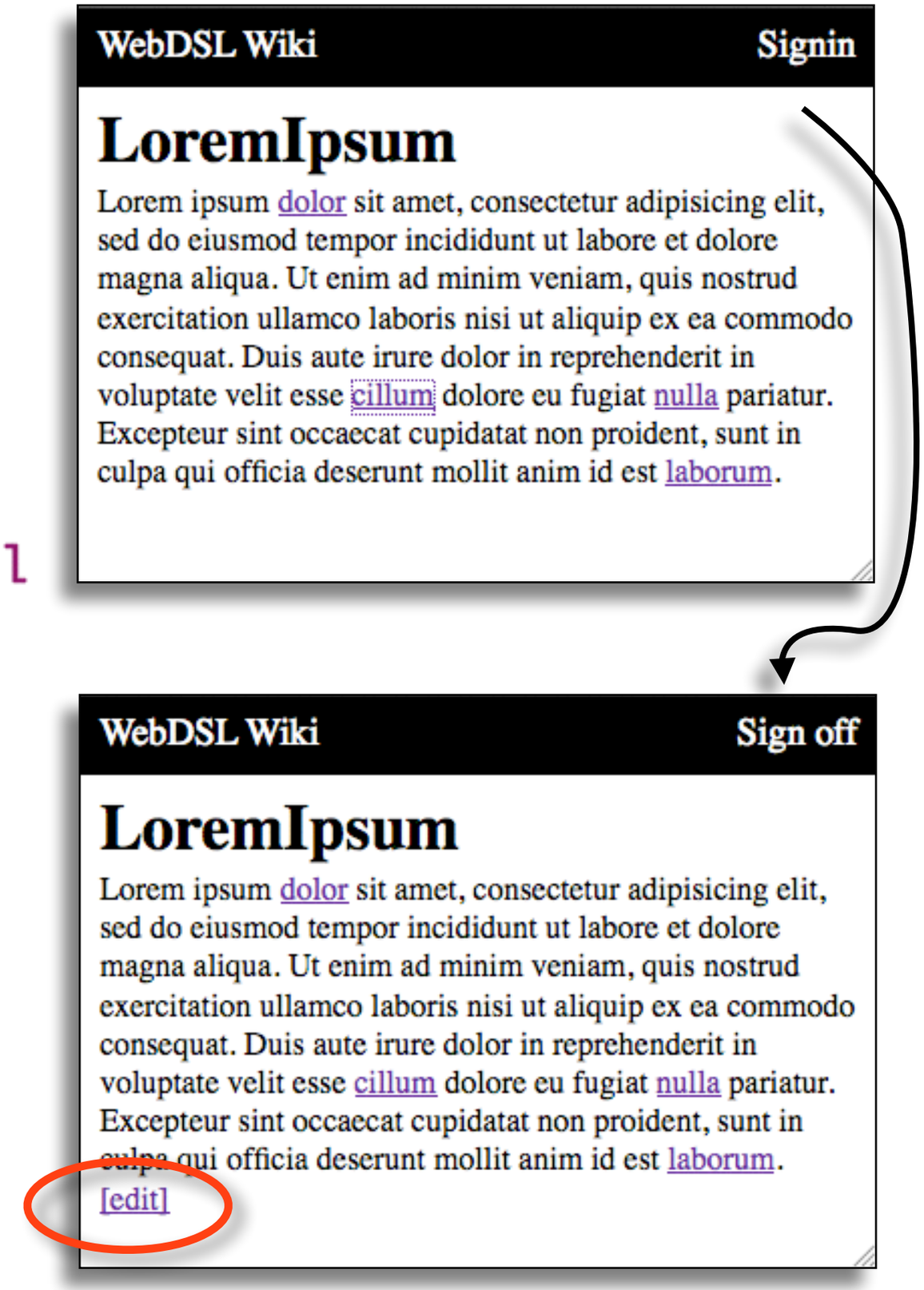
Wiki Access Control Rules

```
access control rules

rule template *(*) { true }

rule page page(n : String) {
  loggedIn() || findPage(n) != null
}

rule page editpage(p : Page) {
  loggedIn()
}
```



TU Delft

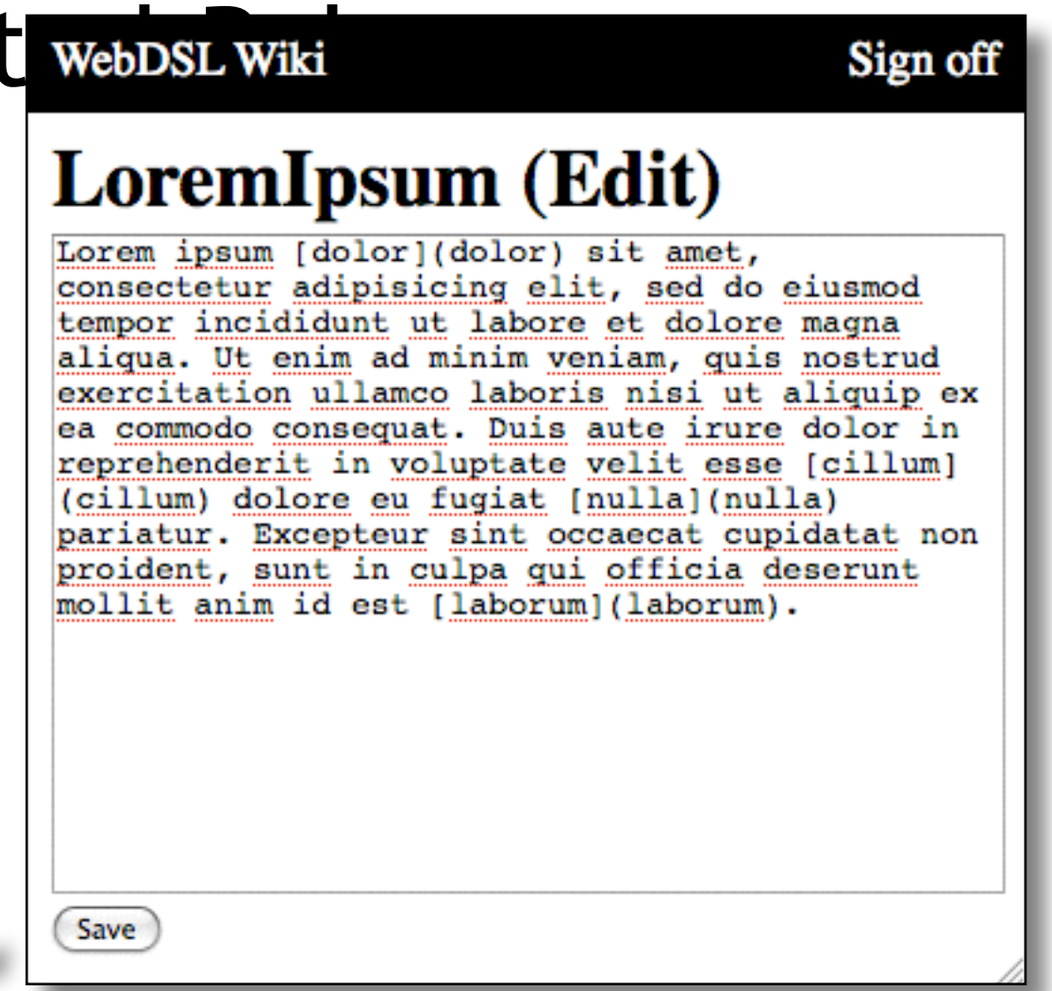
Wiki Access Control Rules

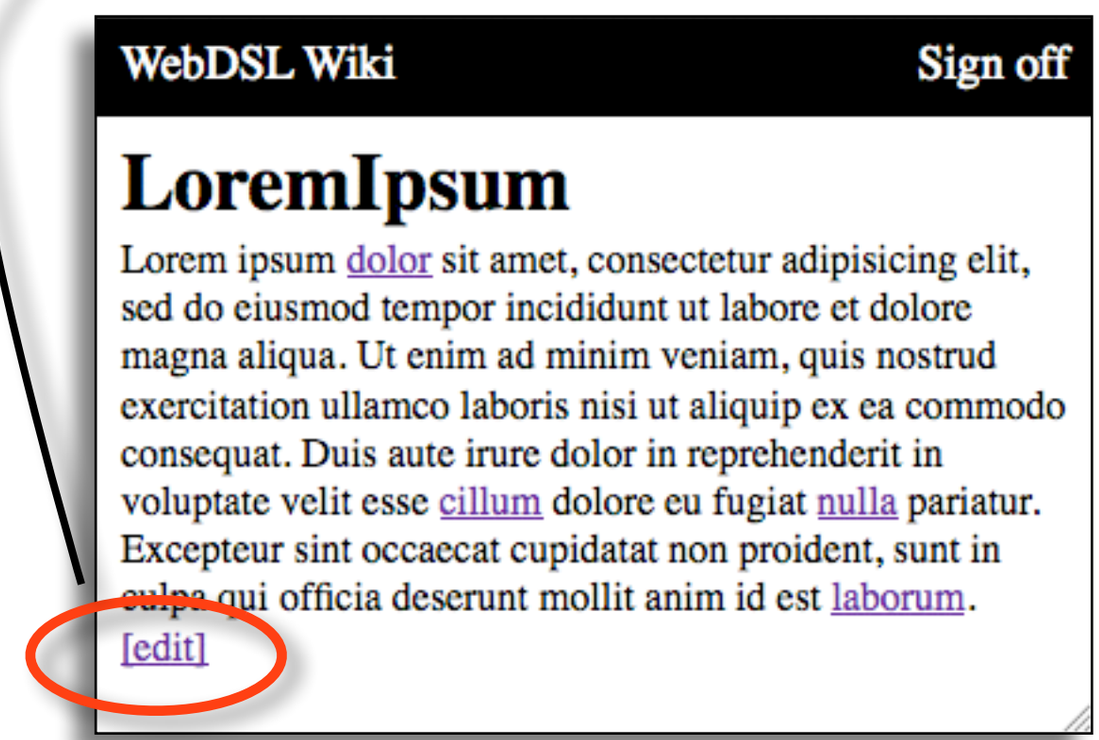
```
access control rules

rule template *(*) { true }

rule page page(n : String) {
  loggedIn() || findPage(n) != null
}

rule page editpage(p : Page) {
  loggedIn()
}
```





TU Delft

Access Control Policies

Standard policies

- Mandatory access control
- Discretionary access control
- Role-based access control

Mixing policies

- Role-based + discretionary access control

WebDSL

- No restrictions on access control policies

TU Delft

Encoding Access Control Policies

Rules

- Who may access which resources?
- Who can apply which actions?

Representation

- How are permissions stored?

Administration

- How can permissions be changed?
- Who can change permissions?

TU Delft

Wiki: Data Model

```
entity Page {
  name     :: String (id)
  content  :: WikiText
  modified :: DateTime
  function init() {
    modified := now();
  }
}

function createPage(n : String) : Page {
  var p := getUniquePage(n);
  p.init();
  p.save();
  return p;
}
```

TU Delft

Wiki: User Interface Templates

```
define page page(name : String) {
  var p : Page init{ p := findPage(name); }
  if(p == null) { pagenotfound(name) }
  else { showpage(p) }
}
define showpage(p : Page) { // ...
  navigate editpage(p) { "[edit]" }
}
define pagenotfound(name : String) { // ...
  action create() { return editpage(createPage(name)); }
  submit create() { "Create" }
}
define page editpage(p : Page) {
  action save() { return page(p.name); }
  form{
    par{ label("Text"){ input(p.content) } }
    changeAccess(p)
    submit save() { "Save" }
  }
}
```

(abbreviated to navigation structure)

Wiki: Generic Access Control Rules

```
access control rules

rule page page(n : String) {
  mayViewPage(n)
}
rule template showpage(p : Page) {
  p.mayView()
}
rule template pagenotfound(n : String) { true
  rule action create() { mayCreatePage() }
}
rule page editpage(p : Page) {
  p.mayEdit()
}
```

TU Delft

MAC: Mandatory Access Control

Security labels

- Classification label protects object
  - Top Secret, Secret, Confidential, Unclassified
  - Labels ordered in lattice
- Clearance indicates access of subject

Confidentiality rules

- Read-down
  - clearance should be higher than or equal to classification of document to read
- Write-up
  - clearance is lower than or equal to classification of document to write
  - prevents leaking confidential information to lower levels

TU Delft

MAC: Representation

```
entity Label {
  name   :: String
  higher -> Set<Label> (inverse=Label.lower)
  lower  -> Set<Label> (inverse=Label.higher)
  function dominates(l : Label) : Bool {
    return l == this
        || Or[l2.dominates(l) | l2 : Label in this.lower];
  }
}
extend entity User {
  clearance -> Label
}
extend entity Page {
  classification -> Label
}
extend session securityContext {
  activeClearance -> Label
}
```

TU Delft

MAC: Predicates

```
extend entity Page {
  function mayView() : Bool {
    return activeClearance.dominates(classification);
  }
  function mayCreatePage(p : Page) : Bool {
    return classification.dominates(activeClearance);
  }
}
```

TU Delft

DAC: Discretionary Access Control

Permissions for subjects to access objects

- Deny by default

Objects have owner

- Administration of rights done by owner
- Owner grants, revokes users access to object
- Ownership transfer
- Administration transfer

DAC representations

- Access control directory
- Access control matrix
- Access control lists

TU Delft

Access Control Directory

List of objects per user

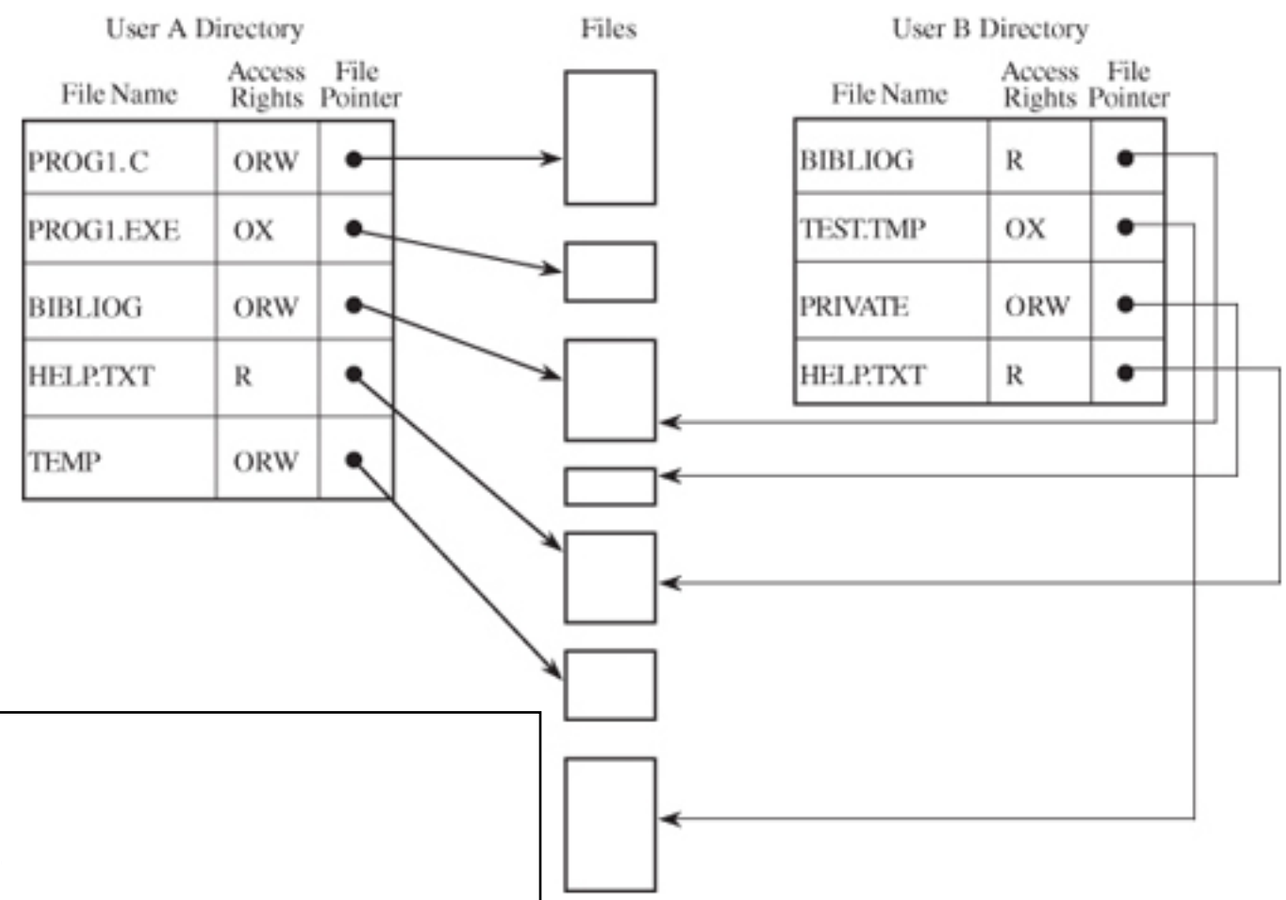
- Object has unique owner
- Owner has control over access rights
- Modifying directory through security interface
- Transfer access to other user

Issues

- Ambiguous access rights
- Revoking access

TU Delft

Access Control Directory



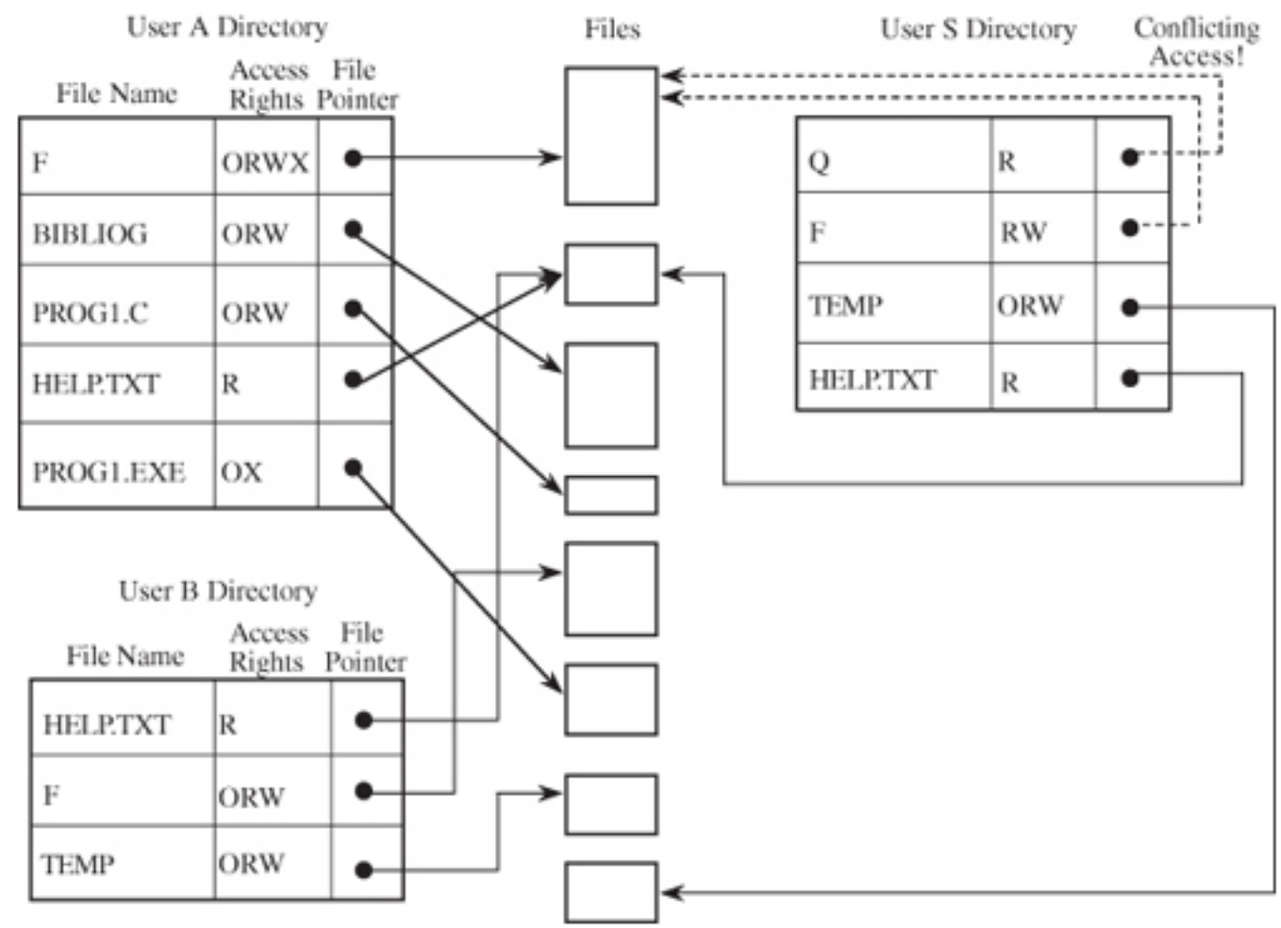

```
extend entity User {
  directory : Set<PageEntry>
  function findPage(name: String): PageEntry {
   return [e | e : PageEntry in directory where e.name == name];
  }
}

entity PageEntry {
  name      : String // identifies entry in directory
  page      : Page
  principal : User (inverse=User.directory)
  owner     : Bool (default=false)
  read      : Bool (default=false)
  write     : Bool (default=false)
}
```

Source: Pfleeger & Pfleeger 2011

Pseudonyms cause Ambiguous Access Rights

Scenario
- A and B have different files named F
- want to share it with S
- directory for S cannot contain two entries called F
- rename to something like A:F and B:F
- S may rename F to Q
- requests access to F again with different rights
- ambiguous access rights!



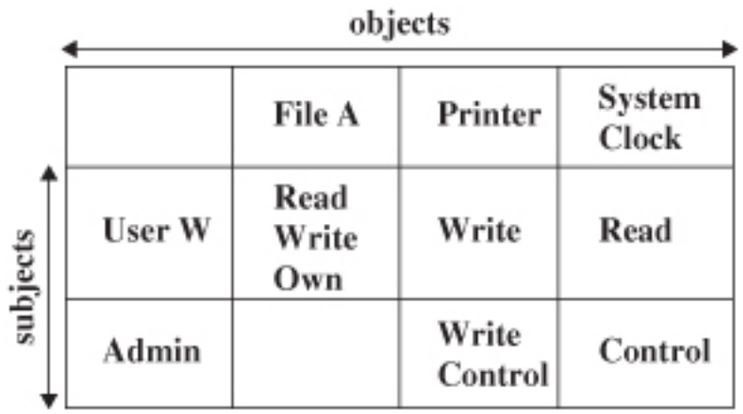

```
extend entity PageEntry {
  function transfer(u: User, write: Bool, n: String) {
    var e := this.clone();
    e.principal := u;
    e.owner := false;
    e.read  := true;
    e.write := write;
    e.name := n;
  }
}
```

Source: Pfleeger & Pfleeger 2011

Access Control Matrix

Access control directory

- Creates aliases for objects
- Complicated administration
- Object not uniquely identified



Access control matrix

- One column per object
- Rows list access for subject
- Typically sparse

	Bibliog	Temp	F	Help.txt	C_Comp	Linker	Clock	Printer
USER A	ORW	ORW	ORW	R	X	X	R	W
USER B	R	–	–	R	X	X	R	W
USER S	RW	–	R	R	X	X	R	W
USER T	–	–	–	R	X	X	R	W
SYS MGR	–	–	–	RW	OX	OX	ORW	O
USER SVCS	–	–	–	O	X	X	R	W

Source: Pfleeger & Pfleeger 2011

TUDelft

Access Control List

Access control lists

- Representation for sparse access control matrix
- Subjects with permission for object
- Deny by default

Objects have owner

- Owner grants, revokes users access to object
- May delegate to moderator
- Ownership transfer

Issues

- What happens when another subject takes over project?
- Reassign ownership of all related objects?

TU Delft

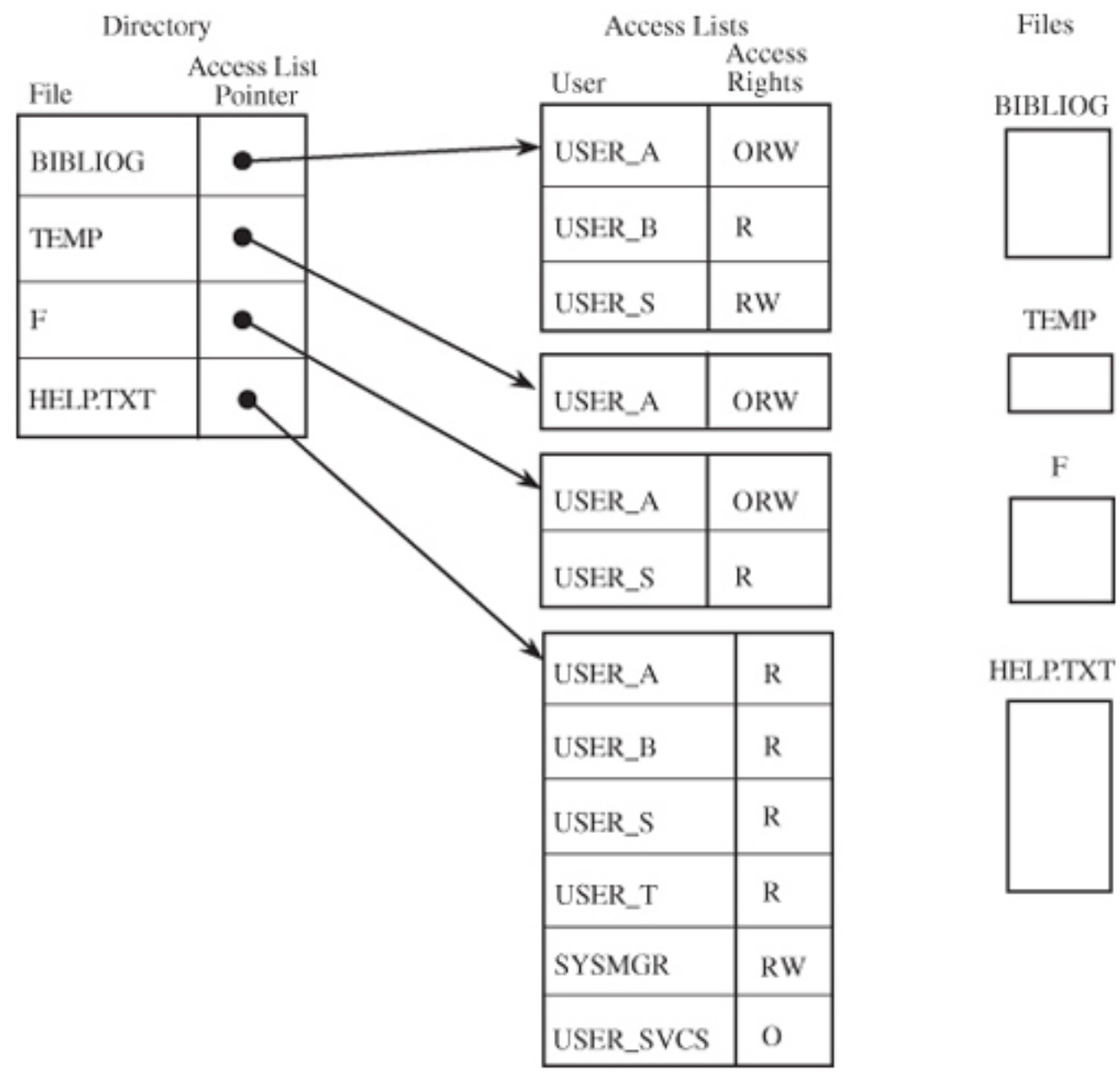
Access Control List

	File A	Printer	System Clock
User W	Read Write Own	Write	Read
Admin		Write Control	Control

access control list

	File A	Printer	System Clock
User W	Read Write Own	Write	Read
Admin		Write Control	Control

privilege list



Source: Pfleeger & Pfleeger 2011

DAC: Representation

```
entity ACL {
  viewers    -> Set<User>
  editors    -> Set<User>
  moderators -> Set<User>
}
extend entity Page {
  owner -> User
  acl   -> ACL
  extend function init() {
    acl := ACL {};
    owner := principal();
  }
}
```

TU Delft

DAC: Predicates

```
extend entity Page {
  function mayView() : Bool {
    return principal() == owner
        || principal() in acl.viewers;
  }
  function mayEdit() : Bool {
    return principal() == owner
        || principal() in acl.editors;
  }
  function mayModerate() : Bool {
    return principal() == owner
        || principal() in acl.moderators;
  }
}
```

TU Delft

DAC: Administration

```
define changeAccess(p : Page) {
  block[class:=editac]{
    label("Viewers"   ){ input(p.acl.viewers)    }
    label("Editors"   ){ input(p.acl.editors)    }
    changeModerators(p)
  }
}
define changeModerators(p : Page) {
  label("Moderators"){ input(p.acl.moderators) }
}

access control rules

rule template changeAccess(p : Page) {
  p.mayModerate()
}
rule page changeModerators(p : Page) {
  principal == p.owner
}
```

Unix File Permissions

Unix file permissions

- Simplified access control matrix

Three categories of subjects

- owner
- group: collection of subjects
- anyone

Permissions per category

- read
- write
- execute

Mix between DAC and RBAC

TU Delft

RBAC: Role-Based Access Control

Role: group of activities

- Authorization assigned to roles instead of subjects
- Users assigned to roles
- Robust to organizational changes
- Administration concerned with user/role assignment

Hierarchical roles

- Session: activated set of roles
- Least privilege: use minimal permissions for task

Separation of duties

- Critical actions require coordination

TU Delft

RBAC: Representation

```
entity Role {
  name     :: String
  juniors -> Set<Role>
  function equalOrSenior(r : Role) : Bool {
    return r == this
        || Or[r2.equalOrSenior(r) | r2 : Role in this.juniors];
  }
  function isActive() : Bool {
    return Or[r2.equalOrSenior(this)
              | r2 : Role in securityContext.activeRoles];
  }
}
extend entity User {
  roles -> Set<Role>
}
extend session securityContext {
  activeRoles -> Set<Role>
}
```

TU Delft

RBAC: Predicates

```
var admin  : Role := Role { name := "Administrator" }
var editor : Role := Role { name := "Editor"
                            juniors := {viewer} }
var viewer : Role := Role { name := "Viewer" }

extend entity Page {
  function mayView() : Bool { return isActive(viewer); }
  function mayEdit() : Bool { return isActive(editor); }
}

extend entity User {
  function mayChangeRoles() : Bool { return isActive(admin); }
}
```

TU Delft

RBAC: Administration

```
define changeRoles(u : User) {
  form{
    label("Roles") { input(u.roles) }
    submit action{ } { "Save" }
  }
}
define activateRole() {
  form{
    label("Active Role"){
      select(securityContext.activeRoles
             from securityContext.principal.roles)
    }
    submit action{ } { "Save" }
  }
}
access control rules
rule template changeRoles(u : User) { u.mayChangeRoles() }
rule template activateRole() { loggedIn }
```

Mixing Access Control Policies

Real policies

- Mix of DAC & RBAC
- Access control rules are constraints over object graph

Access control observations

- Access control matrix for objects is not enough
- Many kinds of access
- Requires fine grained access decisions
- Balance between fine grained access control and understandability/oversight/complexity of administration/ complexity of verification

WebDSL

- No policies built-in

TU Delft

Case Study: Access Control Policies in WebLab

WebLab

- Learning management system as a service
- Multiple courses, with multiple editions
- Used at multiple institutions (Delft, Darmstadt, PSU)
- Programming assignments
  - edit in browser
  - execute on server
  - automatic grading based on unit testing
- Other types of assignments
  - essay questions
  - file submissions
  - multiple choice questions
- Used for labs (homework assignments) and for exams
- **Flexible** course structures, grading policies
  - => complexity!

TU Delft

(Some) Security Requirements

Confidentiality

- asset: text of assignments (before release)
- asset: answers to assignments
- asset: specification tests
- privacy: grades only accessible to student and instructor
- threat: unauthorized user has access to confidential data
- threat: student program accesses and publishes data

Integrity

- course assets: assignments, answers
- student assets: submissions, grades
- threat: unauthorized user modifies data
- threat: student program modifies data

Availability

- threat: student programs use all resources of server
- threat: aselect authentication service (external)
- threat: web server flooded with requests (dos)

TU Delft

Temporal Access Control

Time-dependent access control

- Access depends on principal *and* on the current time

WebLab

- Assignments
  - Edit submission only before the deadline
- Access before exam
  - Only instructor can see exam questions
- Access during exam
  - No access to course material during exam
  - No sharing information during exam
  - No access to internet during exam

TU Delft

Temporal Access Control

TI2606 / 2014-2015 / CPL / EXAM /

Actions 1 2

Exam (April)

Submission | Assignment | Statistics/Dates | Submissions | Edit Assignment | Discussions

Description | Group | Grading | Dependencies | Dates | Roles | Remove | Copy to Other Course

Visible From
09/04/2015 14:00
When are students allowed to read the assignment?

Visible Until

Register Until
09/04/2015 14:45
Until when can students register for assignment with key access (exam)?

Due
09/04/2015 17:40
The due date is your advise to students when to finish the assignment.

Deadline
09/04/2015 17:45
The deadline is final date and time an assignment can be submitted (without penalty).

Extension
A deadline extension allows students to submit after the deadline at the cost of a penalty.

Penalty
0.0
In case of a deadline extension, a penalty is subtracted from the grade for the assignment. The penalty is proportional to the fraction of the extension used. `Final Grade = Base Grade - (Penalty * (Submitted - Deadline) / (Extension - Deadline))`

Done | Save

TU Delft

Spatial Access Control

Location-aware access control

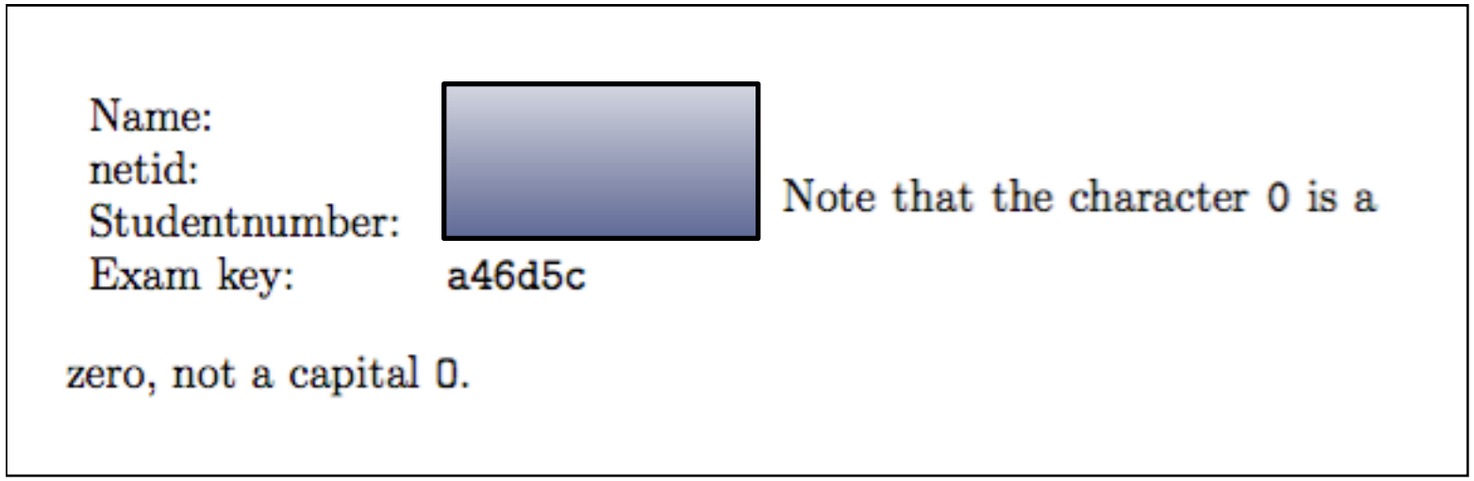
- Access only from certain locations

Spatial authentication

- Is principal in that location?

WebLab

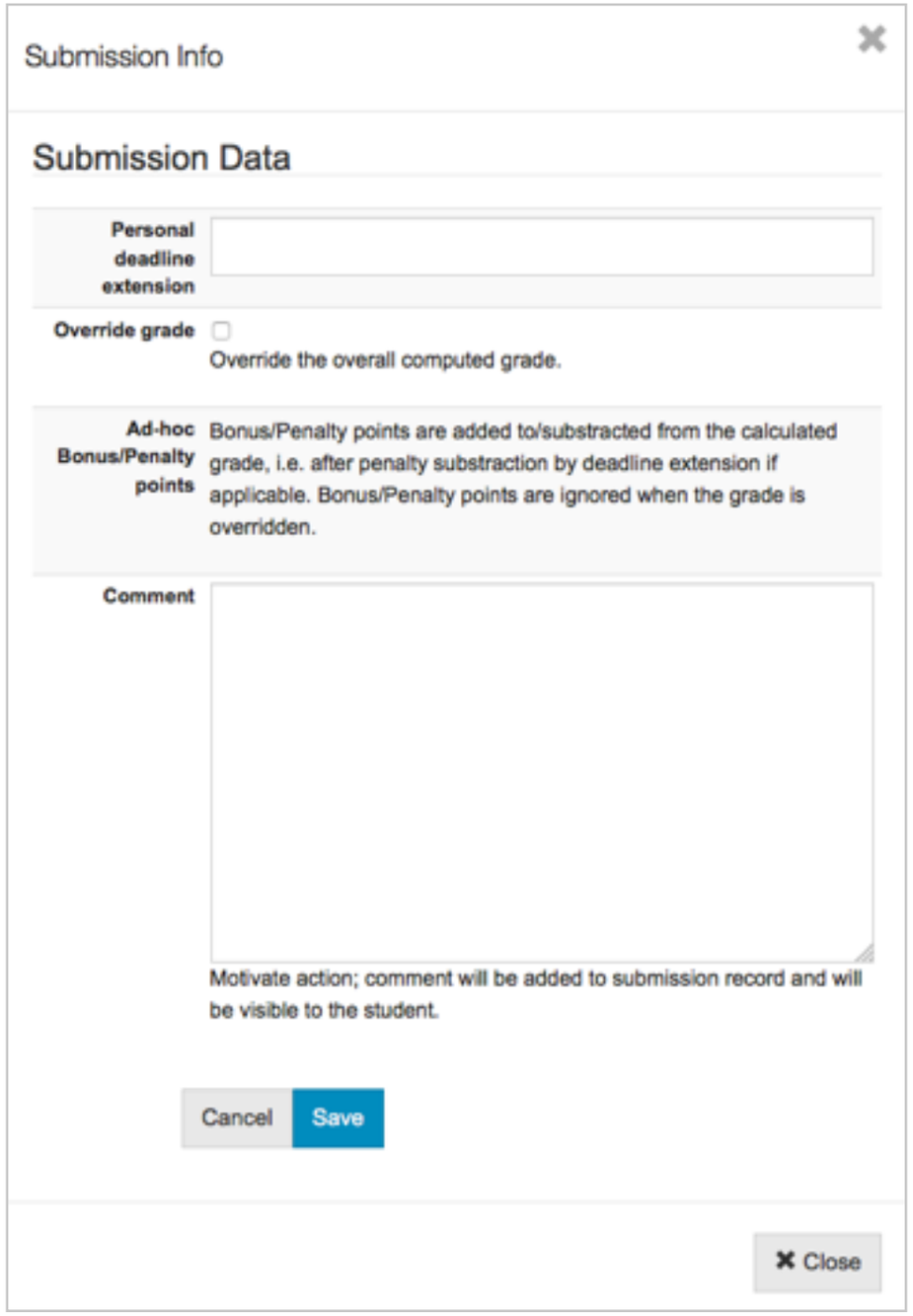
- Taking exam requires physical presence in exam room
- Using computer with restricted network access
- Hand out personal key for exam on paper


Name:
netid:
Studentnumber:
Exam key: a46d5c

Note that the character 0 is a zero, not a capital O.

TUDelft

Flexible Access Control



Allow exceptions to the rules

- don’t be rigid

WebLab

- Personal deadline extension
- Grade override
- => complicates logic

Separate ‘formal’ roles from ‘effective’ roles

- Instructor may delegate tasks to assistant
- Observer may not have formal role in course

TUDelft

Assignments and Submissions

Assignment

- View
- Edit assignment, checklist, reference solution
- Answer
- Statistics
- All submissions
- Discussions

Submission

- View
- Edit
- Grade

TU Delft

Mixing Roles and ACLs

Context-specific roles

- Roles per course
- Manager: manage access control, edit assignments
- Grader: grade submissions
- Reviewer: redundant grading for consistency
- Observer: view assignments and submissions, not edit

Role inheritance

- Roles may be extended in nodes of assignment tree
- Example: Allow assistants to edit Lab assignments, but not exam

Other 'roles'

- Student in course
- Logged in user

TU Delft

Configure Staff

Lecturers	E. Visser
Assistants	D. C. Harkes D. A. A. Pelsmaeker J. Smits-1 S. Y. vandenOever
Managers	E. Visser
Graders	
Reviewers	
Observers	

course

root assignment

TI2606 / 2014-2015 / CPL /

Actions

CPL

Submission | Assignment | Statistics/Dates | Submissions | Edit Assignment | Discussions

Description | Group | Grading | Dependencies | Dates | Roles | Remove | Copy to Other Course

Managers

Managers can create and edit assignments.

Graders

Graders are assigned to grade assignments with a checklist

Reviewers

Reviewers are assigned to double check grading

Observer

Observers can see submissions of students, but cannot edit or grade.

Lab assignment node

TI2606 / 2014-2015 / CPL /

Actions 1 2 3 4 5

Lab

Submission | Assignment | Statistics/Dates | Submissions | Edit Assignment | Discussions

Description | Group | Grading | Dependencies | Dates | Roles | Remove | Copy to Other Course

Managers

Managers can create and edit assignments.

D. C. Harkes
R. M. Hartog
D. A. A. Pelsmaeker
J. Smits-1
E. Visser
S. Y. vandenOever

Graders

Graders are assigned to grade assignments with a checklist

Reviewers

Reviewers are assigned to double check grading

D. C. Harkes
R. M. Hartog
D. A. A. Pelsmaeker
J. Smits-1
E. Visser
S. Y. vandenOever

Roles: Representation

```
extend entity Person {
    lecturer  -> Set<CourseEdition>
    assistant -> Set<CourseEdition>
    manager   -> Set<CourseEdition>
    grader    -> Set<CourseEdition>
    reviewer  -> Set<CourseEdition>
    observer  -> Set<CourseEdition>

    studentMode :: Bool (default=false)

    courseManager -> Set<Course>
}
extend entity CourseEdition {
     lecturers   -> Set<Person> (inverse=Person.lecturer)
     assistants  -> Set<Person> (inverse=Person.assistant)
     managers    -> Set<Person> (inverse=Person.manager)
     graders     -> Set<Person> (inverse=Person.grader)
     reviewers   -> Set<Person> (inverse=Person.reviewer)
     observers   -> Set<Person> (inverse=Person.observer)
}
```

privilege list

access control list

Roles: Predicates

```
extend entity CourseEdition {
    function isManager(): Bool {
      return loggedIn() && !inStudentMode()
           && (principal() in managers || course.isManager());
    }
    function isGrader(): Bool {
      return loggedIn() && !inStudentMode() && (principal() in graders || isManager());
    }
    function isReviewer(): Bool {
      return loggedIn() && !inStudentMode() && (principal() in reviewers || isManager());
    }
    function isObserver(): Bool {
      return loggedIn() && !inStudentMode()
           && (principal() in observers || isReviewer() || isGrader());
    }
    function mayView(): Bool {
      return !retired || isManager();
    }
    function mayEdit(): Bool {
      return isManager();
    }
    function mayViewGrades(): Bool {
      return isObserver();
    }
}
```

Enrollment

```
extend entity CourseEdition {
    function mayEnroll(): Bool {
      return loggedIn() && !isEnrolled(principal()) && openForEnrollment();
    }
    function mayUnenroll(): Bool {
      return loggedIn() && isEnrolled(principal()) && openForEnrollment();
    }
}
```

```
extend entity Assignment {
  function mayView(): Bool {
    return loggedIn()
        && (principal().enrollment(course) != null && isVisible()
            && examAccess() && !isBlocked()) || isObserver();
  }
  function mayViewContent(): Bool {
    return contentIsVisible() && examAccessContent()
        && dependenciesSatisfied(principal().enrollment(course))
        || isObserver();
   }
  function mayEdit(): Bool {
    return loggedIn() && isManager();
  }
  function mayViewAnswer(): Bool {
    return answerPublic;
  }
  function mayViewSubmission(student: StudentInCourse): Bool {
    return mayView()
        && ( isObserver()
             || ( student != null && principal() == student.person
                  && student.course == this.course && student.enrolled )
           );
  }
  function mayEditSubmission(student: StudentInCourse): Bool {
    return !extensionPassed()
        && mayViewSubmission(student)
        && dependenciesSatisfied(student);
  }
```

Access control predicates for assignments

```
extend entity AssignmentSubmission {
  function mayView(): Bool {
    return (assignment == null
        || (isModel && assignment.isObserver())
        || assignment.mayViewSubmission(student))
        || isBeingReviewedBy != null && isBeingReviewedBy.mayEdit();
  }
  function mayEdit(): Bool {
    return isModel && (assignment.isObserver() || assignment.isManager())
        || (assignment == null
        || (isModel && assignment.isObserver())
            || (!extensionPassed()
                && loggedIn()
                && mayView()
                && (student != null)
                && (principal() == student.person)
                && assignment.dependenciesSatisfied(student)));
  }
  function mayViewGrades(): Bool {
    return assignment == null
        || (assignment.isObserver()
            && (assignment.statsPublic() || (principal() != student.person)))
        || (assignment.statsPublic() && (principal() == student.person));

    }
}
```

Access control predicates for submissions

Access Control: It is Complicated

Formal policies do not match requirements of real applications

- => Access control as predicate over application state

Access control is product of

- Roles of subjects (instructor, student, …) in context
- Modes of access (view, edit, …)
- States of objects (public, hidden, …)
- Time (exam, deadline, personal deadline, …)
- Location (in exam room, on the web, in lecture room, …)

Consider all possible combinations

Can we do better?

- Refactor and simplify? (Some complexity is intrinsic)
- Better separation of concerns?
- Separate various dimensions of a policy?
- How to reason about interaction?
- Higher-level, declarative policy languages?

TUDelft

Assignment D3: Designing Security Policies

- Design a security policy, including authentication, authorization, and auditing for the D1 system
- Formalize the design in your favorite (web) programming language
- Can you separate policy from implementation (through a policy language)?
- How close is your formalization to the description? Can it be used as (user) documentation for the policy? Can you verify that the formalization implements the design?

TU Delft

Reading

Pfleeger & Pfleeger

- Chapter 3: Secure Software Design Elements
- Chapter 6: Countermeasure: General Access Control
- Chapter 13: Countermeasure: Access Control

Declarative Access Control for WebDSL

- Danny Groenewegen and Eelco Visser. Declarative Access Control for WebDSL: Combining Language Integration and Separation of Concerns. In Daniel Schwabe and Francisco Curbera (editors) International Conference on Web Engineering (ICWE'08), July 2008.

Role-Based Access Control

- R. Sandhu et. al. Role-Based Access Control Models. IEEE Computer, Vol. 29, No. 2, Feb 1996. http://citeseer.ist.psu.edu/sandhu96rolebased.html

TUDelft

Happy Holidays!

Next (and Last) Lecture

Lecture 7: Secure by Construction?

- Techniques for implementing language-based security
- Wednesday, January 6, 2016

TU Delft